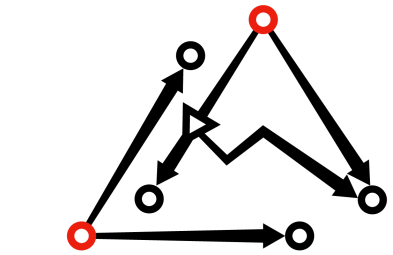

Least Authority

PRIVACY MATTERS

Bi-directional FastBTC

Security Audit Report

Sovryn

Final Audit Report: 02 September 2022

Table of Contents

Overview

Background

Sovryn has requested that Least Authority perform a security audit of Bi-directional FastBTC, which facilitates the transfer of RSK to Bitcoin (rBTC to BTC). Bi-directional FastBTC is an update to FastBTC, which facilitates the transfer of Bitcoin to RSK (BTC to rBTC).

Project Dates

- **June 8 - July 26:** Initial Code Review *(Completed)*
- **July 29:** Delivery of Initial Audit Report *(Completed)*
- **September 2:** Delivery of Final Audit Report *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Nicole Ernst, Security Researcher and Engineer
- Ahmad Jawid Jamiulahmadi, Security Researcher and Engineer
- Rosemary Witchaven, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Bi-directional FastBTC followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

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

- Bi-directional FastBTC:
  https://github.com/DistributedCollective/bidirectional-fastbtc

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

`97d1f3916836260979db9b20ee78c1d5e11bedfd`

For the review, this repository was cloned for use during the audit and for reference in this report:

Bi-directional FastBTC:
https://github.com/LeastAuthority/Sovryn-Bidirectional-Fastbtc-Audit

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

In addition, any dependency and third-party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- README.md:
  https://github.com/DistributedCollective/bidirectional-fastbtc#readme

In addition, this audit report references the following document:

- P.A. Grassi, J.L. Fenton, E.M. Newton, R.A. Perlner, A.R. Regenscheid, W.E. Burr, et al., "Digital Identity Guidelines: Authentication and Lifecycle Management." *NIST Special Publication 800-63B*, 2017, [GFN+17]

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adherence to the specification and best practices;
- Adversarial actions and other attacks on the smart contracts and backend service;
- Potential misuse and gaming of the smart contracts and backend service;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution of the smart contracts and backend service;
- Vulnerabilities in the smart contract code and backend service;
- Protection against malicious attacks and other ways to exploit the smart contracts and backend service;
- 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

Bi-directional FastBTC aims to provide a bridge between the Bitcoin network and the RSK network, an EVM compatible blockchain. The system is composed of two main components. One component is a validator node implementation in typescript where validator nodes validate RSK chain data and sign RSK and BTC batch transactions. The second component is a smart contract suite, which performs bridge operations on the RSK chain.

Our team performed a comprehensive review of Sovryn's Bi-directional FastBTC implementation. We investigated the design of the system and its coded implementation to identify security vulnerabilities and implementation errors and to assess adherence to best practice recommendations.

In addition to reviewing the security of the system and the areas of concern listed above, we reviewed the smart contracts for vulnerabilities that could result in the loss of funds through the impersonation of federators. We verified that reentrancy safeguards are appropriately implemented and checked for any potential overflows or underflows in the implementation of arithmetic operations.

Our team did not identify security vulnerabilities in the design and implementation of the Bi-directional FastBTC system and found that the implementation generally adheres to security best practice recommendations.

System Design

Our team reviewed the design of the Bi-directional FastBTC system and found that security has been taken into consideration as demonstrated by adherence to recommended standards in the use of

cryptography in the validator node implementation, and by appropriate implementation of role-based access controls, freeze, and pause functionalities in the smart contract suite.

Although our team did not identify any security vulnerabilities in the design of the system, we did discover opportunities to improve the overall security of the implementation. In the validator node implementation, we found that buffers are not cleared of secret data appropriately, which could leave this data vulnerable in case of a memory dump. This data could be used to take control of user assets. We recommend that the buffer be cleared after being used (Suggestion 1). We also found that the `config` file, which contains secret data relating to the node participating in the system (including private keys), is encrypted using a user-selected password. However, there are insufficient constraints on the password that the user can select, which could result in the user selecting an insufficiently secure password that is vulnerable to dictionary attacks. We recommend that user-selected passwords be constrained according to NIST best practice recommendations (Suggestion 2).

