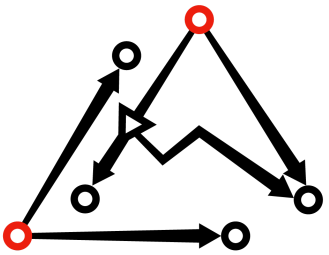

Least Authority

PRIVACY MATTERS

Layer 1 + Smart Contracts

Security Audit Report

Core DAO

Final Audit Report (updated): 17 November 2022

Table of Contents

Overview

Background

Core DAO has requested that Least Authority perform a security audit of their Core DAO Layer 1 and Smart Contracts.

Project Dates

- **August 17 - September 23:** Initial Code Review *(Completed)*
- **September 28:** Delivery of Initial Audit Report *(Completed)*
- **October 31 - November 3:** Verification Review *(Completed)*
- **November 4:** Delivery of Final Audit Report *(Completed)*
- **November 11:** Delivery of updated Final Audit Report *(Completed)*
- **November 17:** Delivery of updated Final Audit Report *(Completed)*

Review Team

- John Amatulli, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer
- Steven Jung, Security Researcher and Engineer
- Giorgi Jvaridze, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Core DAO Layer 1 and Smart Contracts 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:

- Core Chain Repository:
  https://github.com/coredao-org/core-chain
- BTC Mirror Repository:
  https://github.com/coredao-org/btcpowermirror
- Helper Repository:
  https://github.com/coredao-org/core-genesis-contract

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

Core Chain: `6219caa1de6182643e136e27c79355addaee1fc5`

BTC Mirror: `52b70ff629216ce8b40ad05652631da3efb95da3`

Helper: `3690046665d1d33114b93dba80aca65755cd0c1d`

For the verification, we examined the Git revisions:

Core Chain: `6219caa1de6182643e136e27c79355addaee1fc5`

BTC Mirror: `52b70ff629216ce8b40ad05652631da3efb95da3`

Helper: `5a5f87acdef4832fe2e9efcf7e2c44a8519739f7`

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

Core Chain Repository:
https://github.com/LeastAuthority/core-chain

BTC Mirror Repository:
https://github.com/LeastAuthority/core-btc-power-mirror

Helper Repository:
https://github.com/LeastAuthority/core-system-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:

- Relayer Workflows *(shared with Least Authority via email on 22 June 2022)*
- COREWhitepaper_v7.10.22.pdf *(shared with Least Authority via email on 15 July 2022)*
- Core Design Specification *(shared with Least Authority via email on 15 August 2022)*

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the smart contracts;
- Attacks that impacts 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 of the smart contracts or disrupt their execution;
- Vulnerabilities in the L1 and smart contracts' code;
- Protection against malicious attacks and other ways to exploit the L1 code and smart contracts;
- 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 Core DAO Layer 1 and Smart Contracts compose a system that aims to overcome the blockchain trilemma, a belief that a tradeoff is required between decentralization, security, and scalability in the design of distributed consensus mechanisms. The system proposes Satoshi Plus, a Proof of Work (PoW) and Proof of Stake (PoS) consensus mechanism that uses a validator selection algorithm, which considers both the hashing power and amount of stake in the selection of 21 validators for each consensus round.

Our team performed a comprehensive review of the design and implementation of the Layer 1 and Smart Contracts to identify security vulnerabilities. We focused on modifications made to the Core DAO fork of the Binance (BSC) protocol to support the Satoshi Plus consensus mechanism as implemented in `satoshi.go`. Although our team did not find any issues in the design or the implementation, we did identify several opportunities to improve the code quality and overall security of the system.

System Design

Our team investigated the design of the system as proposed in the documentation and implemented in the code. The system aims to provide scalability while maintaining decentralization. However, centralization remains a concern with any PoS mechanism using a limited number of validators. Although the model of utilizing the decentralization of the Bitcoin network could mitigate the centralization concerns of a PoS network, the required number of validators in Satoshi Plus consensus is limited to the 21 validators with the largest stake and hashing power. Consequently, it is difficult to anticipate the optimal number of verifiably independent validators needed to achieve the decentralization claims of the whitepaper.

