HW-0x0c Writeup

School/Grade: 交大資科工所 碩一

Student ID: 309551004 (王冠中)

ID: aesophor

ChristmasGift (150 pts)

觀察一下題目給的檔案 `gift.gz` ，可以發現是 gzip compressed data

```
$ file gift.gz
gift.gz: gzip compressed data, was "gift",
last modified: Thu Dec 24 22:50:38 2020, max speed,
from Unix, original size modulo 2^32 3366872
```

用 `binwalk -e gift.gz` 可以抽出一個 ELF

```
$ binwalk -e gift.gz

$ ls -la
total 3.3M
drwxr-xr-x 2 aesophor aesophor 4.0K Dec 28 10:14 _gift.gz.extracted
-rw-r--r-- 1 aesophor aesophor 3.3M Dec 25 22:15 gift.gz

$ ls -la _gift.gz.extracted
total 6.5M
drwxr-xr-x 2 aesophor aesophor 4.0K Dec 28 10:14 .
drwxr-xr-x 3 aesophor aesophor 4.0K Dec 28 10:12 ..
-rwxr--r-- 1 aesophor aesophor 3.3M Dec 28 10:12 gift
-rw-r--r-- 1 aesophor aesophor 3.3M Dec 28 10:12 gift.gz

$ file _gift.gz.extracted/gift
_gift.gz.extracted/gift: ELF 64-bit LSB pie executable,
x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2,
for GNU/Linux 3.2.0, BuildID[sha1]=
8101153b34bd64c6d20b5f47a7872b1fc8ee437c, stripped
```

接下來開啟 IDA，對這個 ELF 進行 static analysis，可以發現一個 256 bytes 的 magic string：

```
undefined8 main(void)
{
    uint8_t uVar1;
    int32_t iVar2;
    uint64_t uVar3;
    int64_t iVar4;
    char **ppcVar5;
    char ***pppcVar6;
    int64_t in_FS_OFFSET;
    int32_t var_234h;
    char **format;
    char *s1;
    int64_t canary;

    canary = *(int64_t *)(in_FS_OFFSET + 0x28);
    iVar4 = 0x20;
    ppcVar5 = (char **)
              "JZC33MJPDC48UXXJ94BBQOR0JJR4AO0W02PHZ4VZRJAEXL3OUI02FQ4GS
    ;
    pppcVar6 = &format;
    while (iVar4 != 0) {
        iVar4 = iVar4 + -1;
        *pppcVar6 = (char **)*ppcVar5;
        ppcVar5 = ppcVar5 + 1;
        pppcVar6 = pppcVar6 + 1;
    }
    *(undefined *)pppcVar6 = *(undefined *)ppcVar5;
    __isoc99_scanf(0x9e8, &s1, (int64_t)pppcVar6 + 1);
    iVar2 = strcmp(&s1, &format, &format);
    if (iVar2 == 0) {
        var_234h = 0;
        while (var_234h < 0x3347db) {
            uVar1 = *(uint8_t *)((int64_t)var_234h + 0x201020);
            uVar3 = strlen(&format);
            *(uint8_t *)((int64_t)var_234h + 0x201020) =
                 uVar1 ^ *(uint8_t *)((int64_t)&format + (uint64_t)(int6
            var_234h = var_234h + 1;
        }
        puts(0x9ee);
        write(1, 0x201020, 0x3347db);
```

然後直覺上就是把 `gift` 跑起來，並把剛剛這個 256-byte long 的 magic string 餵給程式。接下來程式會印一大堆東西到 stdout，但如果我們把程式的輸出全部 redirect 到一個檔案，可以發現又是一個 gzip compressed data。

遇到這樣的題目，就想到之前 bamboofox 2018 New Year CTF 的一個 Misc 題：具體細節有點忘記，但大概就是不斷 unzip 最後拿到 flag。所以這題大概也是不斷的反覆剛剛的動作。整理一下：

