// Security Assessment

07.15.2025 - 07.17.2025

Android SDK
Infatica

HALBORN

Android SDK - Infatica

Prepared by: HALBORN

Last Updated 08/26/2025

Date of Engagement: July 15th, 2025 - July 17th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	2	1	0

TABLE OF CONTENTS

1. Introduction

`Infatica` engaged Halborn to conduct a code review assessment on their repository, beginning on `July 15th 2025` and ending on `July 17th 2025` . The security assessment was scoped to the private repository that `Infatica` shared with the Halborn team.

2. Assessment Summary

The `Halborn Team` was allocated three days for this engagement, during which one full-time security engineer — expert in blockchain and security — conducted a comprehensive audit of the in-scope components related to `Infatica` repository. This engineer possesses advanced skills in penetration testing, web application security assessments, and an in-depth understanding of multiple blockchain protocols.

The primary objectives of this audit were to:

- Verify that each component performs its intended functions accurately and reliably.
- Identify and assess potential security issues within the source code, particularly those that could impact the privacy and protection of PII (Personally Identifiable Information).

Additional objectives included evaluating adherence to industry best practices, ensuring robust security measures are in place, and identifying opportunities for further strengthening the security posture.

The source code review of `Infatica` repository revealed several weaknesses demanding attention.

In the course of a comprehensive source code review of the `Infatica` SDK, four distinct security shortcomings were identified that merit prompt attention. First, the SDK's Android test application was configured to permit cleartext traffic, thereby exposing user data and control communications to interception on untrusted networks. Second, numerous third-party libraries were found to be outdated or comprised known vulnerabilities, potentially undermining the overall security posture of any integrating application. Third, the release build configuration omitted code minification and obfuscation, leaving internal implementation details readily accessible to reverse engineering and intellectual property theft. Finally, the SDK's logging routines were observed to record network configuration information, such as DNS server addresses, which could inadvertently disclose network infrastructure data.

It is recommended that these areas be addressed holistically to elevate the security baseline of the `Infatica` SDK. Enforcement of encrypted communications, regular dependency management, activation of build-time obfuscation, and adoption of secure logging practices will collectively ensure that the SDK remains robust against emerging threats and integrations uphold the highest standards of confidentiality and integrity.

Finally, based on the review of `Infatica` SDK, it can be confirmed that the code did not collect, process, or store any end-user personal data (PII - *Personally Idenfiable Information*) beyond IP addresses and an anonymized UUID used solely for routing.

3. Test Approach And Methodology

`Halborn` combined manual and automated security testing to strike the right balance between speed, thoroughness, and precision within the scope of the penetration test. Manual techniques were employed to reveal nuanced logical, procedural, and implementation-level flaws, while automated tools broadened the assessment's reach—rapidly pinpointing common vulnerabilities across the entire solution.

Throughout the engagement, we progressed through, among the following —but not limited to— phases (when applied), leveraging both targeted tools and bespoke techniques:

- **Content & Functionality Mapping**
  Cataloging every feature and endpoint exposed by the application.
- **Technology-Stack & Public Code Review**
  Identifying known vulnerabilities in frameworks, libraries, and any publicly accessible source repositories.
- **Software Version Analysis**
  Detecting outdated or unpatched components.
- **Sensitive Data Exposure**
  Hunting for leaks of critical or private information.
- **Business-Logic Testing**
  Uncovering flaws in workflows, transaction flows, and access controls.
- **Access Control Assessment**
  Verifying correct enforcement of permissions and role-based restrictions.
- **Authentication & Authorization**
  Stress-testing login flows, token handling, and privilege escalation paths.
- **Input Validation & Handling**
  Ensuring all user inputs are properly sanitized and parsed.
- **Fuzzing & Parameter Injection**
  Applying randomized and structured payloads—SQL, JSON, HTML, command-line, directory path injections—to provoke unexpected behavior.
- **Brute-Force & Rate-Limiting Tests**
  Validating defenses against credential stuffing, account-lockout bypass, and throttling evasion.
- **Response Manipulation**
  Tampering with server replies to detect insecure assumptions or client-side vulnerabilities.
- **Deep Source-Code Review**
  Manually inspecting critical modules for hidden flaws and backdoors.

4. Caveats

5. Scope

The initial commit ID was **90abe642305d4e9d141c3eb4324ea9e41aa3858b**

- https://git.infatica.io/infatica_golang/android_sdk/-/tree/90abe642305d4e9d141c3eb4324ea9e41aa3858b

However, after two days of the engagement, the commit was updated to **89afb4413271db462a729201af63a0b2128fbdb0** because there were many issues during the compilation process.