Our team examined additional areas of concern provided by the client after the scheduled code review and the initial audit report had been delivered. In examining the security of the system of smart contracts, our team found no evidence that the funds held by the contract are vulnerable to attacks. Although we did not find vulnerabilities in our review, it does not guarantee an absence of vulnerabilities but only means that the risk of vulnerabilities existing is lower as a result of the diligence undertaken.

Our team investigated the possibility of a successful denial of service attack whereby the network is halted. If the validator set is less than the required threshold, then the consensus cannot move to the next round. This could be a possibility if the validator set stops changing for each round and the validator set is sufficiently limited, such that for each round all the validator nodes participating get selected. In this case, if enough of these nodes get disconnected so that the threshold is not met, the network would halt.

Our team examined the governance implementation and did not identify any vulnerabilities. However, given that governance actions can only take effect in the next round, if there is a problem that prevents a current round from completing and a new one from starting, governance will be unable to make any effective changes.

We found that the Solidity compiler version is inconsistently set in the smart contracts. We recommend that all the contracts use the most recent compiler version (Suggestion 8).

Code Quality

The Core DAO Layer 1 and Smart Contracts code is well organized and generally adheres to best practice. However, we identified unnecessary checks and internal variables that can be removed to improve readability and reduce gas consumption (Suggestion 2, Suggestion 3). We also found inconsistency in the use of the `uint256` type. We recommend type consistency and avoidance of unnecessary type conversions (Suggestion 4, Suggestion 5). In addition, we identified an instance of an unnecessary `Boolean` comparison and recommend removing it to improve readability (Suggestion 6).

Tests

We found that sufficient test coverage that checks the correctness of the implementation, and that the implementation functions as intended, has been implemented.

Documentation

The documentation provided for this review offered a high level overview of the system, which was helpful. However, the documentation could be improved by providing an architectural explanation that includes a deeper level of technical detail. The diagrams included in the documentation are conceptual rather than technical. The swimlane diagrams do utilize function names but provide only cursory insight and do not contain detailed functional information.

We recommend that the documentation (README.md) in the Core System Contracts repository be improved to include build instructions with a list of software prerequisites detailing specific versions (e.g., Python 3.8+) and a list of commands required to perform tests.

Code Comments

We found that the codebase contains minimal code comments describing the intended behavior of the functions and components that compose the system. We recommend increasing code comments and adhering to NatSpec guidelines for solidity code comments (Suggestion 1). Furthermore, we found that in several areas, the existing code has conflicting code comments, which can be misleading. We recommend being consistent with code comments throughout the codebase (Suggestion 7).

Scope

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

Dependencies

Our team did not identify security vulnerabilities regarding the use of dependencies in the Core DAO Layer 1 and Smart Contracts. However, the system is susceptible to inheriting any Bitcoin Network related issues. We recommend that the Core DAO team actively monitor security updates in Geth, BSC, and btcsuite for security vulnerabilities.

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: Follow NatSpec Format	Resolved
Suggestion 2: Remove Unnecessary Checks	Resolved
Suggestion 3: Remove Unnecessary Internal Variables	Resolved
Suggestion 4: Be Consistent with uint256 Type Usage	Resolved
Suggestion 5: Remove Unnecessary Type Conversion	Resolved
Suggestion 6: Remove Boolean Comparison	Resolved
Suggestion 7: Be Consistent with the Comments in .sol and .template	Resolved

Suggestion 8: Upgrade Solidity Version and Lock the Pragma	Resolved

Suggestions

Suggestion 1: Follow NatSpec Format

Synopsis

Functions present in the codebase do not follow NatSpec guidelines for Solidity comments. Using NatSpec comments helps code reviewers and users easily understand the inputs and functionality of every method present, and therefore aids in identifying any potential issues.

Mitigation

We recommend adding code comments following NatSpec guidelines.

Status

The Core DAO team has implemented NatSpec code comments to security critical functions and components.

Verification

Resolved.

