



Название:	Полтора рубля застёжка
Категория:	Реверс-инжиниринг
Уровень:	Сложно
Очки:	1200
Описание:	Хорошая игрушка для хорошего реверс-инженера!
Теги:	C, Python, PyInstaller, Go, Rust, ARM, UPX
Автор:	ROP

### Прохождение:

Открываем архив с заданиями. Видим запароленный архив `flag.zip`. И программу `easy.elf`. Изучим её: откроем в IDA.

The screenshot shows the IDA Pro interface with several windows open:

- Functions**: Shows a list of functions: start, sub\_2C994, sub\_2C9D2, sub\_2C86F, sub\_2C870, and sub\_2CB80.
- IDB View-A**: Shows assembly code for the start function:

```
public start
start proc near
    push rax
    push rdx
    call loc_2CC3E
    push rsi
    push rbx
    push rcx
    push rdi
    add rsi, rdi
    push rsi
    mov rsi, rdi
    mov rsi, rdx
    xor ebx, ebx
    xor ecx, ecx
    or ebx, 0FFFFFFFFFFFCh
    call sub_2C9D2
    add ebx, ebx
    jz short loc_2C988
```
- Hex View-1**: Shows memory dump at address 0x100 | -856, -143 | (100, 100) | 0032C972 | 000000000000C972 - Start+12 | (Synchronized with Hex View-1)
- Structures**: Shows a list of structures.
- Enums**: Shows a list of enums.
- Imports**: Shows a list of imports.
- Exports**: Shows a list of exports.
- Graph overview**: Shows a graph of the program's control flow.
- Output**: Shows log messages:

```
Please check the Edit/Plugins menu for more information.
[!] Config file Ponce.cfg not found
[+] Ponce plugin running!
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
IDC
```

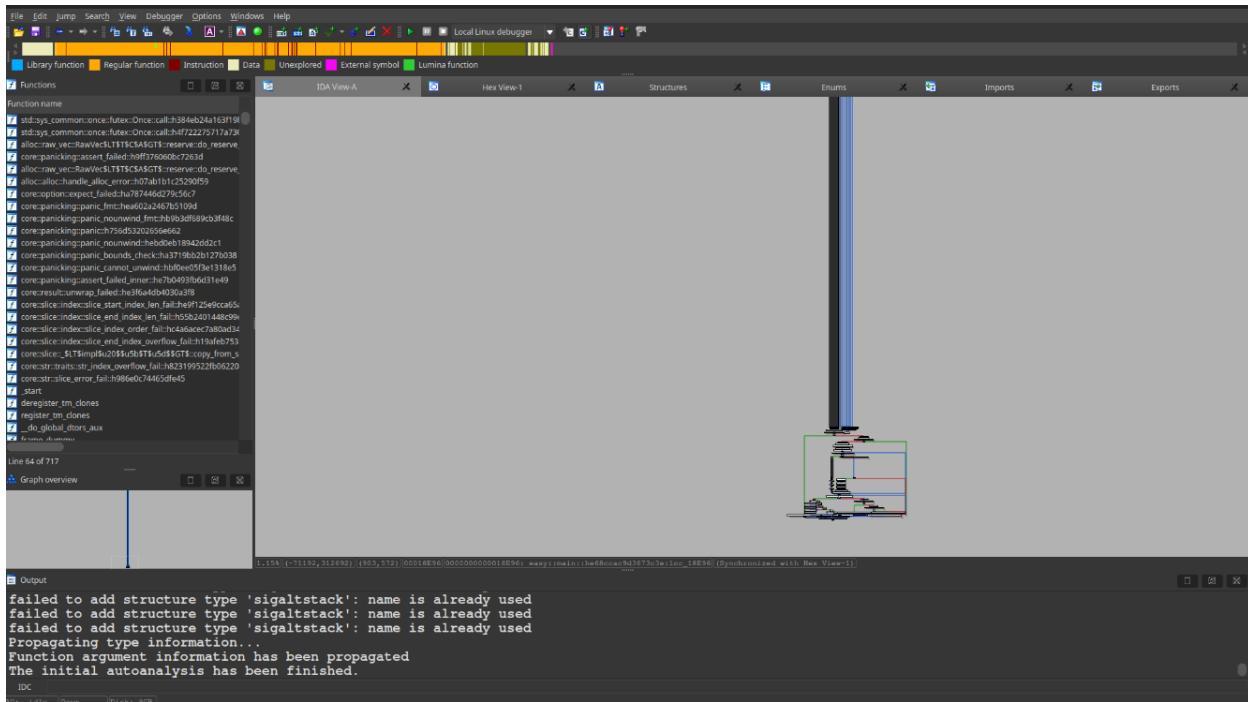
Это заглушка UPX. Мы можем распаковать его вручную или просто в самом файле `easy.elf` поменять сигнатуру `uPX!` (неправильно) на `UPX!` (правильно). А далее через `upx -d easy.elf` распаковать.

```
> upx -d easy.elf
                                Ultimate Packer for eXecutables
                                Copyright (C) 1996 - 2020
UPX 3.96          Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

      File size        Ratio       Format       Name
-----  -----  -----
4372288 <-    1062384    24.30%  linux/amd64  easy.elf

Unpacked 1 file.
/tmp/half_rubles > █
```

Изучаем новый файл. Он написан на **rust**. Откроем в IDA.



