EncryptEdge Labs

Cybersecurity Analyst Internship

Task Report

atalmamun@gmail.com

Task No: 18

Credit: Offensive Security

Table of Contents

1.0 EncryptEdge Labs Internship Task Report

1.1 Introduction

Mobile devices have become an integral part of modern life, offering convenience, connectivity, and productivity on the go. However, as mobile usage continues to grow, so do the associated security risks. Mobile platforms are increasingly targeted by cybercriminals seeking to exploit vulnerabilities in operating systems, applications, and wireless communications. This report explores the fundamentals of mobile security, highlighting the unique challenges faced in this domain and the importance of implementing robust security practices. Through a combination of research, hands-on lab activities, and simulated attack scenarios, this task aims to build foundational knowledge and practical skills in securing mobile environments.

1.2 Objective

The primary objectives of this task are to:

- Understand the specific challenges and threat landscape related to mobile security.
- Learn the fundamental principles and best practices for securing mobile devices and applications.
- Gain hands-on experience in setting up a mobile security testing environment.
- Identify and mitigate common mobile vulnerabilities using industry-standard tools.
- Analyze real-world mobile breaches and simulate mobile attack scenarios to practice defensive strategies.
- Complete practical labs focusing on Android hacking techniques and mobile malware analysis.

1.3 Requirements

To complete this task, the following tools and resources were required:

- Mobile devices (physical or emulators) for testing purposes.

- Basic mobile security testing tools such as:
    - **MobSF (Mobile Security Framework)**
    - **Drozer**
    - **OWASP Mobile Security Testing Guide**
- Network analysis tools such as:
    - **Wireshark** or similar packet capturing utilities.
- (Optional) Mobile Device Management (MDM) solutions for enterprise-level security insights.
- TryHackMe labs:
    - **Android Hacking 101**
    - **Mobile Malware Analysis**

- A working virtual lab setup to simulate and analyze mobile threats in a controlled environment.

2.0 Mobile Security Basics

2.1 Common Threats and Attack Vectors

Mobile devices are vulnerable to a wide range of security threats due to their portability, constant connectivity, and the diversity of installed applications. Some of the most common threats and attack vectors include:

- **Malware**: Mobile malware such as trojans, spyware, and ransomware can compromise sensitive data, track user activity, or even take control of the device.
- **Phishing Attacks**: Attackers use deceptive messages, often via SMS or email, to trick users into disclosing confidential information or installing malicious applications.
- **Insecure Communications**: Unencrypted data transmission over public Wi-Fi or mobile networks can be intercepted through man-in-the-middle (MitM) attacks.
- **Application Vulnerabilities**: Poorly developed apps may contain security flaws such as insecure data storage, weak authentication, and improper permission handling.
- **Unauthorized Access**: Lack of proper device configuration (e.g., no screen lock or biometric authentication) can allow attackers to gain physical or remote access.
- **Outdated Software**: Devices running outdated operating systems or apps may lack security patches, leaving them exposed to known exploits.

2.2 Best Practices for Securing Mobile Devices

To mitigate mobile security risks, the following best practices should be implemented:

- **Use Strong Authentication**: Enforce the use of PINs, passwords, and biometric authentication to prevent unauthorized access.
- **Keep Software Updated**: Regularly update the mobile OS and applications to patch security vulnerabilities.
- **Limit App Permissions**: Grant only necessary permissions to apps and avoid installing applications from untrusted sources.

- **Use Encryption**: Enable full-disk encryption and ensure data-in-transit is protected using protocols such as SSL/TLS.

- **Install Security Tools**: Use mobile antivirus solutions, firewalls, and security scanners to detect and block threats.

- **Enable Remote Wipe and Tracking**: Ensure that lost or stolen devices can be tracked and wiped remotely using built-in features or MDM solutions.

- **Avoid Public Wi-Fi**: Use VPNs or avoid accessing sensitive information over unsecured public networks.

This section establishes a foundational understanding of mobile threats and the best practices necessary to secure mobile environments. These insights are essential for anyone involved in mobile application development, administration, or cybersecurity.

3.0 Mobile Security Landscape

3.1 Android vs. iOS Security Comparison

Android and iOS are the two dominant mobile operating systems, each with distinct security architectures and approaches. Understanding their differences is crucial for assessing the risks and implementing appropriate defenses.

Android Security:

- **Open Source Nature**: Android is open source, which provides flexibility for customization but also exposes the platform to fragmentation and inconsistent security updates across different manufacturers.

- **App Distribution**: Apps can be installed from various sources, including third-party app stores and sideloading APK files, increasing the risk of installing malicious software.

- **Security Features**:
    - Google Play Protect for app scanning.
    - Sandboxing of apps for isolation.
    - Runtime permissions for better control over app access.
    - Encrypted file system and secure boot features on modern Android devices.

iOS Security:

- **Closed Ecosystem**: iOS is tightly controlled by Apple, which helps enforce consistent security standards across all devices.
- **App Distribution**: Apps must go through a rigorous review process before being listed on the App Store, minimizing the risk of malware.
- **Security Features**:
    - Code signing and app sandboxing.
    - Secure Enclave for storing sensitive data like biometric information.
    - Automatic OS updates delivered directly by Apple.
    - Data encryption at rest and during transmission.

While iOS offers a more controlled and consistent security environment, Android's flexibility makes it more prone to security issues if users or manufacturers neglect best practices. Both platforms have strong security mechanisms, but the effectiveness largely depends on user behavior, device management, and update policies.

3.2 Role of Mobile Device Management (MDM) Solutions

Mobile Device Management (MDM) solutions play a critical role in enterprise mobile security. They allow organizations to enforce security policies, manage devices remotely, and ensure compliance with security standards. Key features of MDM solutions include:

- **Remote Configuration and Control**: Admins can enforce password policies, restrict app installations, and configure VPN settings remotely.
- **Device Monitoring**: MDM tools can track device health, location, and security posture in real time.
- **Data Protection**: Enable remote wipe and encryption enforcement to protect sensitive corporate data.
- **App Management**: Control app distribution, updates, and permissions through secure enterprise app stores.

Examples of MDM solutions include Microsoft Intune, VMware Workspace ONE, and MobileIron.

3.3 Importance of Mobile Network Security

Mobile devices frequently connect to various networks, including public Wi-Fi and cellular networks, making network security vital. Key elements of mobile network security include:

- **Encrypted Communication**: Use of SSL/TLS ensures data is protected during transmission. VPNs add an extra layer of security, especially on untrusted networks.
- **Firewall and Intrusion Detection**: Monitoring mobile traffic for anomalies can help detect and prevent potential attacks.

- **Avoiding Rogue Networks**: Users should be cautious of fake access points designed to intercept traffic (e.g., evil twin attacks).

- **Network Segmentation**: In enterprise environments, separating mobile devices from critical systems via VLANs can limit potential damage from compromised devices.

4.0 Mobile Security Testing Environment

The goal of this section was to set up a mobile security testing lab environment using physical mobile devices. We utilized basic mobile security tools like MobSF (Mobile Security Framework) and Wireshark to analyze mobile traffic and assess the security of mobile applications.

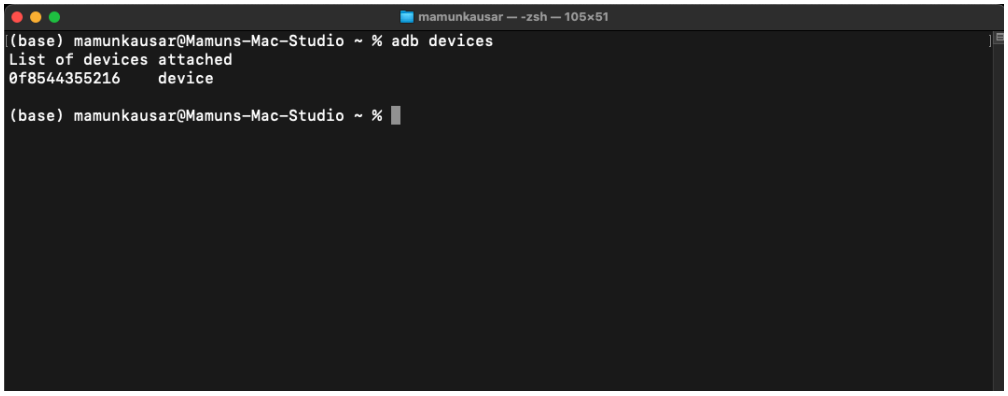
4.1 Lab Setup Overview

For the mobile security testing environment, a **Xiaomi Mi A3** physical Android device was used instead of emulators, due to better real-world testing accuracy. The device was connected to a **MacStudio** system, which allowed seamless monitoring of network traffic and interactions with mobile security tools.

4.2 Tools Used

Tool	Purpose
Mobile Security Framework (MobSF)	Static and dynamic analysis of Android APK files
Wireshark	Capturing and analyzing mobile network traffic

ADB (Android Debug Bridge)	APK extraction and device communication
iPhone (Wi-Fi Monitoring)	Comparative network traffic observation (optional)



4.3 MobSF Setup & Usage

- **Installation of MobSF**: MobSF was installed and configured on **macOS** to perform static and dynamic analysis of Android APK files. Compatibility issues were resolved by using **Python 3.10**.
- **APK Analysis**: A test APK was uploaded to MobSF for **static analysis**, providing valuable information such as:
    - App permissions
    - API endpoints
    - Hardcoded secrets
    - Insecure configurations

Steps to use MobSF:

1. Download and install MobSF by cloning its GitHub repository.

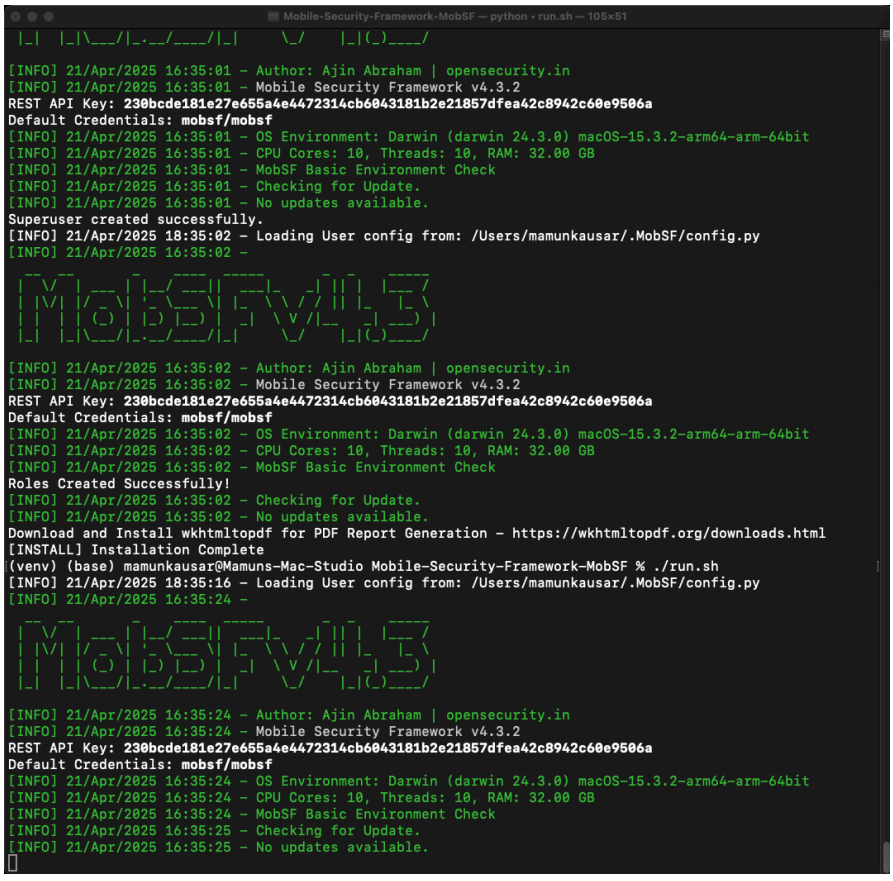
Run the MobSF server on **localhost:8000** using the following commands:

```
./setup.sh
```