- https://git.infatica.io/infatica_golang/android_sdk/-/tree/89afb4413271db462a729201af63a0b2128fbdb0

6. Constrains & Limitations

It was not possible to compile the provided source code due to persistent build errors encountered during the assessment. Although the `Infatica` team kindly supplied an updated version on the second day of our three-day review, the revised code likewise failed to compile. Consequently, Halborn was unable to execute or test a functional, compiled SDK during this engagement.

7. RISK METHODOLOGY

Halborn assesses the severity of findings using either the Common Vulnerability Scoring System (CVSS) framework or the Impact/Likelihood Risk scale, depending on the engagement. CVSS is an industry standard framework for communicating characteristics and severity of vulnerabilities in software. Details can be found in the CVSS Specification Document published by F.I.R.S.T.

Vulnerabilities or issues observed by Halborn scored on the Impact/Likelihood Risk scale are measured by the LIKELIHOOD of a security incident and the IMPACT should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

- **10** - CRITICAL
- **9 - 8** - HIGH
- **7 - 6** - MEDIUM
- **5 - 4** - LOW
- **3 - 1** - VERY LOW AND INFORMATIONAL

8. SCOPE

REPOSITORIES

(a) Repository: 90abe642305d4e9d141c3eb4324ea9e41aa3858b
(b) Assessed Commit ID: 90abe64

(a) Repository: b0abd19070c5fe366b0c90406b77581c2def8e5c
(b) Assessed Commit ID: b0abd19

(a) Repository: 89afb4413271db462a729201af63a0b2128fbdb0
(b) Assessed Commit ID: 89afb44

REMEDIATION COMMIT ID:

- https://git.infatica.io/infatica_golang/android_sdk/-/tree/b3a27148d91ee16105c42363853791d1ec66d102

Out-of-Scope: New features/implementations after the remediation commit IDs.

9. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	1	0

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CLEARTEXT TRAFFIC ALLOWED	MEDIUM	RISK ACCEPTED - 08/11/2025
RELEASE BUILD WITHOUT CODE OBFUSCATION	MEDIUM	SOLVED - 08/11/2025
SENSITIVE INFORMATION DISCLOSURE VIA LOGGING	LOW	SOLVED - 08/11/2025

10. FINDINGS & TECH DETAILS

10.1 CLEARTEXT TRAFFIC ALLOWED

// MEDIUM

Description

The Android test application explicitly allowed unencrypted HTTP traffic. In the file `android_sdk/service-test/app/src/main/AndroidManifest.xml`, the application manifest included:

```
<application...
    android:usesCleartextTraffic="true">
    …
</application>
```

Because `usesCleartextTraffic` was set to `true`, the SDK permitted all cleartext (HTTP) connections at runtime, bypassing Android's default network security protections.

Impact

Allowing cleartext traffic exposed end-users to man-in-the-middle (MITM) attacks. An attacker on the same network (e.g., a public Wi-Fi hotspot) could intercept or modify HTTP requests and responses between the SDK and its back-end servers. This could lead to leakage of sensitive data (such as session tokens or proxy metadata), alteration of proxy instructions, or injection of malicious payloads into the SDK's traffic.

Proof of Concept

- File name: `android_sdk/service-test/app/src/main/AndroidManifest.xml`
- Line number: `6`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="<http://schemas.android.com/apk/res/android>"
    xmlns:tools="<http://schemas.android.com/tools>">

    <application
        android:usesCleartextTraffic="true"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.InfaticaAgentServiceTest"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:theme="@style/Theme.InfaticaAgentServiceTest">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Service.kt | build.gradle M | AndroidManifest.xml

service-test > app > src > main > AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:usesCleartextTraffic="true"
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.InfaticaAgentServiceTest"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:theme="@style/Theme.InfaticaAgentServiceTest">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Score

CVSS:3.1/AV:A/AC:L/PR:N/UI:R/S:U/C:H/I:N/A:N (5.7)

Recommendation

- **Enforce HTTPS**: Remove `android:usesCleartextTraffic="true"` or set it to `false` in all manifest files for production builds.
- **Network Security Config**: Create a Network Security Configuration XML that only permits cleartext for explicitly whitelisted domains (if strictly necessary for legacy endpoints), and reference it in the manifest via `android:networkSecurityConfig`.
- **CI Validation**: Add a build-time check to fail CI if any manifest in the SDK (or test apps) allows cleartext traffic.
- **Review Legacy Dependencies**: Ensure all third-party libraries and endpoints used by the SDK support HTTPS exclusively.

