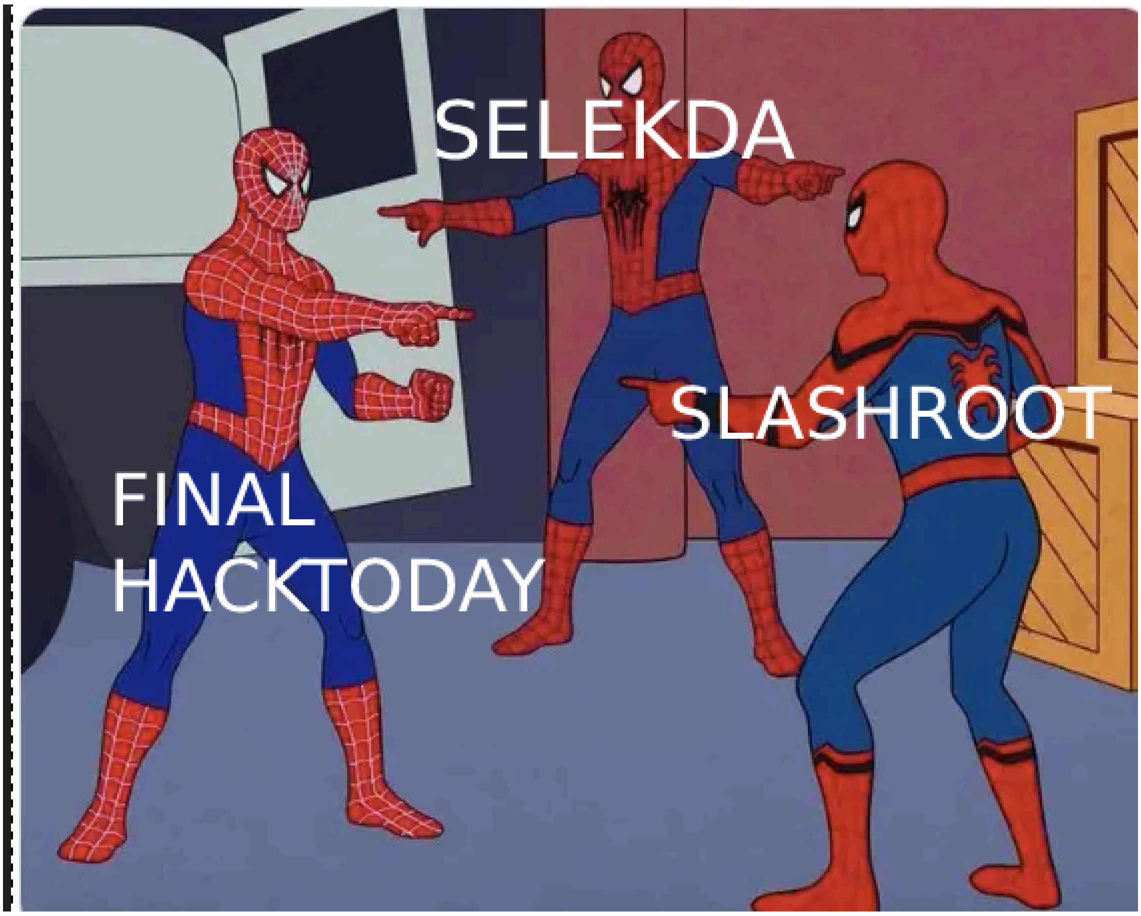
Writeup Day 3 Selekda WS ASEAN 2024

bengsky x msfir



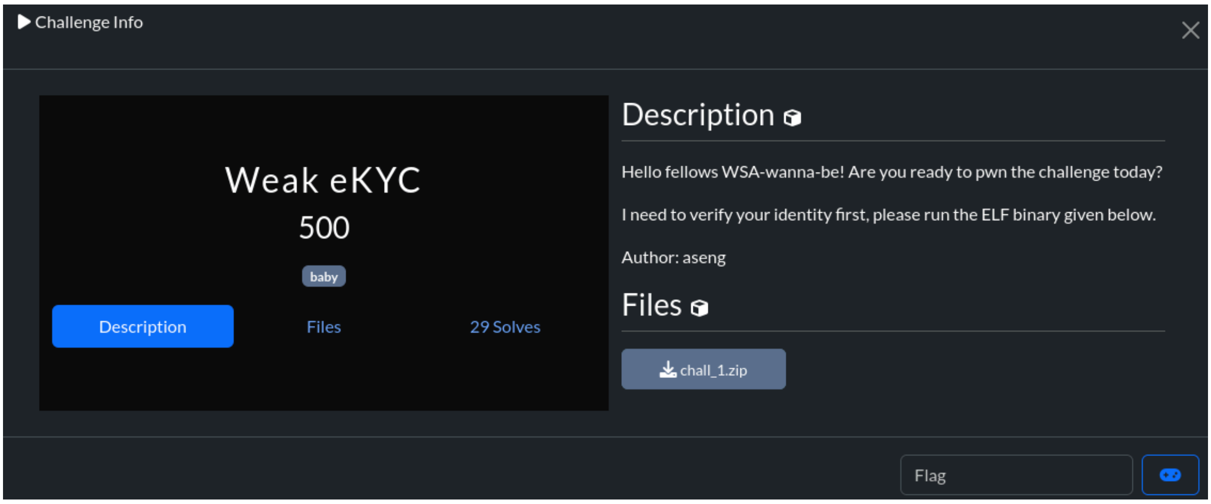
bengsky

msfir

Daftar Isi

Reverse Engineering

[500 pts] Weak eKYC



Challenge Info

Weak eKYC

500

baby

Description

Files

29 Solves

Description

Hello fellows WSA-wanna-be! Are you ready to pwn the challenge today?

I need to verify your identity first, please run the ELF binary given below.

