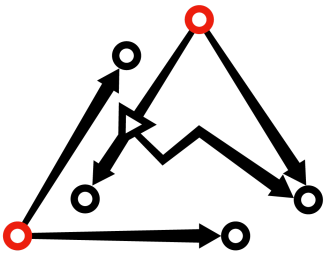

Least Authority

PRIVACY MATTERS

Avalanche Mobile Wallet (2nd Review)

Security Audit Report

Ava Labs

Final Audit Report: 30 November 2023

Table of Contents

Overview

Background

Ava Labs has requested that Least Authority perform a second security audit of their Avalanche Mobile Wallet.

Project Dates

- **July 20, 2023 - August 24, 2023:** Initial Code Review *(Completed)*
- **August 28, 2023:** Delivery of Initial Audit Report *(Completed)*
- **November 28, 2023:** Verification Review *(Completed)*
- **November 30, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Ann-Christine Kycler, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Avalanche Mobile Wallet followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Avalanche Mobile Wallet:
  https://github.com/ava-labs/avalanche-wallet-apps

Specifically, we examined the Git revision for our initial review:

- `0b1bd1c4c6d5ca90042f5a21eacf995a7054e766`

For the verification, we examined the Git revision:

- `91c9490827b863f17f4c967eb8967b05bb8aa71d`

For the review, this repository was cloned for use during the audit and for reference in this report:

- Avalanche Mobile Wallet:
  https://github.com/LeastAuthority/Avalance-Mobile-Wallet

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Ava Labs:
  https://ava-labs.atlassian.net/wiki/spaces/EN/pages/2039840784/Mobile+Environment+React+Native+Setup

In addition, this audit report references the following documents:

- M. Bellare and C. Namprempre, “Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm.” *Springer Link,* 2000, [BN00]
- RFC 9106 - Argon2 Standards Document: https://datatracker.ietf.org/doc/html/rfc9106#name-parameter-choice
- Behavior Changers: all Apps | Android Developers https://developer.android.com/about/versions/12/behavior-changes-all#clipboard-access-notifications
- Android 9 Compatibility Definition: https://source.android.com/docs/compatibility/9/android-9-cdd

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation and adherence to best practices;
- Exposure of any critical information during user interactions with the blockchain and external libraries, including authentication mechanisms;
- Adversarial actions and other attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Vulnerabilities in the code and whether the interaction between the related and network components is secure;
- Proper management of encryption and storage of private keys, including the key derivation process;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Core Mobile Wallet is a mobile wallet application that provides functionality to send and receive assets and collectibles from the Avalanche, Ethereum, and Bitcoin networks. It also allows connecting to dApps via the WalletConnect Protocol.

Our team performed a first audit of the Core Wallet in the time period September - October 2022, and delivered the Final Audit Report on January 30, 2023. In this review, in addition to conducting a full audit of the current implementation, our team also checked if the security issues and suggestions that were identified in the previous review have been addressed in the current implementation.

System Design

Our team found that the Core Wallet developer team approached the design of the application with security in mind. This is evident in that our findings are specific and do not concern the overall design of the application.

In the first report, our team had mentioned that the minimum supported API level had been too low for sufficient security guarantees, and this has been remediated by supporting the minimum API level 28 (Android 9). Our team had also noted that in some instances, the application overview screen of the operating system may leak sensitive information (Issue C in our previous report), which the Core Wallet team mitigated since our team reported it. Additionally, we found that the copying of secrets, such as mnemonics, to the clipboard may result in other malicious applications being able to access them. This

Issue remains unresolved and, thus, our team is reporting it again (Issue A). Similarly, our team found that a lack of user consent prior to communication to a dApp via the WalletConnect Protocol remains an Issue (Issue E). The Issues that have not been addressed from our previous review are characterized by requiring a user experience trade-off. However, we recommend always prioritizing security and explicitly informing users of risks when this is not feasible.

When the user switches back to the application, the Core Wallet implements an access control mechanism to the application by requiring a 6-digit pin or biometrics before access to the wallet is granted. However, our team found that wallets can, in some instances, be deleted without the need for prior authentication via the pin (Issue D).

Storage

Locally stored secrets are handled by the native keystore libraries and, as an additional security layer, the Core Wallet encrypts the mnemonics with a key derived from a 6-digit pin using `Argon2`. Our team found several issues related to secret storage and storage usage.

With the current minimally supported API level 28, for some versions of Android, different security assumptions could apply for the application. The implementation of a hardware-backed keystore is not required by all manufacturer Android versions operating on that API level (Issue B), and additional checks for those features need to be implemented.

