apriorit

Tezori wallet and ConseilJS library

Security audit report

Prepared for Cryptonomic

June 26, 2020

Table of contents

Project summary

Name	Tezori wallet and ConseilJS library
Source	<table><tr><th>Repository</th><th>Revision</th></tr><tr><td>*https://github.com/Cryptonomic/T2*</td><td>branch/develop - 6e5720e</td></tr><tr><td>*https://github.com/Cryptonomic/ConseilJS*</td><td>branch/master - f1672b4</td></tr><tr><td>*https://github.com/Cryptonomic/ConseilJS-softsigner*</td><td>branch/master - 2909180</td></tr><tr><td>*https://github.com/Cryptonomic/ConseilJS-ledgersigner*</td><td>branch/master - db4ffa4</td></tr></table>
Methods	Code review, Static analysis, Manual penetration testing

Glossary

Code injection - an attack that exploits bugs in data processing. The attack is used to introduce (“inject”) malicious code into a vulnerable program.

Certificate pinning - explicit validation of a certificate of a remote host. The host certificate must match the expected value exactly, ignoring all other trust anchors. This is useful when it is necessary to ensure that the connection is made to a specific host directly.

MiTM attack - man-in-the-middle attack. An attack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other.

XSS attack - Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web pages. The malicious script is executed once the web page is displayed to a user.

Electron - an open-source software framework for creating native applications with web technologies like JavaScript, HTML, and CSS. It combines the Chromium rendering engine that is used to display the web-based UI and the Node.js runtime that is used for native functionality.

Executive overview

Apriorit conducted a security assessment of the Tezori wallet to evaluate its current state and risk posture.

This security assessment was conducted in June 2020 to evaluate the application's exposure to known security vulnerabilities, to determine potential attack vectors and to check if any of them can be exploited maliciously.

Summary of strengths

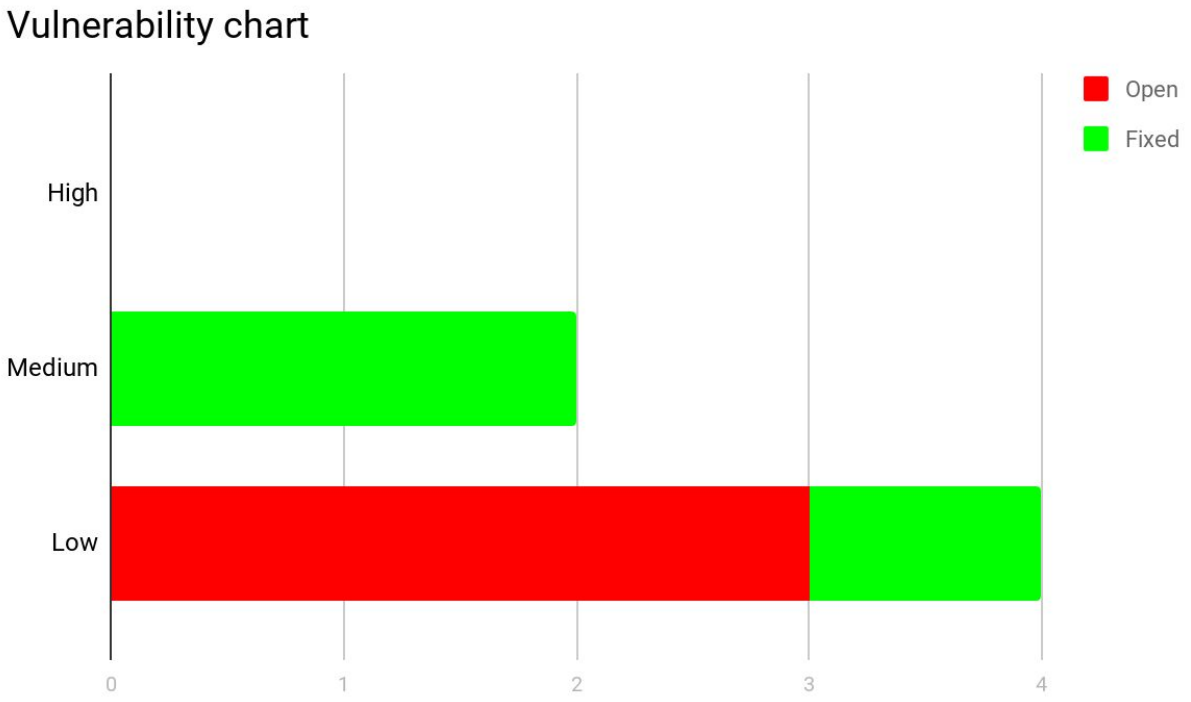
Building upon the strengths of the available implementation can help better secure it by continuing these good practices. In this case, a number of positive security aspects were readily apparent during the assessment:

1. Based on the code review and manual testing
    - The application is well maintained and the code standards are noticeably high.
    - The application does take care to avoid displaying untrusted content.
    - The application seems to perform only the expected functionality, without undocumented or experimental and potentially exploitable features.
    - Strict password requirements encourage users to use secure passwords and protect the wallet from brute-force attacks even if the wallet file is leaked.
2. Based on static analysis
    - Linters and code quality analyzers were used during development.
    - The wallet is covered by unit-tests and the test coverage is fairly good.