Программа просит нас ввести что-то. Изучив программу, восстановим то, что нужно ввести через кейген.

```

#include <stdio.h>

unsigned char bytes[] = {12, 111, 45, 10, 1, 111, 45, 1, 25, 110};

int main() {

    char str[40] = {0};

    unsigned char c = 0;
    unsigned char cf = 0;
    unsigned char cs = 0;
    unsigned char cth = 0;

    for (int i = 0; i < 34; i++) {
        for (int j = 0x20; j <= 127; j++) {
            unsigned char byte = j;
    
```

```
byte ^= 0x41;
byte ^= c;
byte ^= 0x55;
byte ^= 0x88;
byte ^= 0x55;
byte ^= 0x42;
byte ^= 0xff;
byte ^= 0xcd;
byte ^= cf;
byte ^= 0xcd;
byte ^= 0xb;
byte ^= cs;
byte ^= 0xae;
byte ^= cth;
byte ^= 0xaa;
byte ^= cf;
byte ^= 0x12;
byte ^= c;
byte ^= 0x34;
byte ^= cs;
byte ^= 0x56;
byte ^= c;
byte ^= 0x78;
byte ^= c;
byte ^= 0x12;
byte ^= cth;
byte ^= 0x13;
byte ^= c;
byte ^= 0x37;
byte ^= cf;
byte ^= 0xaa;
byte ^= c;
byte ^= 0xbb;
byte ^= cth;
byte ^= 0xa;
byte ^= c;
```

```
        if (byte == bytes[i]) {
            str[i] = j;
            break;
        }
    }
printf("%s\n", str);
return 0;
}
```

Запустили и получили `R1sT_1s_G00d_l@nG_rEaLLy_Tr2sT_mE`

Также получили подсказки:

```
> ./easy.elf
Enter something: R1sT_1s_G00d_l@nG_rEaLLy_Tr2sT_mE
Okay, you're not that bad...
Have you heard anything about the matryoshka?
I think that sometimes you have to look in the memory. In case there's something there.
```

А ещё часть флага:



Открываем архив и распаковываем фейковый флаг в `flag.txt` и `2.exe`. В фейке ничего нет, а вот `2.exe` упакован через UPX. Делаем действия выше.

```

Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1415]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\KEKC\Desktop\upx-4.0.1-win64>upx.exe -d 2.exe
          Ultimate Packer for executables
          Copyright (c) 1996 - 2022
UPX 4.0.1      Markus Oberhumer, Laszlo Molnar & John Reiser   Nov 16th 2022

      File size        Ratio        Format        Name
-----  -----  -----
1502208 <-    638464   42.50%    win64/pe    2.exe

Unpacked 1 file.

C:\Users\KEKC\Desktop\upx-4.0.1-win64>

```

Запустим в IDA и видим, что это Go. В строках находим [capybara.jpg](#).

Address	Length	Type	String
rdata:00000000	0000000C	C	symonwait=
rdata:00000000	0000000C	C	vbuf1=<nil>
rdata:00000000	0000000C	C	vbuf2=<nil>
rdata:00000000	0000000C	C	) p->status=
rdata:00000000	0000000C	C	, cons/mark
rdata:00000000	0000000C	C	UStringName
rdata:00000000	0000000C	C	1525978904625
rdata:00000000	0000000C	C	76203945125
rdata:00000000	0000000C	C	Bidi_Control
rdata:00000000	0000000C	C	GetAddrInfoW
rdata:00000000	0000000C	C	GetConsoleCP
rdata:00000000	0000000C	C	GetLastError
rdata:00000000	0000000C	C	GetEngaged
rdata:00000000	0000000C	C	GetFileHandle
rdata:00000000	0000000C	C	GetTempPathW
rdata:00000000	0000000C	C	Join_Control
rdata:00000000	0000000C	C	LoadLibraryW
rdata:00000000	0000000C	C	Meetei_Mayek
rdata:00000000	0000000C	C	Pahawh_Hmong
rdata:00000000	0000000C	C	ReadConsoleW
rdata:00000000	0000000C	C	RevertToSelf
rdata:00000000	0000000C	C	SetDataFile
rdata:00000000	0000000C	C	Sora_Sompeng
rdata:00000000	0000000C	C	Syloti_Nagn
rdata:00000000	0000000C	C	TransmitFile
rdata:00000000	0000000C	C	UnlockFileEx
rdata:00000000	0000000C	C	VirtualQuery
rdata:00000000	0000000C	C	abi_msmalloc
rdata:00000000	0000000C	C	close all
rdata:00000000	0000000C	C	bad flushden
rdata:00000000	0000000C	C	bad g status
rdata:00000000	0000000C	C	bad g0 stack
rdata:00000000	0000000C	C	bad recovery
rdata:00000000	0000000C	C	can't happen
rdata:00000000	0000000C	C	capybara.jpg
rdata:00000000	0000000C	C	cast4 failed
rdata:00000000	0000000C	C	chan receive
rdata:00000000	0000000C	C	dumping heap

Переходим к коду, где это использовано.

