Least Authority

PRIVACY MATTERS

Beacon SDK

Security Audit Report

Tezos Foundation

Updated Final Report Version: 8 October 2020

Table of Contents

This audit makes no statements or warranties and is for discussion purposes only.

Overview

Background

Tezos Foundation has requested that Least Authority perform a security audit of the Beacon SDK, which allows developers of decentralized applications and wallets on Tezos to implement the wallet interaction standard tzip-10. The Beacon SDK simplifies and abstracts the communication between decentralized applications and wallets over different transport layers.

Project Dates

- **July 16 - August 4**: Code review (*Completed*)
- **August 6**: Delivery of Initial Audit Report (*Completed*)
- **September 14 - 15:** Verification (*Completed*)
- **September 17:** Delivery of Final Audit Report (*Completed*)
- **October 8:** Delivery of Updated Final Audit Report (*Completed*)

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Phoebe Jenkins, Security Researcher and Engineer
- Jan Winkelmann, Cryptography Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Beacon SDK 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:

- Beacon SDK: https://github.com/airgap-it/beacon-sdk

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

```
4002c9b7b5906ec732c7252e5a27fab8b8782947
```

For the verification, we examined the Git revision:

```
c51c040cd9d5e6b45dd68b8d63dee3c39d0c9294
```

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:

- WalletBeacon.io: https://www.walletbeacon.io/
- Tzip-10: https://gitlab.com/AndreasGassmann/tzip/-/blob/master/proposals/tzip-10/tzip-10.md

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;

This audit makes no statements or warranties and is for discussion purposes only.

- Vulnerabilities that currently exist in the code;
- Adversarial actions and other attacks on the interactions;
- Potential exposure of any critical information during communications;
- 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

The Beacon SDK is an implementation of the tzip-10 standard, which describes an interaction between decentralized applications and wallets on Tezos. Our team comprehensively reviewed this interaction, which aims to simplify and abstract the communication between decentralized applications and wallets over different transport layers.

Code Quality + Documentation

Our team found that the code was well organized and appropriately formatted, adhering to development best practices. Furthermore, variables and functions are properly named and functions are written with a level of abstraction that makes the code reusable. Overall, we observed low cyclomatic complexity methods in addition to good compiler and linter configurations that help reduce the risk of potential vulnerabilities in the code to go unnoticed by facilitating early bug detection, less error prone code, more readable code, and reusable components.

However, we found there to be minimal code comments outlining the purpose of different classes and methods, in addition to potential failure modes, which made the code more difficult to comprehend by a reviewer. We recommend that additional comments be included to help define these areas, allowing developers to more easily understand and efficiently implement the code (Suggestion 3). Furthermore, while some tests are present in the code, others were disabled resulting in challenges to understand certain areas of the structure and the intended execution flow. Ideally, more test coverage would be highly desirable, particularly around complex and security-critical components, such as end-to-end integration tests for Matrix communication (Suggestion 4) and focused unit testing around cryptographic methods (Issue E).

We found the Beacon-SDK documentation available for end-users of the Beacon SDK to be very comprehensive and useful. In addition, all examples in the documentation are up to date and working with the new release. In particular, our team appreciated the diagrams showing how communication works between the different components. We recommend adding more API documentation around the user facing classes and their methods in the Beacon SDK, and their possible failure modes, which would help users understand how to properly and securely utilize the Beacon SDK (Suggestion 3).

System Design

The Beacon SDK development team's use of the secure and decentralized Matrix protocol, as well as the modern and heavily reviewed cryptography library, libsodium, demonstrates that security is a priority and has been strongly considered throughout the system design. However, we identified a security critical issue, which results from the Beacon SDK sending all communication through a single Matrix server, creating a single point of failure for all Beacon applications. We suggest having available additional redundant servers hosted by the development team and allowing users of the Beacon SDK to supply

This audit makes no statements or warranties and is for discussion purposes only.

additional Matrix servers that they wish to use for redundancy in order to mitigate the potential risk of a failure with the server (Issue A).

