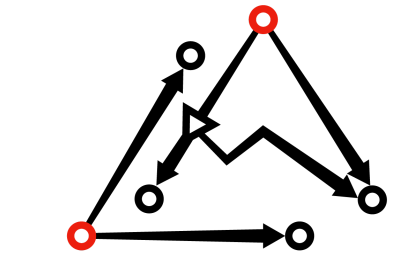

Least Authority

PRIVACY MATTERS

Coloured Coins Implementation

Security Audit Report

Chia Network

Final Report Version: 30 April 2021

Table of Contents

- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - System Design
    - Code Quality
    - Documentation
    - Scope
  - Specific Issues & Suggestions
  - Suggestions
    - Suggestion 1: Write Unit Tests
    - Suggestion 2: Write Property-Based Tests
    - Suggestion 3: Improve and Update Documentation
    - Suggestion 4: Use Consistent Terminology within Code
    - Suggestion 5: Simplify Functions
    - Suggestion 6: Conduct Security Audit of Chia Network
    - Suggestion 7: Conduct Security Audit of BLS Signature Library
    - Suggestion 8: Allow Compatibility for Multiple Coloured Coin Versions
    - Suggestion 9: Create a Style Guide for Chialisp
- About Least Authority
- Our Methodology

Overview

Background

Chia Network has requested that Least Authority perform a security audit of the Coloured coin implementation, a Chialisp smart transaction and its dependencies.

Chia Network aims to build a blockchain and smart transaction platform that is decentralized, efficient, and secure. Chialisp is Chia's new smart transaction programming language. The blockchain is powered by a Nakamoto style consensus algorithm and Proofs of Space and Time aim to replace energy intensive Proofs of Work by utilizing unused disk space.

Project Dates

- **March 1 - April 7**: Code review *(Completed)*
- **April 14**: Delivery of Initial Audit Report *(Completed)*
- **April 29:** Verification *(Completed)*
- **April 30:** Delivery of Final Audit Report *(Completed)*

Review Team

- Phoebe Jenkins, Security Researcher and Engineer
- Ann-Christine Kycler, Security Researcher and Engineer
- David Braun, Security Researcher and Engineer

Coverage

Target Code and Revision

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

The following code repositories contain the Coloured coin implementation that is considered in-scope for the review:

- Coloured Coin High Level Source: https://github.com/Chia-Network/chia-blockchain/blob/main/chia/wallet/puzzles/cc.clvm
- Chia Network CLVM Tools: https://github.com/Chia-Network/clvm_tools
- Chia Network CLVM: https://github.com/Chia-Network/clvm
- Coloured Coin Test Suite: https://github.com/Chia-Network/chia-blockchain/blob/main/chia/wallet/puzzles/test_cc.py
- cc-wallet Python Implementation: https://github.com/Chia-Network/chia-blockchain/tree/main/chia/wallet/cc_wallet
- Block Body Validation: https://github.com/Chia-Network/chia-blockchain/blob/main/chia/consensus/block_body_validation.py

The Coloured coins implementation depends on the rest of the Chia Blockchain repository, CLVM-tools, and CLVM, which will be reviewed to the extent that the dependency is concerned. However, the Proof of Space, Proof of Time/Verifiable Delay Functions (VDF), and CLVM implementations are considered out of scope, as is any dependency and third party code unless specifically included above.

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

Chia-Network/chia-blockchain: `bb06f0b8d49b3525b1d91219615c769896c93d7a`

Chia-Network/clvm_tool: `9179d37e3acb274bc84d2bfa103f43b331f7467b`

Chia-Network/clvm: `1d9bee225d06d72bcb680a1ef22caa1db9ffb55d`

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

https://github.com/LeastAuthority/Chia-Network-chia-blockchain.git

https://github.com/LeastAuthority/Chia-Network-clvm_tools.git

https://github.com/LeastAuthority/Chia-Network-clvm.git

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:

- Chia Blockchain Wiki: https://github.com/Chia-Network/chia-blockchain/wiki
- FAQ:
  - https://github.com/Chia-Network/chia-blockchain/wiki/FAQ
  - https://www.chia.net/faq/
- Aggregated signatures, Taproot, Graftroot, and standard format transactions: https://docs.google.com/document/d/1a4Tor93kqm8d8xHfU3l0ApGS22ytF5sNFMrglv5rlcU/edit
- https://chialisp.com
- Blog post, "*Launching Coloured Coins*": https://www.chia.net/2020/04/29/coloured-coins-launch.en.html

In addition, this audit report references the following documents:

- B. Cohen, K. Piertzak, 2019, "The Chia Network Blockchain." 2019 [CP19]

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation in meeting its requirements of being unforgeable;
- Adversarial actions and other attacks on the smart transactions;
- Potential misuse and gaming of the smart transactions;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Alignment of incentive mechanisms to help prevent unwanted or unexpected behavior;
- Malicious attacks and security exploits that would impact the intended use or disrupt the planned execution;
- Vulnerabilities in the smart transactions, as well as secure interaction between the related and network components;
- Proper management of encryption and signing keys;
- 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 Chia Network is a nascent blockchain protocol that utilizes a Proof of Space (PoS) with verifiable delay functions (VDF) consensus mechanism [CP19]. This includes the Chialisp Virtual Machine (CLVM), a runtime environment and compiler for Chia blockchain smart contract deployment and execution. The Chia team has implemented the Chia Coloured coin, which is developed in a Lisp language variant called Chialisp that is used to interact with the CLVM. The Chia Coloured coins are smart transactions that provide the function of token issuances of alternate coins, assets, and stable coins within the Chia ecosystem. This implementation, along with the system components that relate to the functionality of the Coloured coin, were the core focus of our security review.

Through comprehensive review and analysis, we found that the Chia development team has strongly considered and prioritized the security of the Coloured coin system design. However, a security review of the entire Chia Network system was not in scope for this review (see Scope).

System Design

Our review of the Chia Coloured coin implementation began with a close study of Chialisp, which helped in gaining an appreciation of its subtleties and nuances. We found that Chialisp is an efficient and functional language, which is demonstrated by its brevity and a limited amount of pre-defined primitives. As a result, grasping the language was less complex than understanding other widely used Turing-complete languages that sometimes introduce additional complexity.

In our examination of the correctness of the Chia Coloured coin implementation and the system components that relate to its functionality, we reviewed the `cc.clvm` program extensively. We found that it is not possible for an end-user to unknowingly use a `cc.clvm` version or `genesis-coin-checker` function that have been maliciously modified to pass a forged lineage as a valid transaction. Our team explored whether creating a malicious wallet that creates Coloured coins could pass as valid and found no possibility of forging Coloured coins. Furthermore, we analyzed the security implications of different implementations of the `genesis-coin-checker` and found them to be logically sound. We also found that the components of the Coloured coin implementation that are able to execute code do not execute code arbitrarily, which helps to protect the system against potential vulnerabilities.

The design of the Chia platform greatly restricts state and transaction capabilities, which are common sources of security vulnerabilities found in other platforms. The Coloured coin implementation and `cc_wallet` performs strict lineage checks independently, ensuring a transacted Coloured coin is valid and authentic. Additionally, the Coloured coin program utilizes interlocking assertions and a proof by induction to ensure the Colour of a coin is conserved during transactions, preventing the generation of counterfeit coins.

In our review of the design of the `cc_wallet`, we found that the different versions of the CLVM code will produce incompatible Coloured coins. The current wallet implementation does not consider the possibility of different versions of the Coloured coin source or `genesis-coin-checker` puzzles. In the case that there is an update to either the source puzzle or the `genesis-coin-checker`, this could introduce compatibility issues. We recommend extending `cc_wallet` with an interface for a user to import trusted new versions of the Coloured coin source code, inner puzzles, and `genesis-coin-checkers` (Suggestion 8).

