CORELLIUM®

MATRIX Report

Test ID	1899e005-2413-4b38-ad08-3474a481d5d7
Test Duration	10m 31s
Test Date	August 6th, 2024
Created By	Corellium
App Name	Cafe
App ID	com.corellium.cafe
App Version	1.0
Device Model	Generic Android
Device OS	14.0.0
Device Rooted?	Yes
Corellium Environment	enterprise.corellium.com
Corellium Version	6.4.1-22730

Results

62	18	0	0
Passed	Failed	Artifact	Error

Checks

Name	Category	Status
Application Contains Hardcoded URLs	AUTH-2	Failed
Insecure Biometric Implementation	AUTH-2	Passed
MinSDK Targets Vulnerable Android Versions	CODE-1	Failed
Insecure libpng Library Version	CODE-2	Passed
Insecure libjpeg-turbo Library Version	CODE-2	Passed
Insecure OkHTTP Library Version	CODE-2	Passed
Insecure Joda Library Identified	CODE-2	Passed
Application Utilizes Stack Canaries	CODE-2	Passed
Safe Browsing Not Enabled in WebViews	CODE-2	Passed
Application Utilizes Position Independent Code	CODE-2	Passed
Application Utilizing Insecure Cryptography	CRYPTO-1	Failed
Application Utilizing Insecure Symmetric Encryption Modes	CRYPTO-1	Failed
Insecure Random Number Generation	CRYPTO-1	Failed
Application Contains Hardcoded SQLCipher Key	CRYPTO-2	Failed
Application Contains Hardcoded API Keys	CRYPTO-2	Passed
Application Allows Insecure Network Connections	NETWORK-1	Failed
HTTP Cleartext Transmission of Device IMEI	NETWORK-1	Passed
HTTP Cleartext Transmission of GPS Longitude Coordinates	NETWORK-1	Passed
HTTP Cleartext Transmission of GPS Latitude Coordinates	NETWORK-1	Passed
HTTP Cleartext Transmission of Sensitive Data	NETWORK-1	Passed
HTTP Cleartext Transmission of DNS Address	NETWORK-1	Passed
HTTP Cleartext Transmission of Build Fingerprint	NETWORK-1	Passed
HTTP Cleartext Transmission of Bluetooth MAC Address	NETWORK-1	Passed
HTTP Cleartext Transmission of Device Android ID	NETWORK-1	Passed
HTTPS Traffic URL Contains WiFi MAC Address	NETWORK-1	Passed
HTTPS Traffic URL Contains Device IMEI	NETWORK-1	Passed
Application VPN Opts out of Always-On Feature	NETWORK-1	Passed
HTTPS Traffic URL Contains Device's GPS Longitude	NETWORK-1	Passed
HTTPS Traffic URL Contains Device's GPS Latitude	NETWORK-1	Passed
HTTPS Traffic URL Contains Sensitive Data	NETWORK-1	Passed
HTTPS Traffic URL Contains DNS Address	NETWORK-1	Passed

HTTPS Traffic URL Contains Build Fingerprint	NETWORK-1	Passed
HTTPS Traffic URL Contains Bluetooth MAC Address	NETWORK-1	Passed
HTTPS Traffic URL Contains Android ID	NETWORK-1	Passed
Insecure Security Provider	NETWORK-1	Passed
Insecure TLS configuration	NETWORK-1	Passed
Cookie missing 'Secure' flag	NETWORK-1	Passed
Application Target SDK Allows for Insecure Network Configurati...	NETWORK-1	Passed
Cookie missing 'httpOnly' flag	NETWORK-1	Passed
Application Contains Heartbleed Vulnerable OpenSSL Version	NETWORK-1	Passed
Application Contains Change Cipher Spec Injection Vulnerable ...	NETWORK-1	Passed
Application Contains Insecure HTTP Traffic	NETWORK-1	Passed
HTTP Cleartext Transmission of WiFi MAC Address	NETWORK-1	Passed
Application Manifest Contains Unverified Deep Links	PLATFORM-1	Passed
Application Utilizes Intents Vulnerable to Redirection	PLATFORM-1	Failed
Application Utilizes Broadcast Receivers Without Permissions	PLATFORM-1	Passed
Application Utilizes Potentially Dangerous Permissions	PLATFORM-1	Failed
Application Contains Debuggable WebViews	PLATFORM-2	Failed
Application Utilizes Insecure JavaScript Interface	PLATFORM-2	Failed
Application Contains WebViews with Javascript Enabled	PLATFORM-2	Failed
Application is Vulnerable to Task Hijacking	PLATFORM-3	Passed
Janus Exploitation Possible Due to Current Minimum SDK	RESILIENCE-2	Failed
Application Utilizes a Weak Signing Key	RESILIENCE-2	Passed
Application Utilizes a Weak Signing Scheme	RESILIENCE-2	Passed
Application Contains Debug Library	RESILIENCE-3	Failed
Application Enables Debugging within the Android Manifest	RESILIENCE-4	Failed
Local Data Exposure: DNS Address Stored Insecurely	STORAGE-1	Passed
Local Data Exposure: GPS Longitude Stored Insecurely	STORAGE-1	Passed
Local Data Exposure: GPS Latitude Stored Insecurely	STORAGE-1	Passed
Local Data Exposure: Sensitive Values Stored Insecurely on De...	STORAGE-1	Failed
Local Data Exposure: Global Write Permissions	STORAGE-1	Passed
Local Data Exposure: Global Read Permissions	STORAGE-1	Passed
Local Data Exposure: Bluetooth MAC Address Stored Insecurely	STORAGE-1	Passed

Local Data Exposure: Insecure Hardcoded API Keys	STORAGE-1	Passed
Local Data Exposure: Insecure Hardcoded Passwords	STORAGE-1	Passed
Local Data Exposure: WiFi MAC Address Stored Insecurely	STORAGE-1	Passed
Local Data Exposure: WiFi IP Address Stored Insecurely	STORAGE-1	Passed
Local Data Exposure: Device IMEI Stored Insecurely	STORAGE-1	Passed
Local Data Exposure: Android ID Stored Insecurely	STORAGE-1	Passed
Local Data Exposure: WiFi IP Address Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: Bluetooth MAC Address Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: Sensitive Values Stored in Memory	STORAGE-2	Failed
Local Data Exposure: Device IMEI Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: Application Backups Enabled	STORAGE-2	Failed
Local Data Exposure: Device Fingerprint Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: WiFi MAC Address Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: GPS Longitude Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: GPS Latitude Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: Sensitive Data Logged Insecurely	STORAGE-2	Passed
Local Data Exposure: DNS Address Logged Insecurely	STORAGE-2	Passed

Details

AUTH

Authentication and authorization are essential components of most mobile apps, especially those that connect to a remote service. These mechanisms provide an added layer of security and help prevent unauthorized access to sensitive user data. Although the enforcement of these mechanisms must be on the remote endpoint, it is equally important for the app to follow relevant best practices to ensure the secure use of the involved protocols. Mobile apps often use different forms of authentication, such as biometrics, PIN, or multi-factor authentication code generators, to validate user identity. These mechanisms must be implemented correctly to ensure their effectiveness in preventing unauthorized access. Additionally, some apps may rely solely on local app authentication and may not have a remote endpoint. In such cases, it is critical to ensure

that local authentication mechanisms are secure and implemented following industry best practices. The controls in this category aim to ensure that the app implements authentication and authorization mechanisms securely, protecting sensitive user information and preventing unauthorized access. It is important to note that the security of the remote endpoint should also be validated using industry standards such as the OWASP Application Security Verification Standard (ASVS).

