



**Università
di Genova**

Mobile Security Exam

A.U. Martin Martuccio

Contents

1	Introduction	1
1.1	Aims	1
1.2	Methodology and tools	2
2	APP1 Analysis	3
2.1	Static Analysis	3
2.1.1	Certificate Signature Analysis	3
2.1.2	Surface Analysis	4
2.1.3	Code Analysis	6
2.1.4	Trackers and Data Profiling	9
2.1.5	Hardcoded Strings	9
2.1.6	Comments and Solutions	10
2.1.7	General Considerations	10
2.2	Dynamic Analysis	10
2.2.1	Network Traffic Observations	10
2.2.2	Data Exposure and Risks	12
2.2.3	Traffic Manipulation	13
2.2.4	Drozer Analysis	14
2.2.5	Final Considerations:	17
3	APP2 Analysis	19
3.1	Static Analysis	19
3.1.1	Surface Analysis	19
3.1.2	Trackers and Third-Party Services	20
3.1.3	Hardcoded Strings	21
3.1.4	SHA-1 Collision Vulnerability	23
3.1.5	Warnings inside the application	23
3.1.6	SQL Injection Vulnerabilities	23
3.1.7	WebView Configuration Vulnerabilities	24
3.2	Dynamic Analysis	26
3.2.1	Network Traffic Analysis	26
3.2.2	Recommendations	27
3.2.3	Drozer Analysis	28

Chapter 1

Introduction

This report presents a security assessment of two mobile applications, referred to as **APP1** and **APP2**, aimed at identifying and addressing potential vulnerabilities that could compromise their security. The analysis consists of the following stages:

1. **APK Download:** The first step involves obtaining the APK file of the application, which will serve as the basis for all subsequent analyses.
2. **Static Analysis:** Using a range of security tools, we will perform a static analysis of the application.
3. **Dynamic Analysis:** This phase involves analyzing the application's behavior in a live environment, particularly focusing on its network interactions. By monitoring network traffic, we will assess how the application communicates externally and identify any data exposure risks or privacy concerns.
4. **Vulnerability Contextualization and Verification:** Next, each identified vulnerability will be contextualized within the app's use case and functionality. We will verify the presence of these vulnerabilities and discuss potential remediation options, providing general security considerations and recommendations for the application.

For application testing, we utilized **Android Studio** to create a virtual device with Android version 31, running on an x86_64 architecture, allowing precise testing within a simulated mobile environment.

1.1 Aims

The aims of this report is focuses on the security assessment of two mobile applications, **APP1** and **APP2**.

Application	Description
APP1	A fitness app.
APP2	A GPS-based locator.

Table 1.1: Applications Analyzed

1.2 Methodology and tools

The following tools were employed to conduct a comprehensive security assessment of the applications:

- **MobSF (Mobile Security Framework):** MobSF is a powerful framework for automated static and dynamic analysis of mobile applications. It was used to decompile APK files, scan for common security vulnerabilities, and provide a comprehensive report on potential risks in the application code and configuration.
- **APKTool:** Enabled decompilation and recompilation of APK files to inspect and modify application resources and code.
- **Magisk:** A tool for rooting Android devices and emulators in a systemless way without affecting the Android operating system's integrity.
- **Android Studio (Emulator):** The Android Studio Emulator provided a virtual environment to install and run the applications under controlled conditions.
- **Charles Proxy:** Intercepted and analyzed HTTP/S traffic to identify potential data exposure risks during dynamic analysis.
- **Drozer:** Assessed Android components (e.g., activities, services, content providers) for exploitable vulnerabilities.

Each tool contributed to different stages of the security assessment, from initial static analysis to in-depth exploration of network behavior and application vulnerabilities. This combination of tools allowed for a comprehensive evaluation, uncovering security weaknesses that could pose risks to user data.

Chapter 2

APP1 Analysis

2.1 Static Analysis

The static analysis was conducted using the Mobile Security Framework (MobSF), which provided a detailed report on the application's components and potential vulnerabilities, along with their associated risk levels.

2.1.1 Certificate Signature Analysis

The following table presents key details regarding the signature certificate used for the application binary, as extracted from the MobSF analysis:

Property	Details (censored)
Binary is signed	Yes
v1 signature	False
v2 signature	True
v3 signature	True
v4 signature	False
X.509 Subject	C=US, ST=C*****, L=M*****, O=Google Inc.
Signature Algorithm	rsassa_pkcs1v15
Valid From	2019-02-26 0*.*:52+00:00
Valid To	2049-02-26 0*.*:52+00:00
Issuer	C=US, ST=C*****, L=M*****, O=Google Inc.
Serial Number	0x755abe87*****
Hash Algorithm	sha256
MD5	6b45234461aef0232c*****fff4a
SHA1	fc6da741bfe2*****
SHA256	4b1e3875316ca25ff*****
SHA512	dbd1244ce2825b*****
PublicKey Algorithm	rsa
Bit Size	4096
Fingerprint	g*****5
Unique Certificates Found	1

Table 2.1: Signature Certificate Details

The binary is signed with v2 and v3 signatures, which are modern and secure signing

schemes. The certificate employs a 4096-bit RSA key and SHA256 hashing algorithm, both of which align with industry standards for secure encryption.

2.1.2 Surface Analysis

Below is a summary view of the analysis and the score, including the security score assigned to the application by the tool. In the following sections, we will examine each identified issue in detail, assessing the associated risks, and proposing possible solutions.

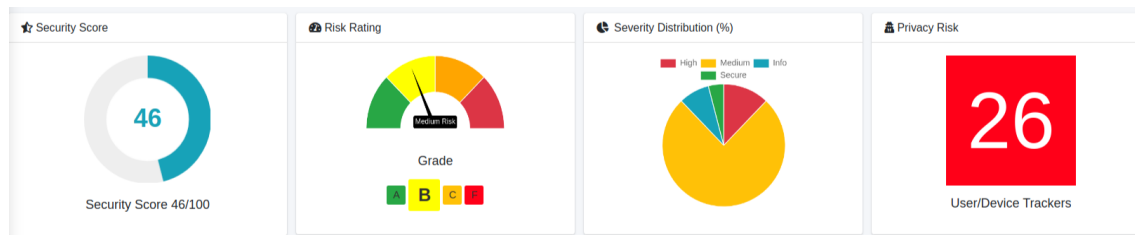


Figure 2.1: Summary of Static Analysis for APP1 (MobSF)

The static analysis revealed a medium security level, with notable privacy risks given the sensitive personal data handled by the application. MobSF identified six potential vulnerabilities, as shown in the following figure:

High	Domain config is insecurely configured to permit clear text traffic to these domains in scope	NETWORK
High	Clear text traffic is Enabled For App	MANIFEST
High	The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks.	CODE
High	Remote WebView debugging is enabled.	CODE
High	The file or SharedPreferences is World Writable. Any App can write to the file	CODE
High	Application contains Privacy Trackers	TRACKERS

Figure 2.2: High level issues

The application requires this permission, as listed in the following figure:

Permission	Info
android.permission.ACCESS_COARSE_LOCATION	coarse (network-based) location
android.permission.ACCESS_FINE_LOCATION	fine (GPS) location
android.permission.ACTIVITY_RECOGNITION	recognize physical activity
android.permission.POST_NOTIFICATIONS	post notifications
android.permission.READ_CONTACTS	read contact data
android.permission.READ_PHONE_STATE	read phone state and identity
android.permission.SYSTEM_ALERT_WINDOW	display system-level alerts
android.permission.WRITE_EXTERNAL_STORAGE	R/M/D external storage contents
com.google.android.gms.permission.ACTIVITY_RECOGNITION	recognize physical activity

Table 2.2: Permissions for APP1 application

The listed permissions align with the application's functionality, as they enable access to features essential for performing the specific tasks the app is designed for.

