INVESTIGATION OF THE FEATURES OF MOBSF

CSE 406 PROJECT REPORT

Abdullah Al Fahad
Student ID: 1805033

Siam Al Mujadded
Student ID: 1805051

Bangladesh University of Engineering and Technology

August 2023

Contents

1 Overview of MobSF

The Mobile Security Framework (MobSF) stands as a multi-faceted guardian for mobile applications, embracing Android, iOS, and Windows platforms. At its core, MobSF boasts a plethora of features designed to holistically safeguard mobile apps throughout their lifecycle.

MobSF operates on a dual analysis approach, encompassing both static and dynamic evaluations. Through static analysis, it scrutinizes the app's code and structure before execution, unveiling potential vulnerabilities lying beneath the surface. On the other hand, dynamic analysis watches the app in action during runtime, uncovering vulnerabilities that might manifest only during real-world usage scenarios.

Beyond its versatile analysis techniques, MobSF exhibits remarkable flexibility in accommodating app binaries, including APK, XAPK, IPA, and APPX formats. Additionally, it delves into zipped source code, fostering a comprehensive assessment of the app's inner workings.

A distinctive trait of MobSF is its prowess in malware detection and analysis. This functionality actively identifies and addresses malicious components within the app, ensuring that users' devices and data remain uncompromised.

MobSF's integration capabilities are nothing short of remarkable. It seamlessly integrates into Continuous Integration/Continuous Deployment (CI/CD) pipelines and DevSecOps workflows, weaving security into the fabric of the development process. This integration is facilitated through REST APIs, which streamline the automation of security assessments.

MobSF becomes a really good friend to the people who make apps. It works smoothly with the steps they follow to create apps and offers many ways to check for problems. This makes MobSF a big helper in the ongoing fight to keep apps safe and strong on phones and tablets, even when things are always changing in the digital world.

2 High-Level Overview of the Source Code

The codes are available at: github the code is written in Python using Django framework. The code is divided into several files and folders. The folders are:

- mobsf
- StaticAnalyzer
- DynamicAnalyzer
- MalwareAnalyzer

2.1 main.py

This is the main file of the project. It contains the main function. It is the entry point of the project. It contains the following functions: Here a main function start the server at '127.0.0.1:8000'

```python
def main():
    if len(sys.argv) == 2:
        listen = sys.argv[1]
    else:
        listen = '127.0.0.1:8000'
    if platform.system() != 'Windows':
        sys.argv = [
            '',
            '-b',
            listen,
            'mobsf.MobSF.wsgi:application',
            '--workers=1',
            '--threads=10',
            '--timeout=3600',
        ]
        from gunicorn.app.wsgiapp import run
        run()
    else:
        from waitress import serve
        from .MobSF import wsgi
        serve(
            wsgi.application,
            listen=listen,
            threads=10,
            channel_timeout=3600)
```

Figure 1: main function at `main.py` file

2.2 mobsf folder

This folder contains the main app of the project. Some important files are:

`init.py` Initialization module and create directory

`settings.py` configuration for this Django project, allowable file types like apk, ipa, jar, aar etc

`urls.py` URL routing

2.3 StaticAnalyzer folder

This folder contains the static analysis app of the project Here we can find some tools which are used for static analysis. Those tools are:

`Apktool` A tool for reverse engineering Android apk files

`baksmali` smali/baksmali is an assembler/disassembler for the dex format used by dalvik, Android's Java VM implementation

`vd2svg` Android vector drawable to SVG converter

`batik` a utility that can convert SVG files to a raster format

`jadx` a decompiler for the Java programming language

2.4 DynamicAnalyzer folder

This folder contains the dynamic analysis app of the project. It contains the following tools:

`Xposed` The Xposed framework is a hack for rooted android phones. It allows the user to replace any JAR file in the system

`Frida` a dynamic code instrumentation toolkit

2.5 MalwareAnalyzer folder

This folder only has some view files. One of them uses virustotal.

3 How To Run *MobSF*

MobSF can be installed using docker.

1. Install docker
2. Pull the docker image

3. Run the docker image
4. Access the web interface
5. Upload the file
6. Analyze the file
7. View the report
8. Stop the docker image

3.1 Install Docker

Follow these links to install docker on your machine:

- Windows
- Mac
- Ubuntu

3.2 Pull the docker image

The docker image can be pulled using the following command:

```
docker pull opensecurity/mobile-security-framework-mobsf
```

It is recomended to pull the latest version of the docker image.

3.3 Run the docker image

The docker image can be run using the following command:

```
docker run -it -p 8000:8000 opensecurity/mobile-security-framework-mobsf
```

The docker image can be run in the background using the following command:

```
docker run -it -d -p 8000:8000 opensecurity/mobile-security-framework-mobsf
```

Here we can see two port numbers. The first one is the port number of the host machine and the second one is the port number of the docker image. The port number of the host machine can be changed to any port number. The port number of the docker image should be 8000. The port number of the host machine can be changed to 4200 using the following command:

```
docker run -it -d -p 4200:8000 opensecurity/mobile-security-framework-mobsf
```

If evrything is ok then image like 2 will be shown.

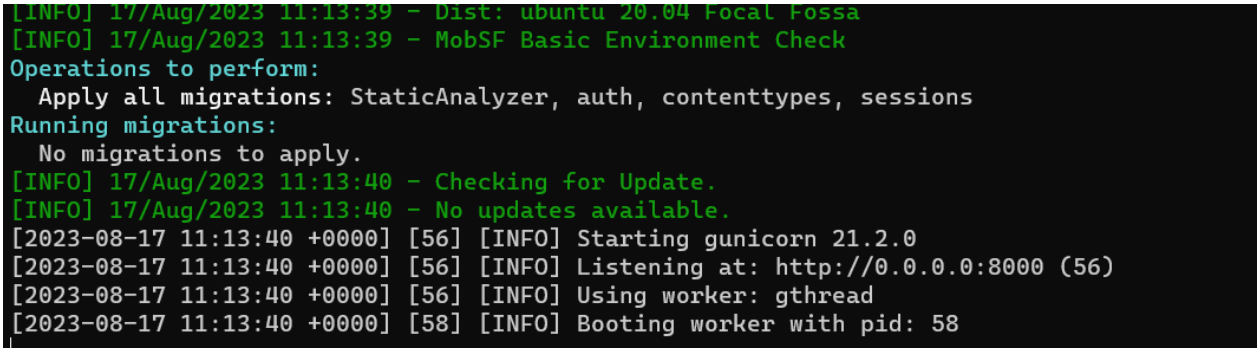


Figure 2: web interface

3.3.1 some issues can arise

1. docker daemon is not running

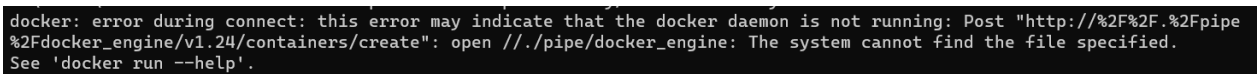


Figure 3: docker daemon is not running

Solution : start docker daemon, on windows run docker desktop

2. Port is not available

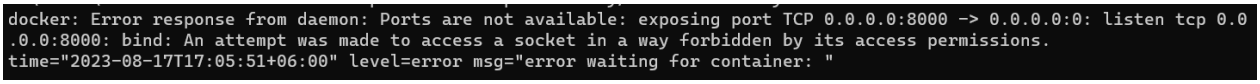


Figure 4: Port is not available

Solution : change the port number of the host machine to any available port number refer to Run the docker image section

3.4 Access the web interface

The web interface can be accessed using the following link:

```
http://localhost:<port number of the host machine>
```

The web interface can be accessed using the following link if the port number of the host machine is changed to 4200:

```
http://localhost:4200
```

The web interface is shown in figure 5.

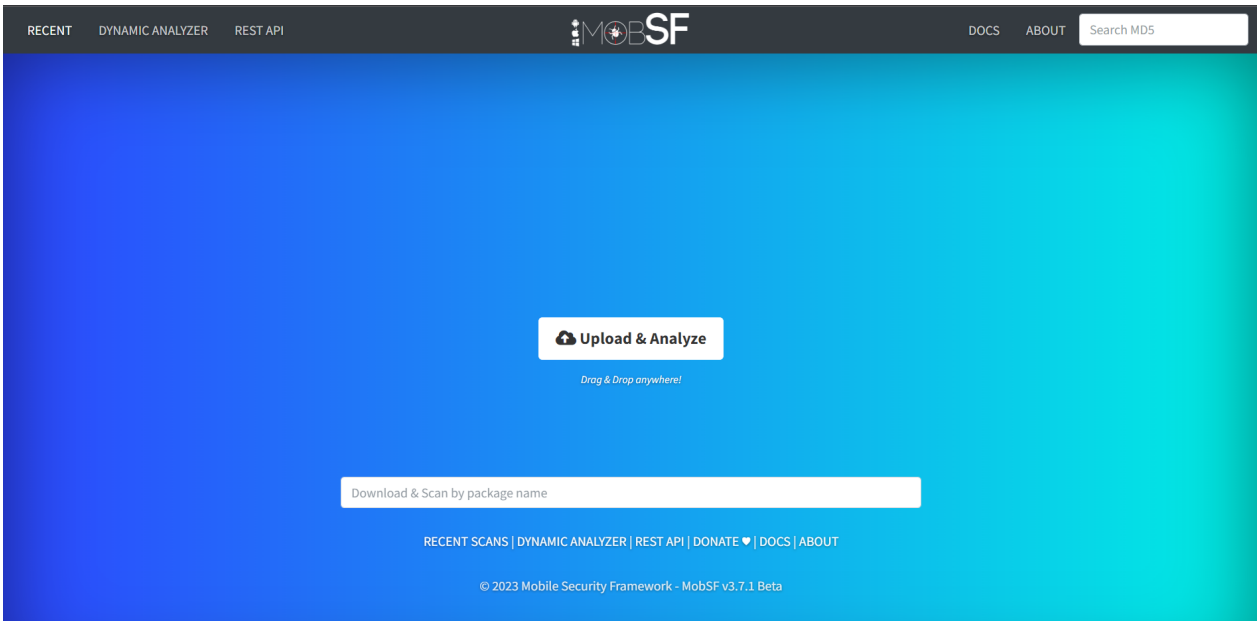


Figure 5: web interface

4 Features of MobSF

All-in-One Solution: MobSF offers a comprehensive package for mobile app security, combining automated pen-testing, malware analysis, and security assessments in a single framework.

Platform Support: It caters to Android, iOS, and Windows platforms, making it versatile for testing apps across different mobile ecosystems.

Static and Dynamic Analysis: MobSF examines apps in two ways. Static analysis looks at the app's code and structure for vulnerabilities before running, while dynamic analysis observes how the app behaves during runtime.

App Binary and Source Code Support: It analyzes various app file formats, including APK, XAPK, IPA, APPX, and even zipped source code, ensuring a thorough assessment of the app's security.

Malware Detection and Analysis: MobSF identifies malicious components within apps, protecting devices and user data from potential threats.

Integration: It seamlessly integrates with CI/CD pipelines and DevSecOps workflows, providing automated security assessment throughout the development and deployment process.

REST API Support: MobSF offers REST APIs, making it easy to integrate into existing development processes and automate security checks.

Dynamic Analyzer: This feature watches the app as it runs, uncovering vulnerabilities that may only appear during actual usage, improving the accuracy of assessments.

User-Friendly Interface: Despite its complexity, MobSF provides an accessible interface that simplifies configuring tests, running analyses, and interpreting results.

Holistic Security Assessment: MobSF combines these features to provide a comprehensive view of app security, from design to deployment, ensuring a robust defense against threats.

Integration with CI/CD and DevSecOps: MobSF seamlessly fits into the development and deployment pipelines, automating security checks.

User-Friendly Interface: Despite its powerful capabilities, MobSF maintains an easy-to-use interface, making it accessible to a wide range of users.