Further Investigation

Since the Beacon SDK project utilizes the Matrix protocol and relies on certain functionality regarding federation and redundancy being appropriately set up, we recommend that this area be further investigated for potential security vulnerabilities. Although we reviewed the Matrix client, a review of the Matrix server configuration would help to ensure that these behaviors work as intended, which could potentially impact the security of the Beacon SDK.

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
Issue A: Potential Single Point of Failure with Hardcoded Matrix Servers	Resolved
Issue B: Provided Tezos Networks Could Run Over Unsecured HTTP	Partially Resolved
Issue C: Avoid Conversion Operations on Generated Keys	Unresolved
Issue D: Use of a Cryptographically Insecure RNG	Resolved
Issue E: Low Test Coverage on Security Critical Packages	Resolved
Suggestion 1: Custom Matrix Integration uses Outdated API	Unresolved
Suggestion 2: Automated Dependency Security Scanning	Resolved
Suggestion 3: Improve Documentation	Resolved
Suggestion 4: Add End-to-End Integration Tests	Unresolved

Issue A: Potential Single Point of Failure with Hardcoded Matrix Servers

Location

beacon-sdk/src/transports/P2PCommunicationClient.ts

Synopsis

The current implementation relies on a single, hard coded Matrix server to mediate communication between all applications using the functionality of Beacon SDK. This represents a single point of failure and, as a result, if this server goes offline (e.g. to perform routine maintenance), this will result in a loss of functionality for all Beacon-enabled applications.

In addition, relying on a single server prevents utilizing the redundant messaging functionality of Beacon, leaving this behavior effectively untested.

This audit makes no statements or warranties and is for discussion purposes only.

Impact

If the hard coded Matrix server goes offline, all applications using this library would be unable to communicate.

Mitigation

An initial approach to mitigate this issue would be to add additional redundant servers that are hosted by the Beacon SDK development team. While this will make the operation much less fragile, it places an increased amount of responsibility on the development team to ensure that multiple servers are always up, running properly, and well maintained.

In addition to this approach, it would be useful to allow users of the Beacon SDK to supply additional Matrix servers that they wish to use for redundancy or alternative servers they wish to maintain themselves. This would assist in sharing or shifting the responsibility of server uptime to multiple parties and that decentralization will result in a more stable and secure operation.

Status

The Papers team has issued a commit addressing this issue. As a result, both the `WalletClient` and `DAppClient` are configurable so that users are able to provide custom Matrix server URLs. If a custom URL is not provided, the Beacon SDK will use the predefined default Matrix server URL.

Verification

Resolved.

Issue B: Provided Tezos Networks Could Run Over Unsecured HTTP

Location

src/types/beacon/Network.ts

Synopsis

When a user provides a custom configuration for connecting to a Tezos node, this connection can be specified to take place over HTTP. Since this communication is unencrypted, it could be read over the network and could be subject to man-in-the-middle attacks without authentication.

Impact

In the most severe cases with HTTP, traffic could be intercepted by a third party, providing invalid responses and making the application believe that some actions have taken place when they have not. This is likely to cause applications to end up in undefined states, unless special precautions have been taken to avoid this.

Preconditions

End-user provides a URL using the HTTP protocol, rather than HTTPS.

Mitigation

The following mitigation strategies are possible:

- Ensure that provided URLs do not begin with `http://`; or
- Do not allow users to provide the protocol for URLs, instead forcing HTTPS.

This audit makes no statements or warranties and is for discussion purposes only.

Remediation

Add a unit test for this specific case and ensure failure in the case of a provided URL using the HTTP protocol.

Status

The Papers team has issued a commit partially addressing this issue.

The Beacon SDK documentation has been updated to recommend that prospective developers should check and warn users whenever an insecure Tezos node RPC URL is being used.

However, the Papers team has chosen not to enforce the use of HTTPS, given that developers may want to enable local testing of applications using Beacon SDK, utilizing the HTTP protocol, and setting up a secure connection would require considerable effort. As a result, we recommend that the Papers team implement warnings in the User Interface (UI) when an insecure Tezos node RPC URL is being used, in order to make developers and end users aware of potentially insecure communication.

