RESOLV TREASURY SECURITY AUDIT REPORT

Nov 04, 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 US Dollar and natively backed by Ether (ETH). The protocol's key features include issuing and redeeming USR in exchange for other tokens, ensuring continuous backing by ETH through hedging with short perpetual futures positions, and maintaining the Resolv Liquidity Pool (RLP), a liquid insurance pool designed to keep USR overcollateralized. Users can mint and redeem both USR and RLP by depositing collateral on a 1:1 basis.

Users can stake USR into stUSR to earn yield. stUSR is a rebasable yield token that can be wrapped into non-rebaseable wstUSR.

RLP minters can earn profits by burning RLP for more collateral than initially provided. The RLP token serves as insurance for the protocol and offers the potential of higher profits though RLP holders face the risk of losses due to less favorable burn rates.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Resolv
Project name	Treasury
Timeline	27.08.2024 - 06.09.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
27.08.2024	2357c0eb348ff87acf240e96c288af6b8cf646e3	Commit for the audit
06.09.2024	65f664227802dfe3fd112cee9fb307c2c9113f20	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
contracts/Treasury.sol	Treasury.sol
contracts/AaveV3TreasuryConnector.sol	AaveV3TreasuryConnector.sol
contracts/LidoTreasuryConnector.sol	LidoTreasuryConnector.sol
contracts/LPExternalRequestsManager.sol	LPExternalRequestsManager.sol

File name	Link
contracts/ExternalRequestsManager.sol	ExternalRequestsManager.sol

Deployments

Deployment verification has not been conducted as there were protocol changes post-audit that were not reviewed because they were urgent and minor.

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	3
Low	4

ID	Name	Severity	Status
M-1	No slippage protection in `LPExternalRequestsManager.requestBurn()`	Medium	Acknowledged
M-2	Funds locked due to the whitelist	Medium	Fixed
M-3	Mismatched token validation in `LPExternalRequestsManager._completeBurn()`	Medium	Fixed
L-1	Unnecessary `safeIncreaseAllowance` call in `Treasury.aaveBorrow()`	Low	Fixed
L-2	Optimization for `LidoTreasuryConnector.deposit()`	Low	Acknowledged
L-3	Incompatibility with fee-on-transfer or rebasing tokens	Low	Acknowledged
L-4	Low-value transactions	Low	Acknowledged

1.6 Conclusion

The codebase is of high quality, with solid test coverage and an organized structure.

Most of the functions are centralized and have access control; only users who have been whitelisted can interact with a few of them.

Key notes:

- If a provider creates a request to mint or burn stablecoins for a specific collateral, and that collateral is subsequently disabled by an admin, the service role can still complete the minting and burning of existing requests for that token.

Key recommendations:

1. We recommend using immutable addresses that can be set in the constructor instead of hardcoded constants, as the protocol might be deployed on other chains in the future, and errors could occur due to human error.
2. The **Treasury** interacts with connectors via the `safeIncreaseAllowance()` -> `safeTransferFrom()` pattern. We recommend using `safeTransfer()` as a more gas-efficient approach.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	No slippage protection in `LPExternalRequestsManager.requestBurn()`
Severity	Medium
Status	Acknowledged

Description

- LPExternalRequestsManager.sol#L215-L218

`LPExternalRequestsManager.requestBurn()` allows users to burn RLP tokens, but providers do not have any slippage protection — the amounts of tokens to transfer in `LPExternalRequestsManager.completeBurns()` are not related to any on-chain oracle and are susceptible to slippage: request providers may receive fewer tokens than they expected.

Recommendation

We recommend adding the `minAmountOut` and `deadline` parameters to the workflow.

Client's commentary

> We agree that while the lack of slippage protection (minAmountOut and deadline) presents a potential risk, the economic design of RLP tokens already accounts for dynamic changes in collateral ratios (CR) and market conditions. Since the protocol operates with "partial complete" logic based on these variables, guaranteeing specific amounts or timing isn't feasible without compromising the system's

> flexibility. Users are informed of slippage risks as part of the RLP's economic structure, reducing the practical impact of this issue.

M-2	Funds locked due to the whitelist
Severity	Medium
Status	Fixed in 65f66422

Description

- LPExternalRequestsManager.sol#L346

The `onlyAllowedProviders` check is applied in `LPExternalRequestsManager.withdrawAvailableCollateral()`. If a provider is delisted after their request has been added to the `processBurns()` or `completeBurns()` operations, they will not be able to withdraw their funds.

