RESOLV SECURITY AUDIT REPORT

Jun 14, 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.

USR is created by depositing liquid assets like USDC, USDT, or ETH on a 1:1 value basis. Users can stake USR into stUSR to earn yield. stUSR is a rebasable yield-generating wrapper on USR.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Resolv
Project name	Resolv
Timeline	May 14 2024 - June 07 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
14.05.2024	a36e73c4be0b5f233de6bfc8d2c276136bf67573	Commit for the audit
24.05.2024	a1575cdc00cf04cc1f4344f5db268670c093dc2b	Commit for the re-audit
10.06.2024	294168806fe2d24add6b6dd4f0c431e00009c65c	Commit from the 29416880

Project Scope

The audit covered the following files:

File name	Link
contracts/RewardDistributor.sol	RewardDistributor.sol
contracts/AddressesWhitelist.sol	AddressesWhitelist.sol
contracts/beta/ExternalRequestsManagerBetaV1.sol	ExternalRequestsManagerBetaV1.sol

File name	Link
contracts/SimpleToken.sol	SimpleToken.sol
contracts/StUSR.sol	StUSR.sol
contracts/ERC20RebasingPermitUpgradeable.sol	ERC20RebasingPermitUpgradeable.sol
contracts/ERC20RebasingUpgradeable.sol	ERC20RebasingUpgradeable.sol

Deployments

File name	Contract deployed on mainnet	Comment
TransparentUpgradeableProxy.sol	0x66a1e3...ecf3e110	Proxy for USR; OpenZeppelin 5.0.1
TransparentUpgradeableProxy.sol	0x4956b5...e4528f96	Proxy for RLP; OpenZeppelin 5.0.1
TransparentUpgradeableProxy.sol	0x6c8984...7d10AAb4	Proxy for stUSR; OpenZeppelin 5.0.1
SimpleToken.sol	0xef4c4b...97a1fabe	Implmentation for USR and RLP
StUSR.sol	0xba1600...45973da7	
AddressesWhitelist.sol	0x594302...BA311255	
ExternalRequestsManagerBetaV1.sol	0x3Ed5Dc...AF9BD45D	
ExternalRequestsManagerBetaV1.sol	0x052B1c...F60FDA1B	
RewardDistributor.sol	0xbE23BB...189c1bA9	

1.5 Summary of findings

Severity	# of Findings
Critical	1
High	0
Medium	4
Low	0

ID	Name	Severity	Status
C-1	StUSR Inflation Attack	Critical	Fixed
M-1	No Slippage Protection	Medium	Fixed
M-2	Funds Locked Due to the Whitelist	Medium	Fixed
M-3	StUSR Transfers Less Than Expected	Medium	Fixed
M-4	StUSR Yield Stealing	Medium	Acknowledged

1.6 Conclusion

The project comprises two main components: smart contracts and an off-chain part. Both components are crucial for the project's overall functionality. This audit specifically focused on the smart contracts, while the off-chain part was outside the scope.

We audited the contracts related to RLP and USR token operations. Our audit included the management and minting of tokens, as well as the distribution of rewards and staking.

Key activities included:

- Checking the transaction flow for minting and burning, managing storage for requests, their statuses, and finalization
- Testing scenarios of setting different smart-contract interconnection, resetting roles
- Ensuring role management flow is secure and doesn't allow risky setups
- Verifying calculations for rebasable math, testing approval management, calculation of amounts used for transfers and approvals adjusted to rebasable specifics
- Ensuring conversions between shares and balances are correct, testing scenarios for potential wrong roundings
- Checking usage of permits including permit implementations for project tokens
- Testing the impact of tokens with the unusual design

Key Observations and Recommendations:

- **Centralization:** The entire architecture is centralized. Minting, burning, and collateral management are executed by controlled accounts, which requires strong trust in the project maintainers. In addition, admins can withdraw all funds from smart contracts. The system doesn't use oracles.

The audited code was published in the public repository: 29416880

2.FINDINGS REPORT

2.1 Critical

C-1	StUSR Inflation Attack
Severity	Critical
Status	Fixed in a1575cdc

Description

- ERC20RebasingUpgradeable.sol#L386-L398

The empty StUSR pool is vulnerable to an Inflation Attack despite having some protective measures, which can lead to the loss of funds for early depositors to the benefit of a hacker, as well as causing a DOS (Denial of Service).

StUSR has two protection mechanisms against an Inflation Attack:

1. A depositor cannot receive 0 shares.
2. The exchange rate is calculated with 1 virtual share and 1 virtual asset.

In practice, 1 virtual share is not sufficient to protect against a profitable attack.

