Security Audit Report: TezBox

Tezos

Report Version: 11 September 2018

Least Authority
Freedom Matters

Table of Contents

Overview

Coverage

- Target Code and Revision
- Areas of Concern

Methodology

- Manual Code Review
- Vulnerability Analysis
- Documenting Results
- Suggested Solutions

Findings

- Code Quality and Results
- Issues
    - Issue A: Unsafe Target Origin Used In window#postMessage
    - Issue B: Unauthenticated Extension Invocation - Fingerprinting
    - Issue C: Unauthenticated Extension Invocation - Phishing
    - Issue D: Opening of Unencrypted Connection
    - Issue E: Seed Phrase Backup Confirmation and Validation
    - Issue F: Weak Password Validation Requirements
    - Issue G: Private Key Can Be Exfiltrated From Disk In Clear Text

Recommendations

- Consider Replacing the Chrome Plugin with the Web Wallet

Overview

Tezos has requested that Least Authority perform a security audit of TezBox, a wallet used by the Tezos community and developed by Stephen Andrews.

The audit was performed from August 6 - 13 2018 by Emery Hall. The initial report was issued on August 15, 2018. This updated report was issued on September 11, 2018 following the discussion and verification phase.

Coverage

Target Code and Revision

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

- TezBox Chrome extension and web wallet built with Angular 1 (share 90% of the code, with the exception being the targeted device / platform):
    - Chrome extension: https://github.com/tezbox/tezbox-chrome-plugin
    - Web Wallet: https://github.com/tezbox/tezbox-web-wallet
- Wallets use JS library for interaction with the chain (eztz.min.js):
    - https://github.com/stephenandrews/eztz

Specifically, we examined the Git revisions:

```
tezbox/tezbox-web-wallet@7ac3bd367e36dd4e7c62af0973fec31084d60279

tezbox/tezbox-chrome-plugin@3c221b1dd71b4182d975f44de8b527d588eca2a5

stephenandrews/eztz@f6982fe7465443571f0b68ac5fc58a177b5c0cbb
```

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:

- Any attack that impacts funds, such as draining or manipulating of funds;
- Application permissions and excess authority;
- Input validation, sanitation, and XSS vulnerabilities;
- User experience and design choices that could lead to phishing or user manipulation;
- Any attack that impacts funds within the wallet;
- The management of private keys within the wallet; and
- Communications between the client wallet and servers.

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 protects users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the smart contract 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 website to get a high level understanding of what functionality the software under review provides. We then meet with 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.

Findings

Code Quality and Results

Overall, the source code for both the web wallet and Chrome plugin is well structured and easy to follow. This is an important property when it comes to maintaining the codebase and running productive audits. The JavaScript follows generally accepted best practices and idiomatic patterns in most areas. In fact, the majority of our findings are almost entirely correlated to excess authority and permissions within the Chrome plugin with a few exceptions. While the web wallet and eztz libraries were found to be reasonable secure, the Chrome plugin was found to contain a number of critical issues that should be addressed immediately.

We recommend that replacing the Chrome plugin code with the web wallet is considered. We've included more details about this in the *Recommendations* below.

Issues

We list the issues we found in the code in the order we reported them.

ISSUE / SUGGESTION	STATUS
Issue A: Unsafe Target Origin Used In window#postMessage	*Resolved*
Issue B: Unauthenticated Extension Invocation - Fingerprinting	*Resolved*
Issue C: Unauthenticated Extension Invocation - Phishing	*Resolved*
Issue D: Opening of Unencrypted Connection	*Resolved*
Issue E: Seed Phrase Backup Confirmation and Validation	*Resolved*
Issue F: Weak Password Validation Requirements	*Resolved*
Issue G: Private Key Can Be Exfiltrated From Disk In Clear Text	*Resolved*

Issue A: Unsafe Target Origin Used In window#postMessage

Synopsis

The script *inject.js*, which is injected into web pages, uses the window.postMessage method to communicate with the content script running from the extension. The window.postMessage method accepts a targetOrigin argument that specifies which windows should receive the message. This argument is provided as a wildcard in the code, which is an unsafe security practice.

Impact

Moderate. Failing to provide a specific target discloses the data you send to any interested malicious site.

Preconditions

User has a malicious site opened.

Feasibility

High. A malicious site can change the location of the window without your knowledge, and therefore it can intercept the data sent using postMessage .

Remediation

Always provide a specific targetOrigin , not *, if you know where the other window's document should be located. In the case of this extension, since the recipient of this message is running as a content script, you can replace * with window.origin to ensure that only scripts running in this window may receive the message. From there the content script uses chrome.runtime.sendMessage , which is safe.

Verification

The offending code has been entirely removed.

Status

Resolved.

