Flooz

Flooz Wallet

Mobile Application Security Review

Version: 2.1

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Flooz Wallet Mobile Application, with particular focus on:

- Static analysis of the application's codebase
- Dynamic analysis of the application's execution flow
- Assessment of key functionality, inclusive of cryptographic functions used for key generation, storage of information and network communication
- Backend APIs

The review focused solely on the security aspects of the application's implementation, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the application reviewed. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Flooz Wallet Mobile Application contained within the scope of the security review.

A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Flooz Wallet Mobile Application.

Overview

Flooz Wallet (`wallet.flooz.mobile`) is a cryptocurrency wallet application, written in `Flutter`, a cross-platform development kit by `Google`, for iOS, Android and Web.

The Flooz Wallet allows users to create new wallet addresses, and backup/recover existing wallets based on the provided mnemonic phrases.

The Flooz Wallet also enables users to purchase various digital tokens directly from within the wallet application by utilising external providers such as `Ramp` or `Moonpay`.

On the backend, the application communicates with `flooz-trade-api` API endpoint for blockchain specific functionality and `Google Firebase` ecosystem for other general operations, such as analytics, crash reporting and database communication.

Security Assessment Summary

This review was conducted on provided APK `app-production-release.apk` (*sha512sum:* `e7fe1afe128aad0a0b55a04bbafe72b274d170c0b45b5c12d7e87be56e086c50`) and backend API `https://api.develop.flooz.trade`.

Note: external libraries and dependencies were excluded from the scope of this assessment.

The testing team performed time-boxed assessment based on the following methodology:

- Decompile the mobile application to reveal application's code, configuration and its internal structure to identify vulnerabilities commonly found with mobile application deployment
- Perform static analysis of the application's configuration and codebase to identify any hardcoded sensitive information and to discover vulnerabilities in implementation of the application's business logic
- Identify and reverse engineer key application's functionality, including its cryptographic functions used for key generation, information storage and networking
- Attempt to circumvent application's jailbreak/root detection mechanisms and SSL certifcate pinning
- Perform dynamic analysis of the application behaviour and attempt to alter its runtime behaviour to perform unintended actions or circumvent security restrictions
- Perform analysis of the network traffic generated by the application and conduct API testing of the application's API endpoints

To support this review, the testing team utilised the following Android testing tools:

- apktool: `https://ibotpeaches.github.io/Apktool/`
- jadx: `https://github.com/skylot/jadx`
- Frida: `https://frida.re/`
- Objection: `https://github.com/sensepost/objection`

Retesting activities targeted commit 981a85b.

Findings Summary

The testing team identified a total of 17 issues during this assessment. Categorised by their severity:

- High: 4 issues.
- Medium: 7 issues.
- Low: 4 issues.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Flooz Wallet Android application. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

Each vulnerability is assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
FLZ-01	API Authentication Tokens and Keys Stored in Local Cache	High	Resolved
FLZ-02	User's PII Stored in Local WebView Cache	High	Resolved
FLZ-03	Application Backups Allowed	High	Resolved
FLZ-04	Flutter's SecureSharedStorage Used for Storing Encryption Keys and Mnemonic Phrase	High	Resolved
FLZ-05	API - Lack of Authentication	Medium	Closed
FLZ-06	Backup Phrase in Persistent Clipboard	Medium	Resolved
FLZ-07	Lack of Application Integrity Checks	Medium	Resolved
FLZ-08	Root Detection Bypass	Medium	Resolved
FLZ-09	TLS Certificate Pinning Bypass	Medium	Resolved
FLZ-10	Passcode Persistent in Application Memory	Medium	Resolved
FLZ-11	Unencrypted Local Database	Medium	Resolved
FLZ-12	Cleartext Storage of a Public Key Used for Verification of Unsigned Transactions	Low	Closed
FLZ-13	Application Snapshots when Backgrounding	Low	Resolved
FLZ-14	Screenshots Allowed within the Application	Low	Resolved
FLZ-15	HTML Input Reflected in HTTP Responses	Low	Closed
FLZ-16	Redundant Application Permissions	Informational	Closed
FLZ-17	Miscellaneous - General Comments	Informational	Resolved