1. decompress gzip file and obtain ELF
2. acquire the magic string from the ELF (it resides at a fixed offset within the ELF)
3. run the program and use the magic string as input
4. the program will print a new gzip to stdout
5. jmp step1

Exploit

```
#!/usr/bin/env python3
# -*- encoding: utf-8 -*-

import os
import gzip
from pwn import *
context.update(arch = 'amd64', os = 'linux', log_level = 'info')

def main():
    i = 0

    while True:
        with gzip.open('./gift.gz', 'rb') as f:
            file_content = f.read()

        with open('./gift', 'wb') as f:
            f.write(file_content)

        os.system('chmod u+x ./gift')

        elf = ELF('./gift')
        passwd = elf.read(0xA10, 256)
        log.info('[{}] {}'.format(i, passwd))

        proc = elf.process()
        proc.sendline(passwd)
        proc.recvuntil('Ok, that sounds good\n')
        output = proc.recvall()

        with open('./gift.gz', 'wb') as f:
            f.write(output)

        i += 1


if __name__ == '__main__':
    main()
```

Flag: `FLAG{what_a_boaring_challnge_but_you_did_it_yeah_yeah}`

JustOnLinux (150 pts)

IDA 打開之後找不到 `main()` ，但是 cutter 可以，所以就用 cutter 看 ghidra 反編譯出來的結果吧。

首先在 `main()` 可以看到以下兩個酷酷的東西：

1. 一個很像某種 character table 的字串
2. malloc() 的大小是 (uVar2 + 2) 的三分之四

```
uVar7 = iVar6 + iVar4 * 0x10000 + iVar5 * 0x100;
*(char *)(var_34h._4_4_ + arg2) =
     "vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~o"[uVar7 >> 0x12 & 0x3f];
*(char *)((var_34h._4_4_ + 1) + arg2) =
     "vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~o"[uVar7 >> 0xc & 0x3f];
iVar4 = var_34h._4_4_ + 3;
*(char *)((var_34h._4_4_ + 2) + arg2) =
     "vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~o"[uVar7 >> 6 & 0x3f];
var_34h._4_4_ = var_34h._4_4_ + 4;
*(char *)(iVar4 + arg2) =
     "vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~o"[uVar7 & 0x3f];
```

```
    pcVar1 = argv[1];
    uVar2 = fcn.004004c0(pcVar1);
    iVar3 = ((int32_t)(uVar2 + 2) / 3) * 4;
    arg2 = fcn.0041f810(iVar3 + 1);
    if (arg2 == 0) {
        fcn.00410590((int64_t)"malloc failed");
    } else {
```

上面兩點感覺很像是 base64，然後看看 character table 感覺和標準 base64 的 character table 不太一樣。

印出 flag 可以發現一串亂碼，但綜合上述觀察到的點，感覺 flag file 的內容是一個 `FLAG{***}` 的字串經過特殊版的 base64 處理後所得到的結果。

```
/home/aesophor/CTF/nctu-secure-programming-2020-fall/hwc-reverse/2-JustOnLinux [git::master *] [aesophor@sqlab] [19:51]
➤ cat flag
M&=wM].]VyA?GR&[GRA%I]Q#HOA_GRz/T%M?H?T@UR_%HBL?GRA.U?w>HSM*WS@ %
```

接下來看看 wiki 列出來的標準 base64 table：

Base64 table [edit]

The Base64 index table:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

接下來我們就去把 flag 的每個 char 掃一遍，然後找到他在題目的 character table 所在的 index。找到 index 之後再從標準 base64 table 抓出對應的 char，最後再 b64decode 就可以得出 flag 了。

Exploit:

```
#!/usr/bin/env python3
# -*- encoding: utf-8 -*-

import base64

def main():
    with open('./flag', 'r') as f:
        encoded = f.read()
```

```
    print(encoded)

    table_old="vwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~o"
    table_new="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

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

    print(base64.b64decode(flag))


if __name__ == '__main__':
    main()
```

flag: FLAG{7h1s-i5-ac7ua11y-a-b4s364enc0d3-alg0r1thm}