Swell Network

Barracuda Update (Withdrawals)

Smart Contract Security Assessment

Version: 2.1

February, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the updated Swell Network smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contracts, 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 changes made to 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 contract changes in scope.

Overview

The Barracuda Upgrade is the next significant upgrade to the Swell Network protocol following the Seawolf Upgrade launched in April 2023.

Barracuda's primary focus is enabling protocol withdrawals allowing users to unstake swETH into ETH at the primary repricing rate. This release also attempts to strengthen the security stance of the protocol by creating more granular roles and adding in slashing safety rules.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to contract files that were introduced or modified in PR-60, specifically:

- `contracts/implementations/AccessControlManager.sol`
- `contracts/implementations/DepositManager.sol`
- `contracts/implementations/RepricingOracle.sol`
- `contracts/implementations/swETH.sol`
- `contracts/implementations/swEXIT.sol`
- `contracts/libraries/Repricing.sol`

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 reopository at commit a95ea79.

Retesting activities were performed on commit 262db73.

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:

- Foundry: `https://github.com/foundry-rs/foundry`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`
- Solstat: `https://github.com/0xKitsune/solstat`
- Halmos: `https://github.com/a16z/halmos`
- Mythril: `https://github.com/ConsenSys/mythril`
- Aderyn: `https://github.com/Cyfrin/aderyn`

Output for these automated tools is available upon request.

Furthermore, as the codebase contained implementations of both ERC20 and ERC721 tokens, compliance against these standards was also assessed.

Coverage Limitations

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

Findings Summary

The testing team identified a total of 5 issues during this assessment. Categorised by their severity:

- Low: 1 issue.
- Informational: 4 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Swell Network smart contract updates in scope. 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
SBAR-01	Snapshot Withdraw State is Not Checked With Onchain Values	Low	Closed
SBAR-02	Inconsistent Timekeeping	Informational	Closed
SBAR-03	Unsafe `transferFrom()` Function	Informational	Resolved
SBAR-04	`BOT` role to `RepricingOracle` contract	Informational	Resolved
SBAR-05	Miscellaneous General Comments	Informational	Resolved

SBAR-01	Snapshot Withdraw State is Not Checked With Onchain Values		
Asset	`contracts/implementations/RepricingOracle.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Unlike the different snapshot values, there is no validation for the snapshot withdraw state with the actual withdraw values in `swEXIT`.

When submitting a reprice snapshot via `submitSnapshot()` or `submitSnapshotV2()`, there are different sanity checks to ensure the validity of the snapshot. The checks are done in the function `_assertRepricingSnapshotValidity()` and `swETH.reprice()`.

Despite that the `snapshot.withdrawState.exitingETH` is used to calculate the `preRewardETHReserves` in line [**229**], this value is not checked with the actual `exitingETH` in `swEXIT`.A significant difference between the onchain and the snapshot value may cause inconsistent rate between `swETH` and `ETH`.

Recommendations

Check the Snapshot Withdraw state with the the onchain values in `swEXIT`.

Resolution

Various checks have been implemented in commit 8056358 for `submitSnapshotV2()` to validate the `withdrawState` before calling `swEXIT.processWithdrawals()`.

Additional checks and tolerances have also been added in PR-79 as per further recommendations.

SBAR-02	Inconsistent Timekeeping
Asset	`contracts/implementations/swETH.sol`, `contracts/implementations/RepricingOracle.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Various parts of the codebase rely on knowledge of the duration since certain events; however, these durations are denominated in different units in their respective locations. Specifically:

- **`implementations/swETH.sol` on line [222]** - the number of seconds that have elapsed since the Unix epoch (i.e., Unix time) is used to calculate the time since the most recent repricing occurred.
- **`implementations/RepricingOracle.sol` on line [314] and line [315]** - the number of blocks that have been confirmed is used to calculate the time since the most recent snapshot was taken.