Summary of discovered vulnerabilities

During the assessment no high risk vulnerabilities were discovered which indicated good attention to security in the implementation.

Overall, only 2 medium risk vulnerability and 4 low-risk vulnerabilities were discovered during the assessment. 3 out of 6 vulnerabilities were fixed during the audit (2 medium and 1 low). Apriorit recommends to consider addressing the following areas. For more detailed information on all of the findings discovered, refer to the detailed findings section (Appendix A: Detailed Findings) of the report.



Risk Rating	Finding Name	Recommendation	Status
Medium risk	Code injection attack in smart contract deployment dialog	It is recommended to fix the issue by sanitizing the input before rendering it and avoiding usage of “dangerouslySetInnerHTML”	FIXED
Medium risk	Unencrypted private key stored in-memory	It is recommended to modify the wallet to store encrypted private keys at all times and decrypt them only when it is necessary to sign data.	FIXED
Low risk	External debugger attacks	It is recommended to check for unwanted console arguments and environment variables in the main node process (main.js). If any suspicious arguments are detected, the app can notify the user about potential tampering.	OPEN
Low risk	Electron debug console attacks	It is recommended to override the key shortcut that opens the debug console (Ctrl - Shift - I) to prevent users from running potentially dangerous code in the app.	FIXED
Low risk	MiTM attacks	It is recommended to add strict certificate pinning to avoid tampering with wallet traffic.	OPEN
Low risk	A trusted node is used without verification	It is recommended to implement certificate pinning to help detect if the trusted node is compromised.	OPEN

Security rating

Apriorit reviewed Cryptonomic security posture in regards to the Tezori wallet, and Apriorit consultants identified security practices that are strengths as well as vulnerabilities that create risks. Taken together, the combination of asset criticality, threat likelihood and vulnerability severity have been combined to assign an assessment grade for the overall security of the application. An explanation of the grading scale is included in the second table below.

In conclusion, Apriorit recommends that Cryptonomic continues to follow good security practices that are already established and further improves security posture by addressing all of the described findings.

	High	Medium	Low	Security	Grade
Tezori wallet and supporting libraries	0	0	3	Moderately Secure	B

Security grading criteria

Grade	Security	Criteria Description
A	Highly Secure	Exceptional attention to security. No high or medium risk vulnerabilities with a few minor low risk vulnerabilities.
B	Moderately Secure	Good attention to security. No high risk vulnerabilities with only a few medium or several low risk vulnerabilities.
C	Marginally Secure	Some attention to security, but security requires improvement. A few high risk vulnerabilities were identified and can be exploited.
D	Insecure	Significant gaps in security exist. A large number of high risk vulnerabilities were identified during the assessment.

Scope of work and approach

The audit was focused on an in-depth analysis of a desktop wallet application for the Tezos blockchain network. Some wallet dependencies (CounseliJS) were also reviewed in regards to the wallet application.

The wallet was checked in regards to the following:

- Wallet key storage analysis, check for exposed private keys.
- Smart contract interaction analysis.
- External dependency analysis.
- Check that best practices followed throughout the source code.

The audit was performed using manual code review and automated static analysis. Once some potential vulnerabilities were discovered, manual attacks were performed to check if they can be easily exploited.

Appendixes

Appendix A. Detailed findings

Risk rating

Our risk ratings are based on the same principles as the Common Vulnerability Scoring System. The rating takes into account two parameters: exploitability and impact. Each of these parameters can be rated as high, medium or low.

Exploitability - What knowledge the attacker needs to exploit the system and what pre-conditions are necessary for the exploit to work:

- **High** - Tools for the exploit are readily available and the exploit requires no specialized knowledge about the system.
- **Medium** - Tools for the exploit available but have to be modified. The exploit requires some specialized knowledge about the system.
- **Low** - Custom tools must be created for the exploit. In-depth knowledge of the system is required to successfully perform the exploit.