Suggestion 2: Remove Unnecessary Checks

Location

`contracts/BtcLightClient.sol#L81`

`contracts/ValidatorSet.sol#L258`

Synopsis

The first reference location has a divisibility check between constants. This is decided before deployment and can be removed. The second reference location has a check to verify if a `uint256` value is greater or equal to zero. It can also be removed, as it is always `true`. Unnecessary checks spend gas needlessly.

Mitigation

We recommend removing unnecessary checks.

Status

The Core DAO team has removed unnecessary or redundant checks.

Verification

Resolved..

Suggestion 3: Remove Unnecessary Internal Variables

Location

`contracts/BtcLightClient.sol#L82`

contracts/BtcLightClient.sol#L169

Synopsis

The internal variables in the reference locations are not needed. Constant or parameter variables can be used directly to save gas.

Mitigation

We recommend removing unnecessary internal variables.

Status

The Core DAO team has removed the unnecessary internal variables.

Verification

Resolved.

Suggestion 4: Be Consistent with uint256 Type Usage

Location

contracts/BtcLightClient.sol#L119

contracts/CandidateHub.sol#L148

contracts/Migrations.sol#L233

contracts/PledgeAgent.sol#L367

contracts/SlashIndicator.sol#L99

Synopsis

`uint` was used instead of `uint256` in several locations, including the referenced locations. `uint` is an alias of `uint256`, but there are many unsigned integer types, such as `uint8`, `uint16`, etc. This might be confusing for readers or maintainers trying to understand the system.

Mitigation

We recommend consistent usage of `uint256`.

Status

The Core DAO team has implemented consistent type usage.

Verification

Resolved.

Suggestion 5: Remove Unnecessary Type Conversion

Location

contracts/PledgeAgent.sol#L496

Synopsis

This code converts an `address` type constant into `uint160` and subsequently back to an `address` type. However, the constant can be used directly without any conversion.

Mitigation

We recommend removing unnecessary type conversion.

Status

The Core DAO team has rewritten the smart contract and removed the above-mentioned code.

Verification

Resolved.

Suggestion 6: Remove Boolean Comparison

Location

`contracts/GovHub.sol#L166`

Synopsis

`Boolean` variables can be used directly as an `if` condition and do not need to be compared against `true` or `false`.

Mitigation

We recommend using `Boolean` variables directly without comparing them against `true` or `false`.

Status

The Core DAO team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 7: Be Consistent with the Comments in .sol and .template

Location

`contracts/ValidatorSet.sol`

`contracts/ValidatorSet.template`

`contracts/SlashIndicator.sol`

`contracts/SlashIndicator.template`

`contracts/PledgeAgent.sol`

`contracts/PledgeAgent.template`

`contracts/CandidateHub.sol`

`contracts/CandidateHub.template`

`contracts/BtcLightClient.sol`

`contracts/BtcLightClient.template`

Synopsis

There are inconsistent code comments describing the same code in `.template` and `.sol` files. This can result in confusion and inhibit readability.

Mitigation

We recommend that code comments be made consistent in the `.template` files and their corresponding `.sol` files.

Status

The Core DAO team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 8: Upgrade Solidity Version and Lock the Pragma

Location

`contracts/BtcLightClient.sol#L1`

`contracts/Burn.sol#L1`

`contracts/CandidateHub.sol#L1`

`contracts/Foundation.sol#L1`

`contracts/GovHub.sol#L1`

`contracts/Migrations.sol#L1`

`contracts/PledgeAgent.sol#L1`

`contracts/RelayerHub.sol#L1`

`contracts/SlashIndicator.sol#L1`

`contracts/System.sol#L1`

`contracts/SystemReward.sol#L1`

`contracts/ValidatorSet.sol#L1`

Synopsis

The pragma on the contracts is version 0.6.x floating. The external libraries offer support for 0.8.x or are no longer needed, so there are no compatibility issues. Version 0.6.x allows the use of unsafe features that were removed in newer versions. This allows for a broader attack surface. Also, a floating pragma is an error-prone practice that could lead to deployment issues in the case that a wrong compiler version is used.

Mitigation

We recommend upgrading to the most recent compiler version, as it may include features and bug fixes for issues that were present in previous versions, and locking the pragma.

Status

The Core DAO team has upgraded the Solidity compiler version as recommended.

Verification

Resolved.

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.