Quantstamp Security Assessment Certificate

December 14th 2020 — Quantstamp Verified

RariCapital V2

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

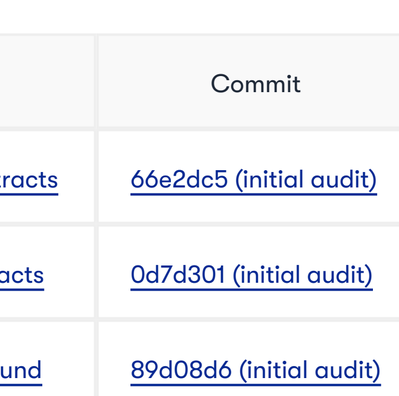
Executive Summary

Type	DeFi Aggregator
Auditors	Ed Zulkoski, Senior Security Engineer Sebastian Banescu, Senior Research Engineer Poming Lee, Research Engineer Jose Ignacio Orlicki, Senior Engineer Fayçal Lalidji, Security Auditor
Timeline	2020-08-10 through 2021-03-04
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
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

Repository	Commit
rari-stable-pool-contracts	66e2dc5 (initial audit)
rari-yield-pool-contracts	0d7d301 (initial audit)
rari-ethereum-pool-fund	89d08d6 (initial audit)
rari-governance-contracts	d83b481 (initial audit)
rari-stable-pool-contracts	feaa246 (final audit)
rari-yield-pool-contracts	479a346 (final audit)
rari-ethereum-pool-fund	75fb256 (final audit)
rari-governance-contracts	83238f7 (final audit)

Total Issues	**41** (23 Resolved)
High Risk Issues	**3** (3 Resolved)
Medium Risk Issues	**6** (5 Resolved)
Low Risk Issues	**9** (4 Resolved)
Informational Risk Issues	**16** (8 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 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 has several instances in the code. In addition to the identified issues one of the most concerning aspects are 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 have 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.

After 4th reaudit: Several new issues were found as listed below (see QSP-34 -- QSP-41, 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` (`feaa246`), `rari-yield-pool-contracts` (`479a346`), `rari-ethereum-pool-contracts` (`75fb256`), `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.

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

ID	Description	Severity	Status
QSP-40	No example of token distribution implementation is presented	? Undetermined	Fixed
QSP-41	Controller unable to pause specific stablecoins	⌄ Low	Acknowledged

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 L234 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 issues was also independently found by the Rari Capital dev team and fixed before the Mainnet deployement.

