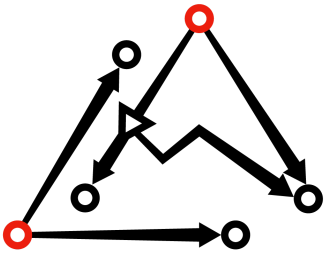

Least Authority

PRIVACY MATTERS

Taquito

Security Audit Report

Tezos Foundation

Final Report Version: 19 June 2020

Table of Contents

Overview

Background

Tezos Foundation has requested that Least Authority perform a security audit of Taquito, a TypeScript library suite for development on the Tezos blockchain.

Project Dates

- **April 15 - April 29:** Code review (*Completed*)
- **May 1:** Delivery of Initial Audit Report (*Completed*)
- **June 15 - 18:** Verification (*Completed*)
- **June 19:** Delivery of Final Audit Report (*Completed*)

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Emery Rose Hall, Security Researcher and Engineer
- Phoebe Jenkins, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of Taquito followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:

- Taquito v6.1.1-beta.0: https://github.com/ecadlabs/taquito/releases/tag/6.1.1-beta.0
    - Taquito High Level Functions: https://github.com/ecadlabs/taquito/tree/master/packages/taquito
    - Taquito RPC: https://github.com/ecadlabs/taquito/tree/master/packages/taquito-rpc
    - Taquito Local Forging: https://github.com/ecadlabs/taquito/tree/master/packages/taquito-local-forging
    - Taquito Michelson Encoder: https://github.com/ecadlabs/taquito/tree/master/packages/taquito-michelson-encoder
    - Taquito Signer: https://github.com/ecadlabs/taquito/tree/master/packages/taquito-signer
    - Taquito Remote Signer: https://github.com/ecadlabs/taquito/tree/master/packages/taquito-remote-signer
    - Taquito Utils: https://github.com/ecadlabs/taquito/tree/master/packages/taquito-utils

However, Taquito Tezbridge Signer, Taquito React Components, and third party vendor code is considered out of scope.

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

`5c113668a9e479d0ebdcd2d01e0a3ef8ad4e6011`

For the verification, we examined the Git revision:

`0f9b6f7fd13b4c97da9bd6fa5afa811f83873b35`

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

Supporting Documentation

The following documentation was available to the review team:

- Taquito High Level Functions: https://tezostaquito.io/typedoc/

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities that currently exist in the code;
- Adversarial actions and other attacks on the network;
- Exposure of any critical information during user interactions with the protocol and external libraries;
- Protection against malicious attacks and other methods of exploitation;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team found Taquito and the related in-scope packages to be logically structured and easy to evaluate. Generally, there is adequate test coverage with only some exceptions (Issue C) and the coding style follows widely accepted idioms and best practices for TypeScript. Our team also found the test coverage in local-forger and michelson-encoder to be excellent.

However, high level, contextual documentation was greatly lacking. Our team strongly suggests the creation of documentation as a security best practice, as it allows for checking the correctness of the implementation and permits new contributors and reviewers to understand the entire system more easily and efficiently (Suggestion 1). Additionally, the practice of writing and updating documentation can help the development team reflect on security practices and potential threats as the system changes over time.

The scope of the audit was sufficient for reviewing Taquito and the related packages as a library for development on the Tezos blockchain. Though local-forging is robust in its operation, the opportunity to look further into the forging process as a whole could be valuable for ensuring the security of this interaction.

Although we did identify several issues worth noting, most were of low severity, the code is in good shape and the packages in scope did not present functionality that deals directly with user secrets or other highly sensitive information. As a result, we commend the Taquito team for adhering to programming best practices and therefore reducing the overall security risk.

Specific 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	VERIFICATION
Issue A: [utils] hexNonce Function Uses An Insecure Random Number Generator	Completed	Resolved
Issue B: [remote-signer] Signature Not Validated Upon Receipt of Response	Completed	Resolved
Issue C: [remote-signer] Missing Adequate Test Coverage	Completed	Resolved
Issue D: [all] NPM Audit Found 47,000+ Known Vulnerabilities in Dependencies	Completed	Resolved
Issue E: [utils] Michelson Parsing Functions are Insecure, Untested, and Undocumented	Planned	Unresolved
Suggestion 1: [all] Human-Readable High Level Documentation is Absent	In Progress	Partially Resolved
Suggestion 2: [remote-signer] Request Creation Does Not Force the Use of TLS	Planned	Unresolved
Suggestion 3: [local-forging] Reject Invalid Inputs When Forging	In Progress	Unresolved
Suggestion 4: Extend Taquito's Linter Rules	Planned	Unresolved

Issue A: [utils] hexNonce Function Uses an Insecure Random Number Generator

Location

https://github.com/ecadlabs/taquito/blob/master/packages/taquito-utils/src/taquito-utils.ts#L134-L141

Synopsis

The utility function `hexNonce` uses JavaScript's built-in `Math.random()`, which is not a secure source of entropy - a widely known design property of most JavaScript implementations since the 1990's.

