Least Authority
PRIVACY MATTERS

Xverse Wallet

Security Audit Report

Secret Key Labs

Final Audit Report: 27 August 2021

Table of Contents

Overview

Background

Secret Key Labs has requested that Least Authority perform a security audit of their Xverse Wallet, a mobile wallet application for the Stacks blockchain.

Project Dates

- **June 30 - July 19**: Code review *(Completed)*
- **July 23**: Delivery of Initial Audit Report *(Completed)*
- **August 23-25**: Verification Review *(Completed)*
- **August 27**: Final Audit Report delivered *(Completed)*

Review Team

- Gabrielle Hibbert, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer
- Sajith Sasidharan, Security Researcher and Engineer

Coverage

Target Code and Revision

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

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

- Xverse Wallet: https://github.com/secretkeylabs/xverse
- Beta version install: https://www.secretkeylabs.com/install

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

`0c361886daf162086a0cae31fa9a581838c19d6b`

For the verification, we examined the Git revision:

`0e643788a70c3a271c6ee59fcd976514f1e02144`

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

https://github.com/LeastAuthority/xverse

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:

- `Xverse Audit Documentation.paper` (*provided by Secret Key Labs to Least Authority via Slack 12 July 2021*)

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, including authentication mechanisms;
- Adversarial actions and other attacks that impact funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Vulnerabilities in the code, as well as secure interaction between the related and network components;
- Proper management of encryption and storage of private keys, including the key derivation process;
- Inappropriate permissions and excess authority;
- Data privacy, data leakage, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Xverse Wallet developed by Secret Key Labs is a mobile wallet application that allows Android and iOS users to generate and store a key pair for sending and receiving Stacks and Bitcoin. Users are also able to use the wallet to participate in Stacking pools or to Stack individually. In addition, the Xverse Wallet allows users to view NFT assets. The Xverse Wallet uses the BIP-39 library for key pair generation, which is widely considered to be sufficiently secure. However, our team did not review the BIP-39 library, as this external dependency was considered out of scope for this security review.

System Design

It is clear that security has been strongly considered by the Secret Key Labs team in the system design of the Xverse Wallet. The system is built with well-maintained React components. Furthermore, the Xverse Wallet stores private keys in adherence with best security practices. However, in investigating if stored keys are susceptible to extraction, our team found that when the keystore is cleared, the unencrypted seed phrase is not cleared from the device. As a result, we recommend deleting all fields in the keystore that may have been set by the application in the `clearKeystore` function (Issue A).

We examined the implementation of PIN authentication and did not identify any implementation errors in that component. However, we recommend that instead of the seed phrase being stored unencrypted in the device keystore, that the seed phrase be encrypted with a PIN or password for storage. Additionally, we suggest allowing users to set longer PINs or passwords, which increases the security of the PIN or password (Issue C).

Our team noted that the security of the Xverse Wallet is completely dependent on the security of the mobile device's operating system (OS). We suggest that this be disclosed in the documentation to the users so that they may be informed in making security considerations (Suggestion 3).

We examined the implementation of the Xverse Wallet for iOS and Android, and found that Android and some iOS versions allow the user to take screenshots from applications that can inspect the screen at all times, including instances of the mnemonic passphrase being displayed by the Xverse Wallet. We recommend that the Secret Key Labs team take measures to prevent or mitigate the impact of screenshots (Issue B).

Finally, our team identified logging messages being generated in the application. If compromised, these logs could leak private user data. As a result, we suggest disabling logging in the production build of the Xverse Wallet (Suggestion 6).

Code Quality

The Xverse Wallet codebase is well written and follows React best practices. The code is organized by modular React components that are reused in different screens, thus making the codebase more readable and concise. However, there are screens in the application that cannot be navigated to, and components that are not used or only used in these unreachable screens. As a result, we suggest that unused code be removed from the repository to increase readability of the code and to reduce potential confusion (Suggestion 2).

