Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Blobstream Zero

Veridise Inc.

Sep. 17, 2024

► **Prepared For:**

RISC Zero
https://risczero.com/

► **Prepared By:**

Benjamin Mariano
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Oct. 7, 2024	V3
Sep. 27, 2024	V2
Sep. 18, 2024	V1

Contents

1 Executive Summary

From Sep. 9, 2024 to Sep. 13, 2024, RISC Zero engaged Veridise to conduct a security assessment of the Blobstream Zero protocol. Veridise conducted the assessment over 10 person-days, with 2 security analysts reviewing the project over 5 days on commit `925bff9`. The security assessment strategy involved a thorough source code review performed by Veridise analysts.

Project Summary. The original Blobstream protocol[1] was introduced to provide an opportunity for L2 Ethereum networks built on top of the Celestia data network, offering cheaper operations and higher throughput. Instead of storing commitments generated by L2 aggregators on Ethereum, they are stored on Celestia. Later, the data inclusion proofs are sent from Celestia to Ethereum via relay nodes. Once the data inclusion proofs have been sent, the L2 network can verify the availability of their data on-chain, as if it were stored on Ethereum.

The Blobstream Zero protocol implements the Blobstream protocol on top of the RISC Zero development stack. Implementing on the RISC Zero stack comes with a number of benefits, including the ability to use RISC Zero ZK Rollup/Coprocessor proofs for simple on-chain verification and simplified maintainability/development as the ZK light client and the bridge can be written entirely in Rust.

The Blobstream Zero protocol software consists of the following modules:

1. `CLI` is an off-chain program that is responsible for regularly relaying the latest block inclusion information to the `Blobstream0` smart contract.
2. `Light Client Guest` is a part of `CLI` that takes recently added block headers as input and generates a SNARK proof that those block headers have been included, starting from a block that is considered trusted.
3. `Blobstream Zero` smart contract is responsible for updating the latest trusted block information in case the corresponding proof has been provided, as well as data attestation functionality.
4. `RiscZeroVerifier` smart contract is responsible for verifying SNARK proofs of data inclusion that were produced by the `Light Client Guest`.

The content of the `CLI` and `RiscZeroVerifier` modules was not within the scope of this security assessment.

Code Assessment. The Blobstream Zero developers provided the source code of the Blobstream Zero contracts for the code review. The source code appears to be mostly original code written by the Blobstream Zero developers. It contains some documentation in the form of documentation comments on functions and storage variables.

To facilitate the Veridise security analysts understanding of the code, the Blobstream Zero developers shared their internal design documents with a brief explanation of the project intent.

[1] `https://docs.celestia.org/developers/blobstream`

The source code included a single test case that tested the functionality of data proof generation, inclusion, and attestation.

Summary of Issues Detected. The security assessment uncovered 4 issues, 1 of which is assessed to be of high severity by the Veridise analysts. Specifically, V-BLOB-VUL-001 identifies a griefing attack that can render the whole protocol nearly useless by front-running the light-client on-chain updates by a motivated attacker. The Veridise analysts also identified 2 warnings, and 1 informational finding . The Blobstream Zero developers promptly responded on all the issues.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the Blobstream Zero:

- Extend the test suite to include unit tests and negative end-to-end tests. Currently, only the sunny-day behavior is being tested.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Blobstream Zero	`925bff9`	Solidity, Rust	Ethereum, RISC Zero zkVM

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Sep. 9–Sep. 13, 2024	Manual	2	10 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	1	1	1
Medium-Severity Issues	0	0	0
Low-Severity Issues	0	0	0
Warning-Severity Issues	2	0	0
Informational-Severity Issues	1	1	1
TOTAL	4	2	2

Table 2.4: Category Breakdown.

Name	Number
Logic Error	1
Access Control	1
Data Validation	1
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of Blobstream Zero smart contracts, the light client guest program, and the helpers in `primitives/` directory. During the assessment, the security analysts aimed to answer questions such as:

- Can an invalid proof be accepted on-chain?
- Can proofs be spoofed/replayed?
- Can signatures from unbonded validators be incorrectly accepted?
- Are all untrusted block headers appropriately verified by the guest program, including the requested untrusted block *and* all other intermediate block headers?
- Can the on-chain component be blocked to appropriate updating of processed blocks?
- Are appropriate checks in place to ensure the state on-chain never goes backwards (i.e., the latest block height never decreases), and progresses with each update?
- Are appropriate access control checks in place?
- Is there a centralized user role that gives too much power over the protocol?
- Can the liveness of the protocol be broken?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved intense scrutiny of the code by the security analysts.

Scope. The scope of this security assessment is limited to the code in the following folders:

- `contracts/`
- `light-client-guest/`
- `primitives/`

Methodology. Veridise security analysts reviewed the provided documentation for Blobstream Zero, inspected the provided tests, and read over resources for related projects. They then began a review of the code.

During the security assessment, the Veridise security analysts met with the Blobstream Zero developers to share findings and ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3.

