Swell Network

Restaking Contracts

Security Assessment Report

Version: 2.0

July, 2024

Contents

Introduction

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

Overview

Swell Network is an Ethereum liquid staking protocol. It provides users with non-custodial means of liquid staking via a transferable ERC-20 token called `swETH`.

Swell Network has recently updated their system to support restaking with services such as EigenLayer. To do this they created the repricing ERC20 token `rswETH`, which allows users of Swell Network to access collateral value that is backing such restaking via use of `rswETH`.

The focus of this security review primarily targets the architecture needed to support `rswETH`'s operations and the Liquid Staking Token collaterals that can be used to mint it.

Security Assessment Summary

Scope

The review was conducted on the files hosted on the Swell Network repository.

The scope of this time-boxed review was strictly limited to the following files at the commit efda60f:

- `implementations/RateProviders/*`
- `implementations/DepositManager.sol`
- `implementations/StakerProxy.sol`
- `implementations/EigenLayerManager.sol`
- `implementations/NodeOperatorRegistry.sol`
- `implementations/RepricingOracle.sol`
- `implementations/RswETH.sol`
- `implementations/RswEXIT.sol`

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

Approach

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`
- Surya: `https://github.com/ConsenSys/surya`

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 allotted, 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 14 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.
- High: 1 issue.
- Medium: 3 issues.
- Low: 3 issues.
- Informational: 6 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
SWL3-01	Token Strategy Balances Do Not Get Tracked in Repricing	Critical	Closed
SWL3-02	Hardcoded Token Exchange Rates	High	Closed
SWL3-03	Frontrunning `addNewValidatorDetails()` To Steal Rewards	Medium	Closed
SWL3-04	Rebasing Rewards Not Received In Contracts	Medium	Closed
SWL3-05	EigenLayer Does Not Support `wstETH`	Medium	Closed
SWL3-06	`_referenceOnChainRate()` Uses Incorrect Execution Layer Balance	Low	Resolved
SWL3-07	Existing Depositors Can Block Validator Deployment	Low	Closed
SWL3-08	Incorrect `maximumReferencePriceDiff` Calculation	Low	Resolved
SWL3-09	Forced Undelegations Do Not Update `operatorToStakers` Mapping	Informational	Resolved
SWL3-10	Specific LST Deposits Cannot Be Paused	Informational	Closed
SWL3-11	Use of Rebasing Liquid Staking Tokens	Informational	Closed
SWL3-12	Admin Restricted Withdrawal System	Informational	Closed
SWL3-13	Liquid Staking Token Compositions	Informational	Closed
SWL3-14	Miscellaneous General Comments	Informational	Closed

SWL3-01	Token Strategy Balances Do Not Get Tracked in Repricing		
Asset	`RepricingOracle.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The repricing bot does not keep track of underlying token balances in EigenLayer strategies. Hence, `rswETH` minted from LST deposits into the Swell Network are inflationary to the `rswETH/ETH` exchange rate, resulting in a loss of funds for all existing `rswETH` holders.

The `rswETH/ETH` exchange rate is repriced via the following formula:

$$rswETHToETHRate = \frac{totalReservesInETH}{rswETHTotalSupply}$$

`totalReservesInETH` does not account for the balance in EigenLayer strategies or token balances in the `DepositManager` contract. This means that any LST deposits into the Swell Network are not accounted for in the `rswETH/ETH` exchange rate, resulting in a loss of funds for existing `rswETH` holders when the rate is repriced.

A decrease in the `rswETH/ETH` rate will trigger a protocol lockdown, causing all protocol functionality to pause via the following check in `submitSnapshot()` and `submitSnapshotV2()`:

```
if (rswETHToETHRate > newRswETHToETHRate) {
    AccessControlManager.lockdown();
}
```

However, if the rewards accrued since the time of the last repricing snapshot are enough to offset the new `rswETH` minted via LST deposits, such that the `rswETH/ETH` rate does not decrease after repricing, then deposits and withdrawals can be facilitated with the incorrect `rswETH/ETH` rate, resulting in losses for existing `rswETH` holders.