Программа в целом не поменялась, если сравнивать с первой. Просто тут новый алгоритм и 2 массива: 1 - капибара с частью ключа, 2 - hex другого таска.

Изучаем программу и пишем кейген:

```
package main

import (
    "fmt"
)

func main() {
    check := [15]byte{71, 171, 188, 205, 70, 57, 59, 184, 54, 205,
        // Записываем строку в файл

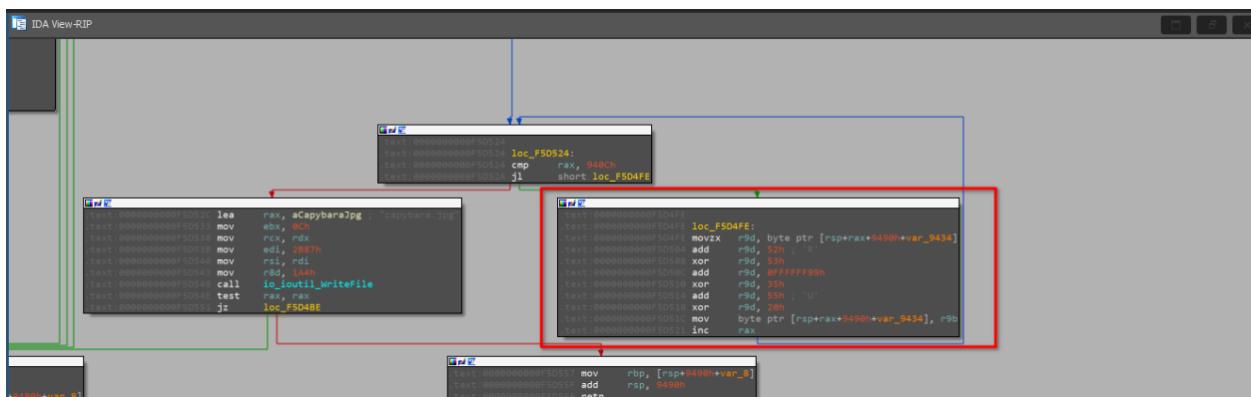
    for i := 0; i < 15; i++ {
        for j := 0x20; j <= 0x127; j++ {
            temp := j^0x15
            temp += 55
            temp ^= 0x44
            temp -= 105
            temp ^= 0x30
            temp += 0xde
            temp ^= 0xaf
            temp += 0xbe
            temp ^= 0xef
            temp += 0xCD
            temp ^= 0xB
            temp -= 0x13
            temp ^= 0xC4

            if (byte(temp) == check[i]) {
                fmt.Println(string(j))
                break
            }
        }
    }
}
```

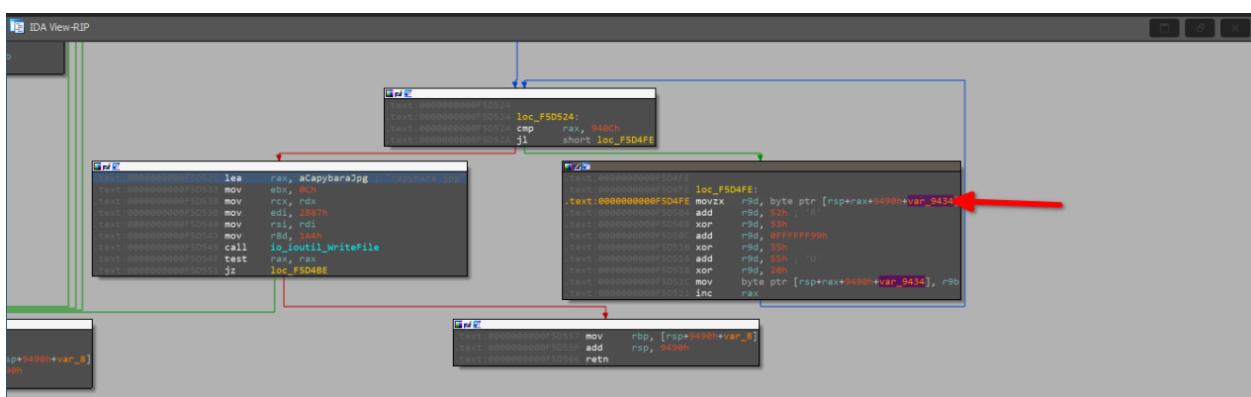
```
}
```

Запускаем, получаем пароль : `CODEBY_H2110_Go`

Видим здесь расшифровку нового файла.