Remediation Comment

`RISK ACCEPTED` : The `Infatica team` accepted the risk of this issue.

10.2 RELEASE BUILD WITHOUT CODE OBFUSCATION

// MEDIUM

Description

During the source code review of the private Infatica SDK repository, it was discovered that the Android test application's Gradle release configuration had code minification and obfuscation disabled. In `android_sdk/service-test-java/app/build.gradle.kts`, the `release` build type was defined as:

```
buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(getDefaultProguardFile("proguard-android-optimize.txt"), "proguard-rules.pro")
        }
    }
```

Because `minifyEnabled` was set to `false`, neither *R8* nor *ProGuard* processed or obfuscated the compiled bytecode, leaving class names, method signatures, and internal logic in plaintext form.

Impact

As a result of missing code shrinking and obfuscation, an attacker who obtained the released APK could easily reverse-engineer the SDK. Readable class and method names would reveal internal implementation details—such as proxy handshake logic, revenue-tracking algorithms, or any embedded credentials/patterns—enabling intellectual property theft, unauthorized modification, or targeted attacks against the SDK's business logic.

Proof of Concept

- File name: `android_sdk/service-test-java/app/build.gradle.kts`
- Line number: `25`

```
buildTypes {
        release {
            isMinifyEnabled = false
            proguardFiles(getDefaultProguardFile("proguard-android-optimize.txt"), "proguard-rules.pro")
        }
    }
```

- File name: `android_sdk/service-test/app/build.gradle`
- Line number: `25`

```
buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
```

- File name: `android_sdk/service/service/build.gradle`
- Line number: `25 and 31`

```
buildTypes {
        debug {
            debuggable true
            minifyEnabled false
        }
        release {
            debuggable false
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
```

Example screenshot of source code:

build.gradle M | build.gradle.kts

service-test-java > app > build.gradle.kts

```
1   plugins {
2       alias(libs.plugins.android.application)
3       alias(libs.plugins.jetbrains.kotlin.android)
4   }
5
6   android {
7       namespace = "com.infatica.agent.service.test.java"
8       compileSdk = 34
9
10      defaultConfig {
11          applicationId = "com.infatica.agent.service.test.java"
12          minSdk = 21
13          targetSdk = 34
14          versionCode = 1
15          versionName = "1.0"
16
17          testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
18          vectorDrawables {
19              useSupportLibrary = true
20          }
21      }
22
23      buildTypes {
24          release {
25              isMinifyEnabled = false
26              proguardFiles(getDefaultProguardFile("proguard-android-optimize.txt"), "proguard-rules.pro")
27          }
28      }
29      compileOptions {
30          sourceCompatibility = JavaVersion.VERSION_1_8
31          targetCompatibility = JavaVersion.VERSION_1_8
32      }
```

Score

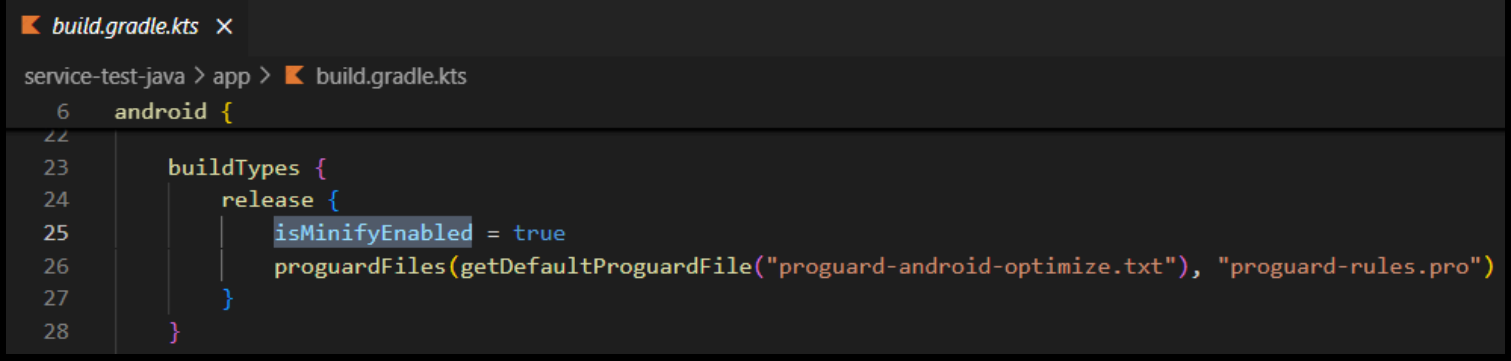
CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:N (4.7)