Recommendations

Account for all LST token balances in the following contracts:

- `DepositManager`
- Every `StakerProxy`
- Every `StakerProxy`'s EigenLayer strategy balances in underlying token using `StrategyBase::userUnderlyingView()`
- Any EigenLayer uncompleted queued withdrawals for token strategies in `DelegationManager`

These LST token balances can be converted into ETH values using the rate providers.

Resolution

The Swell Network team has acknowledged the issue with the following comment:

> *"We won't be supporting LSTs at this time as the RepricingOracle is not ready. We won't be onboarding tokens until it is complete."*

SWL3-02	Hardcoded Token Exchange Rates		
Asset	`SfrxETHRateProvider.sol, StETHRateProvider.sol, WstETHRateProvider.sol, OETHRateProvider.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The `stETH`, `wstETH`, `sfrxETH`, and `OETH` rate providers make major assumptions about exchange rates relative to ETH. This can result in lost funds for the protocol in the event these assumptions are broken.

The `stETH` and `OETH` rate providers hardcode their rates as `1e18`, and the `wstETH` and `sfrxETH` rate providers assume that `stETH` and `frxETH` are 1:1 pegged to ETH. This is particularly dangerous for `sfrxETH` as Frax Finance's documents state the peg of `frxETH` to ETH is loosely constrainted to the range [0.99, 1.01], meaning that the chance of a 1% depeg is higher than with other LSTs.

When the real exchange rate of the LST is lower than the rate provided by the rate provider, an exploiter can deposit the LST to mint `rswETH` at a discount, resulting in a loss of value of the `rswETH` token. When the real exchange rate of the LST is higher than the rate provided by the rate provider, then the depositor will lose some funds after the deposit due to paying a premium for `rswETH`.

Recommendations

It is recommended to not hardcode exchange rates for any of the LSTs. Instead, making use of a trusted price oracle such as Chainlink, or by including secondary price oracle sources such as DeFi liquidity pools to vet the hardcoded exchange rate's accuracy. These could then trigger a revert when attempting to mint `rswETH` if the LST has depegged from the intended exchange rate.

Additional security checks can also be added to reduce the risk of depegs such as LST-specific minting caps that reduce the impact of token depegs on the system.

Resolution

The Swell Network team has communicated that they do not currently intend to launch LSTs, hence this issue has been closed.

SWL3-03	Frontrunning `addNewValidatorDetails()` To Steal Rewards		
Asset	`NodeOperatorRegistry.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

A malicious operator can frontrun an honest operator's `addNewValidatorDetails()` call to compromise their `ValidatorDetails` and steal their `rswETH` rewards.

Node operators register their validators by adding their validator details by calling `addNewValidatorDetails()`. Once their validator is registered, a Swell bot can activate the validator and stake 32 `ETH` on the beacon chain by calling `EigenLayerManager::stakeOnEigenLayer()` with the validator's public key.

The validator details provided by the operator are below:

```
/**
* @dev  Struct containing the required details to setup a validator on the beacon chain
* @param pubKey Public key of the validator
* @param signature The signature of the validator
*/
struct ValidatorDetails {
    bytes pubKey;
    bytes signature;
}
```

`ValidatorDetails::signature` refers to the BLS signature obtained from signing the `DepositMessage` SSZ container. This signature does not prove that the validator belongs to the operator. Hence, it is possible for a malicious operator to frontrun the `addNewValidatorDetails()` call to register the validator details belonging to another operator.

During repricing, the node operators receive a cut of the total `nodeOperatorRewards` based on how many active validators they have registered. Hence, the malicious operator will steal the validator's share of `rswETH` rewards from the honest operator.

Recommendations

The optimal solution is to have the validator sign a message that proves that the validator belongs to the operator. This message can be verified in `addNewValidatorDetails()`. However, this may not be possible as of time of writing due to current restrictions in consensus clients and the lack of a BLS12-381 precompile until the Prague upgrade.

Alternatively, consider introducing a delay period before an operator's validator can be used for staking and allow operators to report instances of frontrunning to the Swell Network team.

Resolution

The Swell Network team has acknowledged the issue with the following comment:

> *"We've decided to go for a backend/bot based solution as this will be easier/cleaner than upgrading the existing contracts and data structures. In a future upgrade, we'll incorporate this check at the smart contract level."*

The backend based solution was not in-scope of this review, hence this issue has been closed.