Manifest Analysis

During the manifest analysis, we discovered that the application is configured to allow clear-text network traffic. This is considered a high severity level because a network attacker can eavesdrop on transmitted data and even modify it without detection.

```

95 <application android:allowBackup="true" android:appComponentFactory="androidx.core.app.CoreComponentFactory"
96     android:extractNativeLibs="true" android:hardwareAccelerated="true" android:icon="@mipmap/notification_icon"
97     android:label="@string/app_name" android:name="com.ttn2r.MainApplication" android:networkSecurityConfig="@xml/network_security_config"
98     android:requestLegacyExternalStorage="true" android:supportsRtl="true" android:theme="@style/BootTheme" android:usesCleartextTraffic="true">

```

Figure 2.3: Manifest Analysis: Clear-Text Traffic Configuration

Additionally, the XML sources include a list of suspicious domains permitted to exchange data in clear text, further exacerbating the risk.

```

xml > network_security_config.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <network-security-config>
3 <domain-config cleartextTrafficPermitted="true">
4 <domain includeSubdomains="true">10.0.2.2/</domain>
5 <domain includeSubdomains="true">10.0.3.2/</domain>
6 <domain includeSubdomains="true">localhost/</domain>
7 <domain includeSubdomains="true">127.0.0.1/</domain>
8 <domain includeSubdomains="true">samplicio.us/</domain>
9 <domain includeSubdomains="true">cint.com/</domain>
10 <domain includeSubdomains="true">qualtrics.com/</domain>
11 <domain includeSubdomains="true">spectrumsurveys.com/</domain>
12 <domain includeSubdomains="true">decipherinc.com/</domain>
13 <domain includeSubdomains="true">ssisurveys.com/</domain>
14 <domain includeSubdomains="true">lucidhq.com/</domain>
15 <domain includeSubdomains="true">paradigmsample.com/</domain>
16 <domain includeSubdomains="true">focusvision.com/</domain>
17 <domain includeSubdomains="true">vi.ga/</domain>
18 <domain includeSubdomains="true">opinionetwork.com/</domain>
19 <domain includeSubdomains="true">surveyrouter.com/</domain>
20 <domain includeSubdomains="true">opinionbar.com/</domain>
21 <domain includeSubdomains="true">prsrvy.com/</domain>
22 <domain includeSubdomains="true">ptrack1.com/</domain>
23 <domain includeSubdomains="true">globaltestmarket.com/</domain>
24 <domain includeSubdomains="true">eocucom.com/</domain>
25 <domain includeSubdomains="true">prodegemr.com/</domain>
26 <domain includeSubdomains="true">swagbucks.com/</domain>
27 <domain includeSubdomains="true">sgizmo.com/</domain>
28 <domain includeSubdomains="true">surveygizmo.com/</domain>
29 <domain includeSubdomains="true">reviewrobin.com/</domain>
30 <domain includeSubdomains="true">questmindshare.com/</domain>
31 <domain includeSubdomains="true">peanutlabs.com/</domain>
32 <domain includeSubdomains="true">marketknowledgesurveys.com/</domain>
33 <domain includeSubdomains="true">dubinterviewer.com/</domain>
34 <domain includeSubdomains="true">confermit.com/</domain>
35 <domain includeSubdomains="true">survia.com/</domain>
36 <domain includeSubdomains="true">insights.supply/</domain>
37 <domain includeSubdomains="true">roirocket.com/</domain>
38 </domain-config>
39 </network-security-config>

```

Figure 2.4: Analysis of suspicious domains allowing clear-text traffic

Exposed Components

During the analysis of the application configuration, several warnings regarding exposed components were found, which can lead to potential security risks.

- **Application Data Backup Enabled** (android:allowBackup="true") : Allows data backup via Android Debug Bridge (ADB), posing a risk if unauthorized users access the device.

Application Data can be Backed up
[android:allowBackup=true]

warning

Figure 2.5: Analysis of Exposed Components and Related Risks

- **Broadcast Receiver Exposed** (android:exported="true"): The Broadcast Receiver, in this case `com.*****.*****.StepServiceReceiver`, can be accessed by other apps, potentially leading to data leakage or unauthorized actions.

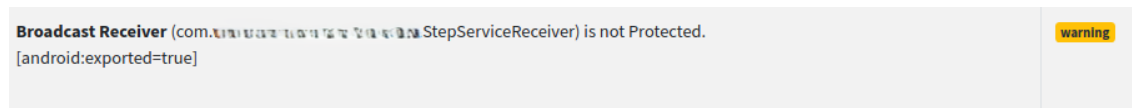


Figure 2.6: Analysis of Exposed Components and Related Risks

- **Activity Alias Exposed** (android:exported="true"): Several Activity Aliases, including `com.*****.MainActivityStreak500`, `com.*****.MainActivityHalloween`, and `com.*****.MainActivityDefault`, are accessible, making them susceptible to unauthorized launches.

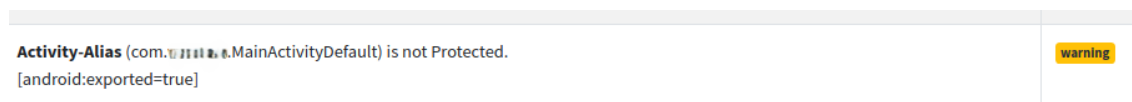


Figure 2.7: Analysis of Exposed Components and Related Risks

Risks Associated with Exposed Components

The risks associated with exposed components include:

- **Data Leakage:** Unauthorized backup of app data can lead to exposure of sensitive information.
- **Unauthorized Actions:** Exposed components can be exploited by other apps to manipulate app behavior or access restricted functions.

2.1.3 Code Analysis

The ability to back up internal application data via ADB was demonstrated. While this is not considered high-risk, as it requires physical access and developer mode, it highlights a potential attack vector.



Figure 2.8: Backup through adb

In the previous picture, we show the commands used and the request for backup from the device.

WebView Debugging Enabled

WebView Debugging is enabled in the application, which is not recommended for production builds. According to OWASP guidelines, this feature should be disabled to prevent potential exploitation.



Figure 2.9: WebView risk

MITM Attack Vulnerability

A Man-in-the-Middle (MITM) attack is possible due to the lack of SSL certificate pinning. This allows an attacker to intercept and manipulate network traffic (against WebView). In general the application implement the SSL certificate, but it can be bypassed, allowing attackers to intercept and manipulate network traffic.



Figure 2.10: MITM risk

CBC Encryption with PKCS5/PKCS7 Padding

The application uses CBC encryption with PKCS5/PKCS7 padding, which is vulnerable to padding oracle attacks. An attacker can exploit this to decrypt sensitive data without the key - oracle attack. The attacker can deduce information from the system based on the way it handles padding during decryption, thus allowing the cipher text to be deciphered without the key.