Issue B: Unauthenticated Extension Invocation - Fingerprinting

Synopsis

The *inject.js* script, which is injected into every web page, exposes an API for interacting with the extension. This API provides methods getActiveAccount and getAllAccounts, which expose all of the user's public keys associated with the wallet.

Impact

Critical. This is a massive privacy concern. Any web page can determine the user's public key and can use this information to follow them around the internet.

Preconditions

None.

Feasibility

High. Any web page can call tbapi.getAllAccounts and store the result of this information to correlate with the user's session, login, or other personally identifiable information.

Technical Details

Much like how Facebook like buttons and perma-cookies work, this allows attackers to correlate traffic and monitor users to build profiles for other types of attacks. If an attacker controls more than one web page that the user visits, the attacker can correlate that traffic to the same individual. Additionally, the knowledge of a public key can be used to determine the user's Tezos balance, further leaking sensitive information about a user to any web page and providing plenty of information for choosing a high yield target.

Mitigation

Disable the methods for getActiveAccount and getAllAccounts as the privacy concerns far outweigh any potential use case for web pages accessing this information automatically.

Remediation

Engineer a request mechanism, where a web page that wishes to read the user's public key may trigger a request popup from the extension and the user may confirm or deny the request to reveal this information.

Verification

The offending code has been entirely removed.

Status

Resolved.

Issue C: Unauthenticated Extension Invocation - Phishing

Synopsis

The *inject.js* script, which is injected into every web page, exposes an API for interacting with the extension. This API provides methods signMessage and initiateTransaction, which can be used to make use of the user's private key for arbitrary signing of some data (including transactions) as well as triggering the sending of funds to an arbitrary address.

Impact

Critical. Visiting a malicious or compromised website could lead to loss of funds or draining of wallet.

Preconditions

User visits a malicious or compromised web page.

Feasibility

High. Phishing attacks are very common and can be very effective, especially when executed through an interface that the user trusts.

Technical Details

Any page can call tbapi.initiateTransaction with arbitrary parameters parameters, destination, and amount, and the extension will show these in a popup interface for sending a transaction. This can be used effectively in phishing attempts or tricking the user into sending tokens somewhere.

Additionally, any webpage can just trigger these popups to spam users in hopes while they are closing them that one accidentally gets confirmed. This is pretty feasible given the cancel button is directly under the confirm button. Since there is no built in protection against this like with traditional popups, any web page can blast these until the host computer runs out of memory.

Any page can call the tbapi.signMessage method - again - from *any* webpage. The extension doesn't automatically sign, it does open the extension and prompt the user to confirm, but, this makes phishing attacks trivial. Any webpage can trigger signature requests and get a signature from the user's private key if the user accidentally or even intentionally clicks the sign button.

Combined with the data leaked in *Issue B*, it is trivial for an attacker to craft a transaction message that drains the user's wallet and request for the user to sign it. Since this prompt is shown in the wallet itself - an application the user trusts - it's reasonable to believe that this could be a very effective phishing attack, especially if executed from a trusted but compromised web page.

Mitigation

Disable the methods for signMessage and initiateTransaction as the security concerns far outweigh any potential use case for web pages triggering these dialogues automatically.

Remediation

Engineer a request mechanism, where a web page that wishes to make use of these features of the extension (such as an ecommerce site or block explorer) can request to be added to a whitelist. Requests to sign a message or initiate a transaction should be verified by a cryptographic signature that corresponds to a public key added to the whitelist to prevent compromised pages from being able to impersonate the host page and trick the user into signing transactions.

Verification
The offending code has been entirely removed.

Status
Resolved.

Issue D: Opening of Unencrypted Connection

Synopsis
Viewing the details of a transaction opens the tzscan block explorer via clear text connection.

Impact
Moderate. It's possible for a man in the middle (MITM) attack to rewrite the response from tzscan and potentially trick a user into resending a transaction.

Preconditions
User's network has been compromised or any link on the network path to tzscan.

Feasibility
Unknown. The MITM attack feasibility is high, but an attacker's ability to use this in order to successfully trick a user into sending funds depends greatly on other facets of such a hypothetical attack.

Remediation
The tzscan website support HTTPS, so simply change the URL to use the SSL version of the site. Contact the site administrators and request that they redirect traffic on port 80 to 443 and enable HSTS.

Verification
The code has been updated to use the SSL version of the tzscan site.

Status
Resolved.

Issue E: Seed Phrase Backup Confirmation and Validation

Synopsis
During the creation of a new wallet, the user experience may drive users to accidentally lose or incorrectly record their seed phrase.

Impact
Moderate. A user that is less diligent may lose their funds.

Preconditions
None.

Feasibility
High. User experience does not encourage best practices for key backup.