The parameters used for `Argon2` deviate from the standard recommendations, and we recommend the usage of recommended parameters (Suggestion 1).

Most user data other than the secrets (which are handled by native key storage) are stored encrypted in a database managed via a third-party library. However, some data that could leak information about user behavior is stored unencrypted. We recommend including this information in the encrypted data to mitigate potential data leaks of the application (Suggestion 4).

dApp Connection

After our team completed the first audit, the Core Wallet switched to version 2 of the WalletConnect Protocol. Version 1 of the protocol has since been sunset and our team recommends not supporting this version and removing code handling the old protocol version from the codebase (Suggestion 2). Previously reported Issues, such as displaying misleading information in a session approval screen (Issue F in our previous report) and not properly parsing version numbers (Suggestion 1 in our previous report) have since been remediated.

Code Quality

The structure of the code resembles the codebases of other applications built using React Native, which makes it easy to navigate. Names of files, types, functions, and variables are descriptive, which further helps readability. Additionally, react components were used idiomatically.

Tests

The Ava Labs team has implemented sufficient tests, which showed high coverage of important logic, especially around WalletConnect v2.

Documentation and Code Comments

Our team found that even though there was not a large amount of documentation or code comments, the complexity of the code does not require extensive comments. Additionally, the project documentation provided a generally sufficient overview of the system and its intended behavior, and the code comments were helpful.

However, we did identify a few areas where more documentation would be helpful. For example, some dependencies are patched, but for some of the patches, it is not clearly explained why the patch is needed. Our team also found some areas of improvement in how users are informed of the security of their choices regarding the usage of the Core Wallet (Issue A, Suggestion 3).

Scope

The scope of this review was sufficient and included all security-critical components.

Dependencies

Our team identified that the dependency `redux-persist-transform-encrypt` does not utilize authenticated encryption. We recommend either forking or patching the dependency to use an authenticated encryption algorithm (Issue C).

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Mnemonics Copied to Clipboard	Partially Resolved
Issue B: Unchecked Security Level Endangers Key Material	Unresolved
Issue C: The Module redux-persist-transform-encrypt Does Not Use Authenticated Encryption	Resolved
Issue D: Wallet Can Be Deleted Without Authentication	Unresolved
Issue E: Mobile Wallet Does Not Enforce User Consent To Establish Websocket Connection	Unresolved
Suggestion 1: Adjust Argon2 Parameters	Resolved
Suggestion 2: Remove Deprecated WalletConnect Version Support	Resolved
Suggestion 3: Display Informative Warning for Jailbroken/Rooted Devices	Resolved
Suggestion 4: Encrypt ViewOnceInformation	Resolved

Issue A: Mnemonics Copied to Clipboard

Location

`app/components/MnemonicScreen.tsx#L69`

`app/utils/DeviceTools.ts`

Synopsis

The mobile wallet currently has a feature, which allows the user to copy the mnemonics to clipboard. On many devices, the clipboard can be accessed by all the applications within the system.

This Issue has already been reported in the previous Audit Report (Issue E) our team delivered on January 30, 2023 and has not been resolved. It is still our belief that the copying of the mnemonics to the clipboard is a security risk. Hence, we report this Issue again in this report.

Impact

With access to the clipboard, a malicious application on the system could store the mnemonics and send the data through the network to an attacker-controlled endpoint in order to gain access to the user's funds.

Preconditions

This Issue is possible if:

- On Android 9: The attacker controls a malicious application on the user's device, and the user uses the copy-to-clipboard functionality; or
- On other Devices: The user uses the copy-to-clipboard functionality and opens a malicious application controlled by the attacker.

Feasibility

If the preconditions are met, the attack is simple. The attacker has to detect mnemonics (for example, using regular expressions) or simply exfiltrate all clipboard contents.

Technical Details

On Android, the `ClipboardManager` can be utilized, and a listener can be used to listen for clipboard changes. Starting at Android 10, only currently active applications and the configured keyboard can access the clipboard at any time. From Android 12 (API level 31) on, the user is notified if an application accesses the clipboard, but applications can process this information nonetheless. The Core Wallet application supports Android 9 and higher, so Android 9 devices are still susceptible.

Mitigation

We suggest displaying a warning when the copy button is clicked that advises the user to only open a trusted application for storing the mnemonic and then immediately return to the Core Wallet application and press the enter button. This event can then be used to put a non-sensitive string in the clipboard (empty or fixed string).

Remediation

We recommend removing the copy-to-clipboard feature and encouraging users to physically write down the mnemonics instead.