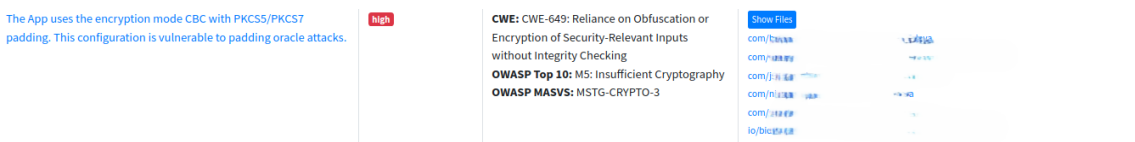


Figure 2.11: CBC risk

Solution: To mitigate padding oracle attacks, implement **Message Authentication Code (MAC)** or use the **Encrypt-then-MAC** construction. This ensures data integrity and reduces vulnerabilities.

Electronic Codebook (ECB) Mode

The application uses ECB mode for encryption, which is inherently insecure. ECB mode leaks information about the plaintext, as identical plaintext blocks produce identical ciphertext blocks.

The App uses ECB mode in Cryptographic encryption algorithm. ECB mode is known to be weak as it results in the same ciphertext for identical blocks of plaintext.	High	CWE: CWE-327: Use of a Broken or Risky Cryptographic Algorithm OWASP Top 10: M5: Insufficient Cryptography OWASP MASVS: MSTG-CRYPTO-2	com.martintuccio:app1
---	------	---	-----------------------

Figure 2.12: ECB risk

Encrypting the same block of plaintext using ECB mode always yields the same block of ciphertext. This can allow an attacker to:

- detect whether two ECB-encrypted messages are identical;
- detect whether two ECB-encrypted messages share a common prefix;
- detect whether two ECB-encrypted messages share other common substrings, as long as those substrings are aligned at block boundaries;
- detect whether (and where) a single ECB-encrypted message contains repetitive data (such as long runs of spaces or null bytes, repeated header fields or coincidentally repeated phrases in text).

Solution: Replace ECB mode with a secure encryption mode such as CBC or CTR, combined with a Message Authentication Code (MAC) using the Encrypt-then-MAC construction.

Shared Preferences Permissions

The application allows other apps to write to its Shared Preferences, which can lead to data tampering, privacy breaches, and unauthorized actions.

The file or SharedPreferences is World Writable. Any App can write to the file	High	CWE: CWE-276: Incorrect Default Permissions OWASP Top 10: M2: Insecure Data Storage OWASP MASVS: MSTG-STORAGE-2	com.martintuccio:app1
--	------	---	-----------------------

Figure 2.13: Shared Preferences risk

Allowing one app to write to another app's Shared Preferences can lead to serious security issues, such as:

1. Data Tampering: Unauthorized apps can modify critical app data, leading to functionality issues.
2. Privacy Breach: Sensitive user data could be exposed or altered by malicious apps.
3. Unauthorized Actions: Malicious apps can trigger actions or configurations in the target app.

Solution: Restrict access to Shared Preferences using `MODE_PRIVATE` and encrypt sensitive data.

2.1.4 Trackers and Data Profiling

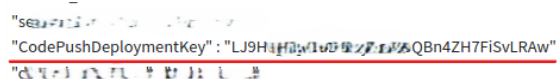
The application contains several trackers that may involve data profiling with third-party advertising and analytics companies. The following table lists all identified trackers, their categories, and links to detailed descriptions.

Table 2.3: List of detected Tracker

Tracker Name	Categories	URL
Adjust	Analytics	Link
Facebook Ads	Advertisement	Link
Facebook Analytics	Analytics	Link
Facebook Login	Identification	Link
Facebook Share		Link
AppLovin (MAX and SparkLabs)	Advertisement, Identification, Profiling, Analytics	Link
IAB Open Measurement	Advertisement, Identification	Link
Ogury Presage		Link
Didomi	Analytics	Link
Pangle	Advertisement	Link
Adjoe	Advertisement	Link
Appodeal Stack	Advertisement, Analytics	Link
BidMachine	Advertisement	Link
TapResearch	Profiling, Analytics	Link
Sentry	Crash reporting	Link
Google Firebase Analytics	Analytics	Link
Fyber	Advertisement	Link
Inmobi		Link
Unity3d Ads	Advertisement	Link
Amplitude	Profiling, Analytics	Link
ironSource	Analytics	Link
PubNative	Advertisement	Link
Tapjoy		Link
Mintegral	Advertisement, Analytics	Link
Google CrashLytics	Crash reporting	Link
Google AdMob	Advertisement	Link

2.1.5 Hardcoded Strings

Hardcoded strings, particularly sensitive ones like API keys, pose a significant security risk. The analysis identified several hardcoded strings which should not be present in the code.



```

"CodePushDeploymentKey": "LJ9H3f0y1u0B27.1.16AQBn4ZH7FiSvLRAW"

```

Figure 2.14: Deployment Key

2.1.6 Comments and Solutions

While most vulnerabilities are medium-risk, the extensive profiling of sensitive user data shared with third parties raises significant privacy concerns. The following measures are recommended:

- Disable clear-text traffic and enforce HTTPS.
- Disable WebView Debugging in production builds.
- Implement SSL certificate pinning to prevent MITM attacks.
- Replace insecure encryption modes (e.g., ECB, CBC with PKCS5/PKCS7 padding) with secure alternatives like GCM or Encrypt-then-MAC.
- Restrict Shared Preferences access using `MODE_PRIVATE` and encrypt sensitive data.
- Remove hardcoded strings from the codebase.
- Minimize third-party trackers and ensure compliance with data privacy regulations.

2.1.7 General Considerations

The APP1 application exhibits several security and privacy issues, many of which are common in modern mobile applications. While most vulnerabilities are medium-risk, the extensive data profiling and sharing with third parties pose significant risks to user privacy. Addressing these issues is essential to ensure user trust and compliance with privacy regulations.

2.2 Dynamic Analysis

2.2.1 Network Traffic Observations

For dynamic analysis, we used Charles Proxy. By configuring a proxy on port 8888, we intercepted the communication between the application and various servers in HTTP format, allowing us to inspect the content. Most traffic consisted of debug information and keep-alive messages. After installing a CA certificate on the device, we attempted to inspect HTTPS traffic as well.

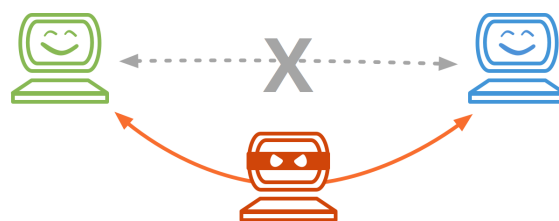


Figure 2.15: MITM attack

During HTTPS packet analysis, an error occurred, indicating that the application employs advanced techniques to prevent Man-in-the-Middle (MITM) attacks, such as certificate pinning.

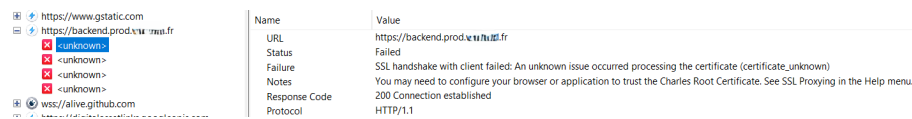


Figure 2.16: SSL Handshake Failure Due to Certificate Pinning

Why Does APP1 Use Certificate Pinning?

