



CredShields

Mobile Application Audit

Mar 4th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Mobile Applications (iOS & Android) audit performed by CredShields Technologies PTE. LTD. on behalf of Juno between Dec 12th, 2022, and Jan 12th, 2023. And multiple retests were performed by Feb 23rd, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

[Juno](#)

Table of Contents

1. Executive Summary	3
State of Security	4
2. Methodology	5
2.1 Preparation phase	5
2.1.1 Scope	6
2.1.2 Documentation	6
2.1.3 Audit Goals	6
2.2 Retesting phase	7
2.3 Vulnerability classification and severity	7
2.4 CredShields staff	10
3. Findings	11
3.1 Findings Overview	11
3.1.1 Vulnerability Summary	11
4. Remediation Status	13
5. Bug Reports	15
Bug ID#1 [Fixed]	15
Clear-Text Traffic Supported	15
Bug ID#2 [Fixed]	17
Unrestricted Google Maps API	17
Bug ID#3 [Fixed]	20
Hardcoded Infura API Keys	20
Bug ID#4 [Fixed]	22
Trusting System Certificates	22
Bug ID#5 [Won't Fix]	24
Application Signed with Weak Algorithms	24
Bug ID#6 [Fixed]	26
Hardcoded Recaptcha Key	26
Bug ID#7 [Won't Fix]	28
Hardcoded FP_JS_PRO Key	28
Bug ID#8 [Fixed]	30
Missing SSL Pinning	30
Bug ID#9 [Won't Fix]	33
Missing Root/Jailbreak Detection	33

Bug ID#10 [Won't Fix]	35
Sensitive Information Disclosed on the Application Memory	35
Bug ID#11 [Won't Fix]	38
Insufficient Debugging Protections	38
Bug ID#12 [Fixed]	41
Background screen caching	41
6. Disclosure	44

1. Executive Summary

Juno engaged CredShields to perform a mobile application audit for their iOS and Android applications from Dec 12th, 2022, to Jan 12th, 2023. During this timeframe, Twelve (12) vulnerabilities were identified. **A retest was performed on Feb 23rd, 2023, and all the bugs have been addressed or acknowledged for won't fix with reasons.**

During the audit, One (1) vulnerability was found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Juno" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	Inf	Σ
Mobile Application (iOS & Android)	0	1	3	8	0	12
	0	1	3	8	0	12

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Mobile Application (iOS & Android)'s scope during the testing window while abiding by the policies set forth by Juno's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Juno's internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Juno can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Juno can future-proof its security posture and protect its assets.

2. Methodology

Juno engaged CredShields to perform a mobile application (iOS & Android) audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

The CredShields team performed both static and dynamic testing of the mobile applications by decompiling the apps and intercepting the APIs originating from the mobile applications.

They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the code.

A testing window from Dec 12th, 2022, to Jan 12th, 2023, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

IN SCOPE ASSETS	
-	https://apps.apple.com/in/app/juno-earn-crypto-rewards/id1525858971
-	https://play.google.com/store/apps/details?id=com.capitalj.onjuno&hl=en_IN&gl=US

Table: List of Files in Scope

2.1.2 Documentation

Documentation was not required as the applications were self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive mobile application security auditing. The majority of the audit is done by manually reviewing the decompiled source code, following [OWASP MSTG](#) standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

Juno is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. Informational

We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do

not pose a direct risk to the organization. Code maintainers should use their own judgment on whether to address them.

2. Low

Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. Medium

Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. High

High-severity vulnerabilities pose a significant risk to the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. Critical

Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds for the users or put sensitive user information at risk of compromise or modification. The client's

reputation and financial stability will be severely impacted if these issues are not addressed immediately.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
 - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and CWE classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, Twelve (12) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	CWE Vulnerability Type
Clear-Text Traffic Supported	Low	Mobile Security Misconfiguration
Unrestricted Google Maps API	Low	Configuration - CWE-16
Hardcoded Infura API Keys	High	Clear Text Storage of Sensitive Information - CWE-312
Trusting System Certificates	Low	Mobile Security Misconfiguration
Application Signed with Weak Algorithms	Low	Mobile Security Misconfiguration

Hardcoded Recaptcha Key	Medium	Cleartext Storage of Sensitive Information - CWE-312
Hardcoded FP_JS_PRO Key	Low	Cleartext Storage of Sensitive Information - CWE-312
Missing SSL Pinning	Medium	Mobile Security Misconfiguration
Missing Root/Jailbreak Detection	Medium	Mobile Security Misconfiguration
Sensitive Information Disclosed on the Application Memory	Low	Mobile Security Misconfiguration
Insufficient Debugging Protections	Low	Mobile Security Misconfiguration
Background screen caching	Low	Mobile Security Misconfiguration

