CORELLIUM®

Automated Mobile Application Security Test Report

Date	App
December 7th, 2023	com.corellium.cafe

MASVS-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 MASVS-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	Analysis
Info	Low	

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/functions/FunctionsKt$sendPost$1.java

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

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

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

MASVS-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 MASVS-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	Analysis
Fail	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: AndroidManifest.xml

```
MinSDK: 23
```

C2 MASVS-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	Analysis
Fail	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 concering for applications that process PNG image files.

Remediation

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

Evidence

File: libpng.so

```
Version: 1.5.28
```

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 process color quantization tables. These vulnerabilities can be exploited using specially crafted JPEG files.

Status	Severity	Analysis
Fail	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

Evidence

File: libturbojpeg.so

```
Version: 2.0.0
```

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	Analysis
Fail	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

Evidence

File: OkHttp_version

```
Version: 3.1.10
```

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	Analysis
Fail	Low	

Impact

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

Remediation

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

Evidence

File: /data/local/tmp/artifacts/output/resources/lib/arm64-v8a/joda-time-2.1.4.jar

MASVS-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 MASVS-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 implementrations. 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	Analysis
Fail	Low	

Impact

The use of insecure cryptographic algorithms can lead to unathorized 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/CryptoManager.java

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

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

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

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

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

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

```
Cipher rc2Cipher = Cipher.getInstance('RC2');
```

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

```
KeyGenerator rc2KeyGen = KeyGenerator.getInstance('RC2');
```

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

```
MessageDigest md5Digest = MessageDigest.getInstance('MD5');
```

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

```
MessageDigest sha1Digest = MessageDigest.getInstance('SHA-1');
```

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	Analysis
Fail	Low	

Impact

Using ECB mode (Electronic Codebook) or 'NoPadding' with symmetric encryption exposes sensitive data to possible descryption by unathorized 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 initilization 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/ui/activities/encryption.java

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

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

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

Application Utilizes Null Initialization Vectors

The application's cryptographic implementation utilizes a null or zero Initialization Vector (IV) when encrypting data. This practice undermines the security provided by the encryption algorithm, as it makes the encrypted data more susceptible to various cryptographic attacks, particularly when the same key is used for multiple encryptions.

Status	Severity	Analysis
Fail	Low	

Impact

The use of a null or predictable IV compromises the encryption scheme's effectiveness This can result in the exposure of sensitive data and violate data confidentiality.

Remediation

Implement the generation of a unique, cryptographically strong random IV for each encryption operation. Ensure that the IV is properly generated by a secure random function. This change is necessary to maintain the integrity and confidentiality of the encryption scheme and meet industry encryption standards.

Evidence

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

```
private static final byte[] keyValue = new byte[16]{'T', 'h', 'e', 'B', 'e', 's', 't', 'S', 'e', 'c', 'r', 'e', 't', 'K', 'e', 'y'};
```

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

```
private static final byte[] ivValue = new byte[32] {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

C2 MASVS-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	Analysis
Fail	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 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/CryptoManager.java

```
String pragmaKeyQuery = 'PRAGMA key=SuperSecret_SQLCipher_Key';';
```

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

```
database.rawExecSQL('PRAGMA key = 'hardcoded_secret_key123'');
```

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 utlizes maliciously to access sesitive data or perform unauthorized actions.

Status	Severity	Analysis
Fail	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.

Evidence

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