A2

AUTH-2

The app performs local authentication securely according to the platform best practices.

Many apps allow users to authenticate via biometrics or a local PIN code. These authentication mechanisms need to be correctly implemented. Additionally, some apps might not have a remote endpoint, and rely fully on local app authentication.

Application Contains Hardcoded URLs

This test evaluates how the application manages hardcoded URLs within its codebase and application resources. Hardcoded URLs can contain sensitive information regarding backend infrastructure, sensitive information, access tokens or provide additional insight for attackers to conduct targeted attacks.

Status	Severity
Failed	Info

Impact

Hardcoded URLs can provide attackers with access to sensitive information including detailed information regarding the application backend infrastructure which can be leveraged to conduct a more in-depth targeted attack.

Remediation

An application codebase and required resources should not contain hardcoded URLs. If your application requires a URL, it should be retrieved

from the application backend.

Evidence

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/ui/activities/WebViewActivity.java

```
https://www.corellium.com/blog
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
https://www.example.com
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/functions/FunctionsKt$sendPost$1.java

```
https://www.corellium.com/
```

Insecure Biometric Implementation

The application uses biometric authentication, but the implementation does not fully adhere to secure coding best practices. This could result in potential security vulnerability that can be exploited by malicious actors to gain unauthorized access to your application.

Status	Severity
Passed	**Medium**

Impact

If the biometric implementation for your application is insecure, it could lead to a wide range of security concerns. These may include unauthorized access to the application, exposure of sensitive user data, and potential misuse of biometric information.

Remediation

Review the implementation of biometric authentication to ensure it follows best security practices. Ensure that secure cryptographic methods are employed when handling biometric data. It is recommended to dynamically test the application to confirm the security of the biometric implementation. Refer to the official Android security guidelines for biometric authentication to implement necessary safeguards.

CODE

Mobile apps have many data entry points, including the UI, IPC, network, and file system, which might receive data that has been inadvertently modified by untrusted actors. By treating this data as untrusted input and properly verifying and sanitizing it before use, developers can prevent classical injection attacks, such as SQL injection, XSS, or insecure deserialization. However, other common coding vulnerabilities, such as memory corruption flaws, are hard to detect in penetration testing but easy to prevent with secure architecture and coding practices. Developers should follow best practices such as the OWASP Software Assurance Maturity Model (SAMM) and NIST.SP.800-218 Secure Software Development Framework (SSDF) to avoid introducing these flaws in the first place. This category covers coding vulnerabilities that arise from external sources such as app data entry points, the OS, and third-party software components. Developers should verify and sanitize all incoming data to prevent injection attacks and bypass of security checks. They should also enforce app updates and ensure that the app runs up-to-date platforms to protect users from known vulnerabilities.

C1

CODE-1

The app requires an up-to-date platform version.

Every release of the mobile OS includes security patches and new security features. By supporting older versions, apps stay vulnerable to well-known threats. This control ensures that the app is running on an up-to-date platform version so that users have the latest security protections.

MinSDK Targets Vulnerable Android Versions

This application is able to be installed on devices running Android 7.1 (API Level 25) or earlier. Applications with a 'minSdkVersion' lower than 26 will not have crucial security and privacy features applied to improve the overall security of your application.

Status	Severity
Failed	Low

Impact

Running your application on outdated versions of Android can expose users to a wide variety of vulnerabilities including outdated versions of TLS.

Remediation

Ensure the application's minimum SDK (minSdkVersion) is set to a value of at least 26.

Evidence

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
android:minSdkVersion=23
```

C2

CODE-2

The app has a mechanism for enforcing app updates.

Sometimes critical vulnerabilities are discovered in the app when it is already in production. This control ensures that there is a mechanism to force the users to update the app before they can continue using it.

Insecure libpng Library Version

The current version of libpng used by your application is vulnerable to CVE-2017-12652 which affects versions before 1.6.32. This vulnerability pertains to an issue in the 'png_set_text_2' function in 'pngset.c'. A malicious attacker can use this vulnerability to cause a denial-of-service attack due to an out-of-bounds read.

Status	Severity
Passed	Low

Impact

When exploited an attacker could crash your application through a denial-of-service attack or, in some cases, execute arbitrary code. This is concerning for applications that process PNG image files.

Remediation

It is recommended to upgrade the libpng library to version 1.6.32 or later

Insecure libjpeg-turbo Library Version

Versions of the libjpeg-turbo library prior to 2.0.2 contain vulnerabilities related to the processing of JPEG image files. These issues arise from the way the vulnerable library handles specific encoding scenarios and processes color quantization tables. These vulnerabilities can be exploited using specially crafted JPEG files.

Status	Severity
Passed	Low

Impact

Due to multiple vulnerabilities within the libjpeg-turbo library the application is potentially vulnerable to denial-of-service conditions along with the potential of sensitive information disclosure.

Remediation

It is recommended to upgrade the libjpeg-turbo library to version 2.0.2 or later

Insecure OkHTTP Library Version

Versions of the OkHTTP library prior to 4.0 may contain vulnerabilities related to the processing and handling of network communications. This includes missing security enhancements and the use of deprecated network protocols.

Status	Severity
Passed	Low

Impact

Utilizing outdated versions of the OkHTTP library can lead to interception of data, various Man-in-the-Middle attacks along with an overall degraded application security posture.

Remediation

It is recommended to upgrade the OkHTTP library to version 4 or later

Insecure Joda Library Identified

Joda time library which historically has been popular for handling data and time operations has been deprecated in favor of 'java.time'. Continued use of deprecated libraries can expose your application to a variety of security risks.

Status	Severity
Passed	Low

Impact

Deprecated libraries do not receive regular updates, patches or bug fixes which can lead to security and functionality issues. Additionally, the use of deprecated libraries can cause incompatibility issues for your application.

Remediation

Migrate from the Joda time library to 'java.time'

Application Utilizes Stack Canaries

This finding pertains to the absence of stack canaries within native application libraries. Stack canaries are a crucial stack smashing protection measure designed to detect and prevent buffer overflow attacks by placing a random value before the return address on the stack.

Status	Severity
Passed	Low

Impact

The absence of stack canaries within native libraries renders the application vulnerable to method buffer overflow attacks and other memory corruption-related exploits. Exploiting these vulnerabilities could lead to compromised user data, financial loss, and reputational damage to the organization.