APP1, like many modern applications, likely uses certificate pinning to:

- Prevent MITM attacks by blocking tools like Charles Proxy from intercepting HTTPS traffic.
- Protect sensitive data by ensuring encrypted and authenticated communication.
- Comply with high-security standards and regulations.

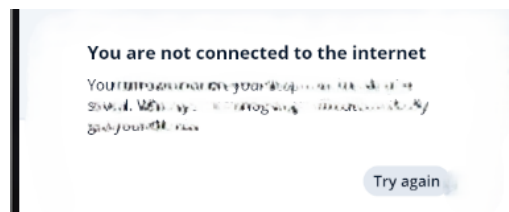


Figure 2.17: Application Behavior When Certificate Pinning is present

Bypassing Certificate Pinning

To bypass certificate pinning, we used Magisk modules, which allowed us to inspect and modify HTTPS traffic. Techniques for bypassing certificate pinning include:

- **Static Techniques:** Modifying the certificate or SSL validation methods within the app, such as repackaging the application.
- **Dynamic Techniques:** Using tools like Objection or Inspeckage to bypass pinning during runtime.

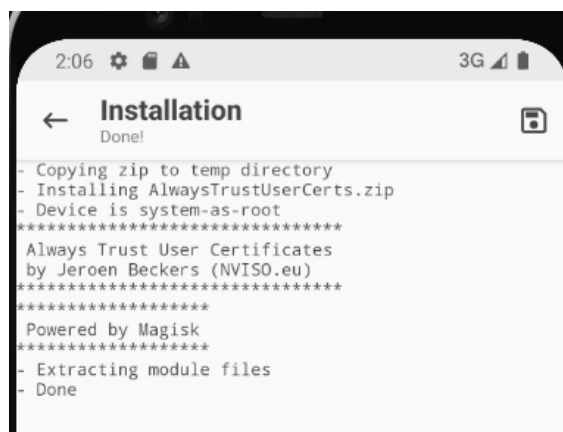


Figure 2.18: Bypassing Certificate Pinning with Magisk

After bypassing certificate pinning, we were able to fully analyze the application and modify the content of network packets.

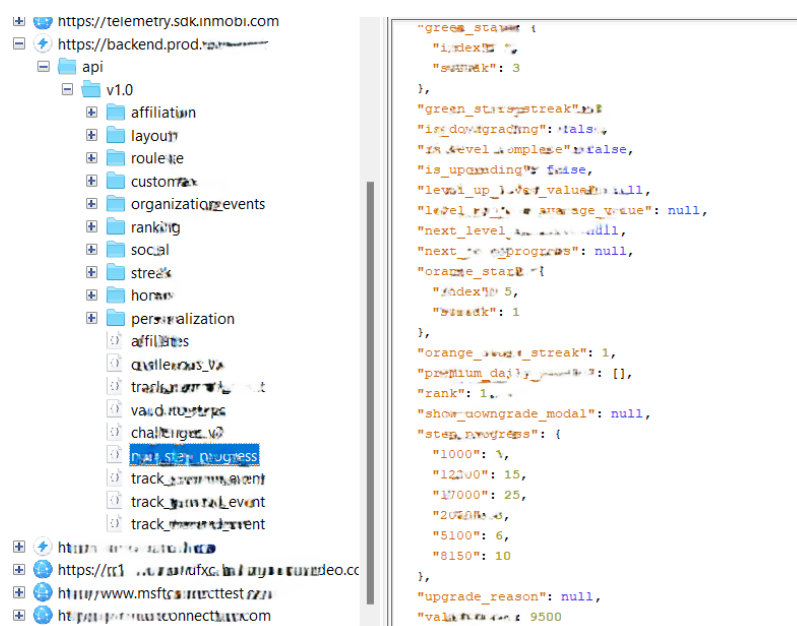


Figure 2.19: Intercepted HTTPS Traffic After Bypassing Certificate Pinning

2.2.2 Data Exposure and Risks

The ability to bypass certificate pinning and decrypt HTTPS traffic highlights a significant vulnerability. An attacker could exploit this to access sensitive data transmitted by the application, especially if the data is not further **encrypted at the application level**.

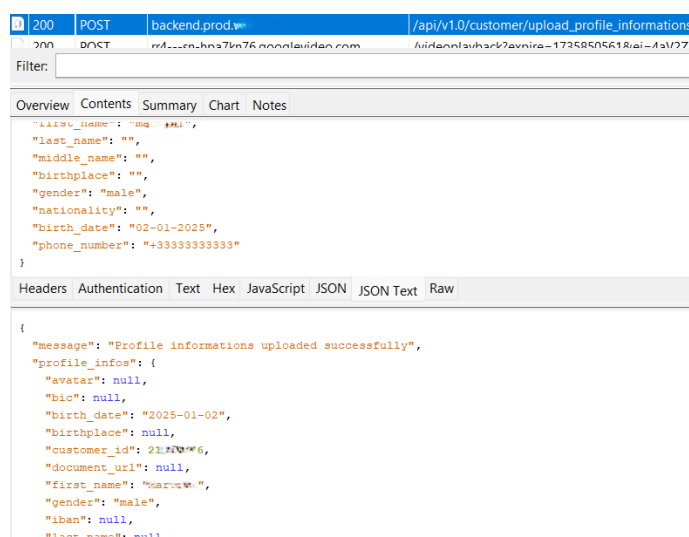


Figure 2.20: Exposure of Sensitive Data in HTTPS Traffic

Mitigation Strategies

To mitigate this vulnerability:

- Implement **end-to-end encryption** at the application level using symmetric (e.g. AES) or asymmetric (e.g. RSA) encryption.
- Use **data signing** to ensure integrity and authenticity.

2.2.3 Traffic Manipulation

We demonstrated how an attacker could modify HTTP/HTTPS traffic to alter requests and responses between the application and the server. For example, we modified the server's response to display a fake balance in the application.

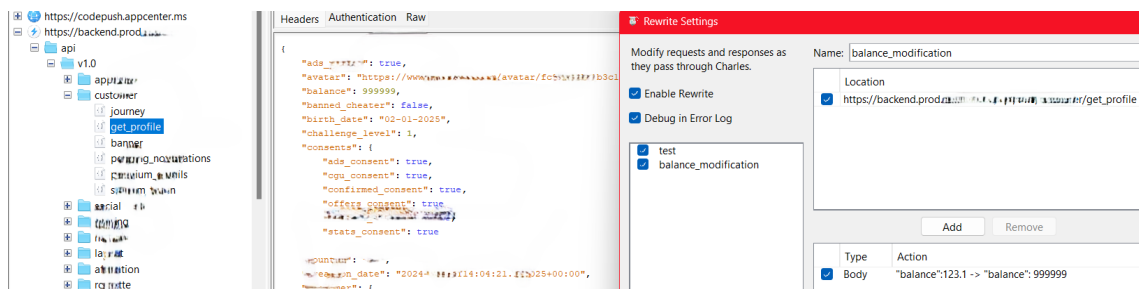


Figure 2.21: Modifying Network Traffic with Charles Proxy

Impact of Traffic Manipulation

An attacker could:

- Modify sensitive data (e.g., user balances) for illicit gains.
- Perform unauthorized transactions or manipulate app behavior.
- Compromise data integrity and application security.