Code Quality

Our team found that the codebase is generally well organized and adheres to best practices. However, in `multisig.ts`, we found that the length and complexity of the functions implemented in this file could cause confusion to readers and maintainers. We recommend that these functions be refactored.

Tests

The unit tests and integration tests in place provide sufficient test coverage of the implementation. This helps identify implementation errors that could lead to security vulnerabilities and verify that the system behaves as intended.

Documentation

The documentation provided for this security review was generally sufficient in describing the architecture of the system, each of its components, and the interaction between those components.

Code Comments

The Bi-directional FastBTC codebase implements sufficient code comments, which describe the intended behavior of security-critical functions and components. The code comments of the smart contracts component adhere to NatSpec guidelines.

Scope

The scope of this security review was sufficient and included all security-critical components.

Dependencies

Our team did not identify any vulnerabilities in the implementation's use of dependencies. However, we found a vendored dependency that is not the latest version. We recommend using up-to-date dependencies, which include the latest security patches and bug fixes, in accordance with best practice (Suggestion 3).

Specific Suggestions

We list the issues and suggestions found during the review, 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.

SUGGESTIONS	STATUS

Suggestion 1: Clear Buffers Holding Secret Values after Usage	Unresolved
Suggestion 2: Enforce High Entropy Passwords for Encrypting Config Secrets Stored in Filesystems	Unresolved
Suggestion 3: Update Vendored Dependencies	Unresolved

Suggestions

Suggestion 1: Clear Buffers Holding Secret Values after Usage

Location

`/packages/fastbtc-node/src/utils/secrets.ts#L111`

`/packages/fastbtc-node/src/utils/secrets.ts#L123`

Synopsis

It is a recommended practice that buffers holding secret values be zero-filled/cleared when they are no longer in use to prevent unencrypted data leakage in the case of a memory dump.

Mitigation

We recommend that the `buff.fill` function be utilized to clear values held by buffers containing secret values or plaintext messages.

Status

The Sovryn team acknowledged this suggestion and has added the recommendations to their backlog but will not be implementing the mitigation at this time.

Verification

Unresolved.

Suggestion 2: Enforce High Entropy Passwords for Encrypting Config Secrets Stored in Filesystems

Location

`/packages/fastbtc-node/src/utils/secrets.ts#L136-L156`

Synopsis

When setting up a node, users are prompted to enter a password to encrypt `config` files, which contain secrets, including private keys. Currently, the system does not enforce any particular level of password strength, and users may choose insufficiently secure passwords. Insufficiently secure passwords can be vulnerable to dictionary attacks, which would put the secret data in the `config` file at risk of compromise.

Mitigation

We recommend increasing the requirement of password entropy and disallowing passwords that are easy to guess by following the NIST Guidelines noted in [GFN+17], which are industry-standard rules and best

practices for handling passwords when building memorized secret authenticators. We recommend utilizing a password strength estimator, such as zxcvbn.

Status

The Sovryn team acknowledged this suggestion and has added the recommendations to their backlog but will not be implementing the mitigation at this time.

Verification

Unresolved.

Suggestion 3: Update Vendored Dependencies

Location

`/packages/fastbtc-node/vendor`

Synopsis

The vendored dependency “Ataraxia” has not been updated for almost 9 months, and the version used in the implementation is several versions behind the main repository. Although no security issues have been reported in the version used in the implementation, it is a best practice to use vendored dependencies that are up to date with their mainstream repositories.

Mitigation

We recommend that pull changes be committed on the mainstream repository and that the Sovryn team continue to monitor the mainstream repository for critical security fixes. Alternatively, if possible, we recommend that the Sovryn team merge the needed changes directly to the main repository, which would enable the use of automated dependency auditing tools in their CI/CD workflow to detect vulnerable dependencies and aid developers in taking necessary action when needed.

Status

The Sovryn team acknowledged this suggestion and has added the recommendations to their backlog but will not be implementing the mitigation at this time.

Verification

Unresolved.

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 multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, and zero-knowledge protocols. 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. We are an international team that believes 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 include 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 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. As 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 and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and 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 before having 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 and comprehensive 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 Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, 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 a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.