Stacks Investor Wallet

Final Security Audit Report

Blockstack

Report Version: 20 May 2019

Least Authority

PRIVACY MATTERS

Table of Contents

Overview

Blockstack has requested Least Authority perform a security audit of the Stacks Wallet, in anticipation of an updated release prior to May 15, 2019.

The major feature that will be included in the updated release is allowing users to create a software-only wallet, able to send transactions without a hardware wallet device. However, the private keys will not be stored in the wallet. Users will be prompted for the seed phrase on each transaction.

The audit was performed from May 2-10, 2019 by Lily Anne Hall and Dominic Tarr. This final report was issued on May 20, 2019, following the discussion and verification phase.

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Stacks Investor Wallet followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code repositories are in scope:

Specifically, we examined the Git revisions:

```
a9c398f0f98135be115c18e50ac420eabc9b01fb
```

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

Areas of Concern

Our investigation focused on the following areas:

- Private keys are cleared from memory properly after usage
- Identify any other potential security issues associated with the wallet and updated features
- Anything else as identified during the initial analysis phase

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.

Findings

Code Quality

The project code was found to be of high quality, separating concerns into well defined modules, and following widely accepted best practices for applications written with React and Electron. Given this and the relative simplicity of the application and small attack surface, it was straightforward to follow the structure and evaluate. Only one vulnerability was identified and noted as an issue below.

Third Party Dependencies

Like many projects written in Node.js, the Stacks Investor Wallet does make liberal use of third party dependencies. It should be noted that use of third party dependencies present a risk for new issues to be introduced. So, we have included a suggestion to reduce the risk. However, per our review, none of the used production dependencies contains any known vulnerabilities at this time.

Issues

We list the issues we found in the code 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: Seed Phrase Can be Exfiltrated From Disk	*Verified*
Suggestion 1: Pin Dependencies To Exact Versions	*Verified*

Issue A: Seed Phrase Can Be Exfiltrated From Disk

Synopsis

The package used to persist the wallet state to disk, for a brief time window, writes the seed phrase to disk in clear text. An attacker could take advantage of this timing vulnerability to steal the seed phrase and all of the user's funds.

Impact

Critical. If an attack was successful, the user could have their balance drained.

Preconditions

Attacker would need to have tricked the user into downloading malware intended to perform the seed exfiltration or an upstream dependency could be modified unknown to the developers of the Stacks wallet.

Feasibility

Moderate. If an attacker managed to get the target user to install a malicious program that cleared the state configuration file and watched for changes, they would be able to steal the seed. Possibly a more likely scenario may be that, similar to the event-stream hack, one of the wallet's third-party dependencies is updated to include this malicious code, thus packaging the hack with the app itself upon release.

Technical Details

The wallet makes use of a package called "electron-store", which is designed to persist arbitrary configuration data to disk for electron applications. This is used in the wallet to persist part of the wallet's state information. If the user does not make use of a hardware wallet, the application will generate a seed phrase, display it, ask for confirmation, then proceed to the dashboard.

After confirming the seed phrase, the state persistence is initialized and writes the state to disk as JSON. This includes a "seed" property, but it's null. If the user never resets the wallet, then the seed will never be written to disk. However, if the user **resets the wallet** and creates a new one, the state persistence is already initialized and there is a time window where the seed phrase is part of the application state and is written to disk in cleartext.

This simple script allowed us to successfully exfiltrate the seed phrase:

```
const fs = require('fs');
fs.watch('%PATH_TO_CONF%', function() {
  try {
    let json = JSON.parse(fs.readFileSync('%PATH_TO_CONF%').toString())
    if (json.wallet && json.wallet.seed) {
      console.log('SEED FOUND', json.wallet.seed);
    }
  } catch (err) {}
});
```

This malicious code could be hidden in a dependency or another application that wallet users may be likely to use (since the code can run anywhere and does not have to be embedded in the application itself).

Remediation

The storage() function in *app/store/persist/index.js* returns a get/set/remove proxy. The set() method can be adapted to filter the seed from the data passed to the underlying store.

Status

Verified.

Verification

The storage proxy's set() method was adapted to force the wallet seed value to null before writing to disk.

Suggestions

Suggestion 1: Pin Dependencies To Exact Versions

Synopsis

Third party dependencies are a target for attackers to insert malicious code into downstream projects if using a "compatible with" semver tag.

Mitigation

Pinning the production dependencies to exact versions can reduce the possibility of inadvertently introducing a malicious version of a dependency in the future.

Status

Verified.

Verification

Dependencies have all been pinned to exact versions.

Recommendations

This final report confirms that the *Issue* and *Suggestion* stated above have been addressed and followed up with verification by the auditing team.

We recommend that the code quality and organization continue to be maintained with industry best practices, as seen here. The risks presented by the third-party dependencies should be continuously evaluated for further mitigation, also with industry best practices.

As usually recommended, additional audits should be conducted on future development releases to ensure that any potential issues and vulnerabilities are identified, addressed and verified as soon as possible.