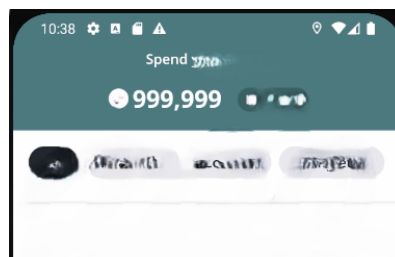


Figure 2.22: Modified balance

Solution:

Application-level encryption is a security technique where sensitive data is encrypted directly within the application before being transmitted or stored. Unlike transport-level encryption (e.g., HTTPS, which protects data during transmission), application-level encryption ensures end-to-end protection, safeguarding data even if the communication channel is compromised.

2.2.4 Drozer Analysis

We used Drozer to analyze the application's exposed components and permissions. Key findings include:

```
dz> run app.package.attacksurface com.VUJf1D9
Attempting to run shell module
Attack Surface:
  9 activities exported
  11 broadcast receivers exported
  0 content providers exported
  3 services exported
```

Figure 2.23: Application Surface Analysis with Drozer

We extracted some basic application information:

```
dz> run app.package.info -a com.*
Attempting to run shell module
Package: com.*
Application Label: null
Process Name: com.*
Version: 7.6.14
Data Directory: /data/user/0/com.*
APK Path: /data/app/~~Gzq5zUJdyzX0tKxxEjS==/com.*-0Uz8yht7URQuDW6_w_m2xA==/base.apk
UID: 10149
GID: [3003]
Shared Libraries: [/system/framework/org.apache.http.legacy.jar, /system_ext/framework/androidx
Shared User ID: null
Uses Permissions:
- android.permission.SYSTEM_ALERT_WINDOW
- android.permission.INTERNET
- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.REQUEST_COMPANION_RUN_IN_BACKGROUND
- android.permission.FOREGROUND_SERVICE
- android.permission.READ_PHONE_STATE
- android.permission.WRITE_INTERNAL_STORAGE
- android.permission.RECEIVE_BOOT_COMPLETED
- com.google.android.gms.permission.ACTIVITY_RECOGNITION
- android.permission.ACTIVITY_RECOGNITION
- android.permission.ACCESS_WIFI_STATE
- android.permission.ACCESS_NETWORK_STATE
```

Figure 2.24: Application Information Extracted with Drozer

Next, we analyzed the application's activities and their required permissions:

```
dz> run app.activity.info -a com.*
Attempting to run shell module
Package: com.*
com.*.MainActivity
  Permission: null
com.*.MainActivity$Alloween
  Permission: null
  Target Activity: com.*.MainActivity
com.*.PermissionsRationaleActivity
  Permission: null
com.*.ViewPermissionUsageActivity
  Permission: android.permission.VIEW_PERMISSION_USAGE
  Target Activity: com.*.PermissionsRationaleActivity
com.appsample.MainActivity
  Permission: null
com.facebook.CrashReportActivity
  Permission: null
com.google.firebase.auth.internal.GenericIdpActivity
  Permission: null
com.google.firebase.auth.internal.RecaptchaActivity
  Permission: null
net.openid.appauth.RedirectUriReceiverActivity
  Permission: null
```

Figure 2.25: Activity info

Database Security

We tested for potential SQL injection vulnerabilities, attempting both projection and selection attacks. No vulnerabilities were found, as shown below:

```
content://com.XXXXXX.provider  
content://com.XXXXXX.com.quarup.picasso/  
content://com.XXXXXX.provider/  
content://com.XXXXXX.reactnativefirebase.provider/  
content://com.XXXXXX.SentryPerformanceProvider/  
content://com.XXXXXX.AudienceNetworkContentProvider  
content://com.XXXXXX.provider.datadogrum  
content://com.XXXXXX.fileprovider/  
content://com.XXXXXX.reactnativesoftwarecrashlyticsinitprovider/  
content://com.XXXXXX.android-startup/  
content://com.XXXXXX.SentryInitProvider/  
content://com.XXXXXX.LifecycleObserverLifecycleProvider/  
content://com.XXXXXX.adsdk.contentprovider.share.preferences.  
content://com.XXXXXX.reactnativefirebasecrashlyticsinitprovider  
content://com.XXXXXX.google.sdk.contentprovider.database.AUTHORITY  
content://com.XXXXXX.androidx-stacup  
content://com.XXXXXX.applovinivprovider
```

Injection in Projection:
No vulnerabilities found.

Injection in Selection:
No vulnerabilities found.

Figure 2.26: SQL Injection Test Results

Content Provider Analysis

Drozer attempted to access multiple content providers but received no response, indicating they are either inactive, require specific permissions, or are protected by security mechanisms. This is a **positive finding**, as it suggests the application employs robust security measures to protect sensitive data.

[illegible]

Figure 2.27: Content Provider Analysis

Exposed Services

Drozer identified three exported services:

- `androidx.health.platform.client.impl.sdkservice.HealthDataSdkService`: Exported without requiring any permissions, posing a potential security risk.
- `com.google.android.gms.auth.api.signin.RevocationBoundService`: Protected by a specific permission.
- `androidx.work.impl.background.systemjob.SystemJobService`: Restricted to system-level access.

```
dz> run app.service.info -a com.
Attempting to run shell module
Package: com.
androidx.health.platform.client.impl.sdkservice.HealthDataSdkService
Permission: null
com.google.android.gms.auth.api.signin.RevocationBoundService
Permission: com.google.android.gms.auth.api.signin.permission.REVOCATION_NOTIFICATION
androidx.work.impl.background.systemjob.SystemJobService
Permission: android.permission.BIND_JOB_SERVICE
```

Figure 2.28: Exported Services Analysis

Security Implications

The `HealthDataSdkService` is part of Android's Health Data Platform and is designed to be exported. While it can be started by third-party apps, its implementation includes security mechanisms to protect user health data. However, the fact that it is exported without requiring permissions could represent a potential vulnerability if not adequately secured. An attacker could exploit this service to access sensitive functionalities or data.

```
(user@kali)-[~/Desktop]
$ adb shell am startservice -n com. /androidx.health.platform.client.impl.sdkservice.HealthDataSdkService
Starting service: Intent { cmp=com. /androidx.health.platform.client.impl.sdkservice.HealthDataSdkService }
```

Figure 2.29: Starting HealthDataSdkService

It is important to note that this potential vulnerability is not directly tied to the APP1 application but rather to the `HealthDataSdkService`, which is part of Android's Health Data Platform. This service is designed to be exported and used by other applications but includes security mechanisms to limit access and protect user health data.

```
* ServiceRecord{9ed0db8 u0 com. /androidx.health.platform.client.impl.sdkservice.HealthDataSdkService}
  intent={cmp=com. /androidx.health.platform.client.impl.sdkservice.HealthDataSdkService}
  packageName=com.
  processName=com.
  baseDir=/data/app/~/com. /base.apk
  dataDir=/data/user/0/com.
  app=ProcessRecord{672e89c 5059:com.}
  allowWhileInUseInFgs=true
  recentCallingPackage=com.android.shell
  allowStartForeground=51
  startForegroundCount=0
  infoAllowStartForeground=[callingPackage: com.android.shell; callingUid: 1; androidState: NONE; intent: Intent { cmp=com. /androidx.health.platform.client.impl.sdkservice.HealthDataSdkService }; code:SYSTRACE_ID; tempAllowListReason:<,reasonCode:SYSTEM_ALLOW_LISTED,duration:9223372036854775807 callingUid:-1; targetSdkVersion:33; callerSdkVersion:-1; startForegroundCount:0; bindFromPackage:null]
  createTime=5s287ms startingBgTimeout=
  lastActivity=5s278ms restartTime=5s278ms startFromFgs=true
  startRequested=true delayedStop=false stopIfKilled=true startId=1
```

Figure 2.30: HealthDataSdkService in Operation

The Health Data Platform library abstracts implementation details and safeguards user health data, as documented in the official documentation: [Health Data Platform Overview](#).