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 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-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 `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 `upgradeFundContoller` 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 `inputCurrenyCodes` was too high.
5. `getAllBalances` in `RariFundController` contains nested loops and a call to potentially expensive external functions inside the inner loop.
6. `marketSell0xOrdersFillOrKill` contains a loop with calls to 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 `marketSell0xOrdersFillOrKill` 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:

```
for (uint256 i = 0; i < _supportedCurrencies.length; i++) if (_acceptedCurrencies[_supportedCurrencies[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-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. L668-683 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. L709 checks if `realContractBalance < amount ? 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 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 `<= 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-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`
2. `RariFundPriceConsumer.getDaiUsdPrice` in `rari-stable-pool-contracts` and `rari-yield-pool-contracts`
3. `RariFundPriceConsumer.getEthUsdPrice` in `rari-stable-pool-contracts` and `rari-yield-pool-contracts`

4. `RariFundPriceConsumer.getPriceInEth` in `rari-stable-pool-contracts` and `rari-yield-pool-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 `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-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*

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-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 `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>0`) 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 `0.6.0`, 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-21 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 implied, 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 L752 within `_withdrawFrom()`, the token transfer should happen after setting the `_netDeposits` and `_netDepositsByAccount` to match this recommended pattern.

QSP-22 Block Timestamp Manipulation

Severity: *Informational*

Status: Acknowledged

File(s) affected: `RariFundController.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.
The `checkLossRateLimit` makes a decision based on the block timestamp on L537. However, the interval there seems to be 24 hours, which is a far larger than 900 seconds. Therefore, the attacker can only benefit by stopping the iteration of the for loop earlier than expected and use another value for `lossRateLastDay` than intended by the developer.

Recommendation: Use `block.number` 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 `13.292` 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-goverance-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 `realOutputAmount = outputAmount.sub(outputAmount.mul(MStableExchangeController.getSwapFee()).div(1e18));`
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(pricesInUsd[_currencyIndexes[currencyCode]]).div(10 ** _currencyDecimals[currencyCode]);` is performed before the multiplication in the following assignment `rftAmount = amountUsd.mul(rftTotalSupply).div(fundBalanceUsd);`
4. In `RariFundPriceConsumer.getMUsdUsdPrice` the following assignment contains a division before the last multiplication `usdSupplyScaled = usdSupplyScaled.add(bAssets[i].vaultBalance.mul(bAssets[i].ratio).div(1e8).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).mul(1e18).div(totalSupply))`
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(_fundBalancesCache[i_scope_0]).div(fundBalanceSum).mul(1e18).div(totalSupply))`
7. In `RariGovernanceTokenDistributor.getRgtDistributedPerRft` the following assignment contains a division before the last multiplication `_rgtPerRftAtLastSpeedUpdate[uint8(pool)].add(rgtToDistribute.mul(ethFundBalanceUsd).div(fundBalanceSum).mul(1e18).div(totalSupply))`
8. In `RariGovernanceTokenDistributor.getRgtDistributedPerRft` the following assignment contains a division before the last multiplication `_rgtPerRftAtLastSpeedUpdate[uint8(pool)].add(rgtToDistribute.mul(_fundBalancesCache[uint8(pool)]).div(fundBalanceSum).mul(1e18).div(totalSupply))`

9. In `RariGovernanceTokenDistributor.getRgtDistributedPerRft` the following assignment contains a division before the last multiplication `rgtPerRftByPool[i_scope_0] = _rgtPerRftAtLastSpeedUpdate[i_scope_0].add(rgtToDistribute.mul(ethFundBalanceUsd).div(fundBalanceSum).mul(1e18).div(totalSupply))`
10. In `RariGovernanceTokenDistributor.getRgtDistributedPerRft` the following assignment contains a division before the last multiplication `rgtPerRftByPool[i_scope_0] = _rgtPerRftAtLastSpeedUpdate[i_scope_0].add(rgtToDistribute.mul(_fundBalancesCache[i_scope_0]).div(fundBalanceSum).mul(1e18).div(totalSupply))`

Recommendation: Move the division after the multiplication to reduce the loss of precision.

Update from dev team: We will refactor our code so that multiplication is always be performed before division.

QSP-25 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: 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 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 `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@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-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 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, 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(1e18).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` staement.

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 `_depositTo`, `_withdrawFrom`, and `withdrawFees` in `RariFundManager.sol@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-staple-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.marketSell0xOrdersFillOrKill` should ensure that `inputErc20Contract` is non-zero.
3. `RariFundController.setFundManager` should ensure that `newContract` is non-zero.
4. `RariFundController.setFundRebalancer` 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

File(s) affected: `rari-stable-pool-contracts/.../DydxPoolController.sol`, `rari-stable-pool-contracts/.../AavePoolController.sol`, `rari-stable-pool-contracts/.../CompoundPoolController.sol`

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

to `rgtEthUniswapV2Pair` after it has already been set.
Update from the Rari Capital team: We have made this clear in the "Security" section of `CONCEPT.md` in all repositories. We have also added a check in `RariGovernanceTokenUniswapDistributor.setRgtEthUniswapV2Pair` to ensure that if users have staked LP tokens already, the LP token contract cannot be changed.

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-ethereum-pool-contracts/.../IRariGovernanceTokenDistributor.sol`

Description: `distributionEndBlock` is a critical parameter used on the interface for governance distribution `IRariGovernanceTokenDistributor`. 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 minting 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-stable-pool-contracts/.../DydxPoolController.sol`, `rari-stable-pool-contracts/.../AavePoolController.sol`, `rari-stable-pool-contracts/.../CompoundPoolController.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:

```
Constant RariFundProxy._weth (RariFundProxy.sol#115) is not in UPPER_CASE_WITH_UNDERSCORES
Function LibMathRichErrors.DivisionByZeroError() (@0x/contracts-exchange-libs/contracts/src/LibMathRichErrors.sol#15-21) is not in mixedCase
Function LibMathRichErrors.RoundingError(uint256,uint256,uint256) (@0x/contracts-exchange-libs/contracts/src/LibMathRichErrors.sol#23-38) is not in mixedCase
Function LibSafeMathRichErrors.Uint256BinOpError(LibSafeMathRichErrors.BinOpErrorCodes,uint256,uint256) (@0x/contracts-utils/contracts/src/LibSafeMathRichErrors.sol#28-43) is not in mixedCase
Function LibSafeMathRichErrors.Uint256DowncastError(LibSafeMathRichErrors.DowncastErrorCodes,uint256) (@0x/contracts-utils/contracts/src/LibSafeMathRichErrors.sol#45-58) is not in mixedCase
Constant ZeroExExchangeController._exchange (lib/exchanges/ZeroExExchangeController.sol#44) is not in UPPER_CASE_WITH_UNDERSCORES
Constant AavePoolController._lendingPool (lib/pools/AavePoolController.sol#41) is not in UPPER_CASE_WITH_UNDERSCORES
Variable RariFundManager._cachePoolBalances (RariFundManager.sol#388) is not in mixedCase
Variable RariFundManager._cacheDydxBalances (RariFundManager.sol#393) is not in mixedCase
Variable RariFundManager._poolBalanceCache (RariFundManager.sol#398) is not in mixedCase
Function RariFundController._getPoolBalance(uint8,string) (RariFundController.sol#265-272) is not in mixedCase
Variable RariFundController._poolsWithFunds (RariFundController.sol#328) is not in mixedCase
Variable RariFundController._aaveReferralCode (RariFundController.sol#342) is not in mixedCase
Function LibRichErrors.StandardError(string) (@0x/contracts-utils/contracts/src/LibRichErrors.sol#34-45) is not in mixedCase
Constant DydxPoolController._soloMargin (lib/pools/DydxPoolController.sol#43) is not in UPPER_CASE_WITH_UNDERSCORES
Parameter Migrations.upgrade(address).new_address (Migrations.sol#28) is not in mixedCase
Variable Migrations.last_completed_migration (Migrations.sol#14) is not in mixedCase
Function LibBytesRichErrors.InvalidByteOperationError(LibBytesRichErrors.InvalidByteOperationErrorCodes,uint256,uint256) (@0x/contracts-utils/contracts/src/LibBytesRichErrors.sol#40-55) is not in mixedCase
```

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` - "@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.
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/#solo-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.
9. Typo `EETH` on `L378` in `RariFundManager.sol@rari-ethereum-pool-fund`.

As of the 4th reaudit:

1. In `AlphaPoolController.sol` on L44, the comment "Assumes that you have already approved >= the amount to the ibETH token contract." is not relevant here, as it is an ETH transfer to the `ibEth` 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".

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., 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 `RariFundProxy.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 dYdX, 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. `_depositFees()` could use an enum to define the return types.
12. Missing return value in `RariFundManager.depositFees()`, because the code comment above it contains a `@return` tag. Moreover, the function declaration does specify `returns(bool)` in the `rari-ethereum-pool-fund` repository, but it does not specify this in the `rari-stable-pool-contracts` and the `rari-yield-pool-contracts`.
    All 3 occurrences are missing an explicit `return` statement.
13. TODOs should be removed before publishing the code. There are 7 TODOs present in the code comments. Some of them are concerning:
    . **[Fixed]** TODO: Factor in prices; for now we assume the value of all supported currencies = $1
    . TODO: Support orders with taker fees (need to include taker fees in loss calculation)
    . TODO: Or revert("No funds available to redeem from Compound cToken.") on L67 in `CompoundPoolController.sol@rari-ethereum-pool-`
    . 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 `0`, `1` and `2` are used to represent the pool IDs for dXdY, Compound and Aave in the constructors of `RariFundController.sol` and `RariFundManager.sol`

. the constant value `86400` is used on L537 on `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:

    . L176 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.

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. L79-82, 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 address `0xe2f2a5C287993345a840Db3B0845fbC70f5935a5`. 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 L218 clones the code of the `refreshDistributionSpeeds` function defined on L207. Instead it could just call that function with a value for `newBalance` equal to `rariFundManagers[uint8(pool)].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 `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. L45 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.

