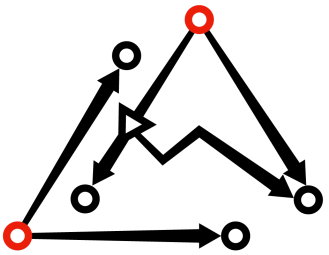

Least Authority

PRIVACY MATTERS

Zebra NU6.1 Network Upgrade

Security Audit Report

ZCG

Final Audit Report: 3 November 2025

Table of Contents

Overview

Background

As the Zcash Ecosystem Security Lead, Least Authority reviewed the changes made to Zebra that will be introduced in the NU6.1 network upgrade.

Project Dates

- **August 20, 2025 -** August 27, 2025: Initial Code Review *(Completed)*
- **August 29, 2025:** Delivery of Initial Audit Report *(Completed)*
- **November 3, 2025:** Verification Review *(Completed)*
- **November 3, 2025:** Delivery of Final Audit Report *(Completed)*

Review Team

- Poulami Das, Security / Cryptography Researcher and Engineer
- Eduardo Morais, Cryptography Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Zebra NU6.1 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:

- Zebra repository:
  https://github.com/ZcashFoundation/zebra

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

- b01446f66eba95748d0b761bf417d2744a44c874
- 3bac7fd6bbeb74df56d4aa2b14c9916c8de6d98e
- 82ff285582df10d809f1a49321b561bba1d71b07
- 639c09af08cebdff8ca9ea0db31d6ac26c0c6705
- a986a5421e3e1319cc004e2913a7a5ed7c13a468
- 5d1dd798c6b69959209612b7f659dcd3cfc799b3

The following Git milestone describes the scope and provides documentation for the revision:

- https://github.com/ZcashFoundation/zebra/milestone/43

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

- https://github.com/LeastAuthority/zebra

For the verification, we examined the Git revision:

- `e422d2f4e5134ce2233282405a8578d3077c160f`

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:

- https://zips.z.cash/draft-ecc-community-and-coinholder
- https://zips.z.cash/zip-0201
- https://zips.z.cash/zip-0255
- https://zips.z.cash/zip-1015
- https://zips.z.cash/zip-1016
- https://zips.z.cash/zip-0271

In addition, this audit report references the following document:

- Previous Least Authority Report for NU6: https://leastauthority.com/wp-content/uploads/2024/11/Least-Authority-ZCG-Zcashd-NU6-Updates-Final-Audit-Report.pdf

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

The Zebra project is a Rust implementation of a Zcash node, developed by the Zcash Foundation as an alternative to the reference `zcashd` client. Its core functionality centers on validating blocks, enforcing consensus rules, and managing the Zcash ledger. Zebra plays a key role in supporting the decentralization of the Zcash network, strengthening security through independent implementations, and facilitating protocol upgrades such as NU6.1.

We audited the Zebra repository updates associated with the NU6.1 milestone, as defined in ZIP specification 255, along with the underlying core ZIPs of 201, 271, and 1016. ZIP 255 specifies important constants, including the consensus branch ID and the activation height for testnet, and mandates minimum supported protocol versions for both testnet and mainnet. The peer-management rules from ZIP 201 apply unchanged in ZIP 255.

ZIP 271 introduces a one-time disbursement of all funds from the Deferred Dev Fund Lockbox to a transparent P2SH multisig address at the moment of activation, from which key-holders will allocate grants, in accordance with the rules defined in the Community and Coinholder Funding Model (ZIP 1016). Additionally, ZIP 271 proposes extending protocol-based development funding beyond the scheduled end of existing streams defined in ZIP 1015, introducing mechanisms for managing lockbox accruals. It emphasizes security by requiring multisig controls that remain resilient even if one of the parties becomes uncooperative or loses their keys. ZIP 1016 focuses on the rules of the Community and Coinholder Funding Model, under which 8% of block rewards go to Zcash Community Grants (ZCG), and 12% accrue to a coinholder-controlled fund, initially seeded by the Deferred Dev Fund Lockbox.

We compared the implementation changes against their specifications in ZIP 255 and related ZIPs, and found that the code matches the required changes. In particular, we reviewed the associated PRs: #9603, #9754, #9747, #9757, and #9762, which implement the mechanisms for one-time lockbox disbursement and funding stream extension, as per ZIP 271 and the corresponding `Testnet` parameters. These mechanisms are not yet implemented for mainnet, as the specification currently lacks the necessary information on activation height, recipient address, funding stream start, and end heights for mainnet.

We examined block and transaction validation logic for susceptibility to fund manipulation or draining attacks and did not identify any critical issues.