Real-time Monitoring: The Dynamic Analyzer watches the app while it's being used, like a bodyguard, to catch any issues as they happen.

Comprehensive Protection: MobSF looks both inside and outside the app, ensuring that it's strong against different types of security problems.

5 Static Analysis

Here are the features of static analysis of Mobsf. Static analysis was carried out for the app named AndroGoat.apk.

Information about the app: The "Information" feature in MobSF's static analysis process provides a valuable snapshot of the mobile application's security and attributes. It presents crucial metrics like tracker detection count, offering insights into potential privacy concerns. The security score sheds light on the app's overall security health. This feature also delves into detailed file information, highlighting potential risks within the app's components. Moreover, it offers a comprehensive overview of the app, including metadata, permissions, and intent filters, aiding in understanding its behavior and potential interaction pathways.

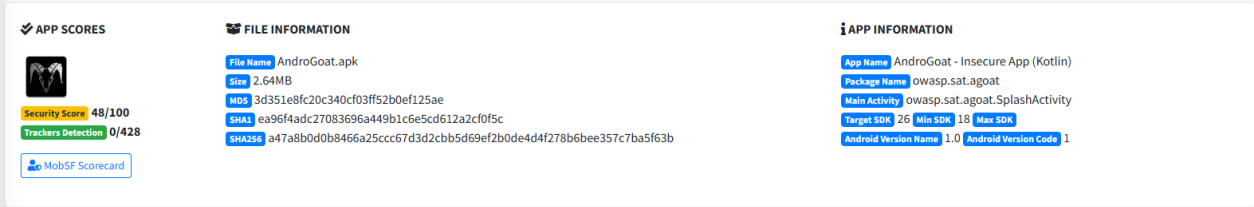


Figure 6: information of the app

Scan options: The "Scan Options" feature in MobSF empowers users with dynamic control over the security assessment process. Within this feature, several important options are available. "Rescan" enables users to reevaluate the app's security based on new information or modifications. "Manage Suppressions" allows for the handling of specific findings that might be false positives or intentionally ignored. Additionally, "Start Dynamic Analysis" initiates the dynamic assessment of the app during runtime.

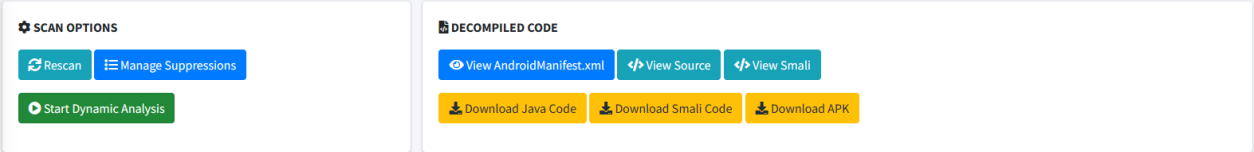


Figure 7: scan option

Signer Certificate: The "Signer Certificate" feature in MobSF is a crucial component that offers insights into the authenticity and trustworthiness of the APK. By presenting information about the signer certificate, including details about the digital signature used to verify the app's source and integrity, this feature ensures that the app hasn't been tampered with or compromised. This information provides an added layer of assurance to users and developers, reassuring them that the app comes from a legitimate source and hasn't undergone any unauthorized modifications.

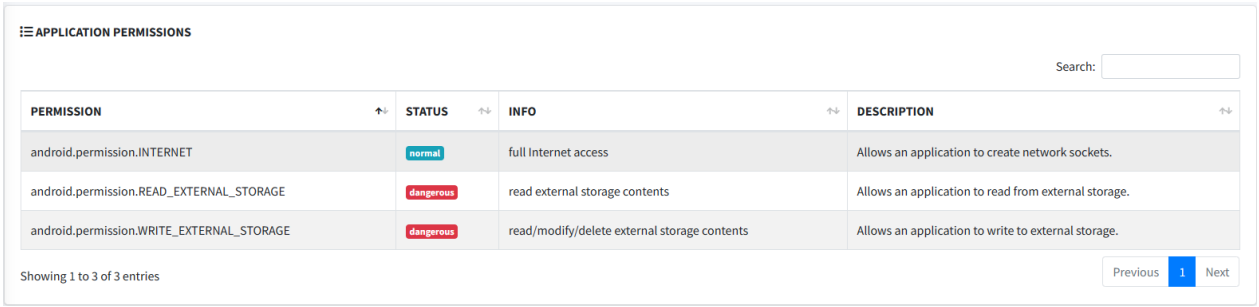
Application permissions: The "Application Permissions" feature within MobSF offers a comprehensive breakdown of the permissions requested by the app, shedding light on its interactions with device resources. Each permission, such as "android.permission.INTERNET" or "android.permission.READ_EXTERNAL_STORAGE," is detailed, providing information on whether it falls under categories like "normal" or "dangerous." This classification informs users about the potential implications of granting these permissions. Furthermore, the feature highlights the extent of access that each permission entails, giving users a clear understanding of what the app can do with specific resources. The included descriptions help bridge the gap between technical terms and user comprehension, empowering individuals to make informed decisions regarding app permissions.

SIGNER CERTIFICATE

```
APK is signed
v1 signature: True
v2 signature: True
v3 signature: False
Found 1 unique certificates
Subject: CN=Android Debug, O=Android, C=US
Signature Algorithm: rsassa_pkcs1v15
Valid From: 2016-09-22 18:39:24+00:00
Valid To: 2046-09-15 18:39:24+00:00
Issuer: CN=Android Debug, O=Android, C=US
Serial Number: 0x1
Hash Algorithm: sha1
md5: af5ff4235e641bae76eb04d55c2eb670
sha1: afbffdbd437c68395723c82190a5ea8a2904c2af
sha256: f1d42f18d738ff2908cc2274a688b6642df0379a0ea2623164e969a9086e0e20
sha512: 4eb8611881cbea7804116388aa5ec677d2054a189abf12ca3badbc2d204770af0ebcfb057f1f96e53e7edb9103c47c21695330ea2b8e02b795429eb8c1a955a7
PublicKey Algorithm: rsa
Bit Size: 1024
Fingerprint: c15e372e3066fe2dd58eef3d86a2c7b6e24647bb6522915ce2fba6c57280b33b
```

Figure 8: signer certificate

By offering transparency into the permissions landscape, this feature enhances user awareness and contributes to a more secure mobile environment where users can confidently assess and manage app access to their device's functionalities.



APPLICATION PERMISSIONS

Search:

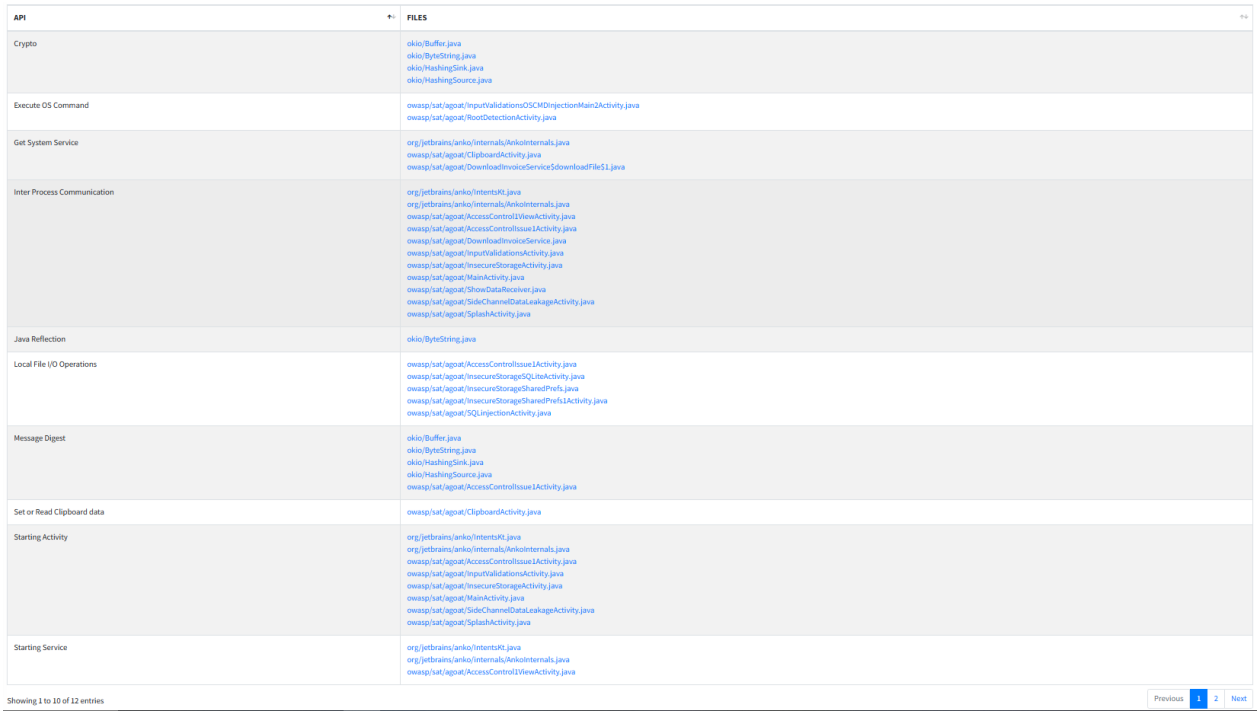
PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.
android.permission.READ_EXTERNAL_STORAGE	dangerous	read external storage contents	Allows an application to read from external storage.
android.permission.WRITE_EXTERNAL_STORAGE	dangerous	read/modify/delete external storage contents	Allows an application to write to external storage.

Showing 1 to 3 of 3 entries

Previous 1 Next

Figure 9: application permissions

Android API: The "Android API" feature in MobSF presents a clear overview of the interaction between the mobile app and the Android API. Displayed in a user-friendly table format, it showcases two essential columns: "API" and "Files." The "API" column lists the specific Android functions, known as APIs, that the app utilizes. On the other hand, the "Files" column reveals the files within the app's code that make use of each corresponding API. This breakdown is immensely valuable as it offers insights into how the app relies on different APIs and the specific parts of the code that engage with them. By mapping out this relationship, the "Android API Details" feature aids developers in comprehending the app's functionalities and dependencies, contributing to a more comprehensive understanding of the app's behavior and potential security implications.



API	FILES
Crypto	okio/Buffer.java okio/ByteString.java okio/HashingSink.java okio/HashingSource.java
Execute OS Command	owasp/sat/agoat/InputValidationsOSCMDInjectionMain2Activity.java owasp/sat/agoat/RootDetectionActivity.java
Get System Service	org/jetbrains/anko/internals/AnkoInternals.java owasp/sat/agoat/ClipboardActivity.java owasp/sat/agoat/DownloadInvoiceService$downloadFile$1.java
Inter Process Communication	org/jetbrains/anko/IntentsKt.java org/jetbrains/anko/internals/AnkoInternals.java owasp/sat/agoat/AccessControl1ViewActivity.java owasp/sat/agoat/AccessControlIssue1Activity.java owasp/sat/agoat/DownloadInvoiceService.java owasp/sat/agoat/InputValidationsActivity.java owasp/sat/agoat/InsecureStorageActivity.java owasp/sat/agoat/MainActivity.java owasp/sat/agoat/ShowDataReceiver.java owasp/sat/agoat/SideChannelDataLeakageActivity.java owasp/sat/agoat/SplashActivity.java
Java Reflection	okio/ByteString.java
Local File I/O Operations	owasp/sat/agoat/AccessControlIssue1Activity.java owasp/sat/agoat/InsecureStorageSQLiteActivity.java owasp/sat/agoat/InsecureStorageSharedPrefs.java owasp/sat/agoat/InsecureStorageSharedPrefs1Activity.java owasp/sat/agoat/SQLInjectionActivity.java
Message Digest	okio/Buffer.java okio/ByteString.java okio/HashingSink.java okio/HashingSource.java owasp/sat/agoat/AccessControlIssue1Activity.java
Set or Read Clipboard data	owasp/sat/agoat/ClipboardActivity.java
Starting Activity	org/jetbrains/anko/IntentsKt.java org/jetbrains/anko/internals/AnkoInternals.java owasp/sat/agoat/AccessControlIssue1Activity.java owasp/sat/agoat/InputValidationsActivity.java owasp/sat/agoat/InsecureStorageActivity.java owasp/sat/agoat/MainActivity.java owasp/sat/agoat/SideChannelDataLeakageActivity.java owasp/sat/agoat/SplashActivity.java
Starting Service	org/jetbrains/anko/IntentsKt.java org/jetbrains/anko/internals/AnkoInternals.java owasp/sat/agoat/AccessControl1ViewActivity.java

Showing 1 to 10 of 12 entries

Previous 1 2 Next

Figure 10: android api

Browsable Activities: The "Browsable Activities" feature within MobSF provides a comprehensive view of the activities within a mobile app that are set as "browsable." These activities are essentially entry points or screens that can be accessed by other apps or components. MobSF highlights these activities, offering valuable insights into which parts of the app can be launched from external sources. This feature is particularly useful in identifying potential security risks, as certain activities being browsable might unintentionally expose sensitive functionalities to unauthorized parties.

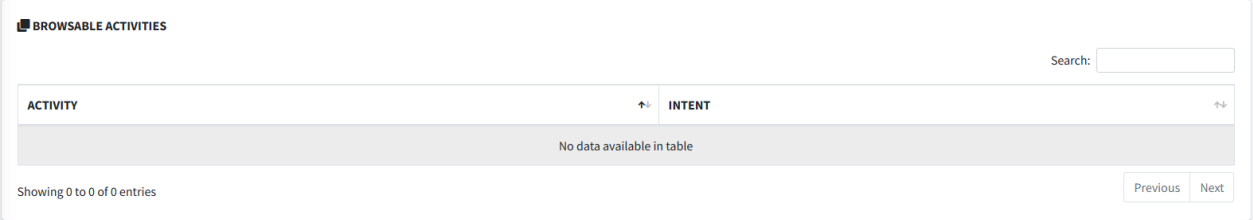


Figure 11: browsable activities

5.1 Security Analysis

Network Security: The "Network Security" feature in MobSF plays a pivotal role in evaluating the security of network-related aspects within a mobile app. It categorizes findings into three distinct groups: "High," "Warning," and "Info," providing an efficient way to prioritize security concerns. The counts associated with each category give a quick overview of the severity and prevalence of network-related issues. This feature furnishes a detailed table that captures critical information about these findings. The table includes columns such as "Scope," "Severity," and "Description." In the "Scope" column, it outlines the specific context in which the network security issue arises. The "Severity" column highlights the potential impact of the issue on the app's security. The "Description" column provides a clear explanation of the problem, aiding developers and security experts in understanding the nature of the threat.

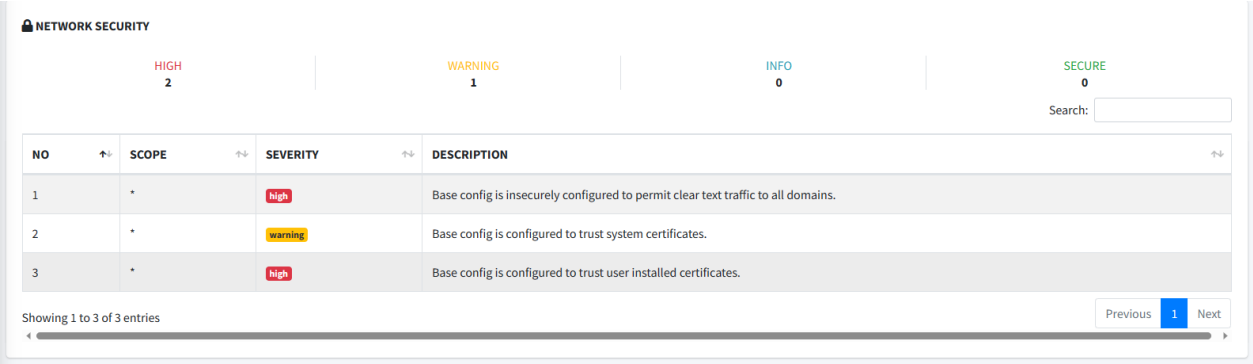


Figure 12: network security

Certificate Analysis: The "Certificate Analysis" feature within MobSF is a critical component that focuses on evaluating the security of certificates used by a mobile app. This feature classifies its findings into distinct categories: "High," "Warning," "Info," and "Secure," providing a clear understanding of the severity of certificate-related issues. To offer a concise overview, the feature includes count values for each category, allowing users to quickly assess the prevalence and severity

of these certificate issues. In addition, the "Certificate Analysis" feature presents a detailed table that comprises essential columns: "Title," "Severity," and "Description." In the "Title" column, it succinctly captures the essence of the certificate-related concern. The "Severity" column offers insights into the potential impact of the issue on the app's security. The "Description" column provides a comprehensive explanation of the problem, guiding developers and security professionals in comprehending the nature and implications of the certificate-related vulnerability.

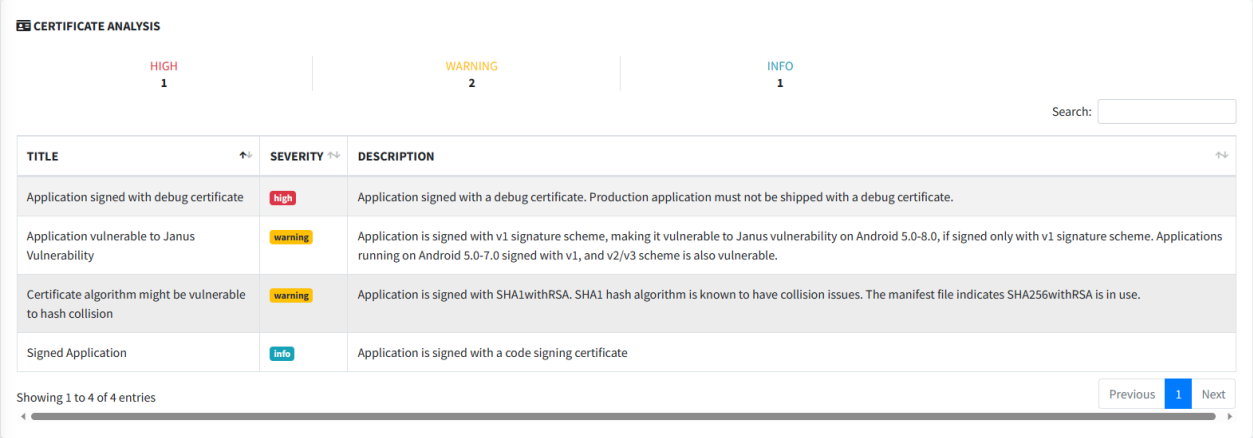
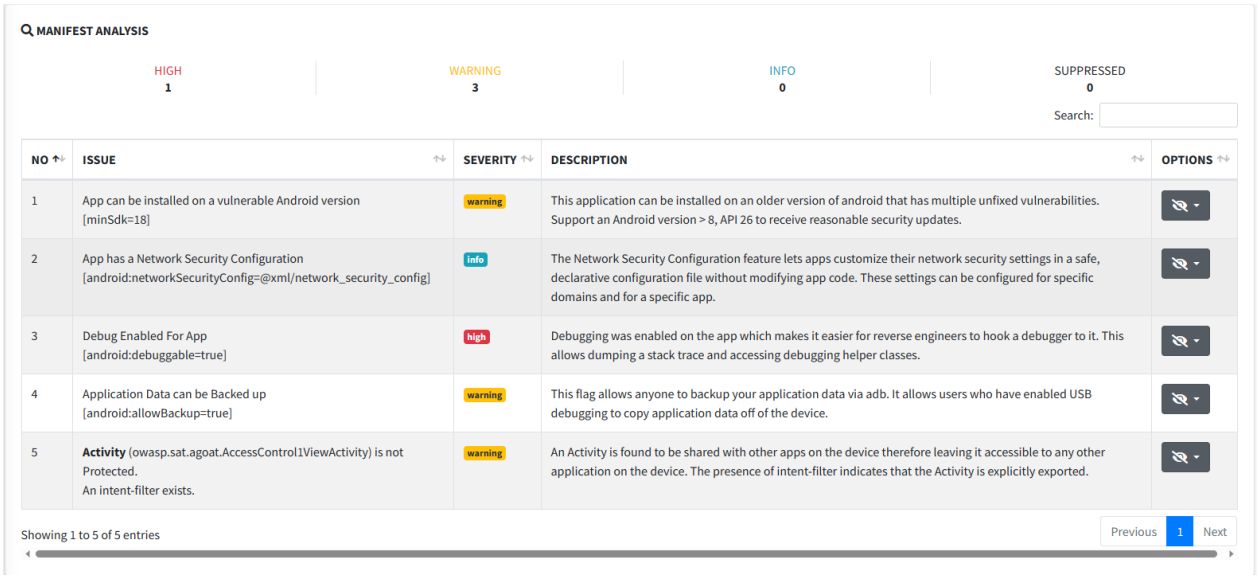


Figure 13: certificate analysis

Manifest Analysis: The "Manifest Analysis" feature in MobSF is a crucial tool that focuses on evaluating the security of an app's manifest file, which holds important information about the app's components and permissions. This feature classifies its findings into different categories: "High," "Warning," "Info," and "Suppressed," allowing users to grasp the severity and nature of manifest-related issues. To provide a quick overview, the feature presents counts for each category, indicating the prevalence and gravity of the manifest-related concerns. Furthermore, the "Manifest Analysis" feature showcases a comprehensive table encompassing key columns: "Issue," "Severity," "Description," and "Options." In the "Issue" column, it succinctly outlines the specific manifest-related problem. The "Severity" column offers insights into the potential impact of the issue on the app's security. The "Description" column furnishes an in-depth explanation of the concern, aiding users in understanding its implications. The "Options" column may offer users choices to suppress certain issues if needed.



MANIFEST ANALYSIS

HIGH	WARNING	INFO	SUPPRESSED
1	3	0	0

Search:

NO	ISSUE	SEVERITY	DESCRIPTION	OPTIONS
1	App can be installed on a vulnerable Android version [minSdk=18]	warning	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. Support an Android version > 8, API 26 to receive reasonable security updates.	
2	App has a Network Security Configuration [android:networkSecurityConfig=@xml/network_security_config]	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	Debug Enabled For App [android:debuggable=true]	high	Debugging was enabled on the app which makes it easier for reverse engineers to hook a debugger to it. This allows dumping a stack trace and accessing debugging helper classes.	
4	Application Data can be Backed up [android:allowBackup=true]	warning	This flag 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.	
5	**Activity** (owasp.sat.agoat.AccessControl1ViewActivity) is not Protected. An intent-filter exists.	warning	An Activity 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 Activity is explicitly exported.	

Showing 1 to 5 of 5 entries

Previous 1 Next

Figure 14: manifest analysis

Code Analysis: The "Code Analysis" feature within MobSF is a vital aspect that focuses on evaluating the security of an app's underlying codebase. This feature categorizes its findings into different levels: "High," "Warning," "Info," "Secure," and "Suppressed," helping users comprehend the seriousness and nature of code-related issues. To give a quick insight, the feature displays counts for each category, offering a concise representation of the prevalence and severity of code-related concerns. This feature presents a comprehensive table featuring key columns: "Issue," "Severity," "Standards," "Files," and "Options." In the "Issue" column, it succinctly describes the specific code-related problem detected. The "Severity" column indicates the potential impact of the issue on the app's security. The "Standards" column might highlight the relevant security standards or best practices that the issue violates. The "Files" column points to the specific code files associated with the concern. The "Options" column provides users with choices to suppress certain issues, if required.