```
Contract: RariFundController, RariFundManager
     ✓ should exchange tokens (35775ms)

  Contract: RariFundProxy
Gas usage of RariFundProxy.withdrawAndExchange: 3562930
     ✓ should withdraw and exchange all input currencies without using too much gas (19205ms)

  Contract: RariFundController, RariFundManager
     ✓ should upgrade the fund manager owner (1758ms)
     ✓ should upgrade the fund controller owner (301ms)
     ✓ should disable and re-enable the fund (2543ms)
     ✓ should upgrade the fund rebalancer (592ms)

  Contract: RariFundManager

Warning: Potentially 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.

     ✓ should upgrade the FundManager implementation to a copy of its code (6503ms)

  Contract: RariFundManager
```

```
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.

     ✓ should upgrade the proxy and implementation of FundManager to new code (5888ms)

  Contract: RariFundController
     ✓ should upgrade the FundController to a copy of its code (12324ms)

  Contract: RariFundController
     ✓ should upgrade the FundController to new code (5615ms)

  Contract: RariFundToken
     ✓ should upgrade the FundToken to a copy of its code (10092ms)

  Contract: RariFundManager
     ✓ should set accepted currencies (3414ms)

  Contract: RariFundController, RariFundManager
     ✓ should deposit to the fund, approve deposits to pools via RariFundController.approveToPool, and deposit to pools via RariFundController.depositToPool (58011ms)
     ✓ should withdraw half from all pools via RariFundController.withdrawFromPool (20469ms)
     ✓ should withdraw everything from all pools via RariFundController.withdrawAllFromPool (5104ms)
     ✓ should claim mStable MTA rewards (675ms)

  Contract: RariFundController, RariFundManager
     ✓ should exchange tokens to and from mStable mUSD via RariFundController.mintMUsd and redeemMUsd (11837ms)

  Contract: RariFundManager, RariFundController
    1) should deposit to the fund, approve and deposit to pools, accrue interest, and withdraw from the fund

    Events emitted during test:
    ---------------------------

    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.PoolAllocation(
      action: <indexed> RariFundController.PoolAllocationAction.Deposit (type: enum RariFundController.PoolAllocationAction),
      pool: <indexed> RariFundController.LiquidityPool.dYdX (type: enum RariFundController.LiquidityPool),
      currencyCode: <indexed> Cannot decode indexed parameter of reference type string
    (raw value 0xa5e92f3efb6826155f1f728e162af9d7cda33a574a1153b58f03ea01cc37e568) (type: string),
      amount: 100000000000000000 (type: uint256)
    )


    ---------------------------

  Contract: RariFundManager
     ✓ should deposit to pools, set the interest fee rate, wait for interest, set the master beneficiary of interest fees, and deposit fees (8213ms)

  Contract: RariFundController
Gas usage of RariFundController.upgradeFundController: 3940600
     ✓ should upgrade the FundController with funds in all pools in all currencies without using too much gas (31285ms)