```
API_KEY=4534543739243290840328094
```

MASVS-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 MASVS-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	Analysis
Fail	Low	

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	Analysis
Pass	Low	

Impact

Transmitting the device's IMEI over HTTP in cleartext poses a security risk. Attackers can potentally 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	Analysis
Pass	Low	

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	Analysis
Pass	Low	

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	Analysis
Pass	Medium	

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	Analysis
Pass	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	Analysis
Pass	Low	

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	Analysis
Pass	**Low**	

Impact

Transmitting the device's Bluetooth MAC Address over HTTP is standard practice in security tools. Attackers can passively 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 unauthorized interception and unauthorized access.

Status	Severity	Analysis
Pass	**Low**	

Impact

Transmitting the device's Android ID over HTTP in cleartext poses a security risk. Attackers can passively 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.

Application VPN Data Not Encrypted

The application's network traffic is not encrypted at the Secure Sockets Layer (SSL). The configuration may expose the application's network traffic to potential interception and compromise the effectiveness of the VPN.

Status	Severity	Analysis
Fail	**Low**	

Impact

Failure to encrypt all the lifeops On Android in a VPN configuration poses a low risk. It may result in intermittent loss of secure VPN connections, potentially exposing sensitive data to unauthorized parties.

Remediation

Reconfigure the application's VPN settings to enable the lifeops On Android, ensuring a persistent secure connection and securing the likelihood of unintended data exposure.

Evidence

TTS: **android.bluetooth.net.pan**

- **true**

Application Target SDK Allows for Insecure Network Configuration

The application's Target SDK version is set to an older one that SDK 19. This may result in the application not enforcing the latest secure network configurations available in the Google Android ecosystem. Attacking your application's data to be at risk to known vulnerabilities, which a newer version, data leaked.

Status	Severity	Analysis
Fail	**Low**	

Impact

Setting the target SDK version to less than 28 poses a low risk. The application may lack support for the latest security configurations, leaving it potentially susceptible to known vulnerabilities.

Remediation

Update the application's target SDK version to 28 or above to ensure compatibility with the latest secure network configurations provided by Google Android versions.

Evidence

TTS: **androidtarget.sdk.ver**

- **TargetSDK: 23**

Application Contains Hardlined Vulnerable OpenSSL Version

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

Status	Severity	Analysis
Fail	**Low**	

Impact

By provoke exploiting this vulnerability may gain unauthorized access to sensitive data transmitted by the application's libraries compromising both data confidentiality and integrity.

Remediation

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

Evidence

TTS: **libssl.so**

- **OpenSSL 1.0.1k**

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 hacktrace the Change Cipher Spec protocol, potentially leading to unauthorized access and manipulation of encrypted communication.

Status	Severity	Analysis
Fail	**Low**	

Impact

By provoke exploiting this vulnerability may gain unauthorized access to sensitive data transmitted by the application libraries compromising 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 maintain the security of network communications.

Evidence

TTS: **libssl.so**

- **OpenSSL 1.0.1k**

HTTP Cleartext Transmission of Wifi IP Address

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

Status	Severity	Analysis
Pass	**Low**	

Impact

Transmitting the device's Wifi IP address over HTTP in cleartext poses a security risk. Attackers can passively 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 IP address, is transmitted over HTTPS.

Evidence

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:3D:fe0:0000:ae:1d7:fe0:ge:1**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:3D:fe0:0000:ae:1d7:fe0:ge:2**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:3D:fe0:0000:ae:1d7:fe0:ge:3**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:1D:fe0:0000:ae:1d7:fe0:ge:4**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:1D:fe0:0000:ae:1d7:fe0:ge:5**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:1D:fe0:0000:ae:1d7:fe0:ge:6**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:1D:fe0:0000:ae:1d7:fe0:ge:7**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:1D:fe0:0000:ae:1d7:fe0:ge:8**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:1D:fe0:0000:ae:1d7:fe0:ge:9**

TTS: **androidnetworkinterfaceinfo.getinetaddresses()**

- **W/W:0A:0F:B7:1F:10:7C:2:AF0:B = 19840:1E0:2c:A43:7fa:ge:2C:ieq:1D:fe0:0000:ae:1d7:fe0:ge:10**

Application Contains Insecure HTTP Traffic

The application contains HTTP traffic, which may be transmitting sensitive information in transmission. The use of unencrypted HTTP connections poses a security risk, as it exposes data to potential interception or modification by unauthorized parties.

Status	Severity	Analysis
Pass	**Low**	

Impact

The use of unencrypted HTTP endpoints exposes transmitted data to potential interception and manipulation. Attackers can exploit this vulnerability to eavesdrop on or alter data transmitted between the application and backend services.

Remediation

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

HTTP Cleartext Transmission of Device Serial Number

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

Status	Severity	Analysis
Pass	**Low**	

Impact

Transmitting the device's Serial Number 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 Serial Number, is transmitted over HTTPS.

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	Analysis
Pass	**Low**	

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.

MASVS-PLATFORM

