RESOLV TREASURY EXTENSION SECURITY AUDIT REPORT

Dec 26, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

Resolv is a protocol that manages USR, a stablecoin pegged to the USD, backed by ETH and hedging with short perpetual futures positions. LidoTreasuryExtension is a contract to wrap and unwrap stETH and wstETH in Treasury.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Resolv
Project name	Treasury Extension
Timeline	16.12.2024 - 16.12.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
16.12.2024	0a249f5bc3d76ed7d27551c76d7cba8e6cee9315	Commit for the audit

Project Scope

The audit covered the following files:

File name	Link
contracts/LidoTreasuryExtension.sol	LidoTreasuryExtension.sol

Deployments

File name	Contract deployed on mainnet	Comment
LidoTreasuryExtension.sol	0x7b10c5...ace5c660	

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	0
Low	0

ID	Name	Severity	Status

1.6 Conclusion

Resolv is a protocol that manages USR, a stablecoin pegged to the USD and backed by ETH, while hedging with short perpetual futures positions. The `LidoTreasuryExtension` contract provides a mechanism to wrap and unwrap Lido's stETH into wstETH, allowing the treasury to manage its stETH positions more flexibly. During the audit, we examined the contract's logic, focusing on safe handling of ERC20 tokens and interactions with external protocols, especially given the nature of stETH as a rebaseable token.

We also went through our detailed checklist, covering other aspects such as business logic, integer overflows, reentrancy attacks, access control, typecast pitfalls, rounding errors and other potential issues.

No vulnerabilities were found.

Some of the considered attack vectors:

1. **Rebaseable token behavior (stETH):**
   Lido's stETH is a rebasing token whose balance may change over time without direct transfers, due to periodic supply adjustments ("rebases"). A common attack vector in protocols integrating rebasing tokens is incorrect accounting: if a contract stores a stETH balance and assumes it remains static between actions, it can lead to mismatches, incorrect state assumptions, or unintended withdrawals.

   The `LidoTreasuryExtension` contract relies on the underlying Lido logic of shares rather than stable absolute balances. It utilizes `LIDO.getSharesByPooledEth()` and `LIDO.transferShares()` instead of directly manipulating stETH balances. This share-based accounting method ensures correct handling of stETH, mitigating potential issues arising from rebasing. Since the contract does not rely on cached balances, sudden changes due to rebases will not lead to state inconsistencies or exploitable conditions.

2. **Incorrect wrapping/unwrapping procedures:**
   Another vector could arise if the wrapping and unwrapping processes were not correctly understood: wrapping stETH produces different amount of wstETH, to wrap and unwrap tokens allowances must be set.

   The functions `wrap()` and `unwrap()` strictly follow a safe pattern:

   - Before transferring tokens, allowances are carefully set via `increaseAllowance()`.
   - `safeTransferFrom()` and `safeIncreaseAllowance()` are used to prevent unexpected token behavior.
   - Wrapping and unwrapping rely on well-established wstETH methods.

   The contract avoids common pitfalls such as stale allowances, unexpected token behavior, or miscalculations of wrapped token amounts.

3. **External contract calls and reentrancy attacks:**
   Another broad category of attacks involves reentrancy—when external calls to non-trusted contracts can

lead to nested calls that manipulate state inconsistently.

The contract's external interactions are limited and follow established safe patterns. Token calls are performed with `SafeERC20`, which reverts on failures. The contract also strictly uses role-based access control via `onlyRole` modifiers. This prevents unauthorized calls and reduces the attack surface for reentrancy. Since Lido's stETH and wstETH are well-known and widely audited tokens, the risk of unexpected callbacks is minimal.

4. **Business Logic and Access Control Validation:**
   We also reviewed the logic for setting and updating the treasury address, performing emergency withdrawals, and ensuring that the contract only accepts instructions from authorized roles.

After examining the full code and integrating considerations for rebasing mechanics, external Lido interactions, and potential attack vectors, no vulnerabilities or logic flaws were identified. The contract properly accounts for the nuances of stETH by using Lido's share-based system and carefully follows best practices for ERC20 interactions. Its strong access control, safe token handling, and correct use of the Lido ecosystem functions ensure that known attack vectors related to rebaseable tokens and Lido integrations are not applicable.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

Not Found

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes