CODESPECT

Swell Repricing Update

SECURITY ASSESSMENT REPORT

17 April, 2025

Prepared for

Swell

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the security assessment conducted by CODESPECT for the smart contracts of Swell. Swell is a non-custodial staking protocol offering liquid staking and restaking tokens.

This audit focuses on a minor update to the repricing oracle, prompted by the EigenLayer slashing update. The oracle now accounts for potential slashing events within the EigenLayer protocol and propagates them to the repricing mechanism of the `rswETH` token.

The audit was performed using:

a) Manual analysis of the codebase.

CODESPECT found 1 point of attention, classified as `Medium`. All of the issues are summarised in Table 2.

Organization of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the update of the Repricing oracle.
- **Section 7** presents the issues.
- **Section 7** discusses the documentation provided by the client for this audit.
- **Section 8** presents the compilation and tests.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
Medium	0	1	0
Total	**0**	**1**	**0**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	Swell
Type of Project	Repricing Oracle Update
Duration of Engagement	2 Days
Duration of Fix Review Phase	1 Days
Draft Report	April 16, 2025
Final Report	April 17, 2025
Repository	v3-contracts-lrt
Commit (Audit)	c1c7cde2e8af6fdc68116ee7d236f45f17d9487a
Commit (Final)	b7a2975d2a3e4a946d506397476ec755b30e97b3
Documentation Assessment	High
Test Suite Assessment	Not Applicable

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	**LoC**
1	Repricing.sol	54
2	RepricingOracle.sol	553
	Total	**607**

Scope information

Only the code changes outlined in PR-109 were considered within the scope of this audit. Specifically, the assessment focused solely on the relationship between the scoped files and the EigenLayer update.

5.2 Findings Overview

	Finding	**Severity**	**Update**
1	Double Accounting for Slashing	Medium	Fixed

6 Swell Update Information

Swell has introduced updates to its `RepricingOracle` contract and `Repricing` library to properly handle slashing events that may occur within the EigenLayer protocol.

Two new variables have been added to the `UpgradeableRepriceSnapshot` struct:

```
struct UpgradeableRepriceSnapshot {
  RepriceSnapshotMeta meta
  RepriceSnapshotState state;
  uint256 rewardsPayableForFees;
  WithdrawSnapshotState withdrawState;
  uint256 totalSlashedEigenLayer; // Total ETH that has been slashed from Eigenlayer (monotonically increasing)
  uint256 totalSlashedBurnedEigenLayer; // Eigenlayer slashed ETH that has been permanently removed (monotonically
   →  increasing)
}
```

As the comments indicate, `totalSlashedEigenLayer` represents the total amount of ETH slashed from EigenLayer, including both beacon chain and operator-related slashing events. Once the slashing is propagated to EigenPods and the corresponding ETH is permanently burned, the `totalSlashedBurnedEigenLayer` is updated to reflect that the slashing has already been accounted for within `state.balances`.

These two variables are closely related and factor into the repricing logic as part of the protocol's liabilities. Specifically, they contribute to the computation of `lockedSlashed`, which represents ETH that has been slashed but not yet burned, and is thus still locked:

```
function liabilities(
  uint256 _exitingETH,
  uint256 _totalSlashedEigenLayer,
  uint256 _totalSlashedBurnedEigenLayer
) internal pure returns (uint256) {
  require(_totalSlashedBurnedEigenLayer <= _totalSlashedEigenLayer, "Burned slashed exceeds total slashed");

  uint256 lockedSlashed = _totalSlashedEigenLayer - _totalSlashedBurnedEigenLayer;
  return _exitingETH + lockedSlashed;
}
```

As shown above, `totalSlashedBurnedEigenLayer` is a subset of `totalSlashedEigenLayer`, and both values must increase monotonically. This invariant is enforced in the `_assertRepricingSnapshotValidity` function:

```
function _assertRepricingSnapshotValidity(
  UpgradeableRepriceSnapshot memory _snapshot,
  ...
) internal view {
  // ...
  if (_snapshot.totalSlashedEigenLayer < upgradeableRepriceSnapshot.totalSlashedEigenLayer) {
    revert TotalSlashedMustMonotonicallyIncrease();
  }

  if (_snapshot.totalSlashedBurnedEigenLayer < upgradeableRepriceSnapshot.totalSlashedBurnedEigenLayer) {
    revert TotalSlashedBurnedMustMonotonicallyIncrease();
  }

  if (_snapshot.totalSlashedBurnedEigenLayer > _snapshot.totalSlashedEigenLayer) {
    revert TotalSlashedBurnedExceedsTotalSlashed();
  }
  // ...
}
```

Since repricing is a highly centralised process managed by Swell, the protocol bears full responsibility for correctly reporting these values. It must actively monitor all slashing events across the beacon chain and the EigenLayer, ensuring they are properly reflected in the asset accounting. In particular, `totalSlashedEigenLayer` and `totalSlashedBurnedEigenLayer` must be accurately maintained to avoid inconsistencies or double-counting in slashing scenarios. Beacon chain slashing events are especially sensitive, as they immediately change validator balances.

7 Issues

7.1 [Medium] Double Accounting for Slashing

File(s): `RepricingOracle.sol`

Description: The recent update to the `RepricingOracle` contract introduced slashing accounting related to EigenLayer. Swell added two new variables that are part of the snapshot used for updating the price of `rswETH`:

- `totalSlashedEigenLayer` – represents the total ETH slashed from EigenLayer;
- `totalSlashedBurnedEigenLayer` – represents the portion of slashed ETH that has been permanently removed (burned). Once this variable is updated, the "slashing" is propagated to `balances.executionLayer`, which tracks ETH across the protocol, including EigenPods;

A potential issue arises when slashing is double-accounted — particularly in scenarios where a slashing event has occurred, but the ETH has not yet been burned on the EigenLayer side. The root cause is that the `reserveAsset()` function, which reflects all ETH under Swell's control, immediately accounts for Beacon Chain slashing (as it tracks the BC state). At the same time, `totalSlashedEigenLayer` may also be incremented, even though `totalSlashedBurnedEigenLayer` has not yet been updated.

In such cases, slashing may be counted twice, especially when calculating `preRewardETHReserves` and `unclampedRewardsPayableForFees`:

```
function _referenceOnChainRate(...) internal view returns (uint256) {
    // ...
    unclampedRewardsPayableForFees =
      // @audit reserve change containing BC slashing already
      reserveAssetsChange -
      int256(ethDepositsChange) +
      int256(ethExitedChange) -
      // @audit Slashed ETH at Eigen
      int256(totalSlashedEigenLayerChange) +
      // @audit Slashed ETH at Eigen which was burned (It does not exist in EigenPod.)
      int256(totalSlashedBurnedEigenLayerChange);
    // ...
}


function handleReprice(...) internal {
    // ...
    uint256 preRewardETHReserves = reserveAssets - liabilities - newETHRewards;
    // ...
}
```

Note: The `liabilities` in the snippet above contains the sum of `lockedShares` (`totalSlashedEigenLayerChange - totalSlashedBurnedEigenLayerChange`) and `exitingETH`.

Additionally, EigenLayer has reported a `DoubleSlashingEdgeCase`, which should be considered. Depending on the order of EigenPod actions during simultaneous BC and AVS slashing, the amount of withdrawable ETH for users can vary.

Impact: Incorrect pricing and potential loss of value for `rswETH`.

Recommendation(s): Consider changing the repricing logic to handle edge cases where slashed shares on EigenLayer are not yet burned at the time of repricing.

Status: Fixed

Update from Swell: Resolved in commit b7a2975d2a3e4a946d506397476ec755b30e97b3.

We've implemented a solution to properly handle dual slashing scenarios between the Beacon Chain and EigenLayer:

1. Introduced a new accounting variable (`totalDualSlashedEigenLayer`) and modified the liability calculation to reduce the effective liability when ETH is slashed in both systems.
2. Added constraints to ensure accounting accuracy while maintaining flexibility for edge cases outlined in EigenLayer's dual slashing documentation.

Update from CODESPECT: Swell introduced a new variable to address the drafted issue. However, due to the centralisation of the process, Swell is responsible for accurately reporting these values.

8 Evaluation of Provided Documentation

The Swell documentation was provided in the form of the PR-109 description, which clearly outlined the changes made to the codebase. Throughout the evaluation process, the Swell team remained consistently available and responsive, promptly addressing all questions raised by CODESPECT.

9 Test Suite Evaluation

Not applicable in this case, as the evaluation of the test suite was not performed due to the nature, duration, and scope of the audit.