Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

RIBBON

Aevo Governance

Veridise Inc.
June 19, 2023

► **Prepared For:**

Ribbon
https://www.ribbon.finance/

► **Prepared By:**

Benjamin Sepanski
Timothy Hoffman
Kostas Ferles

► **Contact Us:** contact@veridise.com

► **Version History:**

June 19, 2023	Initial Draft

Contents

1 Executive Summary

From Jun. 5, 2023 to Jun. 9, 2023, Ribbon engaged Veridise to review the security of their Aevo Governance project. The review covered the Smart Contracts intended for on-chain migration from the `RBN` token to the `AEVO` token. Veridise conducted the assessment over 3 person-weeks, with 3 engineers reviewing code over 1 weeks on commit `0x3470fb69`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Aevo Governance developers provided the source code of the Aevo Governance contracts for review. The code implements a 1-to-1 migration from `RBN` tokens to `AEVO` tokens, with handling to prevent locked funds in cases where users may improperly attempt to migrate `RBN` tokens.

To facilitate the Veridise auditors' understanding of the code, the Aevo Governance developers provided a brief description of the intended functionality. The source code also contained some documentation in the form of documentation comments on functions and storage variables. The source code contained a test suite, which the Veridise auditors noted contains 100% code coverage of both lines and branches.

Veridise auditors found the overall quality of code and documentation to be high.

Summary of issues detected. The audit uncovered 3 issues, 0 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, the Veridise auditors identified one low-severity issue regarding the possibility of mistaken users accidentally locking funds (V-RBN-VUL-001), as well as a few other minor issues. The Aevo Governance developers fixed all issues.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Aevo Governance. Each of these is described in the recommended fix for the issue. The largest suggestion pertained to V-RBN-VUL-001, which recommended adding an additional rescue function. This recommendation has been implemented by the developers.

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
Aevo Governance	`0x3470fb69`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jun. 5 - Jun. 9, 2023	Manual & Tools	3	3 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	0	0
Low-Severity Issues	1	1
Warning-Severity Issues	1	1
Informational-Severity Issues	1	1
TOTAL	3	3

Table 2.4: Category Breakdown.

Name	Number
Locked Funds	1
Gas Optimization	1
Maintainability	1

Audit Goals and Scope 3

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Aevo Governance's smart contracts. In our audit, we sought to answer the following questions:

- Can RBN tokens be duplicated during migration?
- Is it possible for RBN tokens to be locked during migration?
- Can RBN tokens be destroyed during migration?
- What common errors need to be handled to prevent locking of funds?
- Are common Solidity best practices followed?
- Are any common Solidity vulnerabilities present?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

Scope. The scope of this audit is limited to the `contracts/` folder of the source code provided by the Ribbon developers, which contains the smart contract implementation of the Aevo Governance. Namely, the `AevoToken.sol` and `Migrator.sol` contracts.

Methodology. Veridise auditors inspected the provided tests and read the Aevo Governance documentation. They then began a manual audit of the code assisted by automated testing. During the audit, the Veridise auditors regularly met with the Aevo Governance developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

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

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

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

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-RBN-VUL-001	Transfer RBN to AEVO contract may lock funds	Low	Fixed
V-RBN-VUL-002	Transfer amount not validated	Warning	Fixed
V-RBN-VUL-003	Using Deprecated Function	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-RBN-VUL-001: Transfer RBN to AEVO contract may lock funds

Severity	Low	**Commit**	3470fb5
Type	Locked Funds	**Status**	Fixed
File(s)	`contracts/AevoToken.sol`		
Location(s)	Aevo		

The possibility of users mistakenly sending `RBN` to the `Migrator` contract is handled by its `rescue()` function. This function enables the owner of the `Migrator` contract to handle these mistakes on a case-by-case basis.

```
function rescue() external {
    uint256 amount = RBN.balanceOf(address(this));

    RBN.safeTransfer(owner(), amount);

    emit Rescued(amount);
}
```

Snippet 4.1: Snippet from `Migrator.sol`:

However, the analogous possibility of users mistakenly sending `RBN` directly to the `Aevo` contract is not handled.

Impact `RBN` sent to the `Aevo` contract would be lost/locked.

Recommendation Provide a `rescue()` method in the `AevoToken.sol` that is like the one in `Migrator.sol`.

Developer Response The recommendation has been applied.

4.1.2 V-RBN-VUL-002: Transfer amount not validated

Severity	Warning	**Commit**	3470fb5
Type	Gas Optimization	**Status**	Fixed
File(s)	`contracts/Migrator.sol`		
Location(s)	rescue()		

In the `Migrator` contract, the `safeTransfer` function is called without first validating that the second parameter is a non-zero amount. Calling `safeTransfer(_,0)` has no meaningful outcome, thus calling it incurs unnecessary gas cost.

```
function rescue() external {
    uint256 amount = RBN.balanceOf(address(this));

    RBN.safeTransfer(owner(), amount);

    emit Rescued(amount);
}
```

Snippet 4.2: Snippet from `Migrator.sol`:

Impact Calling `safeTransfer` with `amount <= 0` results in wasted gas cost.

Allowing 0-value transactions also makes the token susceptible to phishing attacks. At the very least, it can make vigilant users wary to interact with the contract address in the future. See, for example, https://info.etherscan.com/zero-value-token-transfer-attack/.

Recommendation Insert `require(amount > 0, "!amount");` before calling `safeTransfer`.

Developer Response The recommendation has been applied.

4.1.3 V-RBN-VUL-003: Using Deprecated Function

Severity	Info	Commit	3470fb5
Type	Maintainability	Status	Fixed
File(s)	contracts/AevoToken.sol		
Location(s)	constructor()		

AevoToken.sol uses the _setupRole() function inherited from OpenZeppelin's AccessControl.sol contract. However, the _setupRole() function is deprecated, in favor of _grantRole().

```
constructor(
    string memory name,
    string memory symbol,
    address beneficiary
) ERC20Permit(name) ERC20(name, symbol) {
    // Add beneficiary as minter
    _setupRole(MINTER_ROLE, beneficiary);
        // Add beneficiary as admin
    _setupRole(ADMIN_ROLE, beneficiary);
      // ...
```

Snippet 4.3: Snippet from AevoToken.sol constructor.

```
* NOTE: This function is deprecated in favor of {_grantRole}.
*/
function _setupRole(bytes32 role, address account) internal virtual {
  _grantRole(role, account);
}
```

Snippet 4.4: Snippet from AccessControl.sol in version 4.9.0 of OpenZeppelin

Impact Future versions may require additional changes to work correctly with updated versions of OpenZeppelin.

Recommendation Use _grantRole() in place of _setupRole().

Developer Response The recommendation has been applied.

Fuzz Testing 5

5.1 Methodology

Our goal was to fuzz test Aevo Governance to ensure it implemented the ERC20 standard correctly, and check for created/destroyed tokens during migration. We used OrCa as our fuzzer and wrote invariants–logical formulas that should hold after every transaction. We then encoded those invariants as assertions in [V].

5.2 Properties Fuzzed

Table 5.1 describes the invariants we fuzz-tested. The first column states which component (i.e. `AevoToken` or `Migrator`) the invariant is associated with. The second describes the invariant informally in English, and the third shows the total amount of compute time spent fuzzing this property. The last column notes whether we found a bug when fuzzing the invariant (✗ indicates no bug was found and ✓ means fuzzing this invariant revealed a bug).

The first 11 specifications come from OrCa's built-in specification library. These define the ERC20 standard in the [V] language (including the extension for mintable ERC20s).

The Veridise auditors devoted a total of 19.5 compute-hours to fuzzing this protocol. No bugs were uncovered, providing additional evidence to the soundness of this code.

Table 5.1: Invariants Fuzzed.

Component	Invariant	Time	Bug
`AevoToken`	`transfer` reverts if a user attempts to send more funds than they have	1.5 hrs	✗
`AevoToken`	Funds should be successfully transferred from sender to `to`	1.5 hrs	✗
`AevoToken`	`transfer` should only modify expected variables	1.5 hrs	✗
`AevoToken`	`transferFrom` should revert when funds are unavailable	1.5 hrs	✗
`AevoToken`	`transferFrom` should not fail unexpectedly	1.5 hrs	✗
`AevoToken`	`transferFrom` should not modify unexpected values	1.5 hrs	✗
`AevoToken`	`approve` functional correctness	1.5 hrs	✗
`AevoToken`	`increaseAllowance` functional correctness.	1.5 hrs	✗
`AevoToken`	`decreaseAllowance` functional correctness.	1.5 hrs	✗
`AevoToken`	`mint` will increase `totalSupply` and a user's balance as expected	1.5 hrs	✗
`AevoToken`	`mint` will not modify another user's balance or any allowances	1.5 hrs	✗
`Migrator`	Migration amount cannot exceed `RBN` balance	1.5 hrs	✗
`Migrator`	Migration results in a 1:1 transfer of `RBN` to `AEVO`	1.5 hrs	✗

Glossary

ERC20 Fungible token standard. Read more at `https://ethereum.org/en/developers/docs/standards/tokens/erc-20/`. 11

Smart Contract Usually in reference to programs stored and executed on a blockchain, or other public verifiable medium. See `https://en.wikipedia.org/wiki/Smart_contract` to learn more. 1

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 5