Verification

Partially Resolved.

Issue C: Avoid Conversion Operations on Generated Keys

Location

beacon-sdk/src/utils/crypto.ts

Synopsis

Beacon SDK's encrypted messages system involves generating a cryptographic key of one type (ed25519), and then converting it to a different type (curve25519) under certain situations. The conversion between ed25519 and curve25519 keys is not considered to be well defined and the libsodium documentation itself advises against doing this:
https://doc.libsodium.org/advanced/ed25519-curve25519

Impact

While the impact is unclear, the behavior does not appear to be considered or well-defined. As a result, the consequences may potentially range from signatures forged by an attacker to an attacker being able to compute a shared Diffie-Hellman secret they do not have a private key for, but observed messages signatures.

Mitigation

Rather than converting from ed25519 keys to curve25519 keys when encryption needs to be performed, a curve25519 key can be generated instead, alongside the initial ed25519 key with the `crypto_kx_seed_keypair` function in libsodium. Having a separate dedicated key for signing will avoid any potential issues that could arise from this conversion.

Remediation

Ensuring that the `crypto_sign_ed25519_sk_to_curve25519` and `crypto_sign_ed25519_pk_to_curve25519` keys are not present in the codebase and using a specifically generated key pair instead.

This audit makes no statements or warranties and is for discussion purposes only.

Status

Due to the added complexity of using two seperate keypairs, the Papers team has chosen not to implement the mitigation of this issue and decided to keep the current implementation instead. We suggest that the Papers team reconsider implementing the mitigation or remediation in the future.

Verification

Unresolved.

Issue D: Use of a Cryptographically Insecure RNG

Location

beacon-sdk/src/utils/generate-uuid.ts

Synopsis

The Globally Unique Identifier (GUID) uniqueness relies on the underlying Random Number Generator (RNG). `generateGUID()` generates GUIDs using the browser cryptographically strong RNG. However, it uses JavaScript's `Math.random()` when this functionality is unavailable. `Math.random()` is not a secure source of entropy and its usage should be avoided in such cases.

Impact

Implementation of `Math.random()` does not have sufficient entropy and leaves an opportunity for collisions. This is important because among other things, the GUID is used to create the initial key pairs.

Remediation

Utilize crypto API to generate the seeds instead of using `Math.random()`, or replace the `generate-uuid.ts` code with a call to a well-maintained, cross-platform, and a cryptographically strong RNG utilizing library to generate GUID/seed (i.e. uuid.js).

Status

The Papers team has issued a commit addressing this issue. As a result, the Beacon SDK's `generateGUID()` function now always uses libsoduim's `randombytes_buf()`, which is considered a secure RNG.

Verification

Resolved.

Issue E: Low Test Coverage on Security Critical Packages

Location

All locations (client, transports, managers, matrix-client, utils/crypto.ts)

Synopsis

The automated testing coverage is insufficient and may lead to future bugs and security issues. Certain areas of complexity, such as the cryptographic utility functions, would benefit from having lower-level unit tests to ensure correct the behavior of individual functions.

Impact

The low overall testing coverage may lead to mergers of faulty code changes and potential security vulnerabilities.

Mitigation

We recommend following the npm-generated coverage reports and ensuring that all present classes and methods have associated tests. The test suite should be incorporated into a continuous integration system such as Travis CI to allow early detection of bugs upon changes made to the Beacon SDK's code base.

Remediation

Use of `npm test`'s coverage reports is helpful in determining what code is not being covered in the existing test suite and helps determine if any existing coverage has become inadequate.

Status

The Papers team has issued a commit, which significantly increases the overall unit test coverage ratio. The unit tests added cover critical security hotspots. We recommend that the Papers team maintain a good unit test coverage ratio as they implement new features and add more code.

Verification

Resolved.

Suggestions

Suggestion 1: Custom Matrix Integration uses Outdated API

Location

beacon-node/docker/crypto_auth_provider.py

beacon-node/blob/master/docker/crypto_auth_provider.py

Synopsis

While this area is out of scope for this review, the custom integration used in `beacon-node` makes use of an outdated version of the API, which prevents upgrading Matrix. The inability to upgrade Matrix does not allow for taking advantage of new features and security updates as they are released.