Table: Findings in Mobile Application

4. Remediation Status

Juno is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Feb 23rd, 2023, and all the issues have been addressed or acknowledged for not fixing the bug.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Clear-Text Traffic Supported	Low	Fixed [23/02/2023]
Unrestricted Google Maps API	Low	Fixed [23/02/2023]
Hardcoded Infura API Keys	High	Fixed [23/02/2023]
Trusting System Certificates	Low	Fixed [23/02/2023]
Application Signed with Weak Algorithms	Low	Won't Fix
Hardcoded Recaptcha Key	Medium	Fixed [23/02/2023]
Hardcoded FP_JS_PRO Key	Low	Won't Fix
Missing SSL Pinning	Medium	Fixed [23/02/2023]

Missing Root/Jailbreak Detection	Medium	Won't Fix
Sensitive Information Disclosed on the Application Memory	Low	Won't Fix
Insufficient Debugging Protections	Low	Won't Fix
Background screen caching	Low	Fixed [23/02/2023]

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID#1 [Fixed]

Clear-Text Traffic Supported

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Low

Description

The **android:usesCleartextTraffic** indicates whether the app intends to use cleartext network traffic, such as cleartext HTTP. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". When the attribute is set to "false", platform components (for example, HTTP and FTP stacks, DownloadManager, and MediaPlayer) will refuse the app's requests to use cleartext traffic. This feature is not meant to be enabled in app builds distributed to users.

Vulnerable Endpoint/App

- Android Application

PoC

1. View the Manifest file and observe the "usesCleartextTraffic" flag which is set to true as shown below:


```
tent="@xml/appsflyer_backup_rules" android:usesCleartextTraffic="true" andr
```

Impacts

A network attacker can eavesdrop on transmitted data and also modify it without being detected if the application communicates on clear text protocol such as HTTP.

Remediation

The clear-text traffic can be blocked by changing the following line in the AndroidManifest.xml file to:

android:usesCleartextTraffic="false"

This declares that the app is not supposed to use cleartext network traffic and makes the platform network stacks block cleartext traffic in the app.

Reference:

- <https://android-developers.googleblog.com/2016/04/protecting-against-unintentional.html>
- <https://developer.android.com/guide/topics/manifest/application-element>

Retest

This has been fixed in the network security config.

Bug ID#2 [Fixed]

Unrestricted Google Maps API

Vulnerability Type

Configuration - [CWE-16](#)

Severity

Low

Description

Google Maps API is a paid service. These API keys require security configurations for blocking unauthorized usage by malicious attackers, which is not enabled by default. Developers need to configure these security controls when implementing the API key. While these API keys are designed as public API keys and do not have any impact in terms of customer data confidentiality/integrity, these security configurations still need to be implemented for blocking unauthorized usage, otherwise, attackers can abuse these keys in order to bill the Company for the expenses of using Google Maps API services.

Vulnerable Key

- AlzaSyAfC3TtVQmRZQUbiakxatIEwsQOixyOSgY

PoC

1. The key can be found in the strings.xml file as shown below:

Results	Cost Table/Reference to Exploit:
- Staticmap	\$2 per 1000 requests
- Find Place From Text	\$17 per 1000 elements
- Autocomplete	\$2.83 per 1000 requests
- Autocomplete Per Session	\$17 per 1000 requests
- Place Details	\$17 per 1000 requests
- Nearby Search-Places	\$32 per 1000 requests
- Text Search-Places	\$32 per 1000 requests
- Places Photo	\$7 per 1000 requests

```

1651 <string name="go_back">Go Back</string>
1652 <string name="go_from_cash_to_crypto_in_seconds">Go from cash to crypto in seconds</string>
1653 <string name="go_to_card_tab">Go to Card Tab</string>
1654 <string name="go_to_dashboard">Go to Dashboard</string>
1655 <string name="go_to_jcoin_dashboard">Go to JCOIN Dashboard</string>
1656 <string name="go_to_juno_store">Go to Juno Store</string>
1657 <string name="go_to_profile_details_page">Go to Profile Details Page</string>
1658 <string name="go_to_store">Go to Store</string>
1659 <string name="go_to_transaction_details">Go to Transaction Details</string>
1660 <string name="go_to_txn_page">Go to Transaction Details Page</string>
1661 <string name="google_2fa_desc">Enter the 2-step verification code from your</string>
1662 <string name="google_api_key">AIzaSyBNyurpQxRb03KYB-EAQKGMVQ3Q5NuYm4U</string>
1663 <string name="google_app_id">1:1050397142103:android:488b730bea7addc2e38712</string>
1664 <string name="google_authenticator">Authenticator</string>
1665 <string name="google_authenticator_is_setup">Authenticator is setup</string>
1666 <string name="google_authenticator_is_setup_subtitle">Your 2-step verification settings\n have been
1667 <string name="google_crash_reporting_api_key">AIzaSyBNyurpQxRb03KYB-EAQKGMVQ3Q5NuYm4U</string>
1668 <string name="google_map_static_img_url">https://maps.googleapis.com/maps/api/staticmap?markers=ic

```

Impacts

If this API key is not properly configured, an attacker could consume the company's monthly quota or can over-bill with unauthorized usage of this service and do financial damage to the company, if the company does not have any limitation settings on API budgets.

Remediation

Google suggests adding proper restrictions to the API keys such as -

- HTTP referrers: restricts usage to one or more URLs and is intended for keys that are used in websites and web apps. This type of restriction allows you to set restrictions to a specific domain, page, or set of pages on your website.

- IP addresses: restricts usage to one or more IP addresses, and are intended for securing keys used in server-side requests, such as calls from web servers and cron jobs.
- Android and iOS app restriction: restricts usage to calls from an Android app with a specified package name.

References: https://developers.google.com/maps/api-key-best-practices#restrict_apikey

Retest

The new API key has been validated and proper restrictions are enforced.

Bug ID#3 [Fixed]

Hardcoded Infura API Keys

Vulnerability Type

Clear Text Storage of Sensitive Information - [CWE-312](#)

Severity

High

Description

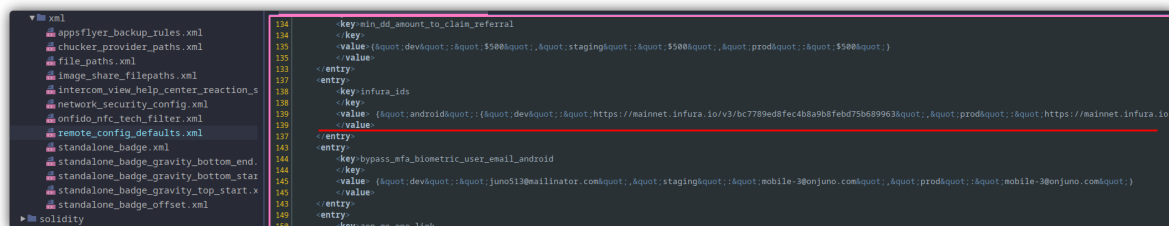
The Android application is storing hard coded plaintext API keys for Infura which can be decompiled and obtained by any user with access to the Android application or the APK file.

Vulnerable Keys

- bc7789ed8fec4b8a9b8febd75b689963
- dfc307ad24d34765827aa60fa0f77e53
- 865248b9a18342d3a9f63de0d81863c1
- 914e59ad52be426e84ac0740ee5d3c99

PoC

1. The keys can be found in the “**res/xml/remote_config_defaults.xml**” file as shown below:



Impacts

Remediation

Do not hardcode sensitive API keys inside the APK as they can be obtained by any external attacker. It is recommended to use Android Keystore API instead.

Retest

Hardcoded Infura API keys have been removed from the APK.

Bug ID#4 [Fixed]

Trusting System Certificates

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Low

Description

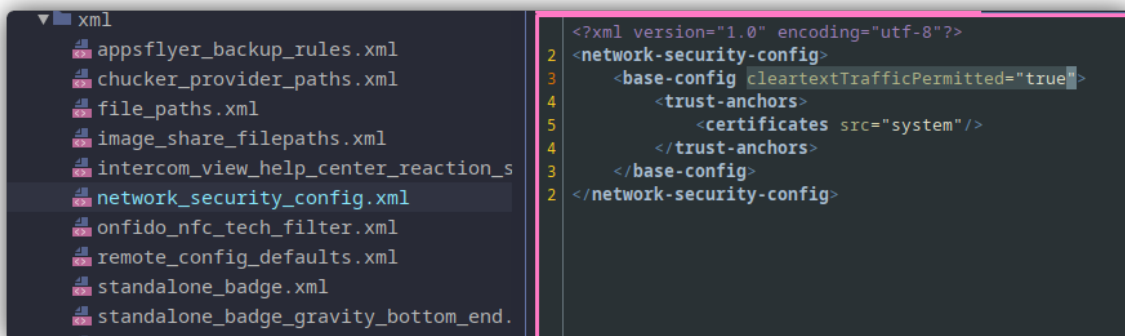
The Android application is configured to trust system certificates in the base config inside the “network_security_config.xml” file. This permits cleartext traffic to the domains trusted by all the CA in the system certificate store. This also includes any malicious certificates installed on a rooted device by an attacker.

Vulnerable File

- res/xml/network_security_config.xml

PoC

1. Go to the file mentioned above and note that the base config is set to trust system certificates and permits cleartext traffic.



```
<?xml version="1.0" encoding="utf-8"?>
2 <network-security-config>
3   <base-config cleartextTrafficPermitted="true">
4     <trust-anchors>
5       <certificates src="system"/>
4     </trust-anchors>
3   </base-config>
2 </network-security-config>
```

Impacts

By permitting clear text traffic and trusting all the CA in the system certificate store, it might be possible to carry out a Man-in-the-Middle attack to intercept and compromise sensitive ongoing traffic since it is possible to install custom certificates in the system store on a rooted device.

Remediation

Do not trust the system certificate store. Only allow cleartext traffic to internal domains or any test servers. It is not recommended to allow cleartext traffic.

