Mobile Bank Assessment and Incident Response Plan

By:

Adubal, Last Cloude

I. Executive Summary

Vulnerability assessment on the Bank APK revealed that the hardcoded API keys, unencrypted data transmissions, and insufficient authentication mechanisms can be easily exploited. The use of reverse engineering tool **APKTool**, enabled the practitioner to bypass security protocols and conduct fraudulent activities.

Recommended response steps included:

- Immediate revocation of exposed API keys and temporary disabling of compromised accounts.
- Release of a patched app version with improved encryption and multi-factor authentication.
- Implementation of enhanced monitoring systems and real-time anomaly detection tools.
- Initiation of legal and compliance reviews to address possible data protection violations.

The attack method involved reverse engineering the Bank APK using APKTool, which exposed sensitive components such as poorly obfuscated code and improperly stored API keys. Attackers injected malicious code to bypass authentication mechanisms and disabled SSL pinning, allowing them to intercept and reroute login credentials.

Affected Systems:

- Android client application (primary target)
- Backend services shared across Android and iOS platforms

- User authentication and financial transaction modules

Mitigation strategies implemented:

- Strengthening code obfuscation and removing all hardcoded secret.
- Enforcing multi-factor authentication and SSL certificate pinning
- Securing APIs with rate limiting, session validation, and behavioral threat analytics
- Enhancing mobile threat detection tools to flag emulator use and root access manipulation

These measures are designed not only to contain the immediate threat but also to future-proof the app against similar sophisticated attacks.

II. Vulnerability Assessment

Static Analysis Findings Overview

An initial static security assessment of the APK was conducted using the Mobile Security Framework (MobSF). The automated analysis assigned the application a high-risk security rating and flagged multiple critical issues related to insecure configuration management and potential sensitive data exposure.

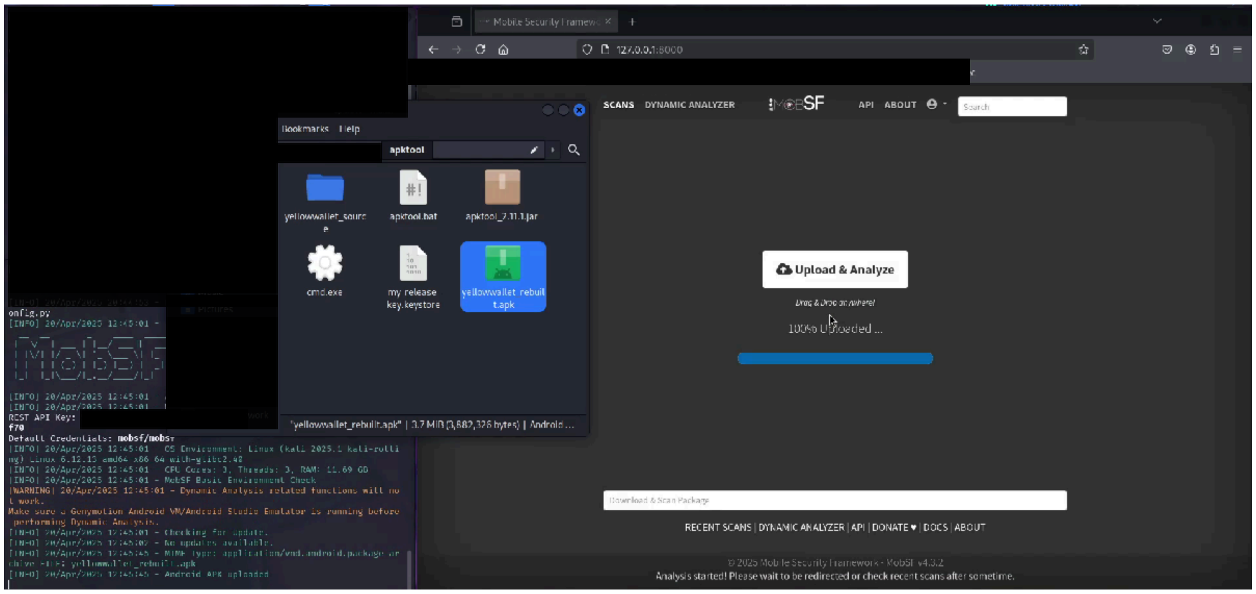
These findings indicated a strong likelihood of embedded configuration artifacts within the application package. As a result, manual reverse engineering was subsequently performed using APKTool to further inspect the application resources and validate the presence of exposed configuration files. Screenshots of the MobSF findings that motivated this deeper analysis are provided in Appendix A.

Reverse Engineering Analysis Using APKTool

In a controlled environment, tools such as MobSF and APKTool were used to perform a detailed static analysis of the application in order to identify vulnerabilities that could potentially be exploited to gain unauthorized access to sensitive user data. This process revealed weaknesses in the application's security architecture, including exposed API keys, insecure communication practices, and flaws in authentication logic.

MobSF Findings

Figure 1.1: Python script was executed to run the MobSF server.



```
onfig.py
[INFO] 20/Apr/2025 12:45:01 -
[INFO] 20/Apr/2025 12:45:01
[INFO] 20/Apr/2025 12:45:01
REST API Key: [redacted]
f70
Default Credentials: mobsf/mobsf
[INFO] 20/Apr/2025 12:45:01   OS Environment: Linux (kali 2025.1 kali-rolli
ng) Linux 6.12.13 amd64 x86 64 with-glibc2.40
[INFO] 20/Apr/2025 12:45:01   CPU Cores: 3, Threads: 3, RAM: 11.69 GB
[INFO] 20/Apr/2025 12:45:01 - MobSF Basic Environment Check
[WARNING] 20/Apr/2025 12:45:01 - Dynamic Analysis related functions will no
t work.
Make sure a Genymotion Android VM/Android Studio Emulator is running before
 performing Dynamic Analysis.
[INFO] 20/Apr/2025 12:45:01 - Checking for update.
[INFO] 20/Apr/2025 12:45:02 - No updates available.
[INFO] 20/Apr/2025 12:45:45 - MIME Type: application/vnd.android.package-ar
chive FILE: yellowwallet_rebuilt.apk
[INFO] 20/Apr/2025 12:45:45 - Android APK uploaded
```

[redacted]

"yellowwallet_rebuilt.apk" | 3.7 MiB (3,682,326 bytes) | Android ...

127.0.0.1:8000

100% Uploaded ...

© 2025 Mobile Security Framework - MobSF v4.3.2

Analysis started! Please wait to be redirected or check recent scans after sometime.

We then opened a web browser and entered the address of the MobSF's local server. Dropping the APK to begin the scanning process.

