RESOLV TREASURY ESCROW SECURITY AUDIT REPORT

March 14, 2025

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

The scope of this audit covers two smart contracts in the Resolv Treasury Escrow system:

1. **TreasuryIntermediateEscrow**: A contract that manages escrow entries for token deposits and payouts. It features:

   - Locking of deposits with minimum payout requirements
   - Release mechanisms with role-based access control
   - Support for both ERC20 tokens and ETH
   - Emergency withdrawal capabilities for admin

2. **UsrRedemptionExtension**: A contract handling USR token redemptions with:

   - Price-aware redemption mechanism using Chainlink oracles
   - Daily redemption limits
   - Configurable redemption fees
   - Integration with Aave for liquidity
   - Support for multiple withdrawal tokens

The system implements role-based access control with distinct roles for locking (LOCKER_ROLE), releasing (RELEASER_ROLE), and force releasing (FORCE_RELEASER_ROLE) escrow entries. The contracts are designed to work with an external Treasury contract for actual token movements.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Resolv
Project name	Resolv Treasury Escrow
Timeline	27.02.2025 - 14.03.2025
Number of Auditors	3

Project Log

Date	Commit Hash	Note
27.02.2025	91c52d9801c6a37f39566e2854722be1700dedc0	Commit for the audit
04.03.2025	bfdf74ba1b830866625bc2b7c2babf14aeea5fae	Commit for the re-audit
05.03.2025	6a0193386fd5d73ccb5a584c97e8987dcacc960a	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
contracts/UsrRedemptionExtension.sol	UsrRedemptionExtension.sol
contracts/TreasuryIntermediateEscrow.sol	TreasuryIntermediateEscrow.sol

Deployments

File name	Contract deployed on mainnet	Comment
contracts/TreasuryIntermediateEscrow.sol	0x84b883...F4aCFeD6	

1.5 Summary of findings

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

ID	Name	Severity	Status
L-1	`FORCE_RELEASER_ROLE` can cause underflow in `lockedBalances`	Low	Fixed
L-2	`RELEASER_ROLE` can transfer unlimited tokens from `_fromAddress` if they have granted unlimited approval to the `TreasuryIntermediateEscrow` contract	Low	Fixed
L-3	Limited Native ETH Association in TreasuryIntermediateEscrow	Low	Fixed

1.6 Conclusion

In this audit, we focused on two core components of the Resolv Treasury Escrow system: the escrow management functionality and the USR token redemption mechanism. Our security assessment concentrated on potential attack vectors specific to these features.

Key vectors examined for the TreasuryIntermediateEscrow:

- **Access control and role separation**. The contract implements proper role-based access control with distinct roles for locking (LOCKER_ROLE), releasing (RELEASER_ROLE), and force releasing (FORCE_RELEASER_ROLE). However, the FORCE_RELEASER_ROLE has unrestricted capabilities that could lead to accounting issues.
- **Asset tracking**. The system tracks locked balances for both ERC20 tokens and ETH. While the accounting logic is generally sound, there are potential issues with ETH tracking due to limited association mechanisms.
- **Emergency controls**. Emergency withdrawal functions are properly restricted to admin role and include appropriate validation checks.

Key vectors examined for the UsrRedemptionExtension:

- **Price manipulation**. The contract uses Chainlink oracles with appropriate heartbeat checks and price validation to prevent price manipulation attacks.
- **Redemption limits**. Daily redemption limits are implemented with proper reset logic to prevent excessive redemptions.
- **Asset security**. The contract interacts safely with external tokens and the treasury, using SafeERC20 and proper validation checks.

We also went through our detailed checklist covering other aspects such as business logic, common ERC20 issues, interactions with external contracts, integer overflows, reentrancy attacks, access control, and rounding errors.

No significant issues were found.

Key notes and recommendations:

- We recommend paying special attention to the backend generation of the `idempotencyKey` to mitigate any potential collisions. In case of a collision, the `redeem` function in the `UsrRedemptionExtension` contract may fail to execute.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	`FORCE_RELEASER_ROLE` can cause underflow in `lockedBalances`
Severity	Low
Status	Fixed in bfdf74ba

Description

The `FORCE_RELEASER_ROLE` can call `TreasuryIntermediateEscrow.forceRelease()` with a `_payoutAmount` of zero multiple times. This leads to a decrease in the `lockedBalances` mapping for a specific asset by an amount greater than the `depositedAmount` of a given `_escrowId`. As a result, `lockedBalances` will decrease faster than expected and may underflow when the last locked entry is released.

TreasuryIntermediateEscrow.sol#L111

Recommendation

We recommend adding a restriction that `_payoutAmount` is not equal to zero in `TreasuryIntermediateEscrow.forceRelease()`.

Client's commentary

> PR-266

L-2	`RELEASER_ROLE` can transfer unlimited tokens from `fromAddress` if they have granted unlimited approval to the `TreasuryIntermediateEscrow` contract
Severity	Low
Status	Fixed in 6a019338

Description

The `TreasuryIntermediateEscrow.releaseFrom()` function allows any address with the `RELEASER_ROLE` to withdraw tokens from `_fromAddress` without explicit approval for each transaction if `_fromAddress` has previously granted unlimited approval to this contract. This can lead to potential fund loss for users who have given such approvals because `RELEASER_ROLE` can withdraw significantly more than expected.

TreasuryIntermediateEscrow.sol#L104

Recommendation

We recommend implementing EIP-712 signatures to require explicit user approval for each transfer instead of relying on `approve()`.

Client's commentary

> Client: PR-267
> MixBytes: The signature should include `escrowId` (alternatively, this logic could be completely replaced by simply transferring funds from `msg.sender`).

L-3	Limited Native ETH Association in TreasuryIntermediateEscrow
Severity	Low
Status	Fixed in bfdf74ba

Description

The TreasuryIntermediateEscrow contract supports handling native ETH. Currently, the workflow involves sending ETH to the contract followed by invoking the `release` / `forceRelease` functions. This approach does not provide a built-in mechanism to clearly associate the received ETH with a specific `escrowId`. As a result, roles like RELEASER_ROLE and FORCE_RELEASER_ROLE might face challenges in accurately identifying the source of the funds, which could lead to potential ambiguities during fund management.

TreasuryIntermediateEscrow.sol#L39

Recommendation

We recommend adding a dedicated payable function (for example, `receiveEth(_escrowId)`) that accepts an `escrowId` as an argument. This enhancement would explicitly link incoming ETH to its respective `escrowId`.

Client's commentary

> PR-268

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