SWL3-04	Rebasing Rewards Not Received In Contracts		
Asset	`DepositManager.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

One of the Liquid Staking Tokens available to use is Origin ETH. When a smart contract holds `OETH` it defaults to not receiving rebasing rewards. This means that `OETH` held by the `DepositManager` would not be earning staking rewards until processed.

Origin ETH requires smart contracts to opt in to receive staking reward rebases to help prevent naive contracts from having their accounting systems broken by the rebasing mechanic. Contracts must call `OETH.rebaseOptIn()` in order to benefit from these rebases.

This issue is rated as both medium severity and likelihood to acknowledge the fact that once `OETH` is fully restaked with Eigenlayer, the `OETH` tokens do receive rebasing rewards. Only funds pending transfer in `DepositManager` cease to receive rebasing rewards. Furthermore, the original deposited sums are not at risk, only additional rewards earned from staking, which limits the severity to medium.

Recommendations

Including a call to `OETH.rebaseOptIn()` in the `initialize()` function would ensure the `DepositManager` still benefits from staking rewards.

Alternatively, the team can use offchain means to ensure regular forwarding of pending `OETH` balances such that any lost rewards are negligible in size.

Resolution

The Swell Network team has communicated that they do not currently intend to launch LSTs, hence this issue has been closed.

SWL3-05	EigenLayer Does Not Support `wstETH`		
Asset	`DepositManager.sol, EigenLayerManager.sol, WstETHRateProvider.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

`wstETH` deposited by users will not be deposited into EigenLayer as EigenLayer does not support `wstETH`.

The existence of the `WstETHRateProvider` suggests that the Swell Network supports deposits of `wstETH` via `DepositManager::depositLST()`. However, EigenLayer's `stETH` strategy does not support deposits of `wstETH`.

This means that `wstETH` deposited by users will not be deposited into EigenLayer and will stay in `DepositManager`. EigenLayer rewards will not be accrued for `wstETH` deposited by users, resulting in a loss of EigenLayer rewards for the Swell Network, diluting rewards to all `rswETH` holders.

Recommendations

Consider removing support for `wstETH` deposits into `DepositManager` or unwrapping `wstETH` before depositing into EigenLayer inside `DepositManager::transferTokenForDepositIntoStrategy()`.

Resolution

The Swell Network team has communicated that they do not currently intend to launch LSTs, hence this issue has been closed.

SWL3-06	`_referenceOnChainRate()` Uses Incorrect Execution Layer Balance		
Asset	`RepricingOracle.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

The `_referenceOnChainRate()` function does not account for `ETH` balances of contracts that aren't `DepositManager`, resulting in a lower reference rate that can cause repricing of `rswETH` to fail.

The `_referenceOnChainRate()` function does not take into account the `ETH` balances of each `StakerProxy` and their `EigenPods`, as well as unclaimed delayed withdrawals in `DelayedWithdrawalRouter`. This means that the reference onchain rate will be inaccurate and report a lower rate if a Beacon Chain or EigenLayer withdrawal has been processed.

If the reference onchain rate differs by an amount greater than `maximumReferencePriceDiffPercentage`, then the `rswETH` rate repricing will fail due to the following check in the `_checkReferencePriceDiff()` function:

```
uint256 referencePriceDiff = _absolute(_newRswETHToETHRate, _referenceRate);

uint256 maximumReferencePriceDiff = (_newRswETHToETHRate *
    _cachedMaximumReferencePriceDiffPercentage) / 1 ether;

if (referencePriceDiff > maximumReferencePriceDiff) {
    revert ReferencePriceDiffTooHigh(
    referencePriceDiff,
    _cachedMaximumReferencePriceDiffPercentage
    );
}
```

This issue has a low likelihood of occurring as the size of withdrawals are small compared to the total amount of `ETH` in the protocol and hence the revert case is unlikely to trigger in practice.

Recommendations

Consider fetching the `ETH` balances of these contracts in `_referenceOnChainRate()`.

However, this may not be economical given the gas costs associated with checking the balances of a large number of `StakerProxy` and `EigenPod` contracts.

Alternatively, these `ETH` balances can be separated from the `DepositManager` balance and `_referenceOnChainRate()` can use values from the provided snapshot instead of fetching them onchain.

