



Audit Report for Reserve Protocol - April 25, 2024

Summary

Audit Report prepared by Solidified covering the Reserve Protocol contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on April 3, 2024, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repository:

Repo: <https://github.com/reserve-protocol/protocol/>

Commit hash: **d8669785599e58ca0370cf3a437bf104c742b8b3**

Fixes received at commit **db72e4044d0a3673565ca9e28680fe163df19043**

```
contracts
├── facade
│   ├── DeployerRegistry.sol
│   ├── FacadeAct.sol
│   ├── FacadeMonitor.sol
│   ├── FacadeRead.sol
│   ├── FacadeTest.sol
│   ├── FacadeWrite.sol
│   └── lib
│       └── FacadeWriteLib.sol
├── interfaces
│   ├── IAsset.sol
│   ├── IAssetRegistry.sol
│   ├── IBackingManager.sol
│   ├── IBasketHandler.sol
│   ├── IBroker.sol
│   ├── IComponent.sol
│   ├── IDeployer.sol
│   ├── IDeployerRegistry.sol
│   ├── IDistributor.sol
│   └── IFacadeAct.sol
```



Audit Report for Reserve Protocol - April 25, 2024

- |— IFacadeMonitor.sol
- |— IFacadeRead.sol
- |— IFacadeTest.sol
- |— IFacadeWrite.sol
- |— IFurnace.sol
- |— IGnosis.sol
- |— IMain.sol
- |— IRToken.sol
- |— IRTokenOracle.sol
- |— IRevenueTrader.sol
- |— IRewardable.sol
- |— IStRSR.sol
- |— IStRSRVotes.sol
- |— ITrade.sol
- |— ITrading.sol
- |— IVersioned.sol
- |— libraries
 - |— Allowance.sol
 - |— Array.sol
 - |— Fixed.sol
 - |— NetworkConfigLib.sol
 - |— Permit.sol
 - |— String.sol
 - |— Throttle.sol
- |— mixins
 - |— Auth.sol
 - |— ComponentRegistry.sol
 - |— Versioned.sol
- |— p0
 - |— AssetRegistry.sol
 - |— BackingManager.sol
 - |— BasketHandler.sol
 - |— Broker.sol
 - |— Deployer.sol
 - |— Distributor.sol
 - |— Furnace.sol
 - |— Main.sol
 - |— RToken.sol
 - |— RevenueTrader.sol
 - |— StRSR.sol
 - |— mixins



Audit Report for Reserve Protocol - April 25, 2024

```
|
|
|   |— Component.sol
|   |— Rewardable.sol
|   |— Trading.sol
|   |— TradingLib.sol
|
|— p1
|   |— AssetRegistry.sol
|   |— BackingManager.sol
|   |— BasketHandler.sol
|   |— Broker.sol
|   |— Deployer.sol
|   |— Distributor.sol
|   |— Furnace.sol
|   |— Main.sol
|   |— RToken.sol
|   |— RevenueTrader.sol
|   |— StRSR.sol
|   |— StRSRVotes.sol
|   |— mixins
|       |— BasketLib.sol
|       |— Component.sol
|       |— RecollateralizationLib.sol
|       |— RewardableLib.sol
|       |— TradeLib.sol
|       |— Trading.sol
|
|— plugins
|   |— assets
|       |— AppreciatingFiatCollateral.sol
|       |— Asset.sol
|       |— EURFiatCollateral.sol
|       |— FiatCollateral.sol
|       |— L2LSDCollateral.sol
|       |— NonFiatCollateral.sol
|       |— OracleLib.sol
|       |— RTokenAsset.sol
|       |— SelfReferentialCollateral.sol
|       |— VersionedAsset.sol
|   |— aave
|       |— ATokenFiatCollateral.sol
|       |— IStaticATokenLM.sol
|       |— StaticATokenErrors.sol
|       |— StaticATokenLM.sol
```