The MobSF scan identified several high- and medium-risk issues provided in Appendix A, including:

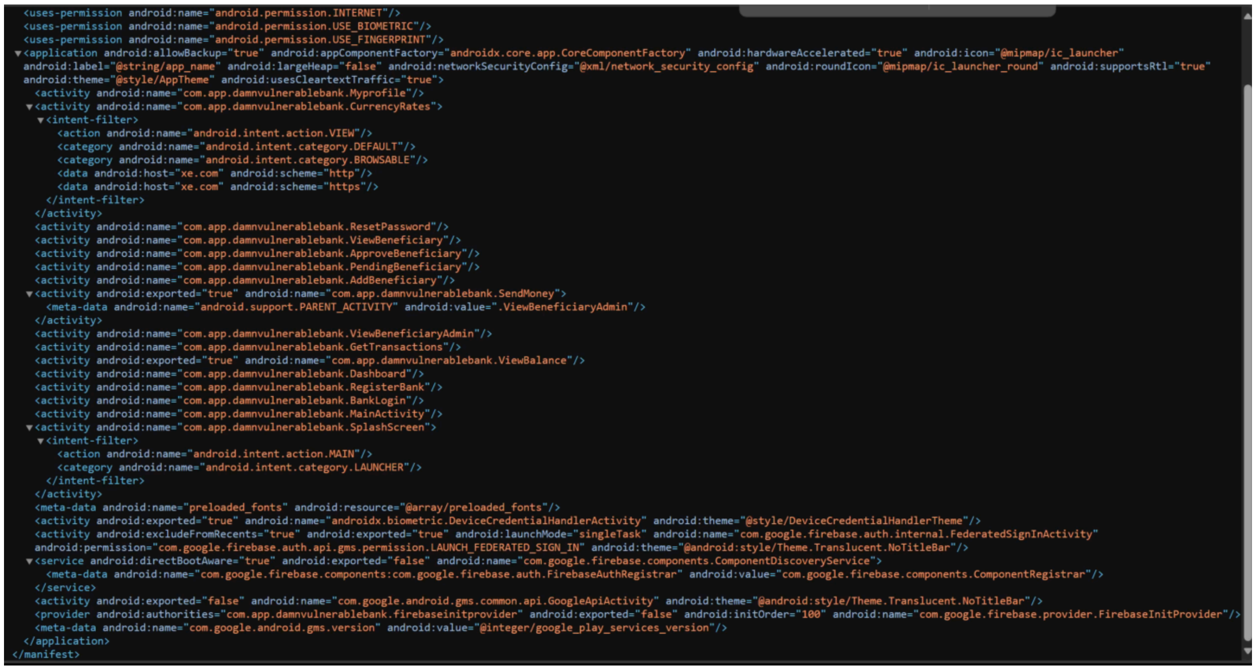
1. Insecure storage of sensitive information
2. Weak or missing encryption mechanisms
3. Unauthenticated or insufficiently protected API endpoints

These automated findings provided the foundation for the manual inspection carried out using APKTool, which focused on confirming and contextualizing the exposed configuration artifacts identified during the initial assessment.

We used APKTool to decompile the Bank APK, which allowed us to examine key configuration files such as “AndroidManifest.xml” and internal resource files. The “AndroidManifest.xml” file revealed important security-relevant settings like:

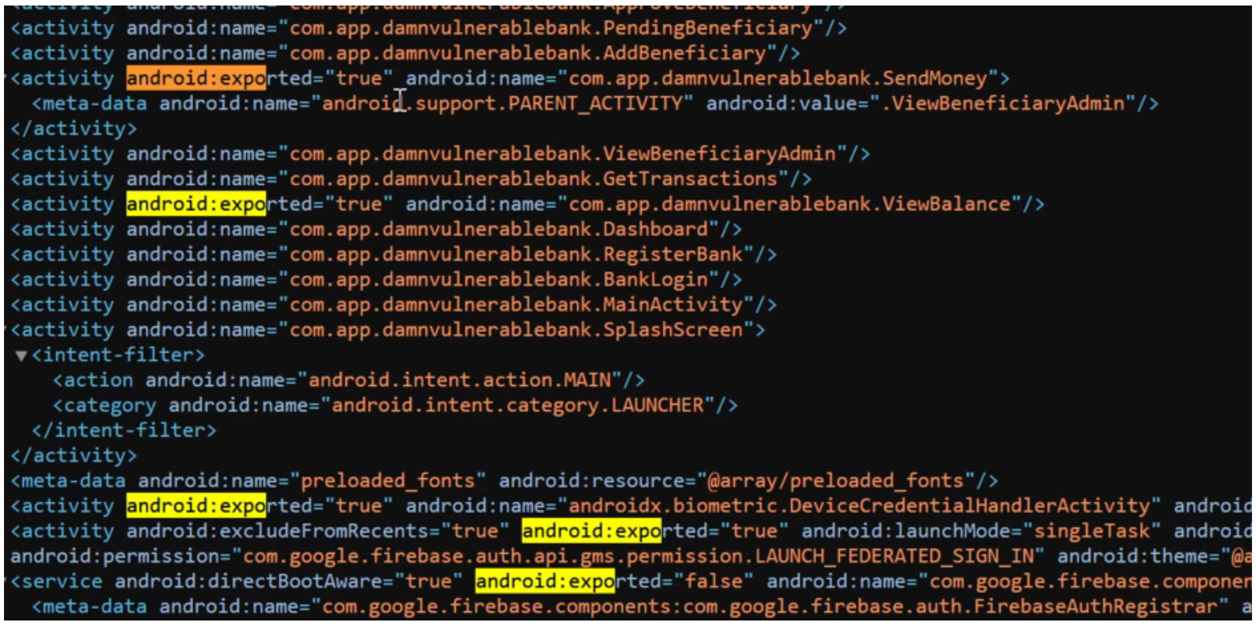
- Declared permissions such as INTERNET, USE_BIOMETRIC, and USE_FINGERPRINT suggest sensitive operation without proof of secure implementation.

