Mobile App Security Tutorial

Isabella Sunderman

# Static Analysis

1.1 Objectives:

The purpose of this lab is to introduce static analysis of Android and iOS applications using the Mobile Security Framework (MobSF). The tutorial users will learn how to set a containerized environment, analyze sample IPA and APK files and interpret findings of potential vulnerabilities, API usage and dangerous permissions.

* 1. Environment Setup

Environment setup instructions are compatible with Linux OS and Windows OS with commands and tools required being the same.

1.2.1 Docker

MobSF requires Python 3.13 or newer, using Docker is recommended to avoid compatibility issues found in Linux OS

For Windows just follow the steps provided on the Docker website:

<https://docs.docker.com/get-started/get-docker/>

For Linux Docker commands:



* + 1. Running MobSF

Once Docker installed running the following commands in the terminal:



With the application running open a browser

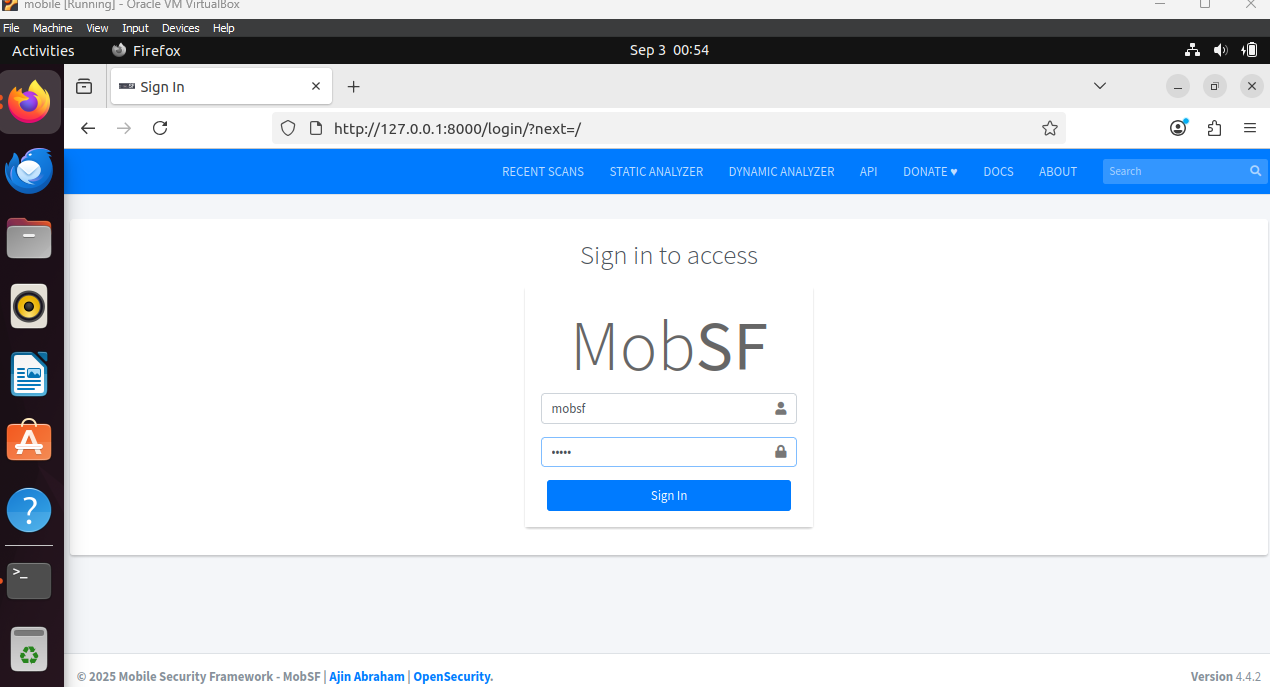
Linux: http:// 127.0.0.1:8000

Windows: <http://localhost:8000>

Default login credentials are

Username: mobsf

Password: mobsf



1.3 Static analysis

There are two application files provided with the tutorial and IPA and APK. These and sample applications for this tutorial and both files will be analyzed in this portion.

1.3.1 Activity 1 iOS IPA report

Upload the IPA file into the MobSF and generate the report. The key findings of the report in that the application is a helloworld and grade is 100/100 and low risk. ThIPA file although outdated, can be observed to not have any vulnerabilities.

The generated reported includes File, App and Binary information, App Transport Security, IPA Binary Code Analysis, Domain Malware check and Scan Logs. Each section provides new context op the ipa file and the test conducted on the file.

1. 3.2 Challenge 1

This Challenge will be performing an analysis on APK file.

* Upload the APK file to MobSF
* Record the reported if there are any detected vulnerabilities
* Identify differences in the generated reports

1: Locate three dangerous permissions requested by the Android app. Explain why these are risky.

2: Search the APK report for hardcoded API keys, tokens, or secrets. Explain why this practice is insecure.

3: Based on the APK and IPA reports, which platform’s sample app shows a higher attack surface? Justify your reasoning with report evidence.

# Dynamic Analysis

2.1 Objectives:

The goal of this portion of the tutorial is to perform a dynamic analysis of an Android application by intercepting and modifying their behavior during runtime using Frida. This lab will require setting up and running an Android emulator, deploying Frida to capture data and bypass protectentions.