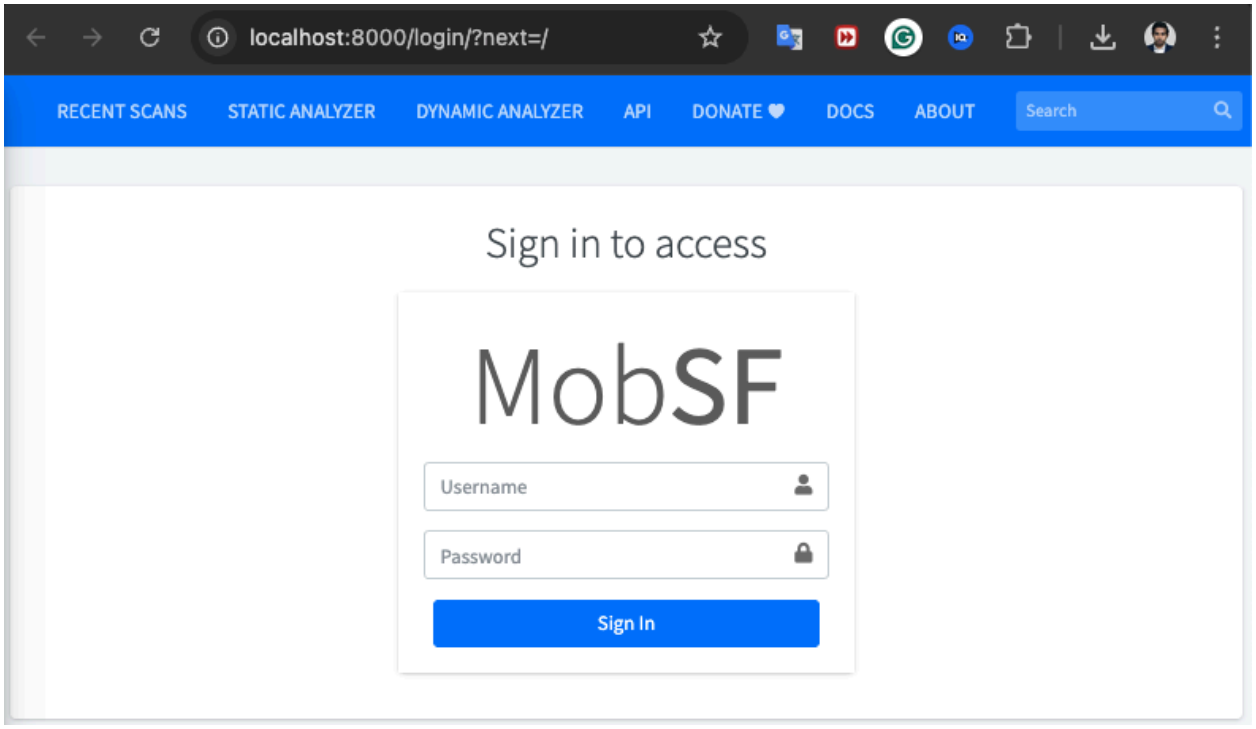
```
./run.sh
```

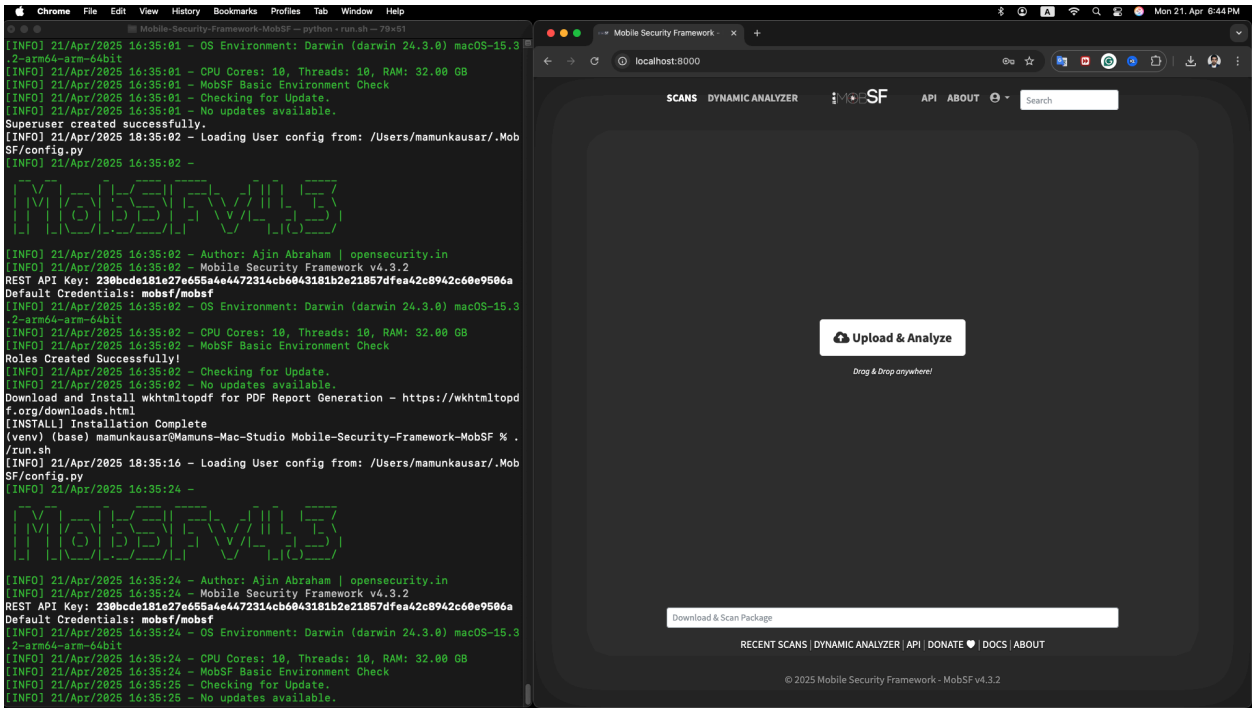
2. Upload an APK file for analysis.

Output: After the APK was uploaded, MobSF provided a comprehensive report detailing potential security issues and suggesting mitigations.

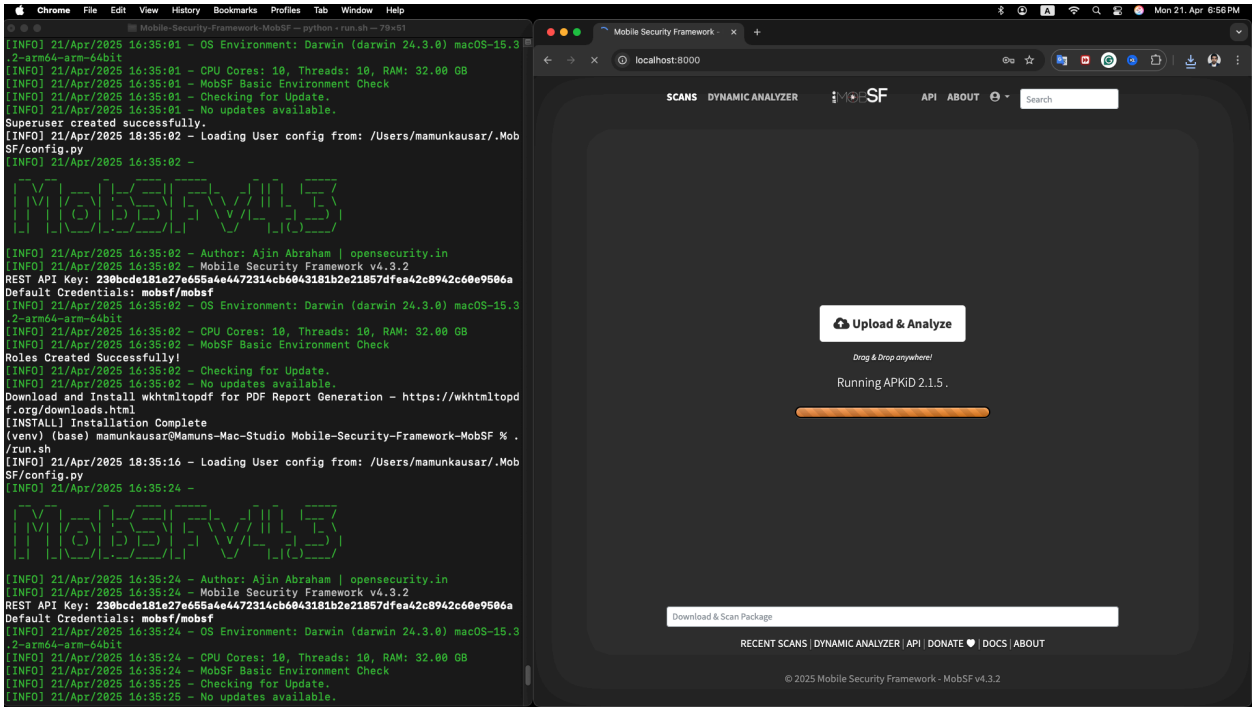


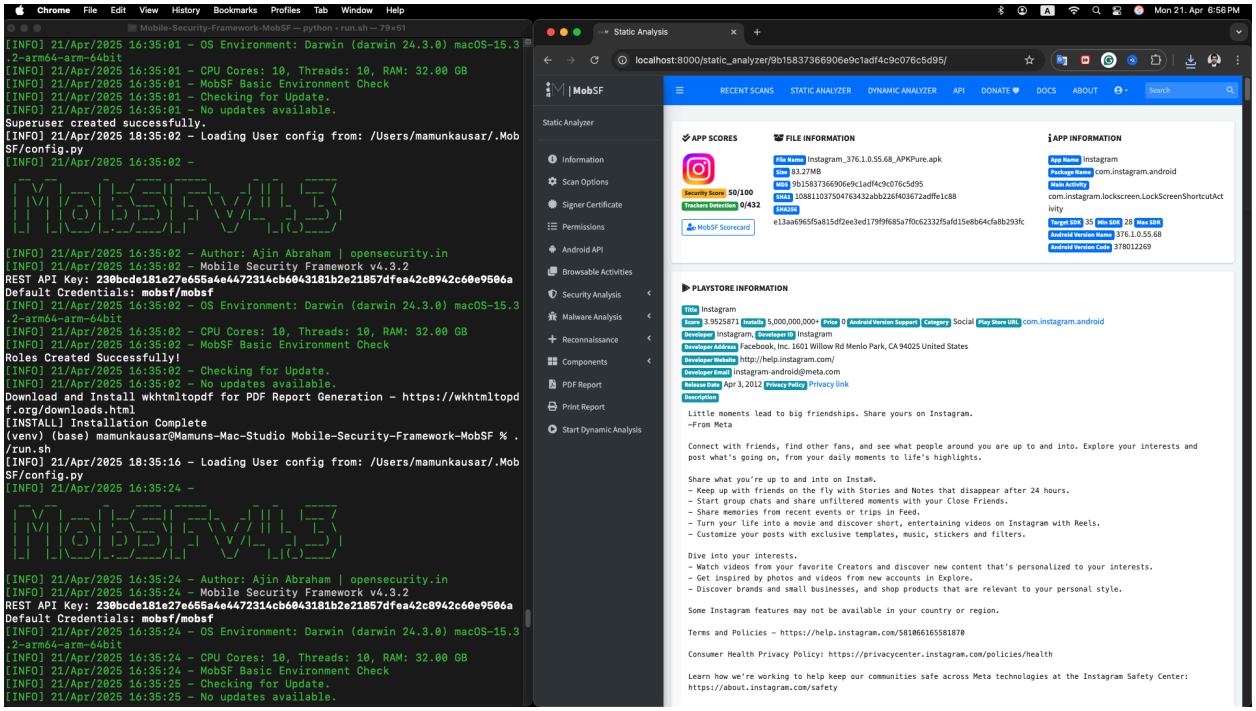

EncryptEdge Labs


localhost:8000/login/?next=/
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API
DONATE
DOCS
ABOUT
Search
Sign in to access
MobSF
Username
Password
Sign In

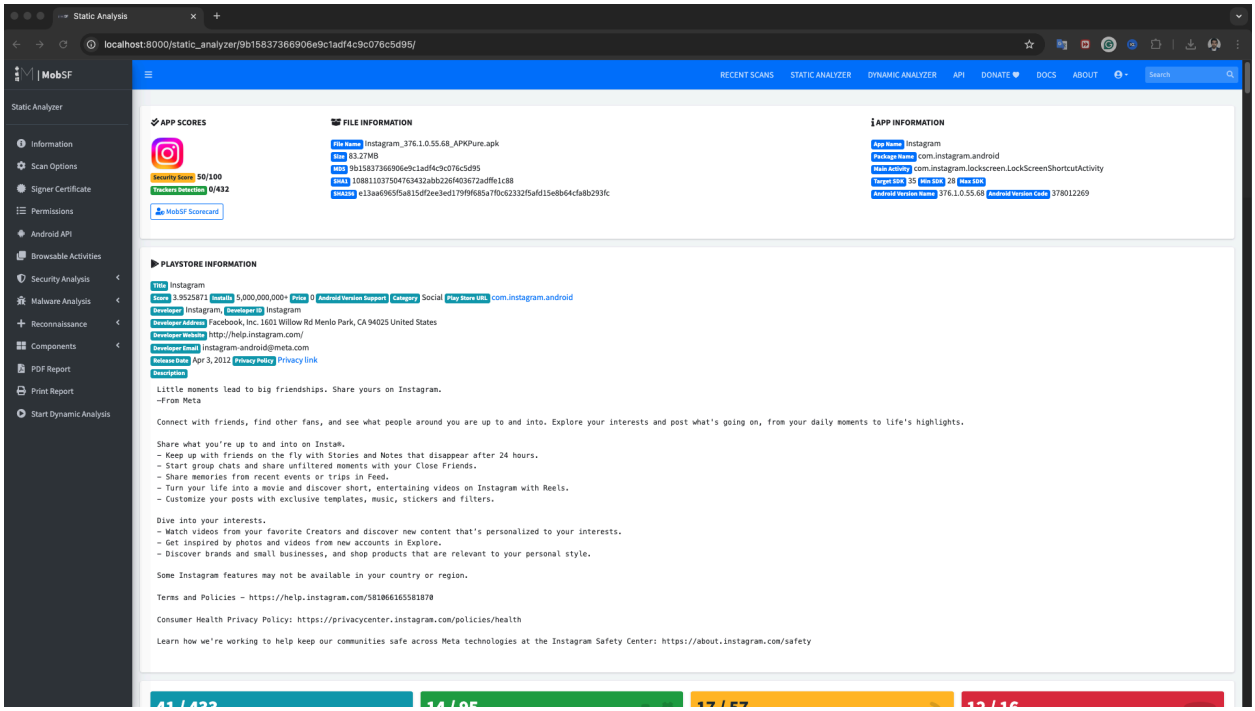

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon 21. Apr 6:44 PM
[INFO] 21/Apr/2025 16:35:01 - OS Environment: Darwin (darwin 24.3.0) macOS-15.3.2-arm64-arm-64bit
[INFO] 21/Apr/2025 16:35:01 - CPU Cores: 10, Threads: 10, RAM: 32.00 GB
[INFO] 21/Apr/2025 16:35:01 - MobSF Basic Environment Check
[INFO] 21/Apr/2025 16:35:01 - Checking for Update.
[INFO] 21/Apr/2025 16:35:01 - No updates available.
Superuser created successfully.
[INFO] 21/Apr/2025 18:35:02 - Loading User config from: /Users/mamunkausar/.MobSF/config.py
[INFO] 21/Apr/2025 16:35:02 -
[INFO] 21/Apr/2025 16:35:02 - Author: Ajin Abraham | opensecurity.in
[INFO] 21/Apr/2025 16:35:02 - Mobile Security Framework v4.3.2
REST API Key: 230bcde181e27e655a4e4472314cb6043181b2e21857dfea42c8942c60e9506a
Default Credentials: mobsf/mobsf
[INFO] 21/Apr/2025 16:35:02 - OS Environment: Darwin (darwin 24.3.0) macOS-15.3.2-arm64-arm-64bit
[INFO] 21/Apr/2025 16:35:02 - CPU Cores: 10, Threads: 10, RAM: 32.00 GB
[INFO] 21/Apr/2025 16:35:02 - MobSF Basic Environment Check
Roles Created Successfully!
[INFO] 21/Apr/2025 16:35:02 - Checking for Update.
[INFO] 21/Apr/2025 16:35:02 - No updates available.
Download and Install wkhtmltopdf for PDF Report Generation - https://wkhtmltopdf.org/downloads.html
[INSTALL] Installation Complete
(venv) (base) mamunkausar@Mamuns-Mac-Studio Mobile-Security-Framework-MobSF % ./run.sh
[INFO] 21/Apr/2025 18:35:16 - Loading User config from: /Users/mamunkausar/.MobSF/config.py
[INFO] 21/Apr/2025 16:35:24 -
[INFO] 21/Apr/2025 16:35:24 - Author: Ajin Abraham | opensecurity.in
[INFO] 21/Apr/2025 16:35:24 - Mobile Security Framework v4.3.2
REST API Key: 230bcde181e27e655a4e4472314cb6043181b2e21857dfea42c8942c60e9506a
Default Credentials: mobsf/mobsf
[INFO] 21/Apr/2025 16:35:24 - OS Environment: Darwin (darwin 24.3.0) macOS-15.3.2-arm64-arm-64bit
[INFO] 21/Apr/2025 16:35:24 - CPU Cores: 10, Threads: 10, RAM: 32.00 GB
[INFO] 21/Apr/2025 16:35:24 - MobSF Basic Environment Check
[INFO] 21/Apr/2025 16:35:25 - Checking for Update.
[INFO] 21/Apr/2025 16:35:25 - No updates available.
Mobile Security Framework
localhost:8000
SCANS
DYNAMIC ANALYZER
API
ABOUT
Search
Upload & Analyze
Drag & Drop anywhere!
Download & Scan Package
RECENT SCANS | DYNAMIC ANALYZER | API | DONATE | DOCS | ABOUT
© 2025 Mobile Security Framework - MobSF v4.3.2