Finally, we identified inconsistent use of terms and variable names in the code base. While the terminology used in the published documentation generally mirrors the terminology used in the code base, some of the terminology is used interchangeably and there are multiple instances of variables being

named inconsistently. Consistent and clear use of the terminology is important in order to avoid confusion or errors. We recommend consistent use of terminology by naming variables and classes consistently by adhering to a single naming convention and naming functions descriptively to indicate their intended purpose (Suggestion 4).

Code Quality

Chia Network is a new ecosystem with ongoing efforts in research and development. As a result, standards, testing frameworks, and tools (e.g. analysis tools such as linters and static analysis programs) have not been developed for use both as part of the development process and in security review and analysis. We recommend that the Chia Network development community continue to work towards the standardization and adoption of security standards and best practices. The development of tools and standardization of best practices will contribute to a closer study and full evaluation of the Chia system's security attributes.

We found the code to be generally well organized. However, we identified instances in which the code did not adhere to programming best practices. For example, some functions produce effects that are not visible in the function signature, decreasing readability and increasing the potential for error. In other instances, functions are longer than a full screen of text. We recommend limiting function bodies so that they fit entirely on a screen, both for readability and to encourage better code design, which adheres to best practices and helps to minimize the opportunity for errors (Suggestion 5).

Tests

The Chia Blockchain development team has taken preliminary steps to provide adequate test coverage. However, we recommend further expanding unit tests by increasing coverage to encompass both success and failure cases (Suggestion 1). In addition, we recommend that unit testing be further supplemented by the addition of property-based tests (Suggestion 2). Property-based testing exercises code with a wide range of randomly generated inputs, including edge cases that are unlikely to be considered during development, and allows observation of how code behaves under certain conditions. Increasing test coverage helps to detect and prevent unintended behavior, allowing future contributors, maintainers, and reviewers of the code base a degree of confidence that the existing code functions properly.

Documentation

The Chia Network's existing documentation is primarily focused on the Chia blockchain consensus algorithms and other cryptographic principles. While the existing documentation was helpful, some of the documentation was out-of-date or incomplete and documentation coverage for certain parts of the system was insufficient. We recommend updating and improving the documentation to encompass technical details of the CLVM, additional information about Chia (e.g. specific explanations of `SpendBundles` and the nature of transactions), and updating the existing documentation to more accurately reflect the coded implementation (Suggestion 3).

Code Comments

The Coloured coin implementation would benefit from additional code comments, which help to increase visibility into the intended functionality of the code. We recommend increasing code comment coverage, which is beneficial for users, security researchers, and the overall security of the system and reduces the opportunity for errors and misunderstanding the intended functionality of the code (Suggestion 3).

Scope

Our team found the scope of the project to be complex in that an in-depth study of a considerable amount of out-of-scope material was necessary in order to reach a comprehensive understanding of the Chia Coloured coin implementation. Furthermore, we found that some components that were considered in

scope (e.g. `block_body_validation.py`) were not particularly informative in helping our team identify vulnerabilities in the Coloured coin implementation, while other components (e.g. offers) were left out of scope that would have been relevant to the security review. We recommend engaging in an audit that examines the entire system beyond the implementation of the Coloured coins (Suggestion 6).

Dependencies

Third-party dependencies were not considered in scope for our review. However, our team noted the use of `blspy`, a Python cryptography library, which has not been formally reviewed for security. We commend the conservative use of dependencies and recommend the utilization of dependencies that have undergone security audits and continue to be well maintained (Suggestion 7).