Status

The Ava Labs team has added functionality to display a warning before a user can copy the mnemonics to clipboard. This warning informs users of the risks of copying the mnemonics.

Verification

Partially Resolved.

Issue B: Unchecked Security Level Endangers Key Material

Location

`app/contexts/EncryptedStoreProvider.tsx`

Synopsis

The Core Wallet application requires a minimum API level of 28 (Android 9). Hence, while using the `react-native-keychain` wrapper for the Keystore functionality, the usage of a hardware-backed keystore is not guaranteed. Android 9 only requires a trusted execution environment (TEE) when a fingerprint sensor is present. Therefore, the availability of secure, trusted hardware cannot be assumed for all the devices the Core application is used on.

Impact

If an attacker is able to extract keys from a software-backed keystore, they could extract stored secrets from devices.

Feasibility

This kind of attack is likely if keys can be extracted from a software-backed keystore, which can occur if there is a vulnerability in the operating system and a functioning exploit for it.
Another attack vector might emerge if the device storage is dumped using physical access to the device. This requires more dedicated hardware and specialized skills, but might be possible even when the operating system itself remains uncompromised.

Remediation

We recommend that the functionality provided by the `react-native-keychain` be implemented to check the available security level. This method can be used to determine whether a hardware-backed keystore is available. If this is not the case, we recommend that a password (as opposed to a 6-digit pin) be required for the application and used to derive a key with `Argon2id`, which can then be used to encrypt inputs to the keystore functionality.
Additionally, we recommend implementing password throttling, increasing the wait time between each failed password/pin input attempt. This mitigates the limited entropy a 6-digit pin provides.

Status

Since the percentage of users affected by this is small, the Ava Labs team has decided not to implement this remediation. However, our team believes that because device percentages can be subject to change, the security level should be checked, and users should be informed if security assumptions of the Core Wallet application do not hold on their device.

Verification

Unresolved.

Issue C: The Module redux-persist-transform-encrypt Does Not Use Authenticated Encryption

Location

`package.json`

Synopsis

Core uses the module `redux-persist-transform-encrypt` for encrypting the data stored in the Redux store. However, that module only encrypts the data with AES-CBC and does not provide any other

means of authenticating the ciphertexts. Therefore, an attacker can tamper with the ciphertext without being detected.

Impact

An attacker can modify the application state.

Preconditions

The attacker would need read and write access to the ciphertext, which is stored in the private data folder of the application. This requires circumventing or breaking the security model of the operating system.

Feasibility

If the preconditions are met, the attack can be executed reliably and does not require special tooling.

Technical Details

The Cipher Block Chaining Mode (CBC mode) of operation is malleable, which means that given a ciphertext, it is possible to generate a second ciphertext that decrypts to a meaningful message when decrypted with the same key.

For this reason, usually an encryption scheme that provides authenticated encryption (AE) or authenticated encryption with associated data (AEAD) is used. These include `AES-GCM`, `NaCl secretbox` (i.e. `ChaCha20-Poly1305`), or the `Encrypt-then-MAC` pattern.

Since rooting an Android device or jailbreaking an Apple device inherently circumvents or breaks the security model of the operating system, these devices are likely at a higher risk of being vulnerable.

Mitigation

We suggest warning users in the jailbreak screen that authenticated data may be modified without detection.

Remediation

We recommend forking or patching the `redux-persist-transform-encrypt` module to use an authenticated encryption scheme.

Status

The Ava Labs team stated that they are exploring the possibility of implementing `AES-GCM` in the future. They are currently using an encrypt-then-mac scheme that uses a secret key string as input to `AES-CBC`, and correctly derives a separate key as input to `HMAC`.

Verification

Resolved.

Issue D: Wallet Can Be Deleted Without Authentication

Location

`screens/drawerNoWallet/NoWalletDrawerView.tsx#L63`

`development/app/AppHook.ts#L51`

Synopsis

The wallet data can be deleted without entering the wallet pin/password.

Impact

Attackers with physical access to the device can access the application and delete the wallet, which could result in users being locked out and unable to retrieve their funds. Attackers can also achieve this goal by deleting the application via the system settings – although this is not possible with managed devices.

Preconditions

Attackers would need physical access to the device, and the `walletstate` should be empty.

Feasibility

If the preconditions are met, the attack is trivial.

Technical Details

When the wallet state is empty but a wallet exists, the "Enter Wallet" button is shown on the Watchlist screen. In this state, the sidebar navigation offers the option to delete the wallet without authentication.

Remediation

We recommend requiring the wallet to ask for user authentication before user data is deleted to ensure that wallet data is only deleted if proper user authentication is provided.

Status

The Ava Labs team stated that in their security model, an attacker with physical access to the unlocked device is not considered. Although we find this to be reasonable, our team noted that such assumptions have to be communicated to the user in order for them to be able to consider it in their own threat model.

Verification

Unresolved.

Issue E: Mobile Wallet Does Not Enforce User Consent To Establish Websocket Connection

Location

`services/walletconnectv2/WalletConnectService.ts`

Synopsis

The application does not require consent for the initial opening of a websocket connection to the pairing wallet but will receive the session request information through the websocket connection and afterwards present a customized screen (including, for example, a custom image for the dApp, the network, and a selection method for the accounts). This allows malicious applications on the same device or malicious websites to start a websocket communication channel without the user noticing.

Our team has already identified and reported this Issue in our previous Audit Report (Issue A), which we delivered on January 30, 2022, and it has since not been resolved.

Impact

Given that there is a vulnerability in the code handling the RPC requests over the websocket or the websocket libraries used, an attacker may be able to use those to, for example, gain access to private or secret information, or render the Core Wallet application unusable.

Preconditions

An attacker would either need to control a malicious application on the same device or be able to have the user open a website controlled by the attacker that can then send a deeplink to the application.

Feasibility

This attack would require another vulnerability in how RPC requests are handled or how websocket connections are handled by the underlying operating system. Therefore, an attack at this point is unlikely, yet still possible.

Remediation

Before the initial session request is performed via the `WalletConnect` library, we recommend showing a generalized screen whenever a dApp connection is attempted. Hereby, the user would be informed that a dApp connection attempt is about to occur and should attempt to retrieve the first set of information. While this is not ideal in terms of user experience, it mitigates the risk of potential vulnerabilities being exploited without the user noticing.

Status

The Ava Labs team has acknowledged the finding but decided against implementing the recommended remediation in favor of a better user experience..

Verification

Unresolved.

Suggestions

Suggestion 1: Adjust Argon2 Parameters

Location

`app/utils/EncryptionHelper.ts#L19-L28`

Synopsis

The `Argon2` standards RFC 9106 document specifies two recommended options for parameter selection. However, the parameters used for deriving an encryption key from the PIN are not consistent with either of them.

More specifically, the core uses 32 MiB space, 2 iterations, and a parallelism of 6, while the RFC recommends using 64MiB space, 3 iterations, and a parallelism of 4.

Mitigation

We recommend implementing the parameters listed in the RFC 9106 document referenced above.

Status

The Ava Labs team has implemented the parameters listed in the RFC 9106 document, as recommended.

Verification

Resolved.

Suggestion 2: Remove Deprecated WalletConnect Version Support

Location

app/contexts/DeeplinkContext/DeeplinkContext.tsx#L81

app/store/walletConnect

Synopsis

Version 1 of the WalletConnect Protocol has been sunset since the 28th of June, 2023. During our code review, our team noticed that code for supporting WalletConnect version 1 is still part of the codebase. Since the old version is already sunset, the old protocol version should not be supported anymore.

Mitigation

We suggest that only WalletConnect version 2 be supported by the Core wallet. We also recommend removing unused code from the codebase because, otherwise, this code may introduce or retain security vulnerabilities.

Status

The Ava Labs team has removed support for WalletConnect version 1.

Verification

Resolved.

Suggestion 3: Display Informative Warning for Jailbroken/Rooted Devices

Location

app/screens/onboarding/JailbrokenWarning.tsx

Synopsis

Currently, only the string “jailbroken” is shown. Users should be warned about the risks of using a jailbroken/rooted device with a cryptocurrency wallet.

Mitigation

We recommend that a warning be displayed that informs users of the risks of using a jailbroken or rooted device and that recommends against using such devices with the Core Wallet. For example, the warning could note that keys and mnemonics are more vulnerable to exfiltration from other applications on rooted/jailbroken devices.

Status

The Ava Labs team has mitigated this suggestion by displaying an informative warning to the user.

Verification

Resolved.

Suggestion 4: Encrypt ViewOnceInformation

Location

app/Repo.ts

Synopsis

`ViewOnceInformation` is stored in `AsyncStorage`. Comments suggest that a `walletId` may be added at a later version of the wallet. In this case, encrypting the data would be preferred.

Mitigation

Any information that could be used to derive user behavior or gain user private information should be encrypted utilizing authenticated encryption when stored within the application-specific storage, such as `AsyncStorage`.

Status

The Ava Labs team has moved `ViewOnceInformation` to the encrypted redux store.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.