December 14th 2020 — Quantstamp Verified

RariCapital V2
This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type  
DeFi Aggregator

Auditors  
Ed Zuilkoski, Senior Security Engineer  
Sebastian Banescu, Senior Research Engineer  
Poming Lee, Research Engineer  
Jose Ignacio Orlicki, Senior Engineer  
Faycal Lalidji, Security Auditor

Timeline  
2020-08-10 through 2021-03-04

EVM  
Muir Glacier

Languages  
Solidity, Javascript

Methods  

Specification  
Rari Stable Pool: Smart Contracts  
Rari Yield Pool: Smart Contracts  
Rari Ethereum Pool: Smart Contracts  
Rari Governance: Smart Contracts

Documentation Quality  
Medium

Test Quality  
Low

Source Code

<table>
<thead>
<tr>
<th>Repository</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>rari-stable-pool-contracts</td>
<td>66e2dc5 (initial audit)</td>
</tr>
<tr>
<td>rari-yield-pool-contracts</td>
<td>07d301 (initial audit)</td>
</tr>
<tr>
<td>rari-ethereum-pool-fund</td>
<td>89d66d6d (initial audit)</td>
</tr>
<tr>
<td>rari-governance-contracts</td>
<td>836a8f1 (initial audit)</td>
</tr>
<tr>
<td>rari-stable-pool-contracts</td>
<td>4e024b (final audit)</td>
</tr>
<tr>
<td>rari-yield-pool-contracts</td>
<td>97a346 (final audit)</td>
</tr>
<tr>
<td>rari-ethereum-pool-fund</td>
<td>75bf255 (final audit)</td>
</tr>
<tr>
<td>rari-governance-contracts</td>
<td>832387 (final audit)</td>
</tr>
</tbody>
</table>

Total Issues  
41 (23 Resolved)

High Risk Issues  
3 (3 Resolved)

Medium Risk Issues  
6 (5 Resolved)

Low Risk Issues  
9 (% Resolved)

Informational Risk Issues  
18 (18 Resolved)

Undetermined Risk Issues  
7 (3 Resolved)

- **High Risk**: The issue puts a large number of users’ sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client’s reputation or serious financial implications for client and users.

- **Medium Risk**: The issue puts a subset of users’ sensitive information at risk, would be detrimental for the client’s reputation if exploited, or is reasonably likely to lead to moderate financial impact.

- **Low Risk**: The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client’s business circumstances.

- **Informational**: The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.

- **Undetermined**: The impact of the issue is uncertain.

- **Unresolved**: Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.

- **Acknowledged**: The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

- **Resolved**: Adjusted program implementation, requirements or constraints to eliminate the risk.

- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.
Summary of Findings

After 5th reaudit: Quantstamp has performed a reaudit of all 4 repositories which were previously audited. The report has been updated accordingly. We recommend addressing all existing issues. New issues have also been identified, which are listed at the end of the findings list, starting with QSP-18. These range across all levels of severity and should be fixed as soon as possible.

After 4th reaudit: Several new issues were found as listed below (see QSP-3н -- QSP-11, and extensions to the best practices and code documentation sections). Additionally, we were unable to successfully run all test suites due to various failures. We recommend expanding the documentation in the README, and in particular making explicit all variables that must be set in the .env file.

After 5th reaudit: The report has been updated for new commits: rari-stable-pool-contracts (f3aa246), rari-yield-pool-contracts (479a346), rari-ethereum-pool-contracts (712b256), rari-governance-contracts (83238f7). Previous issues have been resolved or acknowledged. Note that the updated report only pertains to fixes related to the previous report.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Severity</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSP-1</td>
<td>Inaccurate token prices</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-2</td>
<td>Incorrect Rari Governance Token amount (1)</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-3</td>
<td>Uninitialized_etherPriceFeed</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-4</td>
<td>Divergent mirrored states</td>
<td>Medium</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-5</td>
<td>Gas Usage / for Loop Concerns</td>
<td>Medium</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-6</td>
<td>Unchecked Return Value</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-7</td>
<td>Unfinished token upgrades</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-8</td>
<td>Incorrect value for supported currencies</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-9</td>
<td>Amount in pools may be incorrect</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-10</td>
<td>Missing input argument validation</td>
<td>Low</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-11</td>
<td>Misaligned comments and implementation</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-12</td>
<td>ETH/USD prices could be stale</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-13</td>
<td>Off-by-one error</td>
<td>Low</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-14</td>
<td>Missing input argument validation (2)</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-15</td>
<td>Privileged Roles and Ownership</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-16</td>
<td>Fallback function can receive funds from any address</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-17</td>
<td>Dangerous cost from uint256 to int256</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-18</td>
<td>Allowance Double-Spend Exploit</td>
<td>Informational</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-19</td>
<td>Unlocked Pragma</td>
<td>Informational</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-20</td>
<td>Experimental features should not be used on Mainnet deployments</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-21</td>
<td>Checks-Effects-Interactions Pattern</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-22</td>
<td>Block Timestamp Manipulation</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-23</td>
<td>Duration of RGT distribution may be different from 60 days</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-24</td>
<td>Increased loss of precision due to dividing before multiplication</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-25</td>
<td>Privileged Roles and Ownership (2)</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-26</td>
<td>Unexpected pool</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-27</td>
<td>Single point of failure for price feeds</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-28</td>
<td>Fallback function can receive funds from any address (2)</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-29</td>
<td>Potential funds stuck in contract</td>
<td>Undetermined</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-30</td>
<td>Rounding error</td>
<td>Undetermined</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-31</td>
<td>Rari Governance Tokens can still be claimed after distribution ends</td>
<td>Undetermined</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-32</td>
<td>Upgrading Fund Controller can be done when fund is enabled</td>
<td>Undetermined</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-33</td>
<td>Expired cache</td>
<td>Undetermined</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-34</td>
<td>Faulty env development might not print some enum and strict layout errors</td>
<td>Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-35</td>
<td>Missing input validation</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-36</td>
<td>Hardcoded dependency contracts</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-37</td>
<td>Privileged Roles and Ownership</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-38</td>
<td>Setter function missing event</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-39</td>
<td>Unclear addPool omission in initialize</td>
<td>Undetermined</td>
<td>Fixed</td>
</tr>
</tbody>
</table>
Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.

2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- **Slither v0.6.12**

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Inaccurate token prices

**Severity:** High Risk

**Status:** Fixed

**File(s) affected:** RariFundManager.sol

**Description:** The `getRawFundBalance()` function should return the total balance of all RFT holders' funds and all unclaimed fees of all currencies, in USD. However, the computation on L503 assumes that all currencies are worth 1 USD. This has significant impact on the entire system, including accrued interest, fees, deposits and withdrawals.

**Exploit Scenario:** We assume a malicious user called Mallory does the following steps:
1. Mallory deposits a large amount M of a token that is worth P1 less than 1 USD.
2. Mallory then withdraws an amount M of another token that is worth P2 more than 1 USD.
3. Mallory profits M*(P2-P1) from the price difference between the withdrawn and the deposited tokens.

For example if (P2-P1) is USD 1 cent and M is 1 million, then the attacker makes a profit of 10K USD from a single iteration of the exploit described above. However, an attacker can perform this attack several times to drain all funds. This is especially likely to happen with flash loans where any users can take out a large amount M and perform the exploit described above.

 Recommendation: Do not assume that all currencies are equal to 1 USD. Use secure and reliable price oracles to get the exact currency price.

**QSP-2 Incorrect Rari Governance Token amount**

**Severity:** High Risk  
**Status:** Fixed  
**File(s) affected:** RariGovernanceToken.sol

Description: There is a typo on L27 of RariGovernanceToken.sol, namely 8570000 should be 8750000 according to the comment on L23: "Initializer that reserves 8.75 million RGT for liquidity mining and 1.25 million RGT to the team." This will conflict with L157 of RariGovernanceTokenDistributor.sol: finalRgtDistribution = 8750000e18.

