Resolv Security Review

Pashov Audit Group

Conducted by: T1MOH, MrPotatoMagic, ast3ros

December 9th - December 12th

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **resolv-im/resolv-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Resolv

Resolv is a protocol that issues a stablecoin, USR, backed by ETH and keeps its value stable against the US Dollar by hedging ETH price risks with short futures positions. It also maintains an insurance pool, RLP, to ensure USR remains overcollateralized and allows users to mint and redeem these tokens with deposited collateral. This scope adds new swap and exdends redemption mechanism.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - a66183cfa825338cc2c6fef954bfc03a932a1544

fixes review commit hash - d44104b36b6b94db811e2e66442524983e761931

Scope

The following smart contracts were in scope of the audit:

- `TheCounter`
- `RlpPriceStorage`
- `ExternalRequestsManager`
- `UsrRedemptionExtension`

7. Executive Summary

Over the course of the security review, T1MOH, MrPotatoMagic, ast3ros engaged with Resolv to review Resolv. In this period of time a total of **5** issues were uncovered.

Protocol Summary

Protocol Name	Resolv
Repository	https://github.com/resolv-im/resolv-contracts
Date	December 9th - December 12th
Protocol Type	Stablecoin

Findings Count

Severity	**Amount**
High	1
Low	4
Total Findings	**5**

Summary of Findings

ID	Title	Severity	Status
[H-01]	transferFee() uses an incorrect transfer method	High	Resolved
[L-01]	Missing upper limit validation	Low	Resolved
[L-02]	Missing slippage protection in redeem function	Low	Resolved
[L-03]	Aave V3 price source can be inaccurate	Low	Resolved
[L-04]	Redemption limit bypass	Low	Acknowledged

8. Findings

8.1. High Findings

[H-01] `transferFee()` uses an incorrect transfer method

Severity

Impact: Medium

Likelihood: High

Description

The `transferFee` function in the `TheCounter` contract incorrectly uses `safeTransferFrom` instead of `safeTransfer` when transferring collected fees from the contract to the admin. This implementation error will cause the function to revert due to missing allowances.

As a result, the admin is unable to collect protocol fees.

```
function transferFee() external onlyRole(DEFAULT_ADMIN_ROLE) {
        ...
        token.safeTransferFrom(address
        //(this), msg.sender, feeToTransfer); // @audit should use safeTransfer
        ...
    }
```

Recommendations

```
function transferFee() external onlyRole(DEFAULT_ADMIN_ROLE) {
        ...
-        token.safeTransferFrom(address(this), msg.sender, feeToTransfer);
+        token.safeTransfer(msg.sender, feeToTransfer);
        ...
    }
```

8.2. Low Findings

[L-01] Missing upper limit validation

The `setUpperBoundPercentage` and `setLowerBoundPercentage` functions in `RlpPriceStorage` contract lack validation to ensure bound percentages don't exceed the `BOUND_PERCENTAGE_DENOMINATOR` (1e18).

Especially if `lowerBoundPercentage` > `BOUND_PERCENTAGE_DENOMINATOR`, the `setPrice` function will always revert because: `currentPrice` < `(currentPrice * lowerBoundPercentage / BOUND_PERCENTAGE_DENOMINATOR)`.

It's recommended to add validation to ensure bound percentages don't exceed `BOUND_PERCENTAGE_DENOMINATOR`. Otherwise, setPrice() will revert due to underflow in the lower bound percentage calculation. This would prevent the SERVICE_ROLE from updating the prices in a timely manner, leading to possible stale prices.

[L-02] Missing slippage protection in redeem function

The `redeem` function in `UsrExternalRequestsManager` lacks a minimum expected amount parameter to protect users from price changes between transaction submission and execution. The redemption rate is determined by the Aave oracle price at execution time. If network congestion causes transaction delays or if there is high price volatility:

- User submits redemption expecting X tokens based on the current price
- Transaction remains pending while price moves unfavorably
- When executed, user receives significantly less than X tokens
- User has no control as there was no minimum amount specified

```
function redeem(
        uint256 _amount,
        address _receiver,
        address _withdrawalTokenAddress
    ) public whenNotPaused onlyAllowedProviders {
        IERC20(ISSUE_TOKEN_ADDRESS).safeTransferFrom(msg.sender, address
          (this), _amount);
        usrRedemptionExtension.redeem
          (_amount, _receiver, _withdrawalTokenAddress);

        emit Redeemed(msg.sender, _receiver, _amount, _withdrawalTokenAddress);
    }
```

It's recommended to add a `_minExpectedAmount` parameter to the redeem function.

