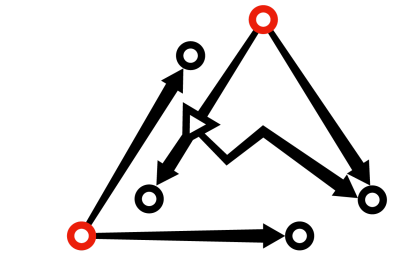

Least Authority

PRIVACY MATTERS

Zebra NU6 Updates

Security Audit Report

ZCG

Final Audit Report: 8 November 2024

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 network upgrade.

Project Dates

- **August 22, 2024 - September 6, 2024:** Initial Code Review *(Completed)*
- **September 10, 2024:** Delivery of Initial Audit Report *(Completed)*
- **November 8, 2024:** Verification Review *(Completed)*
- **November 8, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- Poulami Das, Security / Cryptography Researcher and Engineer
- Nikos Iliakis, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer

Coverage

Target Code and Revision

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

- Zebra : https://github.com/zcashfoundation/zebra
    - limited to the changes required for the NU6 network upgrade implemented in the PRs 8694, 8727, 8729, 8732, 8733, 8742, and 8747

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

- `c32c4f6fec158ad412b981e36fa148e4447ee70d`

For the verification, we examined the Git revision:

- `46c6b6eb38617c07e8a130650b33831759b73263`

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

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

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:

- Full list of ZIPs to be implemented, as listed in ZIP 253: Deployment of the NU6 Network Upgrade: https://zips.z.cash/zip-0253
- Render of the draft for ZIP 253: https://github.com/zcash/zips/blob/8de2dcb19bc33ddeca594b3bd0f176ac18e3f1eb/zips/zip-0253.md
- Website: https://zebra.zfnd.org

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 of the Zcash network upgrades, which aim to update the consensus rules in order to improve the network. For this audit, our team performed a manual code review of the NU6 upgrade changes in Zebra – Zcash's independent, consensus-compatible implementation of a Zcash node, written in Rust.

Our team examined the provided documentation of listed ZIPs included in ZIP 253 and checked it against the implementation of the NU6 update. Key changes are implemented in different PRs, as listed in the Target Code and Revision section, and include new consensus rules, such as updated block template construction and block validation rules that check the correct balancing of the coinbase transactions, in addition to changes regarding the block subsidy distribution between the two funding stream receivers – Financial Privacy Foundation (FPF) and the lockbox.

We additionally assessed the correctness of the implementation, focusing specifically on the possibility of balance violation and the potential loss of funds. We did not find any issues in the implementation for the above attack scenarios. However, we did identify two suggestions regarding inconsistencies between the documentation and code comments (Suggestion 1), as well as the implementation of error handling and the use of panics (Suggestion 2), which we recommend be improved.

Code Quality

We performed a manual review of the changes (PRs) related to the NU6 update and found that the codebase is generally well-organized and adheres to the coding principles of Rust.

Tests

Our team found that sufficient testing has been added to verify the new functionality.

Documentation and Code Comments

The project documentation consisted of a list of ZIPs provided in ZIP 253, which our team mostly referred to in order to check the correctness of the code changes. These ZIPs in scope describe the consensus rule changes regarding the NU6 update. Additionally, although the code comments generally describe the intended behavior of the code, we found some discrepancies between the comments and the code, as well as the ZIP documentation, which we recommend be corrected and updated (Suggestion 1).

Scope

The scope of this review did not include the rest of the Zebra codebase, which our team had to occasionally refer to in order to better assess the logic of the NU6 changes.

Dependencies

Our team did not identify any changes related to the use of dependencies.

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: Update Code Comments and ZIP Documentation	Resolved
Suggestion 2: Improve Error Handling and Avoid Using Panics	Partially Resolved

Suggestions

Suggestion 1: Update Code Comments and ZIP Documentation

Synopsis

Our team found some discrepancies between the comments and the code as well as the ZIP documentation. Below we list a few examples:

- The information in ZIP 253 and ZIP 1015 regarding the Mainnet activation start/end height as well as the Testnet activation start/end height is either inconsistent or incomplete.
- In ZIP 253, the Mainnet activation height (possibly start height) is denoted as TBD. However, in ZIP 1015, the Mainnet activation start height is set as 2726400.
- The information on the Mainnet/Testnet activation end height appears only in ZIP 1015 and not in ZIP 253.
- Our team found some inconsistencies in the code comments in `parameters/network/subsidy.rs`:
    - The constant `LOCKBOX_SPECIFICATION` is currently set to the following configuration. However, ZIP 253 does not refer to this link anymore; hence, the constant should be updated to https://zips.z.cash/zip-1015.
    - In ZIP 1015, the 8% subsidy is said to be distributed to the Financial Privacy Foundation (FPF). However, for Mainnet, in `parameters/network/subsidy.rs`, the fund is

assigned to the Major Grants funding stream, and the addresses are taken as those in `FUNDING_STREAM_MG_ADDRESSES_MAINNET`.

Mitigation

Our team recommends the following:

1. Either being consistent with the information shared in the two ZIPs or providing the activation height information only in one of the ZIPs (possibly ZIP 1015);
2. Updating the outdated link to https://zips.z.cash/zip-1015 in several locations of the NU6 update code; and
3. Updating the inconsistent nomenclature (FPF vs `FUNDING_STREAM_MG_ADDRESSES_MAINNET`) in the code and ZIP 1015 documentation.

Status

The Zebra team has implemented the mitigation as recommended.

Verification

Resolved.

Suggestion 2: Improve Error Handling and Avoid Using Panics

Location

Examples (Non-exhaustive):

`block/subsidy/general.rs#L106`

`block/subsidy/general.rs#L137`

`src/block/check.rs#L189`

`subsidy/funding_streams/tests.rs#L132`

Synopsis

The aforementioned locations indicate where the code could potentially panic, leading to the code crashing unexpectedly. For example, If any of the parameters in `parameters/network/subsidy.rs` are not correctly set, then this will lead to panics in all of the above functions.

Mitigation

We recommend removing panics where possible. One of the possible improvements is to propagate errors to the caller and handle them on the upper layers. Note that error handling does not exclude using panics. In addition, if a caller can return an error, the callee function may not panic but, instead, propagate an error to the caller.

Status

The Zebra team has fixed a selected number of potential panics here (more specifically: `zebra-consensus/src/block/check.rs#189`, `zebra-consensus/src/block/check.rs#L206-213`, and `zebra-consensus/src/block/check.rs#L263-267`). While the Zebra team acknowledges that there are several, other potential panics that may be triggered, they noted that they plan to address the rest of the panics in a future development phase. The team added that the hard coded Mainnet and Testnet parameters have been manually verified and any configured Testnet parameters should be validated when parsing the Zebra's configuration, as avoiding panics in these cases would still leave

Zebra unable to correctly validate blocks and transactions; hence, they stated that the benefit of replacing them with errors is limited.

Verification

Partially 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, 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.