Recommendations

Decide on one method of timekeeping and use it exclusively throughout the entirety of the codebase. The reasons for this are:

- **Reduction of cognitive complexity** - a consistent timekeeping method greatly simplifies the ease with which various properties of the entire system can be reasoned about.
- **Increased chain portability** - different blockchain networks, while EVM-compatible, provide wildly different timekeeping guarantees than Ethereum Mainnet. If Swell should ever decide to deploy on alternative blockchains, mixed-mode timekeeping could prove troublesome.

Resolution

The development team have acknowledged this issue.

SBAR-03	Unsafe `transferFrom()` Function
Asset	`contracts/implementations/swEXIT.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The function `createWithdrawRequest()` in `swEXIT` uses the unsafe `ERC-20` operation `transferFrom()`.

The `ERC-20` functions may not behave as expected and could return values that are not always meaningful, particularly for tokens that do not closely follow the `ERC-20` standard, such as `USDT`.

Recommendations

Consider using OpenZeppelin's `SafeERC20` library's `safeTransferFrom()`.

Resolution

The recommendation has been implemented in commit 172a4ca.

`SafeERC20` library is now used for token transfers.

SBAR-04	`BOT` role to `RepricingOracle` contract
Asset	`contracts/implementations/RepricingOracle.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The access control on both `swEXIT.processWithdrawals()` and `RepricingOracle.submitSnapshotV2()` check for the `SwellLib.BOT` role via `checkRole(SwellLib.BOT)`, even though the latter function calls the former.

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

Recommendations

Consider creating a specific role for the function `swEXIT.processWithdrawals()` and granting it to the `RepricingOracle` contract, and any other EOA address that would process withdrawals when snapshotting is not intended.

This would also mean that only the bot has the `SwellLib.BOT` role.

Resolution

The recommendation has been implemented in commit 0597547.

The development team has created a dedicated role for calling `processWithdrawals()`. This role will only be granted to `RepricingOracle` contract.

SBAR-05	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

1. **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.
2. **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.
3. **Values Could Be Constants**
   Related Asset(s): DepositManager.sol
   (a) `DepositContract` from line [**65**] is a hard coded value and could be a constant.
   (b) `depositAmount` from line [**101**] is a hard coded value and could be a constant.
4. **Unused Import**
   Related Asset(s): RepricingOracle.sol, swEXIT.sol
   Consider removing the unused import of `IswETH` in `RepricingOracle` and the unused import of `UD60x18` in `swEXIT`.

Recommendations

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

Resolution

The development team has addressed the relevant findings in commits 6a1089b, 4d5f872 and 4993354.

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 7 tests for test/DepositManager.t.sol:DepositManagerTest
[PASS] test_initialize() (gas: 23152)
[PASS] test_setupValidators() (gas: 395485)
[PASS] test_setupValidators_BotMethodsPaused() (gas: 153477)
[PASS] test_setupValidators_InsufficientETHBalance() (gas: 194435)
[PASS] test_setupValidators_InvalidDepositDataRoot() (gas: 141726)
[PASS] test_setupValidators_wrong_caller() (gas: 195055)
[PASS] test_transferETHForWithdrawRequests() (gas: 31781)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 46.26ms

Running 17 tests for test/RepricingOracle.t.sol:RepricingOracleTest
[PASS] test_assertRepricingSnapshotValidity_balance_diff_too_high() (gas: 82288)
[PASS] test_assertRepricingSnapshotValidity_block_did_not_increase() (gas: 517030)
[PASS] test_assertRepricingSnapshotValidity_block_too_high() (gas: 73981)
[PASS] test_assertRepricingSnapshotValidity_mismatch_last_block() (gas: 73906)
[PASS] test_assertRepricingSnapshotValidity_old_block_number() (gas: 76362)
[PASS] test_assertRepricingSnapshotValidity_round_data_in_stale() (gas: 84672)
[PASS] test_fallback_call_non_existant_fucntion() (gas: 13422)
[PASS] test_fallback_call_send_ether() (gas: 23243)
[PASS] test_initialize() (gas: 23363)
[PASS] test_setExternalV3ReservesPoROracleAddress() (gas: 218398)
[PASS] test_setMaximumRepriceBlockAtSnapshotStaleness() (gas: 41985)
[PASS] test_setMaximumRepriceV3ReservesExternalPoRDiffPercentage() (gas: 41971)
[PASS] test_submitSnapshot() (gas: 527364)
[PASS] test_submitSnapshotV2() (gas: 529658)
[PASS] test_submitSnapshot_BotMethodsPaused() (gas: 100023)
[PASS] test_submitSnapshot_twice_to_trigger_rewards_event() (gas: 654913)
[PASS] test_submitSnapshot_wrong_caller() (gas: 143539)
Test result: ok. 17 passed; 0 failed; 0 skipped; finished in 28.58ms