Выполняем этот цикл расшифровки.



В инструкции `cmp rax, 940Ch` видим размер данных.

Переходим по стрелочке. Переходим по указателю в переменной. Запоминаем адрес (`0xC0000DDF50`), вычитаем из него размер массива (`0xC0000DDF50 - 0x940C = C0000D4B44`).

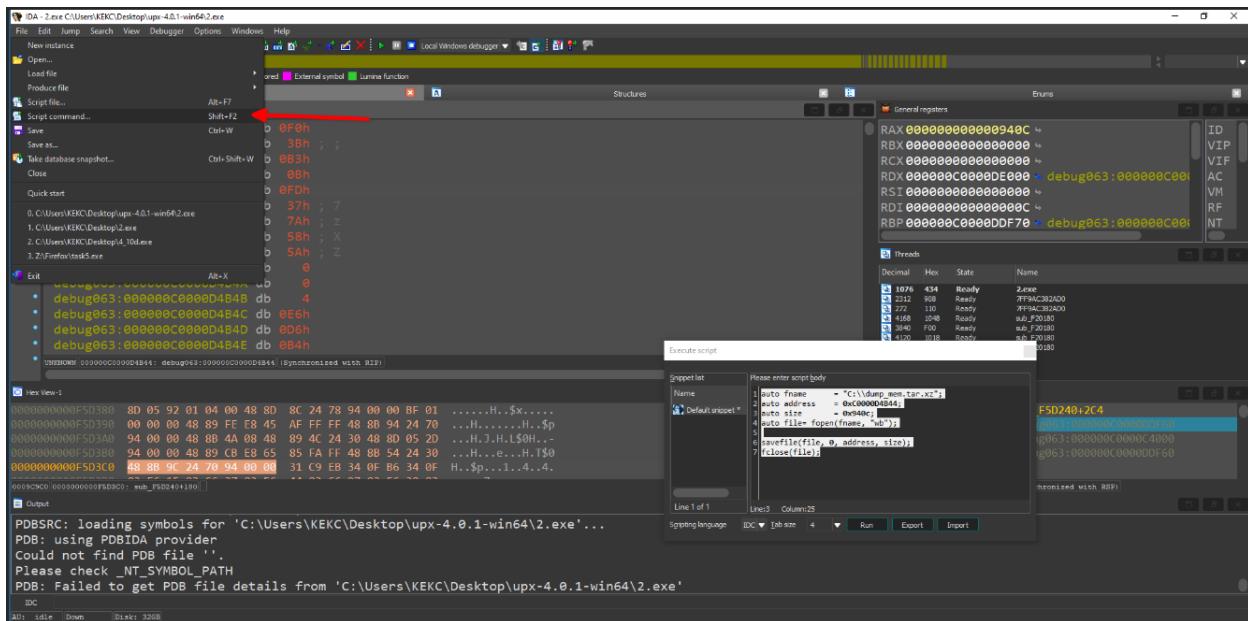
IDA View-RIP

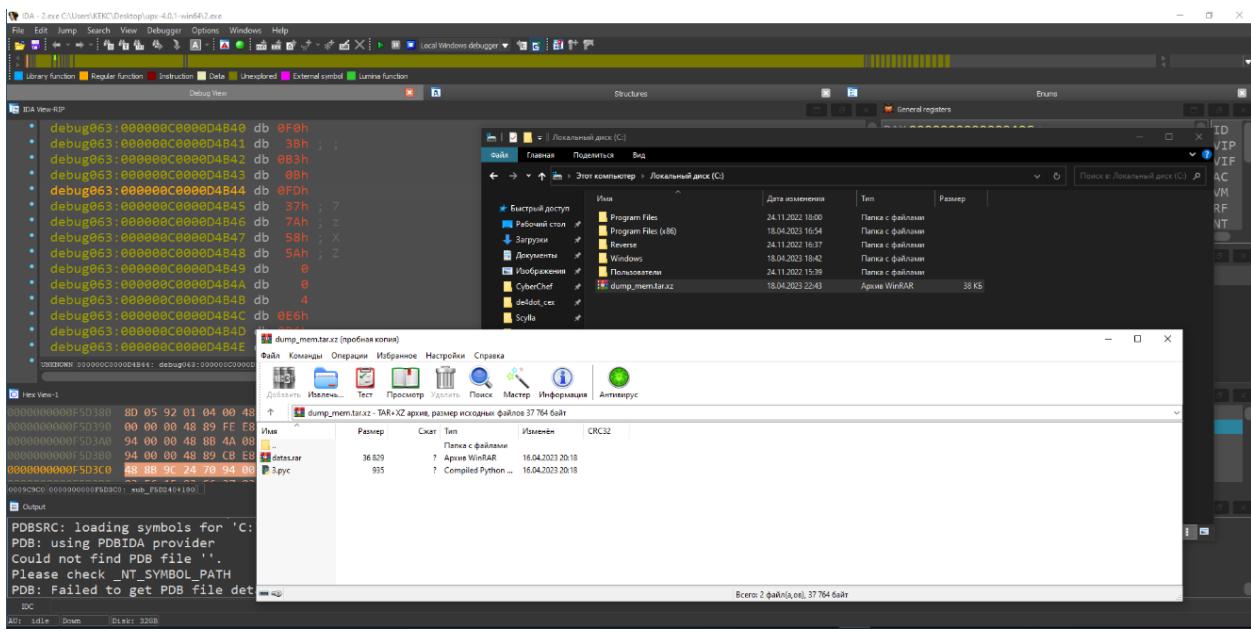
```
• debug062:000000C0000DDF4C db 0
• debug062:000000C0000DDF4D db 4
• debug062:000000C0000DDF4E db 59h ; Y
• debug062:000000C0000DDF4F db 5Ah ; Z
• debug062:000000C0000DDF50 dq offset unk_C0000DE000 | 
• debug062:000000C0000DDF58 db 40h ; @