Figure 1.2: Opening the XML file.



```
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.USE_BIOMETRIC"/>
  <uses-permission android:name="android.permission.USE_FINGERPRINT"/>
▼<application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:hardwareAccelerated="true" android:icon="@mipmap/ic_launcher"
  android:label="@string/app_name" android:largeHeap="false" android:networkSecurityConfig="@xml/network_security_config" android:roundIcon="@mipmap/ic_launcher_round" android:supportsRtl="true"
  android:theme="@style/AppTheme" android:usesCleartextTraffic="true">
    <activity android:name="com.app.damnvulnerablebank.Myprofile"/>
  ▼<activity android:name="com.app.damnvulnerablebank.CurrencyRates">
    ▼<intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:host="xe.com" android:scheme="http"/>
        <data android:host="xe.com" android:scheme="https"/>
      </intent-filter>
    </activity>
    <activity android:name="com.app.damnvulnerablebank.ResetPassword"/>
    <activity android:name="com.app.damnvulnerablebank.ViewBeneficiary"/>
    <activity android:name="com.app.damnvulnerablebank.ApproveBeneficiary"/>
    <activity android:name="com.app.damnvulnerablebank.PendingBeneficiary"/>
    <activity android:name="com.app.damnvulnerablebank.AddBeneficiary"/>
  ▼<activity android:exported="true" android:name="com.app.damnvulnerablebank.SendMoney">
      <meta-data android:name="android.support.PARENT_ACTIVITY" android:value=".ViewBeneficiaryAdmin"/>
    </activity>
    <activity android:name="com.app.damnvulnerablebank.ViewBeneficiaryAdmin"/>
    <activity android:name="com.app.damnvulnerablebank.GetTransactions"/>
    <activity android:exported="true" android:name="com.app.damnvulnerablebank.ViewBalance"/>
    <activity android:name="com.app.damnvulnerablebank.Dashboard"/>
    <activity android:name="com.app.damnvulnerablebank.RegisterBank"/>
    <activity android:name="com.app.damnvulnerablebank.BankLogin"/>
    <activity android:name="com.app.damnvulnerablebank.MainActivity"/>
  ▼<activity android:name="com.app.damnvulnerablebank.SplashScreen">
    ▼<intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
    <meta-data android:name="preloaded_fonts" android:resource="@array/preloaded_fonts"/>
    <activity android:exported="true" android:name="androidx.biometric.DeviceCredentialHandlerActivity" android:theme="@style/DeviceCredentialHandlerTheme"/>
    <activity android:excludeFromRecents="true" android:exported="true" android:launchMode="singleTask" android:name="com.google.firebase.auth.internal.FederatedSignInActivity"
    android:permission="com.google.firebase.auth.api.gms.permission.LAUNCH_FEDERATED_SIGN_IN" android:theme="@android:style/Theme.Translucent.NoTitleBar"/>
  ▼<service android:directBootAware="true" android:exported="false" android:name="com.google.firebase.components.ComponentDiscoveryService">
      <meta-data android:name="com.google.firebase.components:com.google.firebase.auth.FirebaseAuthRegistrar" android:value="com.google.firebase.components.ComponentRegistrar"/>
    </service>
    <activity android:exported="false" android:name="com.google.android.gms.common.api.GoogleApiActivity" android:theme="@android:style/Theme.Translucent.NoTitleBar"/>
    <provider android:authorities="com.app.damnvulnerablebank.firebaseinitprovider" android:exported="false" android:initOrder="100" android:name="com.google.firebase.provider.FirebaseInitProvider"/>
    <meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version"/>
  </application>
</manifest>
```

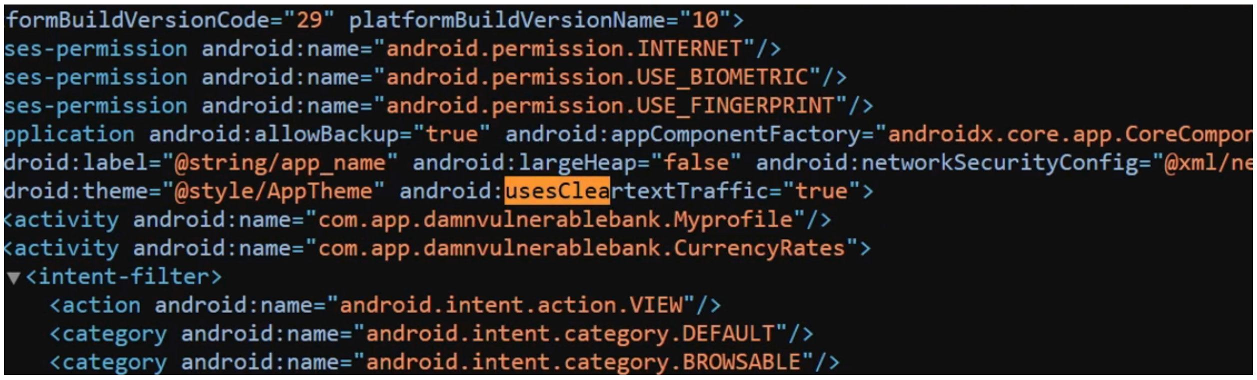
Figure 1.3: XML file fidnings.



```
<activity android:name="com.app.damnvulnerablebank.ApproveBeneficiary"/>
<activity android:name="com.app.damnvulnerablebank.PendingBeneficiary"/>
<activity android:name="com.app.damnvulnerablebank.AddBeneficiary"/>
<activity android:exported="true" android:name="com.app.damnvulnerablebank.SendMoney">
  <meta-data android:name="android.support.PARENT_ACTIVITY" android:value=".ViewBeneficiaryAdmin"/>
</activity>
<activity android:name="com.app.damnvulnerablebank.ViewBeneficiaryAdmin"/>
<activity android:name="com.app.damnvulnerablebank.GetTransactions"/>
<activity android:exported="true" android:name="com.app.damnvulnerablebank.ViewBalance"/>
<activity android:name="com.app.damnvulnerablebank.Dashboard"/>
<activity android:name="com.app.damnvulnerablebank.RegisterBank"/>
<activity android:name="com.app.damnvulnerablebank.BankLogin"/>
<activity android:name="com.app.damnvulnerablebank.MainActivity"/>
<activity android:name="com.app.damnvulnerablebank.SplashScreen">
 ▼<intent-filter>
    <action android:name="android.intent.action.MAIN"/>
    <category android:name="android.intent.category.LAUNCHER"/>
  </intent-filter>
</activity>
<meta-data android:name="preloaded_fonts" android:resource="@array/preloaded_fonts"/>
<activity android:exported="true" android:name="androidx.biometric.DeviceCredentialHandlerActivity" androic
<activity android:excludeFromRecents="true" android:exported="true" android:launchMode="singleTask" androic
android:permission="com.google.firebase.auth.api.gms.permission.LAUNCH_FEDERATED_SIGN_IN" android:theme="@a
<service android:directBootAware="true" android:exported="false" android:name="com.google.firebase.componer
  <meta-data android:name="com.google.firebase.components:com.google.firebase.auth.FirebaseAuthRegistrar" a
```

Several activities are marked with android:exported="true", such as SendMoney, ViewBalance, and MainActivity. This means they can be invoked by external applications or services, potentially leading to unauthorized access or unintended interactions with sensitive functionality.

Figure 1.4: XML file traffic.



```
formBuildVersionCode="29" platformBuildVersionName="10">
ses-permission android:name="android.permission.INTERNET"/>
ses-permission android:name="android.permission.USE_BIOMETRIC"/>
ses-permission android:name="android.permission.USE_FINGERPRINT"/>
pplication android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreCompor
droid:label="@string/app_name" android:largeHeap="false" android:networkSecurityConfig="@xml/ne
droid:theme="@style/AppTheme" android:usesCleartextTraffic="true">
<activity android:name="com.app.damnvulnerablebank.Myprofile"/>
<activity android:name="com.app.damnvulnerablebank.CurrencyRates">
▼<intent-filter>
   <action android:name="android.intent.action.VIEW"/>
   <category android:name="android.intent.category.DEFAULT"/>
   <category android:name="android.intent.category.BROWSABLE"/>
```

The app allows cleartext traffic (HTTP) rather than forcing all communications to be encrypted (HTTPS). It is recommended to set android:usesCleartextTraffic="false" and ensure all communication uses HTTPS.