Remediation

To address this vulnerability, ensure that all native libraries utilized by the Android application are compiled with stack canaries enabled.

Safe Browsing Not Enabled in WebViews

This vulnerability involves checking if Safe Browsing is enabled within the applications AndroidManifest file. Safe Browsing is a feature provided by Google Play Protect that warns users when they attempt to navigate to potentially dangerous websites.

Status	Severity
Passed	Medium

Impact

Failure to enable Safe Browsing for an Android application increases the risk for users. Without this protection, users are vulnerable to accessing potentially malicious websites without warning. This can lead to unauthorized data access, phishing attempts and potential malware infections.

Remediation

To remediate this vulnerability, developers should ensure that Safe Browsing is enabled within the applications AndroidManifest.

Application Utilizes Position Independent Code

This finding pertains to the absence of Position Independent Code (PIC) within native libraries. PIC is a crucial security measure that allows the code

to execute regardless of its absolute address, enhancing the application's resilience against memory corruption exploits.

Status	Severity
Passed	Low

Impact

The absence of Position Independent Code (PIC) within native libraries exposes the application to potential memory corruption vulnerabilities. Exploiting these vulnerabilities could result in compromised user data, financial loss, and reputational harm to the organization.

Remediation

To mitigate this vulnerability, ensure that all native libraries utilized by the Android application are compiled with Position Independent Code (PIC) enabled.

CRYPTO

Cryptography is essential for mobile apps because mobile devices are highly portable and can be easily lost or stolen. This means that an attacker who gains physical access to a device can potentially access all the sensitive data stored on it, including passwords, financial information, and personally identifiable information. Cryptography provides a means of protecting this sensitive data by encrypting it so that it cannot be easily read or accessed by an unauthorized user. The purpose of the controls in this category is to ensure that the verified app uses cryptography according to industry best practices, which are typically defined in external standards such as NIST.SP.800-175B and NIST.SP.800-57. This category also focuses on the management of cryptographic keys throughout their lifecycle, including key generation, storage, and protection. Poor key management can compromise even the strongest cryptography, so it is crucial for developers to follow the recommended best practices to ensure the security of their users' sensitive data.

C1

CRYPTO-1

The app employs current strong cryptography and uses it according to industry best practices.

Cryptography plays an especially important role in securing the user's data - even more so in a mobile environment, where attackers having physical access to the user's device is a likely scenario. This control covers general cryptography best practices, which are typically defined in external standards.

Application Utilizing Insecure Cryptography

The application contains references within the code to insecure cryptography implementations. These outdated cryptographic algorithms do not need industry compliance standards and can be vulnerable to a variety of attacks resulting in a breach of application confidentiality and integrity.

Status

Severity

Failed Low

Impact

The use of insecure cryptographic algorithms can lead to unauthorized decryption of data and overall a lack of data integrity. Additionally, such practices risk non-compliance with various industry standards.

Remediation

Replace insecure cryptographic implementations with the secure industry accepted alternatives. For additional guidance please refer to following android developer guide - https://developer.android.com/privacy-and-security/cryptography

Evidence

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
Cipher.getInstance('DES');
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
Cipher.getInstance('RC4');
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
MessageDigest.getInstance('MD5');
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
KeyGenerator.getInstance('DES');
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
SecretKeyFactory.getInstance('PBKDF2WithHmacSHA1');
```

Application Utilizing Insecure Symmetric Encryption Modes

The application employs insecure modes of symmetric encryption. These modes do not provide strong data confidentiality and may be vulnerable to a variety of cryptographic attacks such as pattern analysis and padding oracle attacks, compromising the confidentiality and integrity of the application's data.

Status	Severity
Failed	Low

Impact

Using ECB mode (Electronic Codebook) or 'NoPadding' with symmetric encryption exposes sensitive data to possible decryption by unauthorized parties. The utilization of these insecure modes will not meet industry compliance standards.

Remediation

It is recommended to switch to a more secure mode of operation for symmetric encryption, such as CBC (Cipher Block Chaining) and ensure the use of a strong, cryptographically secure pseudo-random initialization vector (IV). All cryptographic practices should be aligned with current industry standards and best practices.

Evidence

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
Cipher.getInstance('AES/ECB/NoPadding');
```

Insecure Random Number Generation

The application has been found to utilize the 'java.util.Random' class for generating random values. This class is not suitable for security-related purposes as it produces predictable random values, which can compromise the overall security of the application.

Status	Severity
Failed	Low

Impact

Using the 'java.util.Random' class for random number generation can lead to predictable values that may be exploited by attackers. This could result in potential security vulnerabilities, including unauthorized access and data breaches.

Remediation

Review and update your application code to replace instances of the 'java.util.Random' class with the 'java.security.SecureRandom' class for generating random values. Ensure that all random number generation adheres to best practices for security. Refer to the official Android security guidelines for implementing secure random number generation.

Evidence

File: /data/local/tmp/artifacts/output/sources/okhttp3/internal/ws/RealWebSocket.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/okhttp3/internal/ws/WebSocketWriter.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/okhttp3/OkHttpClient.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/random/FallbackThreadLocalRandom$implStorage$1.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/random/FallbackThreadLocalRandom.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/random/KotlinRandom.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/random/PlatformRandom.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/random/AbstractPlatformRandom.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/random/PlatformRandomKt.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/random/jdk8/PlatformThreadLocalRandom.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/collections/CollectionsKt__MutableCollectionsJVMKt.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/kotlin/collections/CollectionsKt__CollectionsJVMKt.java

```
import java.util.Random;
```

File: /data/local/tmp/artifacts/output/sources/androidx/activity/result/ActivityResultRegistry.java

```
import java.util.Random;
```

C2

CRYPTO-2

The app performs key management according to industry best practices.

Even the strongest cryptography would be compromised by poor key management. This control covers the management of cryptographic keys throughout their lifecycle, including key generation, storage and protection.

Application Contains Hardcoded SQLCipher Key

The application's codebase contains hardcoded instances of a 'PRAGMA Key', indicating the encryption key for a database. Hardcoded keys can be easily extracted from the application bundle, making the database encryption ineffective.

Status	Severity
Failed	Medium

Impact

Hardcoded encryption keys pose a severe risk to data confidentiality and integrity. If an attacker has access to the key, they can decrypt and retrieve any data stored within the database.

Remediation

Remove all hardcoded encryption keys from the source code and instead utilize a secure method of key management, such as leveraging platform-specific key stores or using a secure server for key distribution.