The security of mobile apps heavily depends on their interaction with the mobile platform, which often involves processing data from untrusted sources. Platform-specific APIs and inter-process communication (IPC) mechanisms are fundamental for functionality and for enhancing the user experience. However, these interactions can pose vulnerabilities if attackers or other installed apps potentially compromising the app's security. Furthermore, sensitive data, such as authentication tokens or user preferences, transmitted through these platform interfaces, must be handled with care to ensure that the data is not unintentionally leaked through platform mechanisms such as logs, shared memory, or persistent storage. Secure coding practices are essential to mitigate these risks, ensuring that the app interacts with the platform in a secure manner while maintaining the confidentiality and integrity of user data.

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 through these interfaces. This control ensures that an attacker cannot exploit IPC mechanisms beyond expected.

Application Manifest Contains Unverified Deep Links

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

Status	Severity	Analysis
Fail	**Low**	

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, and ensure the integrity of deep links. Validate deep links on the server side to prevent unauthorized navigation.

Evidence

TTS: **AndroidManifest.txt**

- **com.zoomfocus.ui.activities.SecretActivity**

TTS: **AndroidManifest.txt**

- **com.zoomfocus.ui.activities.MainActivity**

TTS: **AndroidManifest.txt**

- **SuperSecretReceiver**

TTS: **AndroidManifest.txt**

- **SuperSecretPERMISSIONS**

Application Utilizes Intents Vulnerable to Redirection.

The application uses implemented intents, which are susceptible to redirection attacks, allowing potential attackers to manipulate the intended flow of the application.

Status	Severity	Analysis
Fail	**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

TTS: **AndroidManifest.txt**

- **com.zoomfocus.ui.activities.SecretActivity**

TTS: **AndroidManifest.txt**

- **com.zoomfocus.ui.activities.MainActivity**

TTS: **AndroidManifest.txt**

- **SuperSecretReceiver**

TTS: **AndroidManifest.txt**

- **SuperSecretPERMISSIONS**

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	Analysis
Fail	**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.txt file to ensure that exported broadcast receivers have the appropriate permissions defined.

Application Utilizes Potentially Dangerous Permissions

The application requests permissions that are potentially dangerous, such as READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, or SYSTEM_ALERT_WINDOW. These permissions can pose security risks, as they may expose users to potential privacy and security risks, or the application gains access to sensitive data or device features.

Status	Severity	Analysis
Info	**Low**	

Impact

The use of dangerous permissions may grant the application access to sensitive data or device functionalities, which could be exploited by malicious actors if the application is compromised. This may lead to data breaches, or other forms 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

TTS: **AndroidManifest.txt**

- **android.permission.WRITE_EXTERNAL_STORAGE**

M3 Insecure Communication

The app uses WebViews securely.

Attackers can typically exploit web apps that are not hardened against common web attacks. This control ensures that WebViews are configured securely to prevent sensitive data leakage as well as sensitive functionality exposure by an attacker-controlled web page.

Application Contains Debuggable WebViews

The application contains debuggable WebViews which, JavaScript WebViews may allow attackers, using debugging tools, to access data being transmitted, leading to potential disclosure of sensitive data.

Status	Severity	Analysis
Fail	**Low**	

Impact

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

Remediation

Ensure that debug features for WebViews in the application code before releasing the production version. Review the debuggable attributes set in "false", especially for WebViews handling sensitive data.

Evidence

TTS: **androidwebviewdebugging/com/adobe/phonegap/cordova/CordovaWebViewImpl.smali**

- **WebViewImpl/com/adobe/phonegap/cordova/CordovaWebViewImpl.smali**

Application Utilizes Insecure JavaScript Interface

The application contains WebViews with insecure JavaScript interfaces, potentially exposing sensitive data to unauthorized access. Insecure usage of setJavascriptInterface may allow attackers to execute arbitrary code.

Status	Severity	Analysis
Fail	**Low**	

Impact

The presence of an insecure JavaScript interface in the application code introduces a significant risk. Exploitation of this vulnerability may lead to unauthorized access and potential manipulation of sensitive data. Attackers could exploit this interface to execute arbitrary code, compromising the app's security and potentially stealing sensitive data.

Remediation

Review and update the application's code to ensure usage of "setJavascriptInterface". Limit the use of JavaScript interfaces to only essential functionalities, and implement robust input validation and sanitization to prevent malicious code execution.

Evidence

TTS: **androidwebviewjavascriptinterface/com/zoomfocus/ui/activities/WebViewActivity.java**

- **[WebView(JS_ProtCacheFilesyDJS.ID.addJavaObject(getWebImpl().get().setWebViewClient(

MASVS-RESILIENCE

Pending

The app implements anti-tampering mechanisms.

Apps run on a user controlled device, and without proper protections it relatively easy to run a modified version of the app. Attackers can, for example, remove ads, bypass IAP checks, or inject malicious code into vulnerable parts of the third-party app libraries. This control helps increase the integrity of the app's intended functionality by preventing unwitting clients to the original code and resources.

Java Exploitation Protection Due to Current Minimum SDK

The application is potentially vulnerable to the Janus exploit due to a low minimum SDK version. This allows attackers to bypass Dalvik bytecode verifiers and inject malicious code into the application's APK file.

Status	Severity	Analysis
Fail	**Low**	

Impact