Tests

The Xverse Wallet in-scope repository does not implement a test suite and contains no tests. We strongly suggest implementing a test suite to aid in identifying implementation errors and potential security vulnerabilities (Suggestion 1).

Documentation

The Secret Key Labs team provided us with audit documentation that was helpful in explaining the basic functionality of the system. We suggest improving the documentation to include all components of the system, including the biometric authentication component (Suggestion 3).

Code Comments

There are very few code comments present in the codebase. The documentation contained within the code should be comprehensive and document every function and entrypoint, explaining the intended functionality of each of the components to aid both reviewers and contributors in understanding the system's intended behavior (Suggestion 4).

Scope

Our team found that the scope of this security review was generally sufficient. However, the commit provided by the Secret Key Labs team did not include the biometric authentication component, which is intended to be implemented in the Xverse Wallet. Since this is a critical security feature, we recommend a follow up security review of the biometric authentication component (Suggestion 5).

We noted that the Xverse Wallet relies on the `react-native-keychain` library for accessing keystores. This is a security critical dependency and, as a result, we recommend that it undergo an independent security audit (Suggestion 5).

Dependencies

Finally, as a supplement to our review, we ran an `npm-audit` test on the repository in-scope, which reported a large number of vulnerable dependencies, most of them classified as "high". We recommend including `npm-audit` in Continuous Integration (CI) runs and that vulnerable dependencies be promptly updated to patched versions. The use of outdated, unmaintained, and unaudited dependencies increase the attack surface of any system. Use of audited and well maintained dependencies reduces the potential for security exploits (Suggestion 7).

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: Clearing Keystore Does Not Delete Seed	Resolved
Issue B: Wallet Screen Not Protected Against Screen Recording	Resolved
Issue C: Complete Reliance on OS Security Framework	Resolved
Suggestion 1: Implement a Test Suite	Unresolved
Suggestion 2: Remove Unused and Unreachable Code	Resolved
Suggestion 3: Improve Documentation	Unresolved
Suggestion 4: Increase Code Comments	Partially Resolved
Suggestion 5: Conduct Additional Security Reviews	Unresolved
Suggestion 6: Disable Logging in Production Version of the Wallet	Partially Resolved
Suggestion 7: Update and Maintain Dependencies	Partially Resolved

Issue A: Clearing Keystore Does Not Delete Seed

Location

`utils/walletKeychainHelper.ts#L146-L148`

Synopsis

The mobile device keystore can be cleared when the wallet is wiped from the settings menu or when a wrong PIN is entered three times during authentication. Wiping of the device keystore removes the PIN from memory, however, the seed phrase is not cleared from the keystore. If the keystore of the mobile device used is not secure, the seed phrase is vulnerable to extraction.

In the case that the user intends to wipe the wallet from the settings menu or as a result of attempted unauthorized access, the user will likely expect and assume that the seed phrase is unextractable from the wiped wallet, which is not the case.

Impact

Compromise of the seed phrase could lead to loss of all funds controlled by the wallet.

Preconditions

The attacker must have physical access to the device and has compromised the mobile devices keystore mechanism.

Feasibility
The mechanisms for secure storage of keys of both Android and iOS devices have been compromised in the past. A recent research paper pointed out several realistic scenarios for extracting keys. Nonetheless, a skilled and well-resourced attacker is required to successfully carry out this attack.

Technical Details
The `clearKeystore` function only removes the PIN from the keystore, but leaves the seed stored.

Remediation
We recommend deleting all fields in the keystore that may have been set by the application in the `clearKeystore` function.

Status
The Secret Key Labs team has resolved the issue by updating the code to also delete the stored seed phrase from the key store in the `clearKeystore` function.

Verification
Resolved.

Issue B: Wallet Screen Not Protected Against Screen Recording

Location
`android`

`ios`

Synopsis
On Android and some versions of iOS, the user or an application may record videos or individual frames from the Xverse Wallet application.