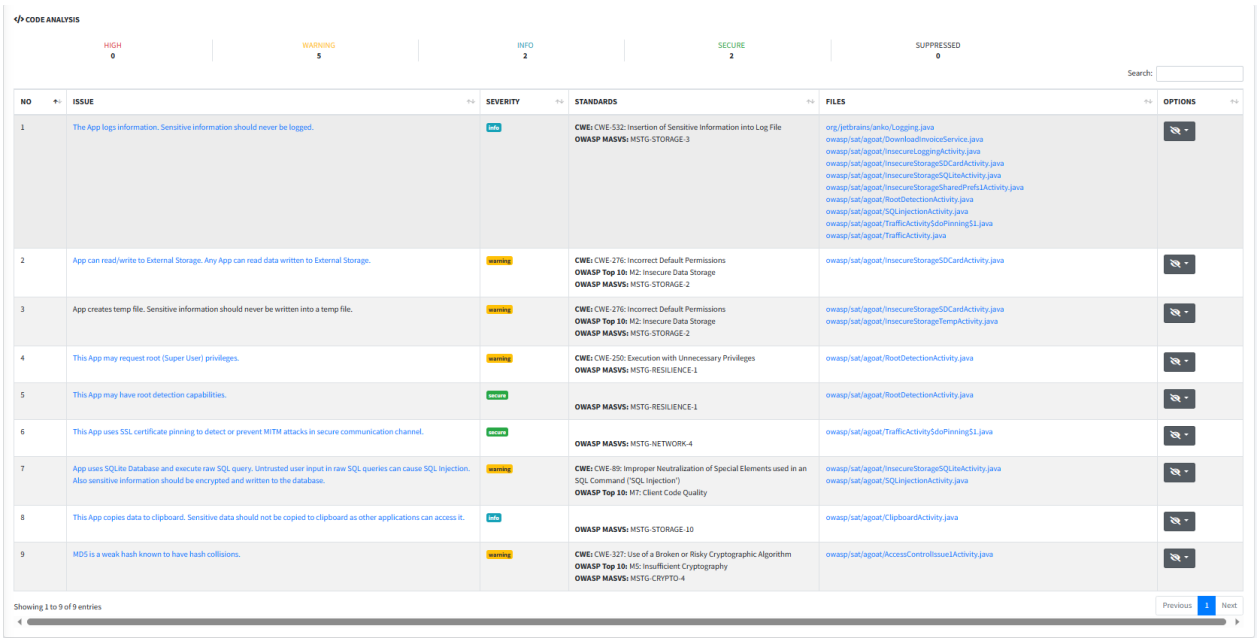


Figure 15: code analysis

Binary Analysis: The "Binary Analysis" feature in MobSF is a crucial component that focuses on evaluating the security of the compiled binary files of an app. This feature presents a detailed table encompassing several key attributes that provide insights into the app's binary makeup. The table includes the following columns:

Shared Object: Indicates whether the binary uses shared objects, which are modular pieces of code that can be shared across multiple programs.

NX (No-Execute) Permissions: Reflects whether the binary's memory pages have NX permissions, a crucial security measure to prevent executing code from certain memory areas.

Stack Canary: Indicates whether the binary implements stack canaries, which are security features that help detect stack buffer overflows.

RPATH: Represents the runtime path (RPATH) of the binary, indicating locations where the binary searches for shared libraries.

RUNPATH: Similar to RPATH, the runtime path (RUNPATH) specifies library search paths but with different loading semantics.

FORTIFY: Indicates whether the binary uses FORTIFY_SOURCE, a protection mechanism to prevent certain types of buffer overflows.

SYMBOLS STRIPPED: Reflects whether the binary's symbols have been stripped, which can make reverse engineering more difficult.

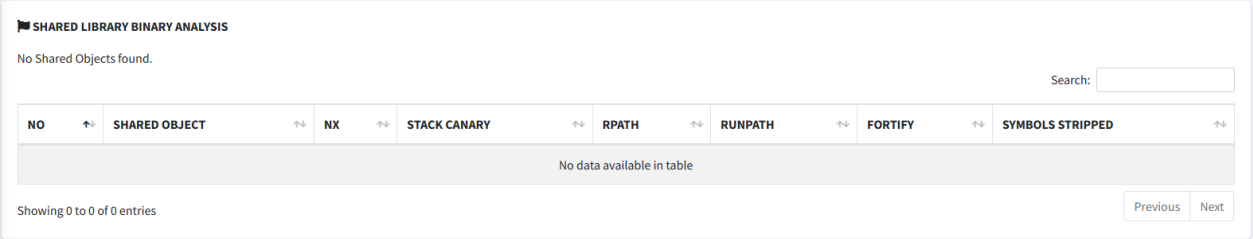


Figure 16: binary analysis

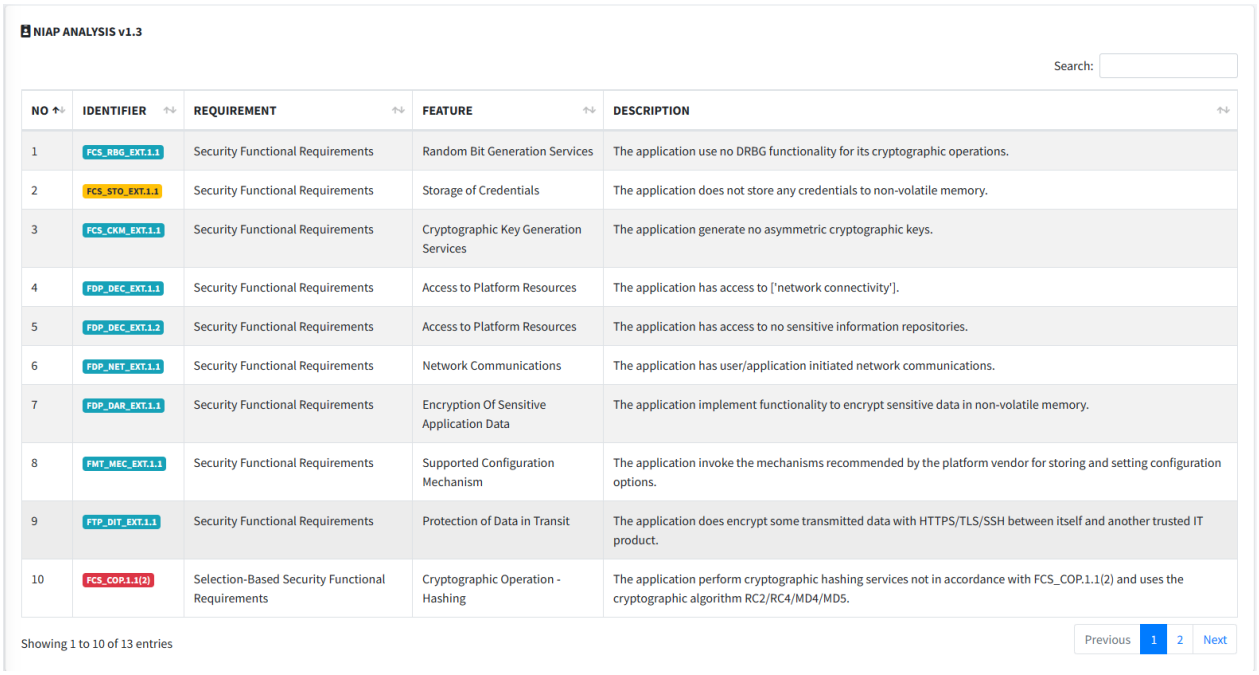
NIAP Analysis: The "NIAP Analysis" feature in MobSF focuses on evaluating how well an app aligns with the requirements set forth by the National Information Assurance Partnership (NIAP) for security and compliance standards. This feature presents a comprehensive table that contains four key columns: "Identifier," "Requirement," "Feature," and "Description."

Identifier: This column displays a unique identifier associated with a specific NIAP requirement. It helps users quickly identify and reference different requirements.

Requirement: The "Requirement" column outlines the specific security or compliance requirement set by NIAP that the app is being evaluated against.

Feature: This column highlights the app's features or attributes that correspond to or fulfill the NIAP requirement. It serves as a direct link between the requirement and the app's characteristics.

Description: The "Description" column provides additional context and explanations for the NIAP requirement, helping users understand the significance of adhering to this particular standard.



NIAP ANALYSIS v1.3

Search:

NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION
1	FCS_RBG_EXT.1.1	Security Functional Requirements	Random Bit Generation Services	The application use no DRBG functionality for its cryptographic operations.
2	FCS_STO_EXT.1.1	Security Functional Requirements	Storage of Credentials	The application does not store any credentials to non-volatile memory.
3	FCS_CKM_EXT.1.1	Security Functional Requirements	Cryptographic Key Generation Services	The application generate no asymmetric cryptographic keys.
4	FDP_DEC_EXT.1.1	Security Functional Requirements	Access to Platform Resources	The application has access to ['network connectivity'].
5	FDP_DEC_EXT.1.2	Security Functional Requirements	Access to Platform Resources	The application has access to no sensitive information repositories.
6	FDP_NET_EXT.1.1	Security Functional Requirements	Network Communications	The application has user/application initiated network communications.
7	FDP_DAR_EXT.1.1	Security Functional Requirements	Encryption Of Sensitive Application Data	The application implement functionality to encrypt sensitive data in non-volatile memory.
8	FMT_MEC_EXT.1.1	Security Functional Requirements	Supported Configuration Mechanism	The application invoke the mechanisms recommended by the platform vendor for storing and setting configuration options.
9	FTP_DIT_EXT.1.1	Security Functional Requirements	Protection of Data in Transit	The application does encrypt some transmitted data with HTTPS/TLS/SSH between itself and another trusted IT product.
10	FCS_COP.1.1(2)	Selection-Based Security Functional Requirements	Cryptographic Operation - Hashing	The application perform cryptographic hashing services not in accordance with FCS_COP.1.1(2) and uses the cryptographic algorithm RC2/RC4/MD4/MD5.

Showing 1 to 10 of 13 entries

Previous 1 2 Next

Figure 17: NIAP analysis

File Analysis: The "File Analysis" feature in MobSF is a pivotal component that focuses on evaluating files within the app to identify potential security concerns or anomalies. This feature presents a detailed table with two key columns: "Issue" and "Files."

Issue: In the "Issue" column, the feature succinctly outlines specific problems, vulnerabilities, or discrepancies detected within the app's files. These issues could range from security vulnerabilities to non-compliance with best practices.

Files: The "Files" column lists the specific files within the app that are associated with each identified issue. This provides a clear connection between the issue and the files in which it was found.

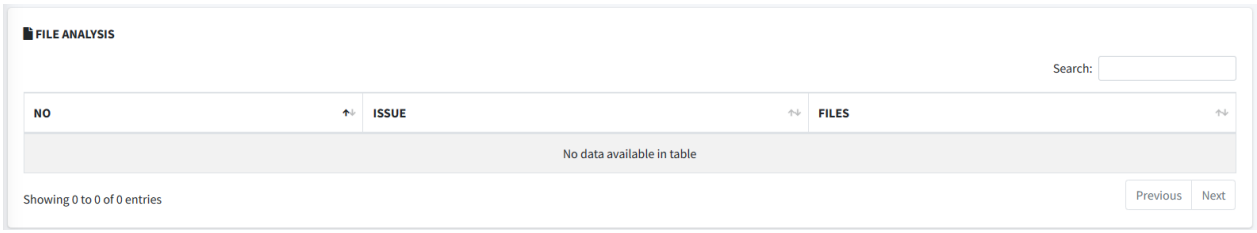


Figure 18: file analysis

5.2 Malware Analysis

APKiD ANALYSIS: The "APKiD Analysis" feature in MobSF is a crucial component that focuses on identifying the packer used to obfuscate the app's code."APKiD Analysis" gives information about how an APK was made. It identifies many compilers, packers, obfuscators, and other weird stuff. This feature presents a detailed table with two key columns: "DEX" and "Detection".