The application can be replaced with a malicious version without your knowledge. The application could be tricked into loading malicious code, potentially leading to data breaches and other security risks for its legitimate application.

Remediation

The application should require a minimum SDK of 28 or higher and utilize a signing scheme more recent than v2. Please note this vulnerability only looks at the minimum SDK and the signing scheme is checked separately.

Evidence

TTS: **AndroidManifest.txt**

- **MINSDK Version : 23**

R3 Anti-Static Analysis

The app implements anti-static analysis mechanisms.

Understanding the internals of an app is typically the first step towards tampering with it either statically or dynamically. Thus, it makes sense that obfuscation techniques are difficult to provide for figure out how an app works using static analysis.

Application Contains Debug Library

The application contains a debug library, which may expose sensitive information or bypass the overall attack surface.

Status	Severity	Analysis
Fail	**Low**	

Impact

An attacker could potentially manipulate 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 unauthorized exposure and enhance the overall security.

Evidence

TTS: **androidnativelibraryfactory/com/androidnativelibraryfactory/NativeFactory.class**

The app implements anti-dynamic analysis techniques.

Sometimes your static analysis is very difficult and time consuming so it typically goes hand in hand with dynamic analysis. Obscuring and hindering an app's running runtime is much easier to escalate by attackers who have root access to the device. This control helps to prevent hooking, debugging, and so on.

Application Enables Debugging within the Android Manifest

The application's Android Manifest file enables debugging, potentially exposing sensitive information and compromising the app's security.

Status	Severity	Analysis
Fail	**Low**	

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.txt file before releasing the application to enhance security and protect sensitive information.

Evidence

TTS: **androidnativelibraryfactory/com/androidnativelibraryfactory/AndroidManifest.txt**

android.permission.True"

MASVS-STORAGE

Mobile applications handle a wide variety of sensitive data, such as personally identifiable information (PII), authentication credentials, and proprietary business logic. Improper storage of this data can lead to unauthorized disclosure to private locations, such as rooted devices, or logs accessed through adb, or in public folders that are accessible by other apps installed on the device. However, sensitive data can also be unintentionally stored or exposed to public locations, such as cloud storage, or shared with third-party services through embedded tracking libraries or ad networks. This category is designed to help developers ensure that the sensitive data intended to remain private and protected, regardless of the target location. It also covers unintended leaks that can occur due to improper use of APIs or system capabilities.

MSTG-STORAGE-1

The application securely stores sensitive data.

Applications handle sensitive data coming from many sources including the user, the backend, system services, and inter-app communication. This data, which includes personally identifiable information (PII), authentication tokens, or proprietary business logic, must be protected at rest. An internal storage, or a public folder that was unexpectedly opened by an other installed application (e.g. public folders such as Downloads). The control ensures that any sensitive data stored on the device is secured by the application's security parameters (encryption, access control).

Local Data Exposure: DNS Address Stored Insecurely

This test evaluates how the application handles DNS addresses stored on the local data directory. Best practices dictate that sensitive details should be protected adequately, utilizing encryption or obfuscation to prevent unauthorized access. The specific focus of this test is to determine whether the application adheres to planned, training-backed security mandates, and whether unauthorized access attempts are successfully blocked.

Status: Failed Severity: Analysis
Type: Low Confidence: 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 data using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive data in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

DNS Address: 10.0.0

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 details should be protected adequately, utilizing encryption or obfuscation to prevent unauthorized access. The specific focus of this test is to determine whether the application adheres to planned, training-backed security mandates, and whether unauthorized access attempts are successfully blocked.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Storing a user GPS Longitude 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 this GPS Longitude. 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 data using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive data in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

GPS_LONG (short decimal location) = 123.0779

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 details should be protected adequately, utilizing encryption or obfuscation to prevent unauthorized access. The specific focus of this test is to determine whether the application adheres to planned, training-backed security mandates, and whether unauthorized access attempts are successfully blocked.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Storing a user GPS latitude 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 this GPS latitude. 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 data using industry-accepted algorithms before storing them in the application's local data directory. Avoid storing sensitive data in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

GPS_LATI (short decimal location) = 37.4219

Local Data Exposure: Sensitive Values Stored Insecurely on Device

This test evaluates how the application handles sensitive values within the local data directory. Best practices dictate that sensitive values should be protected adequately, utilizing encryption or obfuscation to prevent unauthorized access. The specific focus of this test is to determine whether the application adheres to planned, training-backed security mandates, and whether unauthorized access attempts are successfully blocked.