• debug062:000000C0000DDF59 db 22h ; "
• debug062:000000C0000DDF5A db 8
• debug062:000000C0000DDF5B db 0
• debug062:000000C0000DDF5C db 0C0h
• debug062:000000C0000DDF5D db 0
• debug062:000000C0000DDF5E db 0
• debug062:000000C0000DDF5F db 0
• debug062:000000C0000DDF60 db 60h ; `
• debug062:000000C0000DDF61 db 66h ; f
```

Ставим курсор на `unk` и используем этот скрипт:

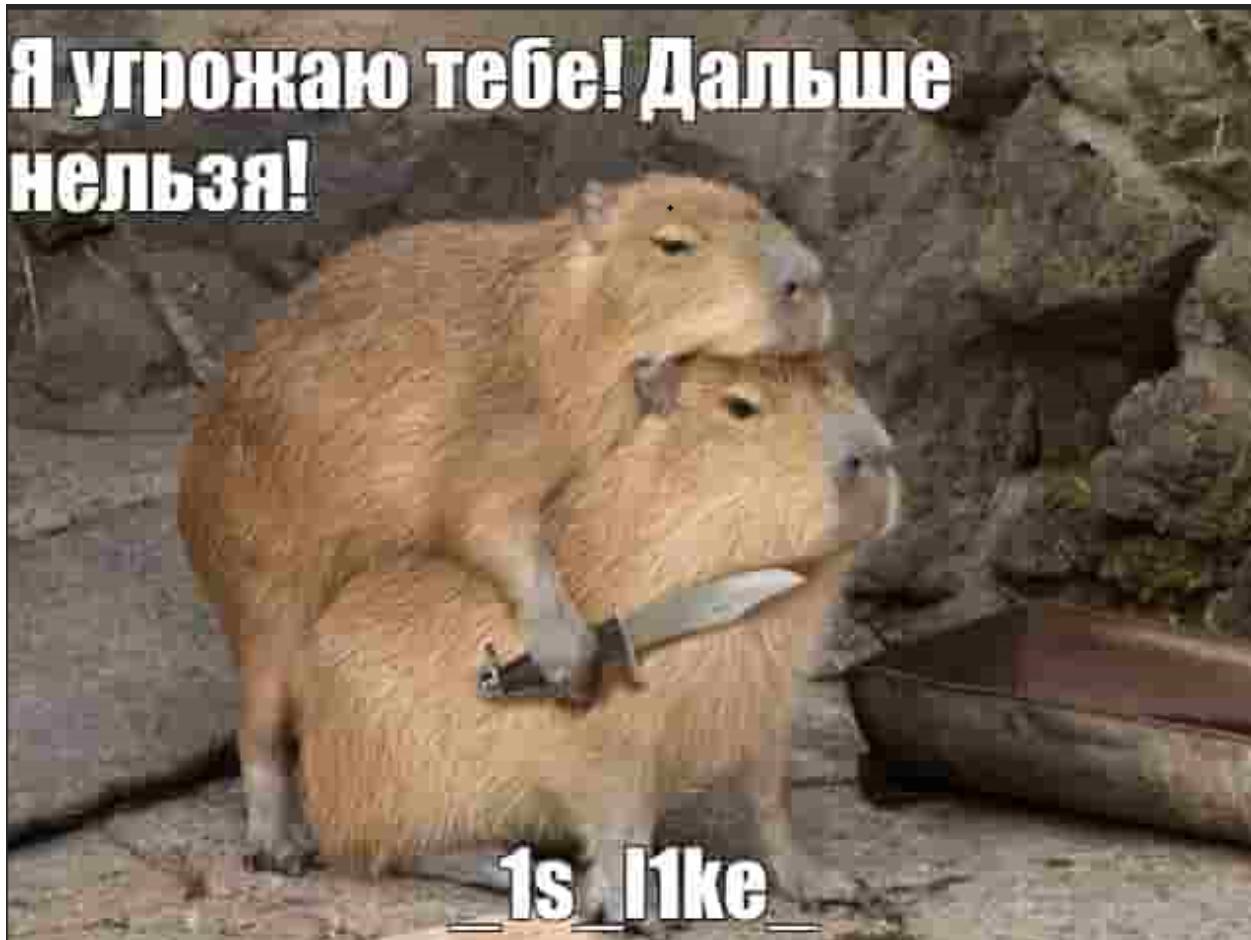
```
auto fname = "C:\\dump_mem.tar.xz";
auto address = 0xC0000D4B44;
auto size = 0x940c;
auto file= fopen(fname, "wb");

savefile(file, 0, address, size);
fclose(file);
```