Evidence

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
Log.i('InsecureCrypto', 'moreInsecurities: key PRAGMA Key');
```

Application Contains Hardcoded API Keys

The application's code contains hardcoded instances of an API key. Hardcoded keys are susceptible to extraction, posing a security risk as they can be utilized maliciously to access sensitive data or perform unauthorized actions.

Status	Severity
Passed	Low

Impact

Hardcoded API keys stored insecurely can pose a significant security risk. If an API key gets extracted, it can be used to gain unauthorized access to sensitive resources or services associated.

Remediation

Remove all sensitive hardcoded API keys from the application source code. Implement a secure method for API key management based on industry best practices.

NETWORK

Secure networking is a critical aspect of mobile app security, particularly for apps that communicate over the network. In order to ensure the confidentiality and integrity of data in transit, developers typically rely on encryption and authentication of the remote endpoint, such as through the use of TLS. However, there are numerous ways in which a developer may accidentally disable the platform secure defaults or bypass them entirely by utilizing low-level APIs or third-party libraries. This category is designed to ensure that the mobile app sets up secure connections under any circumstances. Specifically, it focuses on verifying that the app establishes a secure, encrypted channel for network communication. Additionally, this category covers situations where a developer may choose to trust only specific Certificate Authorities (CAs), which is commonly referred to as certificate pinning or public key pinning.

N1

NETWORK-1

The app secures all network traffic according to the current best practices.

Ensuring data privacy and integrity of any data in transit is critical for any app that communicates over the network. This is typically done by encrypting data and authenticating the remote endpoint, as TLS does. However, there are many ways for a developer to disable the platform secure defaults, or bypass them completely by using low-level APIs or third-party libraries. This control ensures that the app is in fact setting up secure connections in any situation.

Application Allows Insecure Network Connections

The application is susceptible to insecure network connections due to misconfigurations within the network settings. These misconfigurations can lead to the absence of secure network protocols and weak security configurations.

Status	Severity

Failed	Medium

Impact

Insecure network connections can pose a risk to the confidentiality and integrity of data transmitted over the network. Attackers can potentially intercept and manipulate sensitive information being exchanged between the application and the remote servers.

Remediation

Implement secure network protocols, enforce the use of HTTPS for communication with remote servers, and adopt industry best practices for network security configurations.

Evidence

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
android:usesCleartextTraffic=true
```

File: /data/local/tmp/artifacts/output/resources/res/xml/network_security_config.xml

```
cleartextTrafficPermitted=true
```

HTTP Cleartext Transmission of Device IMEI

The application transmits the device's IMEI over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	Medium

Impact

Transmitting the device's IMEI over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information,

compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's IMEI, is transmitted over HTTPS.

HTTP Cleartext Transmission of GPS Longitude Coordinates

The application transmits the device's GPS Longitude Coordinates over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	Medium

Impact

Transmitting the device's GPS Longitude Coordinates over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's GPS Longitude Coordinates, is transmitted over HTTPS.

HTTP Cleartext Transmission of GPS Latitude Coordinates

The application transmits the device's GPS Latitude Coordinates over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	Medium

Impact

Transmitting the device's GPS Latitude Coordinates over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's GPS Latitude Coordinates, is transmitted over HTTPS.

HTTP Cleartext Transmission of Sensitive Data

The application transmits sensitive data over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	High

Impact

Transmitting sensitive data over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as sensitive data, is transmitted over HTTPS.

HTTP Cleartext Transmission of DNS Address

The application transmits the device's DNS Address over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity

Passed	Low

Impact

Transmitting the device's DNS Address over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's DNS Address, is transmitted over HTTPS.

HTTP Cleartext Transmission of Build Fingerprint

The application transmits the device's Build Fingerprint over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	Medium

Impact

Transmitting the device's Build Fingerprint over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's Build Fingerprint, is transmitted over HTTPS.

HTTP Cleartext Transmission of Bluetooth MAC Address

The application transmits the device's Bluetooth MAC Address over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	Medium

Impact

Transmitting the device's Bluetooth MAC Address over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's Bluetooth MAC Address, is transmitted over HTTPS.

HTTP Cleartext Transmission of Device Android ID

The application transmits the device's Android ID over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	Medium

Impact

Transmitting the device's Android ID over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's Android ID, is transmitted over HTTPS.

HTTPS Traffic URL Contains WiFi MAC Address

The application transmits the device's WiFi MAC address in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Low

Impact

Transmitting the device's WiFi MAC Address in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including sensitive information, such as the device's WiFi MAC Address, in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

HTTPS Traffic URL Contains Device IMEI

The application transmits the device IMEI in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Medium

Impact

Transmitting the device IMEI in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including sensitive information, such as the device IMEI, in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

Application VPN Opts out of Always-On Feature

The application, which utilizes a VPN, has opted out of the Always-On feature. This configuration may expose the application's network traffic to potential interception and compromise the effectiveness of the VPN.

Status	Severity
Passed	Medium

Impact

Opting out of the Always-On feature in a VPN configuration poses a low risk. It may result in intermittent lapses of secure VPN connections, potentially exposing sensitive data to unauthorized parties.

Remediation

Reconfigure the application's VPN settings to enable the Always-On feature, ensuring a persistent secure connection and reducing the likelihood of unintended data exposure.

HTTPS Traffic URL Contains Device's GPS Longitude

The application transmits the device's GPS Longitude in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Low

Impact

Transmitting the device's GPS Longitude in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including sensitive information, such as the device's GPS Longitude, in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

HTTPS Traffic URL Contains Device's GPS Latitude

The application transmits the device's GPS Latitude in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Low

Impact

Transmitting the device's GPS Latitude in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including sensitive information, such as the device's GPS Latitude, in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

HTTPS Traffic URL Contains Sensitive Data

The application transmits sensitive data in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Medium

Impact

Transmitting sensitive data in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including sensitive data in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

HTTPS Traffic URL Contains DNS Address

The application transmits the device's DNS Address in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Low

Impact

Transmitting the device's DNS Address in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including the device's DNS Address in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

HTTPS Traffic URL Contains Build Fingerprint

The application transmits the device's Build Fingerprint in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Low

Impact

Transmitting the device's Build Fingerprint in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including the device's Build Fingerprint in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

HTTPS Traffic URL Contains Bluetooth MAC Address

The application transmits the device's Bluetooth MAC Address in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Low

Impact

Transmitting the device's Bluetooth MAC Address in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including the device's Bluetooth MAC Address in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

HTTPS Traffic URL Contains Android ID

The application transmits the device's Android ID in the URL of an HTTPS request, which can be cached and potentially exposed in various ways.

Status	Severity
Passed	Low

Impact

Transmitting the device's Android ID in the URL of an HTTPS request poses a security risk. This data can be cached by browsers, proxies or other intermediary systems, potentially exposing it to unauthorized access and compromising the confidentiality of the data.

Remediation

Refrain from including the device's Android ID in the URL of the HTTPS request. Instead, transmit sensitive data within the request body or as HTTP headers.

Insecure Security Provider

The application has been identified to use an unrecognized security provider for cryptographic operations or lacks the use of a well-established security provider. This practice may introduce vulnerabilities, as unrecognized or custom providers might not adhere to industry standards and best practices.

Status	Severity
Passed	Low

Impact

Utilizing an insecure or unrecognized security provider can expose the application to various security risks, including weak encryption, improper handling of cryptographic keys, and susceptibility to known attacks. This can also lead to unauthorized application access and potential data breaches.