FLZ-01	API Authentication Tokens and Keys Stored in Local Cache	
Asset	`wallet.flooz.mobile`	
Status	**Resolved:** Addressed in version `1.0.0`	
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

API keys and authentication tokens to various external providers were found stored unencrypted in the device's local storage:

- `dev_moonpay_api_key`, `prod_moonpay_api_key`, `ramp_host_api_key` and `fpr_log_transport_key` found under `/data/data/wallet.flooz.mobile/files/frc*`
- `accessToken` and `readonlyAccessToken` to `ramp.network` found under `/data/data/wallet.flooz.mobile/app_webview/Default/Local Storage/leveldb/000003.log`
- `AuthToken` to an unknown service (likely `Firebase`) found under `/data/user/0/wallet.flooz.mobile/files/PersistedInstallation*.json`

The identified keys and tokens are often long-lived and could be used to directly authenticate to the services and obtain sensitive or restricted information.

For example, identified cached `accessToken` to `ramp.network` had an expiry set to 1 month in the future and allowed direct API calls to obtain user's first name, last name, address and masked payment card details.

Recommendations

Do not cache or store unencrypted authentication tokens, keys or cookies on the device's local storage.

Periodically clear WebView and local application cache.

Resolution

The development team has mitigated the issue in version `1.0.0` by removing `dev_moonpay_api_key`, `prod_moonpay_api_key`, `ramp_host_api_key` and `ramp.network` keys.

The team has advised that the `AuthToken` is placed on the device by Firebase during installation and is used for performance monitoring. It is not linked to a particular user and in any case, the application forces a refresh whenever the user signs out or switches accounts.

FLZ-02	User's PII Stored in Local WebView Cache		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

User's personally identifiable information (PII) is stored unencrypted in the local WebView cache.

The information is cached after initiating a transaction (e.g. purchasing a token) through `Ramp` provider. The cache files can be found at:

```
/data/data/wallet.flooz.mobile/cache/WebView/Default/HTTP Cache/Cache Data/*
```

and the disclosed user information includes:

- User ID
- First Name
- Last Name
- Address
- Payment Metadata (card type, provider, country code, date and time of transaction)
- Masked Card Number
- Card Expiry Date

This could be considered a breach of privacy and, such information could be used in further, targetted attacks against the application users.

Recommendations

Clear local cache after each WebView session. Ensure no requests containing any sensitive or PII data are cached or stored unencrypted on the device.

Resolution

The development team has mitigated the issue in version `1.0.0` by not using in-app WebView and hence, not creating HTTP Cache. No PII information was found in any other cache files on the device.

FLZ-03	Application Backups Allowed		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The `android:allowBackup` attribute in Flooz Wallet application's `AndroidManifest.xml` is set to `true`.

The `android:allowBackup` attribute defines whether application data, including all of subdirectories and files contained within an app's private directory, can be backed up.

Performing a backup of the data on an Android device can also back-up sensitive information stored within an app's private directory. Applications that handle and store sensitive information such as passwords, encryption keys or personally identifiable information (PII) should have this setting set to `false`.

Recommendations

Set the `android:allowBackup` attribute in `AndroidManifest.xml` to `false`.

Resolution

The development team has mitigated this issue in version `1.0.0` by setting `android.allowBackup` to `false` in `AndroidManifest.xml`.

FLZ-04	Flutter's SecureSharedStorage Used for Storing Encryption Keys and Mnemonic Phrase		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The application uses Flutter's `SecureSharedStorage` plugin to store sensitive information, such as passcode and encryption keys to wallet's mnemonic phrase.

Flutter's `SecureSharedStorage` plugin stores data in `encryptedSharedPreferences`, which are shared preferences that encrypt keys and values. It handles AES encryption to generate a secret key encrypted with RSA and stored in KeyStore.

The `encryptedSharedPreferences` as utilised by Flutter is simply an XML file (`shared_prefs/FlutterSecureStorage.xml`) containing encrypted key-value pairs, saved in the local storage of the device.

