RESOLV POR ORACLES 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

The project implements a Proof-of-Reserves (PoR) oracle system for integration with Morpho and Euler lending protocols. Project scope includes `UsrRedemptionExtension` contract, which implements a redemption mechanism that allows users to exchange USR tokens for supported withdrawal tokens (like USDC) at current market rates if USR to USD peg is maintained at the 1:1 ratio.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Resolv
Project name	PoR Oracles
Timeline	18.12.2024 - 20.12.2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
18.12.2024	f4a09fa74498b19708b056af6adc070e1272b59b	Commit for the audit
20.12.2024	77d8ed1d04e01c1f756470fe1cf2069a7667c2f3	Commit for the re-audit
20.12.2024	636519d48f52ac5f3471de1bead411743287725a	Commit with updates

Project Scope

The audit covered the following files:

File name	Link

File name	Link
contracts/RlpPriceStorage.sol	RlpPriceStorage.sol
contracts/UsrPriceStorage.sol	UsrPriceStorage.sol
contracts/UsrRedemptionExtension.sol	UsrRedemptionExtension.sol

Deployments

File name	Contract deployed on mainnet	Comment
UsrRedemptionExtension.sol	0xb69B2e...D8c96bE6	
TransparentUpgradeableProxy.sol	0xaE2364...9AF3574d	Proxy for RlpPriceStorage
RlpPriceStorage.sol	0x5e90b0...c98e1aA1	
TransparentUpgradeableProxy.sol	0x7f4518...999c261c	Proxy for UsrPriceStorage
UsrPriceStorage.sol	0xc16B2a...4fc2785d	

1.5 Summary of findings

Severity	# of Findings
Critical	1
High	0
Medium	0
Low	6

ID	Name	Severity	Status
C-1	Flaw in Redemption Price Calculation	Critical	Fixed
L-1	Improper Utilization of Approve Functionality via Permit Call	Low	Fixed
L-2	Redundant Check	Low	Fixed
L-3	Inflexible Hardcoded Decimals	Low	Fixed
L-4	Unsafe Data Type Casting	Low	Fixed
L-5	Unused Return Variable	Low	Fixed
L-6	Missing Validation in Constructor Parameters of `UsrRedemptionExtension`	Low	Fixed

1.6 Conclusion

During the audit, apart from checking well-known attack vectors and vectors that are presented in our internal checklist, we carefully investigated the next attack vectors:

1. The `redeem` function in `UsrRedemptionExtension` **maintains price stability** through a balanced mechanism: when a user redeems USR tokens, two synchronized actions occur:

   - The USR tokens are burned (`ISimpleToken(USR_TOKEN_ADDRESS).burn()`)
   - An equivalent value in withdrawal tokens is removed from the treasury (either from its balance or through Aave borrowing)

   The redemption amount is calculated using current Chainlink oracle prices and USR price from storage. This ensures that redemptions cannot manipulate the next price update in `UsrPriceStorage`.

2. **Reserves are correctly calculated**, including accounting of total borrowed assets in Aave and pending withdrawals in Lido and Dinero.

   The total reserves are computed through a detailed formula that accounts for both liquid and pending assets:

   - Ethereum-based assets:

     - Direct holdings (wstETH, stETH, ETH) with respective protocol exchange rates
     - Aave positions (supplied wstETH and borrowed WETH)
     - Lido positions (requested and claimable withdrawals)
     - Dinero positions (apxETH with exchange rate, initiated and redeemable redemptions)
       All ETH-denominated values are converted to USD using a weighted average futures price based on total short positions

   - Stablecoin positions:

     - Direct USDC and USDT holdings excluding Aave borrowed positions for both stablecoins
       Final values are converted using Pyth (Hermes) oracle prices

3. **Reserves value** can be increased by manipulation, but it doesn't make a lot of sense to increase reserves for manipulating the USR price because it is capped at $1, so it cannot be used in some type of economic attack. Reserves manipulation makes more sense for RLP, but its oracle has an upper bound, so a huge donation will lead only to DoS, which doesn't make a lot of sense for a hacker. So, we haven't found a profitable way for a hacker to manipulate the oracle prices, but it can be done for the RLP price till it reaches the upper bound, so it should be set considering this.

4. The `UsrRedemptionExtension` contract **implements a daily redemption cap** system that makes it impossible to exceed the redemption limit through a combination of time-based and amount-based checks.

The contract continuously tracks the time elapsed since `lastResetTime`, and when a full day has passed (`currentTime >= lastResetTime + 1 days`), it calculates the number of periods passed, updates the `lastResetTime`, and resets the `currentRedemptionUsage` counter to zero. Before each redemption, the contract adds the requested amount to `currentRedemptionUsage` and strictly enforces the cap by reverting with `RedemptionLimitExceeded` if the total would exceed `redemptionLimit`. This mechanism ensures that daily limits are properly reset and no redemption can push usage above the limit, maintaining accurate tracking.

5. **Support for stablecoins with varying decimals** is implemented in the `UsrRedemptionExtension` contract through a decimal adjustment system. When processing redemptions, the contract first retrieves and validates the withdrawal token's decimals using `IERC20Metadata(_withdrawalTokenAddress).decimals()`, ensuring they don't exceed `USR_DECIMALS`. The redemption amount calculation then accounts for these decimal differences by applying the appropriate scaling factor. Additionally, in the `getRedeemPrice` function, the contract handles Chainlink oracle price feeds with different decimal precisions by dynamically adjusting the price value, ensuring accurate price conversions regardless of the stablecoin's decimal configuration.