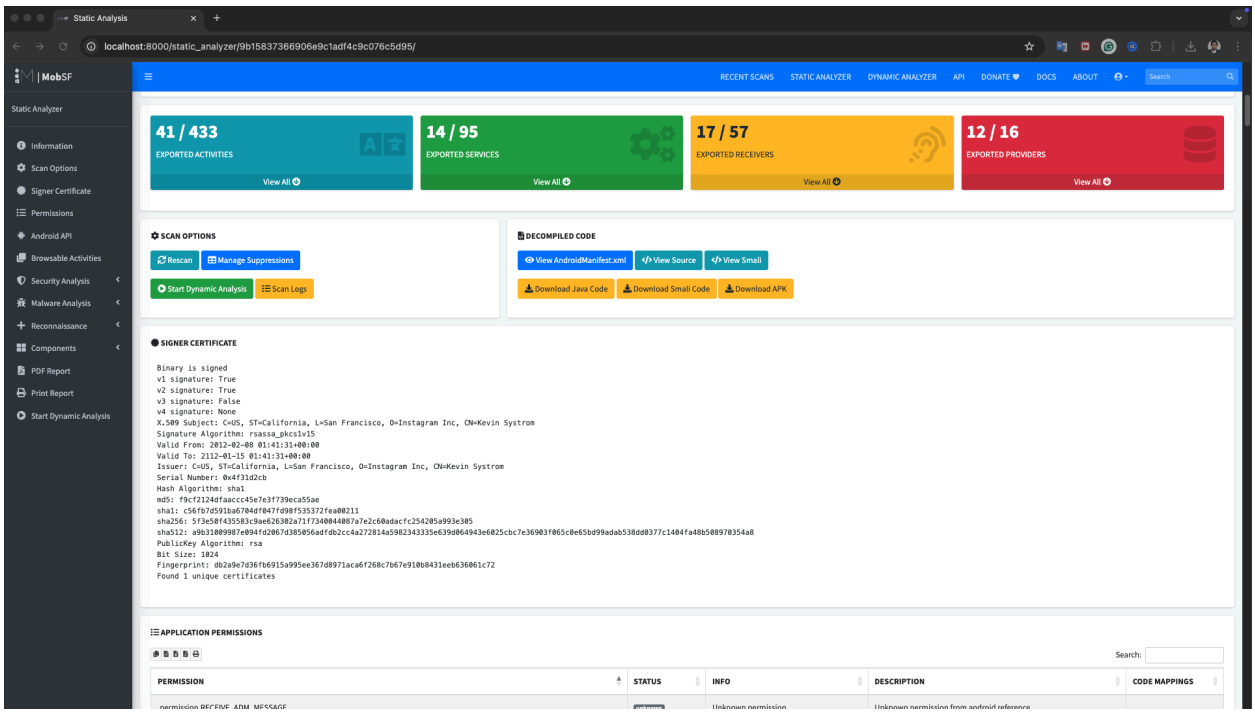
EncryptEdge Labs

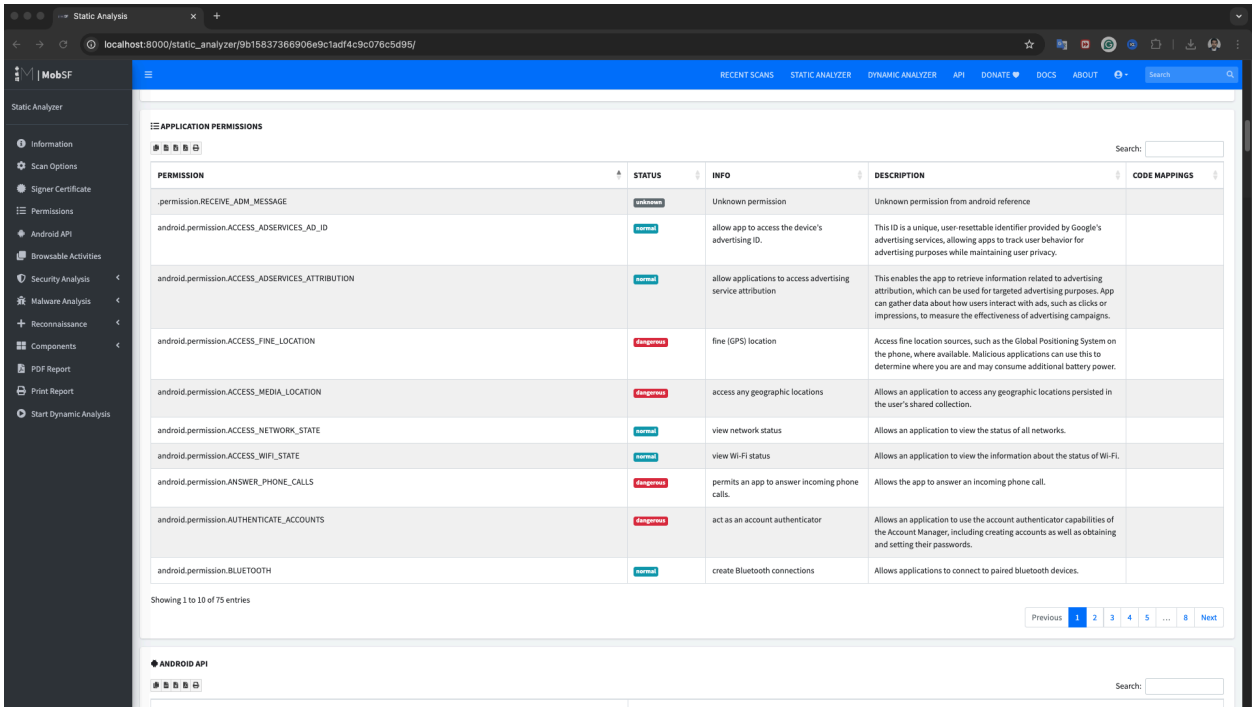


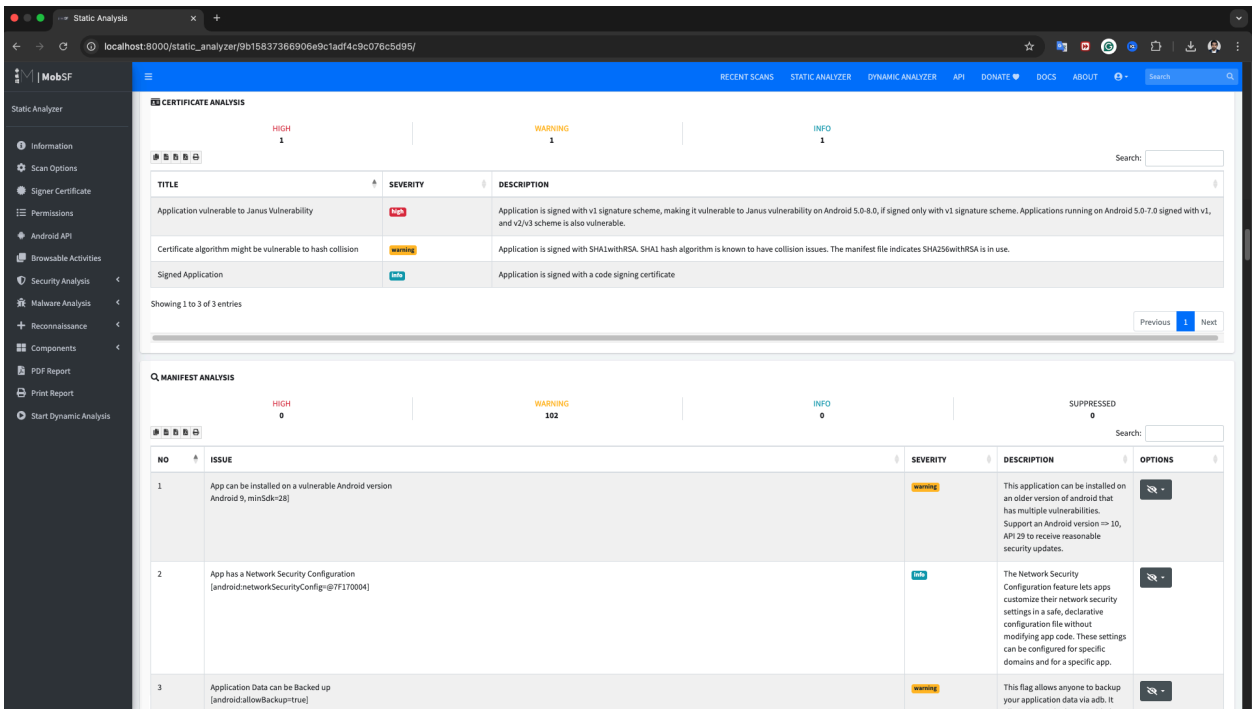


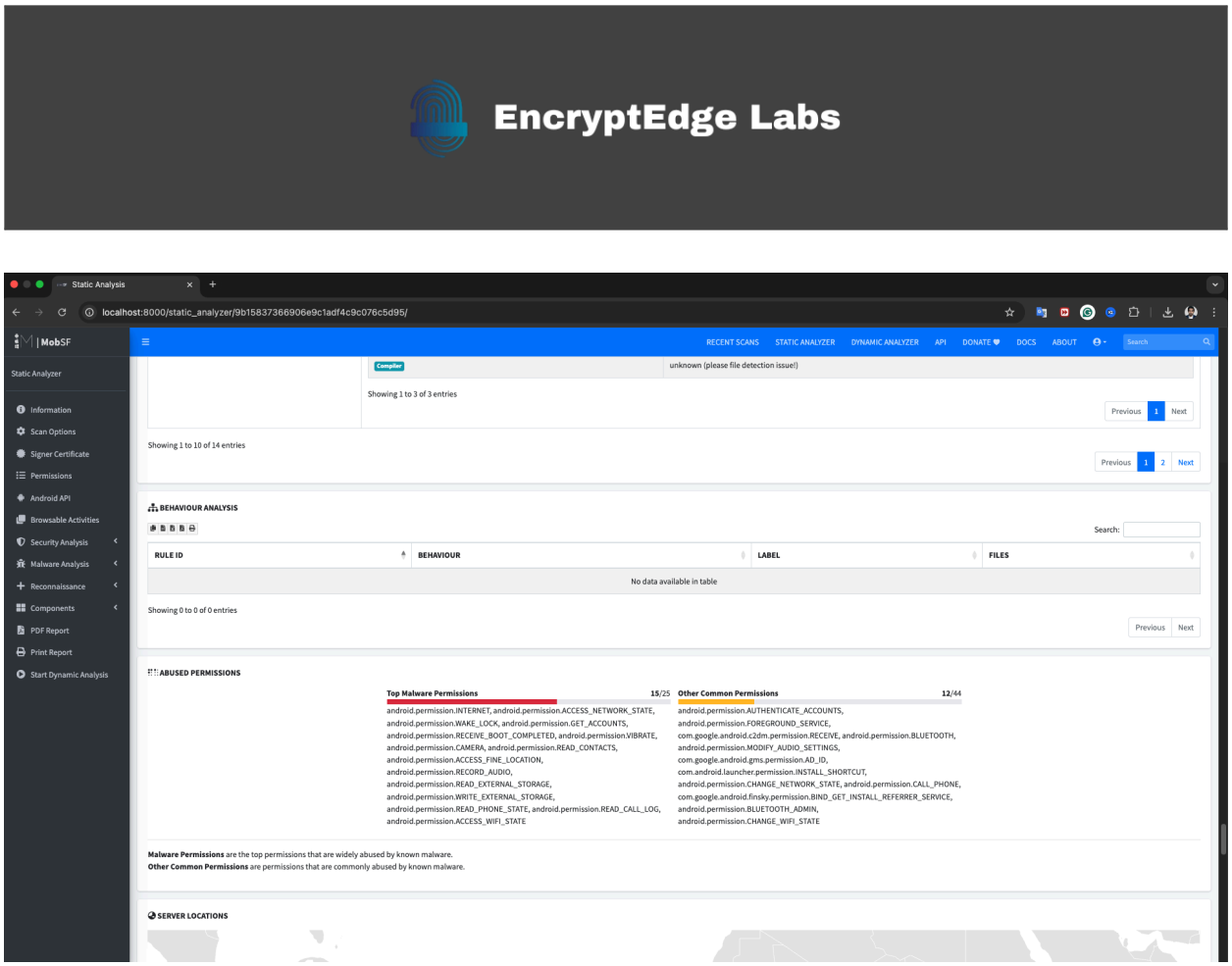

EncryptEdge Labs


Static Analysis
localhost:8000/static_analyzer/9b15837366906e9c1adf4c9c076c5d95/
MobSF
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API
DONATE
DOCS
ABOUT
Static Analyzer
Information
Scan Options
Signer Certificate
Permissions
Android API
Browsable Activities
Security Analysis
Malware Analysis
Reconnaissance
Components
PDF Report
Print Report
Start Dynamic Analysis
APP SCORES
Security Score 50/100
Trackers Detection 0/432
MobSF Scorecard
FILE INFORMATION
File Name instagram_376.1.0.55.68_APKPure.apk
Size 83.27MB
MD5 9b15837366906e9c1adf4c9c076c5d95
SHA1 10881103750476343 2abb226f403672adffe1c88
SHA256 e13aa6965f5a815df2ee3ed179f9f685a7f0c62332f5afd15e8b64cfa8b293fc
APP INFORMATION
App Name Instagram
Package Name com.instagram.android
Main Activity com.instagram.lockscreen.LockScreenShortcutActivity
Target SDK 35 Min SDK 28 Max SDK
Android Version Name 376.1.0.55.68 Android Version Code 378012269
PLAYSTORE INFORMATION
Title Instagram
Score 3.9525871 Installs 5,000,000,000+ Price 0 Android Version Support Category Social Play Store URL com.instagram.android
Developer Instagram, Developer ID Instagram
Developer Address Facebook, Inc. 1601 Willow Rd Menlo Park, CA 94025 United States
Developer Website http://help.instagram.com/
Developer Email instagram-android@meta.com
Release Date Apr 3, 2012 Privacy Policy Privacy link
Description
Little moments lead to big friendships. Share yours on Instagram.
–From Meta
Connect with friends, find other fans, and see what people around you are up to and into. Explore your interests and post what's going on, from your daily moments to life's highlights.
Share what you're up to and into on Insta®.
– Keep up with friends on the fly with Stories and Notes that disappear after 24 hours.
– Start group chats and share unfiltered moments with your Close Friends.
– Share memories from recent events or trips in Feed.
– Turn your life into a movie and discover short, entertaining videos on Instagram with Reels.
– Customize your posts with exclusive templates, music, stickers and filters.
Dive into your interests.
– Watch videos from your favorite Creators and discover new content that's personalized to your interests.
– Get inspired by photos and videos from new accounts in Explore.
– Discover brands and small businesses, and shop products that are relevant to your personal style.
Some Instagram features may not be available in your country or region.
Terms and Policies – https://help.instagram.com/581066165581870
Consumer Health Privacy Policy: https://privacycenter.instagram.com/policies/health
Learn how we're working to help keep our communities safe across Meta technologies at the Instagram Safety Center: https://about.instagram.com/safety


Static Analysis
localhost:8000/static_analyzer/9b15837366906e9c1adf4c9c076c5d95/
MobSF
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API
DONATE
DOCS
ABOUT
Static Analyzer
Information
Scan Options
Signer Certificate
Permissions
Android API
Browsable Activities
Security Analysis
Malware Analysis
Reconnaissance
Components
PDF Report
Print Report
Start Dynamic Analysis
41 / 433
EXPORTED ACTIVITIES
View All
14 / 95
EXPORTED SERVICES
View All
17 / 57
EXPORTED RECEIVERS
View All
12 / 16
EXPORTED PROVIDERS
View All
SCAN OPTIONS
Rescan
Manage Suppressions
Start Dynamic Analysis
Scan Logs
DECOMPILED CODE
View AndroidManifest.xml
View Source
View Smali
Download Java Code
Download Smali Code
Download APK
SIGNER CERTIFICATE
Binary is signed
v1 signature: True
v2 signature: True
v3 signature: False
v4 signature: None
X.509 Subject: C=US, ST=California, L=San Francisco, O=Instagram Inc, CN=Kevin Systrom
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2012-02-08 01:41:31+00:00
Valid To: 2112-01-15 01:41:31+00:00
Issuer: C=US, ST=California, L=San Francisco, O=Instagram Inc, CN=Kevin Systrom
Serial Number: 0x4f31d2cb
Hash Algorithm: sha1
md5: f9cf2124dfaaccc45e7e3f739eca55ae
sha1: c56fb7d591ba6704df047fd98f535372fea00211
sha256: 5f3e50f435583c9ae626302a71f7340044087a7e2c60adacfc254205a993e305
sha512: a9b31009987e094fd2067d385056adfdb2cc4a272814a5982343335e639d064943e6025cbc7e36903f065c0e65bd99adab538dd0377c1404fa48b508970354a8
PublicKey Algorithm: rsa
Bit Size: 1024
Fingerprint: db2a9e7d36fb6915a995ee367d8971aca6f268c7b67e910b8431eeb636061c72
Found 1 unique certificates
APPLICATION PERMISSIONS
Search:
PERMISSION
STATUS
INFO
DESCRIPTION
CODE MAPPINGS