Recommendation: Fix the typo such that the amount is correct.

**QSP-3 Uninitialized _ethUsdPriceFeed**

**Severity:** High Risk  
**Status:** Fixed  
**File(s) affected:** RariGovernanceTokenDistributor.sol in rari-governance-contracts

Description: The AggregatorV3Interface private _ethUsdPriceFeed state variable defined on L23 in RariGovernanceTokenDistributor.sol is never initialized (assigned a value). However, it is used in the getEthUsdPrice function. This means that the getEthUsdPrice will always return 0, which will affect the Ethereum fund pool of Rari.

Recommendation: Initialize the _ethUsdPriceFeed state variable in the initialize function of the contract.

**QSP-4 Divergent mirrored states**

**Severity:** Medium Risk  
**Status:** Acknowledged  
**File(s) affected:** RariFundManager.sol, RariFundController.sol, RariFundProxy.sol

Description: There are several state variables that are mirrored in the following contracts: RariFundManager, RariFundController and RariFundProxy, namely:

1. _fundDisabled: Boolean that, if true, disables the primary functionality of the contract.
2. _rariFundRebalancerAddress: Address of the rebalancer.
3. _supportedCurrencies: Array of currencies supported by the fund.
4. _erc20Contracts: Maps ERC20 token contract addresses to supported currency codes.
5. _currencyDecimals: Maps decimal precisions (number of digits after the decimal point) to supported currency codes.
6. _poolsByCurrency: Maps arrays of supported pools to currency codes.

During deployment (before development), this creates ambiguity which makes maintainability difficult and error prone, because developers (1) might forget to update all the values of these state variables in all contracts they occur or they (2) might update the state variables with different values in different contracts. For example if new supported currencies are added any of the following input parameters could be set differently for different contracts: currencyCode, erc20Contract, decimals and pool. This would have a significant impact on the system as a whole.

After deployment the value of:

1. _fundDisabled can be set independently in different contracts by calling the disableFund and enableFund functions, which could lead the fund to be disable in one contract and enabled in the other contract. This can have an important impact on deposits, withdrawals, orders and/or approvals performed by end-users, when values are set differently during the small time window in which the 2 separate function calls are performed.
2. _rariFundRebalancerAddress can be set independently in different contracts by calling the setFundRebalancer function. This can have an important impact on deposits, withdrawals, orders and/or approvals performed by end-users, when values are set differently during the small time window in which the 2 separate function calls are performed.

Recommendation: Since these 3 contracts already have references to each other, we recommend only storing this information in one of the contracts and allowing the other contracts to access the state variables of the former contract (possibly via getter methods).

Update: From the dev team: "We certainly agree that ideally, we converge these mirrored states, but we did this to save gas, which happens to be a significant amount. We are aware of the risks associated with these mirrored states and we would certainly catch a mistake pretty easily since the tests would fail. We have ensured that our tests would catch such an error."

**QSP-5 Gas Usage / for Loop Concerns**

**Severity:** Medium Risk  
**Status:** Mitigated
File(s) affected: RariFundController.sol, RariFundManager.sol, RariFundProxy.sol

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. The following instances have been found in the code base:

1. The nested for-loops inside upgradeFundManager could reach an out-of-gas error if the total number of pools for all currencies becomes large enough. This would prevent upgrades of RariFundManagers.sol.
2. The nested for-loops inside upgradeFundController could reach an out-of-gas error if the total number of pools for all currencies becomes large enough. This would prevent upgrades of RariFundController.sol.
3. The loop inside setFundManager could reach an out-of-gas error if the number of supported currencies was too high.
4. The withdrawAndExchange function could reach an out-of-gas error if the number of InputCurrencyCodes was too high.
5. getAllBalances in RariFundController contains nested loops and a call to potentially expensive external functions inside the inner loop.
6. marketSellTradeOrdersFillOrKill contains a loop with calls to potentially expensive external functions and could reach an out-of-gas error if the number of orders was too high.
7. checkLossRateLimit contains a loop and could reach an out-of-gas error if the lossRateHistory was too long.
8. cachePoolBalances contains nested loops and could reach an out-of-gas error if the number of supported currencies and number of pools was too high.
9. The loop inside _withdrawFrom could reach an out-of-gas error if the number of pools for a given currency code was too high.
10. The exchangeAndDeposit and the withdrawAndExchange functions in RariFundProxy.sol use transfer() instead of call.value() on L203 and L259, respectively. This might have issues when gas cost changes in the future. This has happened in the Istanbul hard fork, which increased the cost such that several existing smart contracts which were using transfer() broke due to out-of-gas errors. We anticipate that gas cost will continue to change in the future.
11. The marketSellTradeOrdersFillOrKill function in RariFundController.sol uses transfer() instead of call.value() on L524. This might have issues when gas cost changes in the future.
12. The upgrade approach in initNetDeposits() might not be feasible if there are a significant number of users. Consider proxy storage approaches instead.

Recommendation:
1. Avoid loops wherever possible. Otherwise, perform gas analysis and determine the limit where the function would reach an out-of-gas error. This limit should be enforced using checks in the code.
2. Replace calls to transfer() with call.value() .
3. Consider proxy storage approaches for upgrades.

Update: From the dev team: “Fortunately, we can upgrade any function broken due to excessive gas usage as long as we can run withdrawAllFromPool for each currency of each pool and upgradeFundController(address payable newContract, address erc20Contract) individually for each currency (no loops to worry about in either of these functions). We have replaced calls to transfer() with call.value(). We have removed interestAccruedBy, in turn removing initNetDeposits. We have implemented proxy storage for most contracts.”

QSP-6 Unchecked Return Value
Severity: Medium Risk
Status: Fixed
File(s) affected: CompoundPoolController.sol

Description: Most functions will return a value indicating success or failure. It’s important to ensure that every necessary function is checked. Otherwise, the caller just assumes that the function call was successful and continues execution. This is the case for the function call cErc20.accrueInterest() on L49 in CompoundPoolController.sol, whose return value is not checked.

Recommendation: Wrap the statement in a check like so: require(cErc20.accrueInterest() == uint(Error.NO_ERROR), “accrue interest failed”);

QSP-7 Unfinished token upgrades
Severity: Medium Risk
Status: Fixed
File(s) affected: RariFundTokenUpgrader.sol

Description: If a user upgrades, but is then sent old fund tokens (which seems possible since it’s an ERC20), that user cannot upgrade the received tokens. Further, if token transfers from an already updated account occur, the conditional on L69 will never hold, because there will be old tokens in an account that cannot be upgraded (since it was already upgraded). Therefore, finished will never be set to true.

Recommendation: Clarify to end-users that once an upgrade is performed, tokens that are subsequently received cannot be upgraded. Change the strict equality conditional on L69 to allow upgrading any subset of accounts, which would not lead to out-of-gas errors.

Update: The RariFundTokenUpgrader contract has been removed.

QSP-8 Incorrect value for supported currencies
Severity: Medium Risk
Status: Fixed
File(s) affected: RariFundManager.sol in rari-stable-pool-contracts and rari-yield-pool-contracts

Description: The array index of the left-hand side member of the assignment in the following code snippet located in RariFundManager.sol does not change for any loop iteration and it is out of bounds for the acceptedCurrencies array:

```solidity
for (uint i = 0; i < _acceptedCurrencies.length; i++) if (_acceptedCurrencies[i] == acceptedCurrency) acceptedCurrencies[acceptedCurrencyIndex] = _acceptedCurrencies[i];
```

Therefore this loop will not fill in all the supported currencies as the function is expected to do and the return values will be incorrect.
QSP-9 Amount in pools may be incorrect
Severity: Medium Risk

Status: Fixed
File(s) affected: RariFundManager.sol (all repos)

Description: The issue is visible in the rari-yield-pool-contracts repo, in the _withdrawFrom function in RariFundManager.sol:
1. L666 computes: uint256 contractBalance = token.balanceOf(_rariFundControllerContract);
2. L688-893 iterate over all pools in order to withdraw the remaining balance and add it to contractBalance
3. L685 checks: require(amount <= contractBalance, "Available balance not enough to cover amount even after withdrawing from pools.");
4. L686 recomputes the same value as on L666 into another variable: uint256 realContractBalance = token.balanceOf(_rariFundControllerContract);
5. L707 checks if realContractBalance < amount <= contractBalance and transfers the resulting value.