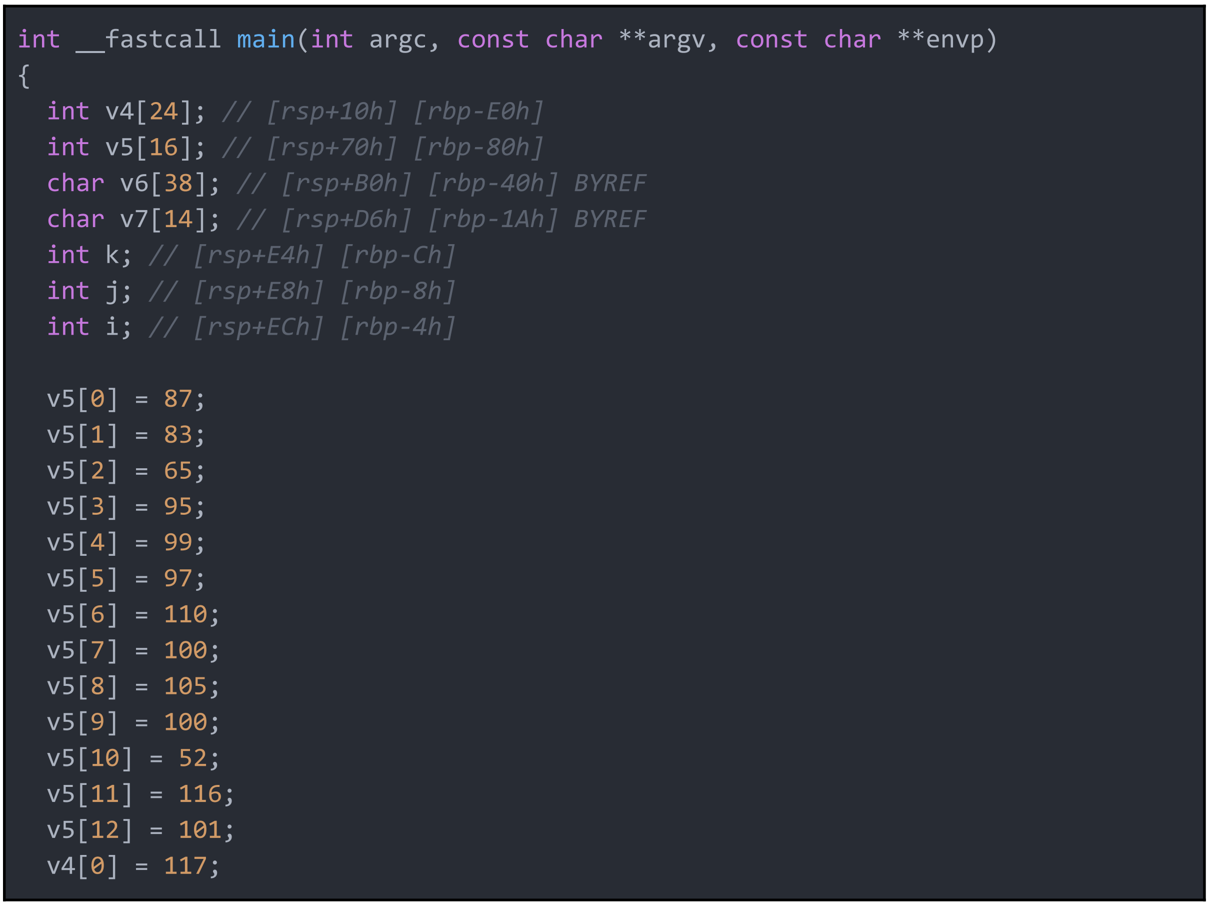
Author: aseng

Files

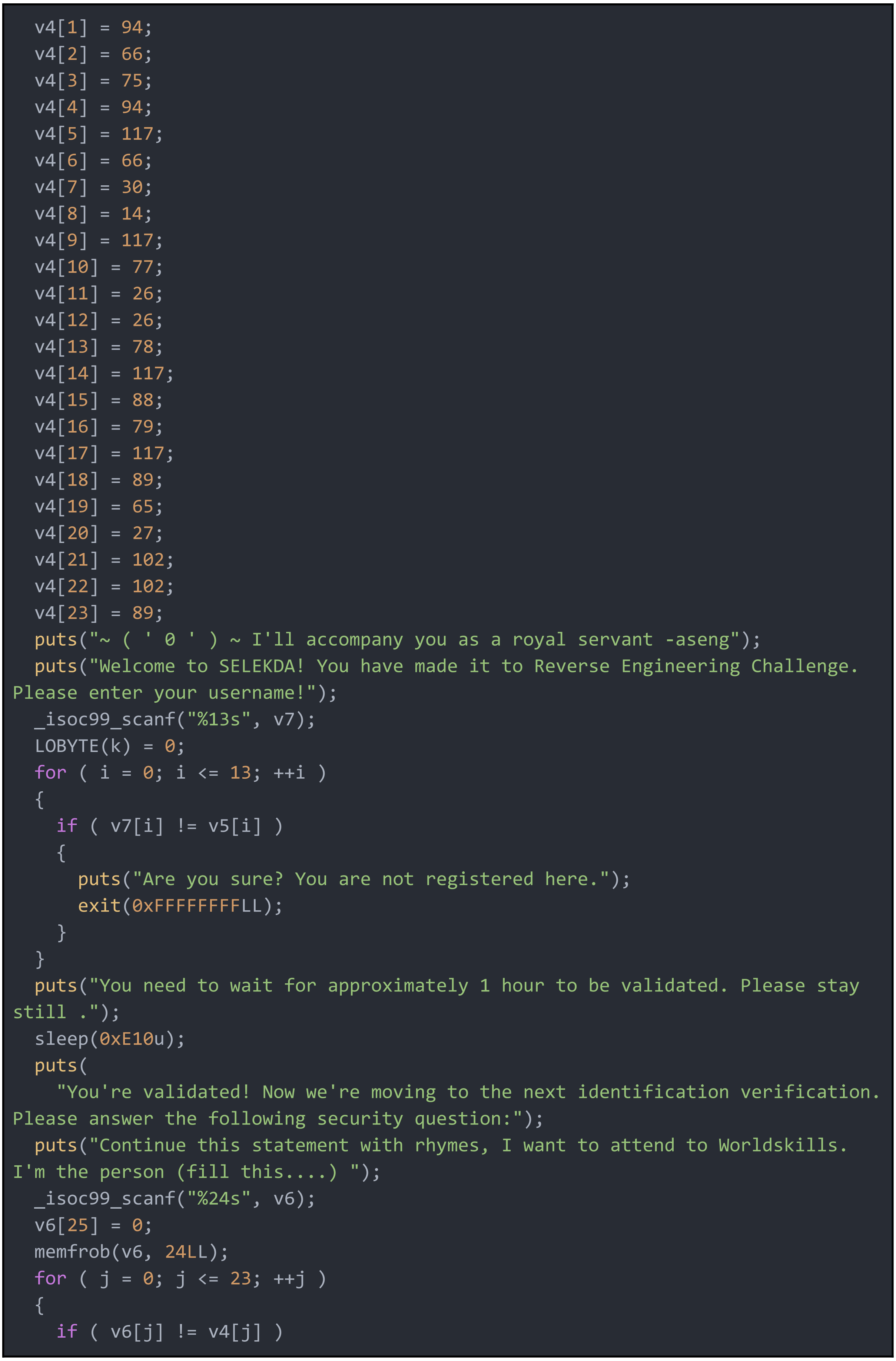
chall_1.zip

Flag

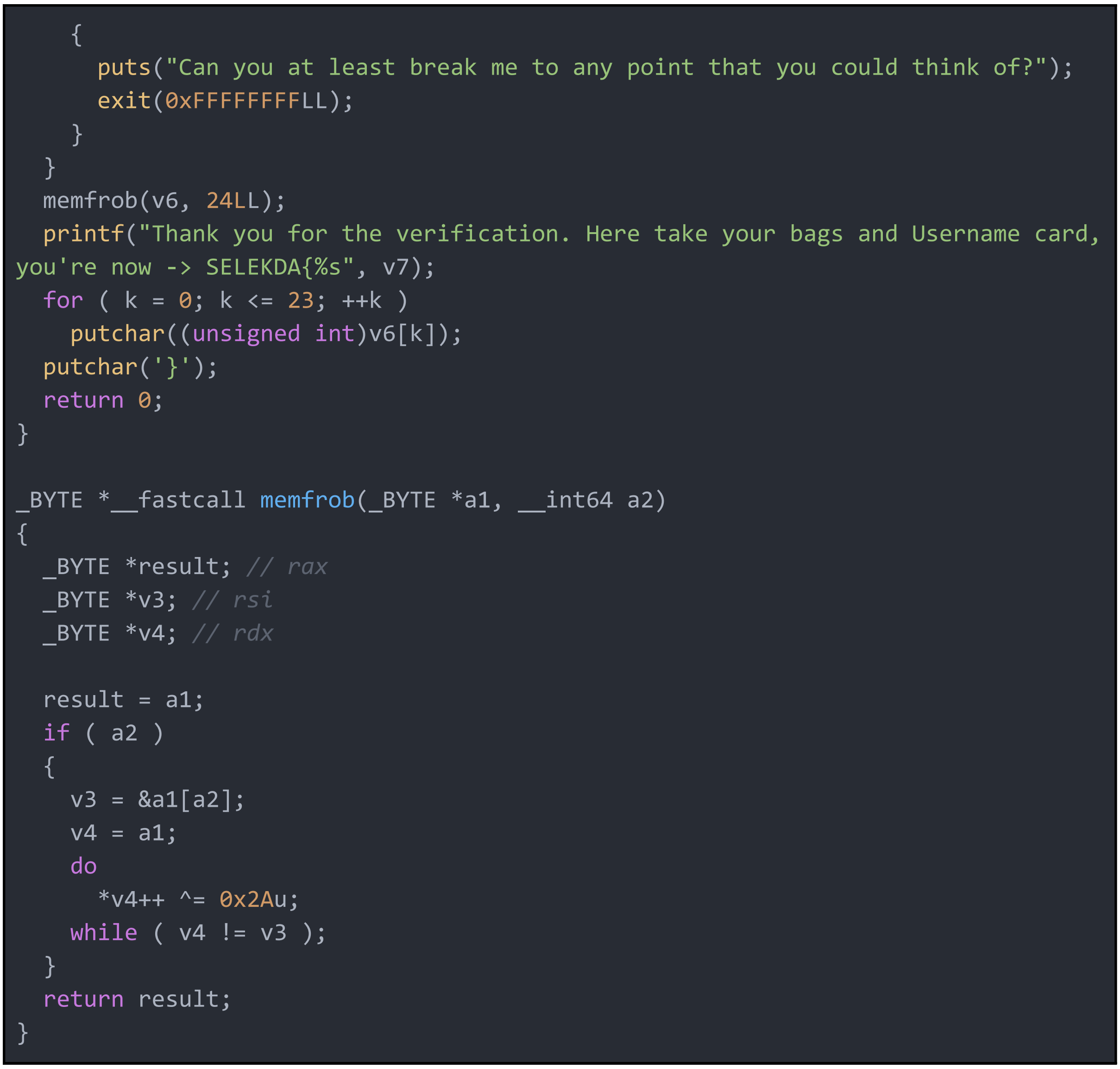
Diberikan sebuah linux binary. Pertama-tama kita buka dengan IDA, lalu decompile. Berikut hasil decompilenya:



```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int v4[24]; // [rsp+10h] [rbp-E0h]
  int v5[16]; // [rsp+70h] [rbp-80h]
  char v6[38]; // [rsp+B0h] [rbp-40h] BYREF
  char v7[14]; // [rsp+D6h] [rbp-1Ah] BYREF
  int k; // [rsp+E4h] [rbp-Ch]
  int j; // [rsp+E8h] [rbp-8h]
  int i; // [rsp+ECh] [rbp-4h]

  v5[0] = 87;
  v5[1] = 83;
  v5[2] = 65;
  v5[3] = 95;
  v5[4] = 99;
  v5[5] = 97;
  v5[6] = 110;
  v5[7] = 100;
  v5[8] = 105;
  v5[9] = 100;
  v5[10] = 52;
  v5[11] = 116;
  v5[12] = 101;
  v4[0] = 117;
```

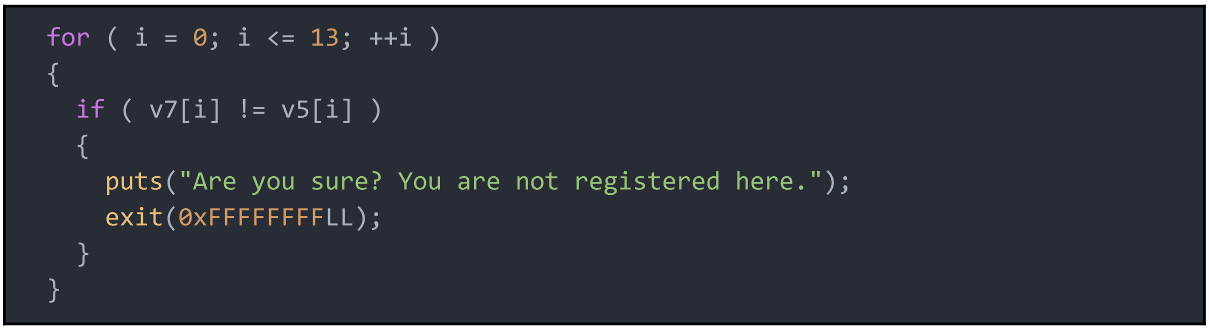


```
  v4[1] = 94;
  v4[2] = 66;
  v4[3] = 75;
  v4[4] = 94;
  v4[5] = 117;
  v4[6] = 66;
  v4[7] = 30;
  v4[8] = 14;
  v4[9] = 117;
  v4[10] = 77;
  v4[11] = 26;
  v4[12] = 26;
  v4[13] = 78;
  v4[14] = 117;
  v4[15] = 88;
  v4[16] = 79;
  v4[17] = 117;
  v4[18] = 89;
  v4[19] = 65;
  v4[20] = 27;
  v4[21] = 102;
  v4[22] = 102;
  v4[23] = 89;
  puts("~ ( ' 0 ' ) ~ I'll accompany you as a royal servant -aseng");
  puts("Welcome to SELEKDA! You have made it to Reverse Engineering Challenge.
Please enter your username!");
  _isoc99_scanf("%13s", v7);
  LOBYTE(k) = 0;
  for ( i = 0; i <= 13; ++i )
  {
    if ( v7[i] != v5[i] )
    {
      puts("Are you sure? You are not registered here.");
      exit(0xFFFFFFFFLL);
    }
  }
  puts("You need to wait for approximately 1 hour to be validated. Please stay
still .");
  sleep(0xE10u);
  puts(
    "You're validated! Now we're moving to the next identification verification.
Please answer the following security question:");
  puts("Continue this statement with rhymes, I want to attend to Worldskills.
I'm the person (fill this....) ");
  _isoc99_scanf("%24s", v6);
  v6[25] = 0;
  memfrob(v6, 24LL);
  for ( j = 0; j <= 23; ++j )
  {
    if ( v6[j] != v4[j] )
```



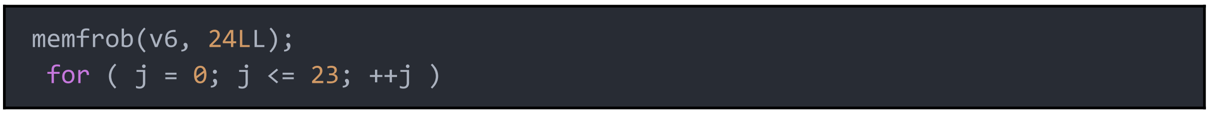
```
    {
      puts("Can you at least break me to any point that you could think of?");
      exit(0xFFFFFFFFLL);
    }
  }
  memfrob(v6, 24LL);
  printf("Thank you for the verification. Here take your bags and Username card, 
you're now -> SELEKDA{%s", v7);
  for ( k = 0; k <= 23; ++k )
    putchar((unsigned int)v6[k]);
  putchar('}');
  return 0;
}

_BYTE *__fastcall memfrob(_BYTE *a1, __int64 a2)
{
  _BYTE *result; // rax
  _BYTE *v3; // rsi
  _BYTE *v4; // rdx

  result = a1;
  if ( a2 )
  {
    v3 = &a1[a2];
    v4 = a1;
    do
      *v4++ ^= 0x2Au;
    while ( v4 != v3 );
  }
  return result;
}
```

Terlihat di atas bahwa kita akan diberikan flag setelah melewati 2 tahap, yaitu menginput registered username lalu melanjutkan sebuah kalimat.



```
  for ( i = 0; i <= 13; ++i )
  {
    if ( v7[i] != v5[i] )
    {
      puts("Are you sure? You are not registered here.");
      exit(0xFFFFFFFFLL);
    }
  }
```

Block for loop di atas digunakan untuk mengecek apakah username kita sudah benar. Artinya, username yang benar ada di variable v5.



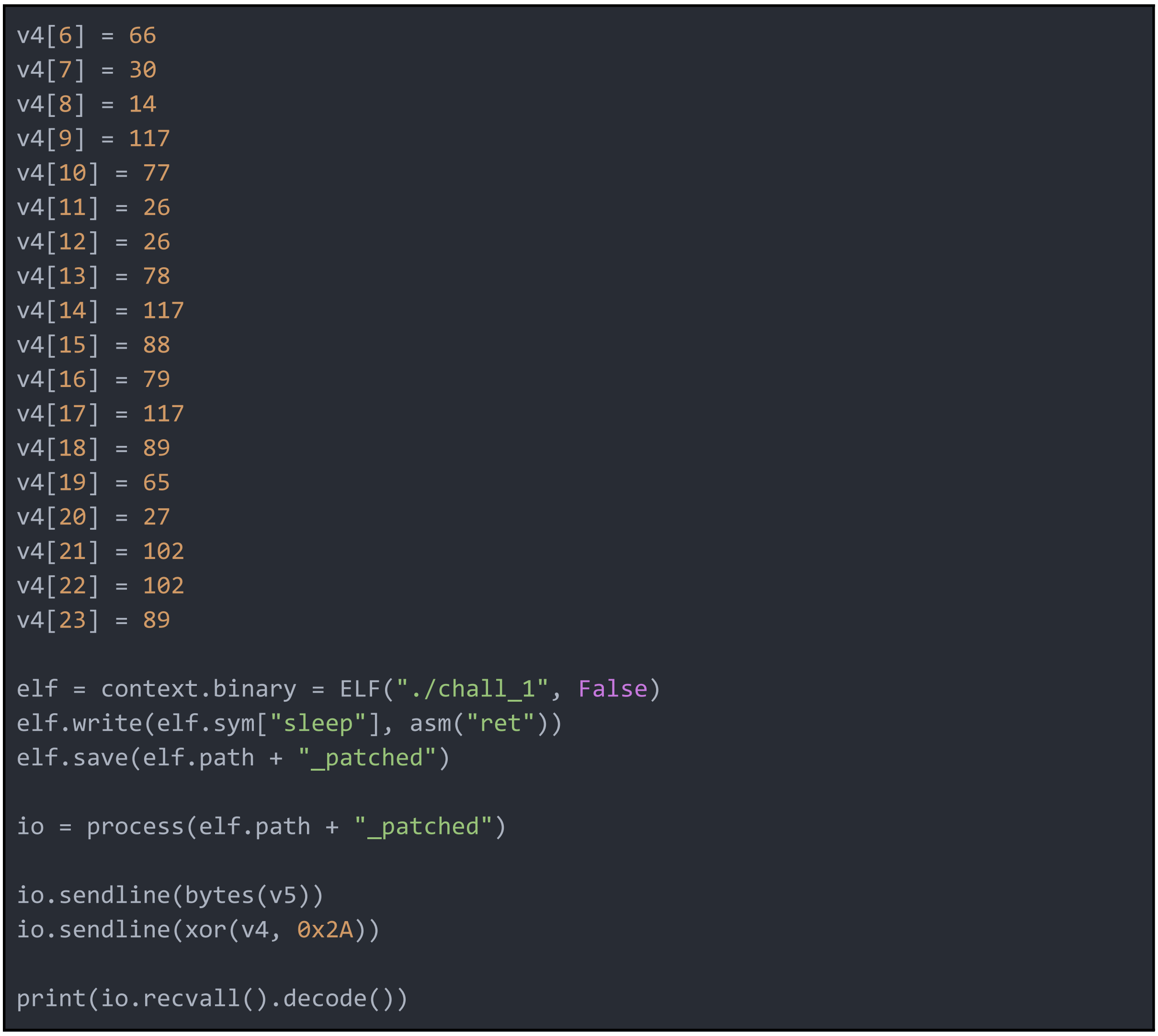
```
 memfrob(v6, 24LL);
  for ( j = 0; j <= 23; ++j )
  {
    if ( v6[j] != v4[j] )
    {
      puts("Can you at least break me to any point that you could think of?");
      exit(0xFFFFFFFFLL);
    }
  }
```

Selanjutnya program mengecek apakah kita menginput kalimat terusan yang benar. Namun, input kita yang ada pada variable v6 ditransformasi terlebih dahulu dengan fungsi memfrob. Fungsi tersebut melakukan xor ciphering dengan key 0x2A. Untuk menyelesaikan soal ini, kita hanya perlu mengambil isi dari variable v5 untuk username dan isi dari v4 yang telah dixor dengan key 0x2A untuk kalimat terusan.

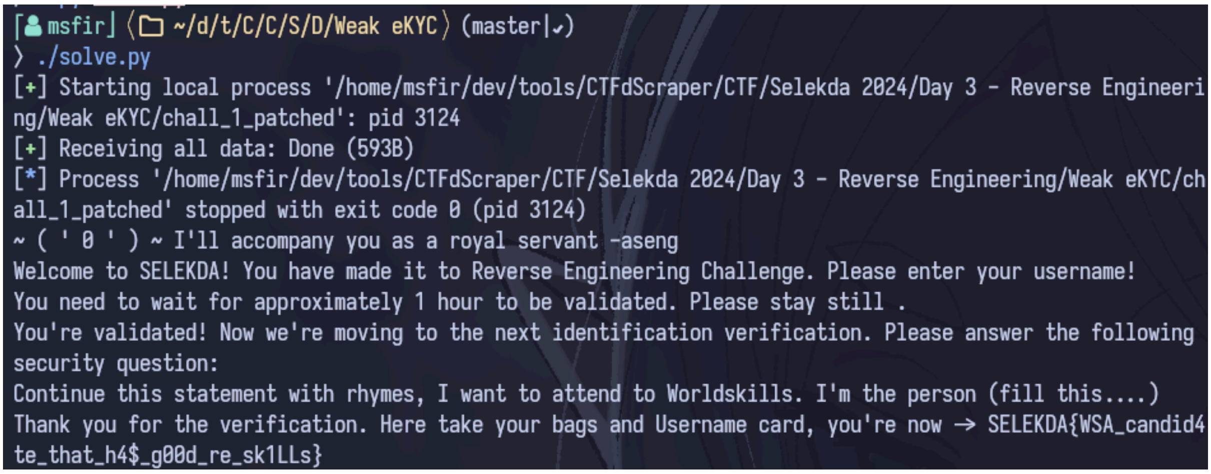
Akan tetapi, pada program tersebut terdapat pemanggilan fungsi sleep(0xe10) yang menyebabkan program akan sleep selama 1 jam. Tentu saja kita tidak ingin menunggu selama itu. Solusinya adalah dengan melakukan patch terhadap fungsi sleep yaitu dengan menyimpan instruksi ret sebagai instruksi pertama dari fungsi sleep. Patch bisa kita lakukan dengan menggunakan pwntools.