Static Analysis
localhost:8000/static_analyzer/9b15837366906e9c1adf4c9c076c5d95/
MobSF
Static Analyzer
Information
Scan Options
Signer Certificate
Permissions
Android API
Browsable Activities
Security Analysis
Malware Analysis
Reconnaissance
Components
PDF Report
Print Report
Start Dynamic Analysis
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API
DONATE
DOCS
ABOUT
APPLICATION PERMISSIONS
Search:
PERMISSION
STATUS
INFO
DESCRIPTION
CODE MAPPINGS
.permission.RECEIVE_ADM_MESSAGE
unknown
Unknown permission
Unknown permission from android reference
android.permission.ACCESS_ADSERVICES_AD_ID
normal
allow app to access the device's advertising ID.
This ID is a unique, user-resettable identifier provided by Google's advertising services, allowing apps to track user behavior for advertising purposes while maintaining user privacy.
android.permission.ACCESS_ADSERVICES_ATTRIBUTION
normal
allow applications to access advertising service attribution
This enables the app to retrieve information related to advertising attribution, which can be used for targeted advertising purposes. App can gather data about how users interact with ads, such as clicks or impressions, to measure the effectiveness of advertising campaigns.
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.ACCESS_MEDIA_LOCATION
dangerous
access any geographic locations
Allows an application to access any geographic locations persisted in the user's shared collection.
android.permission.ACCESS_NETWORK_STATE
normal
view network status
Allows an application to view the status of all networks.
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.ANSWER_PHONE_CALLS
dangerous
permits an app to answer incoming phone calls.
Allows the app to answer an incoming phone call.
android.permission.AUTHENTICATE_ACCOUNTS
dangerous
act as an account authenticator
Allows an application to use the account authenticator capabilities of the Account Manager, including creating accounts as well as obtaining and setting their passwords.
android.permission.BLUETOOTH
normal
create Bluetooth connections
Allows applications to connect to paired bluetooth devices.
Showing 1 to 10 of 75 entries
Previous
1
2
3
4
5
...
8
Next
ANDROID API
Search:


Static Analysis
localhost:8000/static_analyzer/9b15837366906e9c1adf4c9c076c5d95/
MobSF
Static Analyzer
Information
Scan Options
Signer Certificate
Permissions
Android API
Browsable Activities
Security Analysis
Malware Analysis
Reconnaissance
Components
PDF Report
Print Report
Start Dynamic Analysis
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API
DONATE
DOCS
ABOUT
CERTIFICATE ANALYSIS
HIGH
1
WARNING
1
INFO
1
Search:
TITLE
SEVERITY
DESCRIPTION
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.
Certificate algorithm might be vulnerable to hash collision
warning
Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues. The manifest file indicates SHA256withRSA is in use.
Signed Application
info
Application is signed with a code signing certificate
Showing 1 to 3 of 3 entries
Previous
1
Next
MANIFEST ANALYSIS
HIGH
0
WARNING
102
INFO
0
SUPPRESSED
0
Search:
NO
ISSUE
SEVERITY
DESCRIPTION
OPTIONS
1
App can be installed on a vulnerable Android version
Android 9, minSdk=28]
warning
This application can be installed on an older version of android that has multiple vulnerabilities.
Support an Android version => 10, API 29 to receive reasonable security updates.
2
App has a Network Security Configuration
[android:networkSecurityConfig=@7F170004]
info
The Network Security Configuration feature lets apps customize their network security settings in a safe, declarative configuration file without modifying app code. These settings can be configured for specific domains and for a specific app.
3
Application Data can be Backed up
[android:allowBackup=true]
warning
This flag allows anyone to backup your application data via adb. It



4.4 Wireshark Network Traffic Capture

To capture and analyze network traffic:

- The Android device was connected to a **Wi-Fi hotspot** created by the MacStudio system.
- **Wireshark** was used to monitor the traffic sent and received by the mobile device while it was actively browsing and interacting with various apps.

Wireshark Setup:

1. Start **Wireshark** and select the Wi-Fi interface connected to the mobile hotspot.

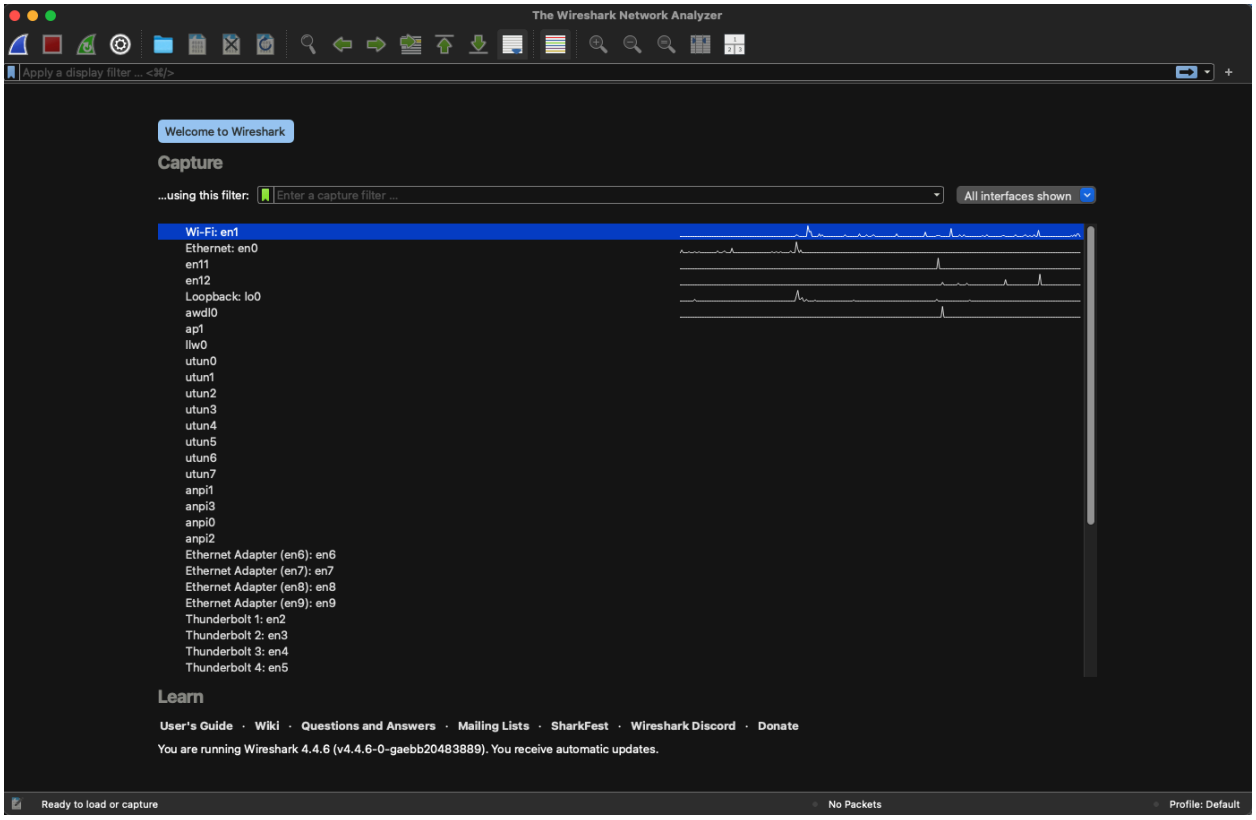
2. Apply filters (e.g., `ip.addr == 192.168.1.102`) to focus on traffic from the mobile device.

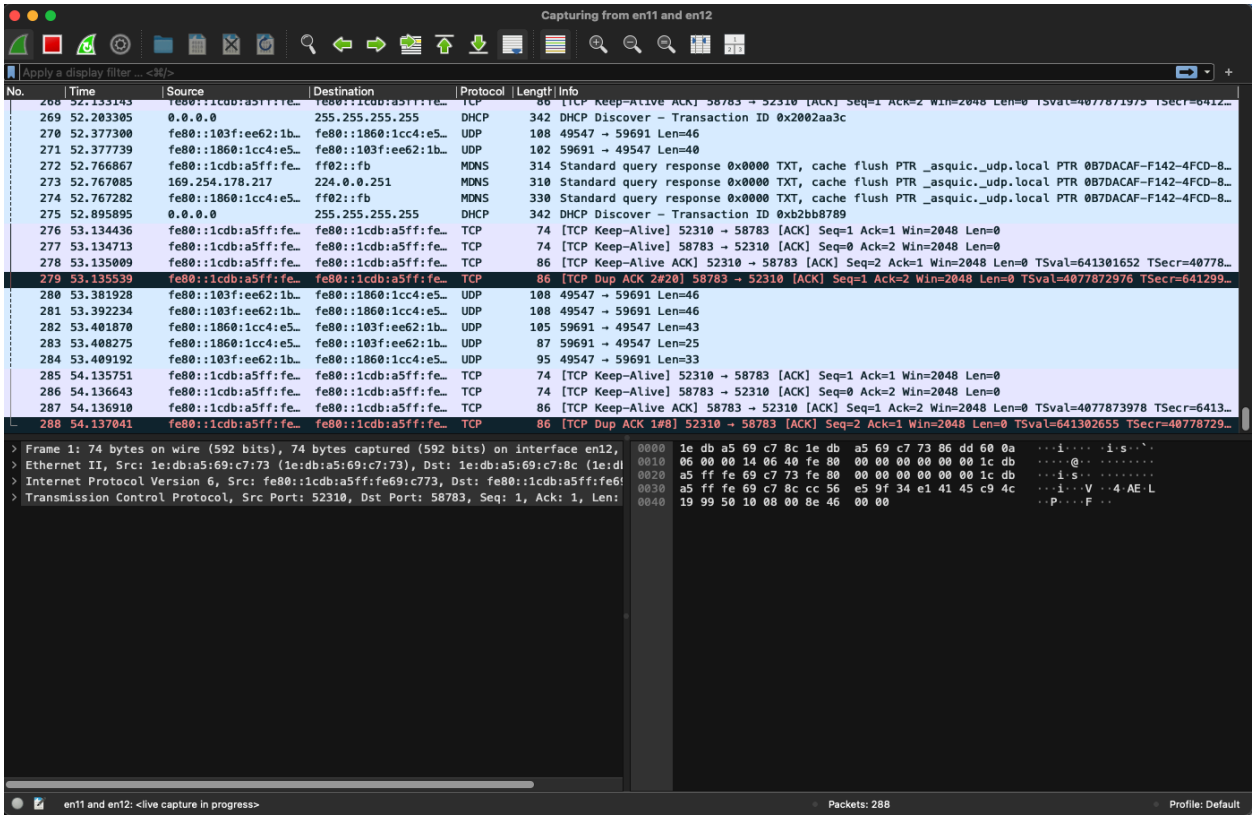
3. Analyze the captured packets, looking for vulnerabilities such as unencrypted communication, DNS queries, and API requests.