Audit Report for Reserve Protocol - April 25, 2024

```
└─ vendor
    ├── ERC20.sol
    ├── IAToken.sol
    ├── IAaveIncentivesController.sol
    ├── RayMathNoRounding.sol
    └─ ReentrancyGuard.sol
└─ aave-v3
    ├── AaveV3FiatCollateral.sol
    └─ vendor
        ├── ERC20.sol
        ├── RayMathExplicitRounding.sol
        ├── StaticATokenErrors.sol
        ├── StaticATokenV3LM.sol
        └─ interfaces
            ├── IAToken.sol
            ├── IERC4626.sol
            ├── IInitializableStaticATokenLM.sol
            └─ IStaticATokenV3LM.sol
└─ ankr
    ├── AnkrStakedEthCollateral.sol
    ├── README.md
    └─ vendor
        └─ IAnkrETH.sol
└─ cbeth
    ├── CBETHCollateral.sol
    ├── CBETHCollateralL2.sol
    ├── README.md
    └─ vendor
        └─ ICBeth.sol
└─ compoundv2
    ├── CTokenFiatCollateral.sol
    ├── CTokenNonFiatCollateral.sol
    ├── CTokenSelfReferentialCollateral.sol
    ├── CTokenWrapper.sol
    └─ ICToken.sol
└─ compoundv3
    ├── CTokenV3Collateral.sol
    ├── CometHelpers.sol
    ├── CusdcV3Wrapper.sol
    ├── ICusdcV3Wrapper.sol
    └─ IWrappedERC20.sol
```



Audit Report for Reserve Protocol - April 25, 2024



Audit Report for Reserve Protocol - April 25, 2024



Audit Report for Reserve Protocol - April 25, 2024

```
└─ vendor
    └─ ERC20PermitUpgradeable.sol
        └─ oz
            └─ ERC4626.sol
                └─ IERC4626.sol
```

Intended Behavior

The audited codebase implements a generic framework to issue tokens that are backed by a rebalancing basket of collateral.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	Extensive test suite with almost 100% coverage. Whenever something is not covered, it is argued why.

Issues Found

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Unregistering latest deployment is possible	Minor	Fixed
2	Emitting the same events twice when claiming rewards in AaveV3FiatCollateral	Minor	Acknowledged
3	The maxPoolOracleTimeout function only takes the second feed's timeout into account	Minor	Fixed
4	Suboptimal incentives to stably overcollateralize RTokens	Minor	Acknowledged
5	Backup assets might not activate if too many backup assets are configured.	Minor	Fixed
6	Validate that the configured oracle address is a price feed contract	Minor	Acknowledged
7	Component addresses not validated to be unique	Minor	Acknowledged
8	Lack of a secondary fallback price feed oracle	Minor	Acknowledged
9	More efficient ERC20 implementation could be used	Note	Acknowledged
10	Behavior of WrappedERC20 allowances	Note	Acknowledged
11	Minimal buying amount is not guaranteed to be below the dust limit	Note	Acknowledged
12	Usage of _msgSender() in Component could lead to problems in the future	Note	Acknowledged
13	Inactive Price Feeds in common/configuration.ts	Note	Fixed
14	The OracleLib does not handle negative prices	Note	Fixed

15	Incorrect parameter name in the init function of p0/BackingManager.sol	Note	Fixed
16	Function variable bal shadows the function from the inherited Asset contract	Note	Fixed
17	CurveStableCollateral emits unnecessary RewardsClaimed events	Note	Acknowledged
18	The SFraxCollateral constructor lacks the defaultThreshold > 0 check	Note	Fixed
19	Redundant checks for defaultThreshold > 0 in CBETHCollateralL2 and CTokenNonFiatCollateral constructors	Note	Fixed
20	Unnecessary approve calls when the allowance is already set to zero	Note	Fixed
21	The pricePerShareHelper is only available on the Ethereum mainnet	Note	Acknowledged
22	Version not updated	Note	Fixed
23	Inherited maxOracleTimeout value has to be ignored in L2LidoStakedEthCollateral	Note	Acknowledged
24	The file docs/security.md does not exist	Note	Fixed
25	The prepareRecollateralizationTrade function in TradingLibP0 has a different implementation from p1	Note	Acknowledged
26	Unnecessary function call in the refresh function	Note	Fixed
27	Incorrect price feed comments	Note	Fixed
28	Unused immutable variables rsr and stRSR in RTokenAsset	Note	Fixed
29	Lack of zero checks in the MorphoNonFiatCollateral constructor	Note	Fixed
30	Users might trigger the RToken to trade by increasing its balance to capture MEV	Note	Acknowledged
31	MAX_TRADE_VOLUME is equal to 1e29 and not	Note	Acknowledged



Audit Report for Reserve Protocol - April 25, 2024

	1e24 as documented.		
32	Ambiguous error message	Note	Acknowledged
33	i++ is less efficient than ++i	Note	Acknowledged
33	Miscellaneous	Note	Fixed
34	Known issues and limitations	Note	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. Unregistering latest deployment is possible

The `DeployerRegistry` has a public variable `latestDeployment` that tracks the latest deployment and could for instance be consumed by other smart contracts. However, it is possible to `unregister` this deployment such that it is no longer a deployment of any version. In such a situation, the variable `latestDeployment` is not reset, meaning it points to an unregistered deployment, which may be undesirable and could lead to inconsistencies.

Recommendation

We recommend either resetting `latestDeployment` if it is unregistered or to disallow the unregistration of the latest deployment.

2. Emitting the same events twice when claiming rewards in `AaveV3FiatCollateral`

The `claimRewards` function in the `StaticATokenV3LM` wrapper contract performs balance checks, claims the rewards, and emits necessary events. However, the current logic for claiming rewards in the `AaveV3FiatCollateral` contract performs similar checks while also calling the

`StaticATokenV3LM.claimRewards` function. This behavior is redundant and will emit the same events twice.

Recommendation

Remove the redundant loops in `AaveV3FiatCollateral.claimRewards` and simply call `erc20_.claimRewards()`.

3. The `maxPoolOracleTimeout` function only takes the second feed's timeout into account

The `PoolTokens` contract maintains up to two oracle timeouts for each token. However, the `maxPoolOracleTimeout` function only takes into account the second timeout of each token. If the longest oracle timeout is not stored, the collateral might be mistakenly marked as `IFFY`.

Recommendation

Consider checking all oracle timeouts when calculating the `maxPoolOracleTimeout`.

4. Suboptimal incentives to stably overcollateralize RTokens

RTokens are overcollateralized by users with staked RSR and a constant split from the RToken revenue is sent to RSR stakers. As the split is constant, RSR stakers have no incentive to provide a certain or reliable target over-collateralization to the token. The first staker gets all the rewards no matter how much insurance they provide or if the total staked RSR value increases relative to the RToken, there is no extra reward for the increase in the value.

Recommendation

We recommend implementing a logic that incentivizes stable over-collateralization.

5. Backup assets might not activate if too many backup assets are configured.

In `p1 BasketLib.sol` the `nextBasket` function might run out of gas if too many `backup.erc20s` are configured because the `max` value of the `BasketConfig` can take on arbitrary large values or configured backup assets might not be `goodCollateral`.

This is a minor issue as it can be caused only by the governance of the respective RToken. However, as the governance can be normal users who create RTokens, the consequences can be significant as the fallback mechanism might not work as intended.

Recommendation

We recommend limiting the amount of `backup.erc20s` that can be configured.

6. Validate that the configured oracle address is a price feed contract

In the `constructor` of `Asset.sol` it is not validated that the configured `chainlinkFeed` address is an oracle interface.

Recommendation

In `Asset.sol` validate the address by calling `tryPrice` on the interface and if it reverts the `chainlinkFeed` address is not valid. This reduces the risk of deploying with a non-valid price feed oracle.

7. Component addresses not validated to be unique

In `p1:Deployer.sol`, the addresses stored in `implementations` might not be unique resulting in a compromised deployment.

Recommendation

We recommend validating the addresses so that they do not contain duplicates.

8. Lack of a secondary fallback price feed oracle

If the ChainLink oracle reports zero or stale price, the price might become IFFY; Therefore, the protocol will not trade the asset during this time but hold it while the price drops quickly in the worst case.

Recommendation

We recommend having a secondary oracle available in case the ChainLink oracle fails.

Informational Notes

9. More efficient ERC20 implementation could be used

The file `contracts/vendor/ERC20PermitUpgradeable.sol` is a slightly modified version of OpenZeppelin's `ERC20PermitUpgradeable`. However, it is based on an older version that still uses `CountersUpgradeable` internally. Newer versions of the contract have changed that and do no longer use a library for the nonces.

Recommendation

Consider updating the dependency to the newest version.

10. Behavior of `WrappedERC20` allowances

In the comment of the file `wrappedERC20.sol`, it is mentioned that it is a “soft-fork” of OpenZeppelin’s ERC20 contract with a few listed differences. Besides these differences, there are two additional small differences because of the changed allowance system:

- While it is possible to set or unset an allowance where the spender address is the same address as the owner address, this does not have any impact on the behavior of the token, as the owner is always allowed to call `transferFrom`. This is different from OpenZeppelin, where a `transferFrom` with `owner == spender` is only allowed if the user has set the allowance previously.
- Similarly, the `allowance` function always returns `type(uint256).max` for `owner == spender`, even if there was no previous `approve` call with these addresses.

The behavior in these cases is not explicitly specified in ERC20, so this is not a violation of the standard. However, if consumers assume that the token behaves the same as OpenZeppelin’s implementation, it could lead to problems. For example, the `AllowanceLib` of Reserve would not be able to set the allowance of such a token with `spender == address(this)` because this code would fail:

```
// 1. Set initial allowance to 0
token.approve(spender, 0);
require(token.allowance(address(this), spender) == 0, "allowance not 0");
```

Recommendation

Consider changing the implementation if it is desired that the behavior is identical to OpenZeppelin’s implementation with respect to allowances.

11. Minimal buying amount is not guaranteed to be below the dust limit

In `p1 TradeLib.sol` the `minBuyAmount` amount might fall below the dust limit of tradable tokens.

Recommendation

We recommend validating that the bought amount is above the dust limit.

12. Usage of `_msgSender()` in `Component` could lead to problems in the future

The `governance()` modifier within the `Component` contract uses `_msgSender()` instead of `msg.sender`. This is typically done if support for meta transactions (ERC2771) is desired at some point in the future, because it makes the migration easier. However, supporting meta transactions would introduce a big vulnerability in the code base: `TradingP1` inherits from `ComponentP1`, but also from `Multicall`. `Multicall` in combination with ERC2771 meta transactions can be abused to spoof the `_msgSender()` return value, which was exploited recently.

Recommendation

Consider replacing `_msgSender()` with `msg.sender` if no support for meta transactions is planned. If this is planned, it should only be done after an update to a more recent OpenZeppelin version where this vulnerability was addressed.

13. Inactive Price Feeds in `common/configuration.ts`

Some of the price feeds listed in the `common/configuration.ts` are inactive and Chainlink does not support them:

- `sUSD: '0xad35Bd71b9aFE6e4bDc266B345c198eaDEf9Ad94'` - sUSD/USD price feed.
- `EURT: '0x01D391A48f4F7339aC64CA2c83a07C22F95F587a'` - EURT/USD price feed.

Recommendation

Consider removing or replacing the sUSD/USD and EURT/USD price feeds.

14. The `OracleLib` does not handle negative prices

The current `OracleLib.price` function logic only handles the case if `p == 0`. However, the Chainlink Data Feeds use `int` instead of `uint` for the `p` variable and the returned value could therefore potentially be negative.

Recommendation

Consider replacing the current `if (p == 0) revert ZeroPrice();` check with `if (p <= 0) revert ZeroPrice();` to revert the transaction in the very unlikely event the price drops below zero as well.

15. Incorrect parameter name in the `init` function of `p0/BackingManager.sol`

The `init` function in the `BackingManagerP0` contract uses the `maxTradeVolume_` parameter to set the `minTradeVolume` value.

Recommendation

Rename `maxTradeVolume_` to `minTradeVolume_` in `contracts/p0/BackingManager.sol:46`.

16. Function variable `bal` shadows the function from the inherited `Asset` contract

In the `claimRewards` function of multiple collateral contracts, the `uint256 bal` variable triggers a *shadowed* warning from the inherited `Asset` contract. While this does not pose a security risk, addressing it could minimize potential confusion and enhance readability.

Recommendation

Consider renaming the function variable `bal` to `_bal` in all collateral contracts that inherit from the `Asset`.

17. `CurveStableCollateral` emits unnecessary `RewardsClaimed` events

The `claimRewards` function in the `CurveStableCollateral` plugin contract triggers both `CRV` and `CVX` events, regardless of whether the rewards were earned. This approach is not gas-efficient and could potentially confuse users.

Recommendation

Consider emitting events only for the tokens that have been earned, by conducting additional balance checks before the events are emitted.

18. The `SFraxCollateral` constructor lacks the `defaultThreshold > 0` check

In the current design, setting `defaultThreshold` to 0 implies no deviation tolerance in either direction. As a result, a depeg is registered even if the market price shifts a mere 1 wei from `FIX_ONE`. Hence, for all non-self-referential collaterals (CToken, LSDs, etc.), a `config.defaultThreshold > 0` check should be included. However, this check is missing in `SFraxCollateral`.

Recommendation

Add `require(config.defaultThreshold > 0, "defaultThreshold zero");` to the `SFraxCollateral` constructor.

19. Redundant checks for `defaultThreshold > 0` in `CBETHCollateralL2` and `CTokenNonFiatCollateral` constructors

The `defaultThreshold > 0` check in the `CBETHCollateralL2` constructor is redundant because the same check already exists in the parent contract `L2LSDCollateral`. This also applies to the `CTokenNonFiatCollateral` constructor since the check is already in the `CTokenFiatCollateral` contract.

Recommendation

Remove `require(config.defaultThreshold > 0, "defaultThreshold zero");` from `CBETHCollateralL2` and `CTokenNonFiatCollateral` constructors.

20. Unnecessary `approve` calls when the allowance is already set to zero

The `safeApproveFallbackToMax` function in `AllowanceLib` sets the allowance to `0` without checking the current `allowance` value. If the `allowance` is already set to `0`, this will result in an unnecessary waste of gas.

Recommendation

Consider checking the current allowance before setting it to zero. For example, add `if(token.allowance(address(this), spender) != 0) {...}`.

21. The `pricePerShareHelper` is only available on the Ethereum mainnet

The `pricePerShareHelper`, which was implemented in a recent `YearnV2CurveFiatCollateral` update, is only available on the Ethereum mainnet:

<https://github.com/reserve-protocol/protocol/pull/1014#issuecomment-1836591995>

Recommendation

Ensure you have an alternative helper contract for other chains.

22. Version not updated

The audited protocol version is 3.3.0. We recommend updating it in the following instances:

- The `VERSION` constant in `contracts/mixins/Versioned.sol:7`.
- The `ASSET_VERSION` constant in `contracts/plugins/assets/VersionedAsset.sol:7`.

23. Inherited `maxOracleTimeout` value has to be ignored in `L2LidoStakedEthCollateral`

The `maxOracleTimeout` stores the maximum of all oracle timeouts in the parent `Asset` contract, based on the submitted `config.oracleTimeout`, and is inherited by collateral. However, as the `config.oracleTimeout` value is ignored in the `L2LidoStakedEthCollateral`, it is redundant to include it in `Math.max` calculations during recalculation in the `L2LidoStakedEthCollateral` constructor.

Recommendation

Consider removing `maxOracleTimeout` from the `Math.max` arguments in the `L2LidoStakedEthCollateral` constructor.

24. The file `docs/security.md` does not exist

Body: In `contracts/p0/mixins/Component.sol:26` and `contracts/p1/mixins/Component.sol:39`, the comment `// === See docs/security.md` refers to a file that does not exist.

Recommendation

Remove or update the comments that mention `docs/security.md`.

25. The `prepareRecollateralizationTrade` function in `TradingLibP0` has a different implementation from `p1`

The arguments for the `prepareRecollateralizationTrade` function of `TradingLibP0` differ from its P1 version. This discrepancy is due to the function not being properly updated after the `tradingContext` calculations were moved to a separate function in the `BackingManager`. This separate function was added to both P0 and P1. It would be advisable to update the P0 contracts to align with their P1 versions. Another instance of this is in the `rebalance` function of `p0/BackingManager.sol`.

Recommendation

Consider utilizing the `tradingContext` function in P0 contracts similarly to P1.

26. Unnecessary function call in the `refresh` function

In `p1/BasketHandler.sol`, the `_setPrimeBasket` function includes the following check:

```
// Confirm reference basket is SOUND
assetRegistry.refresh();
require(status() == CollateralStatus.SOUND, "unsound basket");
```

However, due to the design of the `refresh` function, the status has already been checked and stored in the `lastStatus` variable in the previous line.

Recommendation

Consider replacing `status()` with `lastStatus` in the aforementioned `require` check.

27. Incorrect price feed comments

The price feed assumption in `common/configuration.ts` Base chainlinkFeeds is incorrect: `WBTC: '0xccadc697c55bbb68dc5bcd8d3cbe83cdd4e071e'` on Base L2 has a deviation threshold of 0.1% and a heartbeat of 20 minutes, not the assumed 0.5% and 24 hours.

28. Unused immutable variables `rsr` and `stRSR` in `RTokenAsset`

The immutable variables `rsr` and `stRSR`, assigned in the constructor, were previously used to set the `TradingContext` in the `basketRange` function. However, this functionality has since been transferred to the `BackingManager` contract. Consequently, those variables are no longer in use and can be removed.

29. Lack of zero checks in the `MorphoNonFiatCollateral` constructor

The `MorphoNonFiatCollateral` constructor does not perform the same checks as `CTokenNonFiatCollateral`.

Recommendation

In the `MorphoNonFiatCollateral` constructor, add checks `address(targetUnitChainlinkFeed_) != address(0)` and `targetUnitOracleTimeout_ > 0`, similar to the `CTokenNonFiatCollateral`.

30. Users might trigger the RToken to trade by increasing its balance to capture MEV

Users can send tokens to the RToken to make it trade - they lose their tokens but can gain MEV + gas income (as a validator) from the trade. This might be profitable during network congestion and if the attacker is a validator or has other MEV opportunities.

Recommendation

We recommend considering virtual token balances in a future refactoring, as the fix requires significant effort and the extractable value is small.

31. `MAX_TRADE_VOLUME` is equal to `1e29` and not `1e24` as documented.

It is documented that the maximal trading volume is equal to `1e24`. However, the `MAX_TRADE_VOLUME` in `p0` and `p1` is equal to `1e29`.

Furthermore, the `MAX_TRADE_VOLUME` is not an updatable governance parameter. It is not documented if it should be a governance parameter.

Recommendation

Change the documentation or the code to be consistent. Consider making the `MAX_TRADE_VOLUME` a governance parameter.

32. Ambiguous error message

In `p1/BasketHandler.sol` a "basket unrefreshable" error message might be emitted when either the caller is not the owner or the collateral is `DISABLED` and trading is `PausedOrFrozen`.

Recommendation

We recommend using error messages that indicate singular reasons or all possible options.

33. i++ is less efficient than ++i

Throughout the codebase both forms `value++` and `++value` are used to increment counters, where `++value` is more gas efficient. In the following files inefficiencies can be found.

`i++`, `j++`, `i--`, and `j--` in `p1`

- `BasketHandler.sol`
- `Distributor.sol`
- `StaticATokenV3LM.sol`
- `CurveStableCollateral.sol`
- `ConvexStakingWrapper.sol`

and `tradesOpen++/-` in `Trading.sol`

Recommendation

We recommend using `++value` in all instances.