This clearly shows that the following condition is possible: realContractBalance < amount <= contractBalance, which would indicate that the amounts withdrawn from the pools in the for-loop on L668-883 is discarded.

Recommendation: Clarify why following condition is possible: realContractBalance < amount <= contractBalance. Is this related to QSP-17? Fix the computation such that the values withdrawn from the pools is not discarded.

Update from dev team: This is not related to QSP-17. We withdraw from pools until the sum of the requested pool withdrawal amounts is greater than or equal to the amount missing from the contract balance that is necessary to cover amount. However, if a yVault pool charges a withdrawal fee, we want the user to pay this fee, so if the real contract balance after withdrawing from pools is less than the requested amount, we know a fee has been taken, and the user should pay it, so we only send them the real contract balance.

QSP-10 Missing input argument validation
Severity: Low Risk

Status: Mitigated
File(s) affected: RariFundController.sol, RariFundManager.sol, RariFundProxy.sol, AavePoolController.sol

Description: The following functions are missing validation of input arguments:
1. upgradeFundController does not validate the input parameter newContract, which could lead to sending all funds to any EOA. Fixed
2. setFundManager does not validate the input parameter newContract, which could lead to setting the fund manager to any EOA.
3. setFundController does not validate the input parameter newContract, which could lead to setting the fund controller to any EOA.
4. authorizeFundManagerDataSource does not validate the input parameter authorizedFundManagerDataSource, which could lead to setting a data source value of 0x0 for the fund manager.
5. setFundToken does not validate the input parameter newContract, which could lead to setting the token to any EOA.
6. setFundProxy does not validate the input parameter newContract, which could lead to setting the proxy to any EOA.
7. setGsnTrustedSigner does not validate the input parameter newAddress, which could lead to setting the fund manager to 0x0.
8. setInterestFeeRate() should ensure that the rate is <= 18**18. Fixed

Recommendation: Add input argument validation to every function where it is needed. Check if addresses are different from 0x0 and/or if necessary check if addresses represent smart contracts or EOAs.

Update: Only 2 out of the 8 items above have been fixed. From the dev team: "We have added additional input validation where necessary, particularly in upgradeFundController."

QSP-11 Misaligned comments and implementation
Severity: Low Risk

Status: Fixed
File(s) affected: RariGovernanceToken.sol

Description: The comment on L23 says 20 million tokens will be minted, but on L27 only 10 million are minted.

Recommendation: Align the comment and the implementation such that the right number of tokens are minted.

QSP-12 ETH/USD prices could be stale
Severity: Low Risk

Status: Acknowledged
File(s) affected: RariGovernanceTokenDistributor.sol, RariFundPriceConsumer.sol

Description: The following functions do not check if the ETH/USD price is stale:
1. RariGovernanceTokenDistributor.getEthUsdPrice in rari-governance-contracts

According to the Chainlink documentation:

- **under current notifications**: "If answeredInRound < roundId could indicate stale data."
- **under historical price data**: "A timestamp with zero value means the round is not complete and should not be used."

**Recommendation**: We recommend adding `require` statements that check for the aforementioned conditions in all the occurrences of those functions.

**Update from dev team**: We will add validation to check if the ETH/USD price is stale in the next version of the contracts.

**QSP-13 Off-by-one error**

**Severity**: Low Risk

**Status**: Mitigated

**File(s) affected**: `RariFundToken.sol`

**Description**: There is a recurring condition that appears in 6 methods inside the `RariFundToken` contract, namely: `if (address(rariGovernanceTokenDistributor) != address(0) && block.number > rariGovernanceTokenDistributor.distributionStartBlock())`, which appears in the following functions: `transfer`, `transferFrom`, `mint`, `burn`, `burnFrom` and `fundManagerBurnFrom`.

The second clause in the aforementioned condition is off-by-one, because it only allows claiming RGT one block after the distribution has started.

**Recommendation**: Change the sign from `<` to `>=` such that the `-condition will allow claiming RGT as soon as distribution starts.

**Update from dev team**: No Rari Governance Tokens have been distributed at block zero of the distribution period. Only in the next block have any tokens been distributed.

**QSP-14 Missing input argument validation (2)**

**Severity**: Low Risk

**Status**: Acknowledged

**File(s) affected**: `RariFundController.sol`, `RariFundManager.sol`

**Description**: The following functions are missing input parameter validation:

1. `RariFundController.setFundManager` in `rari-ethereum-pool-fund` does not validate the `newContract` parameter of type `address`.
2. `setFundRebalancer` in all repos and all contracts does not check the `newAddress` parameter of type `address`.
3. `setFundPriceConsumer` in all repos does not check the `newContract` parameter of type `address`.

**Recommendation**: Add input argument validation to every function where it is needed. Check if addresses are different from `0x0` and/or if necessary check if addresses represent smart contracts or EOAs.

**Update from dev team**: These input validation functions will be added in the next version of the contracts.

**QSP-15 Privileged Roles and Ownership**

**Severity**: Informational

**Status**: Acknowledged

**File(s) affected**: `RariFundController.sol`, `RariFundManager.sol`

**Description**: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. There are multiple privileged roles in the system, including: contract owners, rebalancers and Rari fund managers/controllers.

1. The owner of the `RariFundController` contract is allowed to:
   - disable and enable the Rari fund at any point in time.
   - set the daily loss rate limit to any value at any time.
   - forward all funds in the contract to any EOA.
   - change the `RariFundToken` and `RariFundProxy` address at any time.

2. The Rari Fund rebalancer is allowed to:
   - withdraw all funds from any and all pools at any time.
   - approve any amount to 0x exchange.
   - create sell orders on the 0x exchange.

3. The owner of the `RariFundManager` contract is allowed to withdraw all funds (of any token type, including ETH) out of this smart contract to their own account.

**Recommendation**: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

**Update**: New documentation has been added to `CONCEPT.md`.

**QSP-16 Fallback function can receive funds from any address**

**Severity**: Informational
QSP-17 Dangerous cast from uint256 to int256
Severity: Informational
Status: Fixed
File(s) affected: RariFundManager.sol
Description: There is a cast to int256 on L515 in the RariFundManager, which would cause a large enough unsigned value to be converted to a negative value. However, this is highly unlikely to occur.
Recommendation: Add an assertion statement to check if the uint256 is larger than the highest positive number that can be stored in int256, before the cast.

QSP-18 Allowance Double-Spend Exploit
Severity: Informational
Status: Mitigated
File(s) affected: ERC20RFT.sol
Description: As it presently is constructed, the contract is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.
Exploit Scenario: An example of an exploit goes as follows:
1. Alice allows Bob to transfer amount of Alice’s tokens (N) by calling the approve() method on Token smart contract (passing Bob’s address and N as method arguments)
2. After some time, Alice decides to change from N to M (N > M) the number of Alice’s tokens Bob is allowed to transfer, so she calls the approve() method again, this time passing Bob’s address and M as method arguments
3. Bob notices Alice’s second transaction before it was mined and quickly sends another transaction that calls the transferFrom() method to transfer N Alice’s tokens somewhere
4. If Bob’s transaction will be executed before Alice’s transaction, then Bob will successfully transfer N Alice’s tokens and will gain an ability to transfer another M tokens
5. Before Alice notices any irregularities, Bob calls transferFrom() method again, this time to transfer M Alice’s tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as increaseAllowance and decreaseAllowance.
Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on approve() / transferFrom() should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

Update: From dev team: We have added notices about this exploit in the documentation for Rari Fund Token (RFT) in API.md and USAGE.md.