Vulnerabilities Identified:

1. **Insecure API Endpoints** - APIs do not properly check user permissions, allowing unauthorized access and actions through parameter tampering.
2. **Hardcoded Secrets** - Sensitive keys and credentials are embedded in the code and can be extracted via reverse engineering.
3. **Sensitive Information in Logs** - The app logs private data like session tokens and passwords, exposing them on rooted or debug-enabled devices.
4. **Exported Activities** - Activities are exposed without restrictions, allowing unauthorized external apps to trigger internal app functions.
5. **Insecure WebView via Deep Linking** - The app opens arbitrary URLs in WebView through deep links, which can be used for phishing or malicious content injection.
6. **Insecure Data Storage** - User data is stored in plaintext locally, making it accessible on compromised devices.
7. **Weak Root and Emulator Detection** - Root and emulator checks are basic and easily bypassed, allowing analysis in insecure environments.
8. **Improper SSL/TLS Validation** - The app accepts all SSL certificates, enabling interception of traffic through man-in-the-middle attacks.

III. Incident Response Process & Procedures

Please note that changes implemented to mitigate this issue should be tested in a controlled environment or a non-production environment.

A. PREPARATION

Implement robust security practices focused on securing mobile applications. This includes integrating industry standards such as the OWASP Mobile Top 10 into the app development lifecycle. Emphasize secure coding techniques and conduct security assessments using tools like MobSF, APKTool, and Burp Suite routinely.

To manage potential threats effectively, establish a dedicated incident response team composed of specialized roles. The team should include an Incident Commander to oversee the entire operation, Mobile Security Analysts to investigate and evaluate threats, App Developers responsible for patching and securing the code, a Communications Lead tasked with managing internal and public disclosures, and a Legal and Compliance Officer ensuring adherence to legal obligations and regulatory guidelines. In parallel, security policies are to be put in place requiring mandatory MFA, end-to-end encryption of sensitive data, and continuous threat detection via real-time monitoring system.

B. IDENTIFICATION

The vulnerability was identified during a controlled static security assessment of the Android application using Kali Linux. The application package (APK) was uploaded to the MobSF analysis platform via its dropbox interface for automated static analysis, followed by manual inspection using APKTool.

MobSF assigned the application a high-risk security rating, flagging multiple critical issues related to insecure configuration and sensitive data exposure. Among the most severe findings was the presence of an exposed XML configuration file embedded within the APK. The file contained plaintext sensitive information, including database connection details, internal API endpoints, and service-level credentials.

Additional MobSF warnings included:

1. Hardcoded secrets within application resources
2. Insecure storage of sensitive data
3. Lack of proper encryption mechanisms
4. Weak or missing access control on internal endpoints

Manual analysis using APKTool confirmed MobSF's findings and revealed that the XML file could be extracted by any party with access to the APK, enabling potential unauthorized access to backend databases and internal services. While no active exploitation was observable, the vulnerability posed a significant risk of data disclosure and unrestricted database access if the application were reverse engineered by a malicious actor.

Based on the automated risk score and corroborated manual analysis, the issue was classified as a high-severity sensitive information disclosure vulnerability resulting from insecure mobile application configuration practices.

C. CONTAINMENT

As the issue was discovered during a preemptive security assessment, containment actions focused on eliminating exposure prior to exploitation. Distribution of the affected APK version was halted, and the exposed XML configuration file was removed from the application package. All credentials identified by MobSF and confirmed through manual review were treated as compromised and scheduled for immediate rotation. Internal access to backend systems was temporarily restricted until credential changes were completed and verified.

D. ERADICATION

Eradication efforts addressed the root cause of the findings identified by MobSF. Sensitive configuration data was removed from the client-side application and relocated to secure backend services. Hardcoded credentials were eliminated and replaced with server-managed authentication mechanisms and short-lived tokens. Encryption was enforced for any remaining sensitive data, and secure secret management solutions were adopted to prevent sensitive information from being embedded in future builds. The application was rebuilt and reanalyzed using MobSF to confirm the removal of all previously flagged high-risk issues.

E. RECOVERY

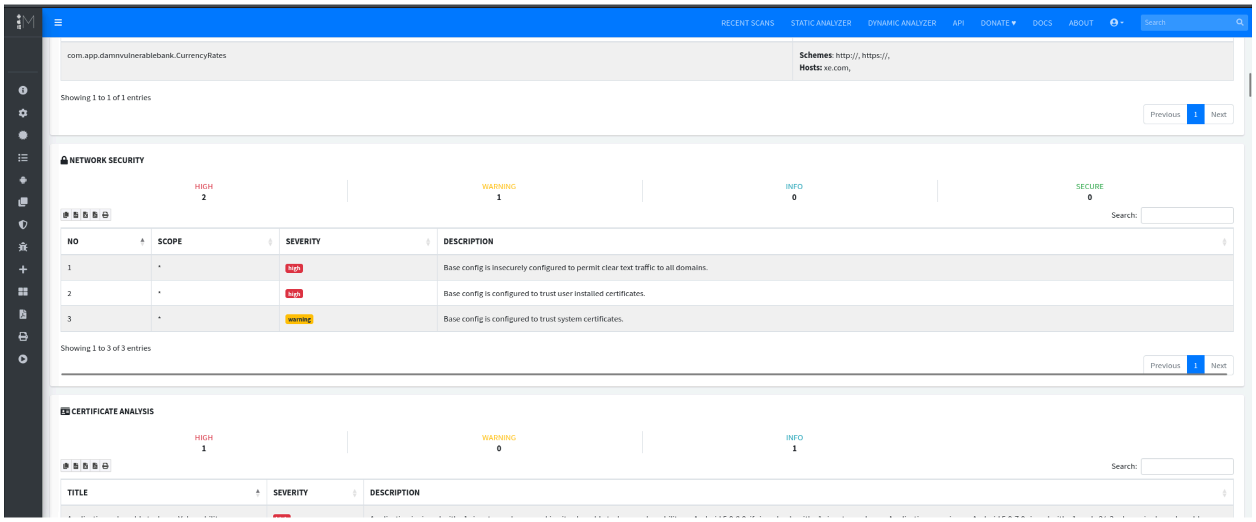
After remediation, the updated APK was resubmitted to MobSF for reassessment. The results indicated a significantly improved security score, with previously identified red flags either resolved or reduced to low-risk informational findings. Following successful validation, the application was approved for release. Security testing using MobSF was formally integrated into the CI/CD pipeline to ensure ongoing detection of similar issues before deployment.