Получили архив с новым заданием. И капибару .



Декомпилируем `3.py` через `uncompyle6`:

```
C:\Users\KEKC\Desktop>uncompyle6 3.py
# uncompyle6 version 3.9.0
# Python bytecode version base 3.8.0 (3413)
# Decompiled from: Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)]
# Embedded file name: 3.py
# Compiled at: 2023-04-16 20:06:25
# Size of source mod 2**32: 205944 bytes
string1 = 'Замени массив на что-то...'
elf = [127, 127]
string2 = 'Замени массив на что-то...'
capybara = [
    5, 12]
check = [
    61, 111, 64, 64, 56, 146, 159, 104, 111, 36, 56, 146, 130, 97, 60,
    105, 126, 121]
inp_str = input('Введи некоторую строку: ')
arr = []
len_str = len(inp_str)
for i in range(len_str):
    temp = ord(inp_str[i]) ^ 96
    temp += 80
    temp ^= 34
    temp -= 20
    temp ^= i
    arr.append(temp)
else:
    if arr == check:
        f = open('./capybara.jpg', 'wb')
        print('Не так плохо)')
        for i in range(len(capybara)):
            c = capybara[i]
            c = (c + 177) % 256
            c = (c - 187) % 256
            c = c ^ 51
            c = (c - 4) % 256
            c = c ^ 85
            c = (c + 119) % 256
            c = c ^ 80
            capybara[i] = c
    else:
        f.write(bytes(capybara))
        f.close()
        for i in range(len(elf)):
            elf[1] = 69
# okay decompiling 3.py
```

Правим:

```
# uncompyle6 version 3.9.0
# Python bytecode version base 3.8.0 (3413)
# Decompiled from: Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)]
# Embedded file name: 3.py
# Compiled at: 2023-04-16 20:06:25
# Size of source mod 2**32: 205944 bytes
string1 = 'Замени массив на что-то...'
elf = [127, 127]
```

```

string2 = 'Замени массив на что-то...'
capybara = [ 5, 12]
check = [ 61, 111, 64, 64, 56, 146, 159, 104, 111, 36, 56, 146,
105, 126, 121]
inp_str = input('Введи некоторую строку: ')
arr = []
len_str = len(inp_str)
for i in range(len_str):
    temp = ord(inp_str[i]) ^ 96
    temp += 80
    temp ^= 34
    temp -= 20
    temp ^= i
    arr.append(temp)
if arr == check:
    f = open('./capybara.jpg', 'wb')
    print('Не так плохо)')
    for i in range(len(capybara)):
        c = capybara[i]
        c = (c + 177) % 256
        c = (c - 187) % 256
        c = c ^ 51
        c = (c - 4) % 256
        c = c ^ 85
        c = (c + 119) % 256
        c = c ^ 80
        capybara[i] = c
    f.write(bytes(capybara))
    f.close()
    for i in range(len(elf)):
        elf[i] = 69
# okay decompiling 3.pyrc

```

Пишем кейген, чтобы получить данные из `data.rar` (под паролем).

```
check = [ 61, 111, 64, 64, 56, 146, 159, 104, 111, 36, 56, 146,
105, 126, 121]
string = ""

for j in range(len(check)):
    for i in range(0x20, 128):
        temp = i ^ 96
        temp += 80
        temp ^= 34
        temp -= 20
        temp ^= j

        if (temp == check[j]):
            string += chr(i)
            break

print(string)
```

Запускаем и получаем: `C0DEBY_1ist_P2th0n` - это пароль к архиву

Восстанавливаем исходный код (приложен к райтапу), используя данные из архива. Добавим некоторый код для удостов.

Получили новую капибару



А также 4.elf .