The data displayed by the Xverse Wallet is critical to both the security of the application and the privacy of the user. This data includes the mnemonic phrase, which is displayed during wallet key pair generation, and at any time after that in the backup screen. A leak of the mnemonic passphrase is a critical breach of the security of the wallet. Additionally, the wallet displays user account balance and transaction history, of which a malicious screenshot could leak private user data.

Private user data must be secured and protected from access by other applications running on the mobile device.

Impact
A malicious application accessing an image of the mnemonic phrase could result in loss of all wallet funds. In addition, account balance or transaction history leaks are a breach of user privacy.

Preconditions
The user must make a screenshot, and grant other applications access to the screenshot. Alternatively, an application that has access to the contents of the screen (e.g. a screen recording application) would be required.

Feasibility
The feasibility of such an attack depends on the specific phone and OS.

On iOS 10 or newer, there is no API for an application to use to access the screen of another application, so this attack vector is not applicable. However, if the attacker plants a malicious application with access to the photo gallery on the device and the user makes a screenshot of the seed phrase or other private information, it is possible for the application to access it.

On Android, the MediaProjection API can be used to record the contents of the screen. The user has to explicitly consent to the access, and during the duration of the screen recording, an icon is shown. This icon represents wireless transmission of the screen contents to a projector or ChromeCast and may not be immediately identifiable as a screen recording icon. Additionally, any application with access to the files of the user can access all screenshots.

Mitigation

We suggest that the Secret Key Labs team take the following measures to prevent or mitigate the impact of screenshots.

Android

On Android, we recommend setting `FLAG_SECURE` on the application window and making sure that no private content is shown in other windows. For more information, we suggest referring to this blog post about vulnerabilities of weak screenshot protection. The flag can either be set manually or by using an existing React module.

iOS

On iOS, there is no API to prevent screenshots, which makes mitigating this issue more difficult. In some instances, the only thing that can be done is to remind the user that taking screenshots of the seed means making it available to other applications. In other cases, such as the seed generation screen or when the application is being AirPlayed or mirrored, stronger measures could be taken. In this case, the iOS event UIApplicationUserDidTakeScreenshotNotification can be handled to display a notice to the user that the action is not secure, and to generate and display a new seed. With this approach, instead of preventing the screenshotting of a seed that is used, using a seed that is screenshotted is prevented. ScreenShieldKit also warrants mention in this case as a potential resource for preventing screenshots.

Status

Android

The Secret Key Labs team has updated the application to make use of `FLAG_SECURE`, in order to protect the screen against screenshotting. We did not identify any component that evades this protection.

iOS

The Secret Key Labs team has updated the application to make use of a Boolean value that indicates whether the system is actively cloning the screen to another destination (AirPlay, recording, or mirroring) in the `AppDelegate.m` file. We did not identify any component that evades this protection.

Verification

Resolved.

Issue C: Complete Reliance On OS Security Framework

Location

`utils/walletKeychainHelper.ts`

Synopsis

Both the PIN and the mnemonic passphrase are stored in the respective OS's key stores without any additional encryption. In recent research, it was found that mobile platforms do not always live up to their security claims. Specifically, the research lists several cases where, given physical access, extraction of keys is possible. Therefore, in order to provide a secure experience for users who store significant amounts of coins in their wallet, we recommend only trusting the security of the mobile OS conditionally, and implementing fail-safe measures to prevent immediate leakage if the device is stolen.

Impact

Leakage of the mnemonic and private key resulting in loss of funds.

Feasibility

This attack requires specialized equipment and expertise, both of which can be acquired by a motivated adversary.

Preconditions

The attacker needs physical access to the phone.

Remediation

We recommend allowing the user to choose longer PINs, as well as allowing the user to choose authentication by password. From the PIN or password, we recommend deriving a key that is used to encrypt the seed. The key derivation function used should be memory hard and configured to compute for roughly one second (we recommend using Argon2id). The encryption should be done using an authenticated encryption algorithm like libsodium's secretbox or AES-GCM. The nonce or IV can be random and stored alongside the ciphertext. The key should not be used for any other purpose without closely considering joint security.