Example of an attack:

1. Before an attack, the pool has 0 shares and 0 assets.
2. The hacker mints 1000 wei shares. Now the pool has 1000 wei shares plus 1 virtual share.
3. The hacker directly transfers `1,001,000 USR - 1001 wei` into the StUSR pool. At this point, the hacker holds 1000 shares worth 1000 USR each, while losing 1000 USR to the pool's 1 virtual share.
4. The first victim deposits 1999 USR and receives `shares = 1999e18 * 1001 / 1001000e18 = 1 wei`. The value of 1 share is now approximately 1001 USR. The victim loses about 1000 USR, which is distributed to the pool. Since the hacker owns 99.8% of the pool, most of the profit goes to the hacker. At this point, the hacker has almost recovered the cost of the attack.
5. The second victim deposits 1999 USR and receives `shares = 1999e18 * (1001 + 1) / (1002999e18 + 1) = 1 wei`. The value of 1 share is now approximately 1002 USR. Again, the victim loses about 1000 USR, mostly to the hacker's benefit. At this point, the hacker is in profit.

State of the contract for each step:

Step	`totalShares()+1`	`_totalUnderlyingTokens()+1`	1 wei share value	Hacker's total profit (approx.)
1	0	0		
2	1000 + 1	1000 + 1	1 wei	0
3	1000 + 1	1,001,000e18	1000 USR	-1000 USD
4	1001 + 1	1,002,999e18	~1001 USR	0
5	1002 + 1	1,004,998e18	~1002 USR	+1000 USD

Recommendation

We recommend using 1000 virtual shares.

Client's commentary

> Fixed with f8cb3222

2.2 High

Not Found

2.3 Medium

M-1	No Slippage Protection
Severity	Medium
Status	Fixed in a1575cdc

Description

- ExternalRequestsManagerBetaV1.sol#L234
- ExternalRequestsManagerBetaV1.sol#L175

The amounts of tokens to mint and transfer in `completeMint()` and `completeBurn()` in the `ExternalRequestsManagerBetaV1` are not related to any on-chain oracle and are prone to slippage: users may receive less tokens than they expected.

Recommendation

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

Client's commentary

> - The `minMintAmount` parameter was added 9c3999fe
> - The `deadline` parameter has not been added since users can cancel any unhandled requests at any time. Adding a deadline does not make practical sense and would only complicate the logic unnecessarily.

M-2	Funds Locked Due to the Whitelist
Severity	Medium
Status	Fixed in a1575cdc

Description

- ExternalRequestsManagerBetaV1.sol#L209
- ExternalRequestsManagerBetaV1.sol#L148

The `onlyAllowedProviders` check is applied in cancel operations. If a provider gets delisted after creating a pending request, they won't be able to cancel it.

This situation may arise when the whitelist was inactive and then got activated via `setWhitelistEnabled()`.

Recommendation

We recommend removing the `onlyAllowedProviders` check from `cancelMint()` and `cancelBurn()`.

Client's commentary

> Fixed with db9ca2a6

M-3	StUSR Transfers Less Than Expected
Severity	Medium
Status	Fixed in a1575cdc

Description

- ERC20RebasingUpgradeable.sol#L135

StUSR, when transferring, converts the underlying amount into shares with a rounding error in favor of the sender (i.e., downwards). This error can be significant if the exchange rate between shares and the underlying asset is inflated. Specifically, for some positive underlying amounts, the number of shares will be zero. In such cases, the `transfer()` function will be executed successfully, but the balances of the recipient and the sender will remain unchanged. A hacker could exploit this issue to attack third-party protocols, for example, if a lending protocol integrates StUSD as collateral.

A similar problem affects `transferFrom()`: the transferred allowance will be fully consumed even if the recipient receives zero shares (and consequently, zero underlying assets).

Recommendation

We recommend documenting this nuance, mentioning that during transfers using either `transfer()` or `transferFrom()`, the user may receive fewer funds than specified in the arguments, and the allowance will still be consumed. A more radical solution would be to abandon the rebasing token in favor of a regular ERC-4626 vault from OpenZeppelin's standard implementation.

Client's commentary

> Documented with natspec a1575cdc

M-4	StUSR Yield Stealing
Severity	Medium
Status	Acknowledged

Description

- RewardDistributor.sol#L50

Rewards in StUSR are vulnerable to yield stealing.

A hacker can sandwich the `RewardDistributor.distribute()` transactions from the public mempool:

1. In the first transaction, a hacker front-runs by staking a significant amount of USR in StUSR to become the primary stakeholder.
2. The transaction then occurs, distributing rewards in StUSR.
3. In the next transaction, the hacker back-runs by withdrawing their funds with a guaranteed profit.

This results in regular users losing the incentive to stake funds, as the hacker can capture all distributed rewards.

Recommendation

We recommend using private mempools for reward distribution transactions.

Client's commentary

> - Private mempools are utilized.
> - Given that the emission of new USR is available only to known and whitelisted users, and redemption is subject to an up to 24hr timelock, there is minimal opportunity for a malicious actor to access a "significant amount of USR" relative to the StUSR supply and significantly dilute the stakes of legitimate users.

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