6. **System configuration changes** in the contracts are strictly **controlled** through **OpenZeppelin's AccessControl** implementation, ensuring that sensitive operations can only be executed by privileged users. All configuration functions are protected by role-based modifiers: `onlyRole(DEFAULT_ADMIN_ROLE)` is used for administrative operations like setting parameters, managing allowed tokens, and controlling system pauses, while `onlyRole(SERVICE_ROLE)` is applied to operational functions like price updates and redemptions. This dual-role system, implemented through inheritance from `AccessControlDefaultAdminRules`, provides a clear separation of administrative privileges, with the admin role having a mandatory time-delay of 1 day for role transfers.

2.FINDINGS REPORT

2.1 Critical

C-1	Flaw in Redemption Price Calculation
Severity	Critical
Status	Fixed in 77d8ed1d

Description

A critical issue was uncovered in the `redeem` function of the `UsrRedemptionExtension` contract. The flaw lies in the way the `redeemPrice` is determined. The current code fetches the `redeemPrice` as the cost of 1 USDC in USR, mistakenly multiplying this price by the USR amount, which leads to incorrect calculations and loss for the protocol or users. For instance, if the USDC market price rises to $1.03, for 100 burned USR, the user will receive 103 USDC, which at the new market price would be approximately $106. This flaw can lead to users receiving more than intended or less in cases of USDC price decrese. The issue is classified as of **critical** severity due to its potential for financial impact.

Recommendation

We recommend modifying the code to divide by `redeemPrice` instead of multiplying it, thereby ensuring the price calculation aligns with potential market fluctuations of USDC.

Client's Commentary

> 8038a8e3

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Improper Utilization of Approve Functionality via Permit Call
Severity	Low
Status	Fixed in 77d8ed1d

Description

A potential issue has been detected within the `redeemWithPermit` function of the `UsrRedemptionExtension` contract. The approach allows updating the approved amount via `permit` call, which is deemed unnecessary as tokens are burnt, not transferred from the receiver. It's more of functional redundancy than a security flaw, though highlights an area of coding optimization.

Recommendation

We recommend removing the `redeemWithPermit` function to improve code efficiency.

Client's Commentary

> a6651f62

L-2	Redundant Check
Severity	Low
Status	Fixed in 77d8ed1d

Description

An issue has been spotted in the `removeAllowedWithdrawalToken` function of the `UsrRedemptionExtension` contract. This function unnecessarily checks the validity of the `_allowedWithdrawalTokenAddress` twice, first through a modifier and then directly in the function body itself. While this does not pose a direct security risk, it is considered as a wasteful operation that could potentially increase gas cost.

Recommendation

We recommend removing the redundant security check for `_allowedWithdrawalTokenAddress`.

Client's Commentary

> 7eefefa5

L-3	Inflexible Hardcoded Decimals
Severity	Low
Status	Fixed in 636519d4

Description

The issue was found in the `getRedeemPrice` function of the smart contract. The presence of the hardcoded decimal value `18` has been detected, which suggests a reduced flexibility of the contract on different blockchain networks. This issue is classified as **low** severity, as it poses minimal immediate security threat. However, it does limit the contract's adaptability across various chains, potentially hindering its broad usage.

Recommendation

We suggest replacing the hardcoded decimal `18` with `USR_DECIMALS` across the contract.

Client's Commentary

> Client: 9b12312b
> MixBytes: There is also a check at UsrRedemptionExtension.sol#L166, which ensures that `withdrawalTokenDecimals` is not greater than `USR_DECIMALS`. If USR is redeployed with fewer decimals, this check will prevent the use of tokens with more decimals. We recommend removing this check and adapting the code to allow `withdrawalToken` to have more decimals than USR.

L-4	Unsafe Data Type Casting
Severity	Low
Status	Fixed in 77d8ed1d

Description

The issue has been discovered in the `getRedeemPrice` function of the `UsrRedemptionExtension` contract. When the `priceDecimals` equals to `USR_DECIMALS`, UsrRedemptionExtension.sol#L246 in the function for the `price` variable is potentially unsafe.

Recommendation

While the current code configuration does not pose a direct security threat, implementing type-safe operations would enhance the contract's reliability and robustness against potential bugs. We recommend the utilization of `SafeCast` for `price` when the decimals are equal.

Client's Commentary

> 8fcf3509

L-5	Unused Return Variable
Severity	Low
Status	Fixed in 77d8ed1d

Description

The issue is identified within the function `getUSRPrice` of contract `UsrRedemptionExtension`.

The function declares a named return variable `usrPrice` but never explicitly assigns a value to it. Instead, the function implicitly returns the `price` variable. While this doesn't affect functionality, it reduces code clarity.

Recommendation

We recommend removing the named return variable.

Client's Commentary

> 4421e92b

L-6	Missing Validation in Constructor Parameters of `UsrRedemptionExtension`
Severity	Low
Status	Fixed in 77d8ed1d

Description

This issue has been identified in the constructor of the contract `UsrRedemptionExtension`.

1. **Potentially Large `_lastResetTime`:**
   The constructor does not include validation to ensure that `_lastResetTime` is within a reasonable range. If an excessively large value is mistakenly provided, it could result in the `currentRedemptionUsage` being reset far into the future. Furthermore, an incorrectly large `lastResetTime` cannot be corrected after deployment.

2. **Duplicate Entries in `_allowedWithdrawalTokenAddresses`:**
   The constructor does not validate that the addresses provided in `_allowedWithdrawalTokenAddresses` are unique. While this does not affect the correctness of the contract, it could make debugging and identification of invalid parameters more challenging.

The issue is classified as **low** severity because it does not compromise the protocol's security directly but could lead to operational inefficiencies and errors.

Recommendation

1. Add a check in the constructor to ensure that `_lastResetTime` is not excessively large and falls within a sensible range.
2. Add a check in `UsrRedemptionExtension.addAllowedWithdrawalToken()` that the token being added has not already been added.

Client's Commentary

> 77d8ed1d

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