2.2.5 Final Considerations:

- **Drozer Analysis:** From the Drozer analysis, no critical vulnerabilities were identified. Activities, content providers, and databases are well-protected, and no SQL injection issues were found.
- **Charles Proxy Analysis:** The application exhibits two primary vulnerabilities:
 - **Cleartext Communication:** It communicates with certain domains in plaintext, exposing data to potential interception. As shown in Figure 2.4
 - **Lack of Application-Level Encryption:** Absence of end-to-end encryption at the application level could allow attackers, through social engineering, to install a malicious certificate authority (CA) on the victim's device, enabling data interception. Incorporating robust encryption mechanisms within the application is advised.

Overall, the application does not present critical security issues but addressing these vulnerabilities would enhance data protection.

Chapter 3

APP2 Analysis

3.1 Static Analysis

3.1.1 Surface Analysis

This section includes the surface analysis of the application, examining its structure and identifying the main components and potential attack vectors.

Signature Certificate Analysis

The following table presents key details regarding the signature certificate used for the application binary::

Property	Details
Binary is signed	Yes
v1 signature	False
v2 signature	False
v3 signature	True
v4 signature	False
X.509 Subject	C=**, ST=**, L=S*****, O=APP2, OU=Unknown
Signature Algorithm	rsassa_pkcs1v15
Valid From	200*_**_** 0*:25:31+00:00
Valid To	20**_**_** 0*:25:31+00:00
Issuer	C=US, ST=**, L=*****, O=APP2, OU=Unknown
Serial Number	0*****bd7
Hash Algorithm	sha1
MD5	8726*****8ff488927d
SHA1	19c0*****70c771d622b
SHA256	86a71b09ec*****3733f90197b131b92333e9a
SHA512	fad1807*****28ac723070c5335fd441
PublicKey Algorithm	rsa
Bit Size	1024
Fingerprint	d5da81*****+650b62a41c5bc4gy86d
Unique Certificates Found	1

Table 3.1: Signature Certificate Details

The certificate uses a 1024-bit RSA key with SHA1 hashing, which is considered outdated and less secure compared to modern standards (e.g., 2048-bit RSA with SHA256).

Security Score and High-Risk Findings

The application received a medium security score, but several high-risk issues were identified during the analysis.

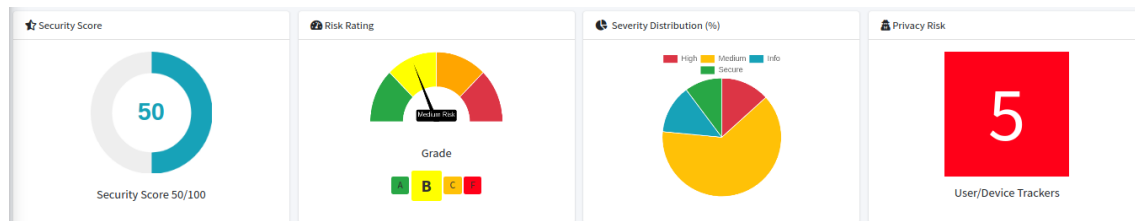


Figure 3.1: Summary of Static Analysis for APP2 (MobSF)

MobSF identified four potential vulnerabilities, as shown in the following figure:

High	Certificate algorithm vulnerable to hash collision	CERTIFICATE
High	The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks.	CODE
High	The file or SharedPreferences is World Writable. Any App can write to the file	CODE
High	Application contains Privacy Trackers	TRACKERS

Figure 3.2: High level issues

3.1.2 Trackers and Third-Party Services

The application integrates several third-party trackers for analytics, advertising, and crash reporting. The following table lists the identified trackers and their categories:

Tracker Name	Categories	URL
AppsFlyer	Analytics	Link
Branch	Analytics	Link
Google AdMob	Advertisement	Link
Google CrashLytics	Crash Reporting	Link
Google Firebase Analytics	Analytics	Link

Table 3.2: List of Detected Trackers

Required Permissions

The application requests several permissions, which are necessary for its functionality. The following table lists these permissions:

Even some api keys are exposed:

```
<meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version" />
<meta-data android:name="com.google.android.geo.API_KEY" android:value="AIzaSyCwPqrUf1rGF~uUfJChn7bgeCLwP6s" />
<meta-data android:name="com.facebook.sdk.ApplicationId" android:value="@string/facebook_app_id" />
<meta-data android:name="com.facebook.sdk.ClientToken" android:value="@string/facebook_client_token" />
```

Figure 3.4: Dangerous Permissions Requested

Recommendations

To address this issue:

- Remove sensitive hardcoded information from the codebase.
- Store sensitive data in secure storage or use environment variables.
- Use secure configuration management tools such as Firebase Remote Config.

Branch.io Integration

Branch.io is a platform used for deep linking, installation tracking, and campaign attribution. Although it provides valuable functionality, the exposure of its API keys in the application poses a security risk.

```
</receiver>
<meta-data android:name="io.branch.sdk.BranchKey" android:value="key_live_nfcJ6wW.WXRB1SropA8PmVexZ2oF" />
<meta-data android:name="io.branch.sdk.BranchKey.test" android:value="key_test_M1DzkuX0PkB7UuHmaoBrku66KJ" />
<receiver android:exported="false" android:name="com.branch.sdk.BranchKeyReceiver" android:permission="android.permission.BROADCAST_STICKY">
  <intent-filter>
```

Figure 3.5: API keys' Branch.io

Security Risks of Exposed Branch.io Keys

- Create fraudulent deep links targeting the application.
- Falsify installation or conversion data, compromising analytics accuracy.
- Overload the Branch.io API, leading to service disruptions or additional costs.
- Perform spoofing attacks to attribute fake campaigns or events.

Recommendations

- Use a secure back-end to handle API calls instead of embedding keys in the app.
- Obfuscate the keys to make them harder to extract.
- Monitor API usage for suspicious activity.
- Use separate keys for development and production environments.

3.1.4 SHA-1 Collision Vulnerability

The application uses SHA-1 for hashing in certain components, which is vulnerable to collision attacks. SHA-1 is considered cryptographically weak, and an attacker could exploit this vulnerability.

Certificate algorithm vulnerable to hash collision	High	Application is signed with SHA1withRSA. SHA1 hash algorithm is known to have collision issues.
--	------	--

Figure 3.6: SHA-1 Vulnerability

Recommendations

- Replace SHA-1 with a more secure hashing algorithm, such as SHA-256 or SHA-3.