Solver:

```
#!/usr/bin/env python3

from pwn import ELF, asm, context, process, xor

v5 = bytearray(13)
v4 = bytearray(24)

v5[0] = 87
v5[1] = 83
v5[2] = 65
v5[3] = 95
v5[4] = 99
v5[5] = 97
v5[6] = 110
v5[7] = 100
v5[8] = 105
v5[9] = 100
v5[10] = 52
v5[11] = 116
v5[12] = 101

v4[0] = 117
v4[1] = 94
v4[2] = 66
v4[3] = 75
v4[4] = 94
v4[5] = 117
```



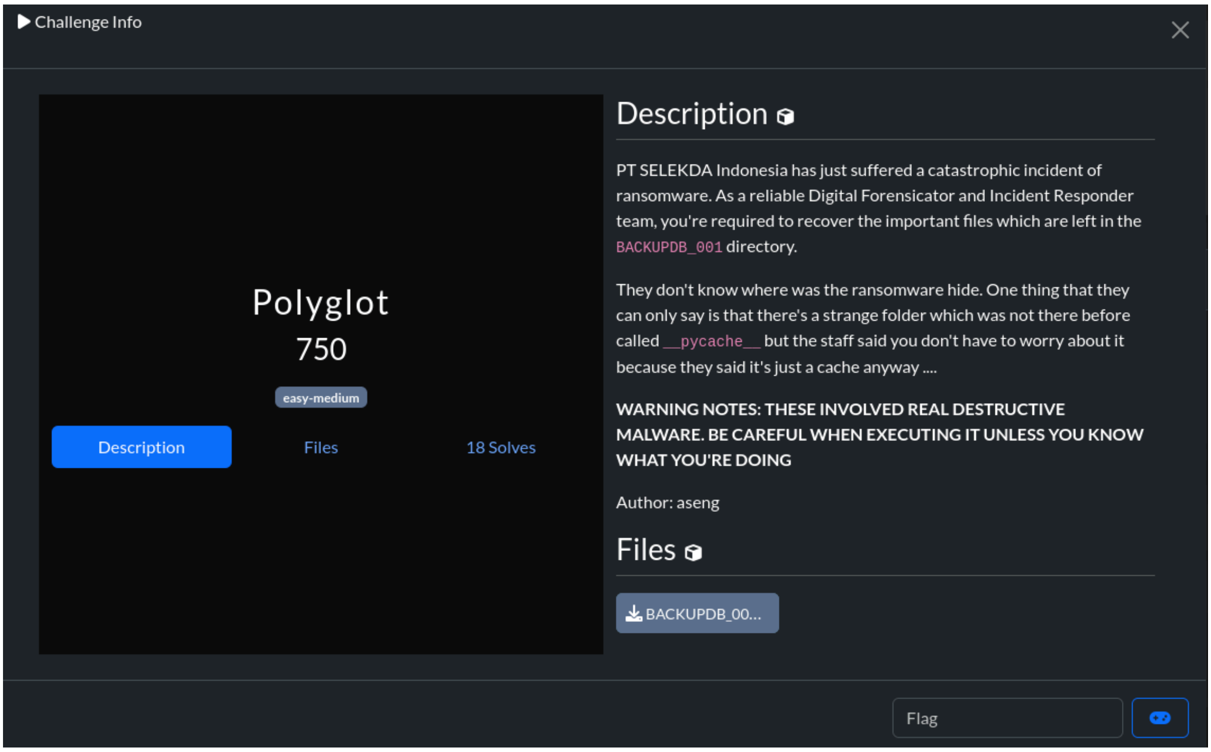
```
v4[6] = 66
v4[7] = 30
v4[8] = 14
v4[9] = 117
v4[10] = 77
v4[11] = 26
v4[12] = 26
v4[13] = 78
v4[14] = 117
v4[15] = 88
v4[16] = 79
v4[17] = 117
v4[18] = 89
v4[19] = 65
v4[20] = 27
v4[21] = 102
v4[22] = 102
v4[23] = 89

elf = context.binary = ELF("./chall_1", False)
elf.write(elf.sym["sleep"], asm("ret"))
elf.save(elf.path + "_patched")

io = process(elf.path + "_patched")

io.sendline(bytes(v5))
io.sendline(xor(v4, 0x2A))

print(io.recvall().decode())
```



```
[msfir] ⟨ ~/d/t/C/C/S/D/Weak eKYC⟩ (master|✓)
⟩ ./solve.py
[+] Starting local process '/home/msfir/dev/tools/CTFdScraper/CTF/Selekda 2024/Day 3 - Reverse Engineeri
ng/Weak eKYC/chall_1_patched': pid 3124
[+] Receiving all data: Done (593B)
[*] Process '/home/msfir/dev/tools/CTFdScraper/CTF/Selekda 2024/Day 3 - Reverse Engineering/Weak eKYC/ch
all_1_patched' stopped with exit code 0 (pid 3124)
~ ( ' 0 ' ) ~ I'll accompany you as a royal servant -aseng
Welcome to SELEKDA! You have made it to Reverse Engineering Challenge. Please enter your username!
You need to wait for approximately 1 hour to be validated. Please stay still .
You're validated! Now we're moving to the next identification verification. Please answer the following
security question:
Continue this statement with rhymes, I want to attend to Worldskills. I'm the person (fill this....)
Thank you for the verification. Here take your bags and Username card, you're now -> SELEKDA{WSA_candid4
te_that_h4$_g00d_re_sk1LLs}
```

Flag: SELEKDA{WSA_candid4te_that_h4$_g00d_re_sk1LLs}

[750 pts] Polyglot



▶ Challenge Info

Polyglot

750

Description

PT SELEKDA Indonesia has just suffered a catastrophic incident of ransomware. As a reliable Digital Forensicator and Incident Responder team, you're required to recover the important files which are left in the `BACKUPDB_001` directory.

They don't know where was the ransomware hide. One thing that they can only say is that there's a strange folder which was not there before called `__pycache__` but the staff said you don't have to worry about it because they said it's just a cache anyway ....

WARNING NOTES: THESE INVOLVED REAL DESTRUCTIVE MALWARE. BE CAREFUL WHEN EXECUTING IT UNLESS YOU KNOW WHAT YOU'RE DOING

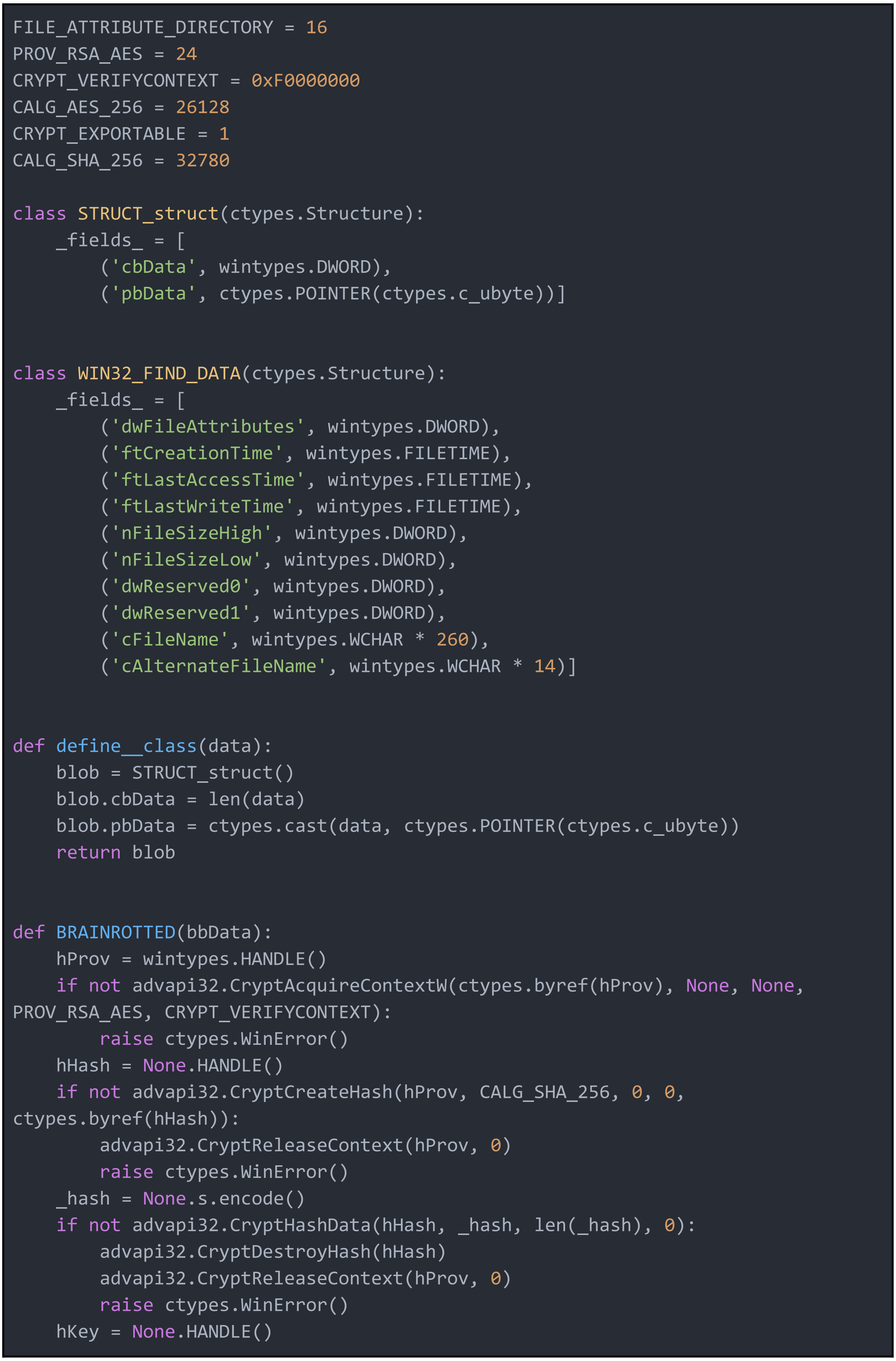
Author: aseng

Files

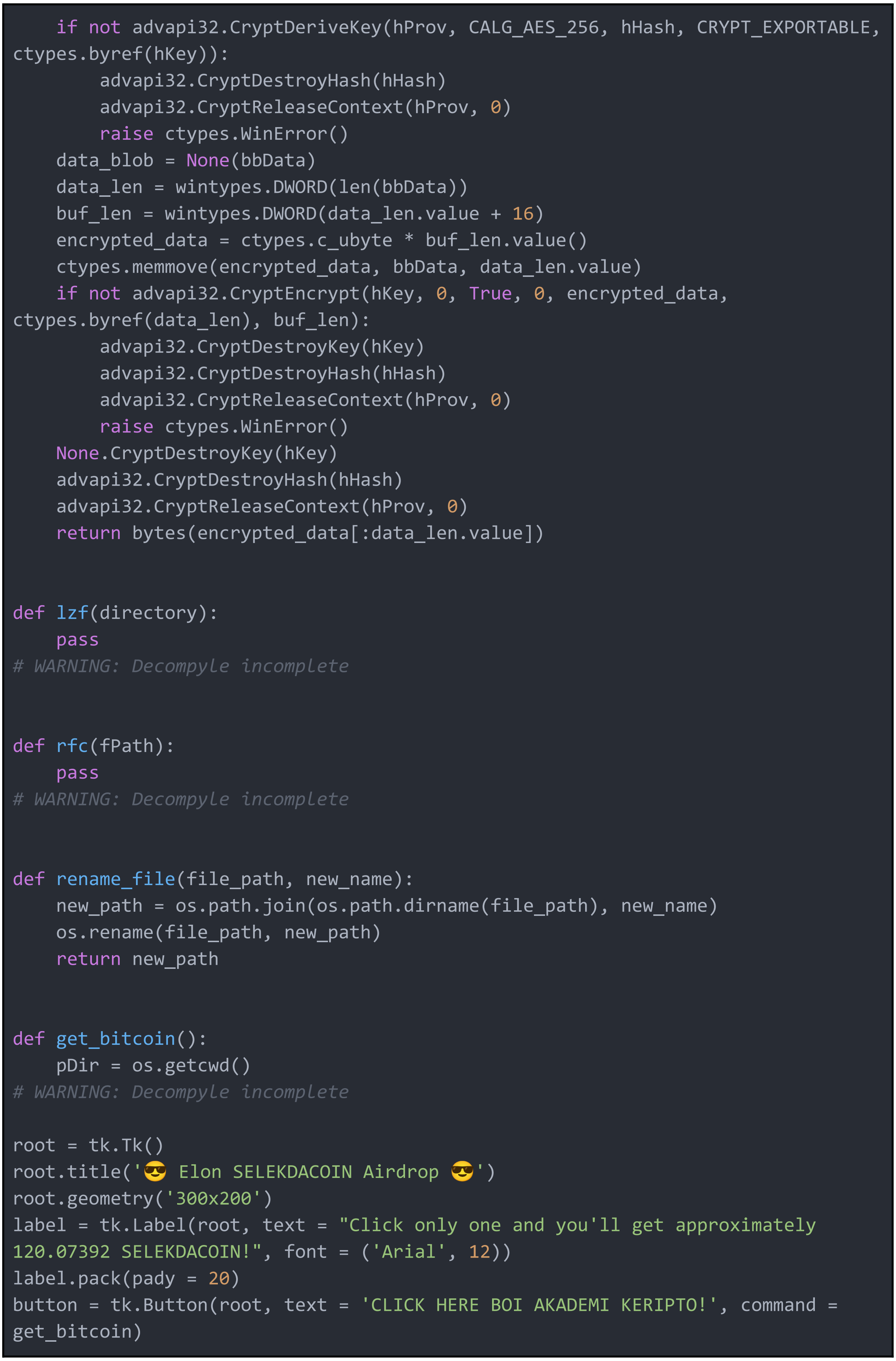
Diberikan sebuah zip file yang berisi file-file yang telah dienkripsi oleh malware. Berdasarkan deskripsi soal, malware tersebut ada di dalam folder __pycache__. Ini artinya, malware yang ada berada dalam bentuk python bytecode.

Kita bisa menggunakan pycdc untuk mengkonversi python bytecode menjadi python script. Sayangnya pycdc tidak bisa melakukan konversi secara sempurna. Berikut hasil dari pycdc:

```
# Source Generated with Decompyle++
# File: thon.cpython-311.pyc (Python 3.11)

import tkinter as tk
from tkinter import messagebox
from Crypto.Util.number import *
import ctypes
from ctypes.wintypes import wintypes
import sys
import os
import re
import hashlib
import this
advapi32 = ctypes.windll.advapi32
kernel32 = ctypes.windll.kernel32
user32 = ctypes.windll.user32
INVALID_HANDLE_VALUE = wintypes.HANDLE(-1).value
```



```
FILE_ATTRIBUTE_DIRECTORY = 16
PROV_RSA_AES = 24
CRYPT_VERIFYCONTEXT = 0xF0000000
CALG_AES_256 = 26128
CRYPT_EXPORTABLE = 1
CALG_SHA_256 = 32780

class STRUCT_struct(ctypes.Structure):
    _fields_ = [
        ('cbData', wintypes.DWORD),
        ('pbData', ctypes.POINTER(ctypes.c_ubyte))]


class WIN32_FIND_DATA(ctypes.Structure):
    _fields_ = [
        ('dwFileAttributes', wintypes.DWORD),
        ('ftCreationTime', wintypes.FILETIME),
        ('ftLastAccessTime', wintypes.FILETIME),
        ('ftLastWriteTime', wintypes.FILETIME),
        ('nFileSizeHigh', wintypes.DWORD),
        ('nFileSizeLow', wintypes.DWORD),
        ('dwReserved0', wintypes.DWORD),
        ('dwReserved1', wintypes.DWORD),
        ('cFileName', wintypes.WCHAR * 260),
        ('cAlternateFileName', wintypes.WCHAR * 14)]


def define__class(data):
    blob = STRUCT_struct()
    blob.cbData = len(data)
    blob.pbData = ctypes.cast(data, ctypes.POINTER(ctypes.c_ubyte))
    return blob


def BRAINROTTED(bbData):
    hProv = wintypes.HANDLE()
    if not advapi32.CryptAcquireContextW(ctypes.byref(hProv), None, None,
PROV_RSA_AES, CRYPT_VERIFYCONTEXT):
        raise ctypes.WinError()
    hHash = None.HANDLE()
    if not advapi32.CryptCreateHash(hProv, CALG_SHA_256, 0, 0,
ctypes.byref(hHash)):
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    _hash = None.s.encode()
    if not advapi32.CryptHashData(hHash, _hash, len(_hash), 0):
        advapi32.CryptDestroyHash(hHash)
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    hKey = None.HANDLE()
```



