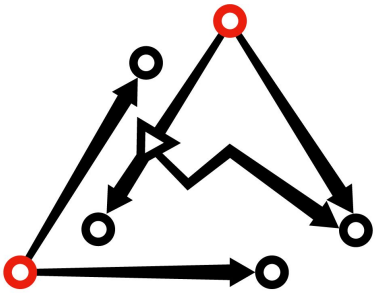

Least Authority

PRIVACY MATTERS

Galleon Wallet

Final Security Audit Report

Tezos

Report Version: 13 July 2018

Table of Contents

Overview

The Tezos Foundation requested that Least Authority perform a security audit of the Galleon Wallet developed by Cryptonomic, in preparation for the upcoming Tezos beta net and main net launches.

The audit was performed from June 11 - 19, 2018 by Gordon Hall and Dominic Tarr. The initial report was issued on June 19, 2018. The updated report was issued on July 13, 2018, following a discussion and verification phase.

Coverage

Target Code and Revision

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

- Typescript-based client side library that manages wallet files and keys, encryption / decryption and coordination with the back end systems, including Conseil and the Tezos node: https://github.com/Cryptonomic/ConseilJS
- Scala-based server side API for querying cached Tezos blockchain data: https://github.com/Cryptonomic/Conseil
- Wallet source code (`Tezos-Wallet-develop.zip` provided by Cryptonomic on June 11, 2018)

Following the delivery of the initial report, we were instructed to verify remediation of the issues and examined:

- https://github.com/Cryptonomic/Tezos-Wallet `(b8988689f321e076f6f568971cbe02162d607b55)`
- https://github.com/Cryptonomic/ConseilJS `(0d0a03aebff8ea186dc385aa1e14fc0038e3c052)`

Manual Code Review

In manually reviewing the Conseil.js and the Wallet code, we looked for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also reviewed for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus was on Conseil.js and the Wallet code, we examined dependency code and behavior when it was relevant to a particular line of investigation.

Our investigation focused on the following areas:

- Any attack that impacts funds within the wallet,
- The management of private keys within the wallet, and
- Communications between the client wallet and servers.

Findings

Code Status

In reviewing the version of the source code provided to us on June 11, 2018 (`Tezos-Wallet-develop.zip`), we found that due to the incomplete status of development, including the UI in particular, the conditions of the code were not optimal for a security audit. Our main source of concern was that there was a number of critical security vulnerabilities in that codebase and the significant changes that needed to take place prior to deploying this wallet for production use.

While we understood this project was in a rush to be completed by a previously scheduled launch date, we recommended that schedules be extended, as necessary, to further develop the code and have it be re-audited. With the wallet being core to the management of Tezos funds in the community, the status of the code security should be a priority.

Our team found the code to be in a much better condition by the time we did our verification, which was clearly delayed to make many improvements.

Issues

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

ISSUE / SUGGESTION	STATUS
Issue A: Missing Passphrase Validation on Wallet	*Unresolved*
Issue B: Function with Unnecessary Knowledge of Private Key	*Resolved*
Issue C: Wallet Ignores Invalid SSL Certificates for Conceil Server	*Partially Resolved*
Issue D: Encryption Utility Does Not Impose Passphrase Restrictions	*Resolved*
Issue E: User Passphrase for Wallet is Overwritten Upon Update	*Resolved*
Issue F: Newly Created Wallets Do Not Persist Between Restart	*Resolved*
Issue G: Nautilus Queries Are Sent Unencrypted	*Unknown*
Issue H: Created Wallets Overwrite Others With Name Conflicts	*Resolved*
Suggestion 1: Avoid Throwing Types That Don't Provide a Stack Trace	*Resolved*

Issue A: Missing Passphrase Validation on Wallet

Synopsis

The `TezosWallet#saveWallet` function accepts a passphrase in order to encrypt the wallet JSON file, however, there are not any requirements for this passphrase or validation to enforce those requirements, allowing the wallet to be encrypted with an empty passphrase or weak passphrase.

Impact

End users are able to effectively bypass securing their wallet with a passphrase, since an empty string is valid. While the strength of the user's passphrase is their own choice, there should exist some basic strength requirements at a minimum.

Preconditions

None.

Feasibility

High. A user is able to pass an empty string as a passphrase and it will be accepted as valid.