  19 passing (4m)
  1 failing

  1) Contract: RariFundManager, RariFundController
       should deposit to the fund, approve and deposit to pools, accrue interest, and withdraw from the fund:
     Error: Returned error: VM Exception while processing transaction: revert
      at Context.<anonymous> (test/5_fund_user.js:72:77)
      at runMicrotasks (<anonymous>)
      at processTicksAndRejections (internal/process/task_queues.js:93:5)

 Contract: RariFundController, RariFundManager
     ✓ should exchange tokens (35247ms)

  Contract: RariFundProxy
Gas usage of RariFundProxy.withdrawAndExchange: 3787158
     ✓ should withdraw and exchange all input currencies without using too much gas (20273ms)

  Contract: RariFundController, RariFundManager
     ✓ should upgrade the fund manager owner (396ms)
     ✓ should upgrade the fund controller owner (292ms)
     ✓ should disable and re-enable the fund (5189ms)
     ✓ should upgrade the fund rebalancer (646ms)

  Contract: RariFundManager

Warning: Potentially 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.

     ✓ should upgrade the FundManager implementation to a copy of its code (10557ms)

  Contract: RariFundManager

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.

     ✓ should upgrade the proxy and implementation of FundManager to new code (3048ms)

  Contract: RariFundController
     ✓ should upgrade the FundController to a copy of its code (12387ms)

  Contract: RariFundController
     ✓ should upgrade the FundController to new code (8173ms)

  Contract: RariFundToken
     ✓ should upgrade the FundToken to a copy of its code (7739ms)

  Contract: RariFundManager
     ✓ should set accepted currencies (8913ms)

  Contract: RariFundController, RariFundManager
     ✓ should deposit to the fund, approve deposits to pools via RariFundController.approveToPool, and deposit to pools via RariFundController.depositToPool (56340ms)
     ✓ should withdraw half from all pools via RariFundController.withdrawFromPool (32715ms)
     ✓ should withdraw everything from all pools via RariFundController.withdrawAllFromPool (16832ms)
     ✓ should claim mStable MTA rewards (596ms)

  Contract: RariFundController, RariFundManager
     ✓ should exchange tokens to and from mStable mUSD via RariFundController.mintMUsd and redeemMUsd (12432ms)

  Contract: RariFundManager, RariFundController
    1) should deposit to the fund, approve and deposit to pools, accrue interest, and withdraw from the fund

    Events emitted during test:
    ---------------------------

    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.PoolAllocation(
      action: <indexed> RariFundController.PoolAllocationAction.Deposit (type: enum RariFundController.PoolAllocationAction),
      pool: <indexed> RariFundController.LiquidityPool.dYdX (type: enum RariFundController.LiquidityPool),
      currencyCode: <indexed> Cannot decode indexed parameter of reference type string
    (raw value 0xa5e92f3efb6826155f1f728e162af9d7cda33a574a1153b58f03ea01cc37e568) (type: string),
      amount: 100000000000000000 (type: uint256)
    )


    ---------------------------

  Contract: RariFundManager
     ✓ should deposit to pools, set the interest fee rate, wait for interest, set the master beneficiary of interest fees, and deposit fees (10772ms)

  Contract: RariFundController
Gas usage of RariFundController.upgradeFundController: 4079092
     ✓ should upgrade the FundController with funds in all pools in all currencies without using too much gas (22809ms)


  19 passing (5m)
  1 failing

  1) Contract: RariFundManager, RariFundController
       should deposit to the fund, approve and deposit to pools, accrue interest, and withdraw from the fund:
     Error: Returned error: VM Exception while processing transaction: revert
      at Context.<anonymous> (test/5_fund_user.js:77:77)
      at runMicrotasks (<anonymous>)
      at processTicksAndRejections (internal/process/task_queues.js:93:5)

  Contract: RariFundManager
     ✓ should deposit to pools, set the interest fee rate, wait for interest, set the master beneficiary of interest fees, deposit fees, wait for interest again, and withdraw fees (10572ms)

  Contract: RariFundController, RariFundManager
     ✓ should upgrade the fund manager owner (224ms)
     ✓ should upgrade the fund controller owner (297ms)
     ✓ should disable and re-enable the fund (1479ms)
     ✓ should put upgrade the fund rebalancer (388ms)

  Contract: RariFundManager

Warning: Potentially 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.

     ✓ should upgrade the FundManager implementation to a copy of its code (6849ms)

  Contract: RariFundManager

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.

     ✓ should upgrade the proxy and implementation of FundManager to new code (1430ms)

  Contract: RariFundController
     ✓ should put upgrade the FundController to a copy of its code by disabling the old FundController and the FundManager, withdrawing all tokens from all pools, and transferring them to the new FundController (2813ms)

  Contract: RariFundController
     ✓ should put upgrade the FundController to new code by disabling the old FundController and the FundManager, withdrawing all ETH from all pools, and transferring them to the new FundController (10046ms)

  Contract: RariFundManager, RariFundController
     ✓ should make a deposit, deposit to pools, accrue interest, and make a withdrawal (3636ms)

  Contract: RariFundManager, RariFundController
     ✓ should make a deposit to keeperdao, then withdraw all (1717ms)


  11 passing (43s)

Contract: RariGovernanceTokenDistributor
   ✓ should have distributed the correct amount of tokens at each checkpoint (345ms)
  1) should distribute tokens evenly across pools
  > No events were emitted

