Secure Programming 2020

HW0C write-up

ChristmasGift

這一題會要你輸入,reverse後可以直接看到要輸的東西,直接輸入會output一個檔案.

```
strcpy(
    s2,

"JZC33MJPDC48UXXJ94BBQOR0JJR4AO0W02PHZ4VZRJAEXL3OUI02FQ4GSQIDGBFT70VESKNAAUEJW4RR
9EQOCJ9PKT7W9FBMJDVK6X9MT7K1HY30MSA4"

"H3Y9FTV0O7Z6FQ5I1J8R6KSCMWKFSDGCMWARIJTLPLRO8KUYQW2F46ZV6YWIVFNCZDQRCTAM5JVGQMEU
2LFPS5DUDOY4130XB50V91PWHCIO0AD1RHTR"
    "673DPX36TA2UWA48FD34Y2W6");
  __isoc99_scanf("%256s", &s1, v7);
  if ( !strcmp(&s1, s2) )
  {
    for ( i = 0; i <= (signed int)&unk_3347DA; ++i )
    {
      v3 = byte_201020[i];
      byte_201020[i] = s2[i % strlen(s2)] ^ v3;
    }
    puts("Ok, that sounds good");
    write(1, byte_201020, (size_t)&unk_3347DB);
```

用file去看會發現是gzip檔,解壓縮又是一個gift,再reverse一次會發現要求輸入的字串又不一樣了.
寫個腳本重複這個pattern,先解壓縮再從elf裡找輸入字串,再輸入,重複.

```
#!/usr/bin/env python3
from pwn import *
import gzip

i = 0
while(1):
    file = open('./gift', 'wb+')
    f = gzip.open('./gift.gz', 'rb')

    content = f.read()
    file.write(content)

    f.close()
    file.close()

    proc = process('./gift')

    e = ELF('./gift')

    s = e.read(0xa10, 256)
    #st = s.decode("utf-8")
    print("i: " + str(i))
    print(s)
    proc.sendline(s)
    proc.recvuntil('Ok, that sounds good\n')
    gift = proc.recvall()


    f = open('./gift.gz','wb+')
    f.write(gift)
    f.close()

    i += 1
```

到1000次左右,檔案格式會壞掉,無法解壓縮,這時再reverse一次,會發現字串變成:

```
v12 = __readfsqword(0x28u);
  strcpy(s2,
"@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@terrynini@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@");
  v7 = 0LL;
  memset(&v8, 0, 0xA0uLL);
  v9 = 0;
  __isoc99_scanf("%256s", &s1, v10);
  if ( !strcmp(&s1, s2) )
  {
    for ( i = 0; i <= 53; ++i )
    {
      v3 = byte_201020[i];
      byte_201020[i] = s2[i % strlen(s2)] ^ v3;
    }
    puts("Ok, that sounds good");
    write(1, byte_201020, 0x36uLL);
  }
```

照著輸入,得到flag:

```
FLAG{what_a_boaring_challnge_but_you_did_it_yeah_yeah}
```

JustOnLinux

這題是base64,一開始會根據輸入的4/3長度malloc.

```
if ( argc > 1 )
  {
    v25 = argv[1];
    v21 = (unsigned __int64)sub_4004C0((__int64)argv[1], (__int64)argv,
(__int64)envp);
    v22 = 4 * ((v21 + 2) / 3);
    v26 = (_QWORD *)malloc(4 * ((v21 + 2) / 3) + 1);
    ...
```

之後會作類似查表的動作.

```
 v15 = v19 + 1;
        *((_BYTE *)v26 + v14) = aVwxyzabcdefghi[(v13 >> 18) & 0x3F];
        v16 = v15++;
        *((_BYTE *)v26 + v16) = aVwxyzabcdefghi[(v13 >> 12) & 0x3F];
        *((_BYTE *)v26 + v15) = aVwxyzabcdefghi[(v13 >> 6) & 0x3F];
        v17 = v15 + 1;
        v19 = v15 + 2;
        *((_BYTE *)v26 + v17) = aVwxyzabcdefghi[v13 & 0x3F];
```

base64是8x3=24bit切成4x6=24bit,然後查表,合理懷疑是作base64.

再看一下表:

```
vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&',27h,'()*+,-./:;<=>?@[\]^_`
```

剛好64個,應該就是base64,只是他的表跟正常的不一樣.如果把他的表再對到正常的表應該就可以得到flag的base64.

把flag encode的值查表然後轉成對應的base64 table的值,得到:

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

table_old = "vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&'()*+,-./:;<=>?@[\]^_`"
table_new = "ABCDEFGHIGKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

f = open("./flag", "r")

content = f.read()

flag = ''
for byte in content:
    idx = table_old.find(byte)
    flag += table_new[idx]
print(flag)
```

```
RkxBR3s3aDFzLWk1LWFjN3VhMTF5LWEtYjRzMzY0ZW5jMGQzLWFsZzByMXRobX0
```

再base64 decode,得到flag:

```
FLAG{7h1s-i5-ac7ua11y-a-b4s364enc0d3-alg0r1thm}
```