December 14th 2020 — Quantstamp Verified

RariCapital

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

Executive Summary

Type
DeFi Aggregator

Auditors
Sebastian Banescu, Senior Research Engineer
Ed Zulkoski, Senior Security Engineer
Poming Lee, Research Engineer

Timeline
2020-08-10 through 2020-12-04

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>0d7d90f (initial audit)</td>
</tr>
<tr>
<td>rari-ethereum-pool-fund</td>
<td>895c98e6 (initial audit)</td>
</tr>
<tr>
<td>rari-governance-contracts</td>
<td>d834a81 (initial audit)</td>
</tr>
<tr>
<td>rari-stable-pool-contracts</td>
<td>4c5de88 (last reaudit)</td>
</tr>
<tr>
<td>rari-yield-pool-contracts</td>
<td>737f56d (last reaudit)</td>
</tr>
<tr>
<td>rari-ethereum-pool-fund</td>
<td>902374 (last reaudit)</td>
</tr>
<tr>
<td>rari-governance-contracts</td>
<td>253cde7 (last reaudit)</td>
</tr>
</tbody>
</table>

Total Issues
33 (19 Resolved)

High Risk Issues
3 (3 Resolved)

Medium Risk Issues
6 (5 Resolved)

Low Risk Issues
5 (3 Resolved)

Informational Risk Issues
14 (7 Resolved)

Undetermined Risk Issues
5 (1 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 audit: Quantstamp has identified several issues spanning over all severity levels, in the rari-contracts code base. Some of these issues contain sub-points which indicate that the respective issues have several instances in the code. In addition to the identified issues, one of the most concerning aspects related to tests, namely that 1 of the tests consistently failed even after several tries and that we were not able to determine the code coverage of the test suite. However, we were able to identify a modest number of 61 assertions in the test files, which indicates that not all of the functionality is accurately tested. Moreover, we identified 23 TODOs, which indicate tests yet to be written. It is of utmost importance for any production-ready project to have a code coverage as close as possible to 100% and a high number of assertions in order to ensure that all the functionality of the smart contracts has been tested. Finally, several deviations from best practices and code documentation issues were found during the audit. We strongly recommend that all of these issues be addressed before deploying the code on the Ethereum mainnet.

After 1st reaudit: Quantstamp has performed a reaudit of the existing code base and an audit of the newly added features. All of the previously identified issues were either resolved (8 issues) or acknowledged (6 issues). All tests are currently passing. Additionally, 3 new issues were identified. The new issues (from QSP-15 to QSP-17) were added at the end of the list of existing issues.

After 2nd reaudit: Quantstamp has performed a reaudit of the existing code base and an audit of 3 new repositories, namely rari-yield-pool-contracts, rari-ethereum-pool-fund and rari-governance-contracts. All of the previously identified issues were either resolved (12 issues) or acknowledged (5 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 3rd reaudit: Quantstamp has performed a reaudit of all 4 repositories which were previously audited. The report has been updated accordingly. We recommend addressing all features marked as Acknowledged as soon as possible. Note that during this reaudit we only checked the fixes to the issues we had discovered in the previous commit and have not looked at newly added features.

<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>Divergent mirrored states</td>
<td>Medium</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-3</td>
<td>Gas Usage / 1x2 Loop Concerns</td>
<td>Medium</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-4</td>
<td>Unchecked Return Value</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-5</td>
<td>Missing input argument validation</td>
<td>Low</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-6</td>
<td>Privileged Roles and Ownership</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-7</td>
<td>Fallback function can receive funds from any address</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-8</td>
<td>Dangerous cast from uint256 to int256</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-9</td>
<td>Allowance Double-Spend Exploit</td>
<td>Informational</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-10</td>
<td>Unlocked Pragma</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-11</td>
<td>Experimental features should not be used on Mainnet deployments</td>
<td>Informational</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-12</td>
<td>Checks-Effects-Interactions Pattern</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-13</td>
<td>Block Timestamp Manipulation</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-14</td>
<td>Potential funds stuck in contract</td>
<td>Undetermined</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-15</td>
<td>Unfinished token upgrades</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-16</td>
<td>Misaligned comments and implementation</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-17</td>
<td>Rounding error</td>
<td>Undetermined</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-18</td>
<td>Incorrect Rari Governance Token amount</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-19</td>
<td>Uninitialized _ethNusdPriceFeed</td>
<td>High</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-20</td>
<td>Incorrect value for supported currencies</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-21</td>
<td>Amount in pools may be incorrect</td>
<td>Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-22</td>
<td>ETH/USD prices could be stale</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-23</td>
<td>Off-by-one error</td>
<td>Low</td>
<td>Mitigated</td>
</tr>
<tr>
<td>QSP-24</td>
<td>Missing input argument validation (2)</td>
<td>Low</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-25</td>
<td>Duration of RGT distribution may be different from 60 days</td>
<td>Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-26</td>
<td>Increased loss of precision due to dividing before multiplication</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-27</td>
<td>Privileged Roles and Ownership (2)</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-28</td>
<td>Unexpected pool</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-29</td>
<td>Single point of failure for price feeds</td>
<td>Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-30</td>
<td>Fallback function can receive funds from any address (2)</td>
<td>Undetermined</td>
<td>Acknowledged</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></td>
<td></td>
</tr>
</tbody>
</table>
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
- Arbitrage 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: `sithler`

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 a 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 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 development (before deployment), 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-3 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 want to 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 for 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 RariFundManager.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 supported currencies was too high.
5. The getAllBalances in RariFundController contains nested loops and a call to potentially expensive external functions (inside the inner loop).
6. The marketSell Orders Fill Or Kill 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. The checkLossRateLimit contains a loop and could reach an out-of-gas error if the _lossRateHistory was too long.
8. The 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 marketSellOrdersFillOrKill 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-4 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.accurateInterest() on line 49 in CompoundPoolController.sol, whose return value is not checked.

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

QSP-5 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 <= 10**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-6 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-7 Fallback function can receive funds from any address
Severity: Informational
Status: Fixed
File(s) affected: RariFundController.sol

Description: The fallback function is meant to only be "called by 0x 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 0x. This way the function will revert if any other address sends funds to it.

QSP-8 Dangerous cast from uint256 to int256
Severity: Informational
QSP-9 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 \( N \) amount of Alice’s tokens (\( N > 0 \)) 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 \( M \) (\( M < N \)) 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, it is recommended 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-10 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 (^)0.5.9`. 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.

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

QSP-11 Experimental features should not be used on Mainnet deployments
Severity: Informational
Status: Mitigated
File(s) affected: Several contracts
Description: Until solidity 0.6.0, 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-12 Checks-Effects-Interactions Pattern
Severity: Informational
Status: Fixed
File(s) affected: RariFundManager.sol
Description: The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implies, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done.

Recommendation: This pattern is not followed in several places, for example on line 57 in `within _withdrawFrom()`, the token transfer should happen after setting the `_netDeposits` and `_netDepositsByAccount` to match this recommended pattern.

QSP-13 Block Timestamp Manipulation
Severity: Informational
Status: Acknowledged
File(s) affected: RariFundController.sol
QSP-19 Uninitialized

Recommendation: Do not use block.number instead of block.timestamp to avoid manipulation. Or clearly document that a 900 second error is possible and acceptable to avoid 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-14 Potential funds stuck in contract

Severity: Undetermined

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 RariFundTokenUpgrade contract has been removed.

QSP-15 Unfinished token upgrades

Severity: Medium Risk

Status: Fixed

Description: In the function 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(8). Wouldn't the DAI funds be stuck in the contract until another withdrawAndExchange() transaction occurs with outputErc20Contract = DAI?

Recommendation: Add a 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-16 Misaligned comments and implementation

Severity: Low Risk

Status: Fixed

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-17 Rounding error

Severity: Undetermined

Status: Fixed

Description: 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(1e18).sub(1).div(exchangeRate).add(1);
```

Recommendation: The dev team has indicated that this is indeed the correct logic. The test _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-18 Incorrect Rari Governance Token amount

Severity: High Risk

Status: Fixed

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.
2.

File(s) affected: RariGovernanceTokenDistributor.sol in rari-governance-contracts

Description: The AggregatorV3Interface private _ethUsdPriceFeed state variable defined on L224 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.

Update: This issue was also independently found by the Rari Capital dev team and fixed before the Mainnet deployment.

QSP-20 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 (uint256 i = 0; i < _supportedCurrencies.length; i++)
// _acceptedCurrencies[acceptedCurrencies.length] = _supportedCurrencies[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.

Recommendation: Change the array index of the left-hand side member of the assignment to an index value that keeps increasing when a new value is added inside the if-statement.

QSP-21 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: `contractBalance = token.balanceOf(_rariFundControllerContract);`
2. L688-683 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` then `realContractBalance = amount` 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 L668-683 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 are 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-22 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-23 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(8) & 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 if-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-24 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. setFundBalanceIn 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 validation argument 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-25 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 LUSI 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 Ethereum we foresee on 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 15.292 seconds per block). We have added the suggested notice to README.md and CONCEPT.md.

QSP-26 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 = 18 >> outputDecimals > inputAmounts[i].div(10 ** (uint256(18).sub(outputDecimals))) : inputAmounts[i].mul(10 ** (outputDecimals.sub(18))); is performed before the multiplication in this assignment

2. In MStablePoolController.withdraw the division in the following assignment uint256 credits = amount.mul(1e18).div(exchangeRate); is performed before the division in the following if-condition if (credits.mul(exchangeRate).div(1e18) < amount)

3. In RariFundManager.depositTo the division in the following assignment uint256 amountUSD = amount.mul(rftAmountToCurrency[CurrencyCode]).div(18 - uint256(currencyDecimal[CurrencyCode])); is performed before the multiplication in the following assignment uint256 rftAmount = amountUSD.mul(rftTotalSupply).div(fundBalanceUsd); in all repos does not check the parameter of type address.

4. In RariFundPriceConsumer.getMStablePrice the following function contains a division before the last multiplication usdSupplyScaled = usdSupply.mulf(asset[m][1].vaultBalance.mul(asset[m][1].ratio ).div(1e18).mul BíassetUsdPrices[1][1]));

5. In RariGovernanceTokenDistributor.storeRgtDistributedPerRft the following contains a division before the last multiplication uint256 amountUsd = amount.mul(1e18).div(exchangeRate); is performed before the division in the following if-condition if (credits.mul(exchangeRate).div(1e18) < amount)

6. In RariGovernanceTokenDistributor.storeRgtDistributedPerRft the following contains a division before the last multiplication uint256 amountUsd = amount.mul(1e18).div(exchangeRate); is performed before the division in the following if-condition if (credits.mul(exchangeRate).div(1e18) < amount)

7. In RariGovernanceTokenDistributor.storeRgtDistributedPerRft the following contains a division before the last multiplication uint256 amountUsd = amount.mul(1e18).div(exchangeRate); is performed before the division in the following if-condition if (credits.mul(exchangeRate).div(1e18) < amount)

8. In RariGovernanceTokenDistributor.storeRgtDistributedPerRft the following contains a division before the last multiplication uint256 amountUsd = amount.mul(1e18).div(exchangeRate); is performed before the division in the following if-condition if (credits.mul(exchangeRate).div(1e18) < amount)

9. In RariGovernanceTokenDistributor.storeRgtDistributedPerRft the following contains a division before the last multiplication uint256 amountUsd = amount.mul(1e18).div(exchangeRate); is performed before the division in the following if-condition if (credits.mul(exchangeRate).div(1e18) < amount)

10. In RariGovernanceTokenDistributor.storeRgtDistributedPerRft the following contains a division before the last multiplication uint256 amountUsd = amount.mul(1e18).div(exchangeRate); is performed before the division in the following if-condition if (credits.mul(exchangeRate).div(1e18) < amount)
QSP-27 Privileged Roles and Ownership (2)

Severity: Informational

Status: Acknowledged

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

Description: 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 reverting 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 withdrawFreePoolKnowingBalanceToManager 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-28 Unexpected pool

Severity: Informational

Status: Acknowledged

File(s) affected: RariGovernanceTokenDistributor.sol

Description: In RariGovernanceTokenDistributor.sol (rari-governance-contracts), the functions setFundManager, 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 2, 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-29 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-30 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 0x, 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.
We have identified the following issues in the code documentation:

**Code Documentation Adherence to Specification**

Additionally slither has found that solidity naming conventions have not been respected:

- Variable Migrations.last_completed_migration (Migrations.sol#14) is not in mixedCase
- Parameter Migrations.upgrade(address).new_address (Migrations.sol#28) is not in mixedCase
- Constant DydxPoolController._soloMargin (lib/pools/DydxPoolController.sol#43) is not in UPPER_CASE_WITH_UNDERSCORES
- Function LibRichErrors.StandardError(string) (@0x/contracts-utils/contracts/src/LibRichErrors.sol#34-45) is not in mixedCase
- Variable RariFundController._aaveReferralCode (RariFundController.sol#342) is not in mixedCase
- Variable RariFundController._poolsWithFunds (RariFundController.sol#328) is not in mixedCase
- Function RariFundController._getPoolBalance(uint8,string) (RariFundController.sol#265-272) is not in mixedCase
- Variable RariFundManager._poolBalanceCache (RariFundManager.sol#398) is not in mixedCase
- Variable RariFundManager._cachePoolBalances (RariFundManager.sol#388) is not in mixedCase
- Constant AavePoolController._lendingPool (lib/pools/AavePoolController.sol#41) is not in UPPER_CASE_WITH_UNDERSCORES
- Constant ZeroExExchangeController._exchange (lib/exchanges/ZeroExExchangeController.sol#44) is not in UPPER_CASE_WITH_UNDERSCORES
- Function LibMathRichErrors.RoundingError(uint256,uint256,uint256) (@0x/contracts-exchange-libs/contracts/src/LibMathRichErrors.sol#23-38) is not in mixedCase
- Constant RariFundProxy._weth (RariFundProxy.sol#115) is not in UPPER_CASE_WITH_UNDERSCORES

**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:

QSP-31 Rari Governance Tokens can indeed be claimed at any time after the starting block of the distribution period.

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()) & block.number < rariGovernanceTokenDistributor.startBlock(); which appears in the following functions: transfer, transferFrom, mint, burn, burnFrom, and FundManager.burnFrom. 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.endBlock().

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 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 _depositToWithdrawFrom, and withdrawFees in RariFundManager.sol in 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.

**Adherence to Specification**

The implementation seems to adhere to the specification.

**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] L74 in AavePoolController.sol - "dYdX" 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 - "exchanges 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 bools indicating if Ethereum addresses are immune to the account balance limit." does not reflect the mapping below, which has no bools.

6. The account balance limit imposed by setDefaultAccountBalanceLimit() will not enforce the restriction on existing balances above the newly set limit, unless they try to invoke depositTo() again. That is, it will only impose this limit on future deposits.

7. The documentation should indicate external resources where users can identify the hardcoded addresses from the source code. For example, the constants on L50-51 in DydxPoolController.sol seem to correspond to here: https://docs.dydx.exchange/#sub-get-v1-markets.

8. 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.


### Adherence to Best Practices

We have identified the following deviations from best-practices:

1. Many protocol and token addresses are re-used throughout (e.g., DA). 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 RariFundController.sol could be split across multiple lines for readability.

4. [Fixed] L87-103 in RariFundController.sol, could use an enum instead of the constants 1, 2, 3 for dYiX, compound, aave.

5. addSupportedCurrency() does not check if the currencyCode or erc20Contract have already been added (although only invoked from constructor).

6. The two upgradeFundController() functions in RariFundController.sol have significantly different semantics. They probably shouldn't have the same name.

7. .getPoolBalance() in RariFundController.sol should likely be declared internal.

8. .poolsWithFunds in RariFundController.sol, as defined on L328, should be declared higher in the contract (it is used above).

9. On L204 of RariFundManager.sol, the check _authorizedFundManagerDataSource != address(0) is not needed since the next condition checks that msg.sender == _authorizedFundManagerDataSource.

10. Hard to read indentation style in getPoolBalance() and several other functions.

11. _depositFee() could use an enum to define the return types.

12. Missing return value in RariFundManager.depositFee(), because the code comment above it contains a return tag. Moreover, the function declaration does specify returns(0) in the rari-ethereum-pool-fund repository, but it does not specify this in the rari-stable-pool-contracts and rari-yield-pool-contracts.

13. All 3 occurrences are missing an explicit return statement.

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

   - TODO: Factor in prices; for now we assume the value of all supported currencies = 1
   - TODO: Support orders with ticker fees (need to include ticker fees in loss calculation)
   - TODO: Or revert("No funds available to redeem from Compound cToken.") on L67 in CompoundPoolController.sol in rari-ethereum-pool-fund.
   - TODO: Import from rari-contracts-governance repository on L19 in RariFundToken.sol

14. getFundBalance, getRawFundBalance, getInterestFeesUnclaimed should be view functions.

15. Avoid using inline constants. Use named constants instead. For example:

   - the constant value 18 is used repeatedly in multiple files.
   - the constant values 8, 1 and 2 are used to represent the pool IDs for dXyY, Compound and Aave in the constructors of RariFundController.sol and RariFundManager.sol.
   - the constant value 6448 is used on L537 in RariFundController.sol.

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 onlyRebalancer 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 poolsByCurrency. There is no need to keep this state information in both contracts.
   - constructors of RariFundController.sol and RariFundManager.sol are identical.
   - addSupportedCurrency, addPoolToCurrency, setFundRebalancer, disableFund, and enableFund functions are declared in both RariFundController.sol and RariFundManager.sol.
   - L627-629, L717-719, L898-900 in RariFundManager.sol are clones.

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

   - L170 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.
   - L177 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.
For the future. The dev team has indicated that this warning is not an issue for the initial deployment of the contracts and will only be relevant when the contracts are upgraded in the Test Suite Results

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.

The dev team has indicated that this warning is not an issue for the initial deployment of the contracts and will only be relevant when the contracts are upgraded in the future.

For the `rari-stable-pool-contracts` and `rari-yield-pool-contracts` 17 of 19 tests are currently passing and 2 tests are failing.

Additionally, the following warning is given:

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.

The following tests are failing:

- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the proxy and implementation of FundManager to new code` (2,896ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
- `should withdraw and exchange all input currencies without using too much gas` (23,869ms)
- `should upgrade the fund controller owner` (481ms)
- `should upgrade the fund manager owner` (1,235ms)
Warning: Could not decode event!
Warning: Could not decode event!
Warning: Could not decode event!

RariFundController+RariFundManager: _safeCall(address) 
- (Indexed) RariFundController+RariFundManagerDeposit(address: uint256, bounty: uint256)
- (Indexed) RariFundController+RariFundManagerWithdraw(address: uint256, bounty: uint256)
- (Indexed) RariFundController+RariFundManagerExchange(address: uint256, bounty: uint256)
  
  amount: 100000000000000000 (type: uint256) 
}


Warning: Potentially unsafe deployment of DummyRariFundManager

Gas usage of RariFundController+RariFundManager 583820

should update the FundManager with funds in all pools in all currencies without using too much gas (38585ms)

17 passing (5s)

2 failing

1) Contract: RariFundController, RariFundManager
   - should exchange tokens to and from collateral ASFLR via RariFundController.mintCollateral and redeemCollateral
     at Object._sendTransaction (node_modules/web3/node_modules/web3-providers-http/src/index.js:262:23)
     at Object.sendTransaction (node_modules/web3-providers-http/src/index.js:99:11)

2) Contract: RariFundController, RariFundManager
   - should deposit to the fund, approve and deposit to pools, accrue interest, and withdraw from the fund:
     at Context.it (test/5_fund_user.js:72:77)
   at IncomingMessage.<anonymous> (node_modules/truffle/build/webpack:/node_modules/xhr2-cookies/dist/xml-http-request.js:289:47)
     at XMLHttpRequest._onHttpResponseEnd (node_modules/truffle/build/webpack:/node_modules/xhr2-cookies/dist/xml-http-request.js:318:1)
     at XMLHttpRequest._setReadyState (node_modules/truffle/build/webpack:/node_modules/xhr2-cookies/dist/xml-http-request.js:208:1)
     at XMLHttpRequest.request.onreadystatechange (node_modules/truffle/build/webpack:/node_modules/web3/node_modules/web3-providers-http/src/index.js:111:1)

Error: Returned error: VM Exception while processing transaction: revert bAsset must exist

at endReadableNT (_stream_readable.js:1327:12)
at XMLHttpRequest._onHttpResponseEnd (node_modules/truffle/build/webpack:/node_modules/xhr2-cookies/dist/xml-http-request.js:318:1)
at XMLHttpRequest._setReadyState (node_modules/truffle/build/webpack:/node_modules/xhr2-cookies/dist/xml-http-request.js:208:1)
at XMLHttpRequest.request.onreadystatechange (node_modules/truffle/build/webpack:/node_modules/web3/node_modules/web3-providers-http/src/index.js:111:1)

at endReadableNT (_stream_readable.js:1327:12)

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 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

38567e50b165836997908c0821574b2122a8277988bc084b923b881553f0912 ./rari-capital-launch/Rari Stable-pool-contracts/RariFundConsumer.sol
cf3b8697fd0c1b30a62b74f5b39f18f8b4af8a8757c2718e2359d2e0b0c051 ./rari-capital-launch/Rari Stable-pool-contracts/RariFundController.sol
77cfabca4a983b1b624f29f2ac96f1b864a0a79b787d5b5948650c5f1cb96 ./rari-capital-launch/Rari Stable-pool-contracts/Migrations.sol
2d05357b694956551be9be17a21a61e41e85d8b631447b49e69 ./rari-capital-launch/Rari Stables-contracts/RariFundToken.sol
564917d5c785b0bdc75d90f72572b46c5d61212034d3fb47ec53adfd7f023a6 ./rari-capital-launch/Rari Stable-pool-contracts/RariFundManager.sol
09da92772f47613c01127aceb6e6b6b4e6c6f86b966d966219fbb8399a9 ./rari-capital-launch/RariStablePoll-contracts/RariFundProxy.sol
38a2722213e102a3682a34502eb6f1358e09a8bb77d7b5948650c5f1cb96 ./rari-capital-launch/Rari Stable-pool-contracts/interfaces/IERariGovernanceTokenDistributor.sol
d2956b6a26662a7eb6d40b3e455f1e531c0d55f1e56d7613b6 ./rari-capital-launch/RariStablePoll-contracts/external/dydx/dydstoplain/sol/IgnoreMargin.sol
5513d1f9da66a2aff702e405f13a6261715b9ad6a651f1a7946eae1f1b ./rari-capital-launch/RariStablePoll-contracts/external/dydx/dydxgetters/sol/IgnoreMargin.sol
76a7185511b82485d5299238b81306e6a3b889a4a72a3f66b4a3fd343 ./rari-capital-launch/RariStablePoll-contracts/external/dydx/Accounts.sol
855f02a6c4d4e67d61f65c1153ebed0313e97487f17519a8222776698b9e ./rari-capital-launch/RariStablePoll-contracts/external/dydx/Accounts.sol
ce26053c4cf82c6db26d8282b3d94d8b403b75e7ead3d3b8e63d799614ece ./rari-capital-launch/RariStablePoll-contracts/external/dydx/Account.sol
8676ab1d53e4c45f4dade63023f91444b1b29bc4c1aa448dada0b7b90119 ./rari-capital-launch/RariStablePoll-contracts/external/0xignore/IgnorePools.sol
1e57f372cd700ac14d14095a1c469f999e9e279f9495c4d36c85b2c25b ./rari-capital-launch/RariStablePoll-contracts/external/compound/CompErrors.sol
411734d6e356c351b5f5d9a9e219f3c430d731b5197d86bdf8b49 ./rari-capital-launch/RariStablePoll-contracts/external/aave/LendingPool.sol
145199ac5f73ee74663a67e3fe3e6e1cbedf6899d53e5e62571246a1e53d727 ./rari-capital-launch/RariStablePoll-contracts/external/aave/Account.sol
ebc570ac9b6568b388bad0af25c47a4af1d94a3ff9f997a7d653d638 ./rari-capital-launch/RariStablePoll-contracts/external/dydx/Account.sol
34d6f23b99f1c16c2e28440160e1132714e3772221cc78c20c8d2e2f6c5b13 ./rari-capital-launch/RariStablePool-contracts/external/dydx/Account.sol
59fa865d8a4fe628d9a4d2f4335f8eed4e987baa3e034a09dd8d6e67bat ./rari-capital-launch/RariStablePool-contracts/external/dydx/Account.sol
bbd5f61ed6a4db19f9d4b4db81f8d2f2d5a5c20f312ce06a97c805973c9a1f0b4 ./rari-capital-launch/RariStablePool-contracts/external/0xignore/IgnorePools.sol
308fb1f57779b9a39473a446b74b6c45f8f4922ca4a54b8ded2ba6d54665 ./rari-capital-launch/RariStablePool-contracts/external/0xignore/AccountExchanger.sol
a5aaad6cb3f1e777757258eb8a807f777fe60b96e260c4676647171eac0f084 ./rari-capital-launch/RariStablePool-contracts/external/0xignore/AccountExchanger.sol
53ab848243f626c463f947a1e31c437c72193a2a416c9c5b8a8b6f3e243f ./rari-capital-launch/RariStablePool-contracts/external/0xignore/Migration.sol
8c0b75e8b70f9d760847874615170a6e8a5666a468f048c87d8b5d8c81123 ./rari-capital-launch/RariStablePool-contracts/external/0xignore/Migration.sol
be225c850b0d598969534729e18a10368536d1b77dbc44d5b2c6b6f055c1 . ./rari-capital-launch/RariStablePool-contracts/external/0xignore/Migration.sol
29b7e48f71a9ec065924aad9e5c1d5f417aaf006cb1d239986fd8ba94 ./rari-capital-launch/RariStablePool-contracts/external/0xignore/Migration.sol
e128b28f065e232a2f5bd8b1c993a242c53555e47099246d889795d5f67a ./rari-capital-launch/RariStablePool-contracts/external/0xignore/Migration.sol
213e585dfc10b25ab798a36552d2b0c6a525ac9a419db6ba8266c9c3d514c ./rari-capital-launch/RariStablePool-contracts/external/0xignore/Migration.sol

09da92772f47613c01127aceb6e6b6b4e6c6f86b966d966219fbb8399a9 ./rari-capital-launch/RariStablePoll-contracts/RariFundProxy.sol
contracts/migrations/2_deploy_contracts.js
e382b404d2677b0d0b2b75f7d7ba9b9e1891da58340ab724d9fc7c07c0e /rari-capital-launch/rari-ethereum-pool-fund/truffle-config.js
b95164b679739b617d4b9abaa12e4d065f9d8855b9f218a124de86c43c8c4 /rari-capital-launch/rari-ethereum-pool-fund/scripts/ganache.js
24322748976a554a3a32ac321af656cb1265a23e17576df5c45f51475259e0ba3 /rari-capital-launch/rari-ethereum-pool-fund/test/block-gas-limit.js
1c849ff4057c0b2c56c4b04c459646cb5541e5056f0ed108f27b9c7aaf8b95f /rari-capital-launch/rari-ethereum-pool-fund/test/fund-fees.js
2cb3b11c14f05f65f1f6f4f2c9f94f2f4e6a35a2f4b9f1f988c6e6b3d1163 /rari-capital-launch/rari-ethereum-pool-fund/test/fund-user.js
f9d35872422877b9f6b19f8e998f1f6e48ac5d5b2a7aa /rari-capital-launch/rari-ethereum-pool-fund/test/fund-owner.js
00ca6194f5622766a4f455d97c51f4e80b7a2b6f8f0ce7c9a2b88199f375e2 /rari-capital-launch/rari-ethereum-pool-fund/test/rebalancer.js
ba0728859322f2b7e18b88d0126e6c27633064d52dc485e4dda75b124019768e80 /rari-capital-launch/rari-ethereum-pool-fund/test/keeperdao-integration.js
dd499b5ebd9a3a774f47c5f97ac8d9bec341212b2e4b5461258d6c8b182943a25 /rari-capital-launch/rari-ethereum-pool-fund/test/exchanges/0x.js
5c8591f7f6b1d6b8189ed1f0236978ee57bb0523d5996dd08941e7749fff7 /rari-capital-launch/rari-ethereum-pool-fund/migrations/1_initial_migration.js
69dda293397f0e68533a297f6c9a6a80b545478388d938386eddaec417dc1 /rari-capital-launch/rari-ethereum-pool-fund/migrations/2_deploy_contracts.js

Changelog

- **2020-08-20** - Initial report based on commit 66e2dcf5
- **2020-09-21** - Updated report based on commit 62b5011
- **2020-10-23** - Updated report based on commit ae98c4f1 and added audit for 3 new repos
- **2020-12-04** - Updated report based on commits: (1) for rari-governance-contracts, (2) for rari-yield-pool-contracts, (3) for rari-stable-pool-contracts and (4) for rari-ethereum-pool-fund
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 $3B 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.