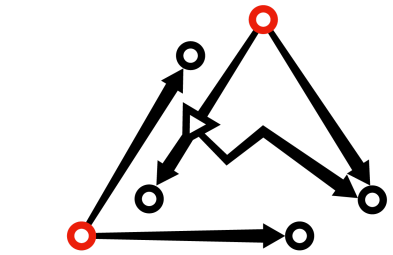

Least Authority

PRIVACY MATTERS

Phantom Wallet

Security Audit Report

Phantom

Final Audit Report: 7 June 2024

Table of Contents

This audit makes no statements or warranties and is for discussion purposes only.

Overview

Background

Phantom has requested that Least Authority perform a security audit of their Phantom Wallet.

Project Dates

- **February 19, 2024 - April 3, 2024:** Initial Code Review *(Completed)*
- **April 5, 2024:** Delivery of Initial Audit Report *(Completed)*
- **June 7, 2024:** Verification Review *(Completed)*
- **June 7, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- Nikos Iliakis, Security Researcher and Engineer
- Ann-Christine Kycler, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Phantom 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:

- Phantom Wallet:
  https://github.com/phantom/wallet

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

- `aea4d38d3c4e9ebc7a02839c94e7b9fb381f1dbf`

For the verification, we examined the Git revision:

- `18c5d927b8063911a3f09f82d31c57e49f171d85`

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

- Phantom Wallet:
  https://github.com/LeastAuthority/Phantom-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:

- Website:
  https://phantom.app

This audit makes no statements or warranties and is for discussion purposes only.

In addition, this audit report references the following documents:

- M. S. Turan, E. Barker, W. Burr, and L. Chen, "Recommendation for Password-Based Key Derivation. Part 1: Storage Applications." *NIST Special Publication 800-132*, 2010, [TBB+10]
- Decision to Revise NIST SP 800-132 | CSRC: https://csrc.nist.gov/news/2023/decision-to-revise-nist-sp-800-132
- Password Storage - OWASP: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- BIP32 Specification: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the wallet;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Malicious attacks and security exploits that would impact the wallet;
- Vulnerabilities in the wallet code, and whether the interaction between the related and network components is secure;
- Exposure of any critical or sensitive information during user interactions with the wallet and use of external libraries and dependencies;
- Proper management of encryption and storage of private keys;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

Summary

Our team performed a security audit of Phantom, a non-custodial, multi-chain wallet, which allows users to trade, receive, send, and store their assets. During our review, we assessed the security aspects associated with deep linking mechanisms and found that all application links are properly configured. We rigorously tested for vulnerabilities that could lead to Open Redirect Exploitation, Cross-Site Scripting (XSS) in WebViews, and the potential theft of information, and did not find any issues.

Since any malicious applications or browser extensions on the same device are out of scope per the threat model, we considered a malicious website for any deep link or dApp vulnerability research and did not find any issues.

In our review of the cryptography used in the wallet, we meticulously examined the components, assessing the suitability of cryptographic primitives, randomness generation, key management, and compliance with security definitions. We also investigated known vulnerabilities and common pitfalls around the use of the primitives and did not find any issues.

In addition to performing a manual review of the codebase, we utilized `Semgrep` and `Sonarqube`, tools for static analysis. Our team ran the tools with their predefined Typescript rules as well as custom rules, investigating OWASP security risks and specific Android security rules, and did not find any issues.

This audit makes no statements or warranties and is for discussion purposes only.

General Comments

Phantom supports Solana, Ethereum, Polygon, and Bitcoin Ordinals. It is available as a mobile application and a browser extension, and also supports hardware wallets. Our team examined the Phantom wallet mobile application, browser extension, and other packages in the monorepo, as indicated by the scope of work (vault, BIP39, BIP32, mnemonic, shared utilities, as well as the `encryptedStorage.ts` and `expoSecureStorage.ts` files). In addition to reviewing the security of the system and the areas of concern listed above, we reviewed the wallet for vulnerabilities and implementation errors and to assess adherence to best practice recommendations. Our team identified suggestions relating to the strength of the key derivation used for the storage encryption key (Suggestion 2), and the implementation of the BIP32 specification (Suggestion 1, Suggestion 2).