Remediation

Review the application's cryptographic implementations to ensure the use of industry-recognized security providers, such as those provided by the Android platform or other well-established libraries. Avoid using custom

security providers unless they have undergone thorough security reviews and testing.

Insecure TLS configuration

The application connects to endpoints using TLS configurations that do not adhere to industry best practices. This includes the use of outdated or insecure ciphers and protocols, which can weaken the overall security of the communication channels.

Status	Severity
Passed	Low

Impact

An insecure TLS configuration can expose the application to various security threats, including man-in-the-middle attacks, eavesdropping, and data tampering. This compromises the confidentiality and integrity of the data transmitted between the application and its endpoints.

Remediation

Review and update the TLS configuration for all endpoints to ensure the use of secure ciphers and protocols. Adhere to industry best practices by disabling outdated protocols and weak ciphers. Regularly consult the latest security guidelines and recommendations to maintain a strong TLS configuration.

Cookie missing 'Secure' flag

Cookies set by the application have been identified without the 'Secure' flag. The 'Secure' flag ensures that cookies are only transmitted over secure HTTPS connections, protecting them from being exposed over unsecured networks.

Status	Severity
Passed	Low

Impact

Without the 'Secure' flag, cookies can be sent over insecure connections, potentially exposing sensitive information to attackers. This can lead to session hijacking, data leakage and other potential security vulnerabilities.

Remediation

Review and update the application's cookie management to ensure that all cookies intended for secure contexts are set with the 'Secure flag'. This ensures cookies are only transmitted over HTTPS connections, enhancing the security of user data and sessions.

Application Target SDK Allows for Insecure Network Configuration

The application's target SDK version is set to a value less than 29. This may result in the application not leveraging the latest secure network configurations available in the newer Android versions. Allowing your application to be run on older versions can potentially result in sensitive data being exposed.

Status	Severity
Passed	Low

Impact

Setting the target SDK version to less than 29 poses a low risk. The application may lack support for the latest network configurations, leaving it vulnerable to potential attacks targeting outdated protocols.

Remediation

Update the application's target SDK version to 29 or above to ensure compatibility with the latest secure network configurations provided by newer Android versions.

Cookie missing 'httpOnly' flag

Cookies set by the application have been identified without the 'httpOnly' flag. The 'httpOnly' flag prevents client-side scripts from accessing cookies, reducing the risk of cross-site scripting (XSS) attacks.

Status	Severity
Passed	Low

Impact

Without the 'httpOnly' flag, cookies are accessible to client-side scripts, making them vulnerable to XSS attacks. Attackers can potentially steal sensitive information, manipulate sessions or perform various malicious actions.

Remediation

Review and update the application's cookie management to ensure that all cookies containing sensitive information are set with the 'httpOnly' flag. This restricts access to cookies from client-side scripts, mitigating the risks of XSS attacks.

Application Contains Heartbleed Vulnerable OpenSSL Version

The application contains a version of OpenSSL that is known to be vulnerable to the Heartbleed CVE. The Heartbleed vulnerability could potentially allow an attacker to read sensitive data from the application's memory.

Status	Severity
Passed	Low

Impact

An attacker exploiting this vulnerability may gain unauthorized access to sensitive data transmitted by the application therefore compromising both data confidentiality and integrity.

Remediation

Update the OpenSSL library to a version that addresses the Heartbleed vulnerability. Regularly monitor for security updates and vulnerabilities in third-party libraries to maintain the security of network communication.

Application Contains Change Cipher Spec Injection Vulnerable OpenSSL Version

The application contains a version of OpenSSL that is known to be vulnerable to Change Cipher Spec (CCS) Injection. This vulnerability may allow an attacker to manipulate the Change Cipher Spec protocol, potentially leading to unauthorized access and manipulation of encrypted communication.

Status	Severity
Passed	Low

Impact

Exploiting the CCS Injection vulnerability may enable an attacker to interfere with the encrypted communication, potentially gaining unauthorized access to application data. This poses a risk to both data confidentiality and integrity.

Remediation

Update the OpenSSL library to a version that addresses the Change Cipher Spec Injection vulnerability. Regularly monitor for security updates and vulnerabilities in third-party libraries to ensure the security of network communication.

Application Contains Insecure HTTP Traffic

The application utilizes HTTP endpoints, potentially exposing sensitive information to interception. The use of unencrypted HTTP endpoints poses a security risk as transmitted information may be intercepted or modified by unauthorized parties.

Status	Severity
Passed	Info

Impact

The use of unencrypted HTTP endpoints exposes transmitted data to potential interception and manipulation. This poses a risk to both the data confidentiality and integrity for any data transmitted between the application and remote servers.

Remediation

Identify and assess all HTTP endpoints used by the application. Implement secure network communication practices by transitioning to HTTPS for sensitive endpoints.

HTTP Cleartext Transmission of WiFi MAC Address

The application transmits the device's WiFi MAC Address over HTTP in cleartext, exposing the sensitive value to potential interception and unauthorized access.

Status	Severity
Passed	Medium

Impact

Transmitting the device's WiFi MAC Address over HTTP in cleartext poses a security risk. Attackers can potentially intercept and retrieve this sensitive information, compromising the confidentiality and integrity of the data.

Remediation

Implement secure network communication practices by ensuring that sensitive information, such as the device's WiFi MAC Address, is transmitted over HTTPS.

PLATFORM

The security of mobile apps heavily depends on their interaction with the mobile platform, which often involves exposing data or functionality intentionally through the use of platform-provided inter-process communication (IPC) mechanisms and WebViews to enhance the user experience. However, these mechanisms can also be exploited by attackers or other installed apps, potentially compromising the app's security. Furthermore, sensitive data, such as passwords, credit card details, and one-time passwords in notifications, is often displayed in the app's user interface. It is essential to ensure that this data is not unintentionally leaked through platform mechanisms such as auto-generated screenshots or accidental disclosure through shoulder surfing or device sharing. This category comprises controls that ensure the app's interactions with the mobile platform occur securely. These controls cover the secure use of platform-provided IPC mechanisms, WebView configurations to prevent sensitive data leakage and functionality exposure, and secure display of sensitive data in the app's user interface. By implementing these controls, mobile app developers can safeguard sensitive user information and prevent unauthorized access by attackers.

P1

PLATFORM-1

The app uses IPC mechanisms securely.

Apps typically use platform provided IPC mechanisms to intentionally expose data or functionality. Both installed apps and the user are able to interact with the app in many different ways. This control ensures that all interactions involving IPC mechanisms happen securely.

Application Manifest Contains Unverified Deep Links

The Android manifest contains unverified deep links, which can potentially expose users to phishing attacks or unintended navigation.

Status	Severity
Passed	Medium

Impact

An attacker can exploit unverified deep links to redirect users to malicious sites or manipulate the flow of the application

Remediation

Implement proper verification mechanisms for deep links in the Android Manifest. Utilize secure protocols, such as HTTPS, to ensure the integrity of deep links. Validate deep links on the server side to prevent manipulation.