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-BLOB-VUL-001	Unrestricted on-chain updates can . . .	High	Fixed
V-BLOB-VUL-002	Centralization Risk	Warning	Intended Behavior
V-BLOB-VUL-003	Insufficient input sanitization	Warning	Intended Behavior
V-BLOB-VUL-004	Typos, incorrect comments, and . . .	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-BLOB-VUL-001: Unrestricted on-chain updates can lead to problems

Severity	High	**Commit**	925bff9
Type	Logic Error	**Status**	Fixed
File(s)	`contracts/src/Blobstream0.sol`		
Location(s)	updateRange()		
Confirmed Fix At	`https://github.com/risc0/blobstream0/pull/49`		

The `updateRange()` function ingests range commitment information from the light client guest program and uses it to update information about trusted blocks on chain. The function has no access controls, meaning any user can call it. It requires the input to have been generated by the light client guest program (along with some other basic checks on the input data). This opens up the possibility of an attack which can either force well-behaving light clients to update very frequently or slows the updating of blocks on-chain. To better understand, consider the following example.

Example Assume Celestia's block time is 12 seconds, and well-behaving light clients aim to refresh on-chain data every 4 hours. Consequently, they call `updateRange()` on-chain with the necessary proof and data to include all 1200 blocks
(`5 blocks a minute x 60 minutes per hour x 4 hours = 1200 blocks`). If an attacker, Alice, observes these calls and preemptively submits updates (e.g., front-running), she can disrupt the sequence. For instance, if Alice intercepts an update to block 1200 with her own update to block 1, the legitimate update fails due to the hash mismatch. This strategy can be repeated, blocking subsequent valid updates by advancing the block sequence ahead of the well-behaving light client's attempts.

Impact One impact of this is that the on-chain tracking of blocks could start to get significantly behind the actual latest blocks, rendering the whole protocol useless for the protocols relying on this information.

Another possible impact would be if an attacker delayed block updates beyond 2 weeks, exceeding the trusting period. In such a case, invalid signatures might be accepted due to trust checks against the latest untrusted block's timestamp. Specifically, Alice could delay the chain's current block to over 2 weeks old. Assuming this block, B_1, has timestamp t and validators v_1, v_2, v_3, Alice can then submit a block B_x with a forged timestamp of $t + P - 1$ (P being the trusting period) and validators v_1, v_2, v_3. Since the timestamp of B_x falls within B_1's trusting period based on the spoofed timestamp, it might wrongfully be accepted, assuming $2/3$ of validators v_1, v_2, v_3 are reliable, despite them being untrustworthy due to the real time exceeding their trusting period

Recommendation There are several ways that developers could address this issue:

1. Make the `updateRange()` function callable only by trusted light clients. This would prevent malicious actors from making updates. However, this would make the system more centralized.

2. Require larger range updates. For example, if blocks update every 12 seconds and light clients update every 4 hours, we could require submitting at least 1200 new blocks on each update. This could prevent the attacks described, but it would prevent more granular updates, which may be useful in some cases. It would also require an adjustable limit if block times or light client behavior change (for example, if the block time on Celestia increased to 20 seconds, the limit would need to be adjusted on-chain accordingly).
3. Manually update the range in cases where an attack is detected. In the worst case, this would mean the attacker forcing updates to every single block, which would be expensive and require careful manual monitoring.

Developer Response The developers acknowledged the issue and have added a minimum amount for range updates.

4.1.2 V-BLOB-VUL-002: Centralization Risk

Severity	Warning	**Commit**	925bff9
Type	Access Control	**Status**	Intended Behavior
File(s)	`contracts/src/Blobstream0.sol`		
Location(s)	N/A		
Confirmed Fix At	N/A		

Similar to many projects, the `Blobstream0` contract declares an administrator role that is given special permissions. In particular, these administrators are given the following abilities:

- Set the current trusted state.
- Update the image ID used to identify the light client guest program.
- Set the verifier contract.

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, a malicious owner could arbitrarily update the verifier which would allow them to then add arbitrary Merkle roots.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or a multi-signature wallet as opposed to a single account, which introduces a single point of failure.

Developer Response The developers plan to use a time-locked multi-signature wallet to avoid this concern.

4.1.3 V-BLOB-VUL-003: Insufficient input sanitization

Severity	Warning	**Commit**	925bff9
Type	Data Validation	**Status**	Intended Behavior
File(s)	`light-client-guest/guest/src/main.rs`		
Location(s)	main.rs		
Confirmed Fix At	N/A		

The light client guest program reads in inputs corresponding to the last trusted block, the latest untrusted block, and all of the block headers in between. If the number of headers is very large, the computation will be slow and might eventually even panic.

Impact Large inputs with many intermediate headers may slow the light client guest program significantly.

Recommendation Add checks to immediately reject inputs that are too large.

Developer Response The developers acknowledged the issue, but have indicated that it is an intended behavior.

4.1.4 V-BLOB-VUL-004: Typos, incorrect comments, and unused program constructs

Severity	Info	**Commit**	925bff9
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	cbd8a06		

Description In the following locations, the auditors identified minor typos and/or potentially misleading comments:

- `contracts/src/Blobstream0.sol`:
  - The comment for the `imageId` field indicates that the guest program "(checks) if a number is even". This is not what the guest program does for this application — we suspect this comment was copied from another program.
  - In the function `updateRange()`, the input bytes `_commitBytes` are decoded into a `RangeCommitment` struct which is then encoded back into the bytes `journal` before getting passed to `verifier.verify(seal, imageId, sha256(journal))`. The re-encoding to `journal` can be removed, instead just passing `sha256(commitBytes)` to the `verifier.verify()` call.

Impact These minor errors may lead to future developer confusion.

Developer Response The developers implemented a fix for this issue.