34. Miscellaneous

The following are some recommendations to improve the code quality and readability:

- Importing the whole `RewardableERC20` abstract contract in `CurveStableCollateral` plugin collateral is unnecessary. Using `../../../../../interfaces/IRewardable.sol` should be enough.
- Unnecessary import of `FixLib` library in the `MainP0` contract.
- Unnecessary import of `Math` and `Ownable` imports within the `p0/RToken.sol` file.
- Unnecessary import `IWSTETH` in the `L2LidoStakedEthCollateral` contract.
- Consider removing the unused `accounts` variable in the `p0/RToken.sol` file.
- Consider removing the `SafeERC20Upgradeable` import in the `TradingP1` contract as the `IERC20Upgradeable` type is never actually used.

- Wrong comment in the `settleTrade` function of `p0/BackingManager.sol`. Incorrect comment: Change `/// Settle a single trade. If DUTCH_AUCTION, try rebalance()` to `// Settle a single trade. If the caller is the trade, try rebalance()`.
- Ensure the names match between p0 and p1 versions:
 - Rename the parameter `wholeBasketsHeld` to `basketsHeldBottom` in the `compromiseBasketsNeeded` function of `p0/BackingManager.sol`.
 - Rename `p0/mixins/TradingLib.sol` to `p0/mixins/RecollateralizationLib.sol`, and `TradingLib0` to `RecollateralizationLib0`.
- Add the `// @custom:protected` comment to the `melt` function in `p0/RToken.sol`
- Complete the comments in the `_claimAssetRewards` function of the `StargateRewardableWrapper` contract.
- Update the following README's according to the latest changes:
 - `contracts/plugins/assets/yearnv2/README.md`
 - `contracts/plugins/assets/morpho-aave/README.md`
 - `contracts/plugins/assets/lido/README.md`
 - `contracts/plugins/assets/frax-eth/README.md`
 - `contracts/plugins/assets/cbeth/README.md`
- Move `contracts/plugins/assets/curve/cvx/README.md` to `contracts/plugins/assets/curve/README.md`.
- Match the import pattern across the codebase. This applies to the files in the `contracts/plugins/assets/curve` folder and `contracts/plugins/assets/curve/PoolTokens.sol`.
- Update the `SFraxEthCollateral:tryPrice` function comments, taking into account recent changes.
- Replace all instances of `> 0` with `!= 0` while working with unsigned integers to save gas.
- Use custom errors to save gas
- In `bid` of `DutchTrade.sol` follow CEI pattern and `assert(status == TradeStatus.OPEN)` right after `require(bidder == address(0))`
- Remove all typos
 - In `DutchTrade.sol` `'pariod'` should be replaced by `'period'`.
 - In `RecollateralizationLib.sol` `'deficit Deficit'` should probably be replaced by `'buy: deficit'`.

35. Known issues and limitations

The protocol has acknowledged issues found in previous audit reports (see e.g. [Audits performed by Solidified](#)) and other known limitations and issues.

- The issuance and redemption throttle can be used against each other, which might significantly impact the RToken's issuance and redemption capacities.
- The document [mev.md](#) contains advice for MEV searchers on how to interact with the protocol. This also highlights when MEV could occur.
- In [StSRS.sol](#) the following grief attack on [resetStakes\(\)](#) is described.

This function [[resetStakes\(\)](#)] is only callable when the stake rate is unsafe. The stake rate is unsafe when it is either too high or too low. There is the possibility of the rate reaching the borderline of being unsafe, where users won't stake in fear that a reset might be executed. A user may also grief this situation by staking enough RSR to vote against any reset. This standoff will continue until enough RSR is staked and a reset is executed. There is currently no good and easy way to mitigate the possibility of this situation, and the risk of it occurring is low enough that it is not worth the effort to mitigate.



Audit Report for Reserve Protocol - April 25, 2024

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUCE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

Oak Security GmbH