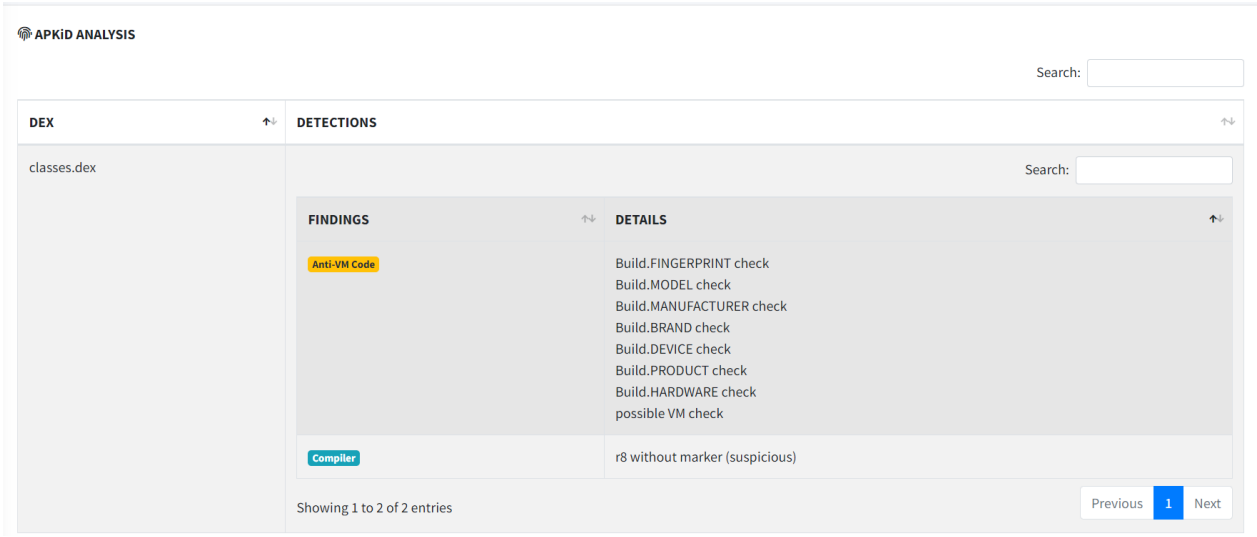


Figure 19: APKiD ANALYSIS

QUARK ANALYSIS Quark-Engine is a full-featured Android analysis framework written in Python for hunting threat intelligence inside the APK, DEX files.

SERVER LOCATIONS: This app may communicate with the following OFAC sanctioned list of countries.

Figure 20: possible servers

DOMAIN MALWARE CHECK: The domain used in this app is checked for malware.

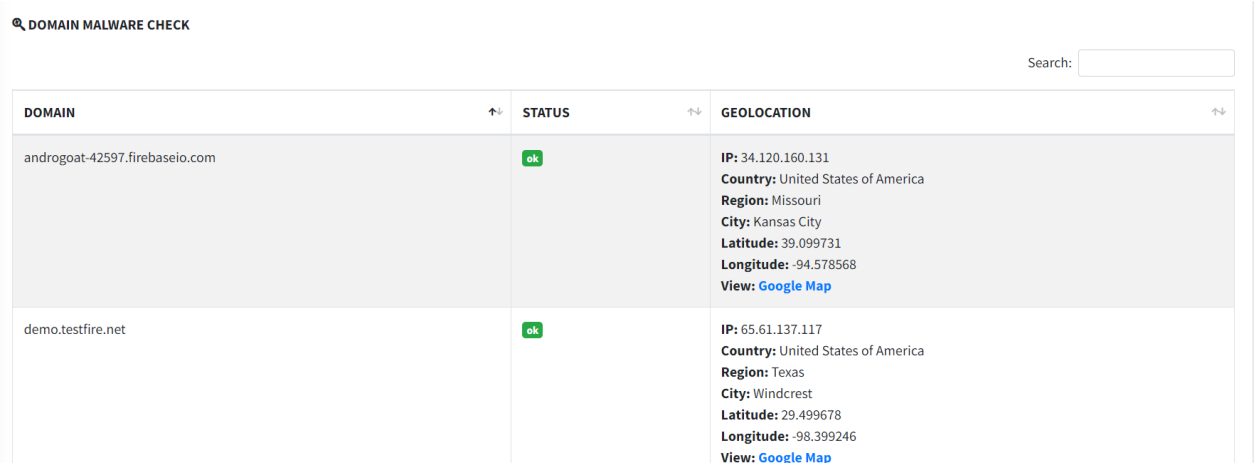
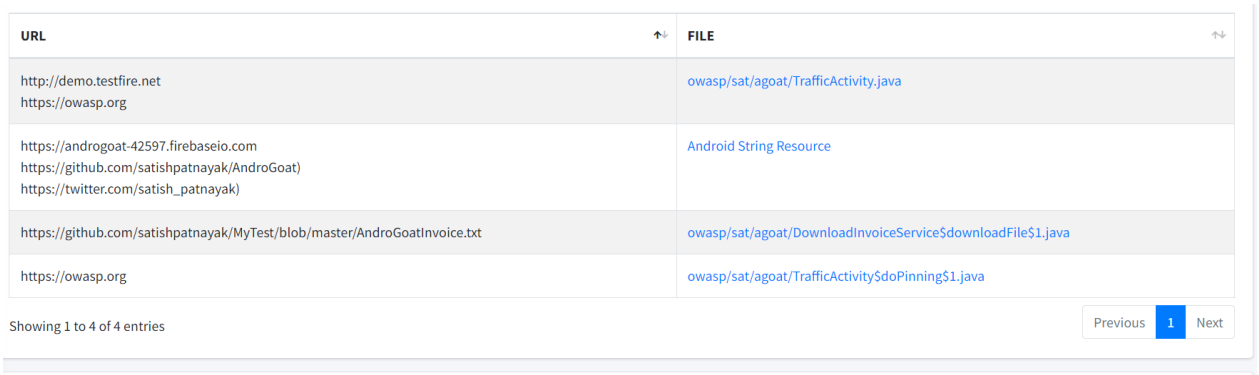


Figure 21: domain malware check

5.3 Reconnaissance

Here URLs, DB, Emails, Trackers, Strings and Hardcoded Secrets are extracted from the app.

URLs: The URLs used in the app are shown here.



URL	FILE
http://demo.testfire.net https://owasp.org	owasp/sat/agoat/TrafficActivity.java
https://androgoat-42597.firebaseio.com https://github.com/satishpatnayak/AndroGoat) https://twitter.com/satish_patnayak)	Android String Resource
https://github.com/satishpatnayak/MyTest/blob/master/AndroGoatInvoice.txt	owasp/sat/agoat/DownloadInvoiceService$downloadFile$1.java
https://owasp.org	owasp/sat/agoat/TrafficActivity$doPinning$1.java

Showing 1 to 4 of 4 entries

Previous 1 Next

Figure 22: URLs

Strings: The strings used in the app are shown here.This gives very long list of strings

Hardcoded Secrets: Any Hardcoded Secrets are shown here.

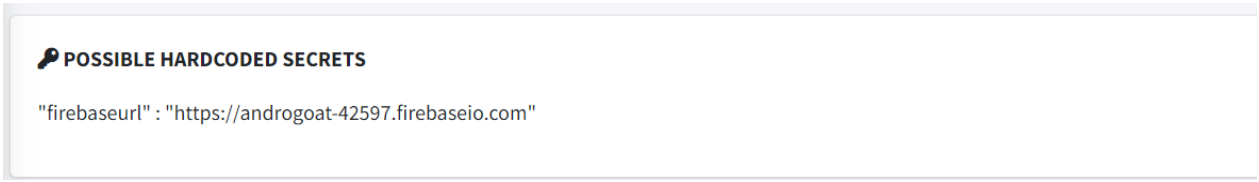


Figure 23: Hardcoded Secrets

5.4 Components

Here we can see the components of the app. Total Activities, services, receivers, providers, libraries and files.

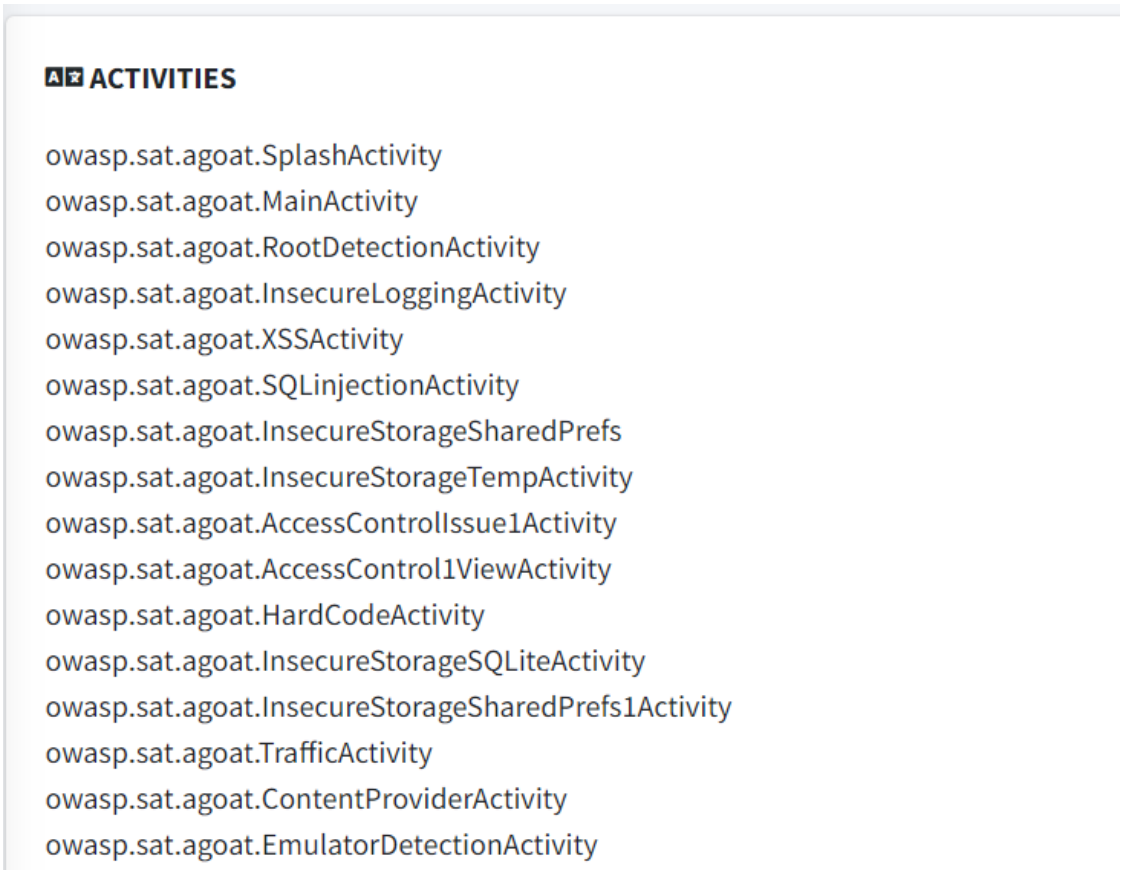


Figure 24: List of Activities

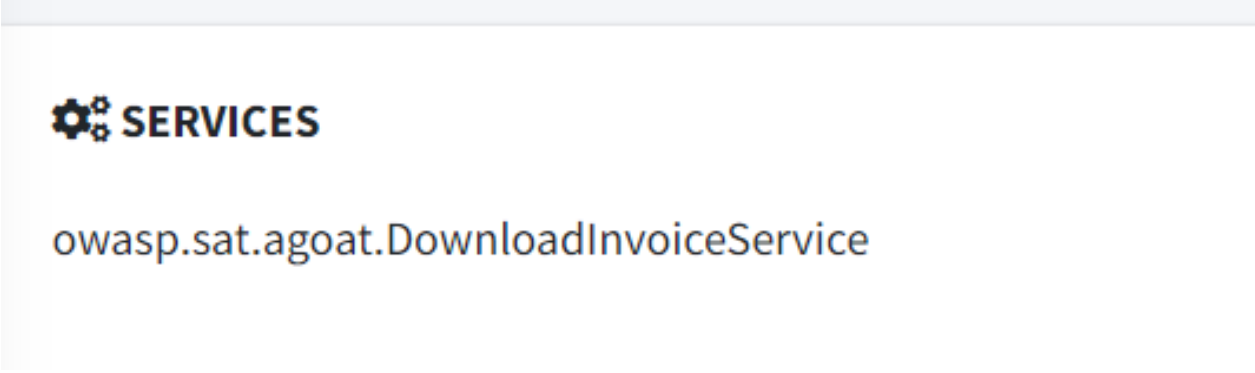


Figure 25: Services

Figure 26: List of Files

5.5 Report

There are two option for report generation. One is for generating pdf and another is to print the report.