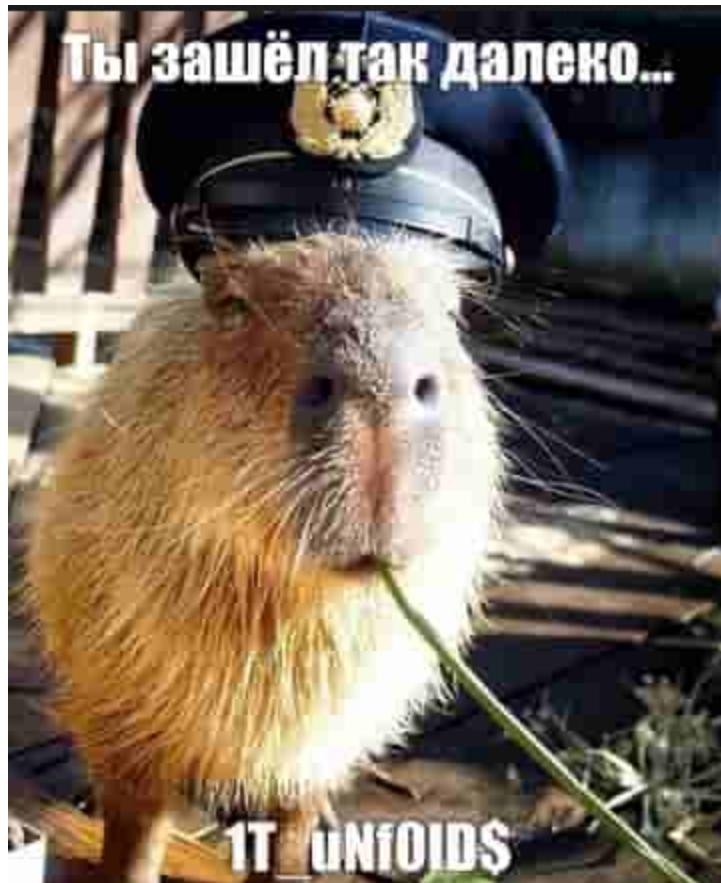
## Изучаем новый файл.

Всё так же как и в прошлый раз. После его изучения пишем кейген.

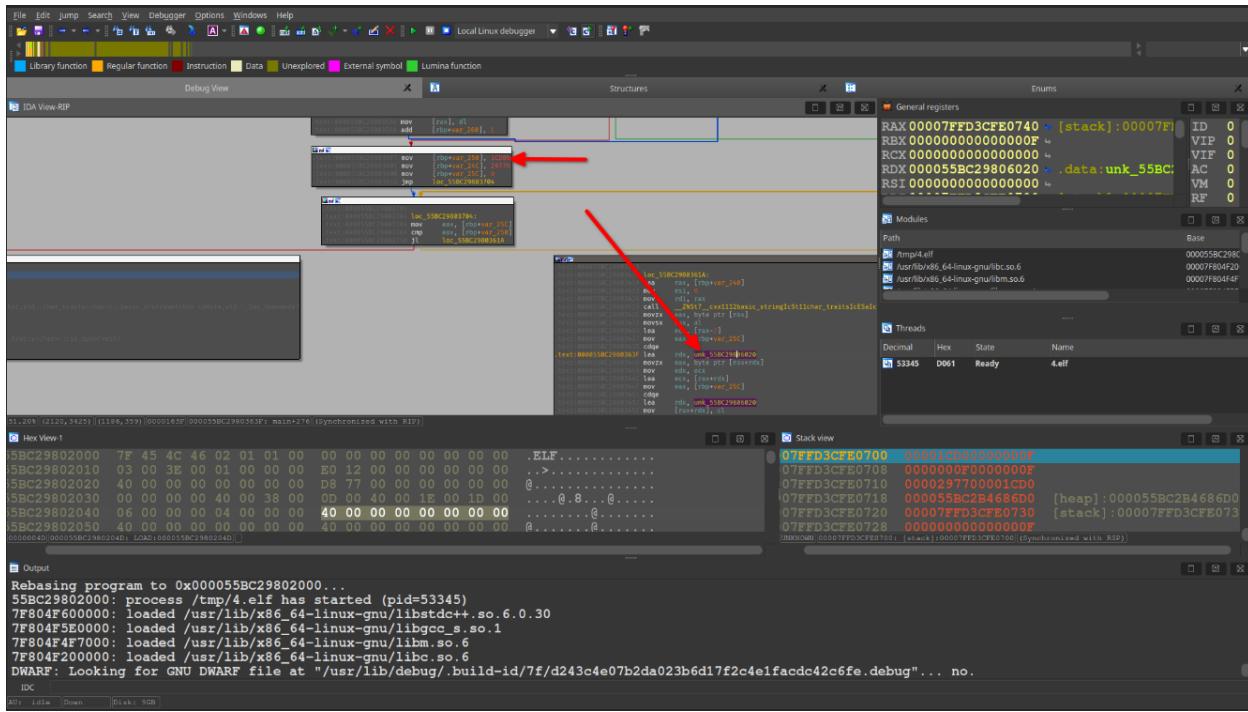
```
int len_check = sizeof(bytes)/sizeof(bytes[0]);\n\nfor (int i = 0; i < len_check; i++) {\n    for (int j = 0x20; j <= 127; j++) {\n        unsigned char temp = j;\n        temp ^= 0x20;\n        temp ^= 0x23;\n        temp -= 0x2;\n        temp ^= 0x55;\n        temp += 0x77;\n        temp ^= 0xff;\n\n        if (temp == bytes[i]) {\n            str[i] = j;\n            break;\n        }\n    }\n}\nprintf("%s\\n", str);\nreturn 0;\n}
```