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
Suggestion 1: Write Unit Tests	Unresolved
Suggestion 2: Write Property-Based Tests	Unresolved
Suggestion 3: Improve and Update Documentation	Unresolved
Suggestion 4: Use Consistent Terminology within Code	Unresolved
Suggestion 5: Simplify Functions	Unresolved
Suggestion 6: Conduct Security Audit of Chia Network	Unresolved
Suggestion 7: Conduct Security Audit of BLS Signature Library	Unresolved
Suggestion 8: Allow Compatibility for Multiple Coloured Coin Versions	Unresolved
Suggestion 9: Create a Style Guide for Chialisp	Unresolved

Suggestions

Suggestion 1: Write Unit Tests

Location

`test_cc.py`

`cc_wallet`

Synopsis

A robust test suite improves code quality by providing a mechanism through which new developers gain an understanding of how the code works through use cases and provides a means to interact with them.

In addition, tests build confidence that the code works as expected for known use cases and provide a basis for the creation of property-based tests (see Suggestion 2).

A comprehensive test suite is particularly security-critical for any system that will be trusted with large amounts of value.

Technical Details

The existing tests provided in `cc_wallet` are a good starting point but represent only the second level (integration tests) of the Testing Pyramid. The implementation is missing unit tests, which are the foundation of a robust test suite and should represent the majority of tests. Additionally, unit tests are effective in that they are fast and tightly isolate the system components being tested. For reference, Integrated Tests Are A Scam provides helpful background information on good test design.

The code in `test_cc.py` more effectively isolates the Coloured coin program than `cc_wallet` but expresses only two use cases, lacks assertions, and does not make use of a proper testing framework.

Mitigation

We suggest the creation of a minimum of twenty-five unit tests for the Coloured coin program while following the principles of Test Driven Development (TDD), a development approach in which the code's behavior and intended functionality are specified and verified by test cases.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 2: Write Property-Based Tests

Synopsis

Anticipation of all of the ways in which the Coloured coin program may be invoked is difficult, however, it is important that certain properties will hold true (e.g. Coloured coins can not be counterfeited). Property-based testing exercises code with a wide range of randomly generated inputs, including edge cases unlikely to be considered by a development team, and allows the user to observe how the code behaves under these conditions.

Mitigation

We recommend the Chia Network team perform property-based tests on the Rust implementation of the CLVM when it is fully deployed. Property-based tests will provide insight into the security characteristics of the Coloured coin implementation.

In addition, we recommend writing property-based tests using the Hypothesis testing framework and run them against the Rust CLVM. The following list is a example of properties to consider for testing:

- A Coloured coin cannot create a coin that is not a Coloured coin of the same lineage;
- All Coloured coins create the Coloured coin conditions when the puzzle reveal is run with the solution;
- With the exception of the genesis coin, parent coins of Coloured coins should always also be Coloured coins of same the lineage and colour;

- A Coloured must not be spent without valid solution (which must include the correct versions of the `genesis-coin-checker` and the `cc.clvm`);
- A Coloured coin must not be spent when a different `cc.clvm` and/or `genesis-coin-checker`
- function was used in its creation;
- A Coloured coin can only create another Coloured coin of the same colour;
- A Coloured coin can not be spent more than once;
- The lineage of a Coloured coin must be traceable to the same genesis coin for all coins of the same colour;
- It must not be possible to validate false announcements in the announcement-ring;
- Given correct puzzle and solution, a Coloured coin should always be spendable;
- The inner puzzle should not influence the Coloured coin's behavior (create coins, announcements, assert announcements, creation of coins of certain lineage, etc.);
- Coloured coins should only be created if their value is 0 or they have a particular parent ID or a particular parent puzzle hash; and
- There should be only one genesis coin.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 3: Improve and Update Documentation

Location

`Chia-Network/chia-blockchain/wiki`

`Chia-Network/chialisp-web`

`wallet/cc_wallet/`

Synopsis

Project Documentation

The existing documentation was helpful in that it provided insight into the Chia blockchain consensus algorithms and other cryptographic principles. However, some of the documentation was out-of-date or incomplete and documentation coverage for certain parts of the system was insufficient.