This situation can occur if the whitelist has been inactive and is later activated via `setWhitelistEnabled()`.

Recommendation

We recommend removing the `onlyAllowedProviders` check in `withdrawAvailableCollateral()`.

Client's commentary

> a117f789

M-3	Mismatched token validation in `LPExternalRequestsManager._completeBurn()`
Severity	Medium
Status	Fixed in 65f66422

Description

- LPExternalRequestsManager.sol#L425

The `_completeBurn()` method in the `LPExternalRequestsManager` contract does not contain a validation check to ensure that `_item.withdrawalToken` is equal to `request.token`. This vulnerability could lead to a scenario where the off-chain backend mistakenly provides an incorrect `_item.withdrawalToken`. If such a mistake occurs, the `withdrawAvailableCollateral()` method may function incorrectly, potentially leading to unintended behavior or loss of funds.

Recommendation

We recommend ensuring that `_item.withdrawalToken` matches `request.token`.

Client's commentary

> 70b24702

2.4 Low

L-1	Unnecessary `safeIncreaseAllowance` call in `Treasury.aaveBorrow()`
Severity	Low
Status	Fixed in 65f66422

Description

- Treasury.sol#L319

When borrowing with the `Treasury.aaveBorrow()` function, there is an unnecessary `safeIncreaseAllowance()` call.

Recommendation

We recommend removing this call.

Client's commentary

> 843f7723

L-2	Optimization for `LidoTreasuryConnector.deposit()`
Severity	Low
Status	Acknowledged

Description

- LidoTreasuryConnector.sol#L52

The `deposit()` method in the `LidoTreasuryConnector` contract currently has a complex logic for obtaining `wstETH`, involving multiple steps and calculations. This process could be simplified to improve gas efficiency and avoid the accumulation of small ether amounts (dust) on the contract due to operations like `getPooledEthByShares()` and `getSharesByPooledEth()`.

The contract at address `0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0` (`WST_ETH_ADDRESS`) already includes the necessary functionality for acquiring `wstETH`:

```
receive() external payable {
    uint256 shares = stETH.submit{value: msg.value}(address(0));
    _mint(msg.sender, shares);
}
```

By directly sending ETH to this contract, the conversion to `wstETH` is handled automatically without additional steps.

Recommendation

We recommend interacting directly with the `WST_ETH_ADDRESS` contract for deposits.

Client's commentary

> The current implementation is intentional to retrieve the exact wstETHAmount within our SC for further processing, avoiding reliance on extracting parameters from external events. While the suggested change may reduce gas costs, it increases dependency on external systems and Lido contracts. Since this is an infrequent operation, gas cost is not a significant concern.

L-3	Incompatibility with fee-on-transfer or rebasing tokens
Severity	Low
Status	Acknowledged

Description

Some parts of the code operate under the assumption that the `safeTransfer()` and `safeTransferFrom()` functions will transfer an exact amount of tokens.

However, the actual amount of tokens received may be less for **fee-on-transfer** tokens. Moreover, the protocol doesn't accommodate **rebasing** tokens whose balances fluctuate over time.

These discrepancies can lead to transaction reverts or other unexpected behaviour.

Recommendation

We recommend implementing verification checks both before and after token transfers.

Client's commentary

> We acknowledge the note. Fee-on-transfer (FOT) and rebasing tokens are not supported by our protocol and would only be added with explicit admin approval. Additionally, the LPExternalRequestsManager is not designed to work with these token types. This may be revisited in the future if logic updates are required.

L-4	Low-value transactions
Severity	Low
Status	Acknowledged

Description

- LPExternalRequestsManager.sol#L144
- LPExternalRequestsManager.sol#L215
- ExternalRequestsManager.sol#L121
- ExternalRequestsManager.sol#L192

The provider may execute transactions to mint or burn small amounts of funds, resulting in a value that is lower than the profit generated for the protocol, or even negative if the gas fees are higher.

This can lead to some provider requests remaining unfulfilled or cause losses to the protocol.

Recommendation

We recommend implementing a minimum threshold for minting and burning stablecoins.

Client's commentary

> Noted; however, the protocol intentionally supports partial burns, which aligns with its design to ensure flexibility in fulfilling provider requests. While small amounts may lead to suboptimal outcomes for individual transactions, users are informed of the potential risks, including gas fees. Introducing a minimum threshold would contradict this flexibility and could complicate request processing. Therefore, while relevant, this issue poses a low risk to the overall functionality and remains within acceptable parameters.

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