Swell Network

Swell Liquid Restaking Token

Smart Contract Security Assessment

Version: 2.0

January, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Swell Network LRT 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 Swell Network LRT 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*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Swell Network LRT smart contracts.

Overview

Swell LRT is an Ethereum liquid restaking protocol. It provides users with a non custodial means of liquid restaking via a transferable ERC-20 token called `rswETH`. This protocol is powered by EigenLayer and its EigenPod service.

This review covers the launch of the liquid restaking token `rswETH` from the previous version of the protocol, which was a liquid staking token, `swETH`.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to changes between `v2-contracts-lrt` and `v3-contracts-lst` captured at the following diff.

Retesting was performed on commit 80acd7f.

Note: third party libraries and dependencies, such as OpenZeppelin, were excluded from the scope of this assessment.

Approach

The review was conducted on the files hosted on the Swell Network repository at commit 97f2535.

The manual review focused on identifying issues 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 identifying vulnerabilities related to known Solidity anti-patterns and attack vectors, such as re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers.

For a more detailed, but non-exhaustive list of examined vectors, see [1, 2].

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

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

Output for these automated tools is available upon request.

Coverage Limitations

Due to a time-boxed nature of this review, all documented vulnerabilities reflect best effort within the alloted, limited engagement time. As such, Sigma Prime recommends to further investigate areas of the code, and any related functionality, where majority of critical and high risk vulnerabilities were identified.

Findings Summary

The testing team identified a total of 4 issues during this assessment. Categorized by their severity:

- Low: 1 issue.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Swell Network 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
SLRT-01	Unexpected Reverts When Updating Operator Address	Low	Resolved
SLRT-02	Cumbersome Access Control Arrangement For Repricing Bot	Informational	Resolved
SLRT-03	Missing Events When Updating Operator	Informational	Resolved
SLRT-04	Miscellaneous General Comments	Informational	Resolved

SLRT-01	Unexpected Reverts When Updating Operator Address		
Asset	NodeOperatorRegistry.sol		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Updating an operator address to the same value via `updateOperatorControllingAddress()` will result in an unexpected revert.

If `updateOperatorControllingAddress()` is called with the same, valid, operator address for both arguments, it will update the operator to the same value it already has and then, on line [**364**], delete the entry in `getOperatorIdForAddress` for that address.

With this entry deleted, all calls to `_getOperatorIdSafe()` will revert for this operator address, and so all functions that call `_getOperatorIdSafe()` will revert for this operator. Of these, the most impactful is likely `addNewValidatorDetails()`, meaning the operator would not be able to add new validators.

This could be fixed by calling `addOperator()` to add the operator again, although the ID number would then be different, which might cause other issues.

This issue is mitigated by the fact that `updateOperatorControllingAddress()` is an admin function, and so is less likely to be called with bad arguments.

Recommendations

Consider adding a check to `updateOperatorControllingAddress()` which reverts or returns if the two arguments are the same as each other.

Resolution

The addition of a check to `updateOperatorControllingAddress()` was made, which reverts if the two arguments are the same.

This issue has been addressed in commit cae2c4f.

SLRT-02	Cumbersome Access Control Arrangement For Repricing Bot
Asset	`RswETH.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The access control on both `RswETH.reprice()` and `RepricingOracle.submitSnapshot()` check for the `SwellLib.BOT` role via `checkRole(SwellLib.BOT)`, even though the latter function calls the former. The development team confirmed that the bot should not call `RswETH.reprice()` directly.

Giving the `SwellLib.BOT` role to the `RepricingOracle` contract, which is not a bot, is misleading and could lead to errors.

It is also undesirable for the bot to have the ability to call `RswETH.reprice()` directly when this is not part of its intended function. If the bot was compromised or accidentally rolled back to an old version, this would allow it to bypass the checks in `RepricingOracle._assertRepricingSnapshotValidity()` and also avoid emitting the monitored events in `RepricingOracle.submitSnapshot()` when repricing.

Recommendations

Consider creating a `reprice` role for the function `RswETH.reprice()` and granting it to the repricing oracle and any other manually controlled addresses that are required for emergencies. This would also mean that only the bot has the `SwellLib.BOT` role.

Resolution

The addition of the `REPRICER` role was made and granted to the `RepricingOracle` address.

This issue has been addressed in commit cae2c4f.

SLRT-03	Missing Events When Updating Operator
Asset	`NodeOperatorRegistry.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `updateOperatorControllingAddress()` effectively adds and removes an operator addresses, but this is not reflected in events.

Chain monitoring tools that maintain a list of operators could have incorrect data when tracking the system's operator list via events.

Recommendations

Consider adding an event for the function `updateOperatorControllingAddress()` that emits both the old and new address for an operator.

Resolution

The addition of an event after updating the operator address was made.

This issue has been addressed in commit cae2c4f.

SLRT-04	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:**
Rating	Informational

Description

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

1. **Possible Snapshot Validity Checks**

   Related Asset(s): RepricingOracle.sol

   In `_assertRepricingSnapshotValidity()`, consider checking that the consensus layer slot and the timestamp have each increased since the previous snapshot.

2. **Check Zero Address**

   Related Asset(s): RepricingOracle.sol

   Consider checking that the argument for `setExternalV3ReservesPoROracleAddress()` is not the zero address.

3. **Function Can Be Called With Empty Parameters**

   Related Asset(s): DepositManager.sol

   The function `setupValidators()` can be called with a zero length array for its parameter `_pubKeys`, in which case it sets up no validators and still emits the `ValidatorsSetup` event.

   Consider whether this behaviour is desirable.

4. **Inconsistent Contract Documentation**

   Related Asset(s): RepricingOracle.sol

   There is no documentation for the `RepricingOracle` contract explaining its purpose.

   Consider adding a brief contract documentation such that the reader quickly understands what the contract does and how/when it is used.

5. **Values Could Be Constants**

   Related Asset(s): DepositManager.sol

   (a) depositAmount from line [**104**] is a hard coded value and could be a constant.

   (b) DepositContract from line [**65**] is a hard coded value and could be a constant.

6. **Unused Import**

   Related Asset(s): DepositManager.sol

   Consider removing the unused import of `AddressUpgradeable` in `DepositManager`.