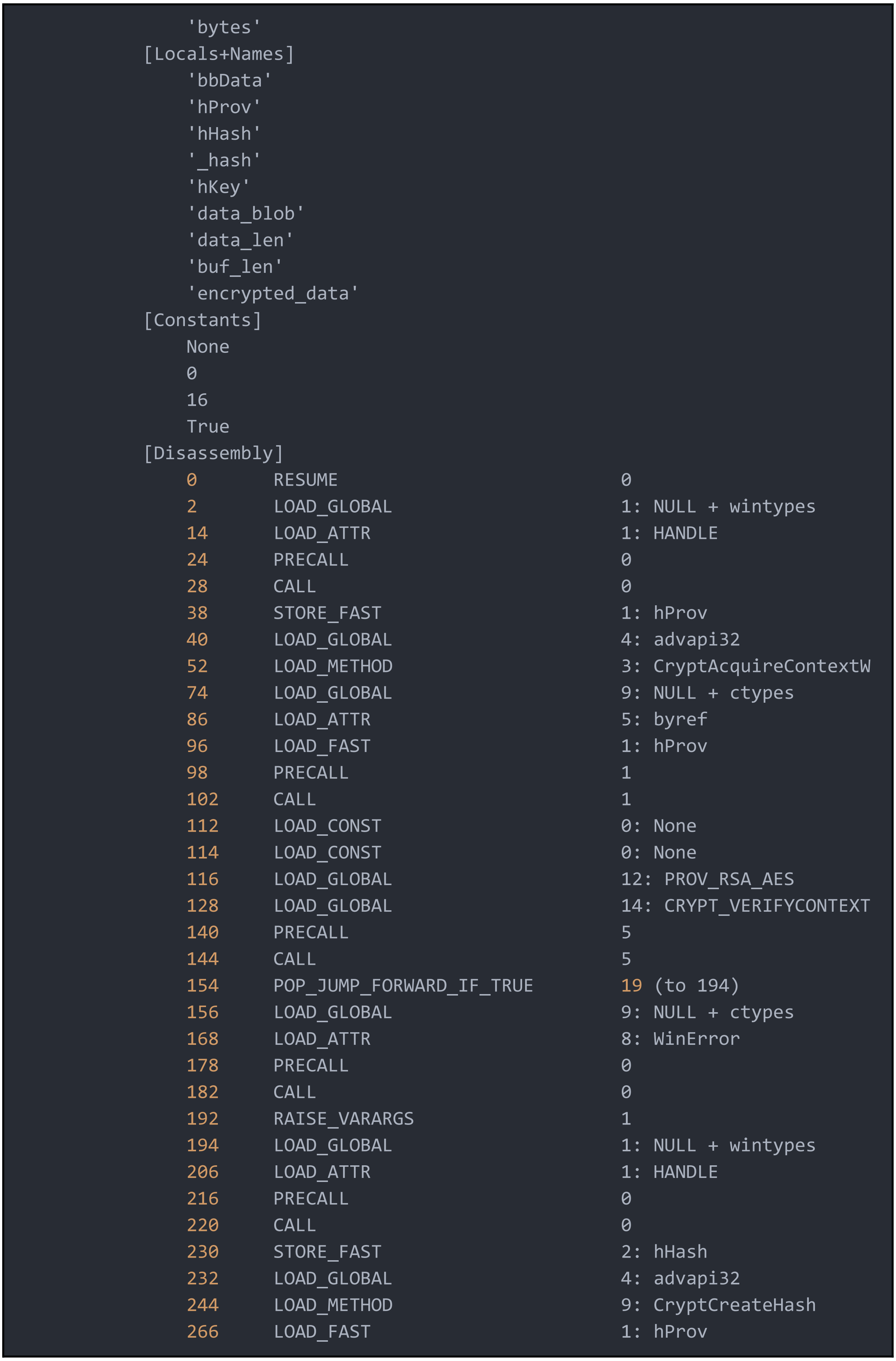
```
    if not advapi32.CryptDeriveKey(hProv, CALG_AES_256, hHash, CRYPT_EXPORTABLE,
ctypes.byref(hKey)):
        advapi32.CryptDestroyHash(hHash)
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    data_blob = None(bbData)
    data_len = wintypes.DWORD(len(bbData))
    buf_len = wintypes.DWORD(data_len.value + 16)
    encrypted_data = ctypes.c_ubyte * buf_len.value()
    ctypes.memmove(encrypted_data, bbData, data_len.value)
    if not advapi32.CryptEncrypt(hKey, 0, True, 0, encrypted_data,
ctypes.byref(data_len), buf_len):
        advapi32.CryptDestroyKey(hKey)
        advapi32.CryptDestroyHash(hHash)
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    None.CryptDestroyKey(hKey)
    advapi32.CryptDestroyHash(hHash)
    advapi32.CryptReleaseContext(hProv, 0)
    return bytes(encrypted_data[:data_len.value])


def lzf(directory):
    pass
# WARNING: Decompyle incomplete


def rfc(fPath):
    pass
# WARNING: Decompyle incomplete


def rename_file(file_path, new_name):
    new_path = os.path.join(os.path.dirname(file_path), new_name)
    os.rename(file_path, new_path)
    return new_path


def get_bitcoin():
    pDir = os.getcwd()
# WARNING: Decompyle incomplete

root = tk.Tk()
root.title('😎 Elon SELEKDACOIN Airdrop 😎')
root.geometry('300x200')
label = tk.Label(root, text = "Click only one and you'll get approximately
120.07392 SELEKDACOIN!", font = ('Arial', 12))
label.pack(pady = 20)
button = tk.Button(root, text = 'CLICK HERE BOI AKADEMI KERIPTO!', command =
get_bitcoin)
```

```
button.pack(pady = 10)
root.mainloop()
```

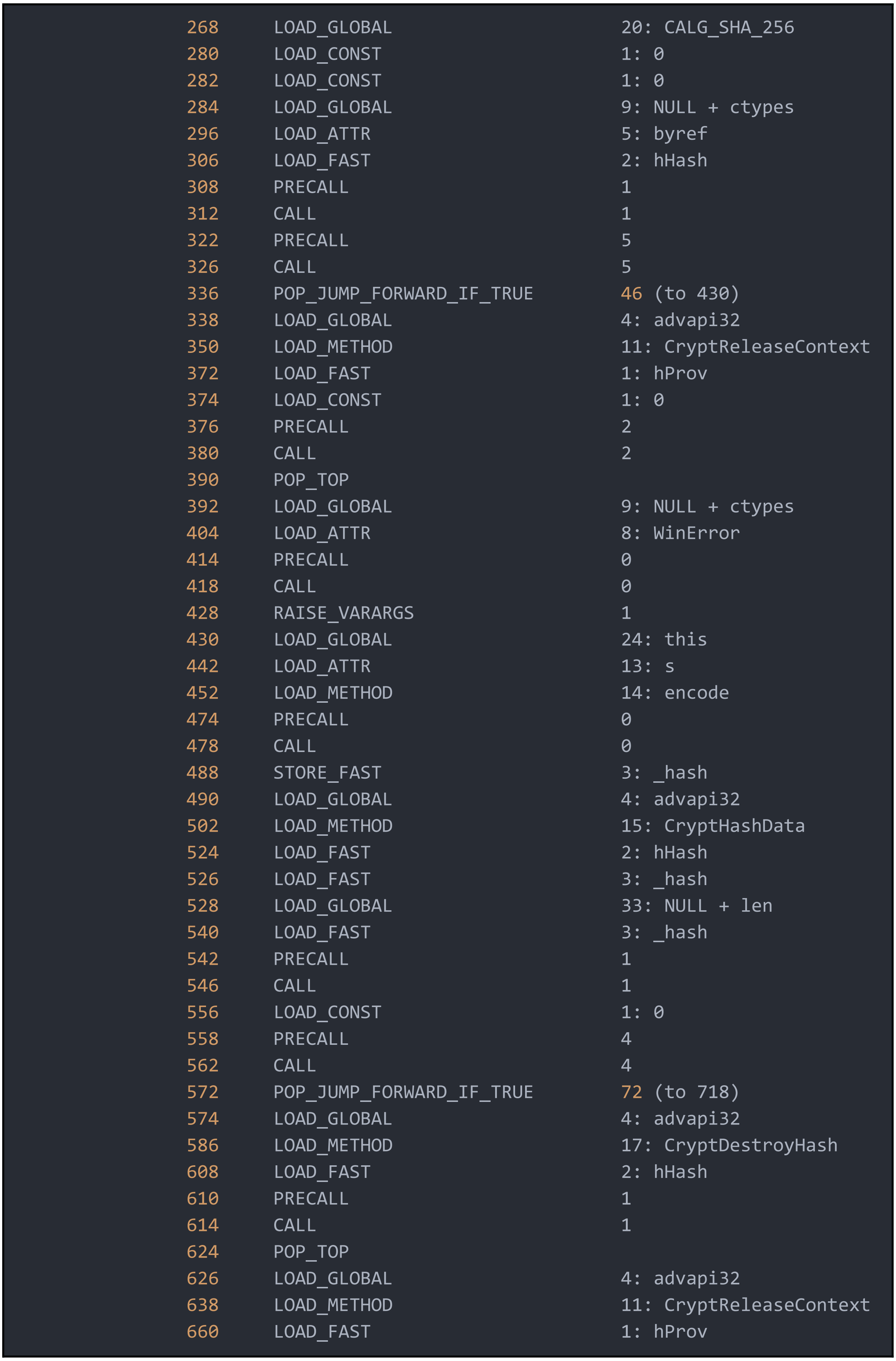
Terlihat di atas, script tersebut bukan sebuah python script yang valid. Namun, kita bisa mencoba memahaminya secara garis besar. Saya berasumsi bahwa fungsi BRAINROTTED adalah fungsi yang bertanggung jawab untuk melakukan enkripsi sehingga saya hanya akan fokus ke fungsi tersebut.

Sebelum kita analisis, kita perlu melakukan perbaikan terhadap fungsi ini. Saya melakukannya secara manual dengan menggunakan pycdas untuk mengubah “None” yang tidak valid.

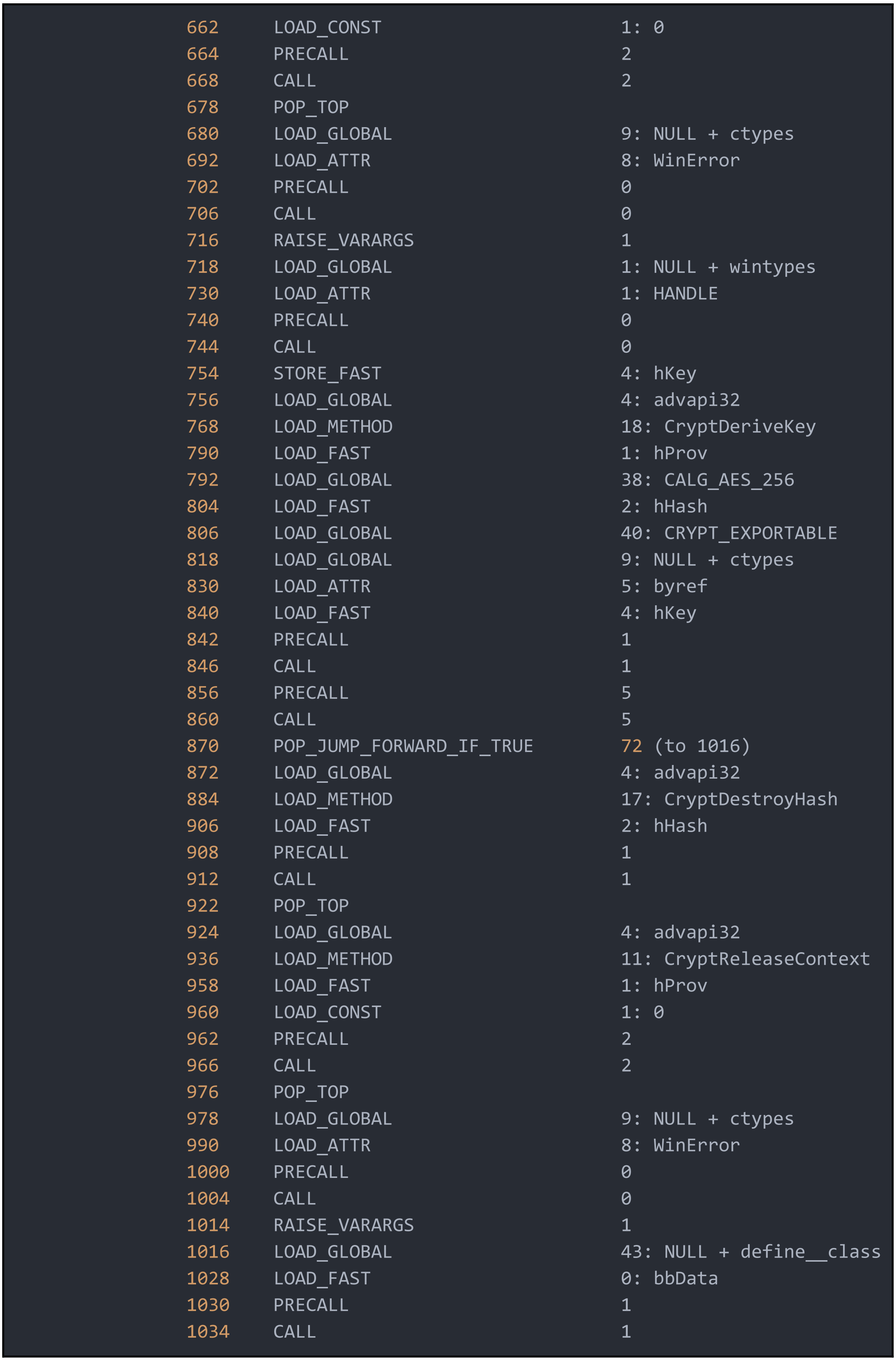
```
        [Code]
            File Name: thon.py
            Object Name: BRAINROTTED
            Qualified Name: BRAINROTTED
            Arg Count: 1
            Pos Only Arg Count: 0
            KW Only Arg Count: 0
            Stack Size: 10
            Flags: 0x00000003 (CO_OPTIMIZED | CO_NEWLOCALS)
            [Names]
                'wintypes'
                'HANDLE'
                'advapi32'
                'CryptAcquireContextW'
                'ctypes'
                'byref'
                'PROV_RSA_AES'
                'CRYPT_VERIFYCONTEXT'
                'WinError'
                'CryptCreateHash'
                'CALG_SHA_256'
                'CryptReleaseContext'
                'this'
                's'
                'encode'
                'CryptHashData'
                'len'
                'CryptDestroyHash'
                'CryptDeriveKey'
                'CALG_AES_256'
                'CRYPT_EXPORTABLE'
                'define__class'
                'DWORD'
                'value'
                'c_ubyte'
                'memmove'
                'CryptEncrypt'
                'CryptDestroyKey'
```