It is possible to dynamically hook, at runtime, into a function decrypting information from `encryptedSharedPreferences` and reveal sensitive information in plaintext.

Moreover, the decryption function is called prior to user authenticating to the application using passcode or biometrics, allowing them to extract sensitive information without needing to successfully authenticate.

Sensitive information that can be obtained through this method includes:

- `argon2` hash of the passcode
- encrypted mnemonic phrase
- key and IV used for decryption of the mnemonic phrase

Recommendations

Implement storage of sensitive information using `Android KeyStore Provider` instead.

The Android Keystore system allows storage of cryptographic keys in a container to make it more difficult to extract from the device. Once keys are in the keystore, they can be used for cryptographic operations with the key material remaining non-exportable.

Android Keystore Provider allows an individual app to store its own credentials that only the app itself can access. This provides a way for apps to manage credentials that are usable only by itself. This method requires no user interaction to select the credentials as the OS itself will derive it from lock screen PIN code, password, pattern, and other variables.

Resolution

The development team has mitigated this issue in version `1.0.0` by not using Flutter's `SecureSharedStorage` and implementing custom method of storing sensitive information in encrypted files.

FLZ-05	API - Lack of Authentication		
Asset	`flooz-trade-api`		
Status	**Closed:** Risk accepted by the project team with plans to implement the fix in the future.		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

The API `flooz-trade-api` does not require authentication.

Anyone with knowledge of the endpoints and their definitions can query the API without needing to authenticate.

Whilst the API does not disclose any sensitive information, it may aid attackers in enumerating the system for further attacks.

The API resources could also be consumed and exhausted by external actors.

Recommendations

Implement API Authentication and Authorization as a means to secure access to API endpoint.

For example, the `Autho.Android` SDK can be used to authorize the user of the mobile app and obtain a valid Access Token which can be used to call the API.

Resolution

The development team advised:

"We are not fixing this at this time but are looking into it and will add authentication in the coming months".

FLZ-06	Backup Phrase in Persistent Clipboard		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Wallet backup phrase (mnemonic phrase) can be saved in the device's clipboard.

Mnemonic phrase (or a *'seed phrase'*) is a 12, 18 or 24-word pattern generated each time a new wallet is created. The phrase can be used to derive a private key and, as such, gain access to and full control of the wallet's assets.

Clipboard can be accessed by other applications on the device, which may lead to an unintended leak of the backup phrase outside of the scope of the Flooz Wallet app.

Recommendations

Do not give users an option to store backup phrase in the device's clipboard.

Resolution

The development team has mitigated this issue in version `1.0.0` by removing ability of copying mnemonic phrase into device's clipboard.

FLZ-07	Lack of Application Integrity Checks		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The application does not perform integrity checks on itself.

As such, it is possible to patch parts of the application compromising its integrity, and successfully reinstall and launch it on the device.

Loss of application integrity can result in theft of sensitive data (passwords, credit card numbers, or other sensitive information that may be exploited), change in application's business logic or even used as a vector to compromise the underlying device.

Recommendations

Implement controls ensuring that the application cannot be launched if its code has been modified. This can be achieved by verifying a checksum or signature of the application prior it being run.

Resolution

The development team has mitigated this issue in version `1.0.0` by utilising `Play Integrity API` via `Firebase AppCheck` to check `SafetyNet` Attestation, only allow users to install the app from a trusted source (e.g. Google Play Store), verify SHA-256 key of the application and perform root detection.

FLZ-08	Root Detection Bypass		
Asset	wallet.flooz.mobile		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Although the application implements Root Detection, it can be trivially bypassed by patching the return value of a single function.

The Flooz Wallet application utilises `com.scottyab.rootbeer` library for its root detection, which is very common, well known and easy to bypass.

Rooting an Android device disables some of the crucial built-in security features of its operating system. Attackers can 'root' a device in order to bypass the Android application sandbox. This can allow access to data that is stored on the device which would have otherwise been restricted. Similarly, malware can exploit known weaknesses in Android to gain elevated permissions on a device while running.

Detecting whether the device is 'rooted' is essential to keeping apps safe as it is difficult to guarantee system security and various safeguards otherwise.