Application Utilizes Intents Vulnerable to Redirection

The application has registered intents that are vulnerable to redirection attacks, allowing potential attackers to manipulate the intended flow of the application.

Status	Severity
Failed	Low

Impact

Intent redirection vulnerabilities may lead to unauthorized actions, compromise user privacy, or expose sensitive information.

Remediation

Implement secure intent handling mechanisms to prevent redirection attacks. Ensure that all intents are properly validated, and use explicit intents where possible. Avoid relying solely on implicit intents, and validate input parameters to ensure their integrity.

Evidence

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
com.corellium.cafe.ui.activities.SecretActivity
```

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
com.corellium.cafe.ui.activities.MainActivity
```

Application Utilizes Broadcast Receivers Without Permissions

The application contains broadcast receivers that are exported without proper permissions, potentially exposing sensitive information and allowing unauthorized parties to interact with the receivers.

Status	Severity
Passed	Low

Impact

Broadcast receivers without proper permissions may lead to unauthorized access, manipulation, or disclosure of sensitive data compromising the overall security and integrity of the application.

Remediation

Review and update the AndroidManifest.xml file to ensure that exported broadcast receivers have the appropriate permissions defined.

Application Utilizes Potentially Dangerous Permissions

The application requests and registers dangerous permissions within the AndroidManifest. Dangerous permissions without a valid reason may expose users to potential privacy and security risks, as the application gains access to sensitive data or device features.

Status	Severity
Failed	Info

Impact

The use of dangerous permissions may grant the application access to sensitive data or device functionalities without explicit user consent. This

vulnerability could lead to unauthorized actions, data exposure, or misuse of privileged capabilities.

Remediation

Conduct a thorough review of the application's requested permissions and only request permissions that are essential for the application's core features.

Evidence

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
android.permission.WRITE_EXTERNAL_STORAGE
```

P2

PLATFORM-2

The app uses WebViews securely.

WebViews are typically used by apps that have a need for increased control over the UI. This control ensures that WebViews are configured securely to prevent sensitive data leakage as well as sensitive functionality exposure (e.g. via JavaScript bridges to native code).

Application Contains Debuggable WebViews

The application code contains debuggable WebViews within. Debuggable WebViews may allow attackers, using debugging tools, to access data during runtime, leading to potential disclosure of sensitive data.

Status	Severity
Failed	Low

Impact

Attackers may exploit this vulnerability during runtime to access debug information from the affected WebViews potentially leading to unauthorized access and exposure of sensitive data.

Remediation

Disable debugging features for WebViews in the application code before releasing the production version. Ensure that debuggable attributes are set to 'false', especially for webViews handling sensitive data.

Evidence

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
setWebContentsDebuggingEnabled(true)
```

Application Utilizes Insecure JavaScript Interface

The application code utilizes an insecure JavaScript interface, potentially exposing sensitive data to unauthorized access. Insecure usage of 'addJavascriptInterface' may allow attackers to execute arbitrary code.

Status	Severity
Failed	Low

Impact

The presence of an insecure JavaScript interface in the application code introduces a significant impact. Exploitation of this vulnerability may lead to unauthorized access and potential manipulation of sensitive data. Attackers could execute arbitrary code, posing risks to user privacy and application integrity.

Remediation

Review and update the application code to ensure secure usage of 'addJavascriptInterface'. Limit the use of JavaScript interfaces to only essential functionalities, and avoid exposing sensitive data. Implement proper input validation and sanitization to prevent malicious code execution.

Evidence

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto$2.java

```
@JavascriptInterface
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
addJavascriptInterface
```

Application Contains WebViews with Javascript Enabled

The application incorporates WebViews with JavaScript enabled, potentially exposing users to various security concerns. Enabling JavaScript in WebViews may allow attackers to execute malicious scripts.

Status	Severity
Failed	Low

Impact

The utilization of WebViews with JavaScript enabled introduces a potential security risk. Attackers could exploit this vulnerability to execute malicious scripts. The consequences may include unauthorized actions, unintended data exposure, and other security threats.

Remediation

Review and assess the necessity of enabling JavaScript in WebViews. If possible, disable JavaScript in WebViews or implement proper input validation and sanitization to mitigate security risks.

Evidence

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/ui/activities/WebViewActivity.java

```
setJavaScriptEnabled(true)
```

File: /data/local/tmp/artifacts/output/sources/com/corellium/cafe/crypto/InsecureCrypto.java

```
setJavaScriptEnabled(true)
```

P3

PLATFORM-3

The app uses the user interface securely.

Sensitive data has to be displayed in the UI in many situations (e.g. passwords, credit card details, OTP codes in notifications). This control ensures that this data doesn't end up being unintentionally leaked due to platform mechanisms such as auto-generated screenshots or accidentally disclosed via e.g. shoulder surfing or sharing the device with another person.

Application is Vulnerable to Task Hijacking

The application may be susceptible to task hijacking, a vulnerability where malicious applications can intercept and manipulate tasks. This issue arises in applications with a minimum SDK version below 29 and specific activity configurations that can be exploited.

Status	Severity
Passed	Medium

Impact

Task hijacking can allow malicious applications to take control of tasks, leading to security risks such as unauthorized access, data leakage, and spoofing attacks. This compromises the application's overall security and the user's trust.

Remediation

To mitigate the risk of task hijacking, ensure that the application adheres to secure task management best practices. For applications with a minimum SDK version below 29, review the AndroidManifest file to identify activities with 'launchmode='singleTask'. For such activities, set 'taskAffinity=""' to

prevent task hijacking. Additionally, consider updating the application's minimum SDK version to 29 or higher to leverage enhanced security features.

RESILIENCE

Defense-in-depth measures such as code obfuscation, anti-debugging, anti-tampering, etc. are important to increase app resilience against reverse engineering and specific client-side attacks. They add multiple layers of security controls to the app, making it more difficult for attackers to successfully reverse engineer and extract valuable intellectual property or sensitive data from it

R2

RESILIENCE-2

The app implements anti-tampering mechanisms.

Apps run on a user-controlled device, and without proper protections it's relatively easy to run a modified version locally (e.g. to cheat in a game, or enable premium features without paying), or upload a backdoored version of it to third-party app stores. This control tries to ensure the integrity of the app's intended functionality by preventing modifications to the original code and resources.

Janus Exploitation Possible Due to Current Minimum SDK

The application is potentially vulnerable to the Janus exploit due to a low minimum SDK version. This attack would allow attacks to inject their own code into the binary package and release it as a legitimate update to the application.

Status	Severity
Failed	Low

Impact

The application can be replaced with a malicious version without your knowledge. The application could then be utilized to steal sensitive data or perform any number of additional attacks while pretending to be a legitimate application.

Remediation

The application should require a minimum SDK of '24' or higher and utilize a signing scheme more recent than v1. Please note this vulnerability only looks at the minimum SDK and the signing scheme is checked separately.