Resolution

The `_referenceOnChainRate()` function no longer gets ETH balances onchain and now trusts this data from the offchain bot's snapshot.

This issue has been addressed at commit e7d96c6.

SWL3-07	Existing Depositors Can Block Validator Deployment		
Asset	`DepositManager.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Due to conditions imposed on the `ETH` balance during calls to `setupValidators()` and `transferETHForEigenLayerDeposits()`, it is possible for another user's withdrawal to prevent Swell's Bot from adding new validators.

When deploying new validators, there are checks in both functions ensuring there is sufficient `ETH` to cover the amount needed for the validators as well as any pending withdrawals:

DepositManager::setupValidators()

```
if (address(this).balance < _pubKeys.length * DEPOSIT_AMOUNT + exitingETH) {
    revert InsufficientETHBalance();
}
```

DepositManager::transferETHForEigenLayerDeposits()

```
if (address(this).balance < amount + exitingETH) {
    revert InsufficientETHBalance();
}
```

If an existing depositor requests to start a withdrawal by calling `rswEXIT.createWithdrawRequest()` just before a call to `setupValidators()` or `transferETHForEigenLayerDeposits()`, the latter call can revert unexpectedly due to not having enough ETH to cover the amount needed for the validators and any pending withdrawals.

The impact and likelihood of this issue are both rated as low as the Swell Network team have communicated that they will make use of Flashbots Protect for broadcasting Bot operations. This means such transactions will not be visible in the mempool and, in the event that they would revert, would not be published which prevents any wasted gas fees.

Recommendations

Consider truncating the `_pubKeys` array inside both functions such that only enough validators are staked into instead of reverting. An example code snippet is provided below as reference:

```
maxValidatorsToStake = (address(this).balance - exitingETH) / 32 ether
if (maxValidatorsToStake == 0) {
    revert InsufficientETHBalance();
} else if (_pubKeys.length > maxValidatorsToStake) {
    _pubKeys = _pubKeys[0:maxValidatorsToStake];
}
```

Alternatively, ensure that offchain countermeasures, such as FlashBots Protect, continue to be utilised and monitor for dropped transactions, so that the Bot can repeat them as necessary.

Resolution

The Swell Network team has acknowledged the issue and will continue to use FlashBots Protect to prevent this issue from occurring on mainnet.

SWL3-08	Incorrect `maximumReferencePriceDiff` Calculation		
Asset	`RepricingOracle.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The function `_checkReferencePriceDiff()` is used to check the `rswETH` to `ETH` exchange rate does not differ too greatly from the reference onchain rate. However, the `maximumReferencePriceDiff` is calculated from `_newRswETHtoETHRate` instead of `_referenceRate`.

```
559 uint256 referencePriceDiff = _absolute(_newRswETHToETHRate, _referenceRate);

561 uint256 maximumReferencePriceDiff = (_newRswETHToETHRate *
        _cachedMaximumReferencePriceDiffPercentage) / 1 ether;
```

This means that the percentage change allowed from `_referenceRate` is actually dependent on the value of `_newRswETHtoETHRate` rather than being a fixed percentage of `_referenceRate`, making it possible for the exchange rate to shrink or grow by a larger amount than intended.

Recommendations

Adjust line [**561**] to use:

```
uint256 maximumReferencePriceDiff = (_referenceRate * _cachedMaximumReferencePriceDiffPercentage) / 1 ether;
```

so that measured deviations from the `_referenceRate` are consistent.

Resolution

The `_checkReferencePriceDiff()` function now implements the new check, as recommended above, alongside the existing check.

This issue has been addressed in commit 071db22.

SWL3-09	Forced Undelegations Do Not Update `operatorToStakers` Mapping
Asset	`EigenLayerManager.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

An EigenLayer operator's `operatorToStakers` mapped value is not updated when undelegations are initiated from EigenLayer's `DelegationManager` contract.

This can occur when the operator or their delegation approver forces one of their delegated stakers to undelegate.

When this occurs, the `StakerProxy` will not be removed from the operator's `operatorToStakers` mapping unless they are delegated to and undelegated from the same operator again through `undelegateStakerFromOperator()`.