2.2 Environment Setup:

2.2.1 Install Frida

Frida requires python 3 and Node.js. the following link can but used to for download: <https://nodejs.org/en/download>

In terminal run the following commands to install Frida:



You also need to install the server binary to be pushed to the Android client device appropriate to the Frida version installed using the following link <https://github.com/frida/frida/releases>

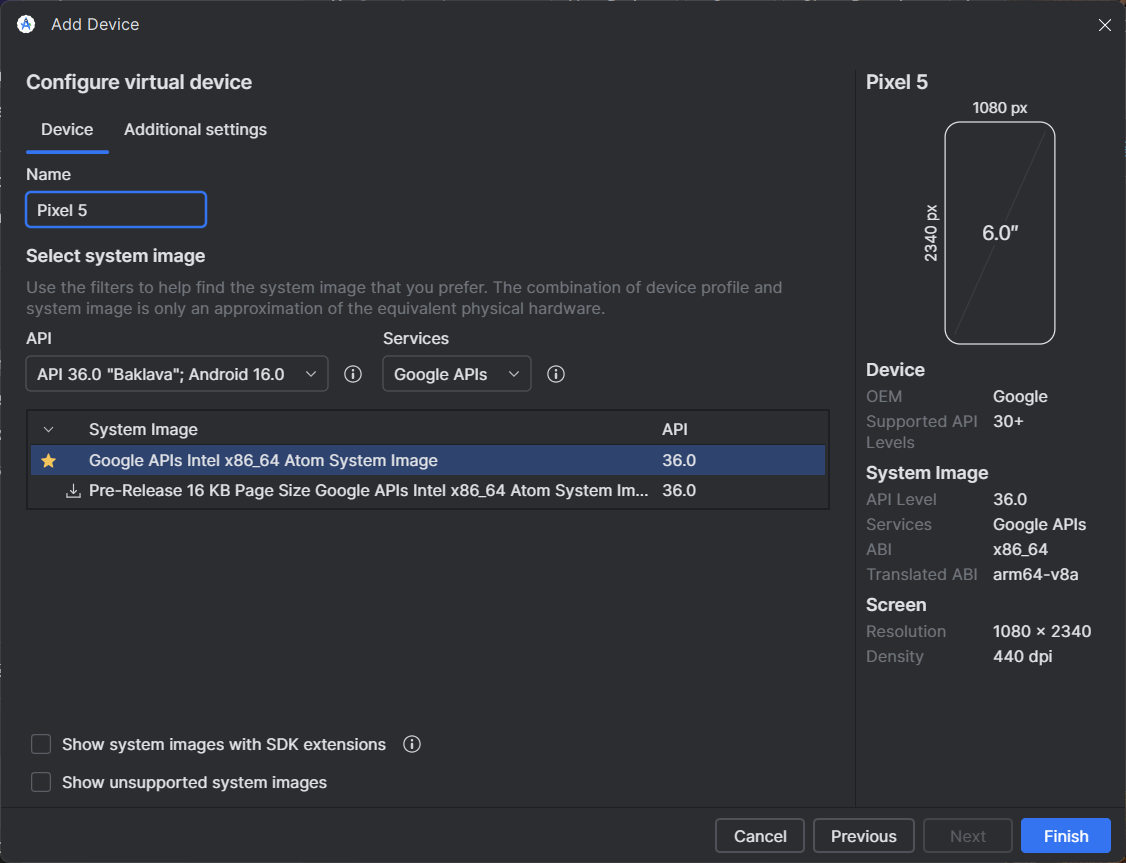


used in tutorial: frida-server-17.3.0-android-x86\_64

2.2.2 Install Android Studio

Android Studio will be used for compiling the application and an android emulated device for the Frida server client.

Download link: <https://developer.android.com/studio>

****

Start Android studio and navigate to the device manager and create a new android device. For this tutorial I made a Pixel 5 and used the default settings and turn the device on the device needs to be running to install the server binary.

Using terminal note the data path addresses of the Android studio and the server binary or execute the following commands in the android studio directory:



\*note the full file paths for your adb.exe file and Frida server binary

A screenshot of a computer program

AI-generated content may be incorrect.

Open a new terminal verify the connection is established between your Frida client and the server on the android device by listing all the processes on the android device run the command:

Frida-ps -U

A screenshot of a computer

AI-generated content may be incorrect.

2.3 Dynamic Analysis

2.3.1 Now that the server is running on the android device and connection is set up we can explore Frida’s capabilities and some of the commands:

* Frida-discover: finds internal functions of the program
* Frida-kill: end processes using PID’s (Frida-ps will list the PID and process)
* Frida-ls-devices: lists devices
* Frida-trace: trace functions calls dynamically, and very versatile in capabilities of launching applications and tracing java methods, API, certificates. (Frida-trace -h is a full list of options)

2.3.2 Activity 2: This required some more set up, using the proved diva-android application. This will also require the full file path of the APK file should be something like:

"C:\...\Downloads\ApiDemos-debug.apk"

Install the APK and run the app

Adb install -r <file path of the app-debug.apk>

Adb shell am start -W -n io appium.andriod.apis/.ApiDemos

A screenshot of a computer program

AI-generated content may be incorrect.

Using the Frida –discover command find the functions that API Demos uses. How many threads does the command follow?

2.3.3 Challenge 2: After finding some of the functions try tracing a function that API Demos uses. A link to a web GUI can appear in the terminal that give some interesting insight of how the applications functions calls.

# Appendix

Challenge 1:

1. APK requested ACCESS\_FINE\_LOCATION, WRITE\_EXTERNAL\_STORAGE, READ\_SMS. These are considered dangerous because they can expose sensitive user data (GPS, files, SMS content).

2. MobSF flagged string values in the APK resources containing API tokens. Developers should store them server-side, not inside the app.

3. The APK presented more risks (permissions, potential secrets). The IPA was low risk but outdated. Android apps are generally more exposed to reverse engineering due to APK packaging.

Challenge 2: can trace any function command format would be:

frida-trace -U -p 3798 -i "open" -i "read" -i "write" -i "connect" -i "send" -i "recv" -i "SSL\_write" -i "SSL\_read"

# References

<https://frida.re/docs/home>

<https://mobsf.github.io/docs/#>