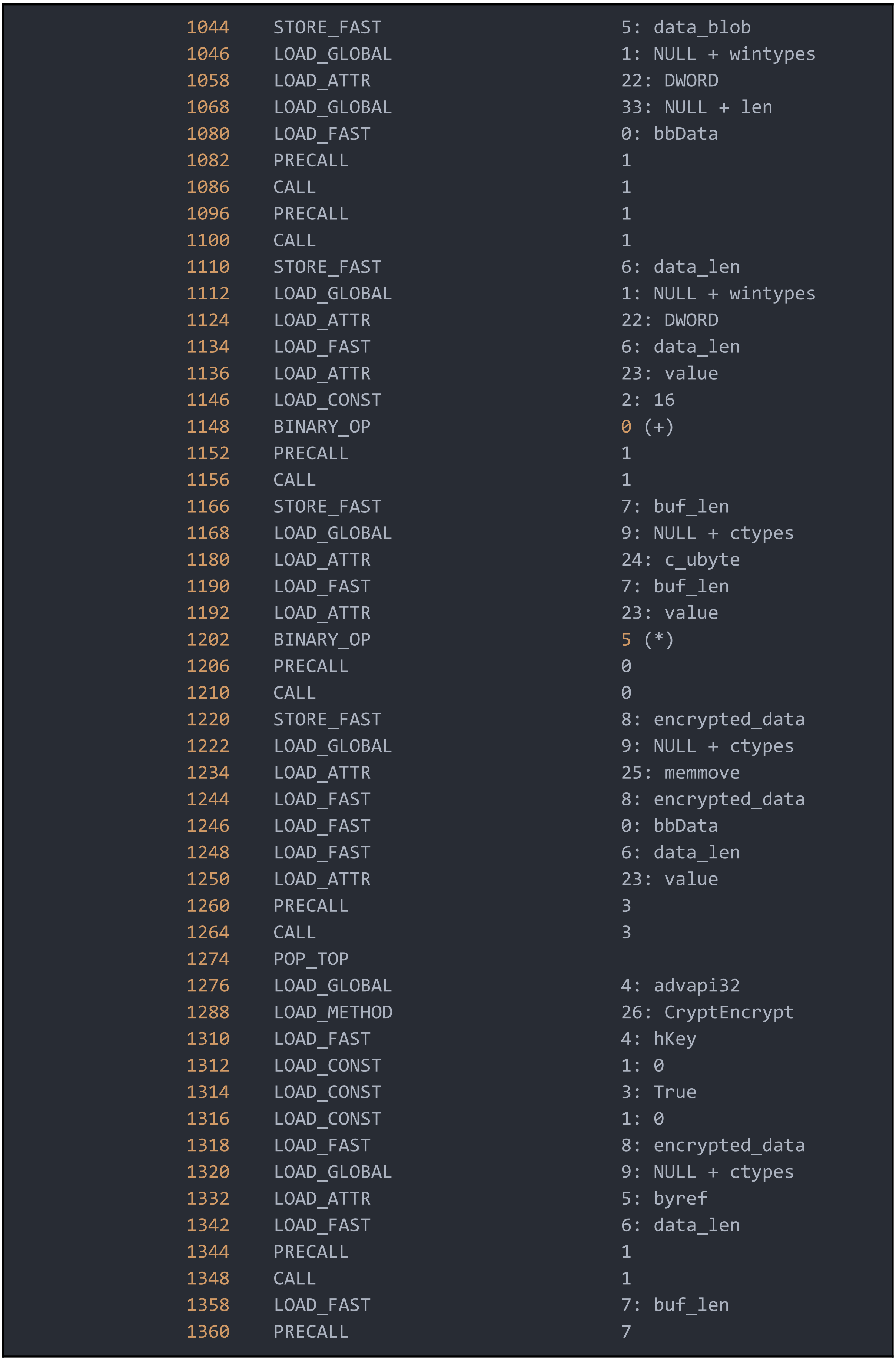
```
        'bytes'
    [Locals+Names]
        'bbData'
        'hProv'
        'hHash'
        '_hash'
        'hKey'
        'data_blob'
        'data_len'
        'buf_len'
        'encrypted_data'
    [Constants]
        None
        0
        16
        True
    [Disassembly]
        0       RESUME                          0
        2       LOAD_GLOBAL                     1: NULL + wintypes
        14      LOAD_ATTR                       1: HANDLE
        24      PRECALL                         0
        28      CALL                            0
        38      STORE_FAST                      1: hProv
        40      LOAD_GLOBAL                     4: advapi32
        52      LOAD_METHOD                     3: CryptAcquireContextW
        74      LOAD_GLOBAL                     9: NULL + ctypes
        86      LOAD_ATTR                       5: byref
        96      LOAD_FAST                       1: hProv
        98      PRECALL                         1
        102     CALL                            1
        112     LOAD_CONST                      0: None
        114     LOAD_CONST                      0: None
        116     LOAD_GLOBAL                     12: PROV_RSA_AES
        128     LOAD_GLOBAL                     14: CRYPT_VERIFYCONTEXT
        140     PRECALL                         5
        144     CALL                            5
        154     POP_JUMP_FORWARD_IF_TRUE        19 (to 194)
        156     LOAD_GLOBAL                     9: NULL + ctypes
        168     LOAD_ATTR                       8: WinError
        178     PRECALL                         0
        182     CALL                            0
        192     RAISE_VARARGS                   1
        194     LOAD_GLOBAL                     1: NULL + wintypes
        206     LOAD_ATTR                       1: HANDLE
        216     PRECALL                         0
        220     CALL                            0
        230     STORE_FAST                      2: hHash
        232     LOAD_GLOBAL                     4: advapi32
        244     LOAD_METHOD                     9: CryptCreateHash
        266     LOAD_FAST                       1: hProv
```



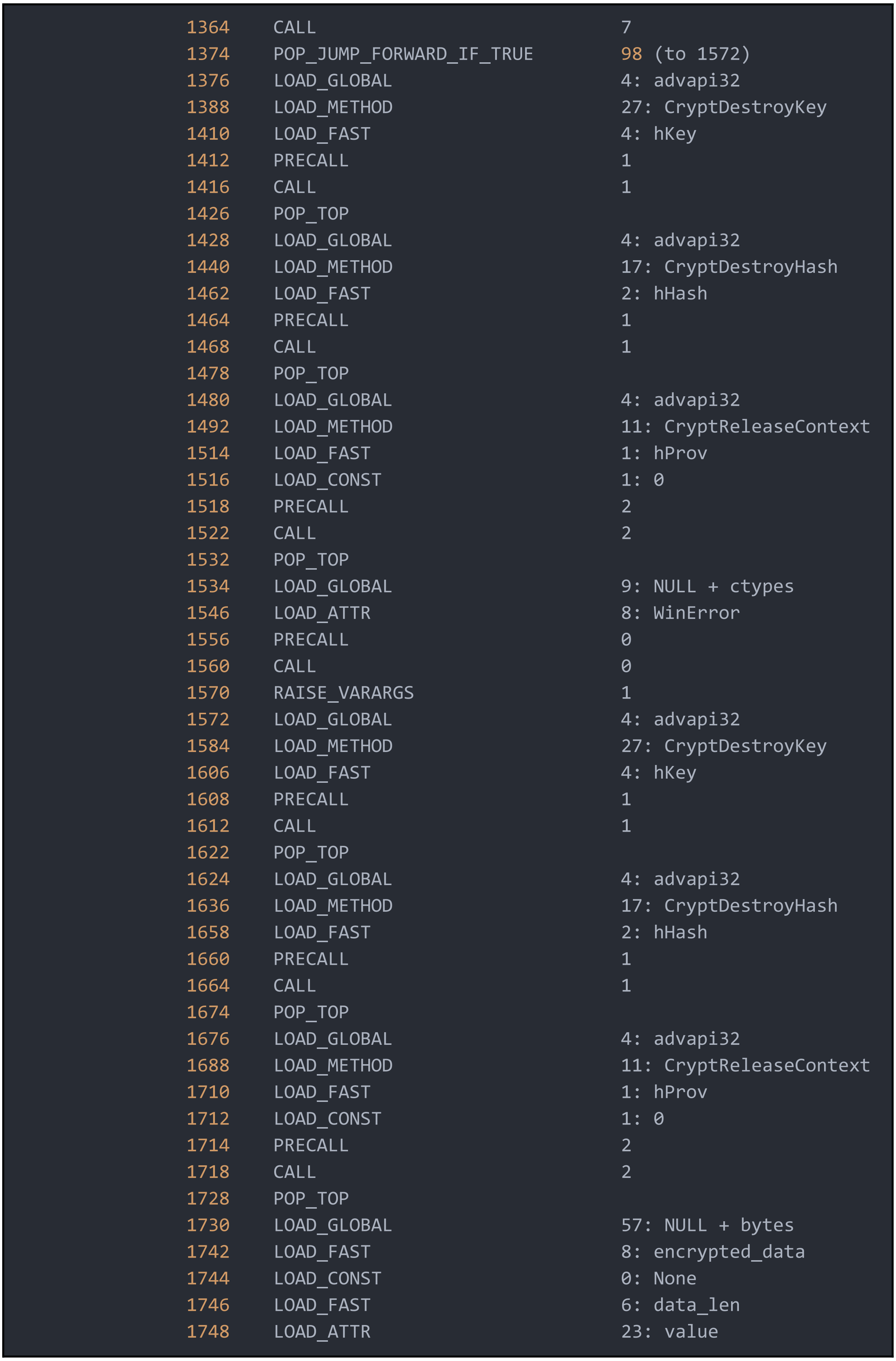
```
268     LOAD_GLOBAL                 20: CALG_SHA_256
280     LOAD_CONST                  1: 0
282     LOAD_CONST                  1: 0
284     LOAD_GLOBAL                 9: NULL + ctypes
296     LOAD_ATTR                   5: byref
306     LOAD_FAST                   2: hHash
308     PRECALL                     1
312     CALL                        1
322     PRECALL                     5
326     CALL                        5
336     POP_JUMP_FORWARD_IF_TRUE    46 (to 430)
338     LOAD_GLOBAL                 4: advapi32
350     LOAD_METHOD                 11: CryptReleaseContext
372     LOAD_FAST                   1: hProv
374     LOAD_CONST                  1: 0
376     PRECALL                     2
380     CALL                        2
390     POP_TOP
392     LOAD_GLOBAL                 9: NULL + ctypes
404     LOAD_ATTR                   8: WinError
414     PRECALL                     0
418     CALL                        0
428     RAISE_VARARGS               1
430     LOAD_GLOBAL                 24: this
442     LOAD_ATTR                   13: s
452     LOAD_METHOD                 14: encode
474     PRECALL                     0
478     CALL                        0
488     STORE_FAST                  3: _hash
490     LOAD_GLOBAL                 4: advapi32
502     LOAD_METHOD                 15: CryptHashData
524     LOAD_FAST                   2: hHash
526     LOAD_FAST                   3: _hash
528     LOAD_GLOBAL                 33: NULL + len
540     LOAD_FAST                   3: _hash
542     PRECALL                     1
546     CALL                        1
556     LOAD_CONST                  1: 0
558     PRECALL                     4
562     CALL                        4
572     POP_JUMP_FORWARD_IF_TRUE    72 (to 718)
574     LOAD_GLOBAL                 4: advapi32
586     LOAD_METHOD                 17: CryptDestroyHash
608     LOAD_FAST                   2: hHash
610     PRECALL                     1
614     CALL                        1
624     POP_TOP
626     LOAD_GLOBAL                 4: advapi32
638     LOAD_METHOD                 11: CryptReleaseContext
660     LOAD_FAST                   1: hProv
```



```
662     LOAD_CONST                  1: 0
664     PRECALL                     2
668     CALL                        2
678     POP_TOP
680     LOAD_GLOBAL                 9: NULL + ctypes
692     LOAD_ATTR                   8: WinError
702     PRECALL                     0
706     CALL                        0
716     RAISE_VARARGS               1
718     LOAD_GLOBAL                 1: NULL + wintypes
730     LOAD_ATTR                   1: HANDLE
740     PRECALL                     0
744     CALL                        0
754     STORE_FAST                  4: hKey
756     LOAD_GLOBAL                 4: advapi32
768     LOAD_METHOD                 18: CryptDeriveKey
790     LOAD_FAST                   1: hProv
792     LOAD_GLOBAL                 38: CALG_AES_256
804     LOAD_FAST                   2: hHash
806     LOAD_GLOBAL                 40: CRYPT_EXPORTABLE
818     LOAD_GLOBAL                 9: NULL + ctypes
830     LOAD_ATTR                   5: byref
840     LOAD_FAST                   4: hKey
842     PRECALL                     1
846     CALL                        1
856     PRECALL                     5
860     CALL                        5
870     POP_JUMP_FORWARD_IF_TRUE    72 (to 1016)
872     LOAD_GLOBAL                 4: advapi32
884     LOAD_METHOD                 17: CryptDestroyHash
906     LOAD_FAST                   2: hHash
908     PRECALL                     1
912     CALL                        1
922     POP_TOP
924     LOAD_GLOBAL                 4: advapi32
936     LOAD_METHOD                 11: CryptReleaseContext
958     LOAD_FAST                   1: hProv
960     LOAD_CONST                  1: 0
962     PRECALL                     2
966     CALL                        2
976     POP_TOP
978     LOAD_GLOBAL                 9: NULL + ctypes
990     LOAD_ATTR                   8: WinError
1000    PRECALL                     0
1004    CALL                        0
1014    RAISE_VARARGS               1
1016    LOAD_GLOBAL                 43: NULL + define__class
1028    LOAD_FAST                   0: bbData
1030    PRECALL                     1
1034    CALL                        1
```



```
1044    STORE_FAST        5: data_blob
1046    LOAD_GLOBAL       1: NULL + wintypes
1058    LOAD_ATTR         22: DWORD
1068    LOAD_GLOBAL       33: NULL + len
1080    LOAD_FAST         0: bbData
1082    PRECALL           1
1086    CALL              1
1096    PRECALL           1
1100    CALL              1
1110    STORE_FAST        6: data_len
1112    LOAD_GLOBAL       1: NULL + wintypes
1124    LOAD_ATTR         22: DWORD
1134    LOAD_FAST         6: data_len
1136    LOAD_ATTR         23: value
1146    LOAD_CONST        2: 16
1148    BINARY_OP         0 (+)
1152    PRECALL           1
1156    CALL              1
1166    STORE_FAST        7: buf_len
1168    LOAD_GLOBAL       9: NULL + ctypes
1180    LOAD_ATTR         24: c_ubyte
1190    LOAD_FAST         7: buf_len
1192    LOAD_ATTR         23: value
1202    BINARY_OP         5 (*)
1206    PRECALL           0
1210    CALL              0
1220    STORE_FAST        8: encrypted_data
1222    LOAD_GLOBAL       9: NULL + ctypes
1234    LOAD_ATTR         25: memmove
1244    LOAD_FAST         8: encrypted_data
1246    LOAD_FAST         0: bbData
1248    LOAD_FAST         6: data_len
1250    LOAD_ATTR         23: value
1260    PRECALL           3
1264    CALL              3
1274    POP_TOP
1276    LOAD_GLOBAL       4: advapi32
1288    LOAD_METHOD       26: CryptEncrypt
1310    LOAD_FAST         4: hKey
1312    LOAD_CONST        1: 0
1314    LOAD_CONST        3: True
1316    LOAD_CONST        1: 0
1318    LOAD_FAST         8: encrypted_data
1320    LOAD_GLOBAL       9: NULL + ctypes
1332    LOAD_ATTR         5: byref
1342    LOAD_FAST         6: data_len
1344    PRECALL           1
1348    CALL              1
1358    LOAD_FAST         7: buf_len
1360    PRECALL           7
```



```
1364    CALL                        7
1374    POP_JUMP_FORWARD_IF_TRUE    98 (to 1572)
1376    LOAD_GLOBAL                 4: advapi32
1388    LOAD_METHOD                 27: CryptDestroyKey
1410    LOAD_FAST                   4: hKey
1412    PRECALL                     1
1416    CALL                        1
1426    POP_TOP
1428    LOAD_GLOBAL                 4: advapi32
1440    LOAD_METHOD                 17: CryptDestroyHash
1462    LOAD_FAST                   2: hHash
1464    PRECALL                     1
1468    CALL                        1
1478    POP_TOP
1480    LOAD_GLOBAL                 4: advapi32
1492    LOAD_METHOD                 11: CryptReleaseContext
1514    LOAD_FAST                   1: hProv
1516    LOAD_CONST                  1: 0
1518    PRECALL                     2
1522    CALL                        2
1532    POP_TOP
1534    LOAD_GLOBAL                 9: NULL + ctypes
1546    LOAD_ATTR                   8: WinError
1556    PRECALL                     0
1560    CALL                        0
1570    RAISE_VARARGS               1
1572    LOAD_GLOBAL                 4: advapi32
1584    LOAD_METHOD                 27: CryptDestroyKey
1606    LOAD_FAST                   4: hKey
1608    PRECALL                     1
1612    CALL                        1
1622    POP_TOP
1624    LOAD_GLOBAL                 4: advapi32
1636    LOAD_METHOD                 17: CryptDestroyHash
1658    LOAD_FAST                   2: hHash
1660    PRECALL                     1
1664    CALL                        1
1674    POP_TOP
1676    LOAD_GLOBAL                 4: advapi32
1688    LOAD_METHOD                 11: CryptReleaseContext
1710    LOAD_FAST                   1: hProv
1712    LOAD_CONST                  1: 0
1714    PRECALL                     2
1718    CALL                        2
1728    POP_TOP
1730    LOAD_GLOBAL                 57: NULL + bytes
1742    LOAD_FAST                   8: encrypted_data
1744    LOAD_CONST                  0: None
1746    LOAD_FAST                   6: data_len
1748    LOAD_ATTR                   23: value
```

```
             1758     BUILD_SLICE                      2
             1760     BINARY_SUBSCR
             1770     PRECALL                          1
             1774     CALL                             1
             1784     RETURN_VALUE
```