5.6 Recent Option

Here we can see a list of recent files. We can also see the details of the files by clicking on the file name.

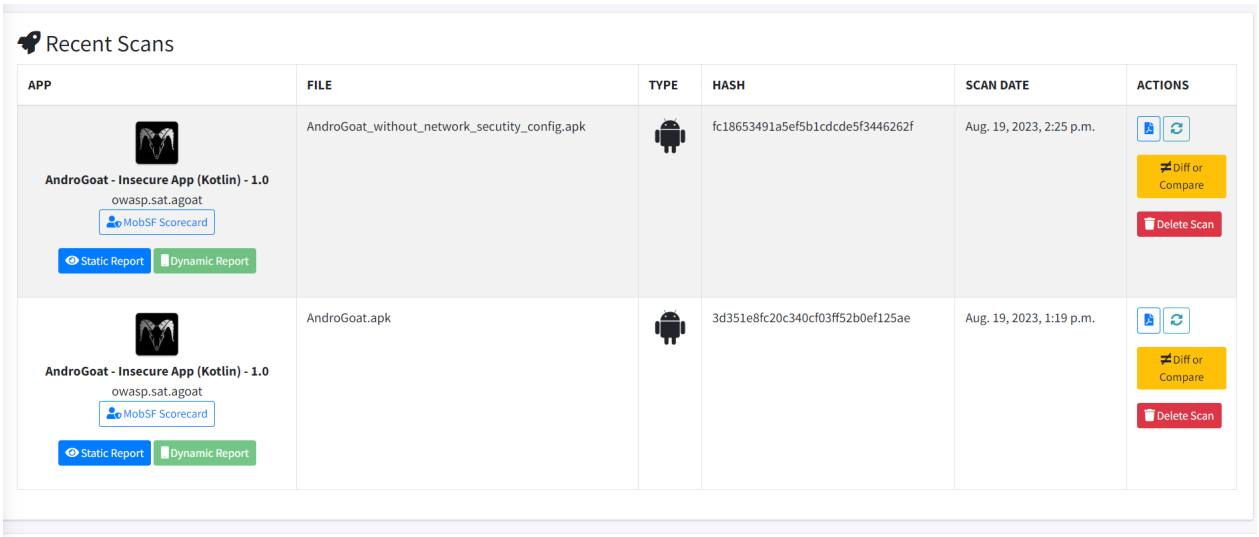


Figure 27: recent files

Here we have some options:

- MobSF Scorecard: A summary overview is shown here

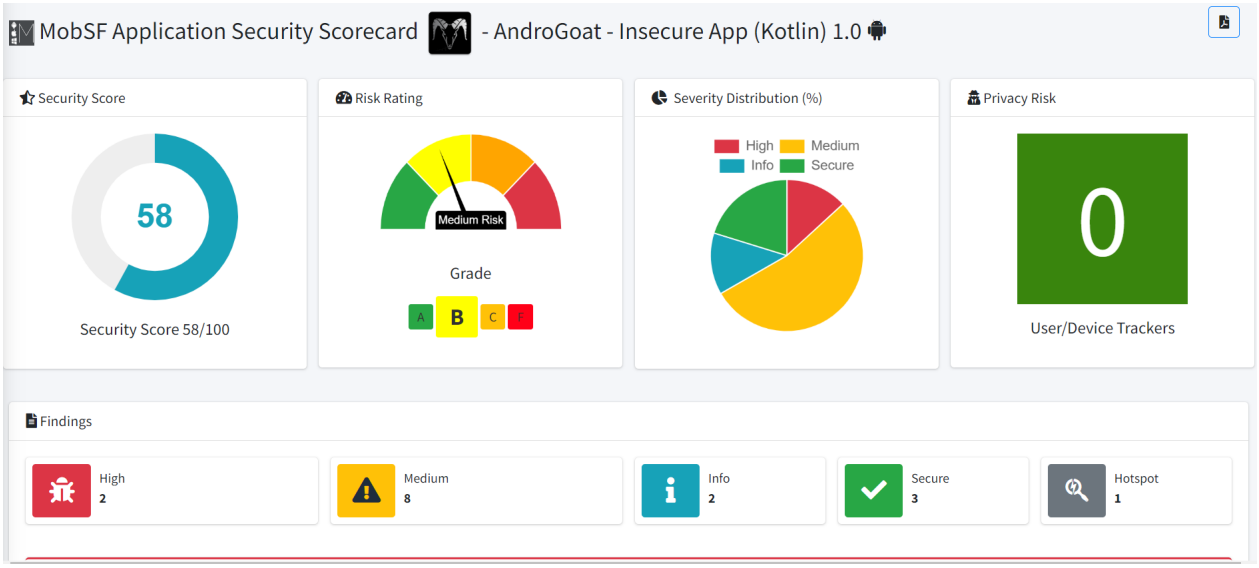


Figure 28: Scorecard

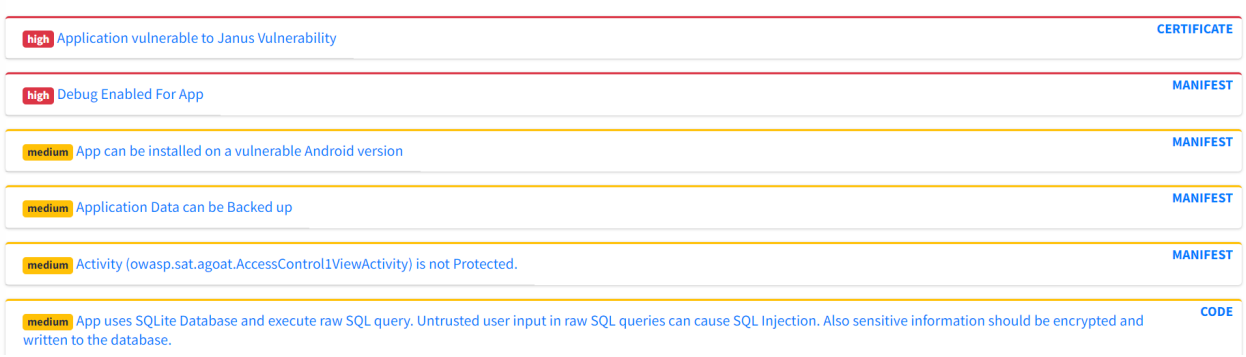


Figure 29: Scorecard 2

- Rescan: rescan the file
- View Report: view the report of the file
- Delete: delete the scan
- Diff or Compare: compare two files

6 Dynamic Analysis

6.1 Starting the anlysis

To run the dynamic analysis we need to follow these steps. The steps shouldn't be alter.

Step 1: We need to run an app on an emulator.

```
emulator -avd Pixel_5_API_28 -writable-system -no-snapshot
```

Step 2: We need to run mobsf with this code.

```
docker run -it --rm -p 8000:8000 -p 1337:1337 -e \\
MOBSF_ANALYZER_IDENTIFIER=emulator-5554 \\
opensecurity/mobile-security-framework-mobsf:latest
```

This Link can be followed for more information

6.2 Running the Analysis

Do the static analysis first. Then click on the **Start Dynamic Analysis** button. You can find it on "Scan Options" tab. After some time this screen will appear.

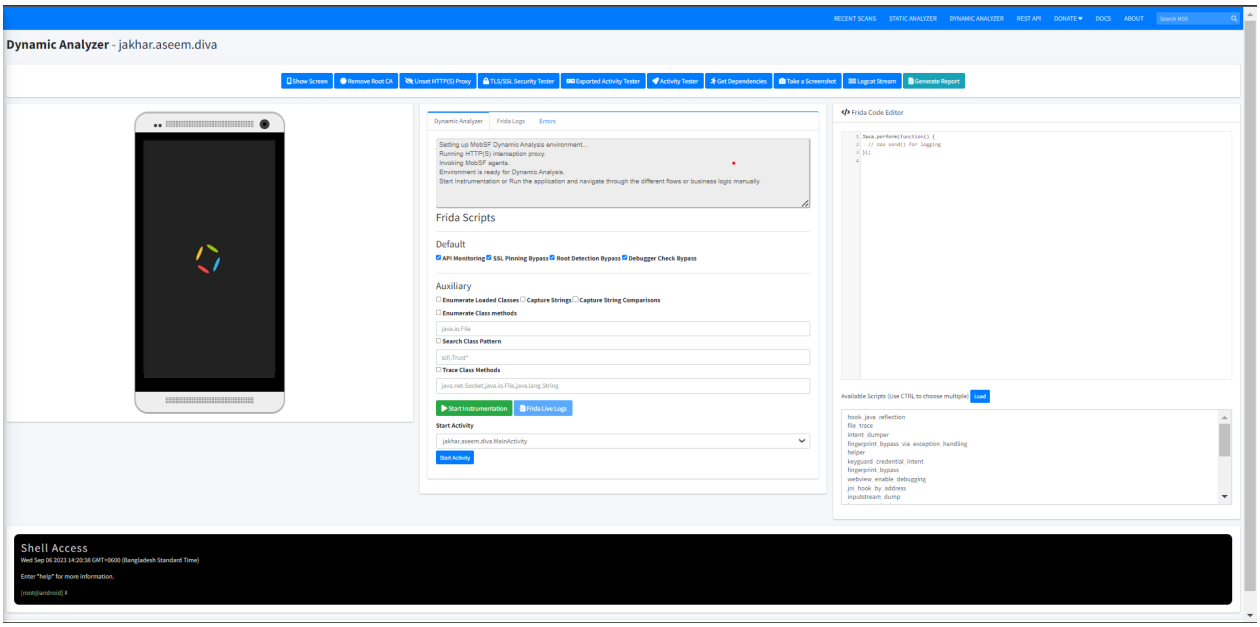


Figure 30: dynamic analysis

6.3 Options

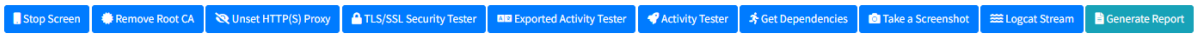


Figure 31: Tools

6.3.1 Show Screen

This option eanble the mapping of the screen. It will show the screen of the emulator.