Retest

The `cleartextTrafficPermitted` has now been set to false for the base config.

Bug ID#5 [Won't Fix]

Application Signed with Weak Algorithms

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Low

Description

The Android application is signed with weak hashing algorithms such as MD5 and SHA1. These algorithms are known to have collisions and could compromise the integrity of the application.

PoC

1. Execute the following command to view the signing algorithms:
2. `./apksigner verify --print-certs ~/Juno_3.0.8_dev_debug.apk`

```
~/Android/Sdk/build-tools/33.0.1
> ./apksigner verify --print-certs ~/Downloads/Juno_3.0.8_dev_debug.apk
Signer #1 certificate DN: C=US, O=Android, CN=Android Debug
Signer #1 certificate SHA-256 digest: dba02d33fa128d55f4d38925623b897ef4892b86dfb2341d1524def165d89d24
Signer #1 certificate SHA-1 digest: 1c6d9e095d262816a4e0ed3610eb6746a77ce005
Signer #1 certificate MD5 digest: 4ee5fe11aceb3d8bc90908e5739b9c58
```

Impacts

An attacker may be able to create an APK with their malicious code, and identical SHA1 digest of your genuine files. Devices with a previous version will consider the crafted APK to be signed by your certificate, and issue no warning when installing it. However, the likelihood is low because a collision is expensive to find.

Remediation

Use APK Signature scheme v2 which defaults to SHA2 and up, and is thus safe against SHA1 collision.

Retest

This won't be fixed since the minimum SDK version allowed by the team is 26, using the v2 signing scheme by default.

Bug ID#6 [Fixed]

Hardcoded Recaptcha Key

Vulnerability Type

Cleartext Storage of Sensitive Information - [CWE-312](#)

Severity

Medium

Description

The Android application is storing hard-coded plaintext API keys for ReCAPTCHA which can be decompiled and obtained by any user with access to the Android application or the APK file.

Vulnerable Keys

- 6LfvlMQhAAAAAM6RvBpNCEzG5xnW5PRgjZVY6Dsx
- 6LdD-fgZAAAAALl-8rcrUIJGwaWoyH4weG2bKNix

PoC

1. The keys can be found in the **"com/bankconjuno/juno/BuildConfig.java"** file as shown below:

```
28. public static final String PUSHER_CLUSTER = "us3";
29. public static final String PUSHER_INSTANCE_ID = "ce04abac-79c6-4b7d-91f1-145913e07c69";
30. public static final String RE_CAPTCHA_ENTERPRISE_KEY = "6LfvlMQhAAAAAM6RvBpNCEzG5xnW5PRgjZVY6Dsx";
31. public static final String RE_CAPTCHA_SITE_KEY = "6LdD-fgZAAAAALl-8rcrUIJGwaWoyH4weG2bKNix";
32. public static final String SARDINE_CLIENT_ID = "ab745e61-e4cf-45f0-a04c-ac320475bdf4";
33. public static final String SYNAPSE_USER_BASE_URL = "https://uat-dashboard.synapsefi.com/v2/users/";
34. public static final String TYPE_FORM_ID = "Fsz5u2V1";
```

Impacts

An attacker may be able to bypass CAPTCHA validation using these API keys.

Remediation

Do not hardcode sensitive API keys inside the APK as they can be obtained by any external attacker. It is recommended to use Android Keystore API instead.

Retest

This key has been restricted to required packages.

Bug ID#7 [Won't Fix]

Hardcoded FP_JS_PRO Key

Vulnerability Type

Cleartext Storage of Sensitive Information - [CWE-312](#)

Severity

Low

Description

The Android application is storing hard-coded plaintext API key for FP_JS_PRO which can be decompiled and obtained by any user with access to the Android application or the APK file.

Vulnerable Keys

- FP_JS_PRO = "uxykyDPjVH6FVtq2V4V9"

PoC

1. The keys can be found in the "**com/bankonjuno/juno/BuildConfig.java**" file as shown below:

```
20. public static final String FIREBASE_PROJECT_ID = "savvy-bay-264609";  
21. public static final String FLAVOR = "dev";  
22. public static final String FP_JS_PRO = "uxykyDPjVH6FVtq2V4V9";  
23. public static final String GOOGLE_STGNTM_CLIENT_ID = "1050397142103"
```

Impacts

This API key is being used to create a unique ID per device across all platforms. The exploitation depends on how these IDs are being used.

Remediation

Do not hardcode sensitive API keys inside the APK as they can be obtained by any external attacker. It is recommended to use Android Keystore API instead.

Retest

From the client: "As per the current code setup, it won't be possible to access this key in another way. Anyways this is a public key controlled by us and can't cause any huge impact."

Bug ID#8 [Fixed]

Missing SSL Pinning

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Medium

Description

Certificate pinning is the process of associating the backend server with a particular X.509 certificate or public key instead of accepting any certificate signed by a trusted certificate authority. After storing ("pinning") the server certificate or public key, the mobile app will subsequently connect to the known server only. Withdrawing trust from external certificate authorities reduces the attack surface (after all, there are many cases of certificate authorities that have been compromised or tricked into issuing certificates to impostors).