Berdasarkan disassembly di atas, saya memperbaiki fungsi BRAINROTTED tersebut menjadi seperti berikut.

```
def BRAINROTTED(bbData):
    hProv = wintypes.HANDLE()
    if not advapi32.CryptAcquireContextW(ctypes.byref(hProv), None, None,
PROV_RSA_AES, CRYPT_VERIFYCONTEXT):
        raise ctypes.WinError()
    hHash = wintypes.HANDLE()
    if not advapi32.CryptCreateHash(hProv, CALG_SHA_256, 0, 0,
ctypes.byref(hHash)):
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    _hash = this.s.encode()
    if not advapi32.CryptHashData(hHash, _hash, len(_hash), 0):
        advapi32.CryptDestroyHash(hHash)
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    hKey = wintypes.HANDLE()
    if not advapi32.CryptDeriveKey(hProv, CALG_AES_256, hHash, CRYPT_EXPORTABLE,
ctypes.byref(hKey)):
        advapi32.CryptDestroyHash(hHash)
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    data_blob = define__class(bbData)
    data_len = wintypes.DWORD(len(bbData))
    buf_len = wintypes.DWORD(data_len.value + 16)
    encrypted_data = ctypes.c_ubyte * buf_len.value()
    ctypes.memmove(encrypted_data, bbData, data_len.value)
    if not advapi32.CryptEncrypt(hKey, 0, True, 0, encrypted_data,
ctypes.byref(data_len), buf_len):
        advapi32.CryptDestroyKey(hKey)
        advapi32.CryptDestroyHash(hHash)
        advapi32.CryptReleaseContext(hProv, 0)
        raise ctypes.WinError()
    advapi32.CryptDestroyKey(hKey)
    advapi32.CryptDestroyHash(hHash)
    advapi32.CryptReleaseContext(hProv, 0)
    return bytes(encrypted_data[:data_len.value])
```

Dari sini, kita bisa memahami bagaimana enkripsinya bekerja:

1. Melakukan hash terhadap this.s.encode(), this merupakan sebuah module python

2. Hasil hash tersebut dijadikan sebuah key
3. Lakukan enkripsi AES CBC 256 bit dengan key tersebut

Kita bisa melakukan dekripsi dengan menggunakan PyCryptoDome. Perlu diperhatikan bahwa malware ini tidak menentukan iv yang digunakan. Setelah mencari tahu di internet, ternyata default iv yang digunakan advapi32 adalah null bytes.

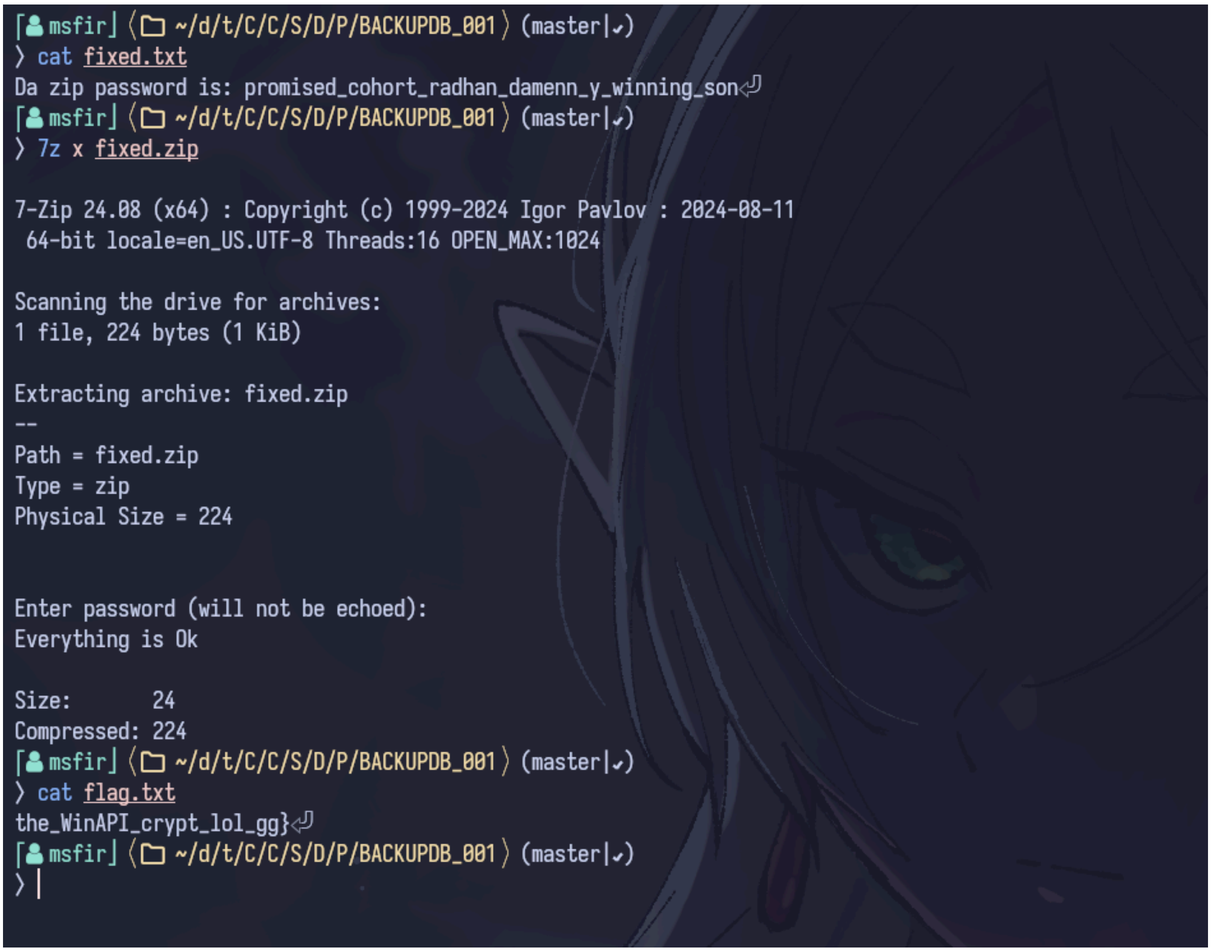
Solver:

```
#!/usr/bin/env python3

import this
from Crypto.Cipher import AES
from Crypto.Hash import SHA256
from Crypto.Util.Padding import unpad


def derive_key(password):
    hash_obj = SHA256.new(password)
    return hash_obj.digest()


def decrypt(ciphertext, key):
    aes = AES.new(key, AES.MODE_CBC, b"\x00" * 16)
    return unpad(aes.decrypt(ciphertext), 16)


key = derive_key(this.s.encode())

with open("./BRAINROT_9ea09c6c9d0aaf810d74baa90a498c2f.zip", "rb") as f, open(
    "./fixed.zip", "wb"
) as g:
    g.write(decrypt(f.read(), key))

with open("./BRAINROT_d2ac58ff3ac678de1924dbf167869360.txt", "rb") as f, open(
    "./fixed.txt", "wb"
) as g:
    g.write(decrypt(f.read(), key))

with open("./BRAINROT_fda35ec74ce961b56de5a189b96ce5c4.png", "rb") as f, open(
    "./fixed.png", "wb"
) as g:
    g.write(decrypt(f.read(), key))
```

Setelah run script tersebut, didapatkan sebuah image yang berisi flag part 1. Flag part 2 terdapat di dalam zip file yang passwordnya ada di sebuah txt file.

SELEKDA{pwn3d_

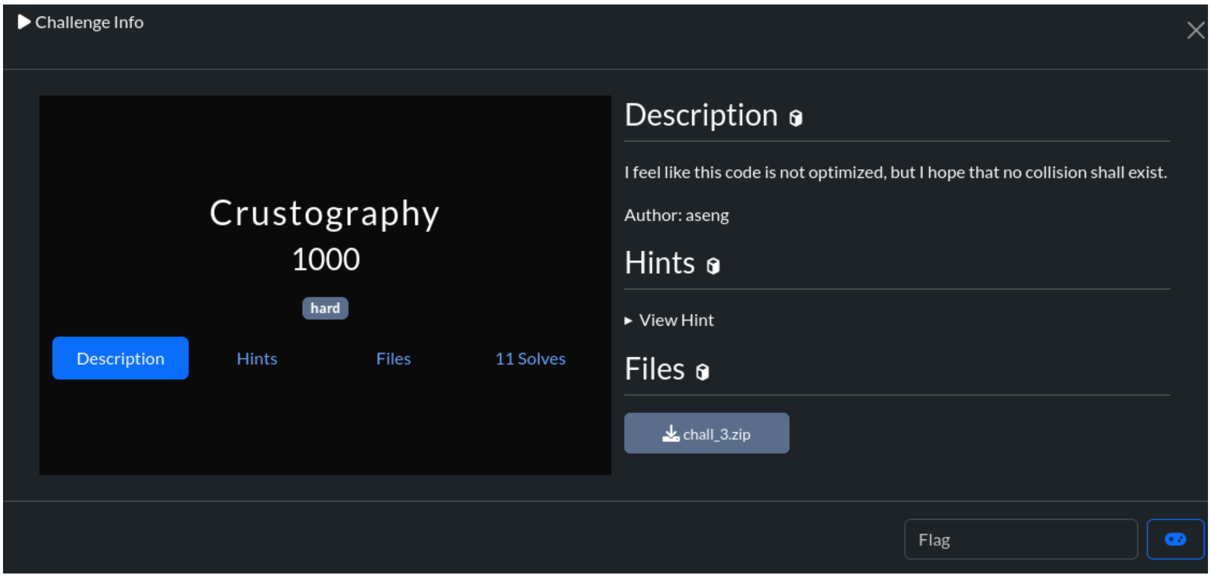
worldskills



```
[👤 msfir] ⟨🗀 ~/d/t/C/C/S/D/P/BACKUPDB_001⟩ (master|✓)
❯ cat fixed.txt
Da zip password is: promised_cohort_radhan_damenn_y_winning_son⏎
[👤 msfir] ⟨🗀 ~/d/t/C/C/S/D/P/BACKUPDB_001⟩ (master|✓)
❯ 7z x fixed.zip

7-Zip 24.08 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-08-11
 64-bit locale=en_US.UTF-8 Threads:16 OPEN_MAX:1024

Scanning the drive for archives:
1 file, 224 bytes (1 KiB)

Extracting archive: fixed.zip
--
Path = fixed.zip
Type = zip
Physical Size = 224


Enter password (will not be echoed):
Everything is Ok

Size:       24
Compressed: 224
[👤 msfir] ⟨🗀 ~/d/t/C/C/S/D/P/BACKUPDB_001⟩ (master|✓)
❯ cat flag.txt
the_WinAPI_crypt_lol_gg}⏎
[👤 msfir] ⟨🗀 ~/d/t/C/C/S/D/P/BACKUPDB_001⟩ (master|✓)
❯ |
```

Flag: SELEKDA{pwn3d_the_WinAPI_crypt_lol_gg}

[1000 pts] Crustography

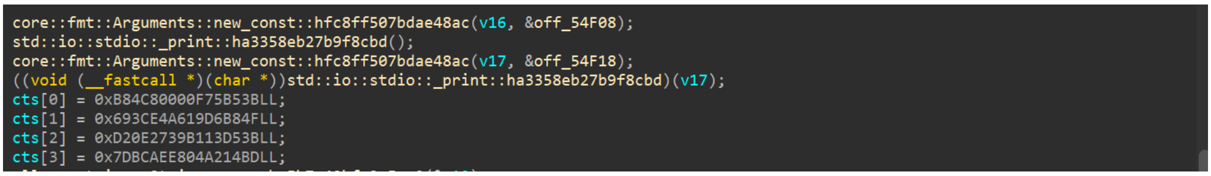


I feel like this code is not optimized, but I hope that no collision shall exist.

Author: aseng

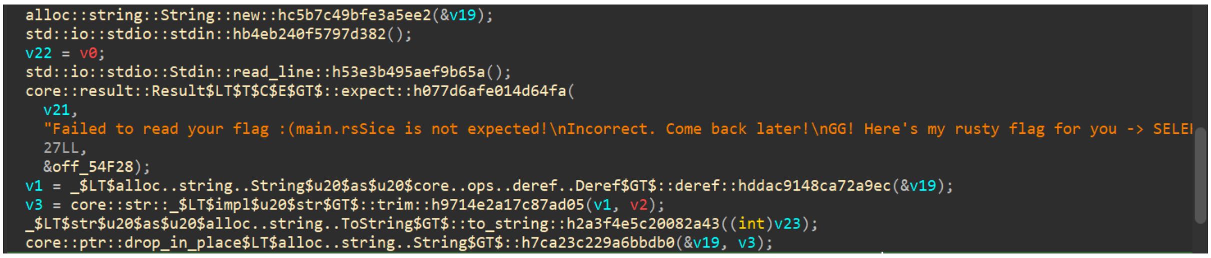
Diberikan sebuah linux binary yang dicompile dari bahasa Rust. Langsung saja kita buka dengan IDA.