Threat Model

In general, it is a common best practice to define a threat model before any security evaluation can be performed. For this audit, the Phantom team provided our team with a specific threat model that excluded theoretical attacks. Our team found this approach of limiting the scope as per the threat model to be reasonable since users should consider the threat model for evaluating the security guarantees provided by the Phantom wallet mobile application and browser extension and responsibly ensure that attacks outside of this threat model are prevented by other measures (e.g., protecting the devices from theft/physical attacks and keeping the devices up to date to reduce the risk of malware infections).

The threat model considers only exploitable security vulnerabilities. If we report an issue here, note that a proof-of-concept for the exploit has been communicated to the Phantom team. Theoretical attacks are not exploitable in our knowledge and may be reported as suggestions or mentioned in the General Comments section. Consequently, any changes to software, platforms, or dependencies should be evaluated to assess the effect this may have on theoretical attacks, as they may result in potential, new exploits.

Furthermore, the threat model does not include compromised devices (e.g., malware being present), physical access to the user's device by the attacker, network attacks, such as man-in-the-middle attacks that require the attacker to be able to intercept the entire network communication of the application, as well as vulnerabilities affecting outdated devices (e.g., devices that are less than two stable versions behind the latest stable version).

We specifically investigated the browser extension and mobile application, which excluded the Phantom backend or any further dependencies, such as the Bluetooth connection protocol, as the connection is handled by the third-party library `@ledgerhq/hw-transport`. We assumed these to be functioning as intended in our approach for this report..

System Design

Our team examined the design of the Phantom wallet and found that security has generally been taken into consideration. The testability and maintainability is increased with the approach of modularized functionalities — that is, individual functionalities are reused between the mobile wallet and browser extension as separate packages that are imported.

Our team reviewed the browser extension and found that the extension has been developed with a strong emphasis on safeguarding against vulnerabilities. The use of `web_accessible_resources` within the extension's manifest has been carefully managed to prevent any sensitive functionality from being exposed to web pages. Communication between the content scripts and web pages is securely established, strategically avoiding any code injections or manipulations that could be leveraged for exploits. Access to web-accessible resources is strictly controlled to prevent sensitive extension features

This audit makes no statements or warranties and is for discussion purposes only.

from being made available to unauthorized web pages. Best practices are observed by steering clear of risky Document Object Model (DOM) operations, such as employing `innerHTML` or `outerHTML`, or the dynamic assignment of `href` attributes, which are known avenues for script injection attacks. The security posture of the extension is robust, characterized by a diligent and security-centric design philosophy that effectively shields against web-based security threats.

Storage

The Phantom team has taken two different approaches for creating secure storage for sensitive data. For the mobile use case (iOS & Android), they utilize the React Native library `expo-secure-store`. For the browser use case, they implement their own secure storage, which is a wrapper of the underlying extension storage and uses the secretbox from `tweetnacl` for the encryption. Our team identified one suggestion relating to the improvement of the key derivation technique (Suggestion 2); however, we have not identified any exploitable issues in this functionality.

DApp Connection

Regarding connecting to dApps, we found that the team has taken the standard approach by injecting the provider in the `window.phantom.* (ethereum etc.)` object for each network. The Ethereum provider is also available at `window.ethereum`.

Deep Links

Phantom's mobile application supports native interaction capabilities through Digital Asset Links (DAL) for the application's deep links. Both the browser extension and mobile application offer dApp connections, hardware wallet connections, as well as specialized phantom deep links. These connections are especially critical because any forged or malicious endpoints of these connections could potentially trigger privileged and protected functionality and, in the worst case, expose secrets or allow unauthorized transactions. Currently, these deep link integrations are exclusive to the Solana blockchain. The primary methods available via these deep links include connecting to Phantom wallet, disconnecting, signing and sending transactions, and browsing web applications within Phantom's in-app browser. Phantom's mobile application also supports sessions that allow continuous interaction between applications. Key generation and shared key derivation used for sessions are handled by the third-party library `tweetnacl`.