QSP-19 Unlocked Pragma
Severity: Informational
Status: Fixed
File(s) affected: All contracts
Description: Every Solidity file specifies in the header a version number of the format pragma solidity (^)0.5.*. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version and above, hence the term "unlocked."

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version. Since the project uses external libraries, which together would only support at least version 0.5.9 of the Solidity compiler, the pragma should be locked at a version of solidity great or equal to 0.5.9.

QSP-20 Experimental features should not be used on Mainnet deployments
Severity: Informational
Status: Mitigated
File(s) affected: Several contracts
Description: Until solidity 0.6.6, the ABIEncoderV2 feature is still technically in experimental state. Although there are no known security risks associated with it, these features should be used judiciously.

Recommendation: Upgrade the contracts to a more recent solidity version such as 0.5.16 or 0.6.6. All contracts that depend upon ABIEncoderV2 functionality should be tested thoroughly.

Update: From dev team: "We have locked all Solidity version pragmas to 0.5.17."

QSP-21 Checks-Effects-Interactions Pattern
QSP-22 Block Timestamp Manipulation

**Severity:** Informational

**Status:** Fixed

**File(s) affected:** RariFundManager.sol

**Description:** Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps up to 900 seconds, for their own purposes. If a smart contract relies on a timestamp, it must take this into account.

**Recommendation:** Use `block.nounce` instead of `block.timestamp` to avoid manipulation. Or clearly document that a 900 second error is possible and acceptable and would not have any impact on the actual logic, because the loss rates in the `lossRateHistory` are not that different from each other.

**Update:** From dev team: "We have added the suggested notice. We will note that in this case, it doesn't really matter in this case if the 1 day measurement is off by <= 900 seconds (15 min) as the loss rate limit does not need to be this precise."

QSP-23 Duration of RGT distribution may be different from 60 days

**Severity:** Informational

**Status:** Fixed

**File(s) affected:** RariGovernanceTokenDistributor.sol

**Description:** The duration of the distribution period is set to 345600 blocks on L152 in RariGovernanceTokenDistributor.sol. Assuming that the average block duration over a 60 day period is 15 seconds results in 60 days. However, according to the latest statistics on Etherscan we foresee an average block duration of 13 seconds, which would reduce the distribution period to 52 days. However, this is also an approximate estimate as the actual duration could be even lower.

**Recommendation:** Add information to the user-facing documentation, which indicates that the duration of the distribution period is 345600 blocks starting with which block such that it is clear to end-users when the distribution period ends.

**Update from dev team:** The distribution period has been changed to 390000 blocks (i.e., 6500 blocks per day or approximately 11.292 seconds per block). We have added the suggested notice to README.md and CONCEPT.md.

QSP-24 Increased loss of precision due to dividing before multiplication

**Severity:** Informational

**Status:** Acknowledged

**File(s) affected:** RariFundProxy.sol (in all repos), MStablePoolController.sol (rari-stable-pool-contracts and rari-yield-pool-contracts), RariFundManager.sol (rari-stable-pool-contracts and rari-yield-pool-contracts), RariFundPriceConsumer.sol (rari-stable-pool-contracts and rari-yield-pool-contracts), RariGovernanceTokenDistributor.sol (rari-governance-contracts)

**Description:** To reduce the loss of precision caused by integer division, multiplication should always be performed before division. Several locations in the code were identified where this rule is not satisfied and hence a larger loss of precision is possible:

1. In `RariFundProxy.withdrawAndExchange` the division in the following assignment `uint256 outputAmount = amount.mul(1e18).div(exchangeRate);` is performed before the multiplication in the following assignment `uint256 outputAmount = outputAmount.mul(outputDecimals).div(10 ** (outputDecimals));`.
2. In `MStablePoolController.withdraw` the division in the following assignment `credits = amount.mul(1e18).div(exchangeRate);` is performed before the multiplication in the following assignment `credits = credits.mul(exchangeRate).div(1e18);`.
3. In `RariGovernanceTokenDistributor.depositTo` the division in the following assignment `uint256 amount = amount.mul(1e18).div(totalSupply);` is performed before the multiplication in the following assignment `uint256 amount = amountUsd.mul(1e18).div(1e8);`.
4. In `RariFundPriceConsumer.getUsdPrice` the following assignment contains a division before the last multiplication `usdSupplyScaled = usdSupplyScaled.add(bAssets[i].vaultBalance.mul(bAssets[i].ratio).div(1e18).mul(bAssetUsdPrices[i]));`.
5. In `RariGovernanceTokenDistributor.storeRgtDistributedPerRft` the following assignment contains a division before the last multiplication `_rgtPerRftAtLastSpeedUpdate[i_scope_0] = _rgtPerRftAtLastSpeedUpdate[i_scope_0].add(rgtToDistribute.mul(ethFundBalanceUsd).div(fundBalanceSum).div(1e18));`.
6. In `RariGovernanceTokenDistributor.storeRgtDistributedPerRft` the following assignment contains a division before the last multiplication `_rgtPerRftAtLastSpeedUpdate[i_scope_0] = _rgtPerRftAtLastSpeedUpdate[i_scope_0].add(rgtToDistribute.mul(ethFundBalanceUsd).div(1e18).div(1e8));`.
7. In `RariGovernanceTokenDistributor.getRgtDistributedPerRft` the following assignment contains a division before the last multiplication `_rgtPerRftAtLastSpeedUpdate = uint8(7); add(_rgtToDistribute.mul(_fundBalancesCache[uint8(pool)]).div(fundBalanceSum).div(1e18));`.
8. In `RariGovernanceTokenDistributor.getRgtDistributedPerRft` the following assignment contains a division before the last multiplication `_rgtPerRftAtLastSpeedUpdate = uint8(7); add(_rgtToDistribute.mul(_fundBalancesCache[uint8(pool)]).div(1e18).div(totalSupply));`.
QSP-25 Privileged Roles and Ownership (2)

Severity: Informational

Status: Acknowledged

File(s) affected: RariFundToken.sol (all repos), RariGovernanceTokenDistributor.sol, RariFundController.sol in rari-echecuenum-pool-fund

Description: 1. The minter of the RariFundToken is allowed to set the `RariGovernanceTokenDistributor` address to any value at any point in time (even if the new `RariGovernanceTokenDistributor` is disabled) if the `force` parameter is set to `true`. It is not clear how, when or why the `force` parameter would be used in `setGovernanceTokenDistributor()`. To prevent revert if the validation checks existent in that function would fail.

1. The owner of the `RariGovernanceTokenDistributor` contract can:
   - Enable and disable the distribution at any time, multiple times.
   - Set the governance token, fund token and fund manager addresses to any non-zero address when the distribution is disabled.
   - Upgrade the contract address to any address, which transfers all RGTs to that address.

2. The owner of `RariFundController` can set the address of the `_rariFundManagerContract` to any address including a EOA and then use that address to withdraw all the funds from all pools using the `withdrawToManager` and/or `withdrawFromPoolKnowingBalanceToManager` functions.

3. The owner of the `RariFundManager` can:
   - Upgrade the fund manager contract.
   - Authorize any address to be the fund manager data source.
   - Set the fund controller, fund proxy, fund rebalancer and fund token to any address.
   - Set the interest fee rate to values even higher than 100%.
   - Set the interest fee master beneficiary to any address different from zero.

Recommendation: Warn end-users about this privileged action that a minter can make and about the consequences via publicly available documentation. Consider adding a validity check for when `force` can be set to `true`.

Update from dev team: We have added a warning to end-users about the privileges of the contract administrators and their potential consequences in `CONCEPT.md`. However, we will soon be relinquishing control of the contracts to the Rari Governance Token holders.

