Resolv Security Review

Pashov Audit Group

Conducted by: Said, btk, ZanyBonzy

October 10th - October 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.

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 - ea4bdfae5bce7ee52d139610214d35da4c77ab4d

fixes review commit hash - c5166ea4c016e1a3f8e3a5185818f4864a8d4317

Scope

The following smart contracts were in scope of the audit:

- `DineroTreasuryConnector`
- `Treasury`

7. Executive Summary

Over the course of the security review, Said, btk, ZanyBonzy 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	October 10th - October 12th
Protocol Type	Stablecoin protocol

Findings Count

Severity	**Amount**
Medium	2
Low	3
Total Findings	**5**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Lack of instantRedeemWithPxEth support	Medium	Resolved
[M-02]	Not supporting IERC1155Receiver interface ID	Medium	Resolved
[L-01]	dineroRedeem uses incorrect operation type	Low	Resolved
[L-02]	apxETH should be deposited directly into the treasury	Low	Resolved
[L-03]	emergencyWithdrawETH is not needed	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Lack of `instantRedeemWithPxEth` support

Severity

Impact: Medium

Likelihood: Medium

Description

Pirex ETH's redemption process using upxETH (via `initiateRedemption` and `redeemWithUpxEth`) requires some time and depends on multiple factors:

- The number of users wanting to withdraw: if `pendingWithdrawal` has not reached `DEPOSIT_SIZE`, need to wait more users need to withdraw until `pendingWithdrawal` reaches `DEPOSIT_SIZE`.
- To withdraw the full balance of a validator, it must first exit entirely. The exiting process takes a variable amount of time, depending on how many others are exiting at the same time. The more exits, the longer a validator has to wait.
- In addition to being dependent on the exit queue of the Beacon chain, it also depends on Pirex ETH's `OracleAdapter`, which utilizes an off-chain process that can introduce additional uncertainty regarding the withdrawal finalization time.

In case of an emergency where Resolv needs to withdraw assets from the Treasury, the assets in `Dinero` cannot fulfill the need because the process is not instantaneous.

Recommendations

Consider implementing `instantRedeemWithPxEth` as an option within `DineroTreasuryConnector`. Although this depends on the available `buffer` and incurs a higher fee, it supports instant withdrawals in case of an emergency.

[M-02] Not supporting `IERC1155Receiver` interface ID

Severity

Impact: Medium

Likelihood: Medium

Description

`DineroTreasuryConnector` inherits from `ERC1155HolderUpgradeable`, which implements ERC-165 to support the `IERC1155Receiver` interface ID. also inherits from `AccessControlDefaultAdminRulesUpgradeable`, which implements ERC-165 to support the `IAccessControlDefaultAdminRules` interface ID.

But the current implementation of `supportsInterface` interface within `DineroTreasuryConnector` remove `IERC1155Receiver` interface ID from ERC-165 checker.

```
contract
   DineroTreasuryConnector is IDineroTreasuryConnector, ERC1155HolderUpgradeable, Acce

    // ...

    function initialize() public initializer {
        __AccessControlDefaultAdminRules_init(1 days, msg.sender);
        __ERC1155Holder_init();
    }

    function supportsInterface(
        bytes4 _interfaceId
    ) public view virtual override(
      ERC1155HolderUpgradeable,
      AccessControlDefaultAdminRulesUpgradeable
    ) public view virtual override
      (ERC1155HolderUpgradeable, AccessControlDefaultAdminRulesUpgradeable
        return super.supportsInterface(_interfaceId);
    }
// ...
}
```

The reason is due to multiple inheritance, as `supportsInterface` is present in both `ERC1155HolderUpgradeable` and `AccessControlDefaultAdminRulesUpgradeable`, which are inherited by `DineroTreasuryConnector`. The `super` call will trigger the right-most contract, which is `AccessControlDefaultAdminRulesUpgradeable`, because Solidity inheritance is linearized from right to left.

As a result, if any contracts check whether `DineroTreasuryConnector` supports `IERC1155Receiver` using ERC-165, it will return false.

PoC :

```
describe("Check ERC165 interface", async () => {
    it(
      "checkinterfaceidofERC1155ReceiverandAccessControlDefaultAdminRules",
      async
    ) => {
      // given
      const {owner, treasury} = await loadFixture(deployOrGetFixture);

      // when
      // ERC1155 interfaceId : 0x4e2312e0
      const result = await treasury.supportsInterface("0x4e2312e0");

      // AdminRules interfaceId : 0x31498786
      const result2 = await treasury.supportsInterface("0x31498786");

      // then
      expect(result).is.false;
      expect(result2).is.true;
    })
  })
```

Recommendations

Modify `DineroTreasuryConnector`'s `supportsInterface` to the following :

```
function supportsInterface(
        bytes4 _interfaceId
    ) public view virtual override(
      ERC1155HolderUpgradeable,
      AccessControlDefaultAdminRulesUpgradeable
    ) public view virtual override
      (ERC1155HolderUpgradeable, AccessControlDefaultAdminRulesUpgradeable
-        return super.supportsInterface(_interfaceId);
+        return interfaceId == type
+ (IERC1155Receiver).interfaceId || super.supportsInterface(interfaceId);
    }
```

8.2. Low Findings

[L-01] `dineroRedeem` uses incorrect operation type

`dineroRedeem` uses `DineroInitiateRedemption` instead of `DineroRedeem` operation type. This might lead to unexpected behavior in the `idempotent` modifier when redeeming ETH as the wrong key will be mapped to the wrong operation type.

Recommend using the appropriate operation type.

```
//...
-    ) external onlyRole(SERVICE_ROLE) idempotent
- (OperationType.DineroInitiateRedemption, _idempotencyKey) whenNotPaused {
+    ) external onlyRole(SERVICE_ROLE) idempotent
+ (OperationType.DineroRedeem, _idempotencyKey) whenNotPaused {
//...
    }
```

[L-02] `apxETH` should be deposited directly into the treasury

`deposit` exchanges `pxETH` for `apxETH` which is first deposited into the connector before being transferred to the treasury. Since `apxETH.deposit` allows specifying receiver, the receiver can be directed to the treasury (i.e. `msg.sender`)instead of `address(this)`. This eliminates the need for the extra token transfer to the treasury.

```
PX_ETH.safeIncreaseAllowance(address(apxETH), pxETHPostFeeAmount);
-       apxETHAmount = apxETH.deposit(pxETHPostFeeAmount, address(this));
-       IERC20(apxETH).safeTransfer(msg.sender, apxETHAmount);
+       apxETHAmount = apxETH.deposit(pxETHPostFeeAmount, msg.sender);
    }
```

[L-03] `emergencyWithdrawETH` is not needed

DineroTreasuryConnector.sol only has one way to receive ETH which is through the payable `deposit` function. In the function, all the ETH sent from the treasury is converted to `PX_ETH` with no refunds processed. So there's no way for the contract to hold ETH. Therefore, having a function to retrieve ETH in the contract is redundant.

Recommend removing the function, or introducing a `receive` function if retrieving ETH is desired.