Recommendations

As root detection is a client-side control, it can always be bypassed by a determined attacker.

As such, it is recommended to implement additional controls and employ 'defense-in-depth' to improve overall security posture of the application. This may include:

- TLS Certificate Pinning
- Application integrity checks
- Code obfuscation and strings encryption

For more information on TLS pinning recommendations, refer to https://owasp.org/www-project-mobile-top-10/2014-risks/m10-lack-of-binary-protections.

Resolution

The development team has mitigated this issue in version `1.0.0` by utilising `Play Integrity API` via `Firebase AppCheck` to check `SafetyNet` Attestation, only allow users to install the app from a trusted source (e.g. Google Play Store), verify SHA-256 key of the application and perform root detection.

FLZ-09	TLS Certificate Pinning Bypass		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

Although the application implements TLS Certificate Pinning, it can be bypassed using common, widely known techniques.

TLS Certificate Pinning is the process of associating a host with its certificate (or public key) and verifying when establishing a HTTPS connection to make sure the app is communicating with a trusted server.

Majority of the TLS Certificate Pinning implementations can be easily bypassed using widely available tools, such as `Frida` with public gadgets, such as `https://codeshare.frida.re/@akabe1/frida-multiple-unpinning/`; or given enough time and effort, by manually patching the application code.

Recommendations

As TLS Certificate pinning is a client-side control, it can always be bypassed by a determined attacker.

As such, it is recommended to implement additional controls and employ 'defense-in-depth' to improve overall security posture of the application. This may include:

- Root detection
- Application integrity checks
- Code obfuscation and strings encryption

For more information on root detection, refer to `https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning`.

Resolution

The development team has mitigated this issue in version `1.0.0` by achieving 'defense-in-depth' through utilisation of `Play Integrity API` via `Firebase AppCheck` to check `SafetyNet` Attestation, only allow users to install the app from a trusted source (e.g. Google Play Store), verify SHA-256 key of the application and perform root detection.

FLZ-10	Passcode Persistent in Application Memory		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The application stores passcode in its memory, even after an explicit 'Sign Out' command has been issued.

Attackers with access to the application memory, even if the user has explicitly signed out (but left it running in the background), can retrieve the passcode in cleartext.

Recommendations

Remove sensitive information from application's memory as soon as the data is not needed (e.g. after authentication) and certain activity is finished.

This could be achieved by releasing memory on specific Events, overwriting used variables containing sensitive information with `null` and utilising Java's garbage collection to cause the memory to be reclaimed faster.

Resolution

The development team has mitigated this issue in version `1.0.0` by utilising `Play Integrity API` via `Firebase AppCheck` to check `SafetyNet` Attestation, only allow users to install the app from a trusted source (e.g. Google Play Store), verify SHA-256 key of the application and perform root detection.

FLZ-11	Unencrypted Local Database		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Local database `firestore.\%5BDEFAULT\%5D.flooz-profiles-staging.\%28default\%29` containing saved contacts (i.e. friend's wallet addresses) is stored unencrypted.

Although no sensitive information is stored in it, attackers could write to it and modify saved contacts, replacing wallet addresses with their own, completely compromising integrity of data.

Recommendations

Encrypt any local databases containing sensitive information and having high integrity requirements. Ensure their passwords (or encryption keys) are also sufficiently protected.

Resolution

The development team has mitigated this issue in version `1.0.0` by removing offline persistence capability and hence, not storing contact information locally on the device.

FLZ-12	Cleartext Storage of a Public Key Used for Verification of Unsigned Transactions		
Asset	`wallet.flooz.mobile`		
Status	**Closed:** Risk accepted by the project team.		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Public keys used for verification of unsigned transactions are not protected from disclosure and are stored in cleartext under `assets/flutter/flutter_assets/certificates` of the APK.

During a transaction workflow, the Flooz wallet application first calls `flooz-trade-api` API to create an unsigned transaction, which then is verified by the app using hardcoded public key, before being signed and sent on-chain.

While public keys are not considered sensitive on their own, in this context, although highly unlikely, they could be replaced with an attacker-controlled ones to successfully verify, and subsequently make the app sign malicious transactions.

