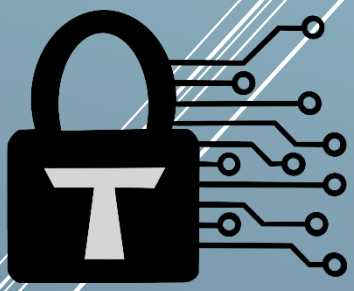
Trust Security



Smart Contract Audit

Reserve Protocol – Convex ETH+/ETH PR

03/07/24

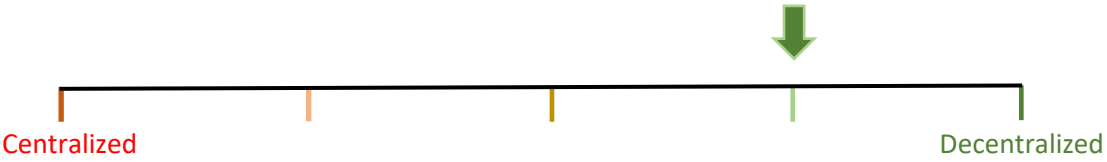
Executive summary

Category	Stablecoin
Auditor	HollaDieWaldfeegjaldon

Findings

Severity	Total	Fixed	Open	Acknowledged
High	1	1	-	-
Medium	1	1	-	-
Low	2	1	-	1

Centralization score



Signature

Digitally signed by Trust
Date: 2024.07.03 19:56:23 +03'00'

Document properties

Versioning

Version	Date	Description
0.1.0	01/06/2024	Client report
0.1.1	03/07/2024	Scope changes

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security serves as a long-term security partner of the Reserve Protocol. It has conducted the audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Additional recommendations have been given when appropriate.

Scope

The following files are in scope of the audit:

- contracts/plugins/assets/AppreciatingFiatCollateral.sol
- contracts/plugins/assets/curve/CurveAppreciatingRTokenFiatCollateral.sol
- contracts/plugins/assets/curve/CurveAppreciatingRTokenSelfReferentialCollateral.sol
- contracts/plugins/assets/curve/CurveStableRTokenMetapoolCollateral.sol

Scoping requests:

- Only non-reweightable RTokens are considered in-scope.

Repository details

- **Repository URL:** https://github.com/reserve-protocol/protocol
- **Commit hash:** 102a7611230106dcd0674e36051aac0a1b53541d
- **Mitigation hash:** 9b4160d4dfb1a212da879d89cbb21a6a90420396

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

About the Auditors

HollaDieWaldfee is a renowned security expert with a track record of multiple first places in competitive audits. He is a Lead Senior Watson at Sherlock and Lead Auditor for Trust Security and Renascence Labs.

Gjaldon is a DeFi specialist who enjoys numerical and economic incentives analysis. He transitioned to Web3 after 10+ years working as a Web2 engineer. His first foray into Web3 was achieving first place in a smart contracts hackathon and then later securing a project grant to write a contract for Compound III. He shifted to Web3 security and in 3 months achieved top 2-5 in two contests with unique High and Medium findings and joined exclusive top-tier auditing firms.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Findings

High severity findings

TRST-H-1 ETH+/ETH Curve pool is vulnerable to read-only reentrancy

- **Category:** Reentrancy issues
- **Source:** CurveAppreciatingRTokenSelfReferentialCollateral.sol
- **Status:** Fixed

Description

The ETH+/ETH Curve pool uses WETH as one of its tokens. However, internally, it converts every deposited WETH into ETH. ETH triggering callbacks opens up the pool to read-reentrancy issues in *CurveAppreciatingRTokenSelfReferentialCollateral* since it uses its *get_virtual_price()* as its *refPerTok()* exchange rate.

```
def get_virtual_price() -> uint256:
    return 10**18 * self.get_xcp(self.D) / CurveToken(self.token).totalSupply()
```

An attacker can call *remove_liquidity()* in the ETH+/ETH pool to manipulate the *get_virtual_price()* output. This is possible since *totalSupply()* of the LP token has already been decreased and the ETH receiver callback is triggered before **self.D** is updated. The effect is an increase in *get_virtual_price()*.

The increase of *get_virtual_price()* can be exploited with the following steps:

1. Increase *get_virtual_price()* by removing liquidity as ETH.
2. In the receiver callback, *refresh()* the Curve collateral and mint RTokens at a discount with *RToken.issue()* by providing inflated Curve LP tokens.
3. *refresh()* the Curve collateral again to make its price drop back to the original and let the collateral default.
4. Redeem RTokens with *RToken.redeemCustom()* and receive more Curve LP tokens than have been provided in (2). Since *RToken.redeemCustom()* allows the attacker to redeem their RTokens for a pro-rata share of collateral, the loss due to the inflated Curve LP tokens is split among all RToken holders, including the attacker. The attacker can wait for recollateralization to occur to maximize the redemption value of his RToken. However, this risks a potential response by the RToken Governance which in principle has the power to punish the attacker.