To reduce the risk of phishing attacks, the Phantom team maintains an extensive blacklist that is checked before the application establishes a connection (i.e., the dApp URL is checked against the custom phishing detection). This does not prevent phishing attacks altogether, but a well-maintained block list will prevent attacks from well-known phishing campaigns. Additionally, the Phantom team should advise users to take precaution before accepting deep link dApp connection requests.

Code Quality

Our team performed a manual review of the Phantom target code and found the code to be well-organized. The structure of the code resembles the codebases of other mobile and browser extension applications using React Native, which makes it easier to navigate the codebase. Additionally, names of files, types, functions, and variables are descriptive, which further improves readability and comprehension of the intended functionality. However, our team found some deprecated functionality, which the Phantom team plans to resolve in the future, as per the documentation.

Tests

The Phantom team has implemented sufficient tests, which showed high coverage of import logic.

This audit makes no statements or warranties and is for discussion purposes only.

Documentation and Code Comments

The project documentation consisted of detailed explanations on the Phantom website, which offer a clear description of the application and different use cases, facilitate a comprehensive understanding of the system, and help onboard new reviewers and users. Additionally, while the implementation is sparsely commented, our team noted that critical components do contain meaningful code comments, which help inform other developers about the correct usage of functions.

Scope

The scope of this review was sufficient and included all security-critical components. Note that our investigation was limited to the areas detailed in the Threat Model noted above.

Dependencies

We analyzed all the dependencies implemented in the codebase and found them to be generally secure, with the exception of one transitive dependency, `http-cache-semantics`, that is vulnerable to a Denial of Service (DoS) via cache-control headers due to inefficient regex complexity. This issue is indirect since it is introduced by the download package that is used by `@phantom/synpress`. During our review, we did not find a way to exploit this in the application, as all headers are predefined. However, changes in any of the dependencies could potentially expose this vulnerability.

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
Suggestion 1: Add Missing Validation in BIP32	Resolved
Suggestion 2: Enhance Key Derivation Security	Resolved

Suggestions

Suggestion 1: Add Missing Validation in BIP32

Location

`bip32/src/ed25519HdKey.ts#L45`

Synopsis

The BIP32 specification mandates that a key should be deemed invalid if `parse256(IL) ≥ n` or if `ki = 0`, necessitating progression to the subsequent iteration. This crucial validation step is presently absent in the implementation.

Mitigation

We recommend adding the proper checks.

This audit makes no statements or warranties and is for discussion purposes only.

Status

The Phantom team has added the checks.

Verification

Resolved

Suggestion 2: Enhance Key Derivation Security

Location

`shared/utils/encryptionUtils.ts#L11-L14`

Synopsis

In the current setup, the Phantom team uses PBKDF2 with 10000 iterations, SHA256, and 16-bit length salt for deriving an encryption key from the user password. This setup does not conform to best practices, and an attacker with access to ciphertext could brute-force the password and thereby gain access to the encryption key relatively fast.

Mitigation

In general, memory-hard key derivation algorithms, such as `Argon2id` or `scrypt`, are recommended over PBKFD2. However, PBKDF2 is still recommended by NIST, as explained in [TBB+10]. Note that an update to this recommendation is still under development. We recommend finding a memory-hard alternative that is fitting and using the recommended parameters. If no such alternative can be used, we recommend that the iteration count for PBKDF2 be updated to a modern recommendation (OWASP recommends 600,000 iterations with SHA256 for password storage). We additionally recommend considering benchmarking on supported modern systems an d aiming for a key derivation time of 1s — or the highest interval deemed acceptable by the Phantom team's usability requirements.

Status

The Phantom team has introduced `scrypt` to the codebase.

Verification

Resolved

This audit makes no statements or warranties and is for discussion purposes only.

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.

This audit makes no statements or warranties and is for discussion purposes only.

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.

This audit makes no statements or warranties and is for discussion purposes only.