Impact - What effect the vulnerability will have on the system if exploited:

- **High** - Administrator level access and arbitrary code execution or disclosure of sensitive information (private keys, personal information).
- **Medium** - User level access with no disclosure of sensitive information.
- **Low** - No disclosure of sensitive information. Failure to follow recommended best practices that does not result in an immediately visible exploit.

Based on the combination of the parameters the overall risk rating is assigned to the vulnerability.

Discovered vulnerabilities

Low risk - External debugger attacks

Description:

Electron provides external debugging and tracing capabilities even on production builds via the Chrome DevTools Protocol. The DevTools protocol provides essentially the same access as the internal DevTools window to external processes and remote debuggers. By default the debugging interface is disabled, but it can be enabled with an “--inspect” command line argument .

External malware can modify a link that is used to execute the app and add the argument to enable remote debugging. Then the malware can use the DevTools protocol to collect a trace of a user’s interaction with the app and to dump private keys from the wallet.

Similarly, a piece of malware can tamper with the environment to disable other wallet security features like certificate check.

Recommendation:

Unwanted console arguments and environment variables can be handled by the main node process. If any suspicious arguments are detected, the app can notify the user about potential tampering.

Low risk - Man-In-The-Middle attacks

Description:

Man-In-The-Middle (MiTM) attacks are executed by proxying requests through a packet sniffer that can read web traffic and tamper with requests or responses. All of the wallet requests are performed using ssl encryption with certificate verification provided by node and electron by default.

However, a piece of malware or a malicious user can tamper with the environment and bypass certificate verification (e.g. by setting

NODE_TLS_REJECT_UNAUTHORIZED environment variable). Alternatively, an attacker may use a certificate signed by a trusted authority to pass validation.

Exploit example:

Wallet traffic does not contain any sensitive information besides transaction details. So simply capturing traffic is not dangerous. Similarly, tampering with wallet requests is pointless as transactions contain a signature which will be checked by the blockchain node.

The only tactic for an attacker would be to tamper with the server responses, which would allow the attacker to modify transaction values and balances that are displayed in the wallet. For example, this can be used in combination with social engineering to prompt users to send coins multiple times by making it look as if the transaction failed.

Recommendation:

Ideally, certificate pinning should be implemented for communication with blockchain nodes. If the certificate provided by the server does not match an expected certificate that is known by the wallet, the connection should be dropped.

This will eliminate any possibility for MiTM attacks. As users can connect to custom own or third-party nodes it may prove difficult to provide more strict certificate checks unless a user adds a trusted certificate into the wallet configuration. At the bare minimum the wallet should be able to detect unsafe environment conditions which may allow MiTM attacks.

Low risk - A trusted node is used without verification

Description:

The wallet is a lightweight blockchain client which does not act as a full node. Instead it connects to a single trusted node which acts as an intermediary between the wallet and the blockchain. If the trusted node is compromised, the wallet will be unable to detect any misbehaviour.

This has similar opportunities for exploitation as the MiTM attack (balance forgery and fake transaction results).

Additionally, an attacker can reroute traffic from the wallet to a fake node. For example, if the connection is made using a domain name, the attacker can change DNS records to point to a malicious node.

Recommendation:

First of all, to ensure that the traffic from the wallet reaches the required node, certificate pinning should be implemented in the wallet. This way, if the attacker reroutes requests to a fraudulent node, the wallet will be able to detect the change and block compromised requests.

As for the trusted connection, unfortunately, there is no way to protect the wallet in case an attacker gains access to the public node. There is an option to use a custom blockchain node connection in the wallet. Users should be encouraged to use their own blockchain nodes or a list of several trusted public nodes can be provided, in case the default node is compromised.

Vulnerabilities fixed during the audit

FIXED - Medium risk - Code injection attack in smart contract deployment

Description:

The smart contract deployment functionality allows users to add custom smart contract code and to deploy it to the blockchain. The smart contract is arbitrary and provided by the user. Once the contract is deployed, however, the source code that was entered by the user is displayed as raw HTML in the “CodeStorage” component using dangerouslySetInnerHTML. This opens a possibility for code injection using a basic XSS attack.

A malicious smart contract with injected code can be posted online as an example or a template. The smart contact can remain functional and the attack can be hidden from the user entirely.