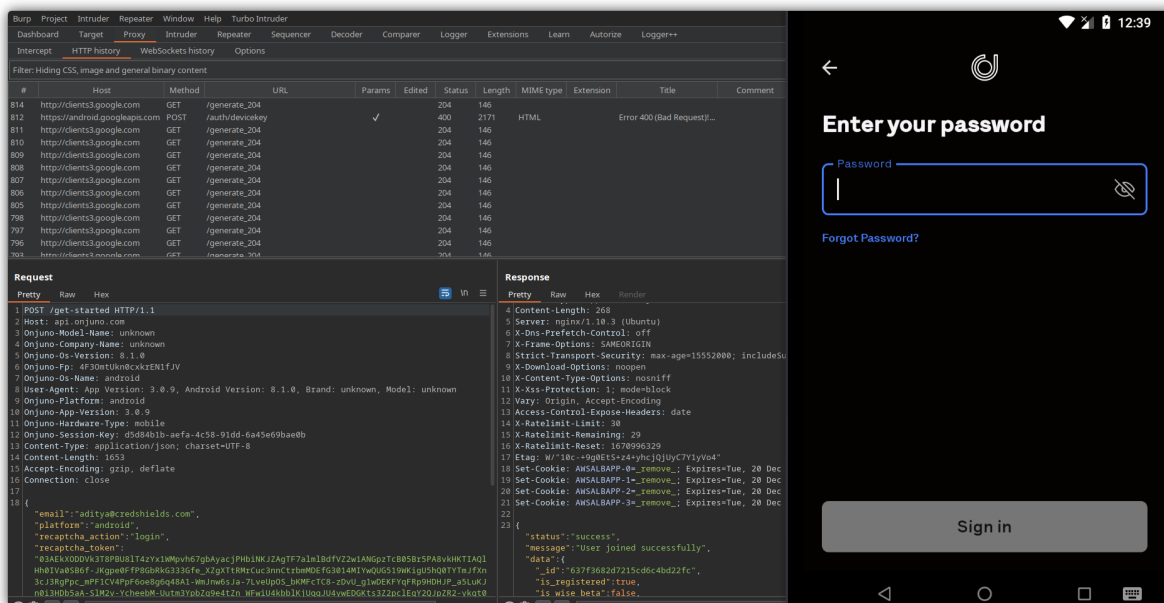
The certificate can be pinned and hardcoded into the app or retrieved at the time the app first connects to the backend. In the latter case, the certificate is associated with ("pinned" to) the host when the host is seen for the first time. This alternative is less secure because attackers intercepting the initial connection can inject their own certificates.

Vulnerable Endpoint/App

- Android

PoC

1. Set up a proxy tool to listen on any port.
2. Install the Burp Suite CA certificate into the mobile with the certificate installer.
3. Follow the WiFi setting and set the host system's IP as a proxy system on the mobile
4. Now you can start navigating through the application, and at the same time you can observe traffic in the burp suite.



Impacts

SSL Pinning is an additional security layer to prevent MITM attacks (Man in the Middle Attack) or sniffing, interception, and tampering of data. To intercept the request, we mostly use a proxy tool. The proxy tool installs its own certificate on the device and the application trust that certificate as a valid certificate and allows the proxy tool to intercept application traffic.

Remediation

- Establish an HTTP Public Key Pinning (HPKP) policy that is communicated to the client application and/or supports HPKP in the client application if applicable.
- Since Android N, the preferred way for implementing pinning is by leveraging Android's Network Security Configuration feature, which lets apps customize their network security settings in a safe, declarative configuration file without modifying app code.
- If devices running a version of Android that is earlier than N need to be supported, a backport of the Network Security Configuration pinning functionality is available via the TrustKit Android library.
- Avoid implementing pinning validation from scratch, as implementation mistakes are extremely likely and usually lead to severe vulnerabilities.

- The Android documentation provides an example of how SSL validation can be customized within the app's code (in order to implement pinning) in the Unknown CA implementation document.
- If the certificate pinning option is not feasible, encrypt all sensitive fields regardless of SSL.

References: [https://www.owasp.org/index.php/Certificate and Public Key Pinning](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)

Retest

This has been remediated.

Bug ID#9 [Won't Fix]

Missing Root/Jailbreak Detection

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Medium

Description

Jailbreak detection mechanisms are added to reverse engineering defense to make running the app on a jailbroken device more difficult. This blocks some of the tools and techniques reverse engineers like to use. Like most other types of defense, jailbreak detection is not very effective by itself, but scattering checks throughout the app's source code can improve the effectiveness of the overall anti-tampering scheme.

In the context of anti-reversing, the goal of root detection is to make running the app on a rooted device a bit more difficult, which in turn blocks some of the tools and techniques reverse engineers like to use. Like most other defenses, root detection is not very effective by itself, but implementing multiple root checks that are scattered throughout the app can improve the effectiveness of the overall anti-tampering scheme.