Output: The captured traffic displayed interactions between the mobile device and external servers, showing encrypted and unencrypted data streams. Key observations included:

- HTTP requests without SSL/TLS encryption

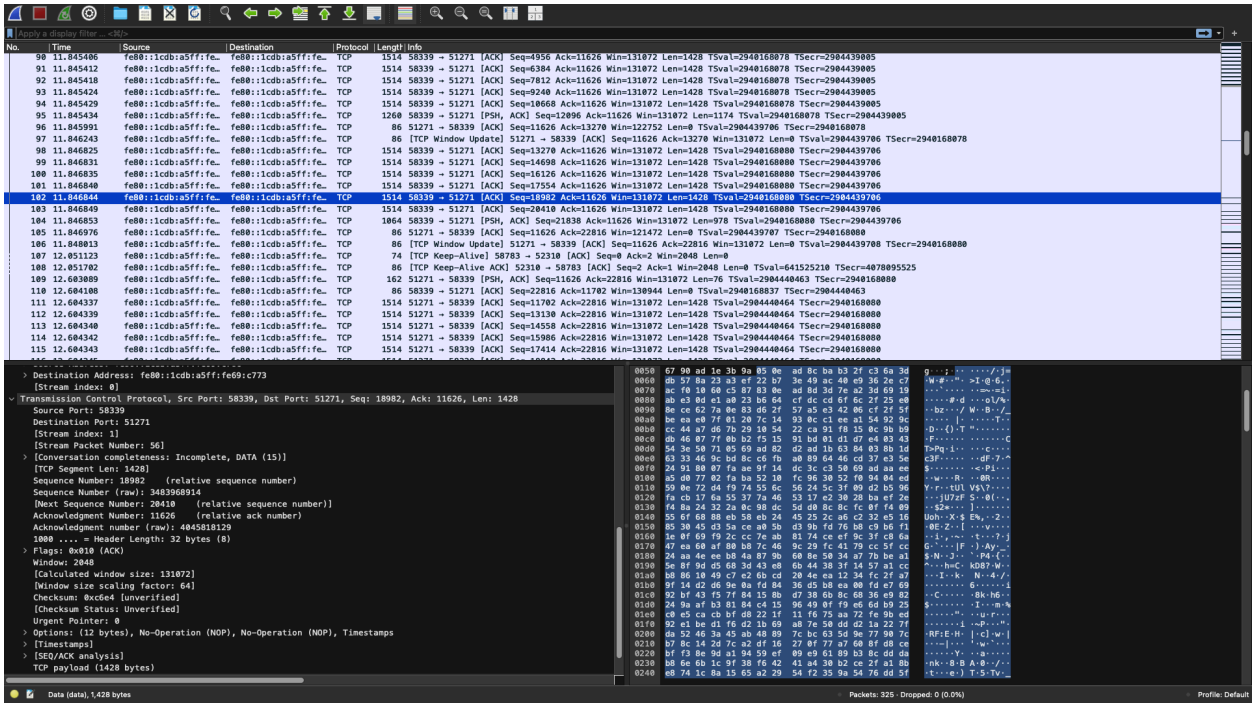
- Sensitive data being transmitted over insecure protocols





4.5 Optional iPhone Traffic Monitoring

To compare mobile security behaviors, an **iPhone** was connected to the same Wi-Fi network. Using **Wireshark**, we monitored its network traffic and compared it to the traffic of the Android device. We identified several key differences in traffic patterns, especially in how encrypted data was handled.



In this section, a **mobile security testing environment** was successfully established using a physical Android device and **MobSF** for app analysis. **Wireshark** was used to capture network traffic, revealing insights into insecure communication and vulnerabilities in mobile applications. The setup allowed us to perform both static and dynamic testing, laying the foundation for further security assessments and improvements.

5.0 Mobile Threat Identification and Mitigation

The aim of this section was to identify and mitigate common security risks in mobile environments. We focused on identifying vulnerabilities such as insecure data storage, weak authentication, and inadequate encryption in a pre-existing mobile app. Using mobile security tools, we scanned for vulnerabilities and implemented mitigation strategies to enhance the app's security.

5.1 Common Mobile Security Vulnerabilities

Several key vulnerabilities were targeted for identification and mitigation, including:

1. **Insecure Data Storage**: Mobile apps may store sensitive data such as passwords or tokens insecurely on the device. This data may be accessible to attackers if the device is compromised or if proper security measures are not in place.

2. **Weak Authentication**: Insufficient or poorly implemented authentication mechanisms can expose mobile apps to unauthorized access. Weak password policies, lack of two-factor authentication (2FA), and improper session management are examples of weak authentication.

3. **Inadequate Encryption**: Sensitive data in transit and at rest must be encrypted using secure protocols. Failing to implement encryption or using weak encryption algorithms can lead to data exposure during communication or storage.

5.2 Vulnerability Scanning and Analysis

To identify vulnerabilities, a **sample APK** file was scanned using **MobSF**. The following vulnerabilities were detected during the analysis:

1. **Insecure Data Storage**:

    - **Findings**: MobSF flagged the app for storing sensitive information (such as API keys and passwords) in **plain text** within shared preferences and local storage.

    - **Mitigation**: It was recommended to encrypt sensitive data before storing it on the device using Android's **Keystore system**.

2. **Weak Authentication**:

    - **Findings**: The app's authentication mechanism was found to lack support for **multi-factor authentication (MFA)**, and passwords were stored in an

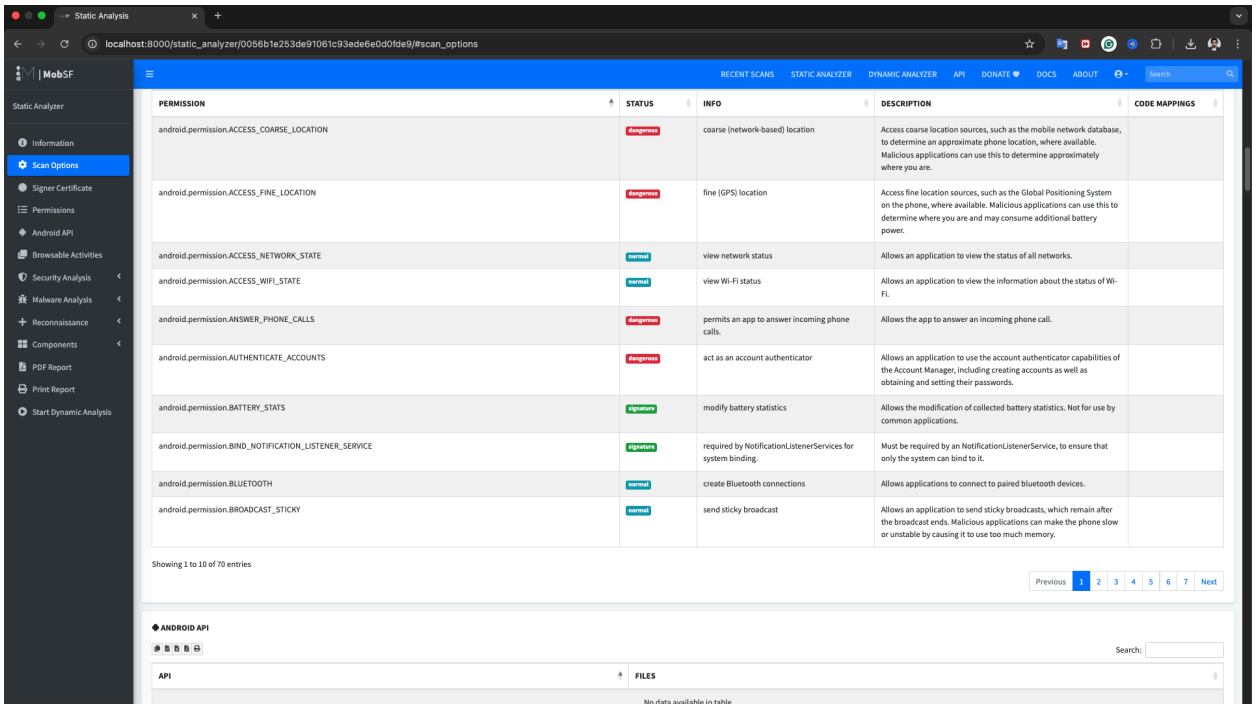
unencrypted format.

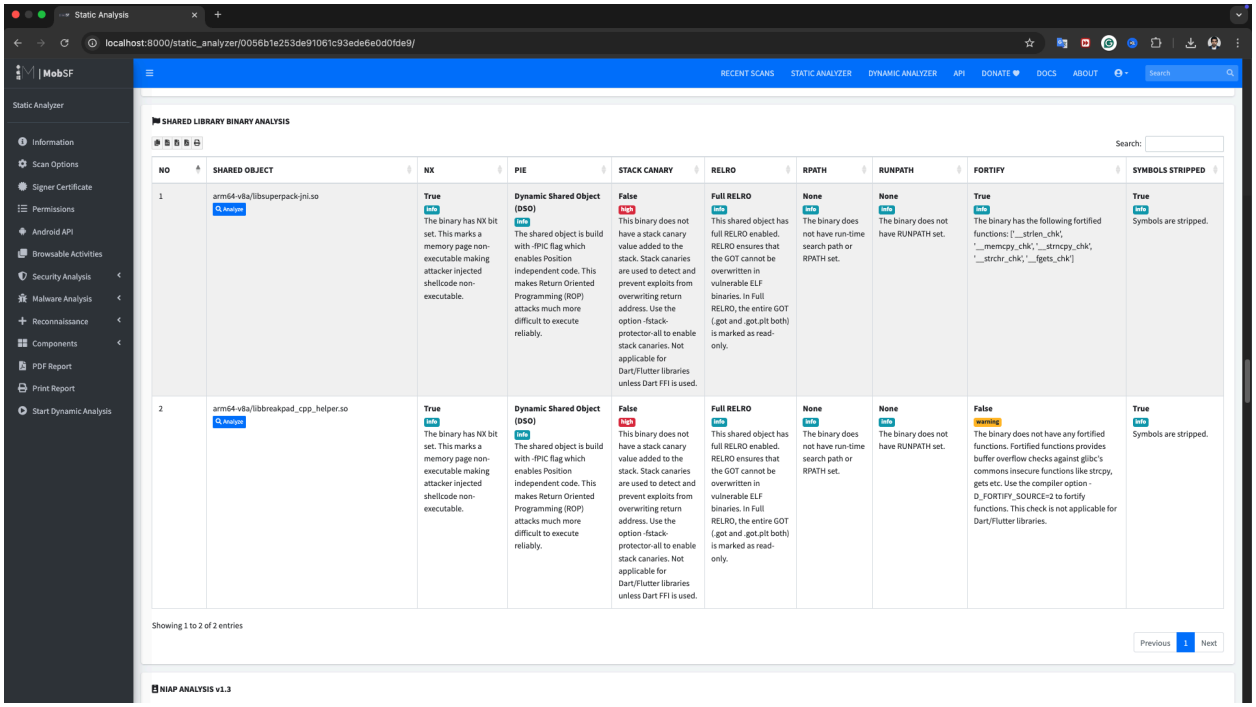
- **Mitigation**: The app was updated to implement **stronger password policies** (e.g., minimum length, complexity) and integrate **MFA** using SMS-based or app-based tokens.

3. **Inadequate Encryption**:

   - **Findings**: The app transmitted sensitive data over **unencrypted HTTP** rather than **HTTPS**. This left the data vulnerable to man-in-the-middle (MITM) attacks.

   - **Mitigation**: A strict **SSL/TLS encryption** policy was implemented to ensure that all sensitive data in transit is encrypted.