The attack acts as a net transfer from other RToken holders or RSR stakers to the attacker.

Recommended mitigation

To support pools with callbacks, call a cheap non-reentrant function like *claim_admin_fees()* from within *underlyingRefPerTok()* or when doing any operation that uses *refPerTok()* as an exchange rate. With this mitigation, any attempt to reenter will revert the transaction.

Team response

Fixed by calling *claim_admin_fees()* during *refresh()*.

Mitigation review

All functions in the ETH+/ETH pool that perform raw calls have a reentrancy guard. Therefore, all manipulation attempts originating from raw_call() can be caught by calling *claim_admin_fees()*. Meanwhile, *underlyingRefPerTok()*, which is the only place where the manipulated Curve *get_virtual_price()* value has an effect on, is only called within *refresh()*, so it triggers the reentrancy guard.

Recently, the visibility of *underlyingRefPerTok()* has been changed to **public** so that it can be used with **try-catch**. This means that any use of *underlyingRefPerTok()* in other functionality that doesn't rely on the successful execution of *refresh()* can introduce the vulnerability again.

In addition, *claim_admin_fees()* is not available for all Curve pools, and the mitigation must be expected to be specific for the ETH+/ETH pool. If other pools with callbacks should be integrated with, their pool code must be assessed separately.

Medium severity findings

TRST-M-1 Missing !fullyCollateralized() soft-default check allows issuing RTokens at a discount

- **Category:** Validation issues
- **Source:** CurveAppreciatingRTokenFiatCollateral.sol, CurveStableRTokenMetapoolCollateral.sol
- **Status:** Fixed

Description

In *CurveAppreciatingRTokenFiatCollateral* and *CurveAppreciatingRTokenMetapoolCollateral,* a check has been introduced that the inner RToken must be ready (**isReady() = true**). This is to prevent a scenario where the inner RToken has lost value and outer RTokens can be issued at a discount. Whenever there is uncertainty about the valuation of the inner RToken, the Curve collateral must be marked **IFFY**, such that the outer RToken can no longer be issued.

The problem is that *isReady()* is an insufficient check. After the inner RToken has switched to a new reference basket, *isReady()* can return **true** while the inner RToken is still undercollateralized, i.e., inner RToken is less valuable than assumed by the collateral plugin. As a result, during recollateralization of the inner RToken, outer RToken can be issued at a discount.

Consequently, bad debt can be injected into the outer RToken, causing a loss to outer RToken holders and outer RToken stakers.

Recommended mitigation

In addition to the *isReady()* check, it must also be checked that the inner RToken is *fullyCollateralized()*.

Team response

Fixed.

Mitigation review

The recommendation has been implemented.

Low severity findings

TRST-L-1 Curve collateral with an inner RToken introduces timing constraints

- **Category:** Time-sensitivity issues
- **Source:** CurveAppreciatingRTokenFiatCollateral.sol, CurveStableRTokenMetapoolCollateral.sol
- **Status:** Acknowledged

Description

When an RToken in a Curve Pool is not ready, the Curve collateral will be marked as IFFY. The inner RToken can only be ready again when its status has become **SOUND** and it has stayed that way for the length of the **warmupPeriod**.

```
function isReady() external view returns (bool) {
      return
          status() == CollateralStatus.SOUND &&
          (block.timestamp >= lastStatusTimestamp + warmupPeriod);
  }
```

With the mitigation for TRST-M-1 applied, the Curve collateral also remains **IFFY** while the inner RToken is undercollateralized which makes the timing constraint even more important.

The issue occurs when the Curve collateral's **delayUntilDefault** period (72 hours) is shorter than the time it takes for the inner RToken to be ready and fully collateralized again. If the inner RToken can remain **!isReady() || !fullyCollateralized()** for longer than **delayUntilDefault**, the Curve collateral can default after being marked **IFFY** even if the inner RToken can still recover.

For an outer RToken to safely use the inner RToken in a Curve collateral, **delayUntilDefaultOuter** for the outer RToken must be longer than **max(delaysUntilDefaultInner) + warmupPeriodInner + maximum expected time for recollateralization of inner RToken**.

An example of an inner RToken configuration that is not possible is when it uses an RToken Curve collateral with 72 hours **delayUntilDefault** itself.

Recommended mitigation

The most practical way to address the timing constraints is through documentation to prevent configurations that would cause the issue.

Team response

Acknowledged.

Mitigation review

No mitigations have been implemented but the client is aware of the issue and its impact.