The Android application does not implement any techniques to detect whether it is being run on a compromised device, such as a rooted device. A privileged process may compromise the security of the device and the integrity of the operating system's controls over data that an application can access.

Restricting the usage of the application to rooted devices will add an extra layer of protection against attackers that are trying to reverse the mobile application and understand the internals.

Vulnerable Endpoint/App

- Android and iOS applications

PoC

1. Install and run the application on a jailbroken or a rooted device and note that the application will work normally.

Impacts

An attacker with a rooted Android can get all the information from the application, bypass the SSL Pinning to see the entire communications in clear text or get the information stored in the device by the application if required. By allowing the application to run on a device that has been rooted, the application risks a heightened exposure to malware, memory dumps, tampering, and other types of malicious activity that could possibly expose the users' credentials or other sensitive data.

Remediation

As a security best practice, it is recommended to implement a mechanism in order to check the rooted status of the mobile device. This can be done either manually by implementing a custom solution or using libraries already built for this purpose.

References:

- <https://owasp.org/www-project-mobile-top-10/2014-risks/m10-lack-of-binary-protections>
- <https://github.com/scottyab/rootbeer>

Retest

Rooted device blocking is not yet implemented but the number is tracked.

Bug ID#10 [Won't Fix]

Sensitive Information Disclosed on the Application Memory

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Low

Description

The user password and all the other PII are still available in memory after submission. This allows for an attacker with physical access to the user's system to access the memory and steal the credentials. The clear text password and other critical data in the memory should be reset after computing the hash on the login function.

Vulnerable Endpoint/App

- Android

PoC

1. Log in to the application with valid credentials
2. Dump process memory (for example using Frida and the tool fridump3 (<https://github.com/rootbsd/fridump3>))
3. `python fridump3.py -s -o juno -u "Juno"`
4. Extract strings from memory dump and search through extracted strings for sensitive data
5. Go to the dump folder then type the following command and replace <password> with your password.
6. `strings strings.txt | grep "<password>" -B 10`

```

MDKxMDE3MSwiZXhwIjoxNjc0TEwNDcxZQ.c5w6gV8LYkp-5MuhhtsLhh6K8A_iBwBbrVTB4UwhNPE
119749 bearer
119750 2022-11-24T09:16:50.384Z
119751 private-user-637f3682d7215cd6c4bd22fc
119752 aditya
119753 credshields.com
119754 -----BEGIN RSA PUBLIC KEY-----
119755 MIICGgKCAgEAl0oWXPq0
119756 IUfPBV/8DdTT3FTMo2Caq0bMwQowTabgrHp1VCYpznB
119757 D8pzH35Y/SAYdvIsk0FlkiZ85ybzrgPpCKxpHu7lhcaMRo2qIikQKw302dS6pC
119758 DfizGbY2oW
119759 SztS0gW/NnR
119760 3D2er0gT0UsZGwMQ64R/ATswshCuTiwGi5
119761 sPn/oAxcxCadva1UNj3LxV5xQAdBBhERmcIUSYN/Mtg/MzKjYICDTiLFptZcbE1R
119762 7a/RhMqP4Iw30uMTg67KadgtU4LMpGP0iI8ICCQx8NC/HerpQRgov0wCAUDX8oVT
119763 kVyQs/Hnrh4jgYwMpnT90HibEBbx0ztbic8mrRkQURmVn9bZwSmK8
119764 /bxvL
119765 YVqUKbYSNFUT0RjvN0rnBmt0fxRBuwScny0tfi7BfhW2EL3mBmyGRSwRtr8Uo63h
119766 gs5n0exxMLMd
119767 CbWauliit4Zm8fbxw9P/7iZEKjRbDhKnY3b0xVAZQaxQJJ3
119768 VxTxFGekSBiaXjhXva41dPyWjQDm9VIGrbpikPJTSlk8wI0L/AnJahi5e/4dump7
119769 8aDeUfgMh0EmkCy/pJLr/Mt5T2/quYS3AqctCghVu0j1lv50i4s8HhYHngTbnz5H
119770 n0kgwRPNNonPpTwnJGfLXU002awBUWI0ru6UZLn/36Ak8o1M9KqjxdKkCAwEAAQ
119771 -----END RSA PUBLIC KEY-----
119772 Chomu
119773 Kumar
119774 2022-12-13T05:30:53.561Z
119775 UK6X2PFC
119776 https://onjuno.onelink.me/TkoI/referral
119777 UK6X2PFC
119778 SEND-AND-RECEIVE
119779 637f4ad5e076852380f63ff3
119780 aditya
119781 credshields.com
119782 Yyp4Brgu#
119783 03AEkX0DBvBbqHYrH7czF5sCR01PRmXQvQjaJ-5d23LvX7kXmNezY299kIdpN9kAbcdc-7Lfnf2768Rf2XaEwcmNrj_-Y
Jje0UtGKZaEsdMmIVVvADTVkoiHZTdxLieik_-61MfC0snE7b1FSeT466nkLQvC0Fm71YYsf7XULZ_2zeu9GY6wS2t9U
gy21ALZijRU6WHFPTDLibePNHiJv5aQHKNiAEVgWQreS2ldSnEza7CAg3SpabQ6GIhRuQBHLd1CniVeLk8a2apDYFKI4o
pRi-sa3K_vRp04_b5W0dhYnb_JtxS-QG7U1n30QNH9r892Q15IfwrIr4_qrPx30i2aaqyNcqQrg04wmhPdb1UsYZuZFp
5n9qAPKb_ZEiNp03dKK00DqCwxNw_FeSFoe50IRunREitopBLasrIbQ-ouz7-xmALmtsJjkFpKRZzNw24TvQGPXEXBRcy
4Cr i5Vn9QNUpm9IYMVZQgfAISZQ78XSn3BIMjdUW9d8G0XvUsYuJZ7RLNGTWABLUXJrb2zAie1ERBiKaAW9NeMqLHLQEG