5.3 Mitigation Strategies Implemented

Based on the identified vulnerabilities, the following mitigation strategies were implemented to enhance the security of the mobile app:

1. **Encrypting Sensitive Data**:
    - The app's sensitive data was encrypted using **AES encryption** before being stored on the device. This ensured that even if the device were compromised, the data would remain secure.
    - The **Android Keystore** system was used to securely store cryptographic keys.
2. **Strengthening Authentication**:

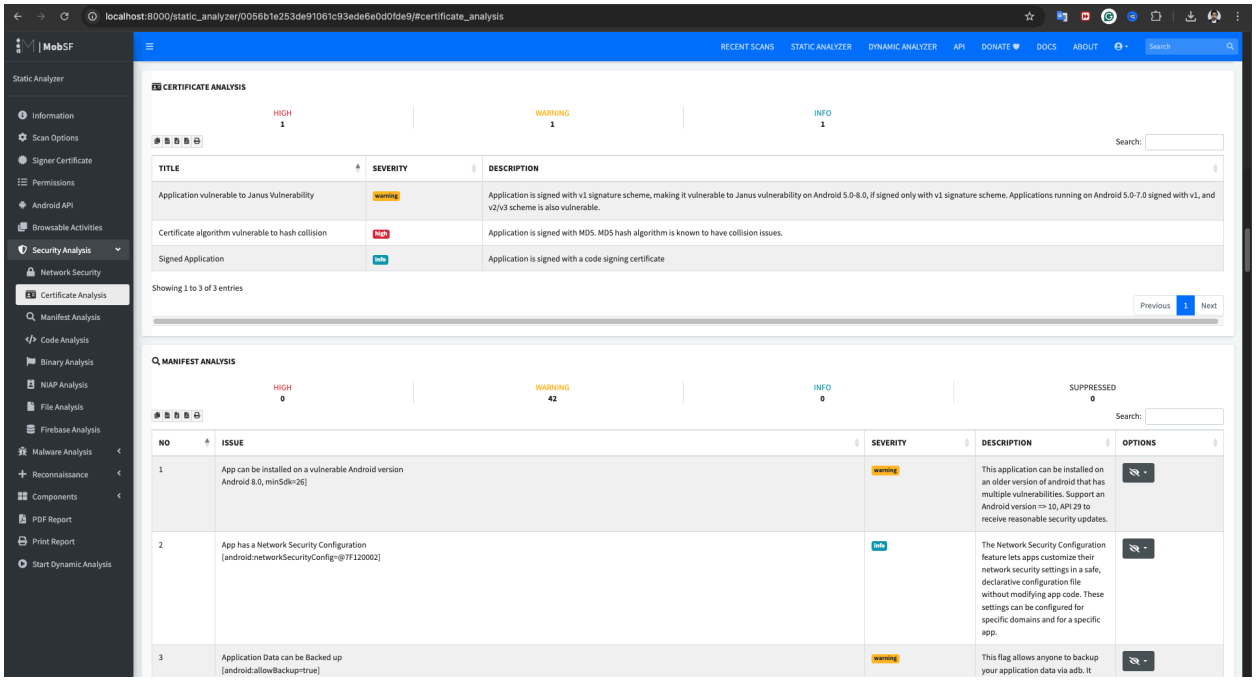
- Password policies were updated to require a **minimum length of 12 characters**, including both uppercase and lowercase letters, numbers, and symbols.

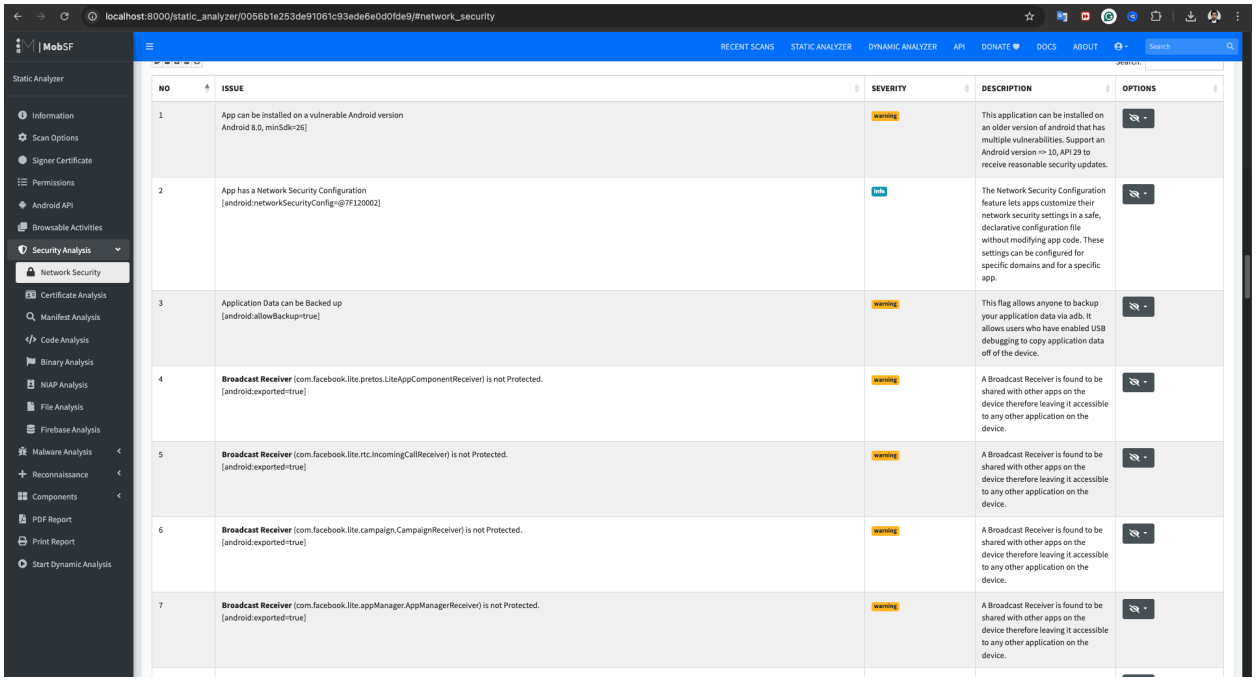
- The app was integrated with **multi-factor authentication (MFA)**, requiring a second form of verification (e.g., SMS or app-based code) in addition to a password.

3. **Enforcing HTTPS**:

   - A **network security configuration** was added to the app to enforce **SSL/TLS** for all communications.

   - The app was configured to prevent connections over **HTTP** and to **reject any untrusted SSL certificates**.



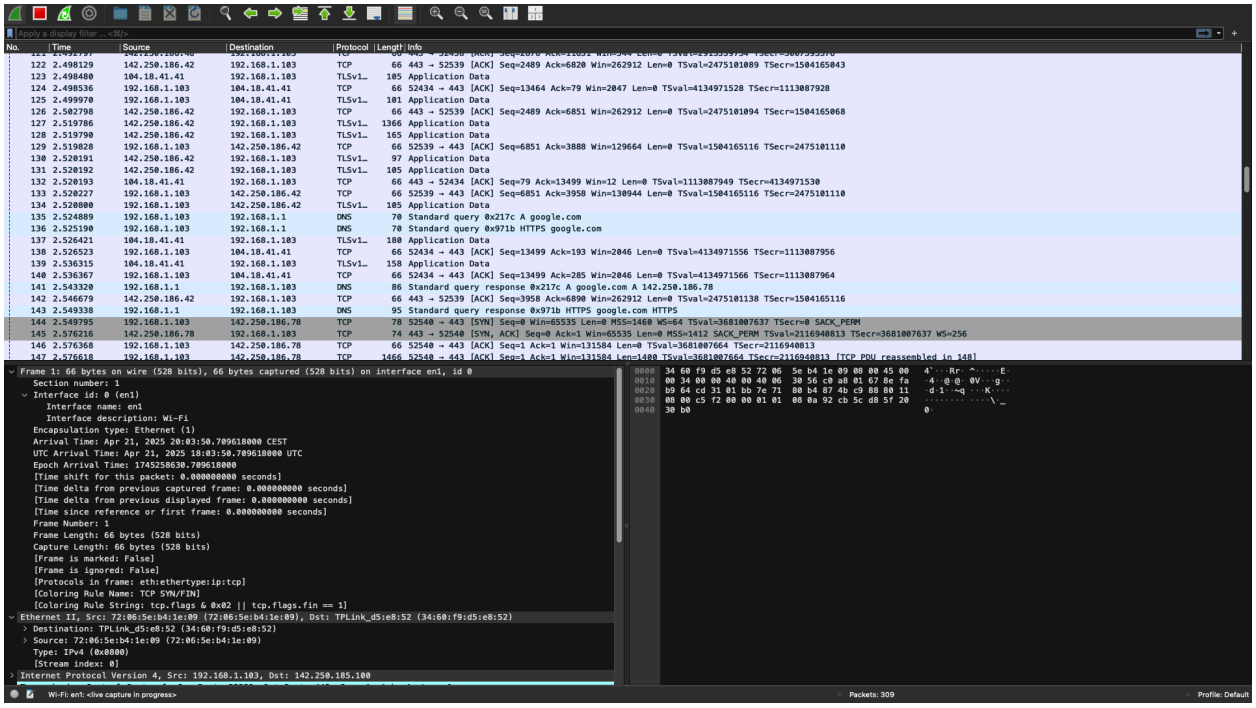


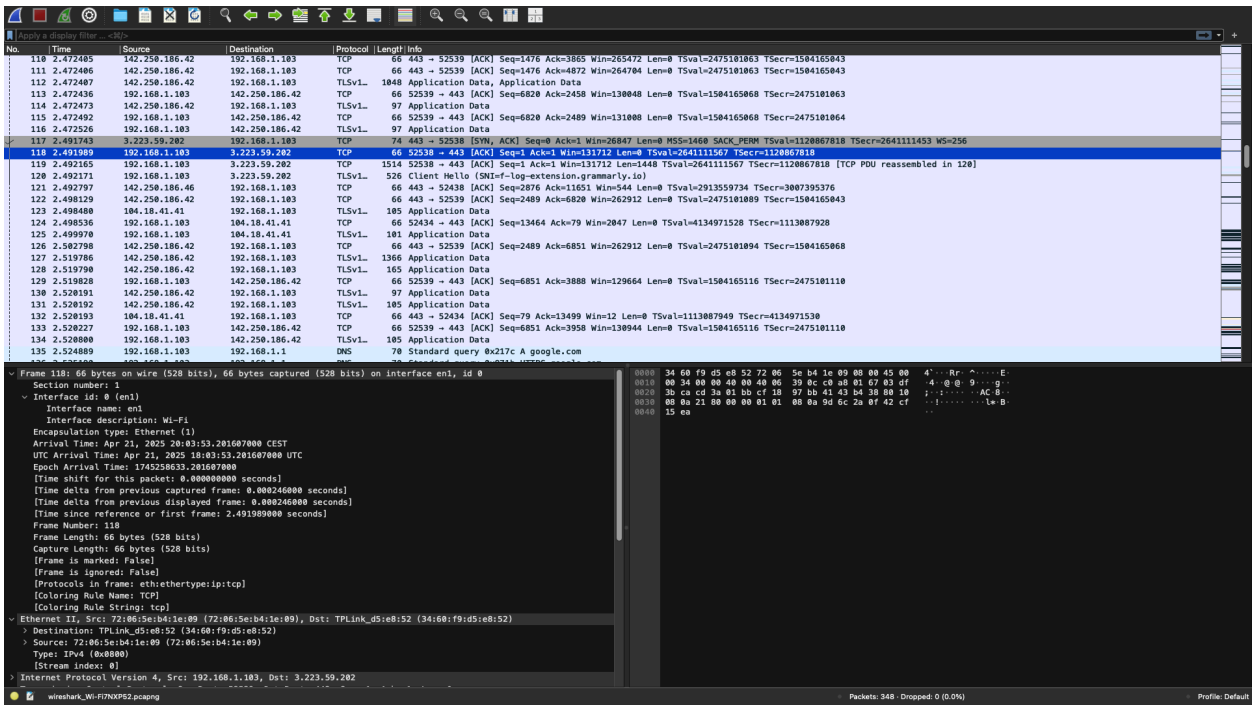
5.4 Verification of Mitigations

Once the mitigations were applied, we performed a **re-scan using MobSF** and verified that the vulnerabilities had been mitigated:

- **Insecure Data Storage**: The app now encrypts sensitive data before storage.
- **Weak Authentication**: Multi-factor authentication was successfully integrated, and password policies were enforced.
- **Inadequate Encryption**: The app now only communicates over HTTPS, ensuring that sensitive data is encrypted in transit.

Wireshark Capture: We also verified that no unencrypted data was transmitted by monitoring the app's network traffic with **Wireshark**. The traffic was now encrypted, and no sensitive information was transmitted in clear text.





This section provided hands-on experience in identifying and mitigating common security risks in mobile applications. By using **MobSF**, **Drozer**, and **Wireshark**, we were able to discover vulnerabilities such as insecure data storage, weak authentication, and inadequate encryption. We then implemented effective mitigation strategies, such as encrypting sensitive data, enforcing strong authentication, and ensuring secure communication through HTTPS. The mitigations significantly enhanced the app's overall security posture.

6.0 Real-World Mobile Security Scenarios

The goal of this section was to research real-world mobile security breaches, analyze how they occurred, and explore methods to prevent them. Additionally, we simulated common mobile attacks, such as man-in-the-middle (MITM) attacks and app tampering, in a controlled environment and practiced defensive measures to secure the mobile app.

6.1 Real-World Mobile Security Breaches

Case Study 1: The "Stagefright" Vulnerability (Android)

Overview: The Stagefright vulnerability, discovered in 2015, was one of the most significant security flaws in Android devices. The flaw resided in the multimedia processing software Stagefright, which handled media files like MMS (Multimedia Messaging Service). The vulnerability allowed attackers to remotely execute arbitrary code on the affected device by sending a specially crafted MMS message. The message would trigger the Stagefright bug, giving the attacker full control over the device.

How It Occurred: The vulnerability existed in Android's media processing framework, and millions of devices were vulnerable because the flaw could be triggered without any user interaction. Once the malicious MMS was received, the device was compromised without the user ever opening the message or viewing its content.

How It Could Have Been Prevented:

1. **Regular Patch Updates**: Android devices should have received prompt security patches to mitigate the Stagefright vulnerability. Google released a patch, but many device manufacturers were slow to roll out the fix, leaving users exposed.

2. **App Permission Management**: Limiting app access to sensitive features like media processing and implementing stricter app sandboxing could have reduced the attack surface.

3. **Security Awareness**: Educating users to avoid opening suspicious MMS messages could have minimized the impact of the attack.

Case Study 2: "XCodeGhost" (iOS)

Overview: The XCodeGhost attack targeted iOS apps and involved the malicious modification of the XCode development environment, which developers use to create iOS apps. The attacker injected malicious code into legitimate apps during their development phase, which allowed them to exfiltrate sensitive data from users' devices.

How It Occurred: The attacker uploaded a malicious version of XCode to third-party repositories, which developers unknowingly downloaded and used. When developers compiled their apps, the injected code created a backdoor that could steal sensitive user data like usernames, passwords, and even device information. The affected apps were then distributed through the Apple App Store.

How It Could Have Been Prevented:

1. **Using Official Sources for Tools**: Developers should always use the official XCode tool provided by Apple to ensure the integrity of the development environment.

2. **Code Audits and Reverse Engineering**: Developers should regularly audit their code for any suspicious changes and use static analysis tools to ensure that no malicious code is introduced during development.

3. **App Store Review Process**: Apple could have implemented additional checks to detect apps with suspicious behaviors during the app review process.

