Star Key

Star Key

Mobile Wallet Audit

April 26, 2024

Iwaki Hiroto

Table of Contents

Overview

This document describes the result of auditing the StarKey Mobile Wallet project.

Star-Key is an advanced multi-chain mobile wallet designed primarily for decentralized finance (DeFi) enthusiasts and experienced cryptocurrency users.

The primary objective behind developing Star-Key is to cater to individuals seeking a versatile and feature-rich self-custodial wallet compatible with multiple popular EVM chains, offering enhanced security, a seamless user experience, and cross-platform compatibility.

An exhaustive examination and inspection of Star-Key's front-end, back-end, smart contracts, infrastructure, and deployment pipelines formed the basis of this independent audit.

Meticulously scrutinizing each aspect allowed for identification of potential vulnerabilities, discrepancies, optimization prospects, and enhancement possibilities.

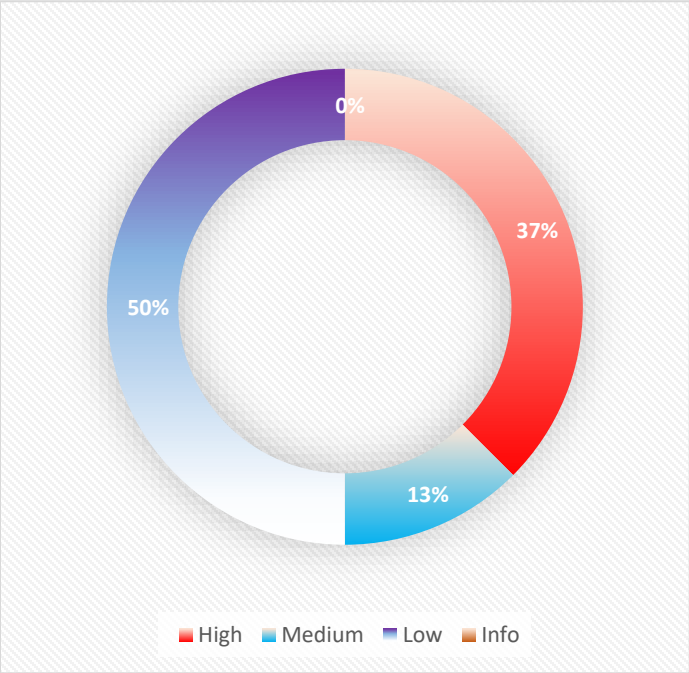
Audit Checklist

Audit Class	Audit Subclass	Check
Source code security	Code decompilation detection	✓
Permission security	App permissions detection	✓
Storage security	File storage security audit	✓
	App cache security audit	✓
Communication security	Communication encryption security audit	✓
Business security	Business logic security audit	✓
Application interaction security	Deeplinks security audit	✓
Authentication security	Client-Based Authentication Security audit	✓
Transfer security	Signature security audit	✓
	Deposit/Transfer security audit	✓
	Transaction broadcast security audit	✓
Secret key security	Secret key generation security audit	✓
	Secret key storage security audit	✓
	Secret key usage security audit	✓
	Secret key backup security audit	✓
	Secret key destruction security audit	✓
	Insecure entropy source audit	✓
	Cryptography security audit	✓
User interaction security	Password complexity requirements	✓
	Anti-phishing	✓

Project Details

Project Name	Star Key
Project Type	Mobile Wallet
Audit Period	April 12 ~ April 26

Finding Classification



High	3
Medium	1
Low	0
Info	4

Finding Details

High

1. Insufficient Input Validation

Mobile-wallet: reedSolomonErasure.js#L33-L124

Description

The encode and reconstruct methods in the ReedSolomonErasure class do not perform adequate input validation, leading to potential unexpected behavior and instability. Specifically, the methods rely entirely on the caller passing accurate values for dataShards, parityShards, shardsAvailable, and shards. Without proper validation, malicious actors or buggy integrations might pass incorrect values resulting in undefined behavior, crashes, or even exploits.

Impact

Insufficient input validation poses risks for downstream consumers, increasing the likelihood of integration issues, instability, and security vulnerabilities.

Recommended Mitigation Steps

Introduce strict input validation checks for the encode and reconstruct methods, including:

Verifying that dataShards and parityShards are positive integers.

Asserting that shardsAvailable is either null or an array of Boolean values equal to the total sum of dataShards plus parityShards.

Confirming that shards is an array of unsigned 8-bit integer typed arrays (Uint8Array) with a size greater than zero and divisible by dataShards plus parityShards.

Throw insightful exceptions or errors when validation fails to signal to the caller that the input is invalid.

Document the expected input formats clearly in the class definition and method signature comments, easing integration efforts.

Implementing input validation strengthens the ReedSolomonErasure class defensively, reducing risk factors for consumers and promoting seamless integration.

2. Insecure randomness

Mobile-wallet: services/EthersService.ts#L97-L118

Description

The createWalletUsingSeed function employs a deterministic algorithm for generating derived addresses, rendering them predictable.

Impact

Malicious actors can exploit predictable derived addresses, putting user funds at risk.

Recommendation

Leverage cryptographic randomness sources to ensure secure and unpredictable derived addresses. Utilize tried-and-tested libraries like secp256k1 or bs58check to produce secure derived addresses.

3. Address the insecure storage

Mobile-wallet: SCR_Accounts/index.tsx#L32-L232

Description

Plain text storage of seed phrases and private keys in local storage represents an alarming security threat. Attackers gaining unauthorized access to devices could easily obtain these vital credentials, resulting in catastrophic consequences.

Impact

Failure to implement strong security protocols for sensitive data leaves the entirety of a mobile wallet application vulnerable to breaches, exposing users' digital assets to theft and fraudulent activities. Given the nature of this oversight, adversaries can potentially gain unauthorized access to users' sensitive information, including seed phrases and private keys, which in turn puts their finances and investments at considerable risk.

When seed phrases and private keys are compromised, cybercriminals can initiate transactions on behalf of unsuspecting victims, draining accounts of their contents instantaneously. Moreover, possession of a victim's seed phrase permits the generation of fresh addresses linked to their original holdings, effectively circumventing standard security precautions. Resultingly, affected users stand to lose substantial sums, and face challenges recovering lost assets.

Additionally, the violation of customer trust stemming from such a lapse can inflict irreversible damage on an organization's brand and business model. Repairing damaged reputations demands immense resources, and often involves litigation costs, regulatory penalties, and settlement fees. Ultimately, neglecting suitable security measures invites substantial monetary losses, legal complications, and lasting negative ramifications for businesses and consumers alike. Therefore, investing in robust security protocols is

paramount to protect sensitive data and preserve trustworthy relationships within the mobile wallet ecosystem.

Recommended Solution

Mitigate the risk by employing robust security protocols for managing sensitive data. Specifically, adopt the following approach:

Hash and Salt passwords before storing them.

Encrypt sensitive information like seed phrases and private keys before committing them to storage.

Restrict access to secured vaults holding these valuable assets.

Adhering to these guidelines fortifies the application against malicious intrusions, bolstering the confidence of users who entrust their financial dealings to the mobile wallet application.

Medium

1. Potential stale data retention

Mobile-wallet: FileManagerService.ts#L8-L124

Description

Functions responsible for removing files or directories, i.e., deleteFile and deleteDirectory, do not return resolved promises. Downstream processing continues assuming the removal succeeds, resulting in potential stale data retention.

Impact

If the functions responsible for removing files or directories fail to return resolved promises, subsequent operations assume successful completion regardless of the outcome. This shortcoming creates a situation wherein residual, obsolete, or irrelevant data persist, leading to data inconsistencies.

As a consequence, these lingering fragments can manifest in various manners, producing outcomes that range from mild annoyances to severe impairments. Examples comprise of corrupted application states, unusable features, reduced performance, or faulty calculations. Even worse, such irregularities might engender widespread distrust, dissatisfied customers, and increased technical debt. Furthermore, persistent anomalous data hinder progression towards achieving objectives, forcing teams to spend extra cycles isolating root causes and executing remediation plans.