Mitigation

One approach is to change all references to the `.hs` property of the `AccountManager` to `._hs`. Since this functionality is being deprecated, further updates may be needed in the future to keep the plugin up-to-date. Alternatively, since it seems the call to `.hs` is unnecessary, it could be deleted to remove the dependence on deprecated functionality

Status

The Papers team has acknowledged the suggestions and has stated their intent to implement a mitigation in the future.

Verification

Unresolved.

Suggestion 2: Automated Dependency Security Scanning

Location

All locations

This audit makes no statements or warranties and is for discussion purposes only.

Synopsis

NPM-Auditing the Beacon SDK package dependencies found 394 vulnerabilities (389 low, 5 high), all of which are devDependencies. Running *npm audit fix* resolves all of these issues by upgrading to compatible new releases.

Mitigation

Our team was unable to verify if automated package dependency auditing is part of the project's CI pipeline flow. As a result, we recommend including it in order to proactively monitor and prevent having outdated and vulnerable dependencies.

Status

The Papers team has issued a commit, which adds an integrated automatic dependency auditing tool "`npm audit`" in their continuous integration tool flow. As a result, the usage of reported vulnerable dependencies has been prevented.

Verification

Resolved.

Suggestion 3: Improve Documentation

Location

Project Documentation

Synopsis

Our team found minimal comments outlining the purpose of different classes and methods, in addition to potential failure modes, which made the code more difficult to comprehend.

We also found that the project would benefit from API documentation around the user facing classes and their methods in the Beacon SDK, and their possible failure modes, which would help users better understand how to properly and securely utilize the Beacon SDK.

Mitigation

We recommend that additional comments and high-level documentation be included to help define these areas, allowing developers to more easily and efficiently understand and implement the code. Preferably, code comments should follow TSDoc standard to allow automatic generation of code documentation.

Additional suggestion related to Issue A: We also recommend further extending the documentation with the Beacon SDK Matrix servers setup and management guides.

Status

The Papers team issued a commit, which adds typedoc compatible inline code comments that can be rendered into HTML documentation. In addition, the Papers team issued a commit that improves the overall documentation of the Beacon SDK. In particular, they have included examples, frequently asked questions, and security notices, which will facilitate easier onboarding and comprehension of the system.

Verification

Resolved.

Suggestion 4: Add End-to-End Integration Tests

Location

All Locations.

Synopsis

Certain areas of complexity, such as how clients utilize Matrix communication flows and Tezos interactions, would benefit from having higher-level, end-to-end integration tests, particularly those that simulate potential failure modes from these external services.

Mitigation

For end-to-end integration tests, utilizing tools such as Selenium can simulate user interactions with the UI components. These can be used in addition to mocking browser-specific components, such as `localStorage` to provide coverage of complex areas, simulate how the application operates under specific failure conditions and edge cases within the browser, or in communication with the Matrix or Tezos servers.

Status

The Beacon SDK team has stated that they do not consider the inclusion of end-to-end integration testing to be a necessity, given that they do not have any critical UI components. As a result, end-to-end integrations tests have not been incorporated into the code base at this time. While our team does not consider this to be a security critical issue, we recommend that a test suite be added to the continuous integration flow as a best practice to help avoid potential code errors.

Verification

Unresolved.

Recommendations

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

We also recommend that an additional review of the Matrix server configuration be conducted, since the Beacon SDK relies on certain functionality being set up properly. Conducting an audit to ensure that these behaviors work as intended will mitigate against the potential for security vulnerabilities that may impact Beacon SDK.

Finally, we commend the Beacon SDK for the measures taken to prioritize the security of the codebase, including following development best practices by properly naming variables and functions, which are written to allow code reusability. Early bug detection, less error prone code, more readable code, and reusable components are also facilitated by good compiler and linter configurations that help reduce the risk of potential vulnerabilities being introduced into the Beacon SDK.

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

This audit makes no statements or warranties and is for discussion purposes only.

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.