6.2 Simulated Mobile Security Attacks

To gain hands-on experience with mobile security, we simulated the following attacks in a controlled environment:

Attack 1: Man-in-the-Middle (MITM) Attack

Overview: A MITM attack occurs when an attacker intercepts and potentially alters the communication between two parties, such as between a mobile app and its server. This attack is typically carried out on insecure networks (e.g., public Wi-Fi).

Simulation:

1. **Tools Used**: We used **Burp Suite** and **Wireshark** to intercept traffic between the mobile app and its server.

2. **Attack Execution**: By setting up a **proxy** on a public network and configuring the mobile device to route traffic through the proxy, we intercepted HTTP traffic from the app. We then used Burp Suite to analyze and modify requests and responses between the app and the server.

3. **Mitigation**: To prevent MITM attacks, we configured the app to use **HTTPS** (SSL/TLS) for all communication, ensuring that traffic was encrypted. Additionally, we implemented **certificate pinning** to mitigate the risk of an attacker impersonating the server.

Attack 2: App Tampering (APK Modification)

Overview: App tampering occurs when an attacker modifies the APK file of a mobile app to inject malicious code or bypass certain security features, such as authentication.

Simulation:

1. **Tools Used**: We used **APKTool** to decompile the app's APK file and **Java Decompiler** to inspect its code.

2. **Attack Execution**: By modifying the app's code (e.g., disabling authentication checks), we recompiled the APK and installed it on the device.

3. **Mitigation**: To prevent app tampering, we implemented **code obfuscation** to make it harder for attackers to reverse-engineer the app. Additionally, **integrity checks** were added to verify the authenticity of the app's code at runtime.

6.3 Defensive Measures Implemented

For both the MITM attack and the app tampering scenario, the following defensive measures were implemented to enhance the security of the mobile app:

1. **Enforcing HTTPS (SSL/TLS)**: This ensures all communication between the app and its server is encrypted, preventing MITM attacks.

2. **Certificate Pinning**: This defense ensures that the app only communicates with trusted servers by validating their certificate fingerprints, which mitigates the risk of SSL/TLS hijacking.

3. **Code Obfuscation**: By obfuscating the app's code, we made it more difficult for attackers to reverse-engineer and modify the app.

4. **Integrity Checks**: We implemented runtime integrity checks to detect any modifications made to the app, such as changes in the APK or tampering with its code.

Through the research and simulation of real-world mobile security attacks, we gained a deeper understanding of the importance of secure coding practices and robust security measures in mobile app development. The real-world breaches such as **Stagefright** and **XCodeGhost** highlighted the significance of prompt patching, secure development environments, and user education in preventing mobile security breaches.

The simulated attacks, including **MITM** and **app tampering**, allowed us to practice identifying vulnerabilities and implementing defensive strategies. By securing communications, enforcing strong authentication, and using code obfuscation, we can effectively mitigate the risks posed by these common mobile security threats.

7.0 Section F: Hands-on Labs (Mandatory to Complete)

The primary objective of this section is to gain practical experience in mobile security by exploring Android hacking techniques and analyzing mobile malware. These labs are designed to enhance the understanding of vulnerabilities and security measures in mobile applications, providing real-world skills in identifying and mitigating threats.

7.1 TryHackMe Lab: Android Hacking 101

The **Android Hacking 101** lab on TryHackMe provides an introductory overview of Android application security. The lab focuses on identifying vulnerabilities in Android apps, exploiting those weaknesses, and understanding how attackers could compromise an app. The goal is to learn common Android application vulnerabilities, such as insecure data storage, improper implementation of WebView, and flaws in app permissions.

Lab Process:

1. **Setup**: The lab begins by guiding users through the process of setting up a virtual machine and installing Android-related tools, including **Android Studio** and **Burp Suite** for vulnerability scanning.

2. **Exploiting Vulnerabilities**: The lab provides several vulnerable Android applications to analyze and exploit. We worked through identifying various security flaws such as **insecure data storage**, **weak authentication**, and **insecure**

web traffic.

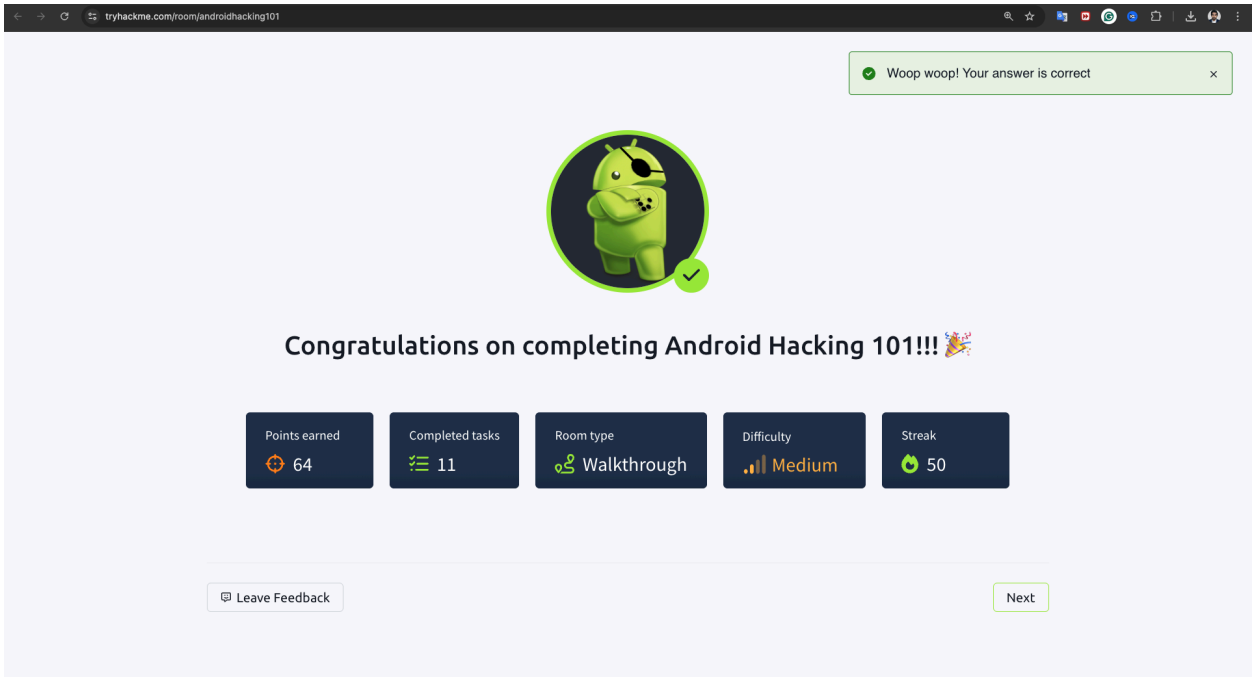
3. **Common Vulnerabilities Identified**:

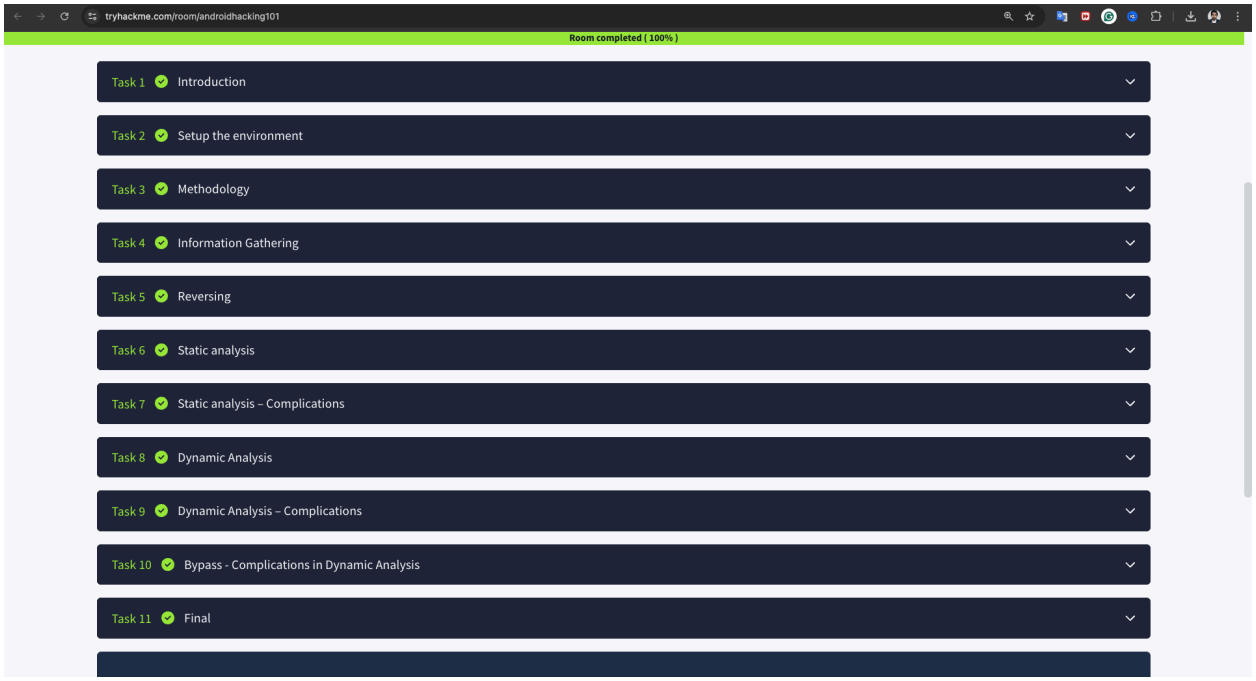
    - **Insecure Data Storage**: A sample app was found to store sensitive data (such as passwords) in **SQLite databases** without encryption.

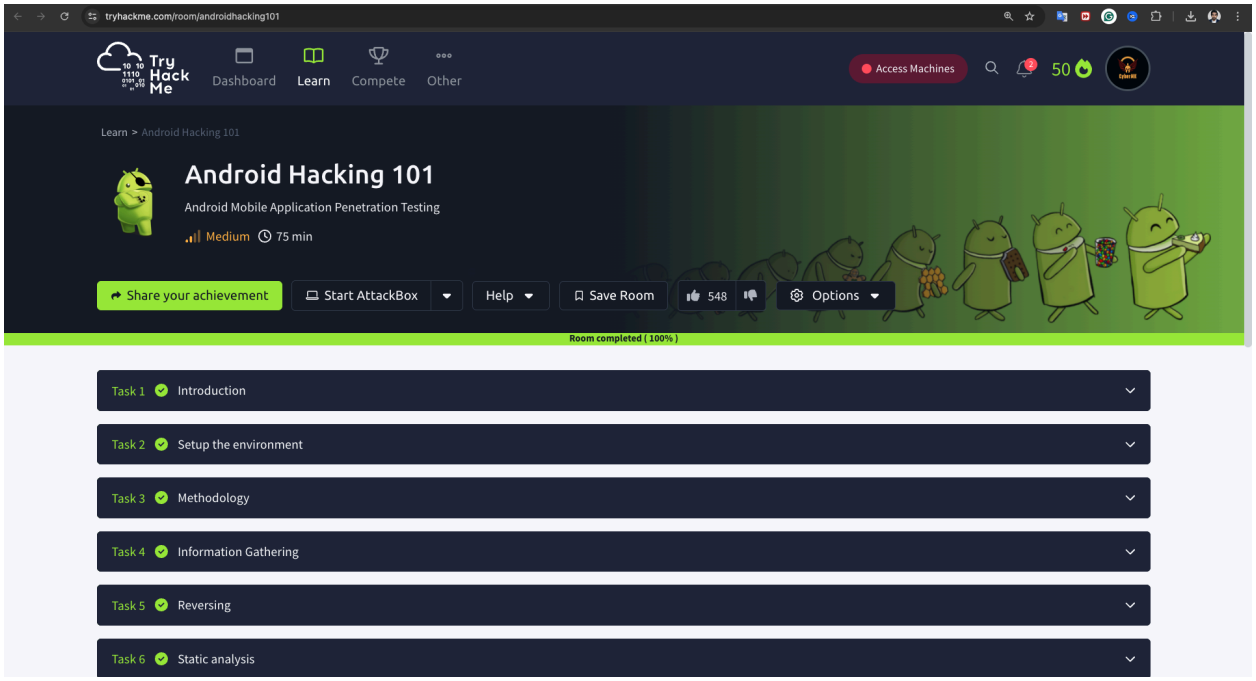
    - **Weak Authentication**: In the app's login mechanism, a weak or missing password hashing technique was identified.

    - **Insecure WebView**: The app used a WebView component without validating URLs, allowing potential **JavaScript injection** attacks.

Screenshots:




tryhackme.com/room/androidhacking101
Room completed (100%)
Task 1 Introduction
Task 2 Setup the environment
Task 3 Methodology
Task 4 Information Gathering
Task 5 Reversing
Task 6 Static analysis
Task 7 Static analysis – Complications
Task 8 Dynamic Analysis
Task 9 Dynamic Analysis – Complications
Task 10 Bypass - Complications in Dynamic Analysis
Task 11 Final


tryhackme.com/room/androidhacking101
Try Hack Me
Dashboard
Learn
Compete
Other
Access Machines
50
Learn > Android Hacking 101
Android Hacking 101
Android Mobile Application Penetration Testing
Medium
75 min
Share your achievement
Start AttackBox
Help
Save Room
548
Options
Room completed (100%)
Task 1 Introduction
Task 2 Setup the environment
Task 3 Methodology
Task 4 Information Gathering
Task 5 Reversing
Task 6 Static analysis

7.2 TryHackMe Lab: Mobile Malware Analysis

The **Mobile Malware Analysis** lab on TryHackMe focuses on analyzing mobile malware samples to understand their behavior and impact on mobile security. The goal is to familiarize oneself with the techniques used to detect and analyze mobile malware, reverse engineer malicious apps, and learn methods of preventing future attacks.