Remediation

The function in question should be adapted to include a few validation checks before accepting the passphrase. Passphrase strength requirements can vary, however a good baseline is:

- Minimum of 8 characters
- Does not contain the username (or in this case the wallet name)
- Must use at least three of the four character types: lowercase letters, uppercase letters, numbers, and symbols

Status

No modifications were made to validate the supplied passphrase to the `TezosWallet#saveWallet` function. However, we have taken note that this logic has been implemented at the application layer within the Tezos Wallet user interface.

Verification

Unresolved.

Issue B: Function with Unnecessary Knowledge of Private Key

Synopsis

The `TezosOperations#forgeOperations` function unnecessarily receives a full instance of `KeyStore`, which contains the user's private key. This function only makes use of the hash of the public key, which is stored as a property on the `KeyStore`.

Impact

Currently there is no impact, however, we recommend following the principle of least authority. Any component of a system should only be granted the authority it requires to perform its duty. In this case, since the function in question does not require the secret key, it should not be aware of it. If a future iteration of this function (or any other) introduces a vulnerability, the private key will have been unnecessarily exposed.

Preconditions

None.

Feasibility

Low or unknown. Currently no impact.

Remediation

The function in question only requires the public key hash to perform its duty, so the function signature should be changed to reflect this. Functions that call this function with a `KeyStore`, should be changed to pass it only the `KeyStore#publicKeyHash`. If there are areas where functions calling this function only accept a `KeyStore` to pass it through, their signature should also be changed to accept the public key hash directly.

Status

The `TezosOperations#forgeOperations` function no longer depends upon or receives a `KeyStore` instance containing the user's private key.

Verification

Resolved.

Issue C: Wallet Ignores Invalid SSL Certificates for Conceil Server

Synopsis

The query made by the function `queryConceilServer` uses an HTTPS agent with `rejectUnauthorized` set to `false`, allowing a man-in-the-middle to intercept requests and impersonate the Conceil server. In addition, the application uses a hardcoded API key "hooman" to authenticate with the remote server.

Impact

Severe. An attacker can present an invalid SSL certificate and decrypt the payload intended for the Conceil server. The attacker may also respond to the request with invalid data to trick end users. The attack may forward the request and intercept the upstream response to learn private and sensitive information about the user including:

- Accounts, which contains private data like balances
- Operations, which contains sensitive data like secrets
- Operation Groups

In addition to eavesdropping on the communication between the wallet and the remote server, the attacker would also be capable of replaying messages, including transaction operations, in order to drain the user's funds if the remote node does not provide appropriate protection against such attacks. Based upon the payloads sent by the wallet, it appears the remote server does not provide such protection.

Preconditions

None.

Feasibility

High. Trivial attacks when using the wallet on a public network like coffee shops or hotels using a rogue access point like a WiFi Pineapple.

Remediation

Reject unauthorized SSL certificates and use X.509 certificate pinning to ensure that communication with the remote server is authenticated.

Status

There is no longer a hardcoded Conceil server connection that explicitly ignores invalid SSL certificates. Without explicitly ignoring unauthorized certificates, the default behavior is to reject such connections.

Users may input their own Conceil server - which may be using an unencrypted connection, however invalid certificates will fail. We have taken note that a ticket is currently open to prevent users from setting unencrypted Conceil servers (https://github.com/Cryptonomic/Tezos-Wallet/issues/174).

Verification

Partially resolved.

Issue D: Encryption Utility Does Not Impose Passphrase Restrictions

Synopsis

The function `encryptMessage` does not impose any restrictions on the supplied passphrase, allowing the user to use an empty passphrase.

Impact

This is similar to *Issue A*, but is implemented as a generic encryption utility within the wallet code, so it may be used to secure more than just wallet data in the future.

Preconditions

None.

Feasibility

High. A user is able to pass an empty string as a passphrase and it will be accepted as valid.

Remediation

The function in question should be adapted to include a few validation checks before accepting the passphrase. Passphrase strength requirements can vary, however a good baseline is:

- Minimum of 8 characters
- Does not contain the username (or in this case the wallet name)
- Must use at least three of the four character types: lowercase letters, uppercase letters, numbers, and symbols

Status

Users are now forced to input strong passphrases in order to encrypt their wallet.

Verification

Resolved.

Issue E: User Passphrase for Wallet is Overwritten Upon Update

Synopsis

The function `saveUpdatedWallet` function calls `TezosWallet#saveWallet` with a hardcoded passphrase of “password”.

Impact

Severe. All user wallets will be encrypted with the same weak passphrase “password”. This both renders all wallets trivial to compromise given access to the encrypted wallet file (either human or another program running on the same machine) and causes users to be unable to access their wallet since they are unaware of the passphrase with which it was actually encrypted.

Preconditions

None.

Feasibility

High. All wallets are encrypted with the same passphrase. Reproducible by simply creating a wallet, re-running the wallet and decrypting it with “password”.

Remediation

Remove the hardcoded passphrase and use the passphrase supplied by the user.

Status

Updating the wallet no longer overwrites the encryption key.

Verification

Resolved.

Issue F: Newly Created Wallets Do Not Persist Between Restarts

Synopsis

The function `submitAddress` writes the encrypted wallet file to `/tmp` causing it to be deleted by the operating system upon the next reboot.

Impact

Severe. All wallets that are created using the application (not imported) will be deleted after a computer restart. If funds are received before restarting, the user will lose access to them permanently.

Preconditions

Wallet is created using the application.

Feasibility

High. All wallets created with the application are written to `/tmp`.

Remediation

Write the wallet file to a non-temporary location. A common practice is `$HOME/.config/$APP_NAME/*` on POSIX systems. The Electron framework also provides the *app.getPath* function to handle this in a cross-platform manner.

Status

Users are now prompted to save their wallet file to a location of their own choosing.

Verification

Resolved.

Issue G: Nautilus Queries Are Sent Unencrypted

Synopsis

The function `runQuery` sends unencrypted HTTP requests to a Tezos node at `http://nautilus.cryptonomic.tech:8732`, allowing trivial man-in-the-middle attacks.

Impact

Moderate. While the payloads sent to this endpoint do not appear to contain any sensitive information, an attacker can deny service or respond with incorrect blockchain data in order to trick the user.

Preconditions

None.

Feasibility

High. Trivial attacks when using the wallet on a public network like coffee shops or hotels using a rogue access point like a WiFi Pineapple.

Remediation

Secure the remote server with SSL. Reject unauthorized SSL certificates and use X.509 certificate pinning to ensure that communication with the remote server is authenticated.

Status

While no unencrypted Tezos node is hardcoded any longer, users are currently allowed to set their own server URL, which is not validated to be using HTTPS. We have taken note that a ticket is currently open to prevent users from setting unencrypted Tezos node URLs (https://github.com/Cryptonomic/Tezos-Wallet/issues/174)..

Verification

Unknown.

Issue H: Created Wallets Overwrite Others With Name Conflicts

Synopsis

In the same code path as *Issue F*, where wallets are created by the application, wallet files are written to disk using the user-provided name (i.e. `wallet-name.json`). Since there is no checking for existing wallets by that name before proceeding, creating a new wallet with a name conflict will overwrite the existing wallet.

Impact

Severe. Creating a new wallet with a name conflict with an existing wallet will cause the user to lose the private key and associated funds.

Preconditions

A wallet is created using the application.

Feasibility

High. Will be triggered any time an additional wallet is created with a name conflict.

Remediation

Use the generated public key hash as a prefix (or postfix) to the wallet name in the written JSON document's file name. This will ensure that there will not be name conflicts and users do not have to worry about how they name their wallets. Example: "`wallet-name-{pubkeyhash}.json`".

Status

Users are now prompted to select the absolute location, including file name, of the wallet.

Verification

Resolved.

Suggestions

Suggestion 1: Avoid Throwing Types That Don't Provide a Stack Trace

Synopsis

The `decryptMessage` function in ConceilJS, throws a string "`The cipher text is of insufficient length`". While this is a perfectly legal statement in JavaScript, generally the idiomatic approach is to throw `Error` objects. This helps ensure that error handling code is consistent and that stack traces are provided in all environments.

Remediation

Instead of throwing a string, wrap the message in an `Error` constructor, `(new Error('...'))`.

Status

The code in question was adjusted to throw an `Error` object.

Verification

Resolved.

Recommendations

We recommend that the remaining *Issues and Suggestion* stated above are addressed as soon as possible and followed up with another verification by the auditing team. Additionally, we recommend that serious consideration be given to the comments included in the *Code Status* section and that discussions are continued. Although the application has been greatly improved before verification, we still recommend that future security audits are completed.