Pertama-tama, program memprint sesuatu



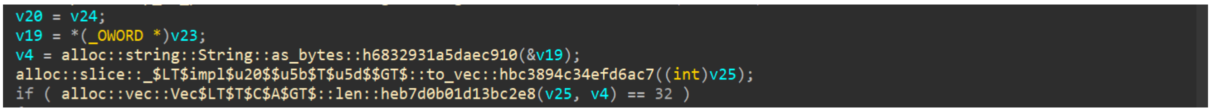
```
core::fmt::Arguments::new_const::hfc8ff507bdae48ac(v16, &off_54F08);
std::io::stdio::_print::ha3358eb27b9f8cbd();
core::fmt::Arguments::new_const::hfc8ff507bdae48ac(v17, &off_54F18);
((void (__fastcall *)(char *))std::io::stdio::_print::ha3358eb27b9f8cbd)(v17);
cts[0] = 0xB84C80000F75B53BLL;
cts[1] = 0x693CE4A619D6B84FLL;
cts[2] = 0xD20E2739B113D53BLL;
cts[3] = 0x7DBCAEE804A214BDLL;
```

Lalu program meminta input string



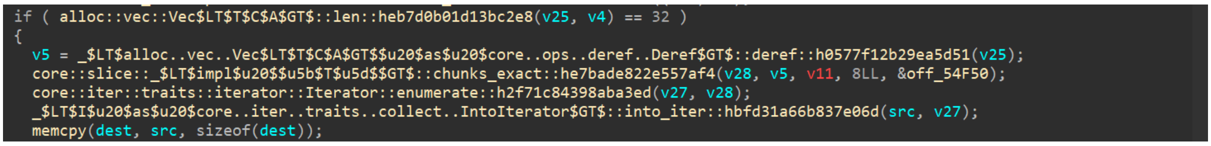
```
  alloc::string::String::new::hc5b7c49bfe3a5ee2(&v19);
  std::io::stdio::stdin::hb4eb240f5797d382();
  v22 = v0;
  std::io::stdio::Stdin::read_line::h53e3b495aef9b65a();
  core::result::Result$LT$T$C$E$GT$::expect::h077d6afe014d64fa(
    v21,
    "Failed to read your flag :(main.rsSice is not expected!\nIncorrect. Come back later!\nGG! Here's my rusty flag for you -> SELEI
    27LL,
    &off_54F28);
  v1 = _$LT$alloc..string..String$u20$as$u20$core..ops..deref..Deref$GT$::deref::hddac9148ca72a9ec(&v19);
  v3 = core::str::_$LT$impl$u20$str$GT$::trim::h9714e2a17c87ad05(v1, v2);
  _$LT$str$u20$as$u20$alloc..string..ToString$GT$::to_string::h2a3f4e5c20082a43((int)v23);
  core::ptr::drop_in_place$LT$alloc..string..String$GT$::h7ca23c229a6bbdb0(&v19, v3);
```

Lalu input string tersebut dijadikan vector of bytes dan dicek apakah panjangnya adalah 32



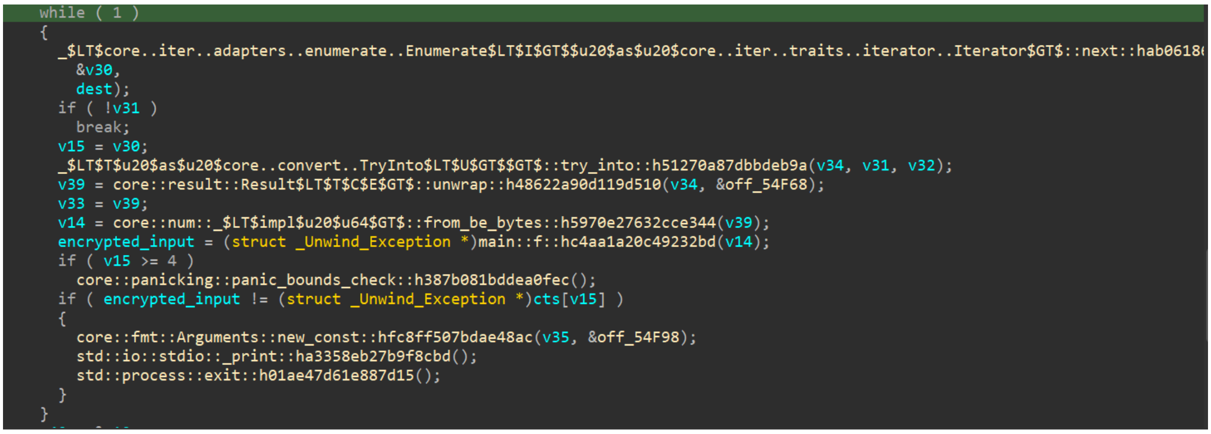
```
v20 = v24;
v19 = *(_OWORD *)v23;
v4 = alloc::string::String::as_bytes::h6832931a5daec910(&v19);
alloc::slice::_$LT$impl$u20$$u5b$T$u5d$$GT$::to_vec::hbc3894c34efd6ac7((int)v25);
if ( alloc::vec::Vec$LT$T$C$A$GT$::len::heb7d0b01d13bc2e8(v25, v4) == 32 )
```

Lalu vector tersebut dibagi menjadi beberapa vector dengan masing-masing berukuran 8, lalu di-enumerate dan dicollect menjadi vector lagi (contoh bentuknya menjadi [(0, [1, 2, 3, 4, 5, 6, 7, 8]), (1, [9, 10, 11, 12, 13, 14, 15, 16])]).



```
if ( alloc::vec::Vec$LT$T$C$A$GT$::len::heb7d0b01d13bc2e8(v25, v4) == 32 )
{
  v5 = _$LT$alloc..vec..Vec$LT$T$C$A$GT$$u20$as$u20$core..ops..deref..Deref$GT$::deref::h0577f12b29ea5d51(v25);
  core::slice::_$LT$impl$u20$$u5b$T$u5d$$GT$::chunks_exact::he7bade822e557af4(v28, v5, v11, 8LL, &off_54F50);
  core::iter::traits::iterator::Iterator::enumerate::h2f71c84398aba3ed(v27, v28);
  _$LT$I$u20$as$u20$core..iter..traits..collect..IntoIterator$GT$::into_iter::hbfd31a66b837e06d(src, v27);
  memcpy(dest, src, sizeof(dest));
```

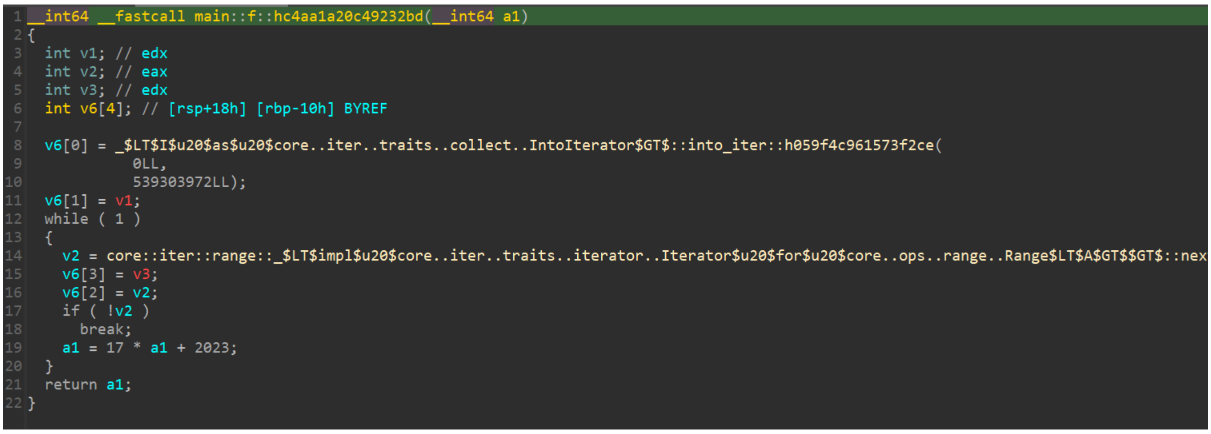
Blok berikut merupakan proses pengecekan



```
  while ( 1 )
  {
    _$LT$core..iter..adapters..enumerate..Enumerate$LT$I$GT$$u20$as$u20$core..iter..traits..iterator..Iterator$GT$::next::hab0618
      &v30,
      dest);
    if ( !v31 )
      break;
    v15 = v30;
    _$LT$T$u20$as$u20$core..convert..TryInto$LT$U$GT$$GT$::try_into::h51270a87dbbdeb9a(v34, v31, v32);
    v39 = core::result::Result$LT$T$C$E$GT$::unwrap::h48622a90d119d510(v34, &off_54F68);
    v33 = v39;
    v14 = core::num::_$LT$impl$u20$u64$GT$::from_be_bytes::h5970e27632cce344(v39);
    encrypted_input = (struct _Unwind_Exception *)main::f::hc4aa1a20c49232bd(v14);
    if ( v15 >= 4 )
      core::panicking::panic_bounds_check::h387b081bddea0fec();
    if ( encrypted_input != (struct _Unwind_Exception *)cts[v15] )
    {
      core::fmt::Arguments::new_const::hfc8ff507bdae48ac(v35, &off_54F98);
      std::io::stdio::_print::ha3358eb27b9f8cbd();
      std::process::exit::h01ae47d61e887d15();
    }
  }
```

Pertama, ambil elemen dari array enumeration tadi, lalu elemen pertamanya (integer) dijadikan index dan elemen keduanya (vector) diubah menjadi integer dengan fungsi from_be_bytes. Integer tersebut lalu dijadikan argumen dari fungsi f, lalu hasilnya dibandingkan dengan elemen cts pada index yang bersesuaian.

Berikut pseudocode dari fungsi f.



```
__int64 __fastcall main::f::hc4aa1a20c49232bd(__int64 a1)
{
  int v1; // edx
  int v2; // eax
  int v3; // edx
  int v6[4]; // [rsp+18h] [rbp-10h] BYREF

  v6[0] = _$LT$I$u20$as$u20$core..iter..traits..collect..IntoIterator$GT$::into_iter::h059f4c961573f2ce(
            0LL,
            539303972LL);
  v6[1] = v1;
  while ( 1 )
  {
    v2 = core::iter::range::_$LT$impl$u20$core..iter..traits..iterator..Iterator$u20$for$u20$core..ops..range..Range$LT$A$GT$$GT$::nex
    v6[3] = v3;
    v6[2] = v2;
    if ( !v2 )
      break;
    a1 = 17 * a1 + 2023;
  }
  return a1;
}
```

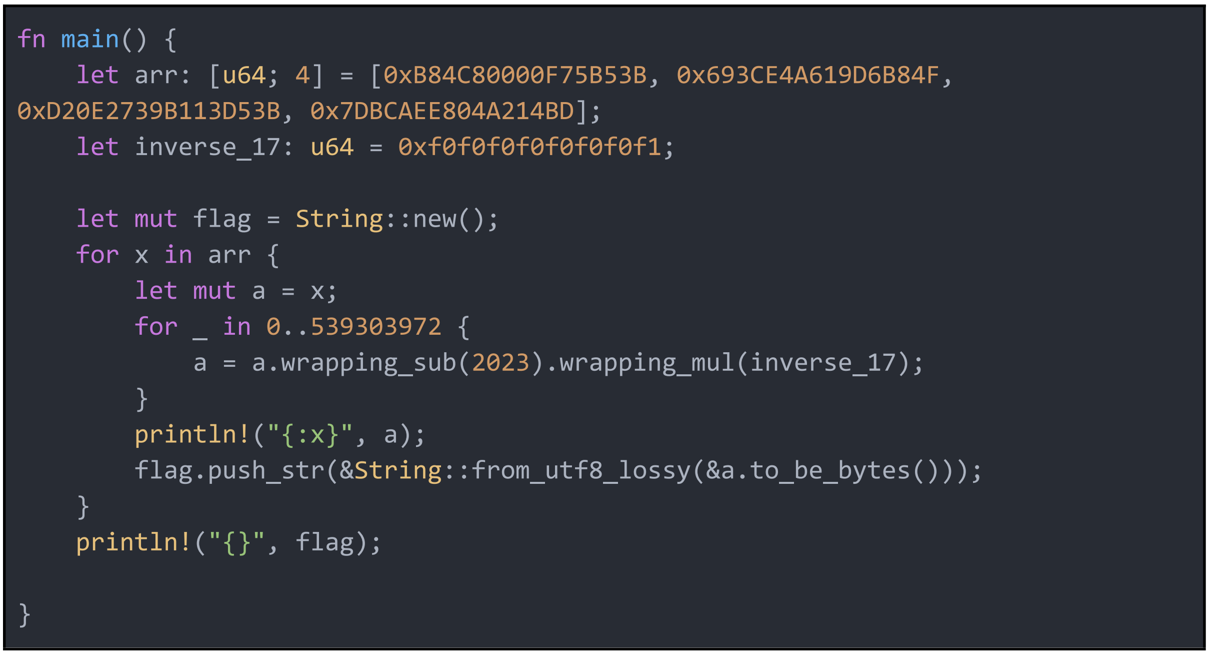
Fungsi tersebut melakukan kalkulasi Linear Congruential Generator

$$X_{n+1} = (aX_n + c) \mod m$$

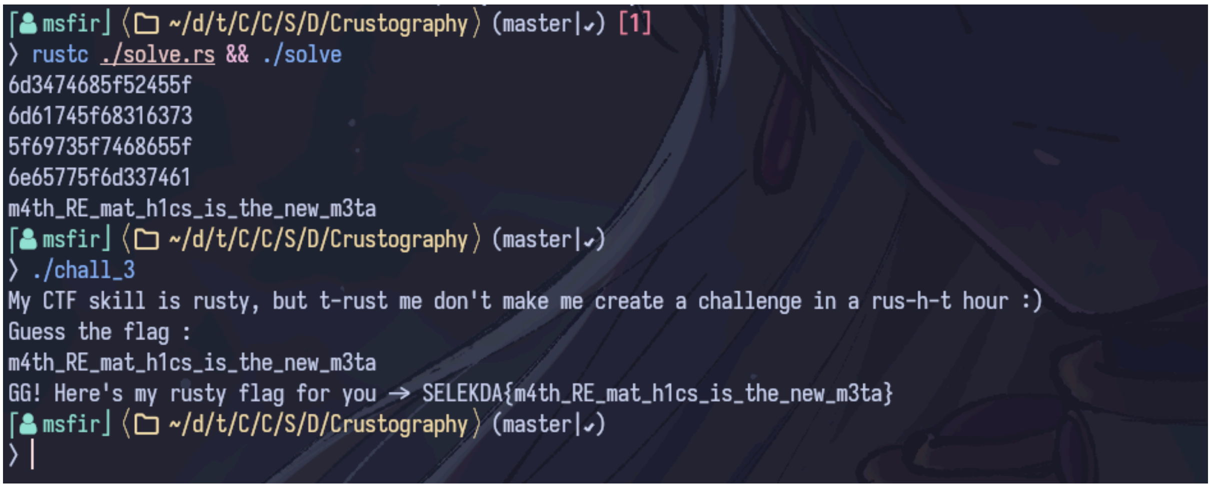
dengan a = 17, c = 2023, dan m = 2^64, sebanyak 539303972 kali.

Dari uraian di atas, kita dapat mendapatkan flagnya dengan cara melakukan invert LCG dari elemen-elemen cts.