Получили: [CODEBY\\_C@PYBARA](#)

При успешном вводе получаем новую капибару.



И дампим последний файл из памяти.

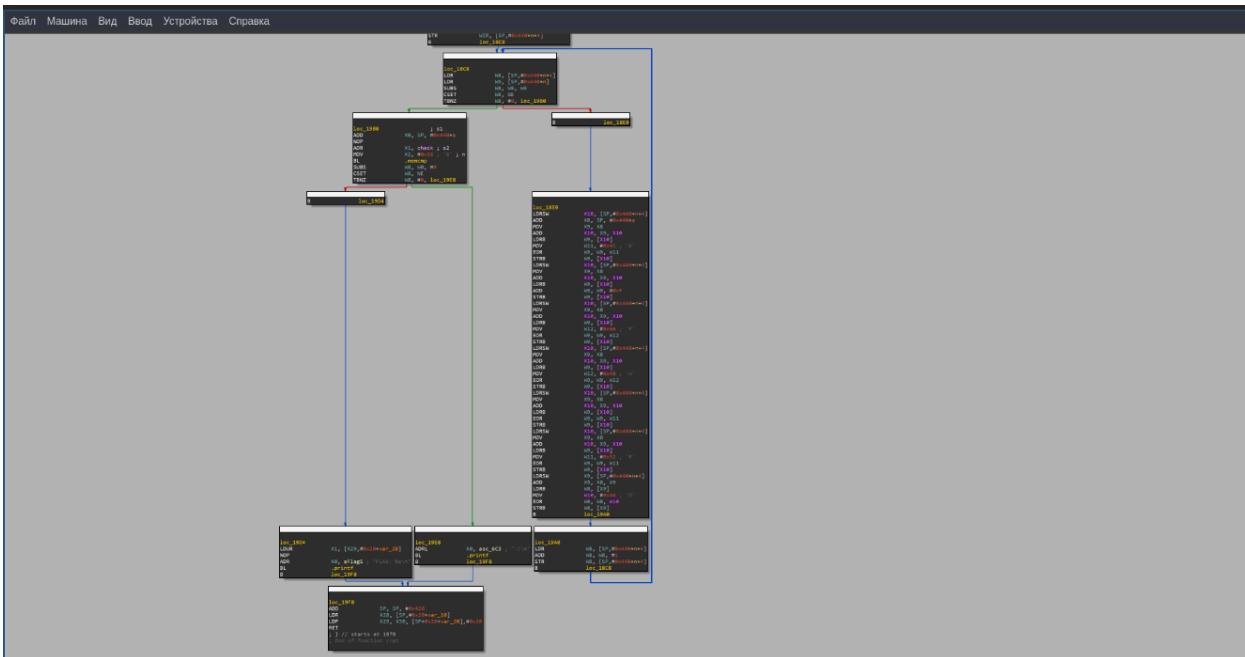


Используем этот скрипт:

```
auto fname = "/tmp/5.elf";
auto address = 0x55BC29806020;
auto size = 7376;
auto file= fopen(fname, "wb");

savefile(file, 0, address, size);
fclose(file);
```

Изучаем последний файл. Он под ARM. Поэтому запускаем его либо под ARM-процессором, либо только статикой изучаем.



Пишем кейген:

```
#include <stdio.h>

unsigned char bytes[] = {0x68, 0x64, 0x6d, 0x6a, 0x6b, 0x5e, 0x6f};

int main() {

    char str[60] = {0};

    int len_check = sizeof(bytes)/sizeof(bytes[0]);

    for (int i = 0; i < len_check; i++) {
        for (int j = 0x20; j <= 127; j++) {
            unsigned char temp = j;
            temp ^= 0x41;
            temp += 15;
            temp ^= 0x66;
```

```
temp ^= 'H';
temp ^= 'A';
temp ^= 'R';
temp ^= 'D';

if (temp == bytes[i]) {
    str[i] = j;
    break;
}
printf("%s\n", str);
return 0;
}
```

Запускаем и получим флаг.