Status: Failed Severity: Analysis
Type: Medium Confidence: Medium

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 these sensitive 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 sensitive values in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

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 not be granted global write permissions, utilizing encryption, obfuscation and sufficient file permissions. The specific focus of this test is to determine whether the application has granted sensitive files with global write permissions, making the files publicly readable from other applications.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Granting sensitive files access to other applications via global write permissions presents a significant risk. Unauthorized access to these files could lead to data breaches, compromising confidentiality and potentially facilitating further attacks. The specific focus of this test is to determine whether the application has granted sensitive files with global write permissions, making the files publicly readable from other applications.

Remediation

Avoid granting sensitive files 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, but ensure they are configured to avoid additional unauthorized access.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

-rwxrwxrwx 1 root root 420 2023-10-26 06:35

File: /data/data/com.owasp.mstg.uncrackable1/files/keyassessment.xml

-rwxrwxrwx 1 root root 231 2023-10-26 06:35

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

-rwxrwxrwx 1 root root 420 2023-10-26 06:35

File: /data/data/com.owasp.mstg.uncrackable1/files/dataSecret.txt

-rwxrwxrwx 1 root root 87 2023-10-26 06:35

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 not be granted global read permissions, utilizing encryption, obfuscation and sufficient file permissions. The specific focus of this test is to determine whether the application has granted sensitive files with global read permissions, making the files publicly readable from other applications.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Granting sensitive files access to other applications via global read permissions presents a significant risk. Unauthorized access to these files could lead to data breaches, compromising confidentiality and potentially facilitating further attacks. The specific focus of this test is to determine whether the application has granted sensitive files with global read permissions, making the files publicly readable from other applications.

Remediation

Avoid granting sensitive files 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, but ensure they are configured to avoid additional unauthorized access.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/keyassessment.xml

-rwxrwxrwx 1 root root 231 2023-10-26 06:35

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

-rwxrwxrwx 1 root root 420 2023-10-26 06:35

File: /data/data/com.owasp.mstg.uncrackable1/files/dataSecret.txt

-rwxrwxrwx 1 root root 87 2023-10-26 06:35

Local Data Exposure: Bluetooth MAC Address Stored Insecurely

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

Status: Failed Severity: Analysis
Type: Low Confidence: Low

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 this 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 sensitive values in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

Bluetooth MAC: 00:43:A5:A0:78:B8

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 sufficient access control storage solutions. The specific focus of this test is to determine whether the application stores API keys in plaintext, making it easily readable and accessible without any additional processing.

Status: Failed Severity: Analysis
Type: Medium Confidence: 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 sensitive values in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

mail_key:AIXMDAXAAFAXAXXXXXXJaz_key=

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

api_key:3TRNNNFRXXXFNXFNXXFX

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

APIKEY:1888888888XXXXXXXXX

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 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 stores passwords in plaintext, making them easily readable and accessible without any additional processing.

Status: Failed Severity: Analysis
Type: Medium Confidence: Medium

Impact

Storing sensitive password 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 these sensitive password 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 sensitive values in plaintext, especially if it's not required for the application's functionality.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

password:supersecretpassword:password=

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

Password: AugustSpringSummer

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

Password: supersecretpassword

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 or obfuscation to prevent unauthorized access. 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: Failed Severity: Analysis
Type: Low Confidence: Low

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 this sensitive 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.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

Secret WiFi Address (Tap Secret): 3c:e0:2f:ab:ba:d8

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: Failed Severity: Analysis
Type: Low Confidence: 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 this sensitive 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.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

WIFI_IP:10.10.0.2

Local Data Exposure: Device MAC Stored Insecurely

This test evaluates how the application handles the device's MAC 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 MAC value in plaintext, making it easily readable and accessible without any additional processing.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Storing the device's MAC value in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve this sensitive 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.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