Running 1 test for test/Repricing.t.sol:RepricingTest
[PASS] testFuzz_reserveAssets_sane_values(uint256,uint256,uint256,uint256) (runs: 1000, u: 5150, ~: 5150)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 286.82ms

Running 28 tests for test/swETH.t.sol:swETHTest
[PASS] test_Fuzz_deposit_FuzzedValue(uint256) (runs: 1000, u: 149953, ~: 149953)
[PASS] test_RevertWhen_burn_ZeroAmount() (gas: 18165)
[PASS] test_RevertWhen_deposit_HappyPath_OneDepositer_CoreMethodsPaused() (gas: 54689)
[PASS] test_RevertWhen_deposit_HappyPath_OneDepositer_ZeroAmount() (gas: 42853)
[PASS] test_RevertWhen_reprice_botMethodsPaused() (gas: 2115564)
[PASS] test_RevertWhen_reprice_nonRepricer() (gas: 2159097)
[PASS] test_RevertWhen_reprice_tooSoon() (gas: 2250905)
[PASS] test_RevertWhen_reprice_zeroReserves() (gas: 2104075)
[PASS] test_RevertWhen_reprice_zeroSupply() (gas: 2104067)
[PASS] test_RevertWhen_setMaximumRepriceDifferencePercentage_NonPLATFORM_ADMIN_ROLECaller() (gas: 92419)
[PASS] test_RevertWhen_setMaximumRepriceswETHDifferencePercentage_NonPLATFORM_ADMIN_ROLECaller() (gas: 92484)
[PASS] test_RevertWhen_setMinimumRepriceTime_NonPLATFORM_ADMIN_ROLECaller() (gas: 92418)
[PASS] test_RevertWhen_setNodeOperatorRewardPercentage_NonPLATFORM_ADMIN_ROLECaller() (gas: 96129)
[PASS] test_RevertWhen_setNodeOperatorRewardPercentage_Overflow() (gas: 37199)
[PASS] test_RevertWhen_setSwellTreasuryRewardPercentage_NonPLATFORM_ADMIN_ROLECaller() (gas: 96152)
[PASS] test_RevertWhen_setSwellTreasuryRewardPercentage_Overflow() (gas: 37332)
[PASS] test_depositWithReferral_HappyPath_OneDepositer() (gas: 146163)
[PASS] test_deposit_HappyPath_NDepositers() (gas: 205646)
[PASS] test_deposit_HappyPath_OneDepositer() (gas: 145936)
[PASS] test_ethToethToSwETHRate_HappyPath() (gas: 15426)
[PASS] test_getRate_HappyPath() (gas: 17015)
[PASS] test_reprice_HappyPath() (gas: 2254204)
[PASS] test_setMaximumRepriceDifferencePercentage_HappyPath() (gas: 42688)
[PASS] test_setMaximumRepriceswETHDifferencePercentage_HappyPath() (gas: 42765)
[PASS] test_setMinimumRepriceTime_HappyPath() (gas: 38019)
[PASS] test_setNodeOperatorRewardPercentage_HappyPath() (gas: 46687)
```

```
[PASS] test_setSwellTreasuryRewardPercentage_HappyPath() (gas: 46644)
[PASS] test_swETHToETHRate_HappyPath() (gas: 15086)
Test result: ok. 28 passed; 0 failed; 0 skipped; finished in 386.56ms