Contract: RariGovernanceTokenVesting
  2) should vest private token distributions
  > No events were emitted

Contract: RariGovernanceTokenUniswapDistributor
   ✓ should have distributed the correct amount of tokens at each checkpoint (211ms)
  3) should distribute tokens evenly across pools
  > No events were emitted

Contract: RariGovernanceTokenDistributorV2
  4) should have distributed the correct amount of tokens at each checkpoint
  > No events were emitted
  5) should distribute tokens evenly across pools
  > No events were emitted

Contract: RariGovernanceTokenVestingV2
  6) should vest private token distributions
  > No events were emitted


2 passing (3s)
6 failing

1) Contract: RariGovernanceTokenDistributor
     should distribute tokens evenly across pools:
   Error: invalid address (argument="address", value=undefined, code=INVALID_ARGUMENT, version=address/5.0.5) (argument="account", value=undefined, code=INVALID_ARGUMENT, version=abi/5.0.0-beta.153)
    at Context.<anonymous> (test/1_governance_token_distribution.js:86:80)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)

2) Contract: RariGovernanceTokenVesting
     should vest private token distributions:
   Error: invalid address (argument="address", value=undefined, code=INVALID_ARGUMENT, version=address/5.0.5) (argument="holder", value=undefined, code=INVALID_ARGUMENT, version=abi/5.0.0-beta.153)
    at Context.<anonymous> (test/2_governance_token_vesting.js:31:42)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)

3) Contract: RariGovernanceTokenUniswapDistributor
     should distribute tokens evenly across pools:
   Error: The send transactions "from" field must be defined!
    at Context.<anonymous> (test/3_governance_token_uniswap_distribution.js:54:27)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)

4) Contract: RariGovernanceTokenDistributorV2
     should have distributed the correct amount of tokens at each checkpoint:
   AssertionError: Unspecified AssertionError
    at Context.<anonymous> (test/4_governance_token_distribution_v2.js:40:7)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)

5) Contract: RariGovernanceTokenDistributorV2
     should distribute tokens evenly across pools:
   Error: invalid address (argument="address", value=undefined, code=INVALID_ARGUMENT, version=address/5.0.5) (argument="account", value=undefined, code=INVALID_ARGUMENT, version=abi/5.0.0-beta.153)
    at Context.<anonymous> (test/4_governance_token_distribution_v2.js:57:80)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)

6) Contract: RariGovernanceTokenVestingV2
     should vest private token distributions:
   Error: invalid address (argument="address", value=undefined, code=INVALID_ARGUMENT, version=address/5.0.5) (argument="holder", value=undefined, code=INVALID_ARGUMENT, version=abi/5.0.0-beta.153)
    at Context.<anonymous> (test/5_governance_token_vesting_v2.js:31:42)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
```

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