Exploit example:

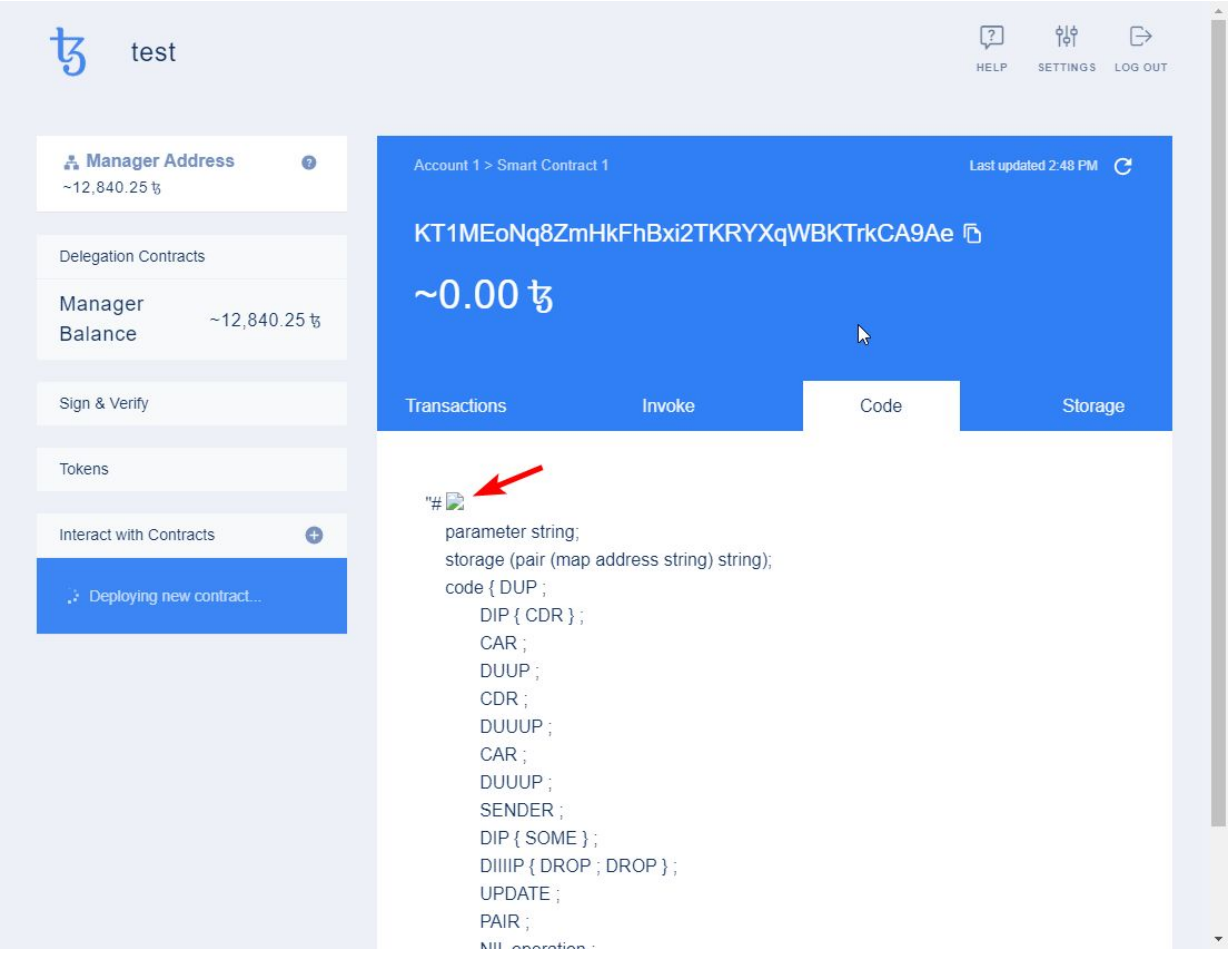
The contract deployment dialog contains the field “Paste in your smart contract code”

The following code is based on an example smart contract:

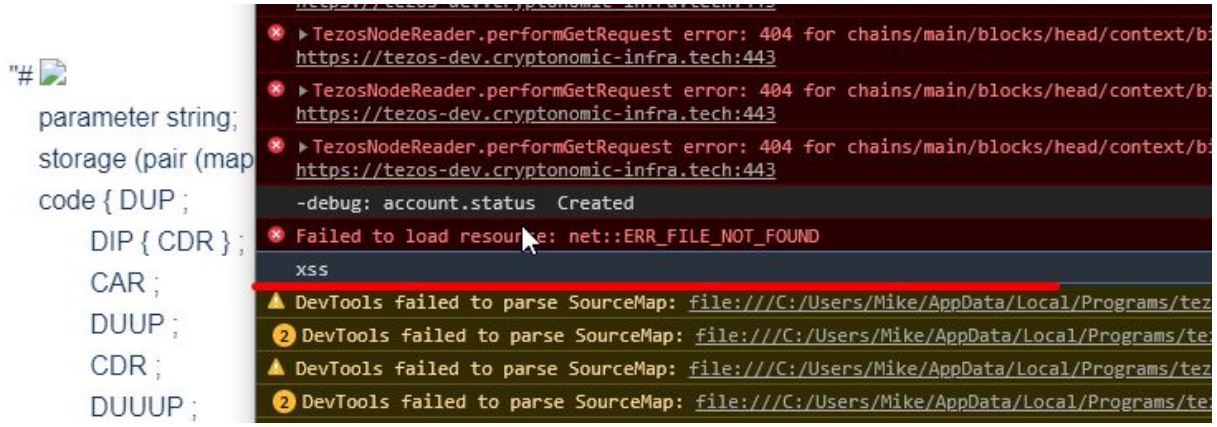
```
# <IMG SRC=x onerror=console.log('xss')>
    parameter string;
    storage (pair (map address string) string);
    code { DUP ;
           DIP { CDR } ;
           CAR ;
           DUUP ;
           CDR ;
           DUUUP ;
           CAR ;
           DUUUP ;
           SENDER ;
           DIP { SOME } ;
           DIIIIP { DROP ; DROP } ;
           UPDATE ;
           PAIR ;
           NIL operation ;
```

```
        PAIR };
```

The first line contains a common XSS injection that logs the string “xss” to the console. The input in the field is passed into a “code” tab on the smart contract page, which displays the malicious code and executes it. As the code is rendered, the injected img element is visible in the page:



And the malicious script is executed:



Recommendation:

The issue is relatively easy to fix by simply sanitizing the input before rendering it and avoiding usage of “dangerouslySetInnerHTML”.

FIXED - Medium risk - Unencrypted private key stored in-memory

Description:

Wallet keys are decrypted once when the wallet is open and the keys are stored in the application memory. The keys can be targeted by malware or leaked with memory dumps. Some attacks that can target private keys through javascript are mentioned in the report. However, malware can also target the memory directly by debugging the electron process and dumping text data.

Recommendation:

The private keys should be encrypted at all times and decrypted only when it is necessary to sign data.

The fix should be fairly easy to implement as the UI already requests the wallet password for every wallet action. Currently, the input simply checks the password and uses unencrypted keys, but it can be modified to decrypt the keys instead.

FIXED - Low risk - Electron debug console attacks

Description:

The Electron (Chrome DevTools) debug console is enabled by default. The console can be used to inspect the internal state of the application. This can be exploited by inciting victims to run exploits in the debug console using common social engineering tactics (e.g. a script that promises free coins but steals private keys instead).

Recommendation:

The debug console can be hidden from users by overriding the key shortcut that opens the console (Ctrl - Shift - I). This will prevent users from running potentially dangerous code in the app.

Appendix B. Methodologies description

In-depth code review

Electron security checks

Electron provides a robust platform that allows to display web applications in a native environment. Unlike in a web environment, applications in Electron are not sandboxed and can interact with the native system. Therefore, it is important to carefully handle content from untrusted sources in order to prevent remote code execution and XSS-injections.

Our security checklist for electron apps is based on the Electron security recommendations:

- Use SSL for loading content and external communication
- Do not use Node.js integration for displaying remote content
- Use context isolation for displaying remote content
- Manually handle session permission requests from remote content
- Do not disable the same-origin policy (webSecurity)
- Use a content security policy for remote content
- Do not enable experimental Chrome features
- Do not use enableBlinkFeatures
- Prevent creation of webViews with possibly insecure options
- Disable or limit navigation
- Disable or limit creation of new windows
- Do not use openExternal with untrusted content
- Disable or filter the remote module
- Use a current version of Electron

React cross-site scripting checks

React.js actually handles most conventional cross-site scripting (XSS) attacks by sanitizing input out of the box. However, in some cases, untrusted content may be rendered without any pre-processing. This may lead to malicious code being

injected into a page which may result in loss of sensitive data or privilege escalation.

There are several cases in which React renders content directly:

- When React components are created directly from user-supplied objects
- When links or other HTML tags with user-provided attributes (e.g. href) are rendered
- When the dangerouslySetInnerHTML prop of an element is explicitly set
- When user-supplied strings are executed using eval()

Analysis of the app in regards to the host network

As a decentralized application, the Tezori wallet depends heavily on its interaction with the underlying blockchain network. Therefore, an analysis of the app in regards to the host network was performed to ensure that external blockchain conditions such as forks, protocol updates or potential attacks on the network cannot impact the application.

The analysis was performed as a part of the manual code review. As the wallet connects to a single trusted node, it will always reflect the state of the blockchain as seen by the node. The wallet should be able to handle chain reorganization and other network-wide events in the same way as the node handles them.