F. LESSONS LEARNED

Some lessons were learned from this assessment. First, periodic security audits and static application security testing could have identified the exposed XML configuration file earlier and prevented the routine exploitation of similar vulnerabilities in production environments. Integrating automated tools such as MobSF into the development lifecycle allows security issues to be detected before release rather than after deployment. The findings also reinforced that hardcoding sensitive information, including database credentials and API keys, is strictly prohibited and that such secrets should be stored securely using centralized vault and backend-managed configuration mechanisms instead. Embedding sensitive data within an APK inherently increases the risk of disclosure due to the ease of reverse engineering.

Furthermore, adopting an attacker's perspective earlier in the QA process, through reverse engineering and manual APK inspection, can significantly enhance defensive measures prior to deployment. By simulating real-world attack techniques during testing, development teams can proactively address weaknesses and strengthen the overall security posture of the application.

IV. Appendices: Static Analysis Findings



com.app.damnvulnerablebank.CurrencyRates	**Schemes**: http://, https://, **Hosts:** xe.com,

Showing 1 to 1 of 1 entries

NETWORK SECURITY

HIGH	WARNING	INFO	SECURE
2	1	0	0

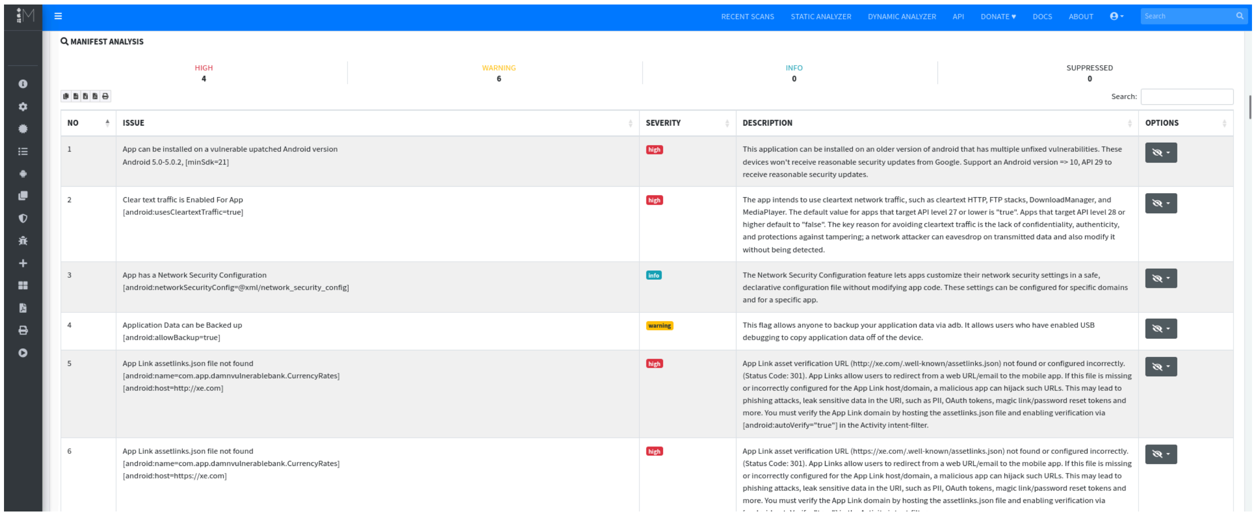
NO	SCOPE	SEVERITY	DESCRIPTION
1	*	high	Base config is insecurely configured to permit clear text traffic to all domains.
2	*	high	Base config is configured to trust user installed certificates.
3	*	warning	Base config is configured to trust system certificates.

Showing 1 to 3 of 3 entries

CERTIFICATE ANALYSIS

HIGH	WARNING	INFO
1	0	1

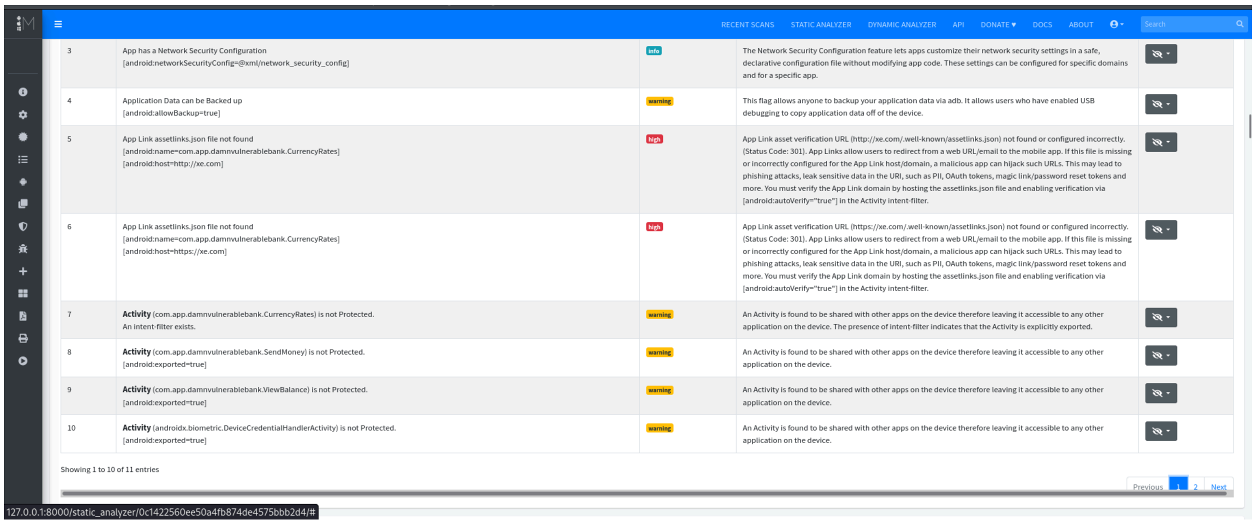
TITLE	SEVERITY	DESCRIPTION



MANIFEST ANALYSIS

HIGH	WARNING	INFO	SUPPRESSED
4	6	0	0

NO	ISSUE	SEVERITY	DESCRIPTION	OPTIONS
1	App can be installed on a vulnerable upatched Android version Android 5.0-5.0.2, [minSdk=21]	high	This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version => 10, API 29 to receive reasonable security updates.	
2	Clear text traffic is Enabled For App [android:usesCleartextTraffic=true]	high	The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and also modify it without being detected.	
3	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.	
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	App Link assetlinks.json file not found [android:name=com.app.damnvulnerablebank.CurrencyRates] [android:host=http://xe.com]	high	App Link asset verification URL (http://xe.com/.well-known/assetlinks.json) not found or configured incorrectly. (Status Code: 301). App Links allow users to redirect from a web URL/email to the mobile app. If this file is missing or incorrectly configured for the App Link host/domain, a malicious app can hijack such URLs. This may lead to phishing attacks, leak sensitive data in the URI, such as PII, OAuth tokens, magic link/password reset tokens and more. You must verify the App Link domain by hosting the assetlinks.json file and enabling verification via [android:autoVerify="true"] in the Activity intent-filter.	
6	App Link assetlinks.json file not found [android:name=com.app.damnvulnerablebank.CurrencyRates] [android:host=https://xe.com]	high	App Link asset verification URL (https://xe.com/.well-known/assetlinks.json) not found or configured incorrectly. (Status Code: 301). App Links allow users to redirect from a web URL/email to the mobile app. If this file is missing or incorrectly configured for the App Link host/domain, a malicious app can hijack such URLs. This may lead to phishing attacks, leak sensitive data in the URI, such as PII, OAuth tokens, magic link/password reset tokens and more. You must verify the App Link domain by hosting the assetlinks.json file and enabling verification via	