```
aeffef8a0aff6557c5329406d91b2f59e6b12284a98a5fc9f9c8ceb864c2624f ./RariFundPriceConsumer.sol
7a57f4b2c913af4e1ca72c64b54cf61a67ffda9bab5897ee578cf1a51da17fc0 ./RariFundController.sol
77cf2a0ca04a8831bb642f9f2ac96f10f8a40aa6e075c2c7110423502de00915 ./Migrations.sol
1ae2b19dddc0b33112d408939cd7e08d4a07dacdd785ccdae58044307cdd4658 ./RariFundToken.sol
22a7bc655913c941da4d33b20b85d91c821c114134a9a3d5e4fa0a7d158f3765 ./RariFundManager.sol
090ae92722f79f89091ff01127a8c60e6bde4e6c6f8680966d96219fbb839ba9 ./RariFundProxy.sol
30a7222c13e1028a3d87a345020eb1358c09a80b778d7bb5948650c35d1c9bd6 ./IRariGovernanceTokenDistributor.sol
d29f66b465a266862cf6bb410631af0046b34555892e1ccdc5fea1c6d17613b6 ./SoloMargin.sol
5513d1f9cdaf628aff707640f5130a626137b5e9d962e9ffa68c17946efe7105 ./Getters.sol
7e671035218f2845db3298f288f390509b81a30088d7abfa720a3f6bb4d3df43 ./Operation.sol
8550fa9ed4d04778d31fd659c11d3ceed0130794817315e0f8022776880bb690 ./Actions.sol
ce2fe53c7fc82dbcb260d288d82823d94d8048d75e7edd8306fa3d7976b14ece ./Account.sol
867682d15be4c4f45fbfa8ed83328914441bf208c41c5aa448dda028b790f119 ./Types.sol
1f837c92dc7fca41d14103938c1649f09909a3f809ab4953c6136c85abc2d5bb ./CErc20.sol
4117f41ad0e5feecd235e31135b53d95a7e21f93ccdae7315b1917d860df8b49 ./LendingPool.sol
145190ac5f73ee74663ade71ee3f3eeb2cbef5847953d3e2632f6ae0a54d6727 ./AToken.sol
bc5cc7a09bf0b9963838380dfeaf5c25ca5afd1d9a9e19fff9f9c7a2fd363de8 ./ISavingsContract.sol
34a6f23b9561c13c2d484041e10ef132174e37722d1cc78c20cc8d2fdbfc5b13 ./IMasset.sol
292bbb87dab48a3ef859d16f14c668a1a1ba6405092880598a4e63a5c3c16bb7 ./IBoostedSavingsVault.sol
59ba0865db8afe7b6f89fa3dbcf24335fcea8d4907bafe2054aead9be687bda1 ./MassetStructs.sol
bbdf57e661ad48bc13b8d64b4d881fd322bd5e27cf32acf0fef097c3d9aa1f04 ./IBasketManager.sol
4af9d295f60116a7082f7417311139a1fa166eb04e502ae5b2ad1c74005cba0e ./IVault.sol
308b5f5f777b980aa93474a3486a3bf46c58ff492caa54a0861d2bea6d254465 ./ZeroExExchangeController.sol
a05aacdbbea3f3777725e6b8ae79377bebe694e62cc617c66643716a1ecf0847 ./MStableExchangeController.sol
72750cdf2d58673e0e530d85a7bfc1eea14c02ef56dcd38fcc857d08c923d6bc ./MStablePoolController.sol
8c07e5be87f50f4c084787461570a6e868a566604690fb4c8cdc7be5d8c81123 ./AavePoolController.sol
2f52f0798b68336412a888d67440a0abf6ffbd597e924d217d9bf6a3d7bb3a96 ./YVaultPoolController.sol
be225c850584d958969354279edaf8a10368591abfdaacaab15ff8cd04e55c16 ./CompoundPoolController.sol
20cbf48071aece05924dade95ad4c6a9dc5f54176aaf00ce1dbf23908f6dba94 ./DydxPoolController.sol
3b567ec501625f6e39798ca89215742b122a82779080a1b04f923801553f0912 ./RariFundPriceConsumer.sol
8b26b75acd0d9690acb6f7ebfe09835a9c1268c15728901fff9c0a44bb4a6c8c ./RariFundController.sol
77cf2a0ca04a8831bb642f9f2ac96f10f8a40aa6e075c2c7110423502de00915 ./Migrations.sol
2d053758e94065351deb9de9b7e3afd5789fc08548bbaac1540c747539d9f89c ./RariFundToken.sol
326342bd019dd32a20e33be9b6f7a85738c217d3a0a4385c06f35271e3b8d268 ./RariFundManager.sol
090ae92722f79f89091ff01127a8c60e6bde4e6c6f8680966d96219fbb839ba9 ./RariFundProxy.sol
30a7222c13e1028a3d87a345020eb1358c09a80b778d7bb5948650c35d1c9bd6 ./IRariGovernanceTokenDistributor.sol
d29f66b465a266862cf6bb410631af0046b34555892e1ccdc5fea1c6d17613b6 ./SoloMargin.sol
5513d1f9cdaf628aff707640f5130a626137b5e9d962e9ffa68c17946efe7105 ./Getters.sol
7e671035218f2845db3298f288f390509b81a30088d7abfa720a3f6bb4d3df43 ./Operation.sol
8550fa9ed4d04778d31fd659c11d3ceed0130794817315e0f8022776880bb690 ./Actions.sol
ce2fe53c7fc82dbcb260d288d82823d94d8048d75e7edd8306fa3d7976b14ece ./Account.sol
867682d15be4c4f45fbfa8ed83328914441bf208c41c5aa448dda028b790f119 ./Types.sol
1f837c92dc7fca41d14103938c1649f09909a3f809ab4953c6136c85abc2d5bb ./CErc20.sol
4117f41ad0e5feecd235e31135b53d95a7e21f93ccdae7315b1917d860df8b49 ./LendingPool.sol
```

145190ac5f73ee74663ade71ee3f3eeb2cbef5847953d3e2632f6ae0a54d6727 ./AToken.sol

bc5cc7a09bf0b9963838380dfeaf5c25ca5afd1d9a9e19fff9f9c7a2fd363de8 ./ISavingsContract.sol

34a6f23b9561c13c2d484041e10ef132174e37722d1cc78c20cc8d2fdbfc5b13 ./IMasset.sol

292bbb87dab48a3ef859d16f14c668a1a1ba6405092880598a4e63a5c3c16bb7 ./IBoostedSavingsVault.sol

59ba0865db8afe7b6f89fa3dbcf24335fcea8d4907bafe2054aead9be687bda1 ./MassetStructs.sol

bbdf57e661ad48bc13b8d64b4d881fd322bd5e27cf32acf0fef097c3d9aa1f04 ./IBasketManager.sol

308b5f5f777b980aa93474a3486a3bf46c58ff492caa54a0861d2bea6d254465 ./ZeroExExchangeController.sol

a05aacdbbea3f3777725e6b8ae79377bebe694e62cc617c66643716a1ecf0847 ./MStableExchangeController.sol

72750cdf2d58673e0e530d85a7bfc1eea14c02ef56dcd38fcc857d08c923d6bc ./MStablePoolController.sol