```

Impacts

This behavior can be used to get access to user account credentials on the stolen phones (or remotely through the chain of exploits) or other sensitive data (such as personal information, database queries, and interaction with APIs) in the same domain.

Remediation

Clear sensitive values such as credentials from the application's memory after they are used/processed by the application on login process.

Also, it's recommended to not store sensitive values (such as passwords) as plain text values. As a mitigation hashing or XOR functions can be used.

References:

- <https://github.com/OWASP/owasp-mastg/blob/master/Document/0x05d-Testing-Data-Storage.md>

Retest

This won't be fixed due to low severity.

Bug ID#11 [Won't Fix]

Insufficient Debugging Protections

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Low

Description

Anti-tamper techniques must be used to prevent attackers from back-dooring legitimate applications. Though none of the solutions are foolproof and a motivated attacker could bypass the protections given a sufficient amount of time, the mitigations do provide a barrier against attackers. Due to the sensitive nature of the application and the data that it accesses, defenses against attacks of this nature greatly increase the security posture of organizations that deploy applications.

It was observed that the Android and ios application does not sufficiently defend themselves against reverse engineering and runtime tampering tools. This allows attackers to attach debuggers and reversing tools with little to no modification of the application, significantly speeding up exploit discovery and development.

The following is needed in order to reproduce this issue:

- A jailbreak iOS device with the Frida server installed and SSH access enabled from a remote machine
- Frida is available here: <https://frida.re>
- A host machine with the Objection tool,
- Objection is available here: <https://github.com/sensepost/objection>

Vulnerable Endpoint/App

- Android and iOS Application

PoC

1. Start the Frida server on the device.

2. Use the Frida-ps to verify that the Frida server is ready. Assuming that the iOS device is connected to the host machine via USB, use the following command on the host machine.
3. Connect to the application using Objection, which will inject the Frida library into the application and allow runtime manipulation. Notice that the application is restarted, but shows no indication that the library injection has occurred.

objection -g pkgname explore

Impacts

This vulnerability is difficult to exploit. However, many off-the-shelf tools exist in order to implement attacks that leverage this vulnerability.

Remediation

- Use anti-debugging techniques. Anti-debugging techniques, such as Android's `Debug.isDebuggerConnected()` available from the `android.os.Debug` class or using `sysctl` to check for the presence of a `ptrace`-based debugger on iOS, will defend the application against debugging, memory manipulation, and reverse engineering. Note that techniques such as using `PT_DENY_ATTACH` will not work as demonstrated on other BSD-based systems, as the `ptrace` syscall itself is not in the public iOS API and will therefore be blocked from release by Apple.
- Perform checks for common reverse engineering tools. Checking for the existence of open D-Bus ports (which are used by Frida), detecting code trampolines, in which the flow of code is diverted into attacker-controlled code and is used commonly by Substrate, and scanning process memory for known artifacts of common runtime manipulation tools would allow the application to detect that runtime manipulation is occurring and take appropriate action.
- Perform application signature checks. Ensure that the application performs a checksum check or some validation mechanism to detect tampering of the application. If the application is tampered with, the detection scheme should take a reactive approach and prevent malicious execution of the application.

Reference:

<https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md>

Retest

This won't be fixed due to low severity.

Bug ID#12 [Fixed]

Background screen caching

Vulnerability Type

[Mobile Security Misconfiguration](#)

Severity

Low

Description

Manufacturers want to provide device users with an aesthetically pleasing experience at application startup and exit, so they introduced the screenshot-saving feature for use when the application is backgrounded. This feature may pose a security risk. Sensitive data may be exposed if the user deliberately screenshots the application while sensitive data is displayed. A malicious application that is running on the device and able to continuously capture the screen may also expose data. Screenshots are written to local storage, from which they may be recovered by a rogue application (if the device is rooted) or someone who has stolen the device.

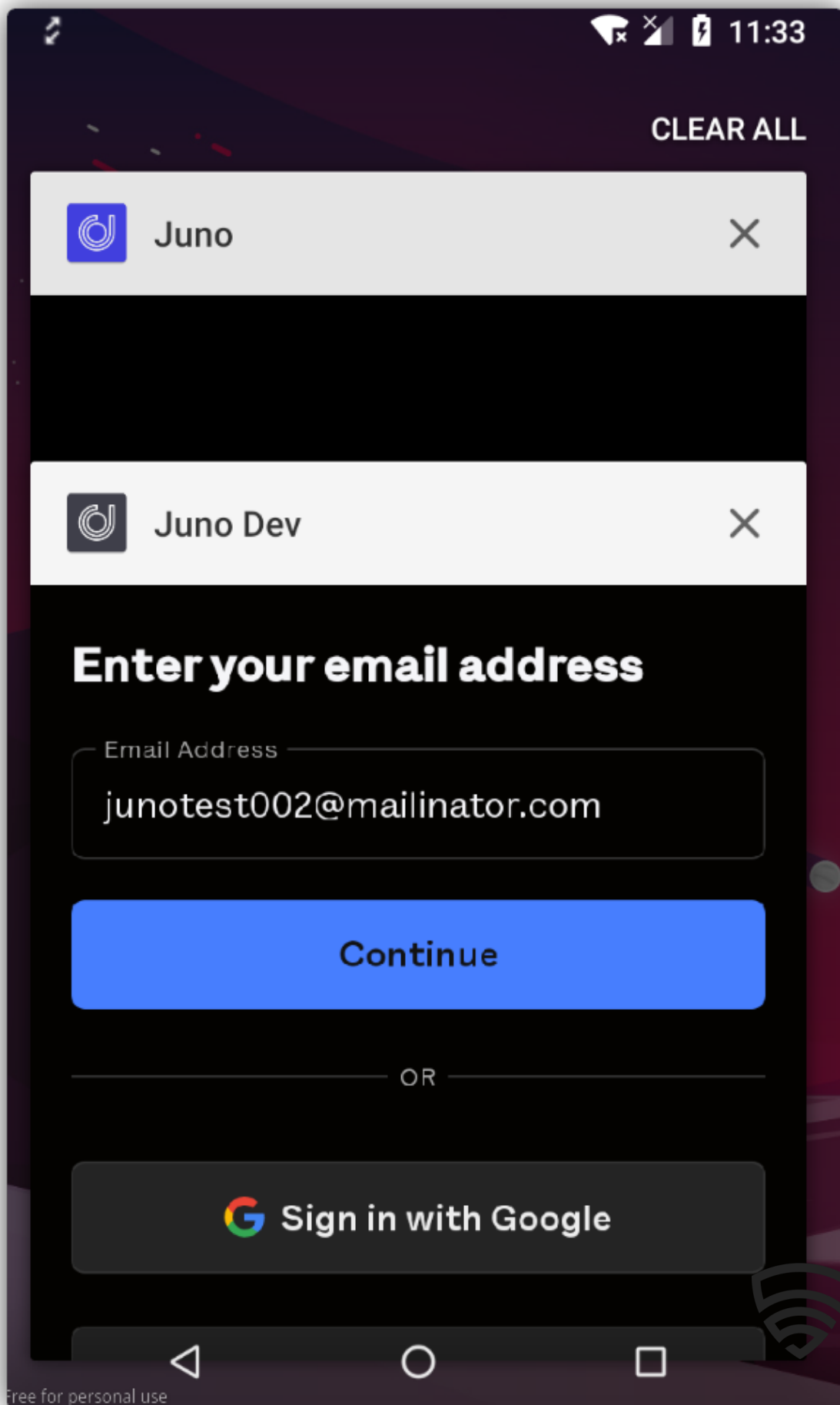
For example, capturing a screenshot of a banking application may reveal information about the user's account, credit, transactions, and so on.

Vulnerable Endpoint/App

- Android Application

PoC

1. Start the Android application and log in.
2. Switch to another application so that the application is sent to the background.
3. Observe that the data inside the application is still visible.



Impacts

This vulnerability represents minimal exposure to exploitation. Only the users of the mobile devices to which the attacker has access are affected by this vulnerability.

The Android application stores snapshots in local storage when the application is backgrounded. The snapshot includes sensitive data such as account balance. An attacker with local access to the device, or one who is able to infect the device with malware, would be able to read this data.

Remediation

As a best practice, consider preventing background screen caching if the application displays sensitive data. You can use FLAG_SECURE for this operation;

Sample Code:

https://developer.android.com/reference/android/view/WindowManager.LayoutParams#FLAG_SECURE

```
public class FlagSecureTestActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
                             WindowManager.LayoutParams.FLAG_SECURE);

        setContentView(R.layout.main);
    }
}
```

Retest

This has been fixed.

6. Disclosure

The Reports provided by CredShields is not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Web3 carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.