Impact

Low/Unknown. Nonces are not generally required to be cryptographically secure, only pseudorandom, due to the simple requirement that they should not be reused with the same key. The purpose of this utility function is not immediately clear, as it is not found elsewhere in the packages that are in scope for this audit.

Remediation

Change the function to use the browser's native cryptography implementation in order to ensure that a secure source of entropy is used.

```
const toHexString = bytes => {

  bytes.reduce((str, byte) => str + byte.toString(16)
```

```
        .padStart(2, '0'), '');
};

export const hexNonce = (length: number): string => {
  return toHexString(crypto.getRandomValues(new Uint8Array(length)));
};
```

Status

The development team has removed the addressed `hexNonce` function.

Verification

Resolved.

Issue B: [remote-signer] Signature Not Validated Upon Receipt of Response

Location

https://github.com/ecadlabs/taquito/blob/master/packages/taquito-remote-signer/src/taquito-remote-signer.ts

Synopsis

The client requests the server to sign a message specifying the public portion of the keypair to be used. However, when the server responds with the signature, the client does not verify it.

Impact

Low. If the signing server is compromised or otherwise contains a bug, it can return invalid signatures. This may cause interruptions in the proper function of the client when other components or software makes an attempt to verify the signature.

Preconditions

Compromise of the signing server.

Remediation

Given that the client knows the public key hash that corresponds to the private key it wishes the server to sign the message with, the server should respond with the fully qualified public key. Furthermore, the client should check it against the public key hash that it knows and use the public key to verify that the signature returned from the server is valid.

Status

The development team implemented the remediation suggestion by adding a function that takes the message bytes and signature from the remote signer and validates it against the public key. Additionally, the function validates that the public key returned from the remote signer and the public key used to initialize the public signer are identical.

Verification

Resolved.

Issue C: [remote-signer] Missing Adequate Test Coverage

Location

https://github.com/ecadlabs/taquito/blob/master/packages/taquito-remote-signer/src/taquito-remote-signer.ts

Synopsis

There are no automated tests for the remote signer package, which may lead to future bugs and security issues.

Impact

Low/Unknown. The lack of a testing baseline allows future changes to the package to introduce new bugs and potential security vulnerabilities.

Remediation

Author a complete test suite to cover the existing remote-signer code end-to-end.

Status

The unit test coverage has been increased up to an acceptable ratio in the aforementioned package.

Verification

Resolved.

Issue D: [all] NPM Audit Found 47,000+ Known Vulnerabilities in Dependency Graph

Synopsis

The dependency versions listed in the existing package.json were automatically scanned by NPM, revealing a very large number of known security issues. Due to the large number of issues, we did not inspect each one individually and cannot speak to the effect these issues may or may not have on the project.

Impact

High/Unknown. Since we did not inspect all 47,000 issues, we are not able to provide an impact assessment. However, it is not feasible to maintain a secure codebase if dependencies are stagnant and outdated, allowing for such a high number of potential issues.

Remediation

Performing an automatic upgrade using `NPM audit fix` and a manual upgrade for the remaining packages that cannot be upgraded automatically was found to have no adverse impact on the existing test suite.

Status

The vast majority of previously reported known vulnerabilities in dependencies have been resolved. The development team has stated that they will proactively monitor and update dependencies moving forward.

Verification

Resolved.

Issue E: [utils] Michelson Parsing Functions are Insecure, Untested and Undocumented

Location

https://github.com/ecadlabs/taquito/blob/master/packages/taquito-utils/src/taquito-utils.ts#L157-L342

Synopsis

The functions `sexp2mic` and `ml2mic` in the `taquito-utils` package are security concerns due to their handling of important user inputs. In addition, they cannot be verified due to having no testing or documentation of their intended behaviors. If these parsers are to convert Michelson expressions into the official JSON representation, they currently handle many inputs incorrectly, such as proper representation of integers, strings, or annotation. They also remove escaping on escaped inputs, which appears to be unintended.

Additionally, these parsers do not attempt validate inputs and will not provide an error on basic syntactic issues such as improperly nested parentheses or brackets. As a result, the functions cannot be used safely and can be used to generate invalid and mangled outputs.

Furthermore, the construction and intended usage of these functions is confusing. It is unclear if `sexp2mic` and `ml2mic` are intended to have different behaviors or operate under different circumstances. This is further complicated by how `ml2mic` will at times recurse into `sexp2mic` instead of itself.