QSP-26 Unexpected pool

Severity: Informational

Status: Acknowledged

File(s) affected: RariGovernanceTokenDistributor.sol

Description: In `RariGovernanceTokenDistributor.sol` and `rari-governance-contracts`, the functions `setManager`, `setFundToken`, `beforeFirstPoolTokenTransferIn`, `getUnclaimedRgt`, `_claimRgt`, `claimRgt` and `refreshDistributionSpeeds` have an input parameter called `pool` of type `RariPool`, which is an enum with 3 values. When end-users call these functions they will be able to pass in an integer value for this parameter, which could be higher than 3, which is the highest value allowed by the enum. This will cause the function to throw without any explicit error message and might be confusing to the end-user as to why the function reverted.

Recommendation: These functions should have a `require` statement that the input parameter `pool` is strictly smaller than 3 and if not it should revert with an error message that tells the user to only use pool values less than 3.

Update from dev team: This input validation function will be added in the next version of the contracts.

QSP-27 Single point of failure for price feeds

Severity: Informational

Status: Acknowledged

File(s) affected: RariGovernanceTokenDistributor.sol, RariFundPriceConsumer.sol

Description: The price feeds rely on a single oracle, namely the Chainlink Aggregator V3, which is indeed robust. However, in the event of any large-scale attack/disruption of the Chainlink network, Rari Capital would be impacted severely.

Recommendation: Consider adding at least one other robust price feed, which is independent of Chainlink.

Update from dev team: We plan to add another robust price feed independent of Chainlink in the next version of our contracts, likely the Coinbase price oracle.

QSP-28 Fallback function can receive funds from any address (2)
Severity: Informational
Status: Acknowledged
File(s) affected: RariFundController.sol in rari-ethereum-pool-fund, RariFundProxy.sol in rari-ethereum-pool-fund
Description: The fallback function is meant to only be "called by Ox exchange to refund unspent protocol fee." However, there are no restrictions/checks in place to guarantee this. This means that anyone could send funds to this contract by mistake.
Recommendation: Add a requirement inside the fallback function to check if the msg.sender address belongs to Ox, as is already done in the same function and contract from the rari-stable-pool-contracts repo. This way the function will revert if any other address sends funds to it.
Update from dev team: This address validation function will be added in the next version of the contracts.

QSP-29 Potential funds stuck in contract
Severity: Undetermined
Status: Acknowledged
File(s) affected: RariFundProxy.sol
Description: In withdrawAndExchange(), does there need to be a check that all orders obtain tokens of the same type (corresponding to outputErc20Contract). For example, suppose one order obtained WETH and another contained DAI, and outputErc20Contract = address(0). Wouldn’t the DAI funds be stuck in the contract until another withdrawAndExchange() transaction occurs with outputErc20Contract = DAI?
Recommendation: Add check that all orders obtain tokens of the same type (corresponding to outputErc20Contract)
Update: From dev team: It costs us a good bit of additional gas to validate all orders, and we want to avoid gas costs as much as possible in the exchangeAndDeposit and withdrawAndExchange functions. Assuming the user's client has not made a mistake, lack of validation on the contract side should not be necessary. However, we will write tests to confirm this could not be an issue in the official SDK, which will soon replace this logic in the web client.

QSP-30 Rounding error
Severity: Undetermined
Status: Fixed
File(s) affected: MStablePoolController.sol
Description: In the function withdraw(), the amount of withdrawal credits is rounded up on L81. It seems that if all users would choose to redeem credits and some would get rounded-up, then the last user to withdraw would fail due to lack of credits.
Recommendation: Round down instead of rounding up. However, if this is indeed the correct logic, the following change could optimize L80-81 to "always round up":
uint256 credits = amount.mul(1618).sub(1).div(exchangeRate), add(1);
Update: The dev team has indicated that this is indeed the correct logic. The test 5_fund_user.js should demonstrate that this practice of rounding is not an issue. The following is an explanation provided by the dev team about why these rounding operations work correctly:
RariFundManager.withdrawFrom is configured not to withdraw more than the mUSD balance in mStable savings (i.e., the output mUSD amount of a withdrawal of all available credits), which is rounded down. Because this mUSD quantity is rounded down, when MStableExchangeController.withdraw is called, the conversion of this mUSD quantity back to credits could underestimate the credits necessary to output this amount by 1 (because Solidity, by default, rounds the quotient of a division operation down). To avoid this, we round up the quantity of credits to withdraw so we make sure to withdraw at least the requested output mUSD amount. These calculations will never cause the quantity of credits to withdraw to exceed the available quantity.

QSP-31 Rari Governance Tokens can still be claimed after distribution ends
Severity: Undetermined
Status: Acknowledged
File(s) affected: RariFundToken.sol
Description: There is a recurring condition that appears in 6 methods inside the RariFundToken contract, namely:
if (address(rariGovernanceTokenDistributor) != address(0) & block.number > rariGovernanceTokenDistributor.distributionStartBlock()), which appears in the following functions: transfer, transferFrom, mint, burn, burnFrom and FundManagerBurnFrom.
This condition does not check whether the current block number is past the end block of the distribution.
Recommendation: Clarify if Rari Governance Tokens can still be claimed after distribution ends. If this should not be allowed, then add the following clause to the conjunction block.number < rariGovernanceTokenDistributor.distributionEndBlock().
Update from dev team: Rari Governance Tokens can indeed be claimed at any time after the starting block of the distribution period.

QSP-32 Upgrading Fund Controller can be done when fund is enabled
Severity: Undetermined
Status: Acknowledged
File(s) affected: RariFundController.sol
Description: In RariFundController.upgradeFundController() function in both the rari-ethereum-pool-fund and rari-stable-pool-contracts repos, it is not required that the fund is disabled, unlike the same function in the rari-yield-pool-contracts repo. It is not clear if this is intentional or not.
Recommendation: Clarify if the Fund Controller can be upgraded even when the fund is enabled. If not, add the same require statement from the rari-yield-pool-contracts repo to the other 2 repos. Otherwise, remove that require statement.
Update from dev team: These updates are planned for the next version of the rari-stable-pool-contracts and rari-ethereum-pool-contracts repos. When we added this feature to the rari-yield-pool-contracts before deployment, we did not consider this single feature important enough to redeploy the existing Stable Pool and Ethereum Pool implementation contracts.
QSP-33 Expired cache

Severity: Undetermined
Status: Acknowledged
File(s) affected: RariFundManager.sol

Description: The functions _depositTo, _withdrawFrom, and withdrawFees in RariFundManager.sol at rari-ethereum-pool-fund do not update _rawFundBalanceCache at all, which is different from the behavior of the same functions in the other repositories: rari-stable-pool-contracts and rari-yield-pool-contracts.

Recommendation: Clarify if this behavior is intentional. If not, update the _rawFundBalanceCache similarly to the other repos.

Update from dev team: Usage of _rawFundBalanceCache was temporarily removed in the Rari Ethereum Pool, but we will be restoring this code in a later version of the contracts.

QSP-34 Faulty dev environment might not print some enum and struct layout errors

Severity: Low Risk
Status: Fixed
File(s) affected: rari-yield-pool-contracts/package.json

Description: As described in this forum update, truffle-upgrades supports the unsafeAllowCustomTypes flag. Due to an implementation error of this flag, older versions might not display some storage layout errors for enum and struct.

Recommendation: Upgrade to the latest version of truffle-upgrades, at least 1.3.1.

QSP-35 Missing input validation

Severity: Low Risk
Status: Acknowledged
File(s) affected: RariFundController.sol, RariFundManager.sol

Description: The following function should perform additional checks:

1. RariFundController.setEnzymeComptroller should ensure that comptroller is non-zero.
2. RariFundController.marketsSellXOrdersFillOrKill should ensure that inputErc20Contract is non-zero.
3. RariFundController.setFundManager should ensure that newContract is non-zero.
4. RariFundController.setFundRebalancers should ensure that newAddress is non-zero.
5. RariFundManager.authorizeFundManagerDataSource should ensure that authorizedFundManagerDataSource is non-zero.
6. RariFundManager.setFundController should ensure that newContract is non-zero.
7. RariFundManager.setFundToken should ensure that newContract is non-zero.
8. RariFundManager.setFundProxy should ensure that newContract is non-zero.
9. RariFundManager.setFundRebalancer should ensure that newContract is non-zero.