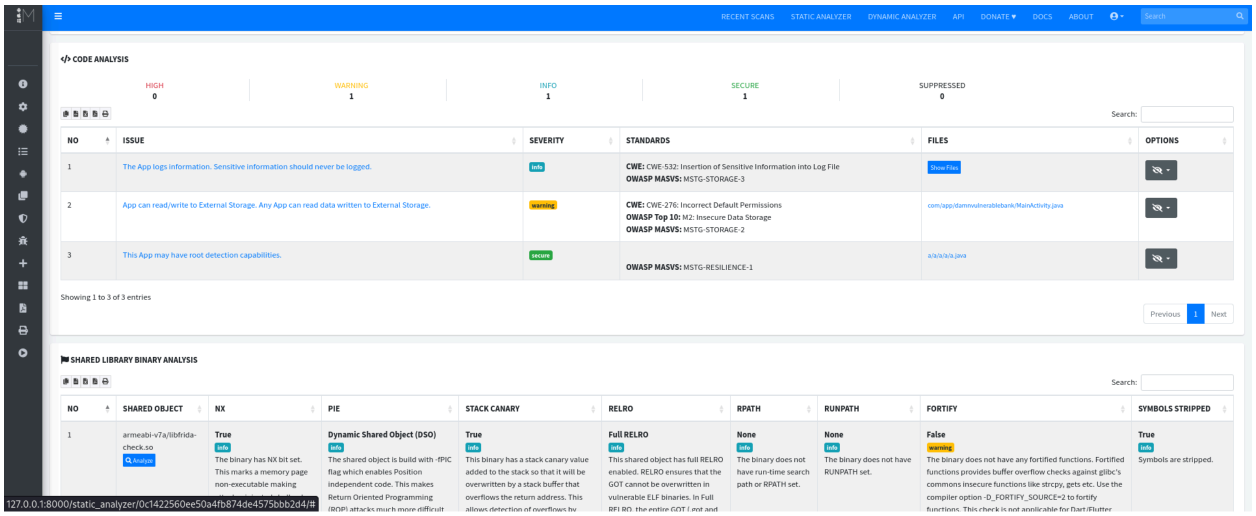


3	App has a Network Security Configuration [android:networkSecurityConfig=@xml/network_security_config]	info	The Network Security Configuration feature lets apps customize their network security settings in a safe, declarative configuration file without modifying app code. These settings can be configured for specific domains and for a specific app.	
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	App Link assetlinks.json file not found [android:name=com.app.damnvulnerablebank.CurrencyRates] [android:host=http://xe.com]	high	App Link asset verification URL (http://xe.com/.well-known/assetlinks.json) not found or configured incorrectly. (Status Code: 301). App Links allow users to redirect from a web URL/email to the mobile app. If this file is missing or incorrectly configured for the App Link host/domain, a malicious app can hijack such URLs. This may lead to phishing attacks, leak sensitive data in the URI, such as PII, OAuth tokens, magic link/password reset tokens and more. You must verify the App Link domain by hosting the assetlinks.json file and enabling verification via [android:autoVerify="true"] in the Activity intent-filter.	
6	App Link assetlinks.json file not found [android:name=com.app.damnvulnerablebank.CurrencyRates] [android:host=https://xe.com]	high	App Link asset verification URL (https://xe.com/.well-known/assetlinks.json) not found or configured incorrectly. (Status Code: 301). App Links allow users to redirect from a web URL/email to the mobile app. If this file is missing or incorrectly configured for the App Link host/domain, a malicious app can hijack such URLs. This may lead to phishing attacks, leak sensitive data in the URI, such as PII, OAuth tokens, magic link/password reset tokens and more. You must verify the App Link domain by hosting the assetlinks.json file and enabling verification via [android:autoVerify="true"] in the Activity intent-filter.	
7	**Activity** (com.app.damnvulnerablebank.CurrencyRates) 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.	
8	**Activity** (com.app.damnvulnerablebank.SendMoney) is not Protected. [android:exported=true]	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.	
9	**Activity** (com.app.damnvulnerablebank.ViewBalance) is not Protected. [android:exported=true]	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.	
10	**Activity** (androidx.biometric.DeviceCredentialHandlerActivity) is not Protected. [android:exported=true]	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.	

Showing 1 to 10 of 11 entries

Previous 1 2 Next

127.0.0.1:8000/static_analyzer/0c1422560ee50a4fb874de4575bbb2d4/#



</> CODE ANALYSIS

HIGH	WARNING	INFO	SECURE	SUPPRESSED
0	1	1	1	0

Search:

NO	ISSUE	SEVERITY	STANDARDS	FILES	OPTIONS
1	The App logs information. Sensitive information should never be logged.	info	**CWE:** CWE-532: Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	Show Files	
2	App can read/write to External Storage. Any App can read data written to External Storage.	warning	**CWE:** CWE-276: Incorrect Default Permissions **OWASP Top 10:** M2: Insecure Data Storage **OWASP MASVS:** MSTG-STORAGE-2	com/app/damnvulnerablebank/MainActivity.java	
3	This App may have root detection capabilities.	secure	**OWASP MASVS:** MSTG-RESILIENCE-1	a/a/a/a/a.java	

Showing 1 to 3 of 3 entries

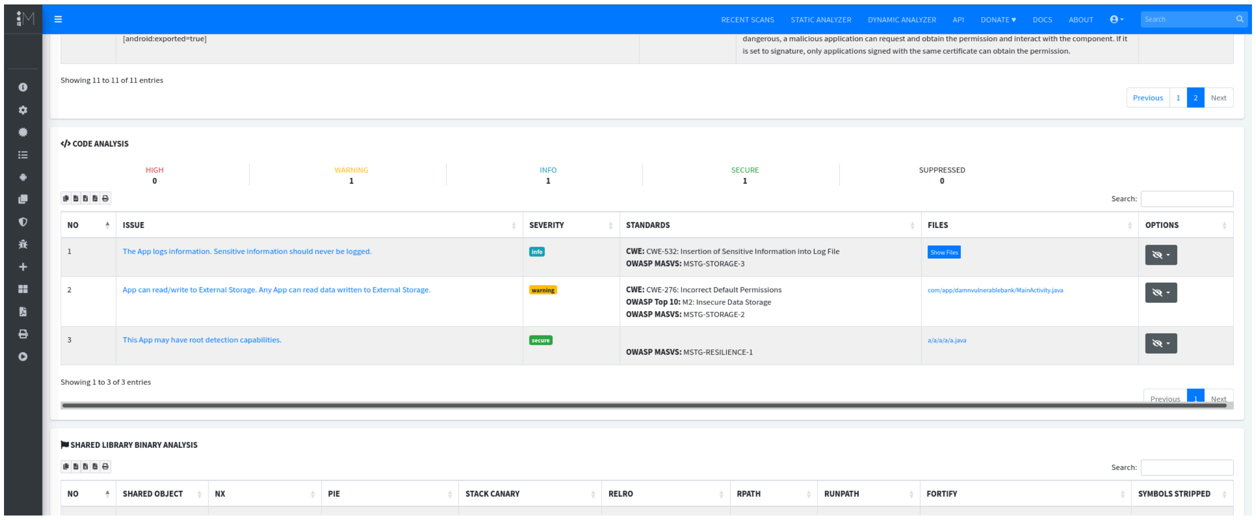
Previous 1 Next

SHARED LIBRARY BINARY ANALYSIS

Search:

NO	SHARED OBJECT	NX	PIE	STACK CANARY	RELRO	RPATH	RUNPATH	FORTIFY	SYMBOLS STRIPPED
1	armeabi-v7a/libfrida-check.so Analyze	**True** info The binary has NX bit set. This marks a memory page non-executable making	**Dynamic Shared Object (DSO)** info The shared object is build with -fPIC flag which enables Position independent code. This makes Return Oriented Programming (ROP) attacks much more difficult	**True** info This binary has a stack canary value added to the stack so that it will be overwritten by a stack buffer that overflows the return address. This allows detection of overflows by	**Full RELRO** info This shared object has full RELRO enabled. RELRO ensures that the GOT cannot be overwritten in vulnerable ELF binaries. In Full RELRO, the entire GOT (.got and	**None** info The binary does not have run-time search path or RPATH set.	**None** info The binary does not have RUNPATH set.	**False** warning The binary does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets etc. Use the compiler option -D_FORTIFY_SOURCE=2 to fortify functions. This check is not applicable for Dart/Flutter	**True** info Symbols are stripped.

127.0.0.1:8000/static_analyzer/0c1422560ee50a4fb874de4575bbb2d4/#



	[android:exported=true]		dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.	

Showing 11 to 11 of 11 entries

Previous 1 2 Next

</> CODE ANALYSIS

HIGH	WARNING	INFO	SECURE	SUPPRESSED
0	1	1	1	0

Search:

NO	ISSUE	SEVERITY	STANDARDS	FILES	OPTIONS
1	The App logs information. Sensitive information should never be logged.	info	**CWE:** CWE-532: Insertion of Sensitive Information into Log File **OWASP MASVS:** MSTG-STORAGE-3	Show Files	
2	App can read/write to External Storage. Any App can read data written to External Storage.	warning	**CWE:** CWE-276: Incorrect Default Permissions **OWASP Top 10:** M2: Insecure Data Storage **OWASP MASVS:** MSTG-STORAGE-2	com/app/damnvulnerablebank/MainActivity.java	
3	This App may have root detection capabilities.	secure	**OWASP MASVS:** MSTG-RESILIENCE-1	a/a/a/a/a.java	

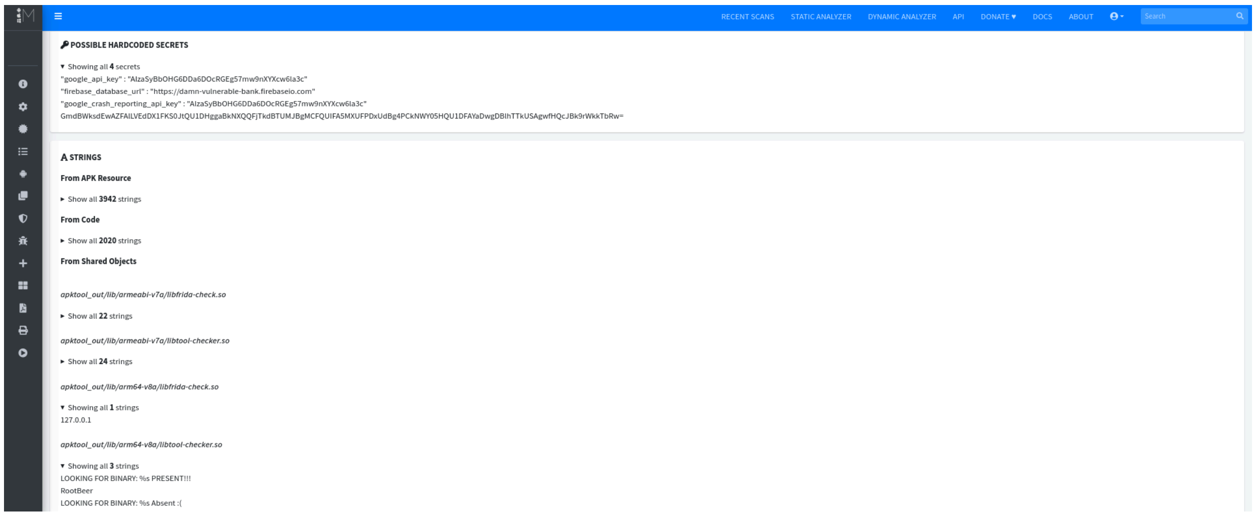
Showing 1 to 3 of 3 entries

Previous 1 Next

SHARED LIBRARY BINARY ANALYSIS

Search:

NO	SHARED OBJECT	NX	PIE	STACK CANARY	RELRO	RPATH	RUNPATH	FORTIFY	SYMBOLS STRIPPED



POSSIBLE HARDCODED SECRETS

▾ Showing all **4** secrets
"google_api_key" : "AIzaSyBbOHG6DDa6DOcRGEg57mw9nXYXcw6la3c"
"firebase_database_url" : "https://damn-vulnerable-bank.firebaseio.com"
"google_crash_reporting_api_key" : "AIzaSyBbOHG6DDa6DOcRGEg57mw9nXYXcw6la3c"
GmdBWksdEwAZFAlLVEdDX1FKS0JtQU1DHggaBkNXQQFjTkdBTUMJBgMCFQUIFA5MXUFPDxUdBg4PCkNWY05HQU1DFAYaDwgDBlhTTkUSAgwfHQcJBk9rWkkTbRw=