We also reviewed interactions with third-party contracts and dependencies, and found no vulnerabilities or untrusted code execution paths.

In addition to these main areas, we identified a few code and documentation concerns that, while not security-critical, could improve clarity and maintainability.

Dependencies

Our team did not identify any vulnerabilities in the implementation's use of dependencies.

Code Quality

We performed a manual review of the in-scope codebase and found it to be of high quality, well organized, and generally aligned with development best practices.

However, during our review, we identified several areas for improvement, including duplicated test vector error messages representing different logic, inconsistent handling of constants, and inconsistent variable naming. We recommend addressing these issues to help maintain consistency, accuracy, and clarity in both the implementation and accompanying comments (Suggestion 2, Suggestion 3, Suggestion 4).

Tests

We found the tests to be sufficient, covering both the basic functionality of the protocol and tests for invalid cases.

Documentation and Code Comments

Along with reviewing the code, our team reviewed ZIPs 201, 255, 271, and 1016. We found the documentation clear and correct, with no ambiguities, and noted one area where clarity could be improved (Suggestion 1). In addition, we found the code comments sufficient.

Scope

The scope of this review was limited to changes introduced in the NU6.1 network upgrade.

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 Constant Comment	Resolved
Suggestion 2: Update Inconsistent Error Messages	Resolved
Suggestion 3: Update Inconsistent Definition of Activation Height	Resolved
Suggestion 4: Update Variable to Plural Form	Unresolved

Suggestions

Suggestion 1: Improve Constant Comment

Location

`diff`

Synopsis

Although the value of the constant `POST_NU6_1_FUNDING_STREAMS_NUM_ADDRESSES_TESTNET` was updated in commit `5d1dd798c6b69959209612b7f659dcd3cfc799b3`, the comment above the constant, starting on line 583, remained unchanged. The explanation for the change is given in the description of PR 9786, as follows:

"*We want to match the number of addresses defined in zcashd and ZIP 255 for the FPF funding streams on Testnet.*

37 addresses was correct for 1.26M blocks, but only 27 are needed now that the Testnet funding stream ends after 939,500 blocks.

This PR will not change Zebra's behaviour (except that it will use less memory, but only 10 pointers worth / 80B)."

Mitigation

We recommend updating the comment to reflect the reasoning provided in PR 9786.

Status

The Zebra team has updated the comment in PR 9952.

Verification

Resolved.

Suggestion 2: Update Inconsistent Error Messages

Location

`network/tests/vectors.rs#L377`

`network/tests/vectors.rs#L394`

`network/tests/vectors.rs#L490`

`network/tests/vectors.rs#L500`

Synopsis

The unit test `check_configured_funding_stream_constraints` performs multiple checks. Each check should have a clear error message. However, the check at line `394` has the same error message as the check at line 377, indicating a copy/paste problem. Analyzing each check, it is possible to see that the first check (at line 377) verifies the expected height, while the second one (at line `394`) checks the expected recipient. Similarly, the check at line `490` has the same error message as the check at line `500`, while carrying out a different check.

Mitigation

We recommend updating the error messages (at lines `394` and `500`) to reflect the unit test behavior.

Status

The Zebra team has updated the error messages in PR 9952.

Verification

Resolved.

Suggestion 3: Update Inconsistent Definition of Activation Height

Location

`diff`

Synopsis

A new constant, `NU6_1_ACTIVATION_HEIGHT_TESTNET`, was introduced to represent the activation height. This constant is used to define the constant vector `TESTNET_ACTIVATION_HEIGHTS`. However, other elements of the vector are not defined in the same way, leading to inconsistency. For instance, the other elements are defined directly by calling `block::Height()`.

Mitigation

To improve clarity, we recommend defining constants for the activation height of each network update. Each constant should include a descriptive comment and a link to the corresponding documentation, following the pattern used for `NU6_1_ACTIVATION_HEIGHT_TESTNET`.

Status

The Zebra team has updated the code in PR 9952 to align it with the implementation, as recommended in this report.

Verification

Resolved.

Suggestion 4: Update Variable to Plural Form

Location

`zebra-network/src/config.rs#L21`

Synopsis

The variable `ConfiguredLockboxDisbursement` should be referred to in its plural form, `ConfiguredLockboxDisbursements`, to align with naming conventions and maintain consistency with other variables such as `expected_one_time_lockbox_disbursements` and `one_time_lockbox_disbursements`.

Mitigation

We recommend renaming the variable to its plural form and updating all corresponding references throughout the codebase.

Status

The Zebra team has opened PR 9952 to update the variable, but the change was not applied.

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.