Recommendation: Add corresponding require statements to above.

QSP-36 Hardcoded dependency contracts

Severity: Low Risk
Status: Acknowledged

Description: Controllers of external contracts have hardcode addresses. In the extreme scenario where these contracts are replaced by new ones (hard upgrade) or the community capital migrates to similar ones (code fork), there is no way to upgrade the addresses of the controlled contracts.

Recommendation: Add functions to update the controlled addresses, either by owner or governance.

QSP-37 Privileged Roles and Ownership

Severity: Informational
Status: Fixed
File(s) affected: RariFundController.sol

Description: Smart contracts will often have owner variables to designate the person with special privileges to make modifications to the smart contract. In particular:

1. In RariFundController.sol, the owner can invoke setEnzymeComptroller at any point.
2. In RariGovernanceTokenUniswapDistributor.sol, if the owner changes the Uniswap pair address using setRgtEthUniswapV2Pair, users that have already deposited may not be able to withdraw.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner. Consider disallowing changes
Additional slither has found that solidity naming conventions have not been respected: Slither has detected a total of 226 issues. We have marked the majority as false positives. Some of the issues were incorporated in the finding and best practices sections.

**QSP-38** Setter function missing event  
**Severity:** Informational  
**Status:** Acknowledged  
**File(s) affected:** RariFundManager.sol  
**Description:** Although the other setters are emitting events, the setter `setFundManagerData()` has no corresponding event declared and is not emitting any event.  
**Recommendation:** Add an event to the function.

**QSP-39** Unclear addPool omission in initialize  
**Severity:** Undetermined  
**Status:** Fixed  
**File(s) affected:** rari-eth-pool-contracts/contracts/RariFundManager.sol  
**Description:** In `initialize`, it is unclear why `addPool(5)` which would correspond to Enzyme, is not invoked.  
**Recommendation:** Clarify if this is omitted intentionally.

**QSP-40** No example of token distribution implementation is presented  
**Severity:** Undetermined  
**Status:** Fixed  
**File(s) affected:** rari-eth-pool-contracts/.../IRariGovernanceTokenDistributor.sol  
**Description:** `distributionEndBlock` is a critical parameter used on the interface for governance distribution. This method is only called in RariFundManager when the block number is below `distributionEndBlock`. No example implementation of `IRariGovernanceTokenDistributor` is presented, and `distributionEndBlock` must be used there to make this parameter effective.  
**Recommendation:** Make sure to include an example implementation and check that after block number `distributionEndBlock` that minter cannot be executed.  
Update from the Rari Capital team: Tests for token distribution are available in rari-governance-contracts, which test both the governance token distribution contracts and the pool contracts in tandem.

**QSP-41** Controller unable to pause specific stablecoins  
**Severity:** Low Risk  
**Status:** Acknowledged  
**File(s) affected:** rari-eth-pool-contracts/.../DydxPoolController.sol, rari-eth-pool-contracts/.../AavePoolController.sol  
**Description:** Stablecoins, like other financial assets on-chain, are likely victims of recurrent attacks or attempted attacks. In case of specific disruption of any stablecoin, the controllers of the pools have no functionality to selectively pause or stop the use of particular stablecoin.  
**Recommendation:** Add functionality to pause specific stablecoins or update the addresses of any stablecoin contract.

---

**Automated Analyses**

**Slither**

Slither has detected a total of 226 issues. We have marked the majority as false positives. Some of the issues were incorporated in the finding and best practices sections. Additionally slither has found that solidity naming conventions have not been respected:

```solidity
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

contract SlitherContract {
    // Function LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes,uint256,uint256) (@0x/contracts-utils/contracts/src/LibBytesRichErrors.sol#40-55) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Variable Migrations.last_completed_migration (Migrations.sol#14) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Parameter Migrations.upgrade(address).new_address (Migrations.sol#28) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Constant DydxPoolController._soloMargin (lib/pools/DydxPoolController.sol#43) is not in UPPER_CASE_WITH_UNDERSCORES
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Function LibRichErrors.StandardError(string) (@0x/contracts-utils/contracts/src/LibRichErrors.sol#34-45) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Variable RariFundController._aaveReferralCode (RariFundController.sol#342) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Variable RariFundController._poolsWithFunds (RariFundController.sol#328) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Function RariFundController._getPoolBalance(uint8,string) (RariFundController.sol#265-272) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Variable RariFundManager._poolBalanceCache (RariFundManager.sol#398) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Variable RariFundManager._cacheDydxBalances (RariFundManager.sol#393) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Variable RariFundManager._cachePoolBalances (RariFundManager.sol#388) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Constant AavePoolController._lendingPool (lib/pools/AavePoolController.sol#41) is not in UPPER_CASE_WITH_UNDERSCORES
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Constant ZeroExExchangeController._exchange (lib/exchanges/ZeroExExchangeController.sol#44) is not in UPPER_CASE_WITH_UNDERSCORES
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Function LibSafeMathRichErrors.Uint256DowncastError(LibSafeMathRichErrors.DowncastErrorCodes,uint256) (@0x/contracts-utils/contracts/src/LibSafeMathRichErrors.sol#45-58) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Function LibMathRichErrors.RoundingError(uint256,uint256,uint256) (@0x/contracts-exchange-libs/contracts/src/LibMathRichErrors.sol#23-38) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Function LibMathRichErrors.DivisionByZeroError() (@0x/contracts-exchange-libs/contracts/src/LibMathRichErrors.sol#15-21) is not in mixedCase
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }

    // Constant RariFundProxy._weth (RariFundProxy.sol#115) is not in UPPER_CASE_WITH_UNDERSCORES
    function libraries() internal pure returns (bytes) {
        return LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes(uint256(1), uint256(2)));
    }
}
```
Code Documentation

We have identified the following issues in the code documentation:

1. Overall more code comments should be used to describe non-trivial lines of code or sequences of lines of code.
2. [Fixed] L49 in AavePoolController.sol. "dfvA" should be "Aave".
3. It appears that if (amount > 0 & allowance > 0) token.safeApprove(); is being used to prevent the allowance double-spend exploit in all pool controllers. While this may work, the functionality may be unintuitive to the user. The documentation should reflect this approach, which is not common in ERC20 contracts.
4. [Fixed] L210 in RariFundProxy.sol. "@notice Exchanges and deposits funds to RariFund in exchange for RFT." does not match the function (copy+paste of L149)
5. [Fixed] On L556 in RariFundManager.sol, the comment "Maps booleans indicating if Ethereum addresses are immune to the account balance limit." does not reflect the mapping below, which has no Booleans.

Adherence to Best Practices

We have identified the following deviations from best-practices:

1. In AlphaPoolController.sol on L44, the comment "Assumes that you have already approved == the amount to the BeETH token contract." is not relevant here, as it is on ETH transfer to the 'BeETH' contract.
2. In AlphaPoolController.withdraw, it should be made explicit that amount refers to an ETH amount, contrary to the comment on L54 stating "the amount of tokens to be withdrawn.
3. In RariGovernanceToken.sol several magic constants (e.g. exchangeLiquidityRewards) are used that should be further documented.
4. In RariGovernanceTokenUniswapDistributor.deposit, the first comment "Transfer RGT in from sender" should instead say "Transfer LP Tokens in from sender".

As of the 4th reaudit:

1. In AlphaPoolController.sol on L44, the comment "Assumes that you have already approved == the amount to the BeETH token contract." is not relevant here, as it is on ETH transfer to the 'BeETH' contract.
2. In AlphaPoolController.withdraw, it should be made explicit that amount refers to an ETH amount, contrary to the comment on L54 stating "the amount of tokens to be withdrawn.
3. In RariGovernanceToken.sol several magic constants (e.g. exchangeLiquidityRewards) are used that should be further documented.
4. In RariGovernanceTokenUniswapDistributor.deposit, the first comment "Transfer RGT in from sender" should instead say "Transfer LP Tokens in from sender".