Running 36 tests for test/AccessControlManager.t.sol:AccessControlManagerTest
[PASS] test_Deployment() (gas: 32374)
[PASS] test_Getters() (gas: 12995)
[PASS] test_RevertWhen_pauseBotMethods_AlreadyPaused() (gas: 28945)
[PASS] test_RevertWhen_pauseBotMethods_NonPauserCaller() (gas: 79599)
[PASS] test_RevertWhen_pauseCoreMethods_AlreadyPaused() (gas: 28966)
[PASS] test_RevertWhen_pauseCoreMethods_NonPauserCaller() (gas: 79556)
[PASS] test_RevertWhen_pauseOperatorMethods_AlreadyPaused() (gas: 28917)
[PASS] test_RevertWhen_pauseOperatorMethods_NonPauserCaller() (gas: 79557)
[PASS] test_RevertWhen_pauseWithdrawals_AlreadyPaused() (gas: 22627)
[PASS] test_RevertWhen_pauseWithdrawals_NonPauserCaller() (gas: 79457)
[PASS] test_RevertWhen_setDepositManager_NonAdminCaller() (gas: 79729)
[PASS] test_RevertWhen_setDepositManager_WithZeroAddress() (gas: 20634)
[PASS] test_RevertWhen_setNodeOperatorRegistry_NonAdminCaller() (gas: 79640)
[PASS] test_RevertWhen_setNodeOperatorRegistry_WithZeroAddress() (gas: 20632)
[PASS] test_RevertWhen_setSwETH_NonAdminCaller() (gas: 79786)
[PASS] test_RevertWhen_setSwETH_WithZeroAddress() (gas: 20679)
[PASS] test_RevertWhen_setSwEXIT_NonAdminCaller() (gas: 79754)
[PASS] test_RevertWhen_setSwEXIT_WithZeroAddress() (gas: 20646)
[PASS] test_RevertWhen_setSwellTreasury_NonAdminCaller() (gas: 79795)
[PASS] test_RevertWhen_setSwellTreasury_WithZeroAddress() (gas: 20655)
[PASS] test_RevertWhen_unpauseOperatorMethods_AlreadyPaused() (gas: 22703)
[PASS] test_RevertWhen_unpauseOperatorMethods_NonPauserCaller() (gas: 79569)
[PASS] test_RevertWhen_unpauseWithdrawals_AlreadyPaused() (gas: 28726)
[PASS] test_RevertWhen_unpauseWithdrawals_NonPauserCaller() (gas: 79458)
[PASS] test_lockdown_HappyPath() (gas: 43399)
[PASS] test_pauseBotMethods_HappyPath() (gas: 27988)
[PASS] test_pauseCoreMethods_HappyPath() (gas: 28009)
[PASS] test_pauseOperatorMethods_HappyPath() (gas: 27956)
[PASS] test_pauseWithdrawals_HappyPath() (gas: 32071)
[PASS] test_setDepositManager_HappyPath() (gas: 30659)
[PASS] test_setNodeOperatorRegistry_HappyPath() (gas: 30506)
[PASS] test_setSwETH_HappyPath() (gas: 30551)
[PASS] test_setSwEXIT_HappyPath() (gas: 30496)
[PASS] test_setSwellTreasury_HappyPath() (gas: 30527)
[PASS] test_unpauseOperatorMethods_HappyPath() (gas: 32113)
[PASS] test_unpauseWithdrawals_HappyPath() (gas: 27978)
Test result: ok. 36 passed; 0 failed; 0 skipped; finished in 461.36ms

Running 20 tests for test/swEXIT.t.sol:swEXITTest
[PASS] testFail_createWithdrawRequest_Maximum() (gas: 46142)
[PASS] testFail_createWithdrawRequest_Minimum() (gas: 46185)
[PASS] testFail_setBaseURI_NotAdmin(string) (runs: 1000, u: 61708, ~: 61668)
[PASS] test_createWithdrawRequest_HappyPath() (gas: 384880)
[PASS] test_createWithdrawRequest_NotWhitelisted(uint256) (runs: 1000, u: 53266, ~: 53266)
[PASS] test_createWithdrawRequest_paused(uint256) (runs: 1000, u: 34930, ~: 34930)
[PASS] test_finalizeWithdrawal_HappyPath() (gas: 2138613)
[PASS] test_finalizeWithdrawal_NoRequest(uint256) (runs: 1000, u: 46399, ~: 46399)
[PASS] test_finalizeWithdrawal_WithdrawalsPaused(uint256) (runs: 1000, u: 30579, ~: 30579)
[PASS] test_getLastTokenIdProcessed_Sanity() (gas: 15125)
[PASS] test_getProcessedRateForTokenId_Sanity(uint256) (runs: 1000, u: 15275, ~: 15362)
[PASS] test_processWithdrawals_HappyPath() (gas: 2248279)
[PASS] test_processWithdrawals_NotRepricingOracle(uint256) (runs: 1000, u: 90544, ~: 90544)
[PASS] test_setBaseURI_HappyPath(string) (runs: 1000, u: 85008, ~: 106874)
[PASS] test_setWithdrawRequestMaximum_HappyPath(uint256) (runs: 1000, u: 60595, ~: 60595)
[PASS] test_setWithdrawRequestMaximum_NotAdmin(uint256) (runs: 1000, u: 78307, ~: 78307)
[PASS] test_setWithdrawRequestMinimum_HappyPath(uint256) (runs: 1000, u: 60187, ~: 60485)
[PASS] test_setWithdrawRequestMinimum_NotAdmin(uint256) (runs: 1000, u: 78218, ~: 78218)
[PASS] test_withdrawERC20_NoBalance() (gas: 38917)
[PASS] test_withdrawERC20_NotAdmin(address) (runs: 1000, u: 61132, ~: 61132)
Test result: ok. 20 passed; 0 failed; 0 skipped; finished in 6.53s

Ran 6 test suites: 109 tests passed, 0 failed, 0 skipped (109 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.

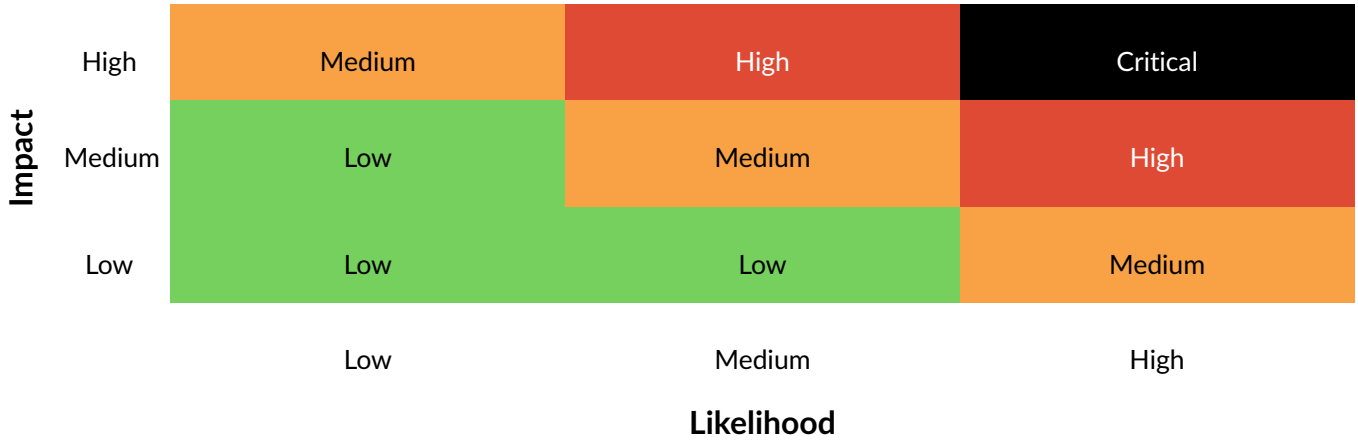

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

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].