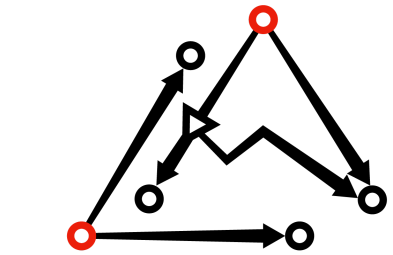

Least Authority

PRIVACY MATTERS

# Zcashd NU6 Updates

## Security Audit Report

# ZCG

Final Audit Report: 21 October 2024

# Table of Contents

# Overview

## Background

As the Zcash Ecosystem Security Lead, Least Authority reviewed the changes made to Zcashd that will be introduced in the NU6 network upgrade.

## Project Dates

- **September 2, 2024 - September 16, 2024:** Initial Code Review *(Completed)*
- **September 17, 2024:** Delivery of Initial Audit Report *(Completed)*
- **21 October, 2024:** Verification Review *(Completed)*
- **21 October: 2024** Delivery of Final Audit Report *(Completed)*

## Review Team

- Will Sklenars, Security Researcher and Engineer
- Dominic Tarr, Security Researcher and Engineer

# Coverage

## Target Code and Revision

For this audit, we performed research, investigation, and review of the Zcashd NU6 Updates followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in scope for the review:

- Zcashd release version 5.10.0:Release tag:
    - https://github.com/zcash/zcash/releases/tag/v5.10.0

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

- `3b9272462a264cf9d0400fba3e4fa84584cde3b1`

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

- https://github.com/LeastAuthority/ZcashFoundation-Zcashd

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:

- For Debian packages see:
  https://zcash.readthedocs.io/en/latest/rtd_pages/install_binary_tarball.html
- Scope details from the ECC team (Google Docs) (*shared with Least Authority on 27 August 2024*)

In addition, this audit report references the following documents and links:

- ZIPs:
  https://zips.z.cash

## Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Protection against malicious attacks and other ways to exploit;
- Consistency of the implementation with the specifications;
- Balance violation;
- Potential loss of funds; and
- Anything else as identified during the initial analysis phase.

# Findings

## General Comments

NU6 is the seventh upgrade to the Zcash protocol, which aims to update the consensus rules in order to implement changes in the way Zcash development is funded, as agreed by the community. For this audit, our team performed a manual code review of the NU6 upgrade changes in Zcashd – the original Zcash node implementation, written in C++. The NU6 update specifies that blocks should balance exactly (ZIP 236), sets the funding rates (ZIP 214), and introduces an in-protocol 'lockbox' (ZIP 2001). Our team also reviewed the relevant documentation.

During our review, we confirmed the critical configurations of the network upgrade. Our team verified that the `CONSENSUS_BRANCH_ID`, which is an arbitrary, unique, and random integer representing the NU6 branch of the Zcash blockchain, is correctly reflected in both `upgrades.cpp` and the Zcash specification.

We also confirmed that relevant addresses in the implementation match the specification (ZIP 214), including the Zcash foundation (ZF) address and the major grants (MG) address. We additionally confirmed the address for the Bootstrap Project (BP), although we noted that BP is still referred to as 'ECC' (Electric Coin Company) in the code (Suggestion 1). Since BP is the parent company of ECC, we recommend updating the code to make the reference accurate.

Furthermore, we confirmed that the funding rates were correct for the ZF, MG funding streams, lockbox, and Zcash community grants (ZCG).

ZIP 201 specifies that once the `ACTIVATION_HEIGHT` for a consensus rule change has been reached, all peers supporting the new consensus rule set should disconnect from any peers that do not support the new consensus rule set. Our team checked this process and did not find any issues.

### System Design

Our team reviewed the network upgrade deployment procedure and found that it has been well-designed, with security at the forefront of the design considerations.

The successful deployment of the network upgrade is crucial, as every node must agree on what a valid block is, which requires that all nodes agree on the consensus rule set. It is possible to change the consensus rule set in a network upgrade, provided that all nodes do so at the same time. The nodes must also ensure that for historical blocks, the correct former consensus rule set is used. To propose a new consensus rule set, the new rules are discussed publicly to give the community stakeholders (such as

miners) an opportunity to raise objections. A new version of the code that implements the new consensus rules is then created.

