verichains

SECURITY AUDIT OF

SPOT ON CHAIN MOBILE WALLET

SPOTONCHAIN

Public Report

Sep 20, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Aptos Blockchain	Aptos is an innovative public blockchain (proof of stake) developed by former Facebook employees, with a primary focus on delivering high throughput and robust security for smart contracts developed using the Move programming language.
Sui Blockchain	Sui is an innovative, decentralized Layer 1 blockchain that redefines asset ownership.
Solana Blockchain	Solana is a blockchain built for mass adoption. It's a high performance network that is utilized for a range of use cases, including finance, NFTs, payments, and gaming. Solana operates as a single global state machine, and is open, interoperable and decentralized.
Near Blockchain	NEAR is a decentralized application platform with the potential to change how systems are designed, how applications are built and how the web itself works.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Sep 20, 2023. We would like to thank Spot On Chain for trusting Verichains Lab in auditing the mobile wallet. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Spot On Chain Mobile Wallet. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Spot On Chain Mobile Wallet

Spot On Chain - Simplify blockchain data analytics and provides actionable insights for traders of all levels to make smart decisions with confidence.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Spot On Chain Mobile Wallet.

It was conducted on commit `9add8c27b3f7d5add2681241a41fb0d887c8b7a9` from git repository *https://github.com/Ambros-Technology/soc_mobile_audit*.

Audit points:

- Wallet creation management logic
- Passcode logic
- Wallet persists logic
- Import seed phrase and private key
- RPC Call

The latest patch version of the Spot On Chain Mobile Wallet is `69c41f95428c1774b9e02ea0df337156e562ba4c` from git repository

1.3. Audit methodology

Our security audit process includes four steps:

- Mechanism Design is reviewed to look for any potential problems.
- Source codes are scanned/tested for commonly known and more specific vulnerabilities using public and our in-house security analysis tool.
- Manual audit of the codes for security issues. The source code is manually analyzed to look for any potential problems.
- Set up a testing environment to debug/analyze found issues and verifies our attack PoCs.

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the functioning; creates a critical risk to the application; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the application with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the application with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure application. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Spot On Chain Mobile Wallet was written in `Typescript` Programming Language using `React Native Framework`.

2.1.1. Wallet creation management logic

The Wallet Factory is located in `src/core/factory/wallet-factory.ts`. It is responsible for creating a wallet and storing it in the Redux store. This wallet factory provides functionality for managing and creating wallets for various blockchains, handling the validation of linked accounts, and checking the presence of a specific private key in any of the wallets.

The wallet supports a list of blockchains (EVM, Aptos, Near, Sui, Solana) and each blockchain has its own wallet creation logic. Users can create a wallet by providing a mnemonic to get a private key from it or by generating a new mnemonic.

The factory validates linked accounts associated with the wallets created by the mnemonic. It fetches linked accounts using an API call and returns the linked accounts if they are available. And the factory has a method called `hasPrivateKey` to check if a given private key matches any of the private keys generated from the mnemonic. It iterates through different chain types and checks the private keys for a match.

2.1.2. Keyring controller

Keyring is the core concept of the secret storage and account management system. The `AmbrosKeyringController` class is the implementation of the keyring. The class has three main responsibilities:

- Initializing and grouping the accounts for different blockchains
- Keep track of local nicknames for those individual accounts.
- Encrypting and persisting the accounts to disk

The `AmbrosKeyringController` uses the `ObservableStore` of a `metamask/obs-store` to define `this.store` and `this.memStore`. The `this.store` is used to store the encrypted accounts in persistence data storage.

An `encryptor` variable that has inspiration from `MetaMask/metamask-mobile` responsibility for encryption and decryption. The `encryptor` has generated a key by using `AES pbkdf2` and `AES Forked pbkdf2` of `react-native`.

2.1.3. Passcode logic

The major logic of the passcode is implemented in the `keyring-controller.ts` file. The keyring controller is responsible for managing the passcode and wallet. It has two important functions: `verifyOwnershipThenExecuteCallback` and `checkIsUnlockedThenExecuteCallback`.

The functions `verifyOwnershipThenExecuteCallback` and `checkIsUnlockedThenExecuteCallback` handle the authentication and verification processes. It checks if the user has set a passcode. If not, it creates a default passcode for them. If the user has set a passcode, it checks if the passcode is correct. If the passcode is correct, it executes the callback function. If the passcode is incorrect, it shows an error message.

The wallet is defined, and a user must have a passcode to use it. However, the first time before the user sets their passcode, the wallet uses the default passcode (hardcore in the code) to unlock the wallet. The default passcode can be leaked by any attacker by reverse engineering the app.

On the other hand, if the user sets their passcode, the passcode will be stored in a `SecureKeychain` class. The `SecureKeychain` class is a wrapper of the `react-native-keychain` library. The `SecureKeychain` class is used to store the passcode in the key chain. The key chain is a secure storage solution for sensitive data. The key chain is encrypted and protected by the user's passcode. The key chain is only accessible when the user enters the correct passcode. The key chain also supports face ID, touch ID, and biometrics.

The `keyring-controller.ts` file also has a function `isUnlocked` to check if the wallet is unlocked. The wallet will be locked every 5 minutes. The `isUnlocked` function checks if the wallet is unlocked. If the wallet is unlocked, it returns true. If the wallet is locked, it returns false. The `isUnlocked` function is used to check if the wallet is unlocked before executing any action that requires the wallet to be unlocked.

2.1.4. Wallet persists logic and Import seed phrase and private key

The function `persistAllAccount` is responsible for ensuring that account data is securely stored using encryption and that it can be retrieved and decrypted when needed. The proper implementation and security of this function are essential for safeguarding sensitive user data.

The function serializes a user's accounts with a private key, then encrypts the serialized data using the user's password. The encrypted data is stored in `this.store` (the app storage).

2.2. Findings

During the audit process, the audit team found some vulnerability in the given version of Spot On Chain Mobile Wallet but we have a best practice recommendation for the Spot On Chain Mobile Wallet team.

#	Issue	Severity	Status
1	Failed to attempt locked time is bypassable	CRITICAL	Fixed
2	Do not lock the user despite the user's efforts to enter the incorrect passcode.	LOW	Fixed
3	Allow users to leave the default passcode alone.	LOW	Fixed
4	Certain dependencies are insecure.	INFORMATIVE	Fixed

2.2.1. Failed to attempt locked time is bypassable - CRITICAL

Effected files:

- src/core/controllers/keyring-controller.ts

A `setUnlocked()` and `unlockAttemptFailed()` function used the `Datetime` API of the `Luxon` library to calculate the counter of the lock. The `Datetime` function gets time from the device. The attacker can change the time of the device to bypass the locked time and continue to try another passcode. This approach does not require rooting and can be automated.

```
import { DateTime } from 'luxon';

public setUnlocked() {
  const lastVerificationTime = DateTime.now().toUnixInteger();
  this.updateMemStoreState({
    disableCounter: 0,
    numOfFailedAttempt: 0,
    isUnlocked: true,
    enableTime: 0,
    lastVerificationTime,
  });
  this.emit('unlock');
}

public unlockAttemptFailed() {
  this.updateMemStoreState({
    numOfFailedAttempt: this.memState.numOfFailedAttempt + 1,
    disableCounter:
      this.memState.numOfFailedAttempt % UNLOCK_ATTEMPT_LIMIT === 0
        ? this.memState.disableCounter + 1
        : this.memState.disableCounter,
    enableTime: DateTime.now()
      .plus({ minutes: 5 * (this.memState.disableCounter || 1) })
      .toSeconds(),
  });
}
```

RECOMMENDATION

We recommend implementing secure datetime measurement using a native module combination of time synchronization from a trusted source and local real lock APIs like `System.Clock.elapsedRealtime` and `SystemClock.elapsedRealtimeNanos`. These return the elapsed time since the system was booted, including time when the device goes into deep sleep. This lock is guaranteed to be monotonic and continues to tick even when the CPU is in power saving mode, so it is the recommended basis for general purpose interval timing.

UPDATES

The Spot On Chain Mobile Wallet team has fixed and updated the application by using an API to get timestamps from the server. The application will check the timestamp from the server to calculate the counter of the lock. The attacker cannot change the timestamp on the server to bypass the locked time.

However, the timestamp is centralized and depends on the server. If the server is a problem, the user cannot use the application. The application should use a decentralized timestamp to calculate the counter of the lock. The problem is rare, but it can happen.

2.2.2. Do not lock the user despite the user's efforts to enter the incorrect passcode - LOW

Effected files

- src/core/controllers/keyring-controller.ts

Following the document of the Spot On Chain Mobile Wallet, the wallet will be locked every 5 minutes. The `isUnlocked` function checks if the wallet is unlocked. If the wallet is unlocked, it returns true. If the wallet is locked, it returns false. The `isUnlocked` function is used to check if the wallet is unlocked before executing any action that requires the wallet to be unlocked.

However, the isUnlocked function does not check if the user has entered the wrong passcode several times. The user can enter the wrong passcode several times, and the wallet does not delay this malicious action by the user. The vulnerability can be exploited by any attacker to brute-force the user's passcode.

```
public isUnlocked() {
    const isUnlocked = this.memState.isUnlocked;
    const lastVerificationTime = this.memState.lastVerificationTime;
    const now = DateTime.now().toUnixInteger();
    const diff = now - lastVerificationTime;
    return isUnlocked && diff < VERIFICATION_TIMEOUT;
}
```

RECOMMENDATION

The `isUnlocked` function should check if the user has entered the wrong passcode several times. If the user has entered the wrong passcode several times, the wallet should be delayed for a period of time.

UPDATES

The Spot On Chain Mobile Wallet team has fixed and updated the application to lock the wallet for 5 minutes if the user has entered the wrong passcode several times.

2.2.3. Allow users to leave the default passcode alone - LOW

Following the document of the Spot On Chain Mobile Wallet, users are not required to set their own passcode. The app creates a default passcode for them. The default passcode is easily leaked by any attacker by reverse engineering the app. The application persists the credentials on the device under encrypted data. The default passcode is used to encrypt the credentials. The default passcode is hardcoded in the code. The attacker can reverse engineer the app to get the default passcode and decrypt the credentials.

RECOMMENDATION

Although the application sets a default passcode for the user, the user should be required to change the default passcode when they first open the app. The user should be required to set their own passcode.

UPDATES

The Spot On Chain Mobile Wallet team has fixed and updated the application to require the user to set their own passcode.

2.2.4. Certain dependencies are insecure. - INFORMATIVE

Some dependencies used in the application are old and affected by vulnerabilities. Dependencies are listed in the table below.

#	Issue	Library	Current version	Patched version	The latest version	Reference
1	Prototype Pollution	@react-native-firebase/app	^15.7.0	17.4.3	18.3.0	*https://security.snyk.io/vuln/SNYK-JS-XML2JS-5414874*
2	Regular Expression Denial of Service (ReDoS)	semver	^4.3.2	5.7.2	7.5.4	*https://security.snyk.io/vuln/SNYK-JS-SEMVER-3247795*
3	Open Redirect	web3	1.6.1	1.7.5	4.1.0	*https://security.snyk.io/vuln/SNYK-JS-GOT-2932019*

RECOMMENDATION

We recommend updating the dependencies to the patched version to avoid the vulnerabilities. The best solution is to update the dependencies to the latest version.

UPDATES

The Spot On Chain Mobile Wallet team has updated the dependencies to the patched version.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Aug 21, 2023*	Public Report	Verichains Lab
1.1	*Sep 15, 2023*	Public Report	Verichains Lab
1.2	*Sep 20, 2023*	Public Report	Verichains Lab

Table 2. Report versions history