Typos EETH on L378 in RariFundManager.sol rari-ethereum-pool-fund.

We have identified the following deviations from best-practices:

1. Many protocol and token addresses are re-used throughout (e.g., DAI). Would be good to define and reuse constants for these addresses.
2. The layout of the code should be consistent. It is often the case that one or more control flow statements (e.g. loops or branches) are written on one line and other times on multiple lines.
3. Complex statements that span more than 80 characters should be split over multiple lines for readability. For example, L181 in DydxPoolController.sol seem to correspond to here: https://docs.dydx.exchange/#solo-get-v1-markets
4. Complex functions such as storeRgtDistributedPerRft could use more inline documentation in order to indicate what the intention behind the code is. Otherwise, independent auditing is hampered.
5. Typos EETH on L378 in RariFundManager.sol rari-ethereum-pool-fund.

TODOS should be removed before publishing the code. There are 7 TODOS present in the code comments. Some of them are concerning:

1. [Fixed] TODO: Factor in prices; for now we assume the value of all supported currencies = $1
2. TODO: Support orders with taker fees (need to include taker fees in loss calculation)
3. TODO: Or revert("No funds available to redeem from Compound cToken.
4. TODO: Import from rari-contracts-governance repository on L19 in RariFundToken.sol
5. getFundBalance, getRawFundBalance, getInterestFeesUnclaimed should be view functions
6. Avoid using inline constants. Use named constants instead. For example:
   - the constant value 18 is used repeatedly in multiple files.
16. Code clones should be avoided, because it decreases the maintainability of the code. Example of code clones in the smart contracts are:

- The `FundEnabled` and `onlyRebalance` modifiers are declared in both `RariFundController.sol` and `RariFundManager.sol`.

- Several state variables are declared in both `RariFundController.sol` and `RariFundManager.sol`, namely: `supportedCurrencies`, `currencyDecimals`, `erc20Contracts`, and `pooledByCurrency`. There is no need to keep this state information in both contracts.

- Constructors of `RariFundController.sol` and `RariFundManager.sol` are identical.

- `addSupportedCurrency`, `addPoolToCurrency`, `setFundRebalancers`, `disableFund`, and `enableFund` functions are declared in both `RariFundController.sol` and `RariFundManager.sol`.

- L267-269, L777-779, L983-980 in `RariFundManager.sol` are clones

17. Duplicate checks can be removed to save gas. For example:

- L316 in `RariFundController.sol` checks if `rariFundManagerContract != address(0)` and then calls `token.safeApprove(_rariFundManagerContract, 0)`. However, the `safeApprove` function also performs the check if `_rariFundManagerContract` is different from `0x0`. Therefore, this check can be removed.

- L317 in `RariFundController.sol` checks if `newContract != address(0)` and then calls `token.safeApprove(newContract, uint256(-1))`. However, the `safeApprove` function also performs the check if `newContract` is different from `0x0`. Therefore, this check can be removed.

18. Checks that do not depend on the loop iterator can be extracted outside of the loop to save gas.

19. All dependency versions inside `package.json` should be specified and locked. Avoid using the caret sign to allow different versions. This can cause issues when running tests, reproducing bugs and most importantly different behavior in production than was observed locally. We recommend locking the version of all dependencies in `package.json`.

20. **[Fixed]** The import `.\RariFundProxy.sol` on L25 in `contracts/RariFundManager.sol` creates a cyclic dependency graph, because the `RariFundProxy.sol` also imports `RariFundManager.sol`. This may cause errors in static analyzers and compilers. Remove the import `".\RariFundProxy.sol"` on L25 in `contracts/RariFundManager.sol`.

21. **[Fixed]** Variable shadowing should be avoided. For example the `owner` input parameter of the `allowance` and `approve` functions inside `ERC20RFT.sol` are shadowing the inherited `owner` state variable from `Ownable.sol`. This makes the use of `owner` ambiguous.

22. There are two different licenses are used throughout the repos. We recommend choosing a single license and removing the other one.

23. L97-98, L218-221, 302-305, 317-320, 370-373 in `RariFundController.sol` in `rari-ethereum-pool-fund` should use an `enum` instead of the constants 0-3, similarly to the other repos.

24. The `RariFundProxy.sol` uses several magic numbers in the form of Ethereum addresses. There are 23 occurrences in that file alone and 9 of these occurrences are for addresses `0x223993934545a84b06380b45b4705f1555a5`. These magic numbers should be defined as named constants such that it is clear what the address refers to without having to look it up.

25. The `refreshDistributionSpeeds` function defined on L128 clones the code of the `refreshDistributionSpeeds` function defined on L122. Instead it could just call that function with a value for `newBalance == rariFundManagers[uint32(r))].getFundBalance()`.

26. The magic number 3 is used about 22 times in the `RariGovernanceTokenDistributor` contract due to the length of the `enum` `RariPool`. We recommend replacing it with a named constant, since it will improve code readability and make it easier to maintain if new items are added to the enum in the future.

27. The magic number 2 is used about 12 times in the `RariGovernanceTokenDistributor` contract instead of the `RariPool.Ethereum`. We recommend replacing it with `RariPool.Ethereum`, since it will improve code readability and make it easier to maintain if new items are added to the enum before `RariPool.Ethereum` in the future.

28. LV5 in `CompoundPoolController.sol` contains commented code and should be removed.

As of the `4th` reaudit:
1. In `RariFundController.sol`, the `LiquidityPool` enum should be used throughout instead of integer constants.
2. In `RariGovernanceTokenUniswapDistributor.sol`, the "double-if" statement on L135 should be refactored into a nested if-statement for better readability.

## Test Results

### Test Suite Results

It appears that the test suites have either several failing tests, or have compatibility issues with our environment.

We have included the output from `rari-governance-contracts` below.
Warning: Potentially unsafe deployment of DummyRariFundManager

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums. Make sure you have manually checked the storage layout for incompatibilities.

1) should deposit to the fund, approve deposits to pools via RariFundController.approveToPool, and deposit to pools via RariFundController.depositToPool (591ms)
   - should withdraw half from all pools via RariFundController.withdrawFromPool (347ms)
   - should withdraw everything from all pools via RariFundController.withdrawFromAllPools (304ms)

2) should claim native USDC rewards (37ms)

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!

Warning: Could not decode event!
Warning: Could not decode event!
Warning: Could not decode event!
Warning: Could not decode event!

RariFundController.PoolsAllocation
- owner: 0x1234567890123456789012345678901234567890123456789012345678901234
- shares: {<index> uint256
- pool: {<index> RariFundController.LiquidityPool.dYdX
- currency: {<index> string

Contract: RariFundManager

Gas usage of RariFundController.upgradeFundController: 4079092

5) Contract: RariGovernanceTokenVestingV2
4) Contract: RariGovernanceTokenDistributorV2
3) Contract: RariGovernanceTokenUniswapDistributor
2) Contract: RariGovernanceTokenVesting
1) Contract: RariGovernanceTokenDistributor

2 passing (3s)
3 passing (43s)
11 passing (49s)
2 passing (7s)
0 failing

1)  should distribute tokens evenly across pools
2)  should have distributed the correct amount of tokens at each checkpoint
3)  should distribute tokens evenly across pools
4)  should have distributed the correct amount of tokens at each checkpoint
5)  should distribute tokens evenly across pools

19 passing (5m)

5) should distribute tokens evenly across pools
4) should have distributed the correct amount of tokens at each checkpoint
3) should distribute tokens evenly across pools
2) should vest private token distributions
1) should distribute tokens evenly across pools

1 failing
1) should have distributed the correct amount of tokens at each checkpoint

19 passing (5m)

Warning: Potentially unsafe deployment of DummyRariFundManager

Warning: Potentially unsafe deployment of RariFundManager

Warning: Potentially unsafe deployment of RariFundManager

Warning: Potentially unsafe deployment of RariFundManager

Warning: Potentially unsafe deployment of RariFundManager

Warning: Potential unsafe deployment of RariFundManager

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.