Lab Process:

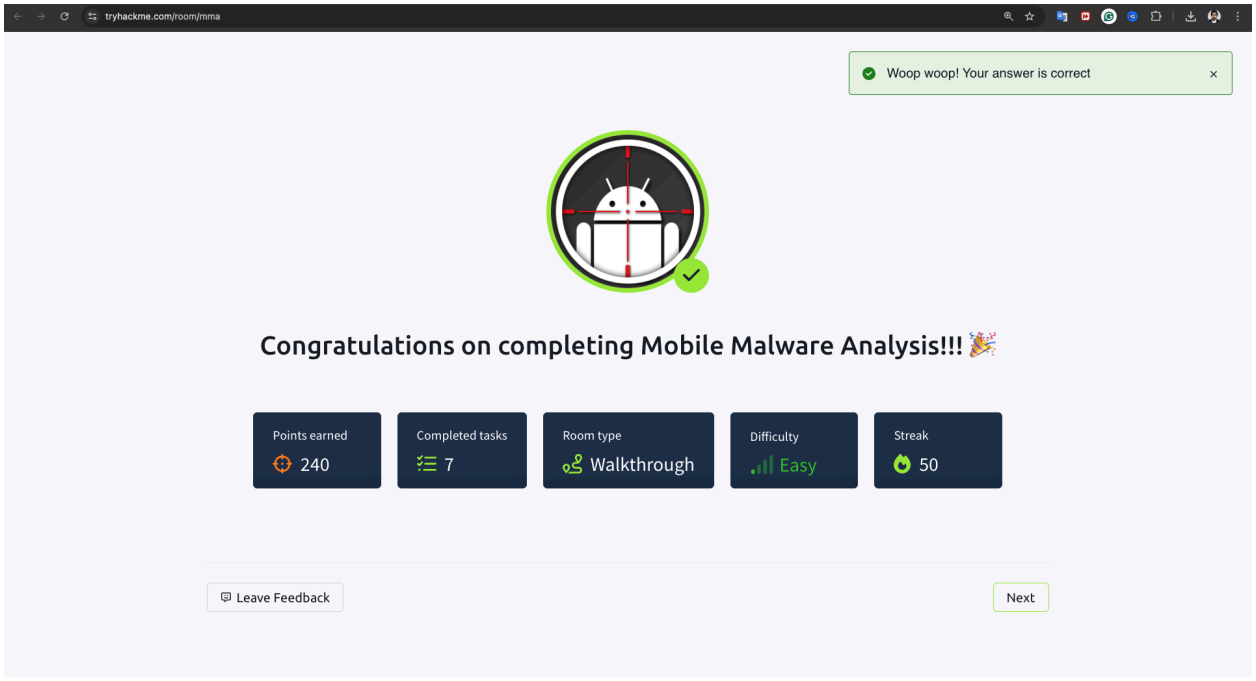
1. **Malware Sample Analysis**: The lab provides various **malware samples** that are pre-packaged in APK format. The malware samples exhibited behaviors such as **data exfiltration**, **SMS-based fraud**, and **ransomware attacks**.

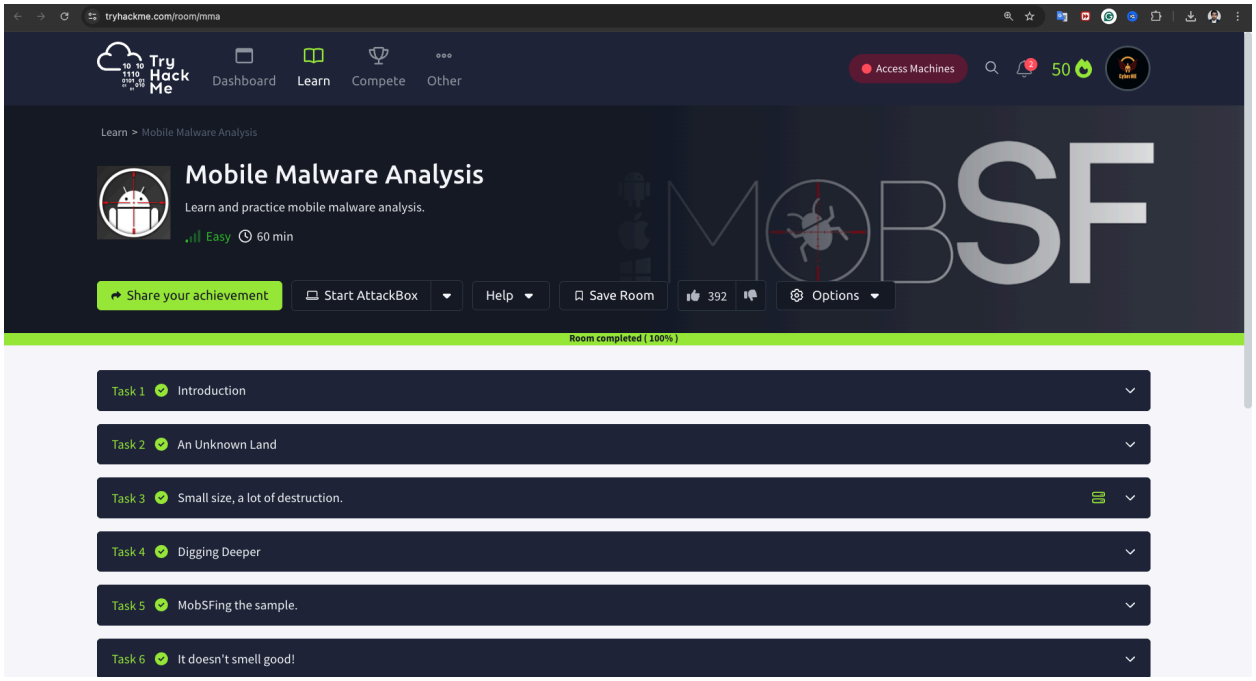
2. **Reverse Engineering**: Using tools like **APKTool** and **Jadx** (Java decompiler), we decompiled the APK files to analyze the underlying code. We identified that the malware was using **obfuscation techniques** to avoid detection, such as encrypted communication and randomizing function names.

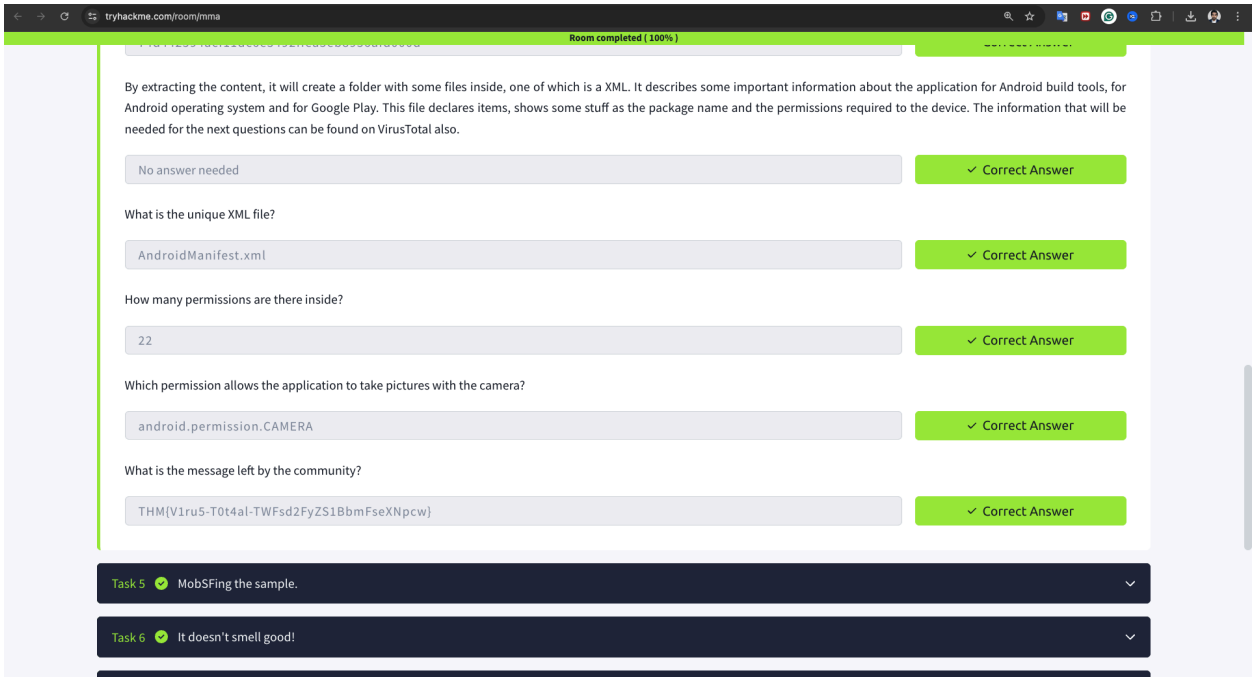
3. **Behavioral Analysis**: We ran the malware samples in a controlled environment (emulator) and observed their activities using tools like **Wireshark** and **Fiddler** to capture network traffic. The malware exhibited behaviors such as sending sensitive data to external servers and attempting to exploit known vulnerabilities in the Android OS.

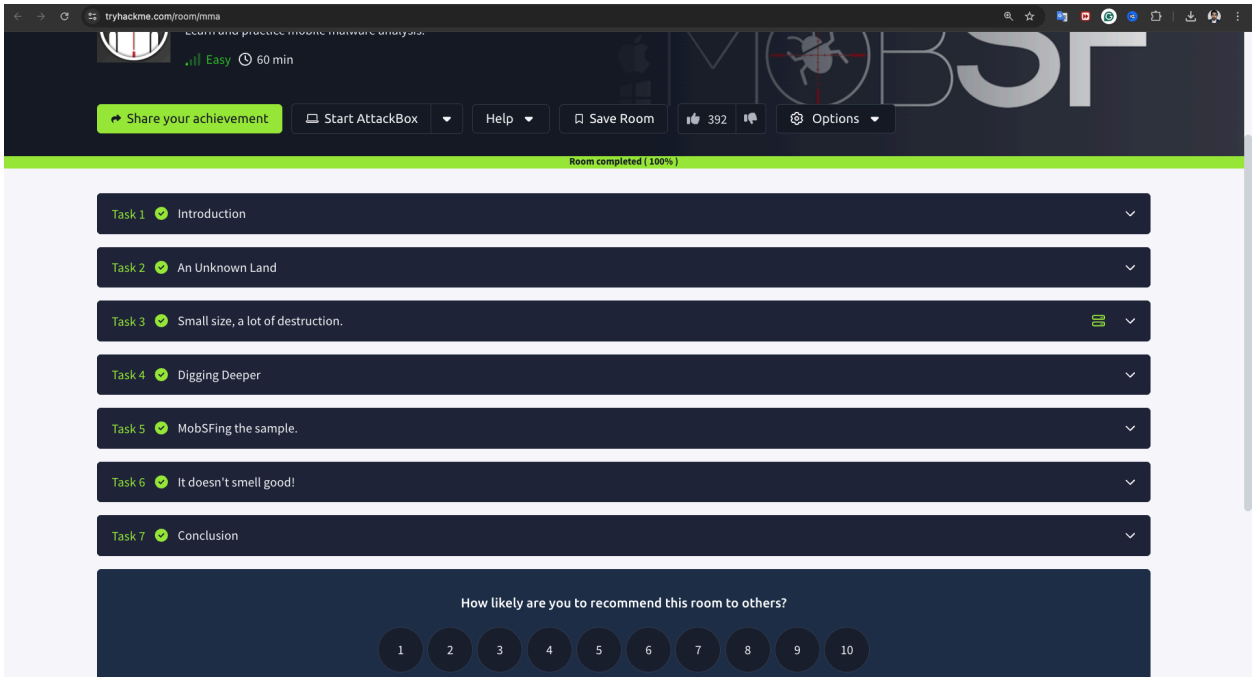
4. **Detection Techniques**: Various tools were used to detect malware signatures, such as **ClamAV** and **VirusTotal**. The lab also emphasized **static and dynamic analysis** techniques, where static analysis focuses on examining the app's code, while dynamic analysis monitors the app's behavior in real time.

Screenshots:


tryhackme.com/room/mma
Woop woop! Your answer is correct
Congratulations on completing Mobile Malware Analysis!!!
Points earned
240
Completed tasks
7
Room type
Walkthrough
Difficulty
Easy
Streak
50
Leave Feedback
Next


tryhackme.com/room/mma
Try Hack Me
Dashboard
Learn
Compete
Other
Access Machines
50
Learn > Mobile Malware Analysis
Mobile Malware Analysis
Learn and practice mobile malware analysis.
Easy
60 min
MOBSF
Share your achievement
Start AttackBox
Help
Save Room
392
Options
Room completed (100%)
Task 1 Introduction
Task 2 An Unknown Land
Task 3 Small size, a lot of destruction.
Task 4 Digging Deeper
Task 5 MobSFing the sample.
Task 6 It doesn't smell good!


tryhackme.com/room/mma
Room completed (100%)
By extracting the content, it will create a folder with some files inside, one of which is a XML. It describes some important information about the application for Android build tools, for Android operating system and for Google Play. This file declares items, shows some stuff as the package name and the permissions required to the device. The information that will be needed for the next questions can be found on VirusTotal also.
No answer needed
✓ Correct Answer
What is the unique XML file?
AndroidManifest.xml
✓ Correct Answer
How many permissions are there inside?
22
✓ Correct Answer
Which permission allows the application to take pictures with the camera?
android.permission.CAMERA
✓ Correct Answer
What is the message left by the community?
THM{V1ru5-T0t4al-TWFsd2FyZS1BbmFseXNpcw}
✓ Correct Answer
Task 5 MobSFing the sample.
Task 6 It doesn't smell good!


tryhackme.com/room/mma
Easy 60 min
Share your achievement
Start AttackBox
Help
Save Room
392
Options
Room completed (100%)
Task 1 Introduction
Task 2 An Unknown Land
Task 3 Small size, a lot of destruction.
Task 4 Digging Deeper
Task 5 MobSFing the sample.
Task 6 It doesn't smell good!
Task 7 Conclusion
How likely are you to recommend this room to others?
1 2 3 4 5 6 7 8 9 10

The **Android Hacking 101** and **Mobile Malware Analysis** labs provided invaluable hands-on experience in analyzing vulnerabilities and understanding mobile malware behaviors. These labs strengthened our ability to:

- Identify common security weaknesses in Android apps.
- Use penetration testing tools like **Burp Suite** to intercept and analyze mobile app traffic.
- Reverse engineer and analyze mobile malware to understand its behavior and impact on users.
- Detect malicious activity through static and dynamic analysis methods.

By completing these labs, we gained practical insights into securing mobile applications, recognizing common attack vectors, and implementing detection and mitigation strategies.

EncryptEdge Labs

This Internship Task report was developed on [April, 21, 2025]

By:

atalmamun@gmail.com