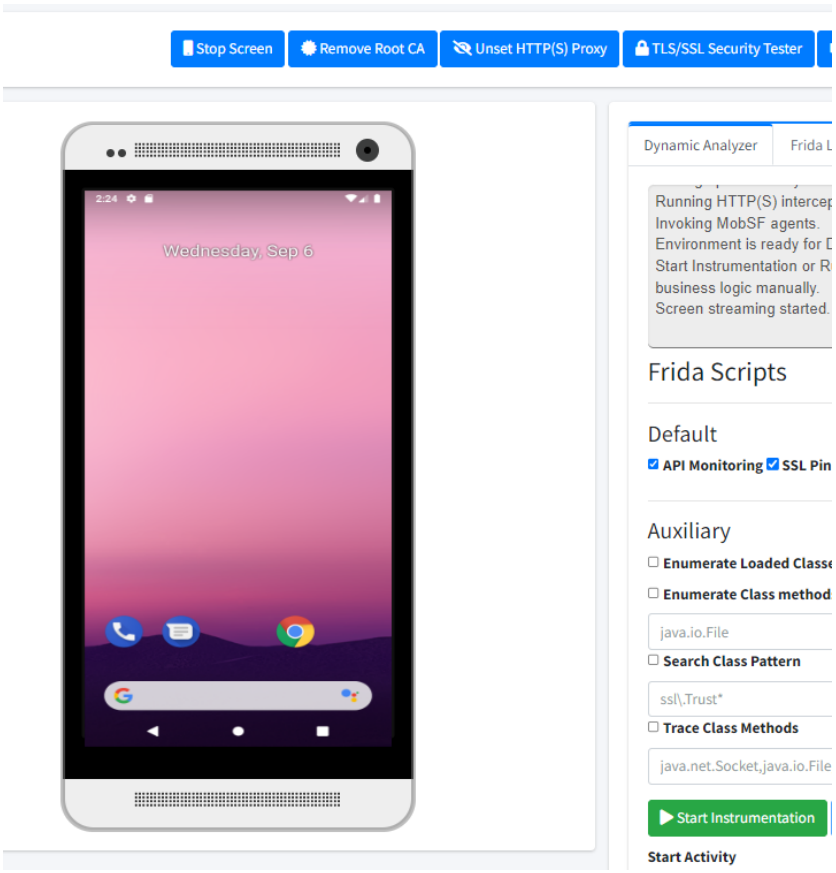


Figure 32: Show Screen

6.3.2 Remove Root CA

This option will remove or install the root CA from the emulator. This helps to intercept the traffic. via Burp Suite.

6.3.3 TLS/SSL Security Tester

This option will test the TLS/SSL security of the app.

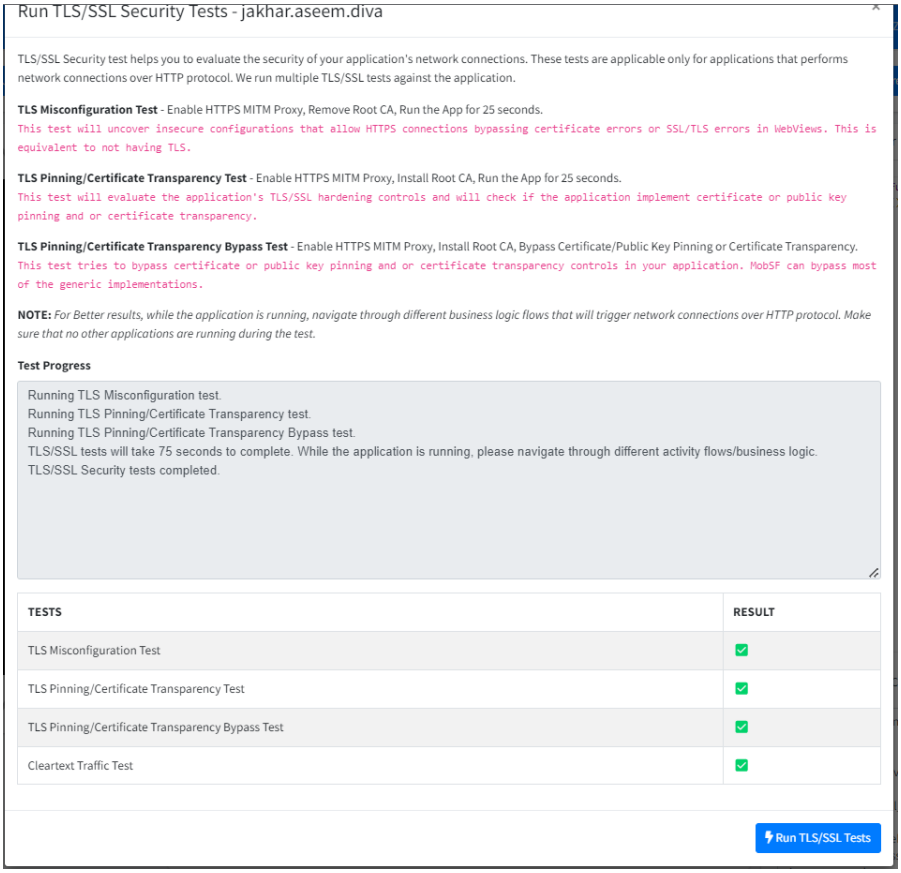


Figure 33: TLS/SSL Security Tester

6.3.4 Exported Activity Tester

This option will test the exported activity of the app. In background MobSF will launch each exported activity of the app and store screenshot of the activity.

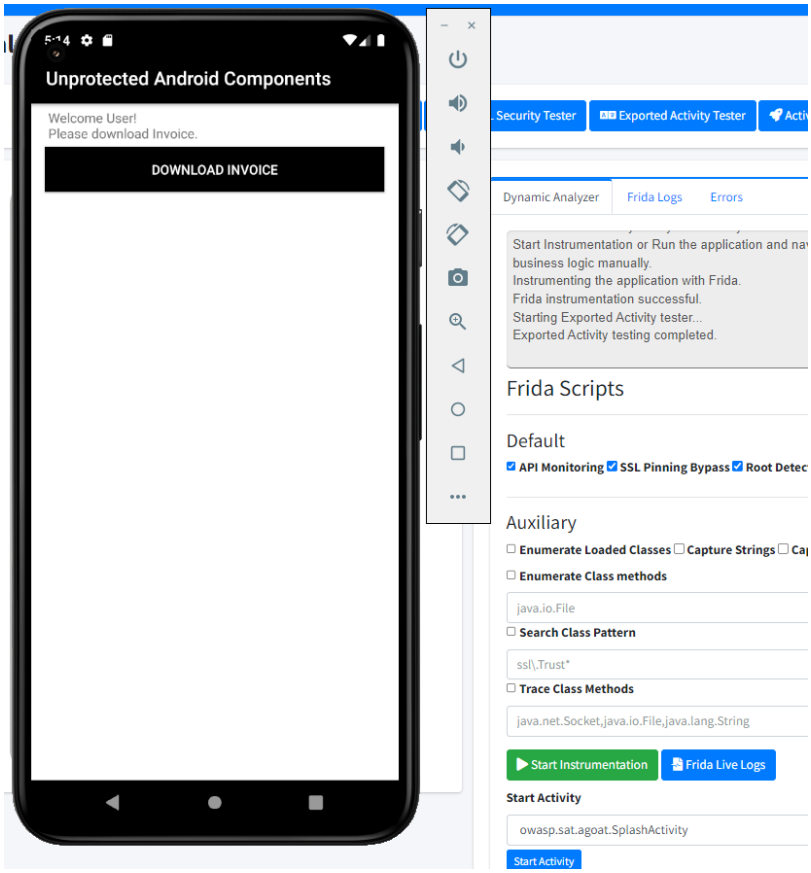


Figure 34: Exported Activity Tester

6.3.5 Activity Tester

It will launch each activity of the app and store screenshot of the activity. It takes a lot of time to complete.

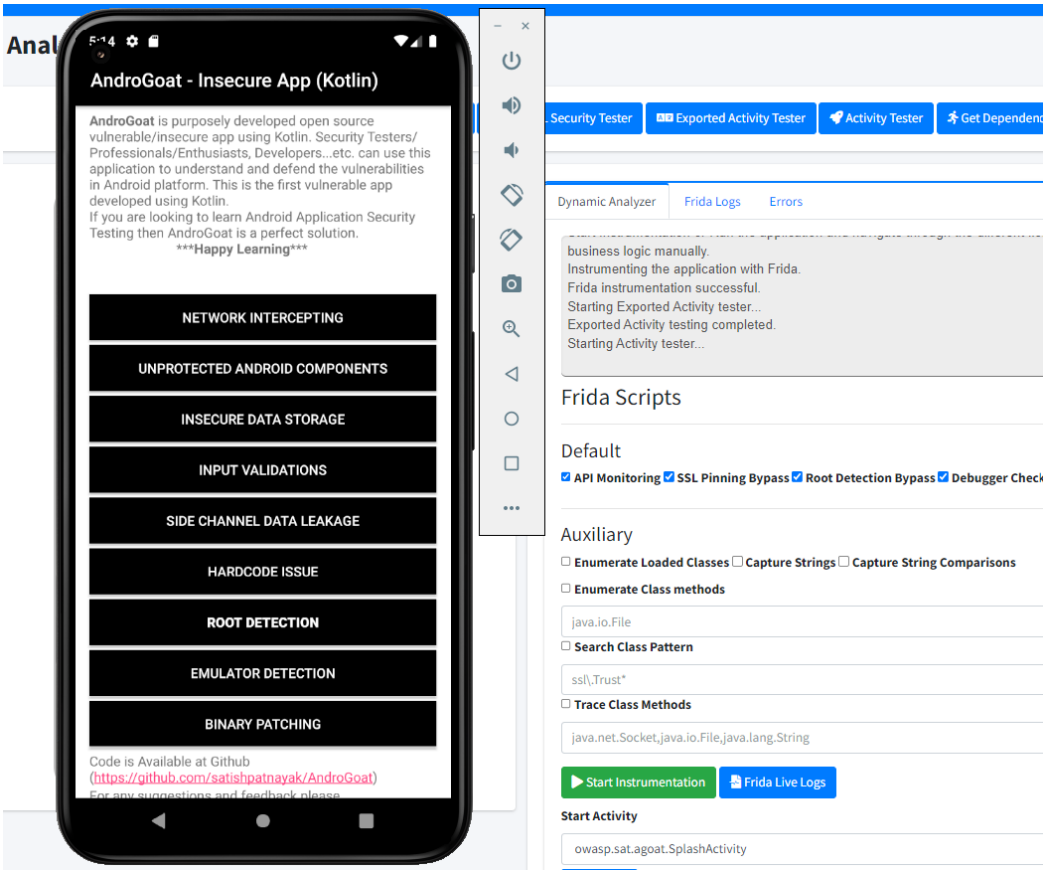


Figure 35: Activity Tester

6.3.6 LogCat Stream

This option will open a new window and show the logcat stream of the app.

Figure 36: LogCat Stream

6.4 Frida Script

Dynamic analyser has a feature to run frida script. To run this script *start instrumentation* should be clicked. default scripts are given here.