3.1.5 Warnings inside the application

Several warnings are identified, including the use of deprecated permissions and insecure configurations. These warnings could lead to potential security vulnerabilities if not addressed.

1	App can be installed on a vulnerable Android version Android 9, minSdk=28	warning	This application can be installed on an older version of android that has multiple vulnerabilities. Support an Android version >= 10, API 29 to receive reasonable security updates.	Show File
3	Broadcast Receiver (com.appsflyer.MultiInstallBroadcastReceiver) is not Protected. [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.	Show File
4	TaskAffinity is set for activity (com.braze.push.NotificationTrampolineActivity)	warning	If taskAffinity is set, then other application could read the Intents sent to Activities belonging to another task. Always use the default setting keeping the affinity as the package name in order to prevent sensitive information inside sent or received Intents from being read by another application.	Show File
5	Service (androidx.work.impl.background.systemjob.SystemJobService) is Protected by a permission, but the protection level of the permission should be checked. Permissions: android.permission.BIND_JOB_SERVICE [android:exported=true]	warning	A Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.	Show File
6	Broadcast Receiver (androidx.work.impl.diagnostics.DiagnosticsReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permissions: com.google.android.permission.DUMP [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.	Show File
7	Broadcast Receiver (com.google.firebase.iid.FirebaseInstanceIdReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permissions: com.google.android.gms.permission.SEND [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.	Show File
8	Broadcast Receiver (androidx.profileinstaller.ProfileInstallReceiver) is Protected by a permission, but the protection level of the permission should be checked. Permissions: android.permission.DUMP [android:exported=true]	warning	A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.	Show File

Figure 3.7: Warnings

3.1.6 SQL Injection Vulnerabilities

The application may be vulnerable to SQL injection attacks if user input is not properly sanitized. SQL injection can allow an attacker to execute arbitrary SQL queries, future investigations will be carried out using other tools (Drozer).

5	App uses SQLite Database and execute raw SQL query. Untrusted user input in raw SQL queries can cause SQL Injection. Also sensitive information should be encrypted and written to the database.	warning	CWE: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') OWASP Top 10: M7: Client Code Quality	Show File
---	--	---------	--	-----------

Figure 3.8: SQL injection warning

The analysis of the application's source code, obtained through reverse engineering techniques, has revealed possible vulnerabilities related to SQL injection. The provided code snippets demonstrate instances where user input is directly incorporated into SQL queries.

```

@Override // vc.t.a
public final Object apply(Object obj) {
    vc.t tVar = (vc.t) this.f8358b;
    oc.t tVar2 = (oc.t) this.f8359c;
    lc.c cVar = vc.t.f78282f;
    tVar.getClass();
    Long k9 = vc.t.k((SQLiteDatabase) obj, tVar2);
    if (k9 == null) {
        return Boolean.FALSE;
    }
    Cursor rawQuery = tVar.h().rawQuery("SELECT 1 FROM events WHERE context_id = ? LIMIT 1", new String[]{k9.toString()});
    try {
        return Boolean.valueOf(rawQuery.moveToNext());
    } finally {
        rawQuery.close();
    }
}

```

Figure 3.9: SQL Injection code

In this snippet, the `rawQuery` method is used to execute a SQL query. The query string is concatenated with `k9.toString()`, which may contain user input or data derived from user input. If `k9` is not properly sanitized, an attacker could manipulate its value to inject malicious SQL code.

```

@Override // vc.t.a
public final Object apply(Object obj) {
    String str = (String) this.f78270b;
    c.a aVar = (c.a) this.f78271c;
    SQLiteDatabase sqLiteDatabase = (SQLiteDatabase) obj;
    boolean booleanValue = ((Boolean) t.a(sqLiteDatabase.rawQuery("SELECT 1 FROM log_event_dropped WHERE log_source = ? AND reason = ?", new String[]{str, Integer.toString(aVar.f69155a)}, new bf.b(3))).booleanValue());
    long j11 = this.f78269a;
    int i11 = aVar.f69155a;
    if (booleanValue) {

```

Figure 3.10: SQL Injection code 2

In this snippet, the `rawQuery` method appears to execute a SQL query with user-supplied input `str` and `AVar.f981588`. If these inputs are not properly sanitized, an attacker could inject malicious SQL code.

Recommendations

- Use parameterized queries or prepared statements to prevent SQL injection.
- Sanitize all user inputs to remove potentially malicious characters.
- Implement input validation to ensure data conforms to expected formats.

3.1.7 WebView Configuration Vulnerabilities

The application uses `WebView` with several insecure configurations, which could expose the application to significant security risks.

```

    WebSettings settings = getSettings();
    Intrinsics.d(settings, "settings");
    settings.setAllowUniversalAccessFromFileURLs(true);
    WebSettings settings2 = getSettings();
    Intrinsics.d(settings2, "settings");
    settings2.setJavaScriptEnabled(true);
    WebSettings settings3 = getSettings();
    Intrinsics.d(settings3, "settings");
    settings3.setDomStorageEnabled(true);
    setDownloadListener(new a(context));
    addJavascriptInterface(bVar, "AdaAndroid");
    Function0 function0 = this.webViewLoadingErrorCallback;
    if (function0 == null) {
        int i11 = defpackage.b.f6871e;
        function0 = defpackage.a.f1g;
    }
    setWebViewClient(new defpackage.b(function0, this.f72380m));
    setWebChromeClient(new cs0.b(this));

```

Figure 3.11: WebView Configuration code

Insecure WebView Settings

The following insecure WebView settings were identified:

- `setAllowUniversalAccessFromFileURLs(true)`: Allows file-based URLs to access content from any origin, posing a risk of unauthorized file access or malicious code execution.
- `setJavaScriptEnabled(true)`: Enables JavaScript execution, which can be exploited if the WebView loads untrusted content.
- `setDomStorageEnabled(true)`: Enables DOM storage, which could be abused to store malicious data or steal information.
- `addJavascriptInterface(bVar, "AdaAndroid")`: Exposes Java methods to JavaScript, creating a potential attack vector if the WebView loads untrusted content.

Recommendations

To secure the WebView:

- Disable universal access from file URLs
- Disable JavaScript if not required
- Disable DOM storage if not required
- Remove or secure the JavaScript interface
- Validate all resources and URLs loaded in the WebView.
- Use a custom WebViewClient to control URL loading:

```
webView.setWebViewClient(new CustomWebViewClient());
```

3.2 Dynamic Analysis

The dynamic analysis of the APP2 application involved examining network traffic and interactions between the application and its servers, focusing on data transmission security.

3.2.1 Network Traffic Analysis

We intercepted and analyzed data packets exchanged between the APP2 application and its servers. By installing a custom CA certificate, we decrypted HTTPS traffic. The analysis revealed that the application communicates with third-party services, such as Branch.io, using unencrypted application-level messages, exposing sensitive information like the user's IP address, which could be exploited to determine the user's location.