The NU6 consensus rule set will be activated at a predetermined block height (`ACTIVATION_HEIGHT`), which will be defined in ZIP 253, but was yet to be defined at the time of this review. It is intended to coincide with the halving of block rewards for miners. Zcash users must upgrade to the new version before the activation height is reached.

Zcashd includes an End-of-Service halt feature, which causes a node running a particular version to automatically shut down approximately 16 weeks after that version was released. Because of this feature, by the time the `ACTIVATION_HEIGHT` for NU6 has been reached, all nodes in operation will support the NU6 consensus rule set. The exception to this rule is if a user has manually modified the code to disable the End-of-Service halt feature. The feature is designed to avoid a hard fork, where two consensus branches continue to be built upon, due to some clients building and accepting blocks that are considered invalid by other clients.

In this audit, the new consensus rules introduced a `lockbox` type for a funding stream (which is currently not possible to extract funds from, but will be defined further in future upgrades), specified new recipient addresses for funding streams, set fractional rates for those funding streams, and added the requirement that the entire coinbase transaction should balance. Before the requirement of balanced transactions, miners were permitted to take any amount up to, and including, the total subsidy and fees, but were not required to take the entire amount. With the introduction of balanced transactions, a miner must take the total subsidy and fees available.

## Code Quality

The code within scope was of high quality. The code was also well-organized and followed best practices. We were not able to identify any issues or suggestions relating to code quality.

#### Tests

We found the tests to be sufficient. Zcashd includes tests that cover the basic functionality of the protocol. The implementation also includes tests for invalid cases.

## Documentation and Code Comments

Along with reviewing the code, our team reviewed the specification, specifically the ZIPs 201, 214, 236, 253, 316, 1014, 1015, and 2001. We found the documentation to be clear and correct, with no ambiguities. However, we were able to identify a few areas where clarity could be improved, and found informational URLs that are out of date (Suggestion 1 and Suggestion 2).

We also found the code comments to be sufficient.

## Scope

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

#### Dependencies

We did not identify any concerns related to the use of dependencies. We recommend performing another audit when the protocol is next upgraded.

# 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: Improve Clarity in ZIPs | Planned |
| Suggestion 2: Replace Outdated URLs | Unresolved |

# Suggestions

## Suggestion 1: Improve Clarity in ZIPs

**Location**

`ZIP 201`

`ZIP 214`

**Synopsis**

We found that in some cases, the ZIPs were not entirely clear and were difficult to understand.

**Mitigation**

We recommend making some of the ZIPs more explicit, as follows:

- In ZIP 201, the network handshake is described as:
  *`L -> R: Send version message with the local peer's version. Then R -> L: Send version message back.`*
  This is ambiguous, as it does not specify which peer's version is being sent back – local or remote. Although the reader can reasonably assume that the remote peer's version is the one that is sent back, it could be stated more explicitly.
- In ZIP 214, the funding stream `FS_ZIP214_BP` is defined (with BP referring to the Bootstrap Project). This funding stream is present in the code, but it is stored in a variable whose name starts with ECC, rather than BP. Later, in ZIP 214, it is noted that ECC shall generate the addresses on behalf of BP. As noted earlier in the General Comments section, we recommend updating the reference to avoid any confusion.

**Status**

The Zcashd team stated that the aforementioned addresses are for pre-NU6 funding streams; however, the team still plans to update them in the future.

**Verification**

Planned.

## Suggestion 2: Replace Outdated URLs

**Location**

consensus/upgrades.cpp#L47-L56

**Synopsis**

The code includes data structure, which contains information about each network upgrade, such as the branch ID, and a URL linking to a page on the Zcash website. However, for both NU5 and NU6, the URLs respond with a 404 error.

**Mitigation**

We recommend either replacing the URLs or removing them.

**Status**

The Zcashd team stated that the websites for NU5 and NU6 should exist, and that the current 404 error is due to a website bug.

**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, 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.