Impact

High/Unknown. It is unclear how these functions are expected to be used or how their results are expected to be handled and, as a result, it is difficult to make accurate conclusions about the attack surface. In cases where these functions are handling untrusted user input that has not been properly sanitized, it is possible to imagine a malicious user exploiting them in order to generate Michelson JSON that has radically different behavior from the input JSON string.

Remediation

At a minimum, the intended behavior of the functions should be properly documented. They should also have comprehensive test coverage and follow the model of the local-forging integration tests, which ideally compare against official values obtained via RPC.

As a suggestion, since writing secure and efficient parsers is challenging, it may be valuable to use a parser generator or parser combinator library such as PEG.js with a typescript plugin, or tsPEG. This will prevent many common mistakes, and result in code that is easier to write, read, maintain, and audit.

Status

The development team has stated that they intend to remove the specified functions and replace them with the more complete and robust `michel-codec` package. While this has not been implemented at the time of the verification review due to the effort required in constructing `michel-codec`, the intended plan should resolve the issue.

Verification

Unresolved.

Suggestions

Suggestion 1: [all] Human-Readable High Level Documentation is Absent

Synopsis

There is complete TypeDoc style documentation automatically generated from the source code. However, there is a severe lack of high level, contextual documentation around how various components work together within the wider system. As a result, this made reviewing the code more difficult as the information had to be inferred from reading the source code several times.

Mitigation

Creating more general documentation for each package, including diagrams and use case examples, would greatly improve onboarding for both developers and reviewers.

Status

The development team has indicated that they intend to implement the suggestion and the development of improved documentation for both high-level package usage and low-level API details is in progress. Currently, updated high-level documentation has been provided for estimation functionality here. This documentation is helpful as it provides a high-level overview of the API and examples of expected usage.

Verification

Partially Resolved.

Suggestion 2: [remote-signer] Request Creation Does Not Force Using TLS

Location

https://github.com/ecadlabs/taquito/blob/master/packages/taquito-remote-signer/src/taquito-remote-signer.ts

Synopsis

While resolving Issue B may reduce the potential impact of the issue, since it would eliminate the possibility of receiving an invalid signature, it could technically be sufficient when dealing with an untrusted server. However, since private information may be transmitted to the server, we consider explicitly requiring the use of TLS to be a best practice when communicating with a remote server.

Mitigation

Ensure that the options given to the request object include `https:` as the protocol.

Status

The development team has acknowledged the benefits of using TLS and have responded that they intend to further investigate forcing TLS on both backend and frontend levels. Furthermore, they have stated their intention to add documentation and a warning to the Remote Signer documentation explaining to users the intended context.

Verification

Unresolved.

Suggestion 3: [local-forging] Reject Invalid Inputs When Forging

Location

https://github.com/ecadlabs/taquito/tree/master/packages/taquito-local-forging

Synopsis

It is possible to provide inputs that can generate invalid outputs from the local-forger package's `localForger.forge()` method. Although there may be more instances of this, our team identified two cases:

- Leaving a field absent that a schema expects, and
- Encoding a `prim` that has an invalid op.

While there does not appear to be potential for vulnerabilities, it could possibly generate scenarios that are confusing and difficult to debug.

Mitigation

Both of these examples can be fixed by checking that the result from the object lookup is not undefined.

Status

The development team is currently researching a best approach for addressing this suggestion by potentially leveraging the new `michel-codec` package. While this will help, it is not a complete solution. As a result, they have stated their intention to investigate further in order to identify how to defensively address this issue, in order to benefit both security and developer experience.

Verification

Unresolved.

Suggestion 4: Extend Taquito's Linter Rules

Synopsis

Our team identified some issues that could be easily detected and fixed by a linter such as, but not limited to, unused declarations and shadowed variables.

Mitigation

Using a more opinionated linter configuration can improve Taquito's code readability, maintainability, and overall quality by enforcing some best practices.

Status

The development team acknowledges the need for improved Linter rules and have stated their intention to improve and enforce linting rules through their continuous integration jobs.

Verification

Unresolved.

Recommendations

We recommend that the remaining unresolved *Issues* and *Suggestions* stated above are addressed as soon as possible and followed up with an additional verification or subsequent audit by the auditing team.

The addition of high level contextual documentation is strongly encouraged, along with continuing to structure the code in a way that adheres to best practices and reduces the overall security risks.

In addition, we recommend a follow up security audit of the React components that were out of scope for this review. Given that end users will interact with the interface code, there is a potential for increased security risk due to user input and interaction. The forging process may also be another area for further investigation for potential issues.

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 C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. 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. Although we are a small team, we believe that 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 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.