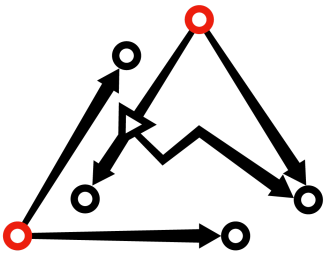

Least Authority

PRIVACY MATTERS

ETH to Chia Bridge

Security Audit Report

Chia Network

Initial Audit Report: 29 June 2023

This Initial Audit Report is intended for internal use and discussion purposes only. We advise against sharing this report beyond trusted team members and recommend that publication take place only after the verification has been completed and the Final Audit Report has been delivered.

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - Code Quality
    - Documentation
    - Scope
  - Specific Issues & Suggestions
    - Issue A: Missing Access Control Modifier
  - Suggestions
    - Suggestion 1: Save Core and Utility Code in One Place and Reference It
    - Suggestion 2: Specify Path When Accessing Chialisp Program
    - Suggestion 3: Create Integration Tests for Smart Contracts
    - Suggestion 4: Improve Code Comments
    - Suggestion 5: Create Bridge Documentation
- About Least Authority
- Our Methodology

Overview

Background

Chia Network has requested that Least Authority perform a security audit of the ETH to Chia Bridge.

Project Dates

- **May 22, 2023 - June 26, 2023:** Initial Code Review *(Completed)*
- **June 29, 2023:** Delivery of Initial Audit Report *(Completed)*
- **TBD:** Verification Review
- **TBD:** Delivery of Final Audit Report

The dates for verification and delivery of the Final Audit Report will be determined upon notification from the Chia Network team that the code is ready for verification.

Review Team

- Nicole Ernst, Security Researcher and Engineer
- Nikos Iliakis, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the ETH to Chia Bridge 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:

- Chia Network Bridge:
  https://github.com/Chia-Network/bridge
- Chia Network Bridge Smart Contracts:
  https://github.com/elkfinance/chia-bridge

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

- Chia Network Bridge: `0e65c1c0568f80b63a466a470e277a7a470b1038`
- Chia Network Bridge Smart Contracts: `96f6a2e41f6b7b035c0b0a093c62105260e7a3b9`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Chia Network Bridge:
  https://github.com/LeastAuthority/Chia-Network-Bridge
- Chia Network Bridge Smart Contracts:
  https://github.com/LeastAuthority/Chia-Network-Bridge-Contracts

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:

- ETH to Chia Bridge technical spec draft (Google Doc) *(shared with Least Authority via email on 21 February 2023)*
- ETH to Chia Bridge summary (Google Doc) *(shared with Least Authority via email on 21 February 2023)*

In addition, this audit report references the following documents:

- Don't repeat yourself (DRY) Principle: https://en.wikipedia.org/wiki/Don%27t_repeat_yourself
- NatSpec Format: https://docs.soliditylang.org/en/v0.8.19/natspec-format.html

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks on the bridge;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) attacks and security exploits that would impact or disrupt the execution of the bridge;
- Vulnerabilities within individual components and whether the interaction between the components is secure;
- Exposure of any critical information during interaction with any external libraries;
- Proper management of encryption and signing keys;
- Protection against malicious attacks and other methods of exploitation;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions and excess authority; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Chia Network ETH to Chia Bridge (bridge) is a custodial bridge managed by a set of semi-trusted validators enabling the transfer of Ethereum assets (ETH, ERC20) to the Chia blockchain. The bridge is composed of a validator node implementation in Chialisp and Ethereum smart contracts that receive user assets for transfer across the bridge. The smart contracts hold the assets until they are returned from the Chia side of the bridge. On the Chia network side, functionality implemented in Python handles the minting and burning of assets. Our team performed a comprehensive review of these components as well as the server command-line interface (CLI). We considered a threat model with an attacker in the role of a user of the bridge.

Our team did not identify critical security vulnerabilities in the design and implementation of the validator node. However, we identified an instance of a missing access control modifier in the smart contracts (Issue A). We have also identified areas within the code where enhancements can be made to improve the system's security. These areas include code quality, testing, and documentation, all of which play a crucial role in enhancing the overall security of the system.

Code Quality

Our team performed a manual review of each of the components and found that the codebases are well-organized and in adherence to best practices. There are sufficient comments explaining parts of the code. However, we recommend making the comments more detailed and consistent in all components of the system (Suggestion 4).

Tests

We found that the validator node implementation has sufficient test coverage. However, our team was unable to run the tests out of the box. We also found that the smart contracts have sufficient unit tests implemented, and recommend improving the test suite by creating integration tests (Suggestion 3).