Technical Details

When creating a new wallet, after the seed phrase is displayed, clicking outside of the extension (or browser), destroys the extension state and loses the seed phrase. So copying the seed phrase into a text file to back it up, resets the wallet back to the beginning. The user can restore the wallet upon reopening, but may not realize they can still enter an additional passphrase since they may not have done when the generated the seed originally.

Furthermore, when creating a new wallet there is no request for the user to confirm the seed phrase, which is a commonly accepted best practice for cryptocurrency wallets.

Remediation

Move the wallet creation flow from the extension popup into either a normal popup or a new tab, so that clicking outside of the window does not close the application and destroy its state. Require that the user confirm the seed phrase by typing it again on the following screen before continuing. Do not allow pasting into this field to force the user to have written the phrase down or otherwise saved it to a file off the clipboard.

Verification

The extension now operates entirely within a new tab instead of a popup window. Users are forced to confirm their seed phrase before continuing.

Status

Resolved.

Issue F: Weak Password Validation Requirements

Synopsis

The wallet key encryption passphrase requirements are weak and do not encourage strong password best practices. This affects both the Chrome plugin and the Web Wallet.

Impact

Low. The public key hash is used as a salt for PBKDF2 and therefore even weak passwords should be protected against dictionary attacks. However, given direct access to the user’s machine, a weak password may be found relatively quickly.

Preconditions

Attacker has direct access to the user’s computer.

Feasibility

Unknown. Depends largely upon the user and their individual threat model.

Technical Details

The only requirement that is enforced on passwords is that they are a minimum of eight characters. This means that “password” is valid and therefore this requirement does not really encourage good password security practices. Additionally the current validation does not trim whitespace, so eight spaces is also valid.

Remediation

Generally passwords should be at least 8 characters, but also contain at least one case change, and possibly a combination of number and/or symbols, and contain more that just whitespace.

Verification

All of the suggested passphrase validation requirements are now enforced.

Status

Resolved.

Issue G: Private Key Can Be Exfiltrated From Disk In Clear Text

Synopsis

Before encrypting the user's private key and after the wallet is unlocked, it is written to local storage in clear text. This allows any other applications running as the user or higher to read the private key from disk.

Impact

Critical. Can lead to complete loss of funds.

Preconditions

None.

Feasibility

High. Any application running on the user's account or higher can read the private key. This means that any malware or other scripts designed to exfiltrate this information will have no problem getting it if the user installs such programs. This functionality could be easily disguised as another Tezos wallet or related application.

Technical Details

Before encryption and after unlocking the TezBox wallet, the user's private key is written to `localStorage` under the key temp.sk. Chrome and other browsers use LevelDB under the hood to store items written to `localStorage` and save this embedded database under the user's home directory.

Due to the nature of LevelDB's design, deleting items from `localStorage` does not delete them from LevelDB, because a "delete" operation in LevelDB does not actually remove data from disk, but instead "tombstones" it for later deletion when the database performs a compaction. Since the amount of data in this LevelDB will always be very small (as it is scoped to the extension), it is unlikely that it will ever reach a compaction checkpoint, therefore leaving the secret key on disk until the extension is uninstalled.

In many cases the data may not even leave the log because it is so small. On GNU+Linux, our team was able to exfiltrate the private key simply with:

```
cat ~/.config/chromium/Default/Local\ Extension\
Settings/fegbnheadgpfhmdkiignjegedpfobajn/000003.log | grep -a edsk
```

Remediation

Do not ever write secrets to local storage in clear text. Only retain them in memory while the application is unlocked. Inform all current users that they must uninstall the extension, reinstall a fixed version, and restore their wallet to remove the cleartext private key from disk.

Verification

The chrome extension now uses the web wallet code, which is not susceptible to this vulnerability

Status

Resolved.

Recommendations

Per our recommendation, the Issues stated in the initial report were addressed and followed up with a verification by the auditing team. We recommend that future audits be conducted on future development releases to ensure that any potential issues and vulnerabilities are identified, addressed and verified.

Consider Replacing the Chrome Plugin with the Web Wallet

As an alternative to the remediations noted for Issues B and C, we suggest that replacing the Chrome plugin with the web wallet is considered. Since the web wallet and the Chrome plugin share a large amount of the same code, but the web wallet was found to be of much sounder security than the plugin, we recommend considering abandoning the parts of the Chrome plugin that are specific to being an extension (script injection) and instead just write a strict Chrome plugin manifest for the web wallet. It is our teams opinion that the ability for any web page to trigger actions within the wallet is an anti-feature and poses more risk than it’s worth in user experience.

Verification

All of the code specific the chrome extension that was the subject of concern in this report has been removed and replaced with the web wallet code per this recommendation.

Status

Resolved.