In addition, we found some areas of the documentation to be insufficient. For example, technical details of the CLVM and detailed explanations of the nature of Coloured Coin transactions (e.g. `SpendBundles`) are missing.

Code Comments

Code comments are minimal and can be further improved to help increase visibility into the intended functionality of the code, which is beneficial for users, security researchers, and the overall security of the system.

Mitigation

Project Documentation

We recommend updating and improving the documentation to encompass technical details of the CLVM, additional information about Chia (e.g. specific explanations of `SpendBundles` and the nature of transactions), and updating the existing documentation to more accurately reflect the coded implementation.

Code Comments

We recommend increasing code comment coverage to encompass all of the functionality within the implementation.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 4: Use Consistent Terminology within Code

Location

Example 1:
`wallet/cc_wallet/cc_wallet.py`
`wallet/cc_wallet/cc_info.py`

Example 2:
`wallet/cc_wallet/cc_wallet.py`
`wallet/cc_wallet/cc_info.py`

Example 3:
`wallet/puzzles/cc.clvm`

Synopsis

The Chia Network has published blog posts, wiki entries, and a green paper [CP19] describing the functionality of the network components. The terminology used in the published documentation generally mirrors the terminology used in the codebase. However, we identified multiple instances of variables being named inconsistently within the implementation. In addition, there are instances in which function names do not provide sufficient insight to the effect of the function.

Example 1:
In the `CCInfo` object of the `CCWallet`, the function `add_lineage` (line 643) extends the attribute `lineage_proofs`.

In this instance, lineage and lineage proof are used interchangeably which is misleading.

Example 2:
The variable in the `CCinfo` class in `cc_info` (an attribute of the `CCWallet` class in `cc_wallet`) is called `my_genesis_checker`. When it is set in `cc_wallet.py` (line 677), it is set with a variable named `genesis_coin_checker`. In another instance (line 166), `my_genesis_checker` is set with `self.cc_info = CCInfo(Program.from_bytes(bytes.fromhex(genesis_checker_hex)),`

[ ]). The hex version of the `my_genesis_checker` attribute of the `CCInfo` object is referred to as colour in `get_colour` (line 276).

In this instance, the connection of the terms “colour”, “my_genesis_checker” and “genesis_coin_checker” is not intuitively clear and different names for the same entity may lead to confusion. Indicating how the terms are connected through naming of the variables and/or through documentation would provide clarity for reviewers and developers of the code. Different names for the same entity may lead to the conclusion that the entities they refer to differ when they are in fact the same.

Example 3*:*
The `cc.clvm` function’s `input-amount-for-coin` (line 215) input argument is called `coin`, however, when this function is referenced (line 359), it is clear that it takes the output value of the function `coin-info-for-coin-bundle`, which, as the name suggests, produces the `coin-info` for a `coin-bundle`. The term `coin-info` is used in the rest of `cc.clvm`, so this is the term that should be used here too.

In this instance, function names do not sufficiently provide definition of the function’s intended purpose.

Mitigation

We recommend consistent use of terminology by naming variables and classes consistently by adhering to a single naming convention and naming functions descriptively to indicate their intended purpose.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 5: Simplify Functions

Location

`Validate_block_body`.

`wallet/cc_wallet/cc_wallet.py`

`wallet/cc_wallet/cc_utils.py`

Synopsis

We found a considerable amount of long functions within the codebase. It is considered a best practice to limit function bodies for readability and to encourage better code design (i.e. small functions composed together).

Additionally, invisible side-effects of functions should be avoided. It should be clear from a function signature how the state is affected. For example, function `generate_new_coloured_coin` (line 658 in `cc_wallet.py`) calls the function `save_info`, which changes the `CCInfo` object stored for the current wallet.