You are using the `unsafeAllowCustomTypes` flag to skip storage checks for structs and enums.
Make sure you have manually checked the storage layout for incompatibilities.
Code Coverage

The code does not have any code coverage scripts set in place due to the dependence on connecting to geth nodes. We strongly recommend measuring the code coverage of the implemented test suite and making sure that the coverage is 100% or close to it. Otherwise, part of the code functionality will not be tested and could include bugs/vulnerabilities.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

aeffe8a9baf6557c5329406d91b2f95eb81228a915f9c9f9e864c2624f . /RariFundPriceConsumer.sol
7a57e4b2c915f4e4ea72c6b54c61a67f66af90ab8e597e587c1a5d1976c8 . /RariFundController.sol
727c24ba048a831b0842f72ac96f1e80a64e075c2c7108423502de0a0915 . /Migrations.sol
1ae2949ddc03b51124089959edc69884a07dacc6783ccac58844380cd465b . /RariFundToken.sol
227e6c5595c9414dd3f5b2bb08594d1c821c14134a93d5e42aa7d1d5f837f65 . /RariFundManager.sol
09aa912721f908e1f1127a7ac86eb8eab4e5b618680966d96219fbb839b9a9 . /RariFundProxy.sol
30a72222c1e10d2a3d87a34502dbab5389c89077d5bb795d468c35d1b9d6 . /EIRariGovernanceTokenDistributor.sol
d29f66b6a266626268d41631a8f046635455592c1edccc5f4a51d27613b96 . /SoloMargin.sol
5513df19c86a262a68179f045f15a8e26175b5e9d62e9f6d817946fe7e18b . /Getters.sol
7e6718152182845d5292f8283f9809e8a83088fd5af7a26f8b4d43f43 . /Operation.sol
8550faed64044783d1f5d6951d13ce0e130794917315e08222768806b96 . /Actions.sol
c1e26f5c37fcd2b3c2b2d382822834d8047d67e67df386af39797b014ec . /Account.sol
8676621d3b4ec4f51f8a8d3323914441d028bc415aa448daa2b2b970a1119 . /Types.sol
1f8397c941d7414f9183c6164c96f909a3e88b94a955630f3d52b . /CErc20.sol
41174a20d5221c44d140b61460a1b466028805998ae46a83c5d1607b . /EBoostedVault.sol
59ba65bd2a76f98f9a3df2234255f843d9e497ba3e205aa9eb687b9da . /MassetStructs.sol
bbdf576e1a4d8c9d840b4d81f322b3c7f93021e0f9756462b6 . /BasketManager.sol
4af9294f901a60829b7f4717111159a11f4e6b4052a5be12a1d7c405c9a8b0 . /IVault.sol
30b81f1577f294aa783474a3a34b461f82f9152aa0468102e2e62dd4466 . /ZeroExExchangeController.sol
a50acdb98f3f77727b81e8b733779ebebe960c6766434716ace0f847 . /MStableExchangeController.sol
72785dfc2b5d67e550eb558s2a77c1e14a0e2f56edc53f85750d96423edc80 . /MStablePoolController.sol
8c07a5ebe87f50f9e84087475170a6e8ad5566848609f4b4d78be5d8b61123 . /Account.sol
2f52789786835641a288867440abbb8f6f5d97a2421d7047a5d77d6a3b9a6 . /YVaultController.sol
b225cc50584df958935427997dad6203865591b0dcaac4aaf15c8d4e65c16 . /CompouPoolToken.sol
20c648871aeacc95924a9d644c5d5f54174a00e0c1dfb3f980fd6ba94 . /DydxPoolController.sol
3b56e750162f57979ca8c2157422122a8777900ba30b40238331533f0r19 . /RariFundPriceConsumer.sol
8b26075a5c9d698a13cb7e8f09835a4a08ac26b57289011c9f4e4b0b4a6c8c . /RariFundController.sol
77cf2abca4a8833b6e42f72ac96f1e80a64e075c2c7108423502de0a0915 . /Migrations.sol
2b35f7958e4b55153b667b4a5f7359c0e8540bbaac15b4c745739d9f98c . /RariFundManager.sol
326a42b80d19d2a2e3b6e9bf6d7d8a758737c21d3baa45856d635271e3bb2d68 . /RariFundProxy.sol
09aa92f727fe90811f0127a7c6be86be4a6ecf68e0966d96219fbb839b9a9 . /RariFundProxy.sol
30a72222c1e10d2a3d87a34502dbab5389c89077d5bb795d468c35d1b9d6 . /EIRariGovernanceTokenDistributor.sol
d29f66b6a266626268d41631a8f046635455592c1edccc5f4a51d27613b96 . /SoloMargin.sol
5513df19c86a262a68179f045f15a8e26175b5e9d62e9f6d817946fe7e18b . /Getters.sol
7e6718152182845d5292f8283f9809e8a83088fd5af7a26f8b4d43f43 . /Operation.sol
8550faed64044783d1f5d6951d13ce0e130794917315e08222768806b96 . /Actions.sol
c1e26f5c37fcd2b3c2b2d382822834d8047d67e67df386af39797b014ec . /Account.sol
8676621d3b4ec4f51f8a8d3323914441d028bc415aa448daa2b2b970a1119 . /Types.sol
1f8397c941d7414f9183c6164c96f909a3e88b94a955630f3d52b . /CErc20.sol
41174a20d5221c44d140b61460a1b466028805998ae46a83c5d1607b . /EBoostedVault.sol
59ba65bd2a76f98f9a3df2234255f843d9e497ba3e205aa9eb687b9da . /MassetStructs.sol
bbdf576e1a4d8c9d840b4d81f322b3c7f93021e0f9756462b6 . /BasketManager.sol
Tests
Changelog

- 2020-08-20 - Initial report based on commit 66e2dc5
- 2020-09-21 - Updated report based on commit 62b501
- 2020-10-23 - Updated report based on commit 66e2dc5 and added audit for 3 new repos
- 2020-12-04 - Updated report based on commits: (1) 66e2dc5 for rari-governance-contracts, (2) 737ffed for rari-yield-pool-contracts, (3) dc5ade8 for rari-stable-pool-contracts and (4) 737ffed for rari-ethereum-pool-fund
- 2021-02-04 - Updated report for new commits: (1) 4bad6202f04259727f9ddaae97ac5ae2676acfa1af1f9c1b08bebe09adcf70, (2) 2d0ea1d1b44ff54bc4d073ca5266f8000bdc55b24318135fe92fe15a7e5daed2e, (3) a9f9ad424b235ac18d9f06c4a5e461375b801098646e6d875f904151b7b6, (4) 17b5a4d2dcd6a9251313a0b6d8f974be52f1fadd1bed1c4517c4c412b355f9f, (5) 0eb63d1f8919f4a1a2e5b56cc69192e05169f6b4f051f40b8db8e37396, (6) d0489b85b03a7745ef47c5f97e89dbec3412212e0b5461258f6cb8192943a25. /tests/exchanges/0x.js

- 2021-03-04 - Updated report for new commit: (1) 737ffed for rari-stable-pool-contracts (74ddf8c), (2) 737ffed for rari-yield-pool-contracts (5bb28f4), (3) 737ffed for rari-ethereum-pool-fund (87bf8e0), (4) 737ffed for rari-governance-contracts (cd0924)
- 2021-03-04 - Updated report for new commit: (1) 737ffed for rari-stable-pool-contracts (feaa246), (2) 737ffed for rari-yield-pool-contracts (479a34e), (3) 737ffed for rari-ethereum-pool-contracts (35b2560), (4) 737ffed for rari-governance-contracts (831285f)
About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp’s team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp’s collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.