[L-03] Aave V3 price source can be inaccurate

The `UsrRedemptionExtension` contract relies on Aave V3's oracle system to determine withdrawal token amounts during redemption. However, the implementation has potential price accuracy issues due to how it interacts with `Chainlink` oracles.

```
function redeem(
        uint256 _amount,
        address _receiver,
        address _withdrawalTokenAddress
    ) public whenNotPaused allowedWithdrawalToken
      (_withdrawalTokenAddress) onlyRole(SERVICE_ROLE) {
        ...
        IAaveOracle aavePriceOracle = IAaveOracle
          (ADDRESSES_PROVIDER.getPriceOracle());
        uint256 withdrawalTokenAmount =
          (_amount * aavePriceOracle.getAssetPrice(_withdrawalTokenAddress))
            / (aavePriceOracle.BASE_CURRENCY_UNIT() * 10 **
              (USR_DECIMALS - withdrawalTokenDecimals));
        ...
    }
```

The issue is from two main problems in Aave V3's oracle implementation:

```
function getAssetPrice(address asset) public view override returns (uint256) {
    AggregatorInterface source = assetsSources[asset];

    if (asset == BASE_CURRENCY) {
      return BASE_CURRENCY_UNIT;
    } else if (address(source) == address(0)) {
      return _fallbackOracle.getAssetPrice(asset);
    } else {
      int256 price = source.latestAnswer(); // @audit call source.latestAnswer
      if (price > 0) {
        return uint256(price);
      } else {
        return _fallbackOracle.getAssetPrice(asset);
      }
    }
```

Let's inspect Aave V3's price source of USDT: `0xC26D4a1c46d884cfF6dE9800B6aE7A8Cf48B4Ff8`

- Unsafe price fetching: The Aave oracle uses `latestAnswer` instead of `latestRoundData` when querying Chainlink price feeds. No validation of price staleness is performed.
- Artificially capped prices may not reflect true market conditions

`PriceCapAdapterStable` contract:

```
function latestAnswer() external view override returns (int256) {
    int256 basePrice = ASSET_TO_USD_AGGREGATOR.latestAnswer
    //(); // @audit call latestAnswer instead of latestRoundData
    int256 priceCap = _priceCap;

    if (basePrice > priceCap) { // @audit price is capped
      return priceCap;
    }

    return basePrice;
  }
```

As a result, users could receive incorrect amounts during token redemptions.

It's recommended to implement direct Chainlink oracle price fetching and validate price freshness.

[L-04] Redemption limit bypass

The `redeem` function in `UsrRedemptionExtension` implements a daily redemption limit that resets every 24 hours. However, the current implementation is vulnerable to a limit bypass attack due to how the reset window is handled.

```
function redeem(
        uint256 _amount,
        address _receiver,
        address _withdrawalTokenAddress
    ) public whenNotPaused allowedWithdrawalToken
      (_withdrawalTokenAddress) onlyRole(SERVICE_ROLE) {
        ...
        uint256 currentTime = block.timestamp;
        if (currentTime >= lastResetTime + 1 days) {
            // slither-disable-start divide-before-multiply
            uint256 periodsPassed = (currentTime - lastResetTime) / 1 days;
            lastResetTime += periodsPassed * 1 days;
            // slither-disable-end divide-before-multiply

            currentRedemptionUsage = 0;

            emit RedemptionLimitReset(lastResetTime);
        }
        ...
    }
```

Consider the scenario:

- An attacker monitors the `lastResetTime` and waits until just before the 24-hour reset window
- They execute a redemption for the maximum allowed amount (e.g., 100,000 USR)
- After the reset triggers (can be in the next block), they immediately execute another redemption for the maximum amount
- This allows redeeming up to 2x the intended limit (e.g., 200,000 USR) within a very short timeframe

According to the redemption extension documentation, the cap is a critical safety parameter: `Redemption Cap per 24hr = USR 100,000`.

It's recommended to implement redemption limit mechanism with rolling windows of 24 hours.