STRINGS

From APK Resource

▸ Show all **3942** strings

From Code

▸ Show all **2020** strings

From Shared Objects

apktool_out/lib/armeabi-v7a/libfrida-check.so

▸ Show all **22** strings

apktool_out/lib/armeabi-v7a/libtool-checker.so

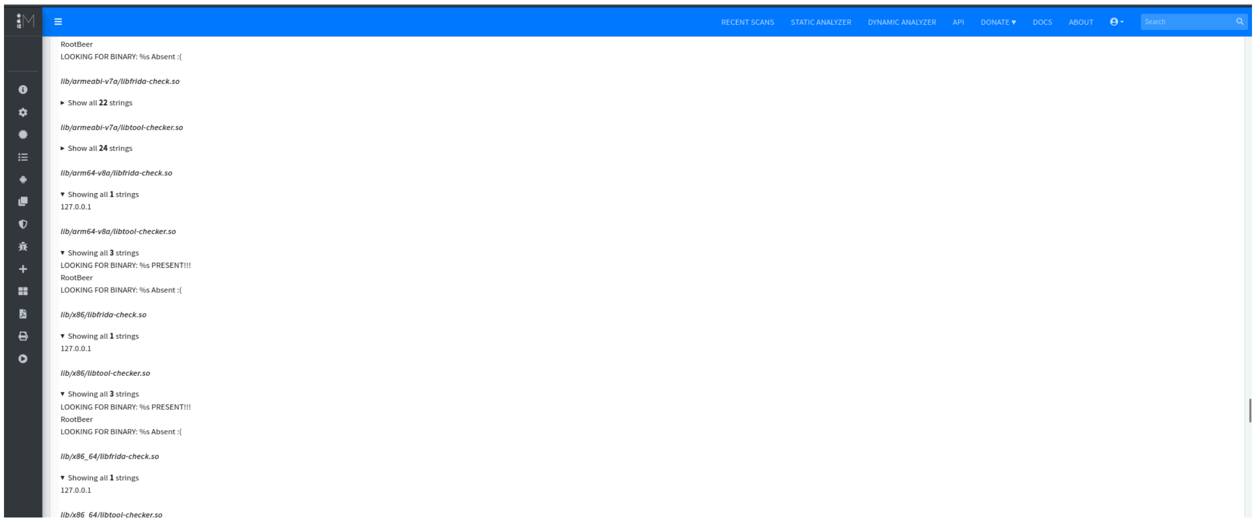
▸ Show all **24** strings

apktool_out/lib/arm64-v8a/libfrida-check.so

▾ Showing all **1** strings
127.0.0.1

apktool_out/lib/arm64-v8a/libtool-checker.so

▾ Showing all **3** strings
LOOKING FOR BINARY: %s PRESENT!!!
RootBeer
LOOKING FOR BINARY: %s Absent :(



RootBeer
LOOKING FOR BINARY: %s Absent :(

lib/armeabi-v7a/libfrida-check.so

▸ Show all **22** strings

lib/armeabi-v7a/libtool-checker.so

▸ Show all **24** strings

lib/arm64-v8a/libfrida-check.so

▾ Showing all **1** strings
127.0.0.1

lib/arm64-v8a/libtool-checker.so

▾ Showing all **3** strings
LOOKING FOR BINARY: %s PRESENT!!!
RootBeer
LOOKING FOR BINARY: %s Absent :(

lib/x86/libfrida-check.so

▾ Showing all **1** strings
127.0.0.1

lib/x86/libtool-checker.so

▾ Showing all **3** strings
LOOKING FOR BINARY: %s PRESENT!!!
RootBeer
LOOKING FOR BINARY: %s Absent :(

lib/x86_64/libfrida-check.so

▾ Showing all **1** strings
127.0.0.1

lib/x86_64/libtool-checker.so

REFERENCES:

Abraham, A. (2021, March 15). *MobSF/Mobile-Security-Framework-MobSF*. GitHub. https://github.com/MobSF/Mobile-Security-Framework-MobSF

Alberts, S. (2023, July 6). *Wireshark: A powerful tool for cybersecurity analysis*. LinkedIn. https://www.linkedin.com/pulse/wireshark-powerful-tool-cybersecurity-analysis-shanneece-alberts

AppSecWarrior. (2019, October 16). *SSL pinning: The right way to secure app*. Medium. https://medium.com/@appsecwarrior/ssl-pinning-the-right-way-to-secure-app-89ff82bdcb7a

Blair, M. A., Raderman, L., Lerchey, J., & Carnegie Mellon University. (2014). *Computer security incident response plan* (pp. 2–11). https://www.cmu.edu/iso/governance/procedures/incidentresponseplanv17.pdf

Crudu, A., & MoldStud Research Team. (2024, March 30). *Mobile App Security Incident Response Plan*. MoldStud. https://moldstud.com/articles/p-mobile-app-security-incident-response-plan

GeeksforGeeks. (2025, February 21). *What is BURP suite?* https://www.geeksforgeeks.org/what-is-burp-suite/

Google. (n.d.). *API key best practices*. Google Cloud. https://support.google.com/googleapi/answer/6310037

Guardsquare. (2022, October 25). *Reinforce your mobile app security with OWASP MASVS recommendations*. https://www.guardsquare.com/blog/mobile-app-security-owasp-masvs-recommendations

How do you use Metasploit for penetration testing on a mobile device? (2024, December 2). *Cyberly*. https://www.cyberly.org/en/how-do-you-use-metasploit-for-penetration-testing-on-a-mobile-device/index.html

Malik, K. (2023b, September 26). *OWASP Mobile app security checklist: 6 easy step*. Astra. https://www.getastra.com/blog/mobile/owasp-mobile-app-security-checklist/

Microsoft. (n.d.). *What is multifactor authentication?* Microsoft Support. https://support.microsoft.com/en-us/topic/what-is-multifactor-authentication-e5e39437-121c-be60-d123-eda06bddf661

Mitra, A. (2023, March 16). *Here are some tips on how to use BURP suite for Android app penetration testing*. LinkedIn. https://do.linkedin.com/posts/abhishekmitra1989_here-are-some-tips-on-how-to-use-burp-suite-activity-7042001690606129152-5cE9

Mobile App Security Testing: SQL Injection Vulnerabilities in Mobile Applications. (n.d.). *Cursa*. https://cursa.app/en/page/mobile-app-security-testing-sql-injection-vulnerabilities-in-mobile-applications

Priya, D. (2023, September 10). *Top 10 GitHub hacking tools for Android*. Analytics Insight. https://www.analyticsinsight.net/latest-news/top-10-github-hacking-tools-for-android

rewanthtammana. (2020, November 7). *GitHub - rewanthtammana/Damn-Vulnerable-Bank: Damn Vulnerable Bank is designed to be an intentionally vulnerable android application. This provides an interface to assess your android application security hacking skills*. GitHub. https://github.com/rewanthtammana/Damn-Vulnerable-Bank

Tumbleson, C. (2021, February 14). *iBotPeaches/Apktool*. GitHub. https://github.com/iBotPeaches/Apktool

What is deep linking and why are deep links important? (n.d.). *Adjust*. https://www.adjust.com/glossary/deep-linking/

What is OWASP? What is the OWASP Top 10? | Cloudflare. (n.d.). *Cloudflare*. https://www.cloudflare.com/learning/security/threats/owasp-top-10/