SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

REDDIO

Overview

Project Summary

- Name: Reddio - GovernanceToken
- Platform: EVM-compatible chains
- Language: Solidity
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Reddio - GovernanceToken
Version	v3
Type	Solidity
Dates	May 23 2025
Logs	Apr 09 2025; May 21 2025; May 23 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	2
Total Low-Severity issues	0
Total informational issues	2
Total	4

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Signatures can be re-exploited	Medium	Business logic	Resolved
2	Centralization risk	Medium	Centralization	Acknowledged
3	Missing two-step transfer ownership pattern	Informational	Code Quality	Resolved
4	Use of floating pragma	Informational	Configuration	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Signatures can be re-exploited	
Severity: Medium	Category: Business logic
Target: - GovernanceToken.sol	

Description

The `GovernanceToken` contract inherits the `ERC20Votes` contract from the openzeppelin library. Prior to version 4.7.3, `ECDSA.tryRecover()` called by the `delegateBySig()` function in the `ERC20Votes` contract accepted both legacy-format signatures and EIP2098-format signatures, so it was possible to pass in the same signature in different formats, leading to signature replay problems.

Recommendation

It is recommended to use the `OpenZeppelin` version greater than `4.7.3`.

Status

This issue has been resolved by the team.

2. Centralization risk	
Severity: Medium	Category: Centralization
Target: - GovernanceToken.sol	

Description

GovernanceToken.sol:L16 - L26

```
constructor(address initialOwner)
    ERC20("Reddio", "RDO")
    ERC20Permit("Reddio")
    Ownable(initialOwner)
{
    _mint(initialOwner, TOTAL_SUPPLY);
}

function mint(address _account, uint256 _amount) public onlyOwner {
    _mint(_account, _amount);
}
```

The `GovernanceToken` contract contains an `owner` privileged address. The `owner` has the right to mint a number of tokens for any address. When the contract is deployed, $RDO is sent to the owner. This account then has full control over the token distribution.

If the private key of the `owner ` address is compromised, an attacker can create a large number of tokens for himself, thus destroying the normal price of the tokens.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Missing two-step transfer ownership pattern	
Severity: Informational	Category: Business logic
Target: - GovernanceToken.sol	

Description

The `GovernanceToken` contract inherits from the `Ownable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2Step contract from OpenZeppelin instead.

Status

This issue has been resolved by the team.

4. Use of floating pragma	
Severity: Informational	Category: Configuration
Target: - GovernanceToken.sol	

Description

```
pragma solidity ^0.8.20;
```

All contracts use a floating compiler version `^0.8.20`.

Using a floating pragma `^0.8.20`statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been resolved by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
GovernanceToken.sol	a6411abd8f484a03eae44fc11c52ec3803222ae5

And we audit the fix version covered the following files:

File	SHA-1 hash
GovernanceToken.sol	9a206af9dd77f14ace01deacd32b282a346a23db
package.json	48c2eb189480fbf0a5c22cd7ba9a37f51997ffc5

We audited the file that introduced new features:

File	SHA-1 hash
GovernanceToken.sol	c16306a5625ab360f7df143324644dadab92dc90