Audit Report

FreshCut Wallet

v1.0

Nov 3, 2022

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by FreshCut Interactive, Inc. to perform a security audit of the FreshCut Diamond Polygon wallet, integrated into the FreshCut Flutter app (for iOS and Android).

The objectives of the audit are as follows:

1. Determine the correct functioning of the system, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract integrations bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/freshcutgg/mobile/

Commit hash: `9726981128bc8ce0c56ecdd23bc864fcef8554cf`

The wallet is part of a larger mobile application with various features and libraries. These parts of the application were not audited. Because of the shared codebase, a vulnerability in a different part of the application could lead to a compromise of the wallet.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

FreshCut is a platform where creators can publish gaming clips. The Polygon ERC20 token FreshCut Diamonds (FCD) is used to reward the FreshCut community for various activities. The mobile application contains a wallet where the tokens (FCD and MATIC) can be managed.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	app.dart: Logging of sensitive information possible	Critical	Resolved
2	No limit for unsuccessful PIN entries	Major	Resolved
3	Disconnecting the wallet does not require the PIN	Major	Resolved
4	security_manager.dart: Initialization vector reuse in AES CBC mode	Major	Resolved
5	wallet_address.dart: Insecure mnemonic generation	Minor	Resolved
6	wallet_transaction_bloc.dart: Unnecessary gas estimation for FCD transfers	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Low-Medium	-
Code readability and clarity	High	-
Level of documentation	Low	-
Test coverage	Low	-

Detailed Findings

1. `app.dart`: Logging of sensitive information possible

Severity: Critical

In `initCrashlytics`, the following error handler is configured to record flutter errors in Firebase:

```
FlutterError.onError = FirebaseCrashlytics.instance.recordFlutterError;
```

This configuration applies to the whole app and therefore also the wallet part. There, it is possible that sensitive information (mnemonic, private key) are included in a stack trace and sent to firebase. When an account that has access to these information is breached, an attacker can steal all coins that belong to these wallets, which happened in the past.

Recommendation

Use a more fine-grained logging configuration. It should never be possible that sensitive information such as private keys or mnemonics are sent to an external server.

Status: Resolved

2. No limit for unsuccessful PIN entries

Severity: Major

The wallet is protected by a 6-digit PIN, meaning there are 1,000,000 possible combinations (and roughly 3,000 of them are invalid because of the complexity requirements). Therefore, it will take on average 498,500 tries to find the correct PIN, meaning a brute force attack is feasible. The wallet currently does not enforce an upper limit on the number of tries and also does not implement a measure like an exponential backoff to protect users against such attacks.

Recommendation

Restrict the maximum number of tries or implement an exponential backoff after every unsuccessful entry.

Status: Resolved

3. Disconnecting the wallet does not require the PIN

Severity: Major

When a user wants to disconnect his current wallet, he is not asked for the PIN. While this may be desirable from a UX perspective, disconnecting a wallet is a sensitive operation with potential financial effects. For instance, when the user did not backup the wallet previously, he will lose access to the coins in the wallet.

Recommendation

Ask for the PIN when disconnecting the wallet such that attackers with physical access to the phone cannot cause a financial loss for users.

Status: Resolved

4. `security_manager.dart`: Initialization vector reuse in AES CBC mode

Severity: Major

In `onPinCodeChanged`, the old key is decrypted with the old PIN and the resulting plain text then encrypted again with the new PIN. Because `SecretKeyEncryptor._getInitVector` returns the current initialization vector (if one exists), the same IV is used for both operations, which should not be done in AES CBC mode.

Recommendation

Generate a new IV when changing the PIN code.

Status: Resolved

5. `wallet_address.dart`: Insecure mnemonic generation

Severity: Minor

To generate a mnemonic, the function `generateMnemonic` of the package `bip39` is used. This library does not generate the mnemonic in a cryptographically secure way because of an off-by-one error: It calls _randomBytes within generateMnemonic. The `_randomBytes` function calls `nextInt(_SIZE_BYTE)` on the `Random.secure()` PRNG to get a random byte. This function generates random integers from 0, inclusive, to the provided maximum, exclusive. Because `_SIZE_BYTE` is set to 255, there is therefore no way that `_randomBytes` returns the value 255, meaning non-uniform randomness is used to generate the mnemonic.

Recommendation

Use a different library for generating the mnemonic.

Status: Resolved

6. `wallet_transaction_bloc.dart`: Unnecessary gas estimation for FCD transfers

Severity: Informational

In `_transferFromConnectedToExternal`, the gas limit for FCD transfers is estimated. This value is then passed to `fcdWallet.transfer`, which further passes it to `_contractService.send`. However, because `ContractService.send` has a hard-coded limit of 62,000 for FCD transfers, the whole estimation is unnecessary and never used.

Recommendation

Either do not hard-code the limit in `ContractService.send` or use the hard-coded limit and do not perform the estimation.

Status: Resolved