- **API Monitoring**
- **SSL Pinning Bypass**
- **Root Detection Bypass**
- **Debugger Check Bypass**

```
2023-09-06 17:26:42 [INFO] 06/Sep/2023 11:26:42 - Frida Server is already running
2023-09-06 17:26:42 [INFO] 06/Sep/2023 11:26:42 - Spawning owasp.sat.agoat
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] Loaded Frida Script - root_bypass
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] Loaded Frida Script - ssl_pinning_bypass
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] Loaded Frida Script - debugger_check_bypass
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] okhttp CertificatePinner not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] DataTheorem trustkit not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] Appcelerator PinningTrustManager not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] Apache Cordova SSLCertificateChecker not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] Wultra CertStore.validateFingerprint not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] Xutils not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] httpclientandroidlib not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] Cronet not found
2023-09-06 17:26:44 [DEBUG] 06/Sep/2023 11:26:44 - [Frida] [SSL Pinning Bypass] certificatetransparency.CTInterceptorBuilder not f
2023-09-06 17:27:02 [INFO] 06/Sep/2023 11:27:02 - Starting API monitor streaming
2023-09-06 17:28:20 [INFO] 06/Sep/2023 11:28:20 - Launching Activity - owasp.sat.agoat.MainActivity
```

Figure 37: Frida Script

6.4.1 Live API Monitoring

This script will show the API calls of the app

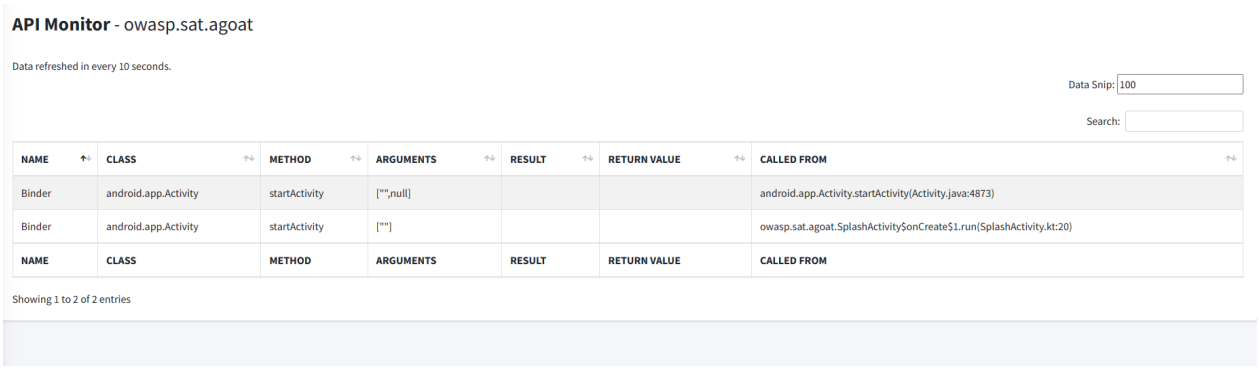


Figure 38: Live API Monitoring

6.4.2 Auxiliary Scripts

- **Enumerate Loaded Classes**
- **Capture Strings**
- **Capture String Comparisons**
- **Enumerate Class methods**

```
[*] [String Compare] layout_inflater == autofill ? false
[*] [String Compare] THISISTEST == THISISTEST ? true
[*] [String Compare] window == autofill ? false
[*] [String Compare] search == autofill ? false
[*] [String Compare] layout_inflater == autofill ? false
[*] [String Compare] THISISTEST == NEW2019 ? false
[*] [String Compare] merge == LinearLayout ? false
[*] [String Compare] res/layout/transient_notification.xml ==
[*] [String Compare] shape == drawable ? false
```

Figure 39: String Comparisons

6.4.3 Frida Code Editor

This section can be use to write custom frida scripts and upload it to app.

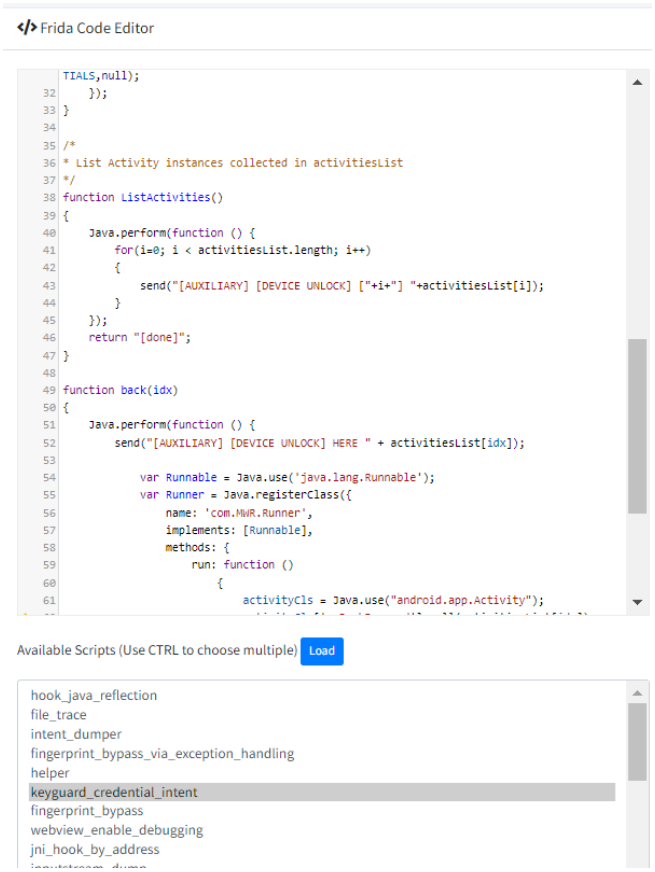


Figure 40: Frida Code Editor

6.4.4 Shell Access

There is also a shell access to the emulator.Which can run adb commands.

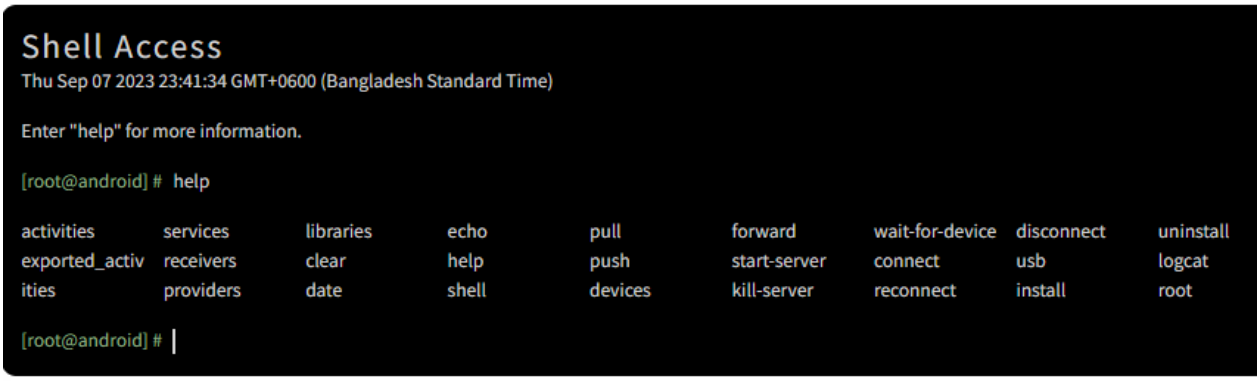


Figure 41: Frida Code Editor

6.5 Report Generation

Generate Report: This option will stop the analysis and generate a report of the dynamic analysis.

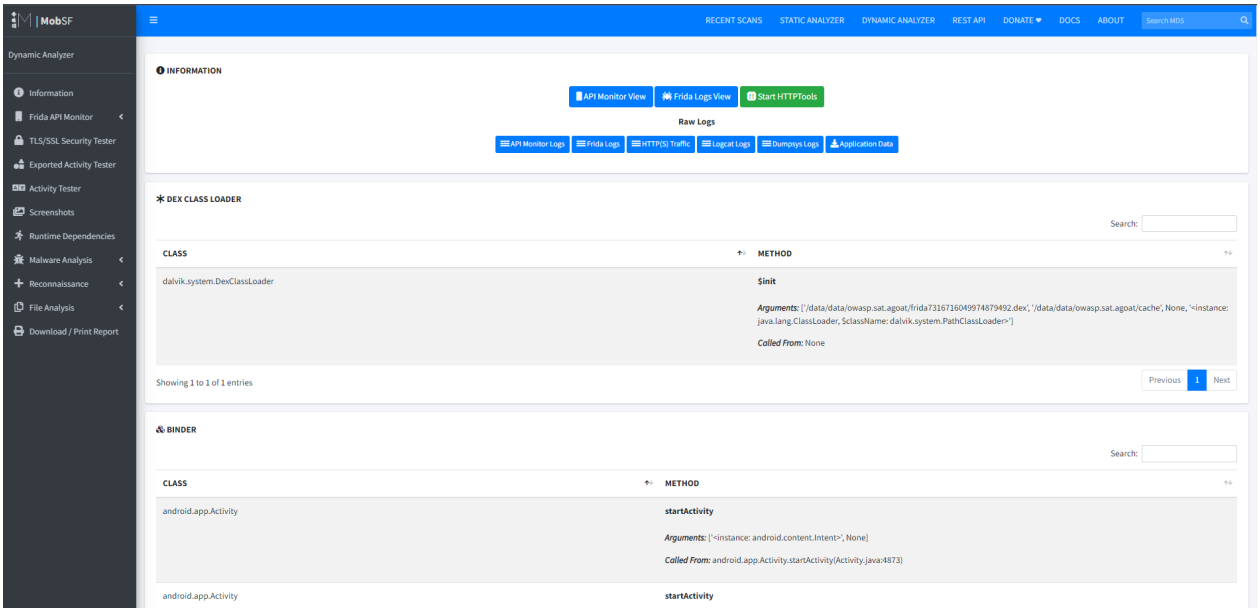


Figure 42: Report Generation

Here we also get some options other than saving the report as pdf.

6.5.1 StartHTTPTools

This open a new window with captured traffic. Shows the request and response of the app and ca send it to fuzzer(Burp Suite).

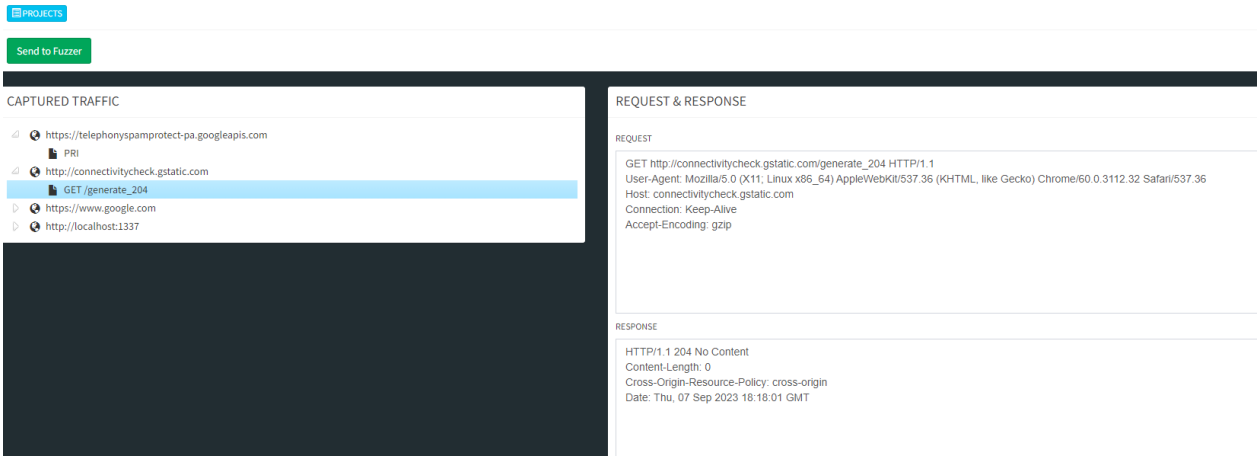


Figure 43: Report Generation

Others

Further options are

- **API Monitor Log** : This will show the API calls of the app
- **Frida Logs** : This will show the frida logs
- **HTTP(s) traffic** : This will show the captured traffic
- **Logcat Logs** : This will show the logcat logs
- **Dumpsys Log** : This will show the dumpsys logs
- **Application Data** : This will download the application data

7 Conclusion

We examined the Mobile Security Framework (MobSF), a crucial instrument for evaluating the security of mobile applications, in this paper. We looked at its key characteristics, such as its static and dynamic analysis tools, highlighting how effective it is in finding vulnerabilities in mobile applications at different stages.

The static analysis capability gives us the ability to examine the source code and resources of programmes without actually running them, providing insightful information about possible security flaws. Dynamic analysis, on the other hand, offers a real-time assessment of an application's behaviour, enabling us to identify runtime vulnerabilities that could otherwise go unnoticed.

We faced some difficulties with MobSF, particularly in properly setting and operating the framework. Make that all required dependencies and settings are in place.

It is advised that anyone looking to integrate MobSF into their security workflow start by having a thorough understanding of the target application and its intended environment. This background information will enable more accurate and significant analytical outcomes.

To demonstrate how MobSF may be included into current security processes, we gave a small code overview. Users can automate the evaluation of mobile applications by using its API, enabling effective and scalable security testing.

In conclusion, MobSF is a strong tool for mobile application security. Although not without difficulties, its mix of static and dynamic analysis capabilities offers a thorough method for locating vulnerabilities. MobSF may dramatically improve existing procedures with careful configuration and integration.