Technical Details

Tools like Radon that can be used to measure code complexity. As an example, we provide below the output of Radon applied to some of the files in scope. Radon rates the functions using the academic

alphabetic grading system of the United States. We used cyclomatic complexity as the target metric. The following functions were highlighted as too complex (regarding the cyclomatic complexity metric) and are only functions that received a grade less than a B:

Example 1:
`src/wallet/cc_wallet/cc_utils.py`
`spend_bundle_for_spendable_ccs - C`

Example 2:
`src/wallet/cc_wallet/debug_spend_bundle.py`
`debug_spend_bundle - D`

Example 3:
`src/wallet/cc_wallet/cc_wallet.py`
`CCWallet.generate_signed_transaction - C`

Example 4:
`src/consensus/block_body_validation.py`
`validate_block_body - F`

Mitigation

We recommend utilizing tools like Radon that can be used to measure code complexity.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 6: Conduct Security Audit of Chia Network

Synopsis

Our security audit was limited to the Coloured coin implementation and other components within the Chia Network have not been audited, to the best of our knowledge.

Mitigation

We recommend a security audit of the security-critical components of the Chia Network.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 7: Conduct Security Audit of BLS Signature Library

Location

`pypi.org/project/blspy/`

Synopsis

The Coloured coin implementation uses the `blspy` dependency, which has not undergone a formal review for security. Cryptography code can be very subtle and small errors can lead to significant issues. The use of maintained and audited dependencies, in addition to conducting regular reviews of dependencies, helps to limit the attack surface and potential vulnerabilities.

Mitigation

We recommend a security audit of the BLS Signatures Library.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 8: Allow Compatibility for Multiple Coloured Coin Versions

Location

`src/wallet/cc_wallet`

Synopsis

In order for the `cc_wallet` to spend Coloured coins, it needs to be capable of revealing the entire source matching the coin's puzzle hash. At present, the wallet is designed to work with a single version of the Coloured coin CLVM source code, one version of the genesis coin checker function curried into it at deployment, and one possible inner puzzle that the Coloured coin may be wrapping. As such, any deviation from this base configuration will require a bespoke wallet for exchanges of that particular Coloured coin. This may be caused by individuals writing custom logic into their own inner puzzles and `genesis-coin-checker`, or could break if there are ever changes to the Coloured coin source, the CLVM compiler or the CLVM language itself - such as renumbering the produced opcodes.

Adding this functionality does introduce potential security issues. New inner puzzles will not be able to violate the guarantees of the Coloured coin implementation or the `genesis-coin-checker`, however, new versions of Coloured coins or the `genesis-coin-checker` puzzles will have that ability. While we may be able to assume that new releases of the base `cc.clvm` will be rare enough that switching to a new wallet could be reasonable, the modular design of the coin checkers implies that different Coloured coins may have entirely different implementations for wallets. Since the coin checkers control the logic of the lineage validation, it would be trivial for an imported coin checker to break this process and introduce security issues allowing them to exchange invalid coins.

While this is a complex issue, a bespoke wallet for every Coloured coin would be a preferred design choice to try to mitigate security issues. However, a security audit for each individual wallet would be necessary to ensure that the implementations have not been altered to introduce vulnerabilities.

Mitigation

We recommend extending `cc_wallet` with an interface for a user to import trusted new versions of the Coloured coin source code, inner puzzles, and `genesis-coin-checkers`.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

Verification

Unresolved.

Suggestion 9: Create a Style Guide for Chialisp

Location

`src/wallet/puzzles`

Synopsis

Much of the Chialisp source code is inconsistent in its layout and indentation, which may potentially confuse users and contributors. In the context of Chialisp's minimal code syntax, consistent layout is very helpful in conveying structure and intended functionality.

Mitigation

We recommend developing a guide for a consistent Chialisp style and apply it to existing and future Chialisp programs.

Status

The Chia Network team has responded acknowledging this suggestion and that they may choose to implement the changes at a future time. As a result, the suggested mitigation remains unresolved at the time of this verification.

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 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.