RAY v2

Rapid Code Review

June 2, 2020

Prepared For:
Devan Purhar | *Staked*
devan@staked.us

Prepared By:
Michael Colburn | *Trail of Bits*
michael.colburn@trailofbits.com

Review Summary

On May 31, 2020, Trail of Bits performed an assessment of version 2 of Staked's RAY smart contracts with one engineer, and reported no issues.

Throughout this assessment, we sought to answer various questions about the security of the RAY system. We focused on flaws that would allow an attacker to:

- Gain unauthorized access to user funds
- Bypass access controls to modify contract state
- Interfere with interactions between RAY components

Our review identified no issues in the codebase. The project had thorough documentation, functions were well-organized and adhered to a consistent style, and common Solidity issues were avoided or otherwise documented. Due to the complexity of the system and the short length of the engagement, review of the arithmetic was focused more on standard Solidity arithmetic pitfalls rather than potential domain-specific issues relating to exchange rates and net asset value calculations.

On the following page, we review the maturity of the codebase and the likelihood of future issues. In each area of control, we rate the maturity from strong to weak, or missing, and give a brief explanation of our reasoning. Staked should consider these steps to improve their security maturity:

- Integrate fuzzing or symbolic execution to test the correctness of contract functionality.
- Use crytic.io for any new code development.
- Conduct further in-depth review focused on the system's arithmetic.

Code Maturity Evaluation

Category Name	Description
Access Controls	**Satisfactory.** Appropriate access controls were in place for transferring tokens and modifying allowances.
Arithmetics	**Further Investigation Required.** The contracts made consistent use of `SafeMath` to prevent overflow, even in areas where overflow was not possible. The Staked team was also aware of areas where precision may have been lost due to integer division. However, the limited timeframe of the review did not allow for deeper investigation of the system's frequent use of arithmetic.
Assembly Use	**Not Applicable.** The contracts did not include any assembly outside of the vendored OpenZeppelin libraries.
Centralization	**Moderate.** Governance was carried out via an `AdminTwo` contract which had the ability to carry out a number of administrative operations. The architecture of the admin contract would allow it to be easily migrated to a decentralized governance model.
Contract Upgradeability	**Satisfactory.** The contracts made use of both the OpenZeppelin proxy upgradeability implementation as well as the eternal storage pattern for different components in the system. The Staked team had documented the necessary requirements to avoid common issues with these upgradeability patterns.
Function Composition	**Strong.** Functions were organized and scoped appropriately.
Front-Running	**Further Investigation Required.** Due to time constraints, front-running issues were not heavily considered. However, `RoboTokens` did include the `increaseAllowance` and `decreaseAllowance` functions to help mitigate the ERC20 race condition.
Monitoring	**Satisfactory.** All functions that modified balances or contract state emitted events. The events themselves, though, were emitted from the `AdminTwo` contract, where they were invoked instead of `RAYv2`, where the state changes took place.
Specification	**Strong.** The project was accompanied by thorough documentation and the code itself had comprehensive comment coverage.
Testing & Verification	**Satisfactory.** The repositories included tests for a variety of scenarios.

Project Dashboard

Pull requests reviewed from the `ray-smart-contracts` repository:

- PR 33
- PR 36
- PR 38
- PR 39
- PR 40

Appendix A. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.