StakeWise

OsToken Vault Escrow

Smart Contract Security Review

Version: 2.0

September, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the StakeWise smart contract changes in scope. 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 StakeWise smart contract changes 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 StakeWise smart contracts.

Overview

StakeWise is a non-custodial, decentralised staking solution that launched in early 2021. StakeWise recently underwent a major upgrade, introducing StakeWise V3, a brand new model for liquid staking allowing anyone to stake on their own terms.

StakeWise V3 acts as a white-labelled Liquid Staking Solution, allowing any node operator or DApp to launch its own liquid staking solution by leveraging the V3 architecture.

StakeWise infrastructure, combined with tailored tokenomics, aims to provide high staking yields for its users. As a liquid staking platform, users are free to un-stake at any time or utilise their staked ETH capital to earn enhanced yields throughout DeFi.

Security Assessment Summary

Scope

The scope of this time-boxed review was strictly limited to changes in PR#97 at commit dc5f9d8. The fixes to the raised issues were assessed at commit 3765d9d.

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`
- Aderyn: `https://github.com/Cyfrin/aderyn`

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 4 issues during this assessment. Categorised by their severity:

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

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the StakeWise smart contract changes 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
STW3-01	Function `processExitedAssets()` May Become An Operational Hurdle	Low	Closed
STW3-02	Minting Maximum `OsToken` Increases Liquidation Risk	Low	Closed
STW3-03	Independent Rates May Cause Inconsistency	Low	Closed
STW3-04	Miscellaneous General Comments	Informational	Resolved

STW3-01	Function `processExitedAssets()` May Become An Operational Hurdle		
Asset	`OsTokenVaultEscrow.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The function `processExitedAssets()` claims exited assets from the vault to the escrow contract. This function must be called before calling `claimExitedAssets()`, `liquidateOsToken()`, or `redeemOsToken()`.

Therefore, the call to function `processExitedAssets()` becomes an intermediate step before completing an operation. This could be a limitation or impediment in the following cases:

1. For a user, they should execute two transactions before receiving the exited assets, instead of just one. Sequentially, they have to call `processExitedAssets()` then `claimExitedAssets()`.
2. For a liquidator, the liquidation process of a loan position can be disrupted. This occurs if the call to `liquidateOsToken()` right after `processExitedAssets()` is interrupted by a call to `redeemOsToken()` targeting the same loan position which prevents finalisation of the liquidation process.

Recommendations

Consider integrating the logic in `processExitedAssets()` into `claimExitedAssets()`, `liquidateOsToken()`, and `redeemOsToken()` so that each operation can be finalised in one transaction.

Resolution

The finding has been acknowledged by the development team with the following comment:

> *"It might be useful to first call `processExitedAssets()`, then check how many assets are present by calling `getPosition()`, and afterward claim, liquidate, or redeem. The contract includes `multicall`, allowing the combination of actions into a single transaction."*

STW3-02	Minting Maximum `OsToken` Increases Liquidation Risk		
Asset	`GnoGenesisVault.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Maximum amount of `OsToken` is minted for the `receiver` when the `GnoGenesisVault.migrate()` function is called:

```
// mint max possible OsToken shares
uint256 mintOsTokenShares = Math.min(
  _calcMaxMintOsTokenShares(receiver),
  _calcMaxOsTokenShares(assets)
);
if (mintOsTokenShares > 0) {
  _mintOsToken(receiver, receiver, mintOsTokenShares, address(0));
}
```

This puts the loan position at risk of liquidation as it exercises maximum acceptable *loan-to-value* (LTV) rate by the vault.

Liquidation may occur if the health