Returning resolved promises post-deletion operations alleviate these predicaments by pausing downstream processing, waiting for successful termination, and subsequently continuing once guaranteed eradication transpires. Thus, rest assured that clean slates emerge, paving the way for accurate, predictable, and steadfast outcomes that foster productivity and satisfaction.

Recommendation

Modify the deleteFile and deleteDirectory functions to incorporate resolved promises signaling successful completion of deletion routines before moving onto subsequent stages. Doing so instills discipline and precision, yielding benefits that span accuracy, cohesion, and sustainability.

Info

1. Potentially missing data integrity validation

Mobile-wallet: addressBook/index.ts#L4-L58

Description

Currently, the recentTransactionAddressList object stores recent transactions for each user. When storing a new recent transaction address, the code checks if the address already exists within the past 3 items for the respective network short name.

However, it lacks validation to check if the userId, networkShortName, and address altogether already exist within the nested structure, thus possibly duplicating records. Duplicate records could negatively affect the intended functionality of tracking recent transaction addresses.

Impact

Neglecting to implement a validation mechanism in the storeRecentTransactionAddress reducer can result in duplicate entries accumulating in the recentTransactionAddressList object.

Space Efficiency: Space consumed expands proportionally to the volume of redundant entries, consuming unnecessary storage space and increasing computational overhead.

Time Complexity: Search queries demand longer durations to sift through bulky datasets, prolonging response latencies and hampering user experience.

Correctness and Predictability: Without a validation mechanism, the application cannot accurately reflect the latest transaction history. Consequently, the intended functionality of tracking recent transaction addresses deteriorates, introducing ambiguity and skewed analytics.

Therefore, establishing a robust validation mechanism becomes imperative for conserving resources, expediting search query responses, and sustaining correctness and predictability in the application's behavior.

Recommendation

Implement a validation mechanism

Modify the storeRecentTransactionAddress reducer to iterate over the existing user data and networks, searching for an identical combination of userId, networkShortName, and address. Prevent addition if the combination already exists.

2. Incomplete Edge Case Handling

Description

The editUserData function handles editing user data in the usersData and importedUsersData arrays. Nevertheless, it doesn't handle the condition where the specified user isn't present in either array.

Impact

Ignoring this edge case may result in silent failures and leave the user data unchanged, leaving developers guessing whether the modification occurred successfully. This opacity hinders effective debugging and problem isolation.

Recommendation

Ensure the function's robustness by introducing a guard clause to check if the user exists in both arrays before attempting to modify their data. If the user isn't found, emit a warning or throw an error to signal the absence of the targeted record.

3. Security

Mobile-wallet: EncryptionService.ts#L11-L57

Description

Utilizing a single master password for encryption and decryption operations can raise security concerns. Choosing weak passwords might put user accounts at risk, making them susceptible to attacks

Impact

Attackers can attempt to crack weak master passwords using brute force techniques or precomputed tables, eventually gaining unauthorized access to sensitive data and tampering with user accounts.

Recommendation

Improving security measures is essential to mitigate these concerns. Consider implementing the following recommendations:

Multi-factor Authentication: Combine multiple factors, such as knowledge (password), possession (security token), and inherence (biometrics), to create robust barriers against unauthorized entry.

Password Hashing Algorithms: Derive encryption keys using advanced algorithms such as PBKDF2, scrypt, or Argon2. These algorithms resist attacks by demanding significant computational power, slowing down hackers trying to crack the master password.

By incorporating multi-factor authentication or derived keys using algorithms like PBKDF2, scrypt, or Argon2, you can considerably strengthen security and protect user accounts from unauthorized access.

4. Error Handling

Some parts of the code do not effectively handle errors thrown during execution.

Mobile-wallet: services/Erc20TokenService.ts#L65-L169

Description

The fetchAllErc20TokenBalanceByNetwork function in services/Erc20TokenService.ts does not completely handle errors, and axios configuration is created repetitively inside the loops.

Impact

Errors occurring in the function may result in an incomplete list of token balances, confusing users or breaking dependent functionality. Repeated axios configuration creations lead to performance loss.

Recommendation

Surround the for loop block inside a try-catch statement to properly handle errors and ensure a complete list of token balances.
Move the axios configuration creation outside of the loops to avoid repetition, improving performance.