To demonstrate the impact of a larger alphabet of the PIN/password, as well as increasing the number of characters, we note that a compute-optimized server with 60 cores costs less than 600 USD per month on Google Cloud Engine. Since each derivation, and therefore each password guess takes about a second (we assume a phone core is roughly as fast as a GCE core), we know we get a guess price of:

$$C := \frac{600\frac{USD}{mon}}{60\frac{guess}{s}\times 3600\frac{s}{h}\times 24\frac{h}{day}\times 30\frac{day}{mon}} = 3.858\times 10^{-6}\frac{USD}{guess}\ .$$

However, the attacker may rent multiple instances to increase the guess rate. Given the number a of available characters in a PIN or password (i.e. 10 for a PIN, 36 for a lowercase alphanumeric password), and the amount of money the adversary is willing to spend m, we can get the length a PIN or password needs to have in order to be secure for a month from

$$f(m,\ a)\ = ceil\left(\frac{log(\frac{m}{C})}{log(a)}\right)$$

This formula demonstrates that, while for attackers with tight budgets (less than 36,000 USD), a 10-digit PIN is sufficient to protect a wallet for a month, an attacker that invests several hundred thousand USD will require a 12-digit PIN to achieve the same effect. In both cases, a seven-digit alphanumeric upper and lowercase password or an eight-digit pure lowercase password would have had at least the same effect.

Status

The Secret Key Labs team has resolved this issue by implementing an optional enhanced security mode, where the PIN is used to derive an encryption key. In that mode, the PIN also is not restricted in length or to numerical input. The key is derived using the memory-hard function Argon2, using safe parameters.

Verification

Resolved.

Suggestions

Suggestion 1: Implement a Test Suite

Location

`tests__/App-test.tsx`

Synopsis

Our team found no tests in the repository in-scope. Sufficient test coverage should include tests for success and failure cases, which helps identify potential edge cases, and helps protect against errors and bugs, which may lead to vulnerabilities or exploits. A test suite that includes a minimum of unit tests and integration tests adheres to development best practices. In addition, end-to-end testing is also recommended so that it can be determined if the implementation behaves as intended.

Mitigation

We recommend that the Secret Key Labs team create a test suite for the Xverse Wallet system to facilitate identifying implementation errors and potential security vulnerabilities by developers and security researchers.

Status

The Secret Key Labs team has responded that implementing a test suite will take additional time to implement. As a result, this suggestion remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 2: Remove Unused and Unreachable Code

Location

`components/stackingStatus.tsx`

`components/stackingComponent`

`screens/stxLock`

`screens/confirmLock`

Synopsis

The implementation includes code that is imported and included in the React DOM, but cannot be navigated to. This code is a partially implemented feature that appears in more than one instance. Unused code hinders navigating and reasoning about the source code, which reduces the effectiveness of both security researchers and maintainers.

Mitigation
We recommend that the Secret Key Labs team review the codebase for redundant or unused code and that it be subsequently removed from the codebase.

Status
The Secret Key Labs team has removed the unused and redundant code identified by our team.

Verification
Resolved.

Suggestion 3: Improve Documentation

Location
`https://github.com/LeastAuthority/xverse`

Synopsis
The general documentation provided by the Secret Key Labs team was minimal. Robust and comprehensive general documentation allows a security team to assess the in-scope components and understand the expected behavior of the system being audited. In addition, clear and concise user documentation provides users with a guide to utilize the application according to security best practices.