Documentation

The project documentation provided for this review does not offer a sufficient description of the components of the bridge and how the functionality is implemented in the codebase. We recommend creating technical documentation for the bridge (Suggestion 5).

Scope

The scope of this review included all security-critical components. However, the bridge is in an early stage of completion with critical functionality, such as a validator node being able to read from the Ethereum blockchain, not yet implemented. As a result, we recommend further review by third-party security teams at a later stage of completion.

Dependency

Our team examined the use of dependencies in the bridge implementation and found that the Python library used for BLS signatures has not been audited. We recommend the use of audited and well-maintained libraries where possible.

Specific Issues & 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.

ISSUE / SUGGESTION	STATUS
Issue A: Missing Access Control Modifier	Reported
Suggestion 1: Save Core and Utility Code in One Place and Reference It	Reported
Suggestion 2: Specify Path When Accessing Chialisp Program	Reported
Suggestion 3: Create Integration Tests for Smart Contracts	Reported
Suggestion 4: Improve Code Comments	Reported
Suggestion 5: Create Bridge Documentation	Reported

Issue A: Missing Access Control Modifier

Location

`contracts/ChiaBridgeHead.sol#L139`

Synopsis

The `WithdrawEther` function is missing the `onlyRole(OPERATOR_ROLE)` modifier, which enables only a subset of operators to call the function, as intended by the design.

Impact

The funds of the smart contract would be at risk, as any user would be able to call the function and withdraw ETH arbitrarily.

Remediation

We recommend adding the proper access control modifier to the function so only eligible operators can call it.

Status

Reported.

Suggestions

Suggestion 1: Save Core and Utility Code in One Place and Reference It

Location

`bridge/clsp/include/utility_macros.clib`

`p2_got/clsp/include/utility_macros.clib`

Synopsis

The implementation of the macros in the above files is duplicated. According to the DRY principle, this is not a recommended practice since it makes maintainability difficult and may result in parts of the codebase being left un-updated.

Mitigation

We suggest gathering all core functionality and utility code in one place (repo, package, etc.) and using it whenever needed instead of copying and pasting. Note that this mitigation does not only apply to the bridge but is a recommended general practice that should be implemented throughout the Chia codebase.

Status

Reported.

Suggestion 2: Specify Path When Accessing Chialisp Program

Location

`p2_got/__init__.py#L352`

`bridge/__init__.py#L405`

Synopsis

The function `Program.at` is used in different instances throughout the codebase to access different properties of the Chialisp programs. The input of the function is a string, consisting of r's and f's that specify the path of a list (program). As the length of the string increases, it becomes more difficult to read it, which makes it prone to errors.

Mitigation

We recommend creating constants or enumerations for the different properties and using them in the codebase. This will improve readability and facilitate the verification process from people reading the comment.

Status

Reported.

Suggestion 3: Create Integration Tests for Smart Contracts

Synopsis

Currently, there are comprehensive unit tests using mock contracts for the Solidity smart contracts, but no integration tests. Consequently, some security issues, such as (Issue A), cannot be detected.

Mitigation

We recommend implementing integration tests that simulate a production equivalent environment and work to spot errors arising from the way the individual components of the system work together, in addition to how they work within the ecosystem they will be employed in.

Status

Reported.

Suggestion 4: Improve Code Comments

Location

`Chia-Network-Bridge`

`Chia-Network-Bridge-Contracts`

Synopsis

Currently, the codebase lacks explanation in some areas. This reduces the readability of the code and, as a result, makes reasoning about the security of the system more difficult. Comprehensive in-line documentation explaining, for example, expected function behavior and usage, input arguments, variables, and code branches can greatly benefit the readability, maintainability, and auditability of the codebase. This allows both maintainers and reviewers of the codebase to comprehensively understand the intended functionality of the implementation and system design, which increases the likelihood for identifying potential errors that may lead to security vulnerabilities.

Mitigation

We recommend expanding and improving the code comments within the Chialisp and Python components to facilitate reasoning about the security properties of the system.

Although the Solidity code comments adhere to NatSpec guidelines, they are brief and low-level. More high-level comments or accompanying documentation would also facilitate reasoning about the security properties of the system.

Status

Reported.

Suggestion 5: Create Bridge Documentation

Synopsis

The bridge and the components that compose it are insufficiently documented, which inhibits maintenance, security review, and safe use of the system by users.

Mitigation

We recommend creating comprehensive documentation for the bridge that provides an overview of the architecture and a detailed technical description of the design and implementation. We also recommend creating user documentation to help users make informed security decisions and use the bridge safely.

Status

Reported.

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, zero-knowledge protocols, and consensus 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.