ROCKET POOL

Rocket Pool Protocol

Rocket Pool Swap Router Review

Version: 1.0

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Rocket Pool smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Rocket Pool smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Rocket Pool smart contracts.

Overview

The Rocket Pool Swap Router is designed to optimally swap ETH and rETH using a combination of Uniswap and Balancer. This review focuses solely on the single smart contract implementation that handles the swapping logic between these two tokens.

Security Assessment Summary

The Rocket Pool Swap Router review was conducted on the files hosted on the rocket-pool/rocketpool-router repository and were assessed at commit e14d3e.

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 3 issues during this assessment. Categorised by their severity:

- Low: 1 issue.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Rocket Pool smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as "informational".

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
RPS-01	Deposited Ether Gets Taken By The Next User To Call SwapFrom	Low	Open
RPS-02	In-Protocol Swaps Only Occur For The Ideal Case	Informational	Open
RPS-03	Miscellaneous General Comments	Informational	Open

RPS-01	Deposited Ether Gets Taken By The Next User To Call SwapFrom		
Asset	`contracts/RocketSwapRouter.sol`		
Status	**Open**		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The contract currently allows Ether to be sent to the contract via the `receive()` function on line [**55**]. Any Ether accidentally sent by a user can be claimed by anyone through the use of the `swapFrom()` function, which sends the entire Ether balance of the contract to the caller on line [**167**].

The receive/fallback function is required in order to receive Ether from unwrapping `WETH` in the swap process. To avoid external Ether from being sent to the contract, this could be limited by only allowing Ether sent from the `WETH` contract.

Recommendations

Limit the Ether that can be sent to the contract. In the `receive()` function add a `require(msg.sender == WETH)` to prevent external Ether being sent.

RPS-02	In-Protocol Swaps Only Occur For The Ideal Case
Asset	`contracts/RocketSwapRouter.sol`
Status	**Open**
Rating	Informational

Description

The logic for using the RocketPool minting/burning process to swap tokens is only activated if the rate produces the ideal token amounts (or more). If the ideal token amounts cannot be achieved solely by the in-protocol swaps, `uniswap` and/or `balancer` is used to perform the conversion.

There is the possibility that the in-protocol swap out-performs the defi procotol swaps when generating tokens which yeild between the `_minTokensOut` and `_idealTokensOut` .

Recommendations

This appears by design and is only raised so that the authors are aware of potential efficiencies that could be gained. It could be possible to allow the caller to specify the ratio of in-protocol swaps and determine the best ratio's off-chain. Handling the logic on-chain is also a solution however this would increase the complexity and gas cost of regular swaps and these downsides may outweigh the gain in implementing this logic.

RPS-03	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Open**
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **One sentance title.**
   Details of issue if more lines are needed.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: `https://blog.sigmaprime.io/solidity-security.html`. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: `http://www.dasp.co/`. [Accessed 2018].