TRST-L-2 underlyingRefPerTok() calculation in CurveAppreciatingRTokenFiatCollateral assumes the RToken basket unit is worth one reference unit in the collateral

- **Category:** Logical issues
- **Source:** CurveAppreciatingRTokenFiatCollateral.sol,
- **Status:** Fixed

Description

The *underlyingRefPerTok()* calculation expects the RToken's basket unit to be worth one reference unit in the collateral.

```
    function underlyingRefPerTok() public view virtual override returns (uint192) {
        uint192 virtualPrice = _safeWrap(curvePool.get_virtual_price());
        uint192 rTokenRate = divuu(rToken.basketsNeeded(), rToken.totalSupply());
        return virtualPrice.mul(rTokenRate.sqrt()).mulu(2);
    }
```

However, the RToken can be configured with arbitrary target amounts which the basket unit follows.

Recommended mitigation

Consider accounting for the different ratios for basket unit to reference unit when calculating *underlyingRefPerTok()*. This is optional, and if the outer RToken is supposed to check the inner RToken for compatibility before using it as collateral, the requirement should be documented.

Team response

The assumption that the RToken's basket unit must be worth one reference unit has been documented. Changing the calculation is not worth the overhead.

Mitigation review

The assumption has been documented in the *underlyingRefPerTok()* function.

Additional recommendations

De-peg check for inner RToken is made redundant by underlyingRefPerTok()

CurveAppreciatingRTokenSelfReferentialCollateral and *CurveAppreciatingRTokenFiatCollateral* both aim to implement de-peg checks for their inner RToken against the target unit (usually ETH or USD).

It has been determined that the *underlyingRefPerTok()* function is sufficient for both collateral plugins to protect against de-pegs and the de-peg check can be removed from *_anyDepeggedInPool()*.

A de-peg of the inner RToken to the downside, which is the only direction possible based on its smart contract logic, causes a hard-default in *underlyingRefPerTok()*. This is because *underlyingRefPerTok()* is calculated based on **basketsNeeded / totalSupply** of the inner RToken. One basket unit programmatically tracks the value of the inner RToken's target unit (which must be the same as the collateral plugin's target unit). Therefore, **basketsNeeded / totalSupply** tracks the amount of target units per RToken. And *underlyingRefPerTok()* is more restrictive than a de-peg check implemented in *_anyDepeggedInPool()*, which only causes a soft-default.

The assessment has some known nuances:

1) It is assumed that **revenueHiding** is sufficiently smaller than the deviation threshold for the de-peg check in *_anyDepeggedInPool()* such that *underlyingRefPerTok()* detects a de-peg of at least the size of the deviation threshold or smaller.
2) **basketsNeeded / totalSupply** is a lagging measure of the actual target units per RToken and, if the rate drops, **basketsNeeded** is only updated after rebalancing has occurred. A de-peg in the inner RToken can be assumed to only occur when **!isReady()** or **!fullyCollateralized()** which already cause a soft-default. This is only an assumption since a misconfiguration in the inner RToken can certainly make it de-peg without entering the **!isReady() || !fullyCollateralized()** state.

No de-peg checks needed for self-referential tokens

CurveAppreciatingRTokenSelfReferentialCollateral checks WETH/ETH for de-pegs. Self-referential token pairs like WETH/ETH do not need de-peg checks since the exchange rate is guaranteed to be equal to one.

Conditional branches in CurveStableRTokenMetapoolCollateral can be simplified

There are two conditional branches in *CurveStableRTokenMetapoolCollateral* that mark the collateral as **IFFY**. The two branches can be merged into one for clarity.

```
diff --git a/contracts/plugins/assets/curve/CurveStableRTokenMetapoolCollateral.sol
b/contracts/plugins/assets/curve/CurveStableRTokenMetapoolCollateral.sol
index 36bde1b1..838283cb 100644
--- a/contracts/plugins/assets/curve/CurveStableRTokenMetapoolCollateral.sol
+++ b/contracts/plugins/assets/curve/CurveStableRTokenMetapoolCollateral.sol
@@ -91,11 +91,7 @@ contract CurveStableRTokenMetapoolCollateral is
CurveStableMetapoolCollateral {

                // Check RToken status
                try pairedBasketHandler.isReady() returns (bool isReady) {
-                   if (!isReady) {
-                       markStatus(CollateralStatus.IFFY);
-                   } else if (low == 0 || _anyDepeggedInPool() ||
_anyDepeggedOutsidePool()) {
-                       // If the price is below the default-threshold price, default
eventually
-                       // uint192(+/-) is the same as Fix.plus/minus
+                   if (!isReady || low == 0 || _anyDepeggedInPool() ||
_anyDepeggedOutsidePool()) {
                        markStatus(CollateralStatus.IFFY);
                    } else {
                        markStatus(CollateralStatus.SOUND);
```

Centralization risks

The audited PR does not introduce new centralization risks beyond the risks that have been documented in previous audits.

Systemic risks

The audited PR does not introduce new systemic risks beyond the risks that have been documented in previous audits.