Recommendations

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

Resolution

The comments above have been acknowledged by the development team, and relevant changes actioned in cae2c4f where appropriate.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `forge` framework was used to perform these tests and the output is given below.

```
Running 1 test for src/contracts/mocks/MockRepricingOracleForUpgrade.sol:MockRepricingOracleForUpgrade
[PASS] testValue() (gas: 2388)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 554.55µs

Running 1 test for test/Repricing.t.sol:RepricingTest
[PASS] testRepricingTotalReserves() (gas: 810)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.53ms

Running 32 tests for test/AccessControlManager.t.sol:AccessControlManagerTest
[PASS] test_Deployment() (gas: 31262)
[PASS] test_Getters() (gas: 12727)
[PASS] test_RevertWhen_pauseBotMethods_AlreadyPaused() (gas: 28428)
[PASS] test_RevertWhen_pauseBotMethods_NonAdminCaller() (gas: 79332)
[PASS] test_RevertWhen_pauseCoreMethods_AlreadyPaused() (gas: 28490)
[PASS] test_RevertWhen_pauseCoreMethods_NonAdminCaller() (gas: 79309)
[PASS] test_RevertWhen_pauseOperatorMethods_AlreadyPaused() (gas: 28443)
[PASS] test_RevertWhen_pauseOperatorMethods_NonAdminCaller() (gas: 79377)
[PASS] test_RevertWhen_pauseWithdrawals_AlreadyPaused() (gas: 22535)
[PASS] test_RevertWhen_pauseWithdrawals_NonAdminCaller() (gas: 79418)
[PASS] test_RevertWhen_setDepositManager_NonAdminCaller() (gas: 79482)
[PASS] test_RevertWhen_setDepositManager_WithZeroAddress() (gas: 20475)
[PASS] test_RevertWhen_setNodeOperatorRegistry_NonAdminCaller() (gas: 79525)
[PASS] test_RevertWhen_setNodeOperatorRegistry_WithZeroAddress() (gas: 20562)
[PASS] test_RevertWhen_setRswETH_NonAdminCaller() (gas: 79571)
[PASS] test_RevertWhen_setRswETH_WithZeroAddress() (gas: 20476)
[PASS] test_RevertWhen_setSwellTreasury_NonAdminCaller() (gas: 79493)
[PASS] test_RevertWhen_setSwellTreasury_WithZeroAddress() (gas: 20486)
[PASS] test_RevertWhen_unpauseOperatorMethods_AlreadyPaused() (gas: 22434)
[PASS] test_RevertWhen_unpauseOperatorMethods_NonAdminCaller() (gas: 79375)
[PASS] test_RevertWhen_unpauseWithdrawals_AlreadyPaused() (gas: 28450)
[PASS] test_RevertWhen_unpauseWithdrawals_NonAdminCaller() (gas: 79330)
[PASS] test_pauseBotMethods_HappyPath() (gas: 27739)
[PASS] test_pauseCoreMethods_HappyPath() (gas: 27803)
[PASS] test_pauseOperatorMethods_HappyPath() (gas: 27818)
[PASS] test_pauseWithdrawals_HappyPath() (gas: 28268)
[PASS] test_setDepositManager_HappyPath() (gas: 30220)
[PASS] test_setNodeOperatorRegistry_HappyPath() (gas: 30243)
[PASS] test_setRswETH_HappyPath() (gas: 30166)
[PASS] test_setSwellTreasury_HappyPath() (gas: 30132)
[PASS] test_unpauseOperatorMethods_HappyPath() (gas: 28194)
[PASS] test_unpauseWithdrawals_HappyPath() (gas: 27775)
Test result: ok. 32 passed; 0 failed; 0 skipped; finished in 9.23ms

Running 13 tests for test/RepricingOracle.t.sol:RepricingOracleTest
[PASS] testRoAssertRepricingSnapshotValidityErrors() (gas: 154490)
[PASS] testRoAssertRepricingSnapshotValiditySkip() (gas: 100063)
[PASS] testRoAssertRepricingSnapshotValidityVanilla() (gas: 57948)
[PASS] testRoCheckZeroAddress() (gas: 1790354)
[PASS] testRoFallback() (gas: 15608)
[PASS] testRoInitialize() (gas: 23045)
[PASS] testRoReceive() (gas: 23535)
[PASS] testRoSetExternalV3ReservesPoROracleAddress() (gas: 43538)
[PASS] testRoSetMaximumRepriceBlockAtSnapshotStaleness() (gas: 40889)
[PASS] testRoSetMaximumRepriceV3ReservesExternalPoRDiffPercentage() (gas: 40813)
[PASS] testRoSubmitSnapshot() (gas: 658680)
[PASS] testRoUnsetExternalV3ReservesPoROracleAddress() (gas: 40063)
[PASS] testRoWithdrawERC20() (gas: 262381)
Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 9.92ms

Running 22 tests for test/NodeOperatorRegistry.t.sol:NodeOperatorRegistryTest
[PASS] testNorAddNewValidatorDetailsErrors() (gas: 344545)
```

```
[PASS] testNorAddNewValidatorDetailsVanilla() (gas: 491209)
[PASS] testNorAddOperator() (gas: 170863)
[PASS] testNorCheckZeroAddress() (gas: 4011246)
[PASS] testNorDeleteActiveValidatorsErrors() (gas: 2021320)
[PASS] testNorDeleteActiveValidatorsVanilla() (gas: 1815020)
[PASS] testNorDeletePendingValidatorsErrors() (gas: 1920390)
[PASS] testNorDeletePendingValidatorsVanilla() (gas: 1794486)
[PASS] testNorDisableEnableOperator() (gas: 119941)
[PASS] testNorFallback() (gas: 15655)
[PASS] testNorGetNextValidatorDetails() (gas: 2351265)
[PASS] testNorGetPoRAddressList() (gas: 2476815)
[PASS] testNorGetRewardDetailsForOperatorId() (gas: 2044339)
[PASS] testNorInitialize() (gas: 17060)
[PASS] testNorReceive() (gas: 23631)
[PASS] testNorUpdateOperatorControllingAddress() (gas: 132058)
[PASS] testNorUpdateOperatorName() (gas: 82940)
[PASS] testNorUpdateOperatorRewardAddress() (gas: 98914)
[PASS] testNorUsePubKeysForValidatorSetupErrors() (gas: 2107453)
[PASS] testNorUsePubKeysForValidatorSetupVanilla() (gas: 2142792)
[PASS] testNorWithdrawERC20() (gas: 262742)
[PASS] testPocUpdateOperatorToSelf() (gas: 53222)
Test result: ok. 22 passed; 0 failed; 0 skipped; finished in 12.68ms

Running 22 tests for test/RswETH.t.sol:RswETHTest
[PASS] test_depositWithReferral_HappyPath_OneDepositer() (gas: 145360)
[PASS] test_deposit_FuzzedValue(uint256) (runs: 1000, : 148818, ~: 148818)
[PASS] test_deposit_HappyPath_NDepositers() (gas: 204086)
[PASS] test_deposit_HappyPath_OneDepositer() (gas: 145101)
[PASS] test_deposit_HappyPath_OneDepositer_CoreMethodsPaused() (gas: 53735)
[PASS] test_deposit_HappyPath_OneDepositer_ZeroAmount() (gas: 42150)
[PASS] test_ethToRswETHRate_HappyPath() (gas: 15190)
[PASS] test_getRate_HappyPath() (gas: 16475)
[PASS] test_reprice_HappyPath() (gas: 2254733)
[PASS] test_rswETHToETHRate_HappyPath() (gas: 14892)
[PASS] test_setMaximumRepriceDifferencePercentage_HappyPath() (gas: 42267)
[PASS] test_setMaximumRepriceDifferencePercentage_NonAdminCaller() (gas: 92035)
[PASS] test_setMaximumRepricerswETHDifferencePercentage_HappyPath() (gas: 42254)
[PASS] test_setMaximumRepricerswETHDifferencePercentage_NonAdminCaller() (gas: 92079)
[PASS] test_setMinimumRepriceTime_HappyPath() (gas: 37453)
[PASS] test_setMinimumRepriceTime_NonAdminCaller() (gas: 92046)
[PASS] test_setNodeOperatorRewardPercentage_HappyPath() (gas: 45964)
[PASS] test_setNodeOperatorRewardPercentage_NonAdminCaller() (gas: 95631)
[PASS] test_setNodeOperatorRewardPercentage_Overflow() (gas: 36738)
[PASS] test_setSwellTreasuryRewardPercentage_HappyPath() (gas: 46009)
[PASS] test_setSwellTreasuryRewardPercentage_NonAdminCaller() (gas: 95587)
[PASS] test_setSwellTreasuryRewardPercentage_Overflow() (gas: 36652)
Test result: ok. 22 passed; 0 failed; 0 skipped; finished in 114.46ms

Running 9 tests for test/DepositManager.t.sol:DepositManagerTest
[PASS] testDmCheckZeroAddress() (gas: 1978580)
[PASS] testDmCreateEigenPod() (gas: 453046)
[PASS] testDmFallback() (gas: 15652)
[PASS] testDmInitialize() (gas: 20848)
[PASS] testDmReceive() (gas: 26461)
[PASS] testDmSetupValidatorsEmpty() (gas: 505188)
[PASS] testDmSetupValidatorsErrors() (gas: 411795)
[PASS] testDmSetupValidatorsVanilla() (gas: 746568)
[PASS] testDmWithdrawERC20() (gas: 253547)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 115.62ms

Ran 7 test suites: 100 tests passed, 0 failed, 0 skipped (100 total tests)
```

Appendix B 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].

σ'