8c07e5be87f50f4c084787461570a6e868a566604690fb4c8cdc7be5d8c81123 ./AavePoolController.sol

be225c850584d958969354279edaf8a10368591abfdaacaab15ff8cd04e55c16 ./CompoundPoolController.sol

20cbf48071aece05924dade95ad4c6a9dc5f54176aaf00ce1dbf23908f6dba94 ./DydxPoolController.sol

d264ad89fa42513e492cdda74a4134f8c576d8cd93e7c02bb32c13ee622bb251 ./RariGovernanceToken.sol

77cf2a0ca04a8831bb642f9f2ac96f10f8a40aa6e075c2c7110423502de00915 ./Migrations.sol

923d128c34a8ce8654615e5802f3c05c8a85d73dc09358abab9480c1cafb1264 ./RariGovernanceTokenDistributor.sol

5f20d11075d8c062017e3aaf9088c356f06cb9f1610d9af5780704b140885aac ./RariGovernanceTokenUniswapDistributor.sol

290ac6d6bde72191e229ede8f9043bdb27c6ba63ef2437a7d0ecaddde3c06b31 ./RariGovernanceTokenVesting.sol

e1fa2fa54ee493a3bc644a28cbf32fa42e69a213400d0aaffdbc370d892109f0 ./RariGovernanceTokenDistributorV2.sol

f816be56ea5d0d6c9838f3ac98da1b70eb400562375e6fc7f03a78e58c85d214 ./RariGovernanceTokenVestingV2.sol

f8356127357d195067dbd03d989b93c580794210e292467c77cd9e837642a5e7 ./IRariFundToken.sol

cd980d5e956da705aa08d728d5eca9c624cf52f8627d86ba0eb5785182d2dd0e ./IRariFundManager.sol

a5d1754fe1eb2c2f2b3782ce53444f175d96d7c1003f45e2b096a9f9e91b030a ./contracts/RariFundController.sol

77cf2a0ca04a8831bb642f9f2ac96f10f8a40aa6e075c2c7110423502de00915 ./contracts/Migrations.sol

93bbbed6f248f639adfc0b2db17da24c0dcea6c89bb3e43adfe77ce45d64fff9 ./contracts/RariFundToken.sol

62f9be9e413eaa62e3ef1a251665651fdc0ba3d2c01fcb92dcccdc19c5327ead ./contracts/RariFundManager.sol

6d8b06b33cc7c3916a04f34d338dfc367b250fd4626657dd82856a5dbacbf03a ./contracts/RariFundProxy.sol

30a7222c13e1028a3d87a345020eb1358c09a80b778d7bb5948650c35d1c9bd6 ./contracts/interfaces/IRariGovernanceTokenDistributor.sol

a18dc30171210e9cff9d0145b8572e10e143624713ddb3e585b8c16bbc93954a ./contracts/external/alpha/Bank.sol

d29f66b465a266862cf6bb410631af0046b34555892e1ccdc5fea1c6d17613b6 ./contracts/external/dydx/SoloMargin.sol

5513d1f9cdaf628aff707640f5130a626137b5e9d962e9ffa68c17946efe7105 ./contracts/external/dydx/Getters.sol

7e671035218f2845db3298f288f390509b81a30088d7abfa720a3f6bb4d3df43 ./contracts/external/dydx/Operation.sol

8550fa9ed4d04778d31fd659c11d3ceed0130794817315e0f8022776880bb690 ./contracts/external/dydx/lib/Actions.sol

ce2fe53c7fc82dbcb260d288d82823d94d8048d75e7edd8306fa3d7976b14ece ./contracts/external/dydx/lib/Account.sol

867682d15be4c4f45fbfa8ed83328914441bf208c41c5aa448dda028b790f119 ./contracts/external/dydx/lib/Types.sol

785541e17e5f0c00196ecf568457a8177a08f8aab06ef65c65e3b0b560b32da5 ./contracts/external/compound/CEther.sol

f485c7d5e273b3b07e129a89bfe43d55ffc6d35ee41fef668c7f6f13eaff9b55 ./contracts/external/aave/LendingPool.sol

f284d79b5b46b9a2d0d95722205915e961ef3f1e636f56aeaa42ebeb73ca4761 ./contracts/external/aave/AToken.sol

ba5f587247133ff66c9adc9edd35a4d67ddcecbd8b134a3ea93abe599dc31c46 ./contracts/external/enzyme/ComptrollerLib.sol

b9085d46579c616cb76d658892ab5a0d98fd034ba0a4e122bf4ca8590bff2db7 ./contracts/external/keeperdao/IKToken.sol

0923ec8fcbde7cd58201f6d6f8030fc6453c0e7a1d66317d4abeb286afd769cf ./contracts/external/keeperdao/ILiquidityPool.sol

c933d5a52a081b6294006225d00b36753b76991fcb396603ff02e10533f56d69 ./contracts/lib/exchanges/ZeroExExchangeController.sol

ba5b8add9ecc255ff8c187b6f31bbe0d68eabf65bb6ad5e5b95628ba19598bb7 ./contracts/lib/pools/AlphaPoolController.sol

b1fe0018736f929c3818298e055ec9a61185791dce8f02fecd496a6a3dc63488 ./contracts/lib/pools/KeeperDaoPoolController.sol

c452837cfca3422a988ce42e32d180e69193a79d37cb7dcf40f3bd0d4c414d9c ./contracts/lib/pools/AavePoolController.sol

da2307f859ae1d2fb22074e0d20dfeee61bcc51740dea08a969cac8605dc973d ./contracts/lib/pools/CompoundPoolController.sol

069db27f2a4c69b957fbfddba095589532a52eed259101e6763be3a60e171a32 ./contracts/lib/pools/EnzymePoolController.sol

93198800235b113044fd394643704de33d8795ef2a6054b1f5d0f521ab0f9bf4 ./contracts/lib/pools/DydxPoolController.sol

Tests

421a18f9be96106a97ba068068b63f9380488de92b7427a2d8dd7974a8e461b3 ./1_fund_rebalancer_exchange_0x.js

442484b20f4fd515cbe9441a32222c7840f77302f87a03aa748cbe9c937c759c ./4_fund_rebalancer.js

ebca33eb1e90547d6c4392d85f5ee606e233cd330c62e5914475c4de39711c18 ./5_fund_user.js

17729dac7b5bd52fa235d8490f77a2f2ca2994d21bdfb7a62413650e1077b414 ./2_fund_user_exchange_gas.js

08acf7aa3be495082ba899add21267c301bee8094d0dc4de1aa653befc0a375e ./6_fund_fees.js

453e74f2872a81c549e45ab26b735fcdfddf6f9789a6f2d7f3f145e1bbfdbc33 ./7_fund_upgrade_gas.js

18348d3ee0b80e7b766306592e60bdd0b345884933d18c8c0562d461932dab24 ./3_fund_owner.js

557f3edfac30e6091a5054f7b07272a31fdad800c903cc7950d50d07d74ac9d2 ./0x.js