Recommendation

- **Enable Minification**: Set `isMinifyEnabled = true` for the `release` build type to invoke R8/ProGuard.
- **Customize Rules**: Review and harden `proguard-rules.pro` to preserve only required SDK entry points (e.g., public API) and obfuscate internal classes.
- **Automate Checks**: Incorporate a CI gate that fails if `minifyEnabled` is `false` on production builds.
- **Separate Test vs. Production**: Ensure test applications use relaxed settings, but enforce strict obfuscation and shrinking on all production SDK releases.

Remediation Comment

`SOLVED` : The `Infatica team` addressed this issue.



Remediation Hash

https://git.infatica.io/infatica_golang/android_sdk/-/tree/b3a27148d91ee16105c42363853791d1ec66d102

10.3 SENSITIVE INFORMATION DISCLOSURE VIA LOGGING

// LOW

Description

During a source code review of the private Infatica SDK repository, it was discovered that the Android implementation was logging sensitive network information. Specifically, the method `networks()` collected the end-user's DNS server IP addresses and immediately emitted them to the debug log via:

```
Log.d(tag, "networks(): $json")
```

This JSON payload could reveal the user's network configuration or ISP details—information that was considered personal data and should not have been exposed in application logs.

Impact

Because Android debug logs (logcat) were accessible to any application holding the `READ_LOGS` permission or via USB debugging (ADB), an attacker or malicious app could retrieve the DNS IPs of end users. This unintended disclosure of network configuration could facilitate targeted network attacks, fingerprinting of users' ISPs, or correlation of user activity across different networks.

Proof of Concept

- File name: `android_sdk/service/service/src/main/java/com/infatica/agent/service/Service.kt`
- Line number: `255`

```
Log.d(tag, "networks(): $json")
```

build.gradle M | Service.kt

service > service > src > main > java > com > infatica > agent > service > Service.kt

```
57   class Service : android.app.Service() {
198      private fun networks(): String {
217          cm?.allNetworks?.forEach { network ->
218              val caps = cm.getNetworkCapabilities(network)
219              val props = cm.getLinkProperties(network)
220
221              val hasInternet = caps?.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET) == true
222              if (!hasInternet) {
223                  return@forEach
224              }
225
226              var type = 0
227              if (caps?.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR) == true) {
228                  type = 1
229              }
230              if (caps?.hasTransport(NetworkCapabilities.TRANSPORT_WIFI) == true) {
231                  type = type.or(2)
232              }
233              if (caps?.hasTransport(NetworkCapabilities.TRANSPORT_VPN) == true) {
234                  type = type.or(4)
235              }
236
237              val dns = mutableListOf<String>()
238              props?.dnsServers?.forEach {inetAddress ->
239                  inetAddress.hostAddress?.let { dns.add(it) }
240              }
241
242              val active = if (VERSION.SDK_INT >= VERSION_CODES.M) {
243                  network.equals(cm.activeNetwork)
244              } else {
245                  false
246              }
247
248              networks.add(Network(type, dns, active))
249          }
250
251          val json = "[${networks.joinToString(separator = ",") { network ->
252              network.toJsonString()
253          }}]"
254
255          Log.d(tag, "networks(): $json")
256
257          return json
258      }
```

Score

CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:L/I:N/A:N (3.8)

Recommendation

- **Remove or sanitize**: The call to `Log.d(tag, "networks(): $json")` should have been removed from production builds.
- **Adjust build configurations**: Ensure that verbose or debug logging of any network-related data is disabled in release APKs (e.g., by guarding with `BuildConfig.DEBUG`).
- **Limit data exposure**: If logging of network diagnostics is required for troubleshooting, log only non-identifying metrics (e.g., the count of interfaces) and avoid actual IP values.
- **Adopt secure logging practices**: Use a configurable logging framework that can filter or redact sensitive fields based on environment (development vs. production).

Remediation Comment

SOLVED : The Infatica team addressed the issue by removing the log information.

Remediation Hash

https://git.infatica.io/infatica_golang/android_sdk/-/tree/b3a27148d91ee16105c42363853791d1ec66d102

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.