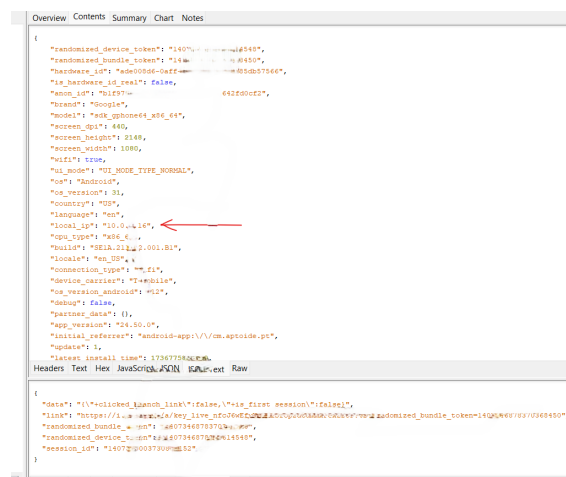


Figure 3.12: IP Address Exposure via Branch.io

Additionally, the application transmits location data (*latitude* and *longitude*):

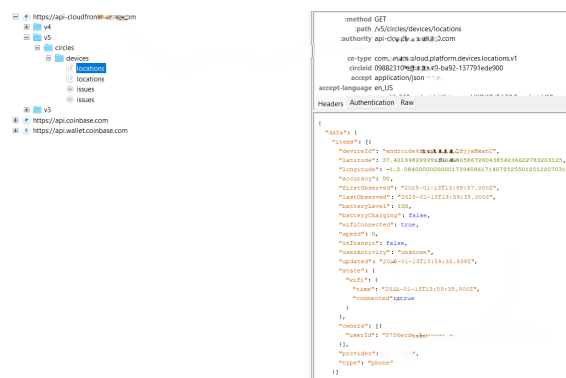


Figure 3.13: Unencrypted Location Data Transmission

The 'Active Bubble' feature, which allows users to create a bubble to keep others alerted while in a specific area, also transmits data without application-level encryption. After the activation of the service, the network traffic communicate the historical position clearly

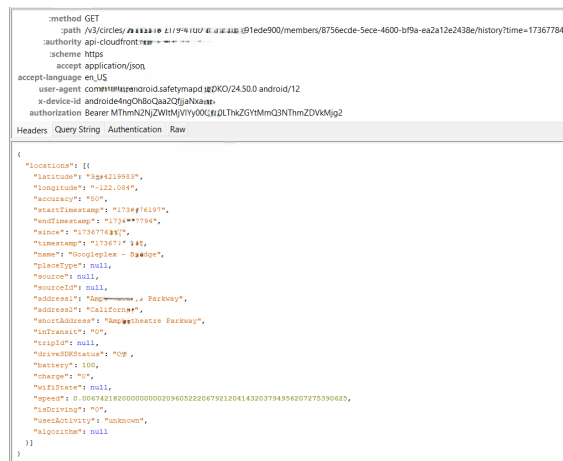


Figure 3.14: Historical Position Data Transmission

In this scenario an attacker can decrypt packets and read sensitive information in plaintext, such as geographic coordinates, physical addresses, and authorization tokens.

Risks

- Interception of sensitive data.
- Violation of user privacy.
- Impersonation through theft of authorization tokens.
- Tracking of user behavior.

3.2.2 Recommendations

- Implement application-level encryption for all sensitive data.
- Use short-lived authorization tokens and frequently refresh them.
- Avoid transmitting unnecessary sensitive data or obfuscate it before transmission.
- Conduct regular security audits to identify and address vulnerabilities.

Realistic Attack Scenarios

Description If an attacker convinces a user to install an unverified CA certificate, they can decrypt HTTPS traffic and access sensitive data.

Realistic Attack Scenarios

- **Enterprise Environments:** An attacker with access to the MDM system could distribute a malicious CA certificate to all managed devices.
- **Malicious Apps:** An app could require the installation of a CA certificate as part of its configuration process.
- **Public Wi-Fi Networks:** An attacker could convince a user to install a CA certificate to access a public Wi-Fi network.

3.2.3 Drozer Analysis

We used Drozer to analyze the application's exposed components and permissions. Key findings include:

```
dz> run app.package.info -a com.android.safetymapd
Attempting to run shell...
Package: com.android.safetymapd
Application Label: 
Process Name: com.android.safetymapd
Version: 24.37.0
Data Directory: /data/user/0/com.android.safetymapd
APK Path: /data/app/~ahZP23Uy8GG03Mbq~com.android.safetymapd-mNMD01Q/base.apk
UID: 10156
GID: [3003]
Shared Libraries: [/system/framework/org.apache.http.legacy.jar, /system_ext/framework/android.hudoc.sidecar.jar]
Shared User ID: null
Uses Permissions:
- android.permission.CAMERA
- android.permission.CHANGE_WIFI_STATE
- android.permission.NFC
- android.permission.CALL_PHONE
- android.permission.INTERNET
- android.permission.ACCESS_COARSE_LOCATION
```

Figure 3.15: Application Surface Analysis with Drozer

We extracted some basic application information:

```

d2> run app.package.attacksurface com.👤.android.safetymap
Attempting to run shell module
Attack Surface:
  1 activities exported
  4 broadcast receivers exported
  0 content providers exported
  1 services exported

```

Figure 3.16: Application Information Extracted with Drozer

Next, we analyzed the application's activities and their required permissions:

```
dz> run app.activity.info -a com.unity3d.android.safety
Attempting to run shell module
Package: com.unity3d.android.safety
com.unity3d.koko.root.RootActivity
Permission: null
```

Figure 3.17: Activity info

Database Security

We tested for potential SQL injection vulnerabilities, attempting both projection and selection attacks. No vulnerabilities were found, as shown below:

[illegible]

Figure 3.18: SQL Injection Test Results

Content Provider Analysis

Drozer attempted to access multiple content providers but received no response, indicating they are either inactive, require specific permissions, or are protected by security mechanisms. This is a **positive finding**, as it suggests the application employs robust security measures to protect sensitive data.

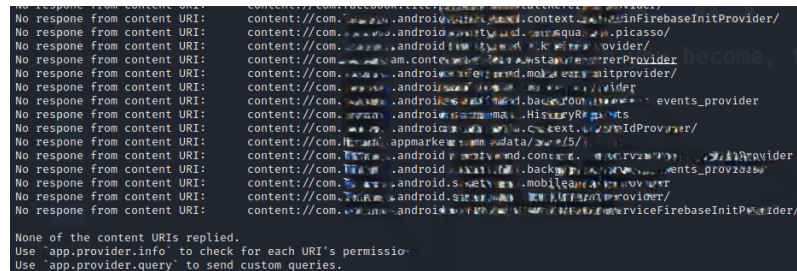


Figure 3.19: Content Provider Analysis

Exposed Services

Drozer identified one exported service in the application. This service is protected by the `android.permission.BIND_JOB_SERVICE` permission, which restricts its access to applications.

- `androidx.work.impl.background.systemjob.SystemJobService`

This protection mechanism reduces the risk of unauthorized access.

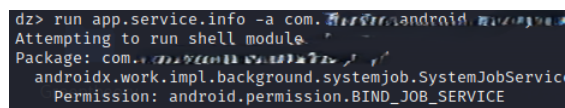


Figure 3.20: Exported Services Analysis

Final Considerations:

- **Drozer Analysis:** From the Drozer analysis, no critical vulnerabilities were identified. Activities, content providers, and databases are well-protected, and no SQL injection issues were found.
- **Charles Proxy Analysis:** The main vulnerability identified is the lack of application-level encryption, which leaves sensitive data susceptible to interception and manipulation, even with HTTPS and certificate pinning in place. Implementing end-to-end encryption at the application level would significantly enhance data security.