Recommendations

Consider adding a function in `EigenLayerManager` that allows the `SwellLib.EIGENLAYER_DELEGATOR` role to remove a `stakerId` from an operator's `operatorToStakers` mapping.

Resolution

The `unassignStakerFromOperator()` and `assignStakerToOperator()` functions have been added to allow the `SwellLib.EIGENLAYER_DELEGATOR` role to remove and add a `stakerId` to an operator's `operatorToStakers` mapping manually.

This issue has been addressed in commit f65c548.

SWL3-10	Specific LST Deposits Cannot Be Paused
Asset	`DepositManager.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The `depositLST()` function cannot pause deposits of specific tokens once their `exchangeRateProvider` has been set.

The `depositLST()` function whitelists tokens that are allowed to be deposited through the following check:

```
if (exchangeRateProviders[_token] == address(0)) {
    revert NoRateProviderSet();
}
```

Exchange rate providers are set through the `setExchangeRateProvider()` function. However, the function does not allow the zero address to be set as an `exchangeRateProvider` through the use of the `checkZeroAddress()` modifier. This means that deposits from specific tokens cannot be paused once their exchange rate provider has been set.

Recommendations

Consider removing the `checkZeroAddress()` modifier from `setExchangeRateProvider()` to allow setting an exchange rate provider to the zero address.

Alternatively, an exchange rate provider registry can be used that supports pausing deposits from any specific token.

Resolution

The Swell Network team has communicated that they do not currently intend to launch LSTs, hence this issue has been closed.

SWL3-11	Use of Rebasing Liquid Staking Tokens
Asset	`DepositManager.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The system supports `stETH` and `OETH` as Liquid Staking Tokens. Both of these tokens are rebasing tokens, which means they algorithmically alter the balance of users in order to maintain a 1:1 price ratio with `ETH` instead of appreciating in price.

The end result of this is that contracts are far more likely to introduce bugs or vulnerabilities when integrating these tokens, as observed with SWL3-02 .

Recommendations

Be mindful of supporting rebasing tokens and other LSTs which differ from traditional token designs such as ERC20s. Ensure that, when integrating new tokens, the team is familiarized with unique properties of each token.

Resolution

The Swell Network team has communicated that they do not currently intend to launch LSTs and will be mindful of supporting rebasing tokens, hence this issue has been closed.

SWL3-12	Admin Restricted Withdrawal System
Asset	`EigenLayerManager.sol and StakerProxy.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Currently, the only mechanism to withdraw LST tokens from the protocol must be processed by the Swell Network admin address. Withdrawal of LSTs via the `rswEXIT` token used for `ETH` withdrawals is not possible. This adds a centralized point of failure that could lock up tokens if it became inaccessible.

In addition, numerous contracts make use of functions only callable by the admin that allow removal of any ERC20 balance. While these are intended to be used only in an emergency, it is worth noting that several of these contracts natively hold ERC20 tokens as part of their lifecycle and could be drained if admin keys are compromised.

Recommendations

Make end users aware of current withdrawal limitations.

Ensure the controlling admin address is a multi-sig with a suitable number of discreet signers and prioritise the development of a trustless LST withdrawal system.

Resolution

The Swell Network team has communicated that they do not currently intend to launch LSTs, hence this issue has been closed.

SWL3-13	Liquid Staking Token Compositions
Asset	`/*`
Status	**Closed:** See Resolution
Rating	Informational

Description

By design, `rswETH` supports multiple different LSTs run by different groups and organisations. Swell Network should monitor and react to changes in their operational designs to avoid magnifying risk.

For example, `OETH` is backed by other LSTs, some of which are also natively supported by `rswETH` as collateral. It is possible `rswETH` could become overly reliant on a single LST by this doubled exposure.

Other changes that would need monitoring are the economic decisions of supported LSTs and structural changes, such as which consensus and execution clients node operators use, as these may cause bugs or centralisation issues, which could lead to mass slashing or other economic impacts to `rswETH`.

Recommendations

Swell Network should be aware of the risks posed by each supported LST and plan accordingly to track changes made for each.

Caution is advised when adopting new LSTs or continuing to support LSTs which make major design changes. Being an active steering contributor of each LST community is recommended.