Solver:



```rust
fn main() {
    let arr: [u64; 4] = [0xB84C80000F75B53B, 0x693CE4A619D6B84F,
0xD20E2739B113D53B, 0x7DBCAEE804A214BD];
    let inverse_17: u64 = 0xf0f0f0f0f0f0f0f1;

    let mut flag = String::new();
    for x in arr {
        let mut a = x;
        for _ in 0..539303972 {
            a = a.wrapping_sub(2023).wrapping_mul(inverse_17);
        }
        println!("{:x}", a);
        flag.push_str(&String::from_utf8_lossy(&a.to_be_bytes()));
    }
    println!("{}", flag);

}
```

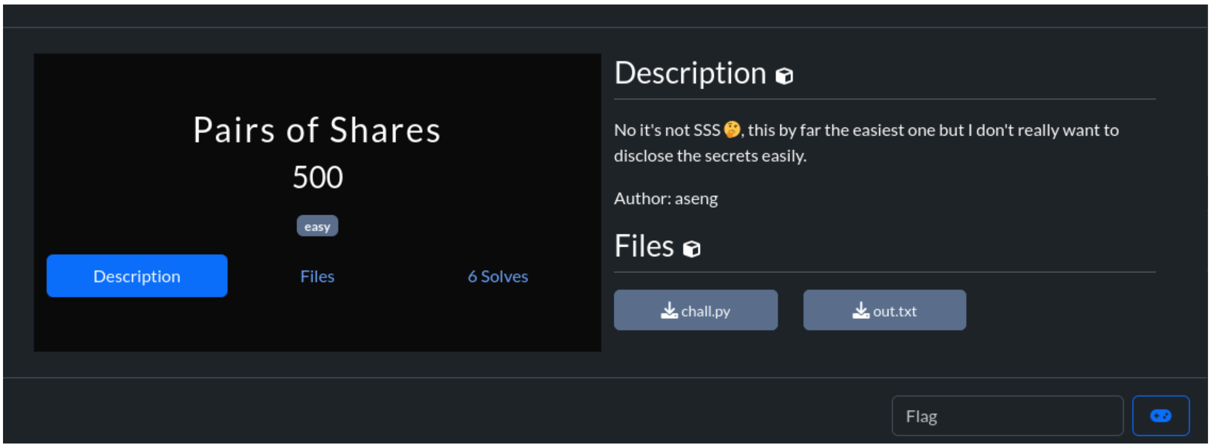


```
[msfir] ⟨🗀 ~/d/t/C/C/S/D/Crustography⟩ (master|✓) [1]
❯ rustc ./solve.rs && ./solve
6d3474685f52455f
6d61745f68316373
5f69735f7468655f
6e65775f6d337461
m4th_RE_mat_h1cs_is_the_new_m3ta
[msfir] ⟨🗀 ~/d/t/C/C/S/D/Crustography⟩ (master|✓)
❯ ./chall_3
My CTF skill is rusty, but t-rust me don't make me create a challenge in a rus-h-t hour :)
Guess the flag :
m4th_RE_mat_h1cs_is_the_new_m3ta
GG! Here's my rusty flag for you → SELEKDA{m4th_RE_mat_h1cs_is_the_new_m3ta}
[msfir] ⟨🗀 ~/d/t/C/C/S/D/Crustography⟩ (master|✓)
❯ |
```

Flag: SELEKDA{m4th_RE_mat_h1cs_is_the_new_m3ta}

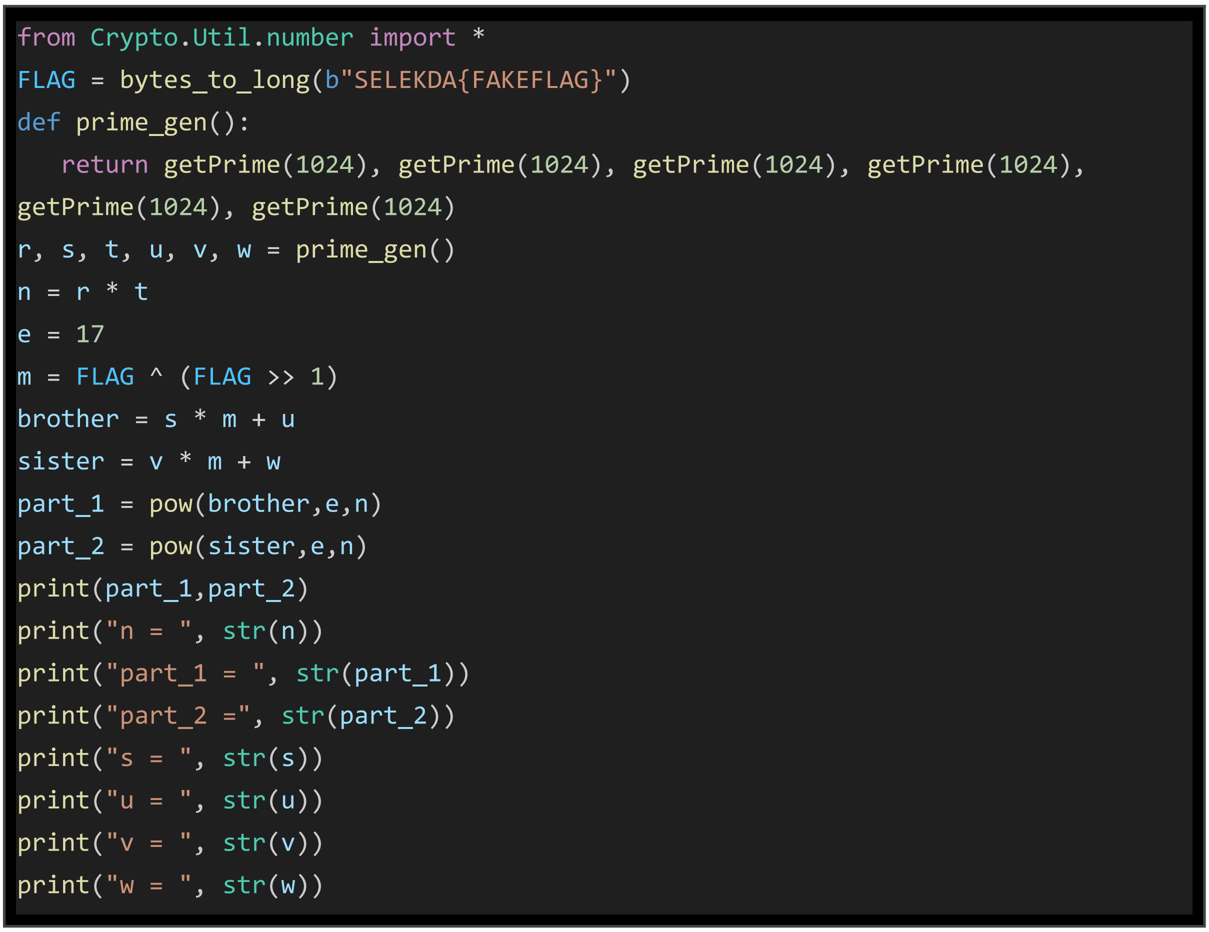
Cryptography

[500 pts] Pairs of Shares



No it's not SSS 🤔, this by far the easiest one but I don't really want to disclose the secrets easily.

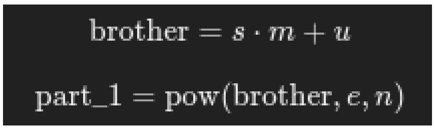
Author: aseng



```
from Crypto.Util.number import *
FLAG = bytes_to_long(b"SELEKDA{FAKEFLAG}")
def prime_gen():
   return getPrime(1024), getPrime(1024), getPrime(1024), getPrime(1024),
getPrime(1024), getPrime(1024)
r, s, t, u, v, w = prime_gen()
n = r * t
e = 17
m = FLAG ^ (FLAG >> 1)
brother = s * m + u
sister = v * m + w
part_1 = pow(brother,e,n)
part_2 = pow(sister,e,n)
print(part_1,part_2)
print("n = ", str(n))
print("part_1 = ", str(part_1))
print("part_2 =", str(part_2))
print("s = ", str(s))
print("u = ", str(u))
print("v = ", str(v))
print("w = ", str(w))
```

Flag didefinisikan menjadi long lalu akan di rubah menjadi graycode menggunakan xor dan shifting (**m**)

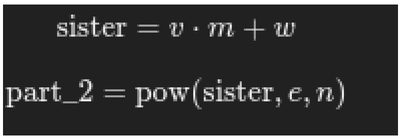
Brother cipher



$$\text{brother} = s \cdot m + u$$

$$\text{part\_1} = \text{pow}(\text{brother}, e, n)$$

sister cipher



$$\text{sister} = v \cdot m + w$$

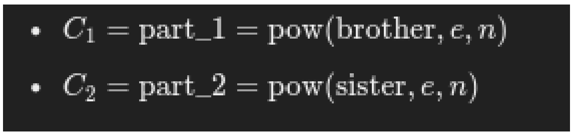
$$\text{part\_2} = \text{pow}(\text{sister}, e, n)$$

kedua ciphertext tersebut menggunakan (**m**) yang sama meskipun dengan kombinasi linear yang berbeda

untuk mendapatkan pesan original kita dapat memanfaatkan persamaan yang diturunkan dari ciphertext

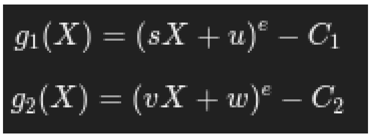
kita dapat menggunakan Franklin-Reiter Related Message Attack (FRRMA)

Serangan Franklin-Reiter memanfaatkan bahwa jika dua ciphertext berbeda diturunkan dari pesan plaintext yang sama melalui bentuk polinomial yang sedikit berbeda, hubungan antara bentuk-bentuk ini dapat dieksploitasi untuk memulihkan pesan aslinya.



- $C_1 = \text{part\_1} = \text{pow}(\text{brother}, e, n)$
- $C_2 = \text{part\_2} = \text{pow}(\text{sister}, e, n)$

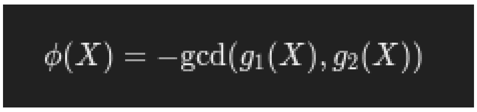
kurang lebih berikut adalah persamaannya



$$g_1(X) = (sX + u)^e - C_1$$

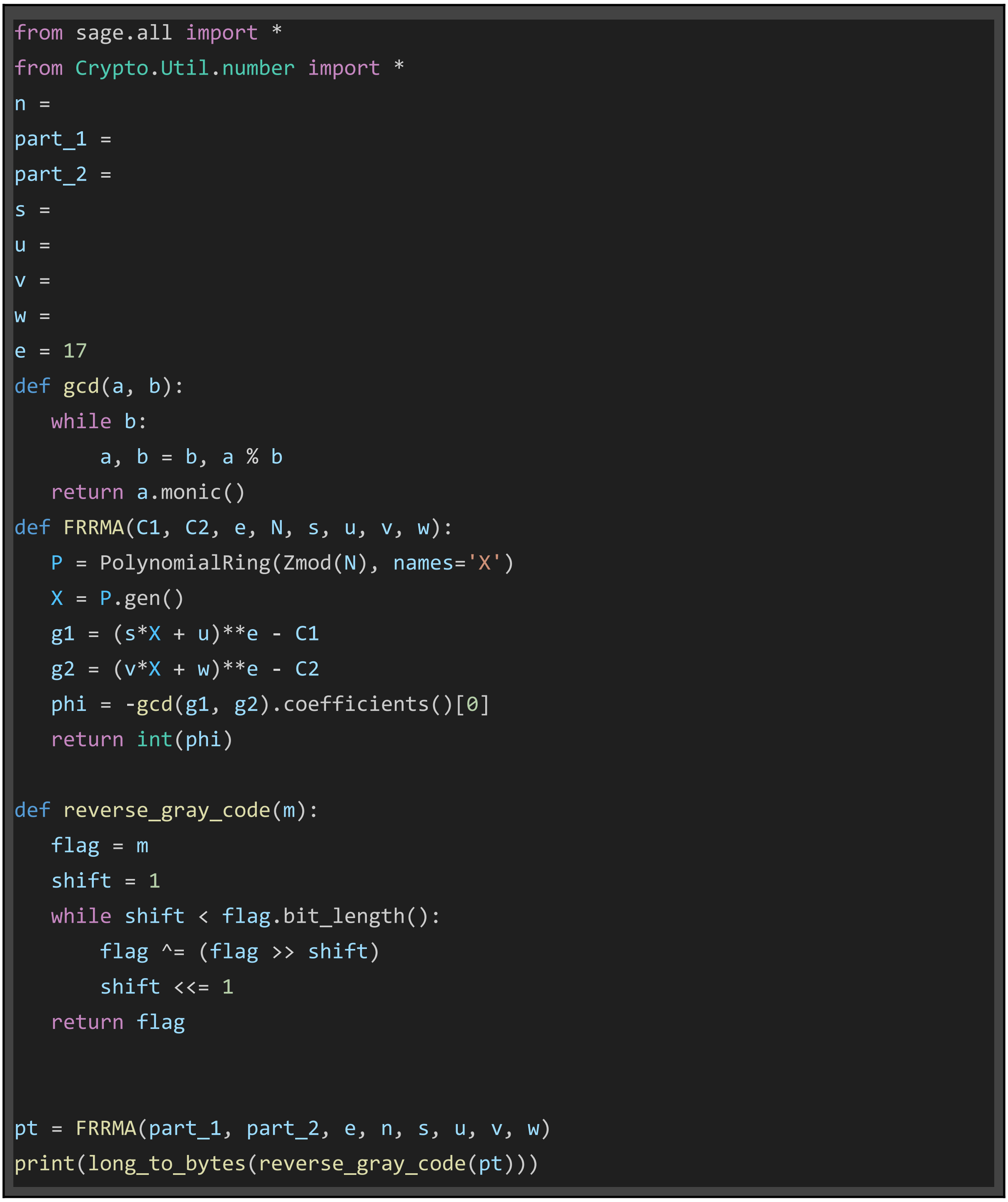
$$g_2(X) = (vX + w)^e - C_2$$

menggunakan gcd untuk mendapatkan relationship

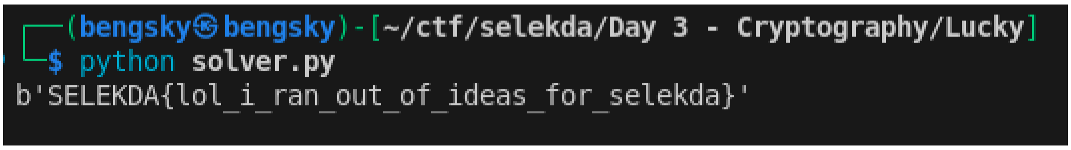


$$\phi(X) = -\text{gcd}(g_1(X), g_2(X))$$

dan terakhir hanya perlu melakulan reverse gray code



```python
from sage.all import *
from Crypto.Util.number import *
n =
part_1 =
part_2 =
s =
u =
v =
w =
e = 17
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a.monic()
def FRRMA(C1, C2, e, N, s, u, v, w):
    P = PolynomialRing(Zmod(N), names='X')
    X = P.gen()
    g1 = (s*X + u)**e - C1
    g2 = (v*X + w)**e - C2
    phi = -gcd(g1, g2).coefficients()[0]
    return int(phi)

def reverse_gray_code(m):
    flag = m
    shift = 1
    while shift < flag.bit_length():
        flag ^= (flag >> shift)
        shift <<= 1
    return flag


pt = FRRMA(part_1, part_2, e, n, s, u, v, w)
print(long_to_bytes(reverse_gray_code(pt)))
```



```
┌──(bengsky㉿bengsky)-[~/ctf/selekda/Day 3 - Cryptography/Lucky]
└─$ python solver.py
b'SELEKDA{lol_i_ran_out_of_ideas_for_selekda}'
```

Flag: SELEKDA{lol_i_ran_out_of_ideas_for_selekda}