Recommendations

Obfuscate the public key used for verification of untrusted transactions and do not store it in the `pem` format, rather as a `Byte` array in the application source code to increase difficulty of reverse engineering.

Resolution

The development team advised:

"We decided to not do anything about this as the certificate is public and can be fetched from the website anyway."

FLZ-13	Application Snapshots when Backgrounding		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The application snapshot is taken when user presses the 'task switcher' button to put the application in the background.

The application snapshots may contain sensitive information, such as mnemonic phrase, which will be stored locally on the device as an image file.

Malicious actors can then extract this data from the device, compromising the confidentiality of any sensitive information.

Recommendations

Hide the application snapshot when backgrounding. Replace the application snapshot with a static image, such as splash screen or a logo.

Resolution

The development team has mitigated this issue in version `1.0.0` by obfuscating snapshots of sensitive screens.

FLZ-14	Screenshots Allowed within the Application		
Asset	`wallet.flooz.mobile`		
Status	**Resolved:** Addressed in version `1.0.0`		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The Flooz Wallet app allows taking screenshots of the mnemonic phrase page.

Mnemonic phrase (or a 'seed phrase') is a 12, 18 or 24-word pattern generated each time a new wallet is created. The phrase can be used to derive a private key and, as such, gain access to and full control of the wallet's assets.

Taking screenshots allows for a digital copy of the seed phrase to be stored on the device, which significantly increases risk of it being accessed by other applications or malicious actors with remote or local access to the device's storage.

Recommendations

Do not allow screenshots on pages with sensitive information, such as the seed phrase.

Resolution

The development team has mitigated this issue in version `1.0.0` by blocking ability of taking screenshots of the application.

FLZ-15	HTML Input Reflected in HTTP Responses		
Asset	`flooz-trade-api`		
Status	**Closed:** Risk accepted by the project team with plans to implement the fix in the future.		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The following `flooz-trade-api` endpoints reflect request parametrers in their responses without sanitisation, which may lead to reflected Cross-Site Scripting (XSS) attacks:

- `/v1/resolve/domain/`
- `/v1/tokens/`
- `/v1/referral/`

Note, the list above should not be considered exhaustive.

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. If an attacker can control a script that is executed in the victim's browser, then they can typically fully compromise that user.

Note, due to `content-type` of the responses being `application\json`, this issue is not directly exploitable. However, care should be taken when using the content of server responses in any other applications or systems, particularly where it may be rendered as HTML.

Recommendations

Do not reflect user-supplied input in HTML responses without sanitisation. Sanitise and/or escape HTML and JavaScript input before returning it to users.

Resolution

The development team advised:

"We are not fixing this at this time but are looking into it and will fix in the coming months"

FLZ-16	Redundant Application Permissions
Asset	`wallet.flooz.mobile`
Status	**Closed:** Risk accepted by the project team with plans to implement the fix in the future.
Rating	Informational

Description

The application requests permissions not needed for its regular operation:

```
<uses-permission android:name="android.permission.FLASHLIGHT"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

It is best practice to limit an application's permissions to minimum required for it to function properly.

Recommendations

Remove unneccessary permissions from `AndroidManifest.xml` file.

Resolution

The development team has advised:

"We are currently working on revoking those permissions."

FLZ-17	Miscellaneous - General Comments
Asset	`wallet.flooz.mobile`
Status	**Resolved:** Addressed in version `1.0.0`
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Cleartext traffic permitted in configuration file `res/xml/network_security_config.xml`**

   Cleartext traffic opt-out protects apps from accidental usage of unencrypted network traffic. The setting `<base-config cleartextTrafficPermitted="true">` has not been disabled.

   Note: this configuration file is not referenced in `AndroidManifest.xml` and as such, does not appear to be in use - consider removal.

2. **"Received" transactions appear as "Sent"**

   In the UI, under "Activities" tab, incoming transactions are labelled as 'Sent'.

   Ensure each type of transaction is labelled correctly to avoid confusion.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team has acknowledged and mitigated these issues in version `1.0.0`.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.