Other actions could be taken to reduce risk posed by individual LSTs, such as `rswETH` minting caps per LST.

Resolution

The Swell Network team has communicated that they do not currently intend to launch LSTs and will pay attention to specific LST risks when updating the codebase, hence this issue has been closed.

SWL3-14	Miscellaneous General Comments
Asset	All contracts
Status	**Closed:** See Resolution
Rating	Informational

Description

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

1. **Misleading Natspec Comments**

   Related Asset(s): RepricingOracle.sol

   Some of the Natspec comments are misleading and should be corrected. For example on lines [**464-465**] it states:

   ** @param _state The state of the snapshot that will be used in repricing*
   ** @param _withdrawState The withdraw state of the snapshot that will be used in repricing*

   These statements are inaccurate as some of the variables used in repricing are fetched from onchain sources which overwrite the relevant fields in `_state`.

   Review the Natspec comments mentioned and improve their accuracy.

2. **Magic Numbers**

   Related Asset(s): AnkrETHRateProvider.sol, RswETH.sol

   The `AnkrETHRateProvider` contract makes use of `1e18 * 1e18` on line [**38**]. Using magic numbers should be discouraged to provide reader clarity. Also noted was: line [**298**] of `RswETH.sol`

   Replace magic numbers with named constants.

3. **Misleading Variable Names**

   Related Asset(s): ETHxRateProvider.sol, DepositManager.sol

   There are instances where misleading variable names are used:

   (a) The Rate Provider for the `ETHx` token is supplied by the `StaderOracle` contract yet the contract variable is named `ETHx`. This could make the underlying contract unclear given other Rate Providers query the token directly. It is recommended to change the `ETHx` variable name to something that is easier to identify such as `ETHxStaderOracle`.

   (b) The beacon chain deposit contract's deposit root is checked to prevent frontrunning deposits. However, the input parameter is named `_depositDataRoot`, which suggests that it's the Merkleisation of the specific `DepositData` SSZ container, as opposed to the list of all deposits. This occurs in `stakeOnEigenLayer()` in `EigenLayerManager`, and `setupValidators()` and `transferETHForEigenLayerDeposits()` in `DepositManager`. It is recommended to change the parameter name to `_depositRoot`.

4. **Adopt More Defensive Price Feed For `osETH`**

   Stakewise have recently added a `latestAnswer()` function to their `PriceFeed.sol` contract that Swell Network uses as the price feed for `osETH`. This function contains an additional boundary check not included in the currently used `getRate()` function. The Swell Network team should consider adopting this newer function.

5. **Incorrect Use of Fixed Point Math Library**

   Related Asset(s): RswETH.sol

   The `reprice()` function incorrectly uses the `UD60x18` library twice, as the denominators are not scaled before wrapped into the `UD60x18` type.

```
UD60x18 rewardsPerValidator = wrap(nodeOperatorRewards).div(
    wrap(totalActiveValidators)
);
...
    uint256 operatorsRewardShare = rewardsPerValidator
        .mul(wrap(operatorActiveValidators))
        .unwrap();
...
```

   This results in `rewardsPerValidator` being `1e18` times higher than intended. However, `operatorActiveValidators` is also not scaled in the second calculation, so the result `operatorsRewardShare` is still correct.

   Consider scaling both `totalActiveValidators` and `operatorActiveValidators` by `1e18`.

   However, using the unscaled versions of these values provides more precise results, as there is greater decimal precision in `rewardsPerValidator`. It is extremely unlikely for `rewardsPerValidator` to overflow as `nodeOperatorRewards` is bounded by the total supply of `ETH`. Alternatively, add inline comments to the code explicitly explaining the lack of scaling and/or rename the variables to be more descriptive.

Recommendations

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

Resolution

Code comments and variable names have been updated for accuracy in commits 21806d9 and 3379718.

Other miscellaneous comments have been acknowledged.

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.

```
Ran 1 test for test/tests-local/RepricingOracle.t.sol:RepricingOracleTest
[PASS] test_referenceOnChainRate_executionLayerBal_Vuln2() (gas: 7884353)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 112.17ms (7.77ms CPU time)

Ran 3 tests for test/tests-local/RswETH.t.sol:RswETHTest
[PASS] test_deposit() (gas: 231)
[PASS] test_rswETHToETHRate_initial() (gas: 17969)
[PASS] test_withdrawERC20() (gas: 267459)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 117.62ms (890.64µs CPU time)

Ran 8 tests for test/tests-local/NodeOperatorRegistry.t.sol:NodeOperatorRegistryTest
[PASS] testFuzz_parsePubKeyToString(bytes) (runs: 1004, µ: 44820, ~: 38984)
[PASS] test_addNewValidatorDetails() (gas: 1317865)
[SKIP] test_addNewValidatorDetails_frontrunStealRewards_Vuln() (gas: 0)
[PASS] test_addOperator() (gas: 175096)
[PASS] test_deleteActiveValidators() (gas: 4874679)
[PASS] test_deletePendingValidators() (gas: 4506558)
[PASS] test_getNextValidatorDetails() (gas: 7070636)
[PASS] test_usePubKeysForValidatorSetup() (gas: 3532922)
Suite result: ok. 7 passed; 0 failed; 1 skipped; finished in 249.01ms (171.07ms CPU time)

Ran 4 tests for test/tests-fork/RswETH.fork.t.sol:RswETHForkTest
[PASS] testFuzz_deposit(uint256) (runs: 1004, µ: 155684, ~: 155810)
[PASS] test_deposit_Multi_Fuzz(uint256[5],uint256[5]) (runs: 1004, µ: 450068, ~: 458012)
[PASS] test_reprice() (gas: 8007458)
[PASS] test_rswETHToETHRate_exchangeRateChanges() (gas: 5852796)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 2.16s (1.06s CPU time)

Ran 7 tests for test/tests-fork/RswEXIT.fork.t.sol:RswEXITForkTest
[PASS] testFuzz_getProcessedRateForTokenId(uint256,uint256,uint256[],uint256[]) (runs: 1003, µ: 427372, ~: 409279)
[PASS] test_createWithdrawRequest() (gas: 384693)
[PASS] test_createWithdrawRequest_Fuzz(uint256[5]) (runs: 1004, µ: 776358, ~: 675072)
[PASS] test_finalizeWithdrawals() (gas: 583592)
[PASS] test_finalizeWithdrawals_Fuzz(uint256[5]) (runs: 1004, µ: 935507, ~: 823529)
[PASS] test_processWithdrawals() (gas: 545836)
[PASS] test_processWithdrawals_Fuzz(uint256[5]) (runs: 1004, µ: 859744, ~: 767673)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 2.40s (3.06s CPU time)

Ran 6 tests for test/tests-fork/DepositManager.fork.t.sol:DepositManagerForkTest
[PASS] test_depositLST() (gas: 726536)
[SKIP] test_originETH_notRebasing_Vuln() (gas: 0)
[PASS] test_setupValidators() (gas: 1723676)
[SKIP] test_setupValidators_withdrawRequestDoS_Vuln() (gas: 0)
[SKIP] test_transferETHForEigenLayerDeposits_DoS_Vuln() (gas: 0)
[PASS] test_withdrawERC20() (gas: 325662)
Suite result: ok. 3 passed; 0 failed; 3 skipped; finished in 3.24s (20.17ms CPU time)

Ran 24 tests for test/tests-fork/EigenLayerManager.fork.t.sol:EigenLayerManagerForkTest
[PASS] test_batchDelegateToWithSignature() (gas: 621180)
[PASS] test_batchUndelegateStakerFromOperator() (gas: 460059)
[PASS] test_batchWithdrawERC20() (gas: 719413)
[PASS] test_claimDelayedWithdrawals() (gas: 2926703)
[PASS] test_completeQueuedWithdrawal() (gas: 1257800)
[PASS] test_createStakerAndPod() (gas: 7472329)
[PASS] test_delegateToWithSignature() (gas: 239410)
[PASS] test_depositIntoEigenLayerStrategy() (gas: 838088)
[PASS] test_depositIntoEigenLayerStrategy_stETH(uint256) (runs: 1004, µ: 512624, ~: 512302)
[PASS] test_getDelegatedStakers() (gas: 2945352)
[PASS] test_isValidStaker() (gas: 33829)
[PASS] test_queueWithdrawals() (gas: 1207166)
[PASS] test_registerStakerProxyImplementation() (gas: 191994)
```

```
[PASS] test_setAdminSigner() (gas: 175856)
[PASS] test_setDelayedWithdrawalRouter() (gas: 175633)
[PASS] test_setDelegationManager() (gas: 175482)
[PASS] test_setEigenLayerStrategy() (gas: 3057124)
[SKIP] test_setEigenLayerStrategy_wstETHUnsupported_Vuln() (gas: 0)
[PASS] test_setStrategyManager() (gas: 175427)
[PASS] test_stakeOnEigenLayer() (gas: 820761)
[PASS] test_undelegateStakerFromOperator() (gas: 352033)
[PASS] test_upgradeStakerProxy() (gas: 2749573)
[PASS] test_verifyAndProcessWithdrawals() (gas: 2743885)
[PASS] test_verifyPodWithdrawalCredentials() (gas: 2144878)
Suite result: ok. 23 passed; 0 failed; 1 skipped; finished in 3.24s (1.93s CPU time)

Ran 5 tests for test/tests-fork/StakerProxy.fork.t.sol:StakerProxyForkTest
[PASS] test_generateWithdrawalCredentialsForEigenpod() (gas: 27005)
[PASS] test_implementation() (gas: 24353)
[PASS] test_sendFundsToDepositManager() (gas: 1002604)
[PASS] test_sendTokenBalanceToDepositManager() (gas: 1017073)
[PASS] test_sendTokenBalanceToDepositManager_Fuzz(uint256,uint256) (runs: 1004, μ: 275933, ~: 309502)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 3.24s (1.10s CPU time)

Ran 4 tests for test/tests-fork/RepricingOracle.fork.t.sol:RepricingOracleForkTest
[PASS] testFuzz_submitSnapshot(uint256) (runs: 1004, μ: 858432, ~: 858464)
[PASS] test_referenceOnChainRate_executionLayerBal() (gas: 607296)
[PASS] test_submitSnapshot_penalizedValidator() (gas: 1952422)
[SKIP] test_submitSnapshot_tokenStrategyBalances_Vuln() (gas: 0)
Suite result: ok. 3 passed; 0 failed; 1 skipped; finished in 4.09s (2.71s CPU time)
```

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.

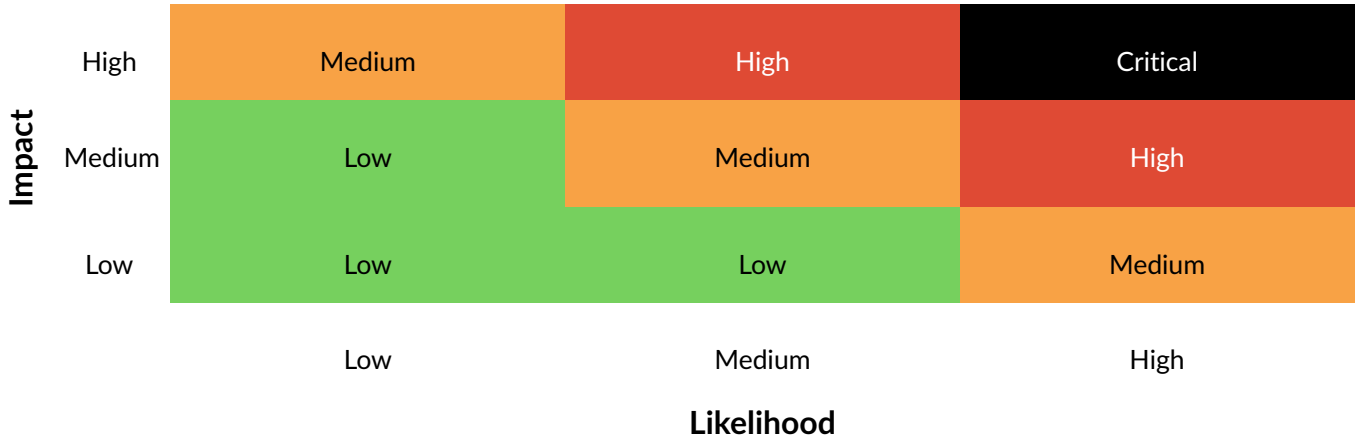

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].