```
37890bf572ba2c95c88944243a22141ee112fa988cd14c62e3ae06e19837aca2 ./1_fund_rebalancer_exchange_0x.js
00abdf77e3ee337447efef1c6867e6ff961fc43ebc8212715dc1e789837277fe ./4_fund_rebalancer.js
ce9b1bea3deb2364b690882ccdcae14da33408229a768cb4c9148766a2d0a00b ./5_fund_user.js
83081324e345f124e056e28e1936c3cf32b848b5627d2f9189689df9c7d00659 ./2_fund_user_exchange_gas.js
d5bb49682e9cd7f7e76a6a7446a7b26d59fddb983455a81cd3582aae0d7ee578 ./6_fund_fees.js
71e81a11ebb5768a25d63ad884eef55451436a99d45f6eb4087199cf1c96bde0 ./7_fund_upgrade_gas.js
3efbdd78dbc664385079fdff8351d89ba77b9e8e4d0505dee66605ebf4f2d765 ./3_fund_owner.js
557f3edfac30e6091a5054f7b07272a31fdad800c903cc7950d50d07d74ac9d2 ./0x.js
8beca87801d9bb99c969a1f3995dffc64230d86d2c401522bd8d2f4101446c31 ./2_governance_token_vesting.js
74798d469378177c36d1c4ed2d09950116720c6d4f5fe26833c4f377b755f34b ./3_governance_token_uniswap_distribution.js
05653814a0d39a1d48be911885d6146ca9dd6c2a9d54d5ec905c3d7197f39713 ./1_governance_token_distribution.js
16a5813cd27e0f62db688d3391015f57140d8e451c737532038c6ab5063277d8 ./5_governance_token_vesting_v2.js
62b38375a2a2655e7104634113214c45e7a6d1af9e459537d0b7a846613b892a ./4_governance_token_distribution_v2.js
3b0f1f1175b8cc402db553331d2a3738d66d2049014382b73c4f61f432edf445 ./test/block-gas-limit.js
4bad6202f04259727f9ddaaae97ac45ae2676acf3a1af191c808ebe609edcf70 ./test/fund-fees.js
2d0ea1d1b44f5c41b4d073ca5266f83910dc255b24331815fe92fe15a7e50de2 ./test/fund-user.js
a9fa84242b35ac18d9f06c70a4f561375b04109064aece6d875fa9d4155b7bbe ./test/fund-owner.js
17b5d2dcd9b66a09251313a0eb609747be92f1fad0be1dc45f7c4c412b836f9f ./test/fund-rebalancer.js
0eb63d18f979a40ada12ae5db36cc69f92e5016896fb64f0b5ff405bd8e37396 ./test/keeperdao-integration.js
dd0490b5bc0b3a7743f47c5f97e8d9bec341212be4b5461250f6cb8192943a25 ./test/exchanges/0x.js
```

Changelog

- 2020-08-20 - Initial report based on commit `66e2dc5`
- 2020-09-21 - Updated report based on commit `62b5011`
- 2020-10-23 - Updated report based on commit `ae98c4f` and added audit for 3 new repos
- 2020-12-04 - Updated report based on commits: (1) `200cde7` for rari-governance-contracts, (2) `737ff0d` for rari-yield-pool-contracts, (3) `dc5de88` for rari-stable-pool-contracts and (4) `390237d` for rari-ethereum-pool-fund
- 2021-02-04 - Updated report for new commits: `rari-stable-pool-contracts` (`749d4f8`), `rari-yield-pool-contracts` (`3bb28f4`), `rari-ethereum-pool-fund` (`b87de06`), `rari-governance-contracts` (`ccd9424`)
- 2021-03-04 - Updated report for new commit: `rari-stable-pool-contracts` (`feaa246`), `rari-yield-pool-contracts` (`479a346`), `rari-ethereum-pool-contracts` (`75fb256`), `rari-governance-contracts` (`83238f7`)

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.