MAC=8E:F6:D5:D3:A7:B8

Local Data Exposure: Android ID Stored Insecurely

This test evaluates how the application handles the device's 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 device's Android ID in plaintext, making it easily readable and accessible without any additional processing.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Storing the device's Android ID in plaintext within the local application data directory presents a significant risk. Anyone with device access could retrieve this sensitive 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.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/files/Support_Properties.txt

Android ID: 20ec0f94a8c9f217 - Don't lose this ID, this is a private device

MSTG-STORAGE-2

The app prevents leakage of sensitive data.

There are cases when sensitive data is unintentionally stored or exposed to publicly accessible locations, such as logs, or shared with third-party services through embedded tracking libraries or ad networks. This control covers the risk of unintended leaks where the developer's security intent is bypassed.

Local Data Exposure: WiFi IP Address Logged Insecurely

This test evaluates how the application handles the device's WiFi IP address within the Android logcat logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is requested to be used within the logs the data 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 logs the device's WiFi IP address within logs or plaintext, making it easily readable and accessible without any additional processing.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Logging a device's WiFi IP Address within the Android device logs or plaintext presents a significant risk. Anyone with device access could retrieve the logs, and the 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.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/logcat_logs.txt

WIFI_IP:10.10.0.2

Local Data Exposure: Bluetooth MAC Address Logged Insecurely

This test evaluates how the application handles the device's Bluetooth MAC address within the Android logcat logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is requested to be used within the logs the data 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 logs the device's Bluetooth MAC address within logs or plaintext, making it easily readable and accessible without any additional processing.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Logging a device's Bluetooth MAC Address within the Android device logs or plaintext presents a significant risk. Anyone with device access could retrieve the logs, and the 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.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/logcat_logs.txt

Bluetooth MAC: 8E:D3:E2:B7:B8

Local Data Exposure: Device MAC Logged Insecurely

This test evaluates how the application handles the device's MAC address within the Android logcat logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is requested to be used within the logs the data 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 logs the device's MAC address within logs or plaintext, making it easily readable and accessible without any additional processing.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Logging a device's MAC address within the Android device logs or plaintext presents a significant risk. Anyone with device access could retrieve the logs, and the 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.

Evidence

File: /data/data/com.owasp.mstg.uncrackable1/logcat_logs.txt

MAC=8E:F6:D5:D3:A7:B8

Local Data Exposure: Application Backups Enabled

This test evaluates how the application handles application backups via AndroidManifest. The "AndroidManifest" flag can be used to force the OS to backup application data (including sensitive data) to remote storage accessible by other parties.

Status: Failed Severity: Analysis
Type: Low Confidence: Low

Impact

Enabling "AndroidManifest" grants a malicious risk, as it permits extraction of sensitive application data, circumventing mobile security controls and data confidentiality.

Remediation

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

Evidence

File: AndroidManifest.txt

android:allowBackup:True

Local Data Exposure: Device Serial Logged Insecurely

This test evaluates how the application handles serial data within the Android device logs. Best practices dictate that sensitive information should not be contained within the device logs, additionally, if data is requested to be used within the logs the data 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 logs the device's serial within logs or plaintext, making it easily readable and accessible without any additional processing.

The specific focus of this test is to determine whether the application has stored the device's serial number within logs in plaintext, making it easily readable and accessible without any additional processing.

Status	Severity	Analysis
Pass	Low	

Impact

Storing a device's serial number 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: 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	Analysis
Pass	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	Analysis
Fail	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.

Evidence

File: /data/local/tmp/artifacts/application_logs.txt

```
Secret WifI MAC Address (Top Secret) - 7c:c5:37:a4:ba:b9
```

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	Analysis
Fail	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.

Evidence

File: /data/local/tmp/artifacts/application_logs.txt

```
GPS_LONG (don't disclose location) =-122.0774
```

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	Analysis
Fail	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.

Evidence

File: /data/local/tmp/artifacts/application_logs.txt

```
GPS_LAT (don't disclose location)=37.4143
```

Local Data Exposure: Sensitve 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	Analysis
Pass	Medium	

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	Analysis
Fail	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.

Evidence

File: /data/local/tmp/artifacts/application_logs.txt

```
DNS Address: 1.1.1.1
```