Evidence

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
android:minSdkVersion=23
```

Application Utilizes a Weak Signing Key

Due to the deprecated status of 1024-bit RSA keys by the NIST standards, the use of such keys in your application's signing configuration presents a significant security concern. This outdated key length poses inherent vulnerabilities, potentially resulting in compromising your application's overall security posture.

Status	Severity
Passed	Medium

Impact

The use of a weak signing key (1024-bit RSA) exposes your Android application to various security risks. These risks can include tampering with the application's code or resources, injecting malicious payloads, or impersonating the applications identity through fake updates.

Remediation

To mitigate this vulnerability, developers should move to using a stronger signing key length recommended by industry standards (2048-bits minimum).

Application Utilizes a Weak Signing Scheme

The vulnerability relates to only using the V1 signing scheme for the Android application. V1 scheme solely validates ZIP file entries, making it susceptible to exploitation where attackers can inject malicious code without invalidating the APK.

Status	Severity
Passed	Medium

Impact

Exclusive reliance on the V1 signing scheme exposes the application to significant risks. Attackers can modify an application with malicious code without detection leading to unauthorized code execution, data breaches and compromised user privacy.

Remediation

To mitigate this vulnerability, developers should implement a multi-layered approach to the APK signing by incorporating both V1 and V2 signing schemes. Leveraging V2 signatures ensures the integrity of the entire APK while optimizing verification speed.

R3

RESILIENCE-3

The app implements anti-static analysis mechanisms.

Understanding the internals of an app is typically the first step towards tampering with it (either dynamically, or statically). This control tries to impede comprehension by making it as difficult as possible to figure out how an app works using static analysis.

Application Contains Debug Library

The Android application contains a debug library, which may expose sensitive information and increase the overall attack surface.

Status	Severity

Failed	Low

Impact

An attacker could potentially leverage the presence of the debug library to gain insights into the application's internal workings, posing an increased security risk.

Remediation

Remove application debug libraries in the production version of the application to minimize the risk of information exposure and enhance the overall security.

Evidence

File: /data/local/tmp/artifacts/output/sources/unknown/DebugProbesKt.bin

File: /data/local/tmp/artifacts/output/resources/DebugProbesKt.bin

R4

RESILIENCE-4

The app implements anti-dynamic analysis techniques.

Sometimes pure static analysis is very difficult and time consuming so it typically goes hand in hand with dynamic analysis. Observing and manipulating an app during runtime makes it much easier to decipher its behavior. This control aims to make it as difficult as possible to perform dynamic analysis, as well as prevent dynamic instrumentation which could allow an attacker to modify the code at runtime.

Application Enables Debugging within the Android Manifest

The Android application enables debugging within the AndroidManifest, potentially exposing sensitive information and increasing the attack surface.

Status	Severity
Failed	Info

Impact

Enabling debugging in the manifest may expose sensitive information and pose a security risk, providing attackers with insights into the application's internal workings.

Remediation

Disable debugging in the AndroidManifest.xml file before releasing the application to enhance security and protect sensitive information.

Evidence

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
android:debuggable='true'
```

STORAGE

Mobile applications handle a wide variety of sensitive data, such as personally identifiable information (PII), cryptographic material, secrets, and API keys, that often need to be stored locally. This sensitive data may be stored in private locations, such as the app's internal storage, or in public folders that are accessible by the user or other apps installed on the device. However, sensitive data can also be unintentionally stored or exposed to publicly accessible locations, typically as a side-effect of using certain APIs or system capabilities such as backups or logs. This category is designed to help developers ensure that any sensitive data intentionally stored by the app is properly protected, regardless of the target location. It also covers unintentional leaks that can occur due to improper use of APIs or system capabilities.

S1

STORAGE-1

The application securely stores sensitive data.

Applications handle sensitive data coming from many sources including the user, the backend, system services or other applications on the device and often need to store it locally. The storage locations may be private to the application (e.g. its internal storage) or be public and therefore accessible by the user or other installed applications (e.g. public folders such as Downloads). This control ensures that any sensitive data that is intentionally stored by the application is properly protected independently of the target location.

Local Data Exposure: DNS Address Stored Insecurely

This test evaluates how the application handles DNS addresses within its local data directory. Best practices dictate that sensitive details should be protected adequately, utilizing encryption or obfuscation techniques. The specific focus of this test is to determine whether the application has stored a DNS address in plaintext, making it easily readable and accessible without any additional processing.

Status

Severity

Passed	Low

Impact

Storing a DNS Address in a plaintext format within the local data directory presents a significant risk. If these files are not adequately safeguarded, anyone with device access could retrieve this DNS Address. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing information in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: GPS Longitude Stored Insecurely

This test evaluates how the application handles user GPS data, specifically longitude coordinates within the local data directory. Best practices dictate that sensitive information should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored a user's GPS longitude in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Medium

Impact

Storing a user's GPS longitude in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the associated value if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive values in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: GPS Latitude Stored Insecurely

This test evaluates how the application handles user GPS data, specifically latitude coordinates within the local data directory. Best practices dictate that sensitive information should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored a user's GPS latitude in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Medium

Impact

Storing a user's GPS latitude in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the associated value if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive values in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: Sensitive Values Stored Insecurely on Device

This test evaluates how the application handles the storage of the provided sensitive values within the local data directory. Best practices dictate that sensitive values should be protected adequately, utilizing encryption or obfuscation techniques. The specific focus of this test is to determine whether the application has stored any of the provided sensitive values in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Failed	High

Impact

Storing sensitive values in plaintext within the local application data directory presents a significant risk. If these files are not adequately safeguarded, anyone with device access could retrieve the values. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing information in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.corellium.cafe/databases/CUSTOMER_PAYMENT

```
1234567891011121
```

File: /data/data/com.corellium.cafe/shared_prefs/Customer Data.xml

```
<string name="First Name">David</string>
```

File: /data/data/com.corellium.cafe/shared_prefs/Customer Data.xml

```
<string name="Last Name">Walker</string>
```

Local Data Exposure: Global Write Permissions

This test evaluates how the application utilizes global write permissions for potentially sensitive files within the local data directory. Best practices dictate that application files containing sensitive information should be protected adequately, utilizing encryption, obfuscation and sufficient file permissions. The specific focus of this test is to determine whether the application has stored sensitive files with global write permissions, making the files easily accessible outside the local data directory.

Status	Severity
Passed	Medium

Impact

Storing sensitive files within the local application data directory with global write permissions presents a significant risk. Anyone with device access or any device application could review and modify the contents of the files.

Remediation

Use the appropriate file permissions to grant access only to specific users or processes that require it, preventing unauthorized modifications. Utilize content providers if other applications are required to access data that is stored by your application to avoid additional unauthorized access.

Local Data Exposure: Global Read Permissions

This test evaluates how the application utilizes global read permissions for potentially sensitive files within the local data directory. Best practices dictate that application files containing sensitive information should be protected adequately, utilizing encryption, obfuscation and sufficient file permissions. The specific focus of this test is to determine whether the application has

stored sensitive files with global read permissions, making the files easily readable outside the local data directory.

Status	Severity
Passed	Low

Impact

Storing sensitive files within the local application data directory with global read permissions presents a significant risk. Anyone with device access or any device application could review the contents of the files.

Remediation

Use the appropriate file permissions to grant access only to specific users or processes that require it, preventing unauthorized access. Utilize content providers if other applications are required to access data that is stored by your application to avoid additional unauthorized access.

Local Data Exposure: Bluetooth MAC Address Stored Insecurely

This test evaluates how the application handles the device's Bluetooth MAC address within the local data directory. Best practices dictate that sensitive values should be protected adequately, utilizing encryption or obfuscation techniques. The specific focus of this test is to determine whether the application has stored the device's Bluetooth MAC address in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Medium

Impact

Storing the device's Bluetooth MAC address in plaintext within the local application data directory presents a significant risk. If these files are not adequately safeguarded, anyone with device access could retrieve the device's Bluetooth MAC address. This exposure makes the associated

information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing information in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: Insecure Hardcoded API Keys

This test evaluates how the application handles API keys within the local data directory. Best practices dictate that sensitive API keys should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored sensitive API keys in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Medium

Impact

Storing sensitive API keys in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the hardcoded API keys if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing information in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: Insecure Hardcoded Passwords

This test evaluates how the application handles hardcoded passwords within the local data directory. Best practices dictate that sensitive values, including passwords, should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored any password values hardcoded in plaintext, making them easily readable and accessible without any additional processing.

Status	Severity
Passed	Medium

Impact

Storing sensitive password values in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the password values if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive values in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: WiFi MAC Address Stored Insecurely

This test evaluates how the application handles the device's WiFi MAC Address within the local data directory. Best practices dictate that sensitive information should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored the device's WiFi MAC Address in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity

Passed	Medium

Impact

Storing the device's WiFi MAC Address in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the associated value if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive values in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: WiFi IP Address Stored Insecurely

This test evaluates how the application handles the device's WiFi IP Address within the local data directory. Best practices dictate that sensitive information should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored the device's WiFi IP Address in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Low

Impact

Storing the device's WiFi IP Address in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the associated value if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive values in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: Device IMEI Stored Insecurely

This test evaluates how the application handles the device's IMEI value within the local data directory. Best practices dictate that sensitive information should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored the device's IMEI value in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Medium

Impact

Storing the device's IMEI value in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the associated value if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive values in plaintext, especially if it's not required for the application's functionality.

Local Data Exposure: Android ID Stored Insecurely

This test evaluates how the application handles the Android ID within the local data directory. Best practices dictate that sensitive information should be protected adequately, utilizing encryption, obfuscation and platform-specific secure storage solutions. The specific focus of this test is to determine whether the application has stored the Android ID in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Medium

Impact

Storing the Android ID in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve the associated value if these files are not adequately safeguarded. This exposure makes the associated information or services more susceptible to unauthorized access and can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt sensitive values using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive values in plaintext, especially if it's not required for the application's functionality.

S2

STORAGE-2

The app prevents leakage of sensitive data.

There are cases when sensitive data is unintentionally stored or exposed to publicly accessible locations; typically as a side-effect of using certain APIs, system capabilities such as backups or logs. This control covers this kind of unintentional leaks where the developer actually has a way to prevent it.

Local Data Exposure: WiFi IP Address Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should

not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored the device's WiFi IP Address within logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Low

Impact

Storing a device's WiFi IP Address within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: Bluetooth MAC Address Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored a Bluetooth MAC Address within the logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Low

Impact

Storing a Bluetooth MAC Address within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: Sensitive Values Stored in Memory

This test evaluates how the application handles sensitive data within the Android device memory during runtime. Best practices dictate that sensitive information should not be stored in memory longer than necessary. The specific focus of this test is to determine whether the application clears sensitive data from memory promptly after its intended use.

Status	Severity
Failed	Low

Impact

Capturing sensitive data within the device memory during runtime poses a potential risk. If sensitive information remains in memory after its intended use, it could be accessible to potential malware running on a device or attackers who gain unauthorized access to the device. This exposure may lead to unauthorized access to sensitive data and increase the likelihood of data breaches or privacy violations.

Remediation

Ensure sensitive data is promptly cleared from memory after its intended use, especially upon application termination. Implement secure coding practices to prevent unintentional data retention in memory and consider using encryption or secure data wiping techniques where applicable.

Evidence

Keyword: 1234567891011121

```
0x13328300
0x136f8260
0x6d68a91c21
0x6e48b02a5c
```

Keyword: David

```
0x13662d80
0x136cec48
0x6d68a91bef
```

Keyword: Walker

```
0x13662d48
0x136cc970
0x6d68a91c03
```

Keyword: 00000000-0000-0000-0000-000000000000

```
0x705c2df0
```

Local Data Exposure: Device IMEI Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored the device's IMEI within logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	**Low**

Impact

Storing a device's IMEI within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: Application Backups Enabled

This test evaluates if the 'allowbackup' flag is set to 'true' within the AndroidManifest. The 'AllowBackup' flag can be used to backup the entire application data directory exposing the contents to anyone with access.

Status	Severity
Failed	Medium

Impact

Enabling 'allowBackup' poses a business risk, as it permits extraction of sensitive application data, undermining mobile security controls and data confidentiality.

Remediation

Set the 'allowBackup' flag to 'false' within the AndroidManifest. If a backup is necessary for the application, it is possible to define backup conditions including utilizing encryption.

Evidence

File: /data/local/tmp/artifacts/output/resources/AndroidManifest.xml

```
android:allowBackup="true"
```

Local Data Exposure: Device Fingerprint Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored the device's build fingerprint within logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Low

Impact

Storing a device's build fingerprint within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: WiFi MAC Address Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored the WiFi MAC Address within logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Low

Impact

Storing a WiFi MAC Address within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: GPS Longitude Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored GPS Coordinates within the logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Low

Impact

Storing GPS Coordinates within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: GPS Latitude Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored GPS Coordinates within the logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	Low

Impact

Storing GPS Coordinates within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: Sensitive Data Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored any sensitive data provided within the logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	High

Impact

Storing sensitive data within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.

Local Data Exposure: DNS Address Logged Insecurely

This test evaluates how the application handles sensitive data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is required to be used within the logs the data should be protected adequately, utilizing encryption. The specific focus of this test is to determine whether the application has stored a DNS address within the logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity
Passed	**Low**

Impact

Storing a DNS address within the Android device logs in plaintext presents a significant risk. Anyone with device access could retrieve the logs, and this exposure can provide valuable data to adversaries for orchestrating follow-up attacks.

Remediation

Always encrypt log data using industry-accepted algorithms and ensure only required data is being written to the device logs.