Mitigation
We recommend the Secret Key Labs team improve the project's general documentation by creating a high-level description of the system, each of the components, and interactions between those components. This can include developer documentation and architectural diagrams.
In addition, we recommend that comprehensive user documentation be created to help users interact with the system correctly and as intended, which encourages secure and correct usage. For example, we noted that the security of the Xverse Wallet is completely dependent on the security of the mobile device's OS, which we suggest disclosing in the documentation to the users so that they may be informed in making security considerations.

Status
The Secret Key Labs team has responded that they intend to improve documentation, however, the undertaking will take additional time to implement. As a result, this suggestion remains unresolved at the time of this verification.

Verification
Unresolved.

Suggestion 4: Increase Code Comments

Location
`https://github.com/LeastAuthority/xverse`

Synopsis
The documentation within the Xverse Wallet codebase is minimal. The documentation contained within the code should be comprehensive and document every function and entrypoint, which significantly aids security researchers in identifying implementation errors and potential security vulnerabilities.

Mitigation
We recommend that the documentation within the code be improved to explain the intended functionality of each of the components, entrypoints, and function.

Status
The Secret Key Labs team has implemented code comments for some parts of the code, particularly for the screen classes as to what is displayed on the screen. However, the average ratio of comment lines per line of code has not significantly improved. As a result, we consider this suggestion to be partially resolved and encourage the Secret Labs team to further increase code comments.

Verification
Partially Resolved.

Suggestion 5: Conduct Additional Security Reviews

Location
`react-native-keychain library`

Synopsis
The Secret Key Labs team intends to implement a biometric authentication component in the Xverse Wallet, however, this component was not in the scope of this security review. Since biometric authentication is a critical security feature, we recommend a review of the biometric authentication component by a third- party security team. Additionally, the Xverse Wallet relies on a react-native-keychain library for the implementation of and access to keystores. As a critical security dependency, we suggest that this library undergo a security review from an independent auditing team.

Mitigation
We recommend additional security reviews of the above mentioned security-critical dependencies.

Status
The Secret Key Labs team has responded they are planning additional security reviews in the future, which have yet to be conducted at the time of this verification.

Verification
Unresolved.

Suggestion 6: Disable Logging in Production Version of the Wallet

Location
`screens/scanQR/scanQR.tsx#L57`

`store/wallet/saga.ts#L283`

`utils/stacksTransactionHelper.ts#L166`

[utils/stacksTransactionHelper.ts#L224]()

[utils/stacksTransactionHelper.ts#L228]()

[utils/stacksTransactionHelper.ts#L233]()

[utils/stacksTransactionHelper.ts#L263]()

[utils/stacksTransactionHelper.ts#L267]()

Synopsis

The Xverse Wallet application has several log message statements in multiple locations in the codebase. While we did not find evidence of leaks of critical data as a result of these log statements, we suggest that logs be only enabled for developer builds of the wallet application.

Mitigation

We recommend that the Secret Key Labs team disable the logging of messages in production builds of the application.

Status

The Secret Key Labs team has removed or commented out code that performs logging in several places. However, we identified instances where logging is still done. As a result, we consider the suggestion partially resolved and recommend that logging be fully disabled.

Verification

Partially Resolved.

Suggestion 7: Update and Maintain Dependencies

Synopsis

Running `npm audit` on the codebase reveals that several dependencies are outdated and have vulnerabilities. While it is not clear to what extent these would be exploitable, it is good practice to keep dependencies up-to-date in order to avoid importing vulnerable code.

Mitigation

The Github Dependabot tool automatically sends emails when a security advisory that affects a codebase is published. We recommend taking these seriously and swiftly updating the affected dependency to a version that is not vulnerable.

Status

The Secret Key Labs team has updated several dependencies and running `npm audit` now reports significantly fewer vulnerable dependencies. However, all vulnerable dependencies have not been updated, thus we consider the suggestion to be partially resolved. We recommend updating all remaining vulnerable dependencies.

Verification

Partially 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 C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. 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. Although we are a small team, we believe that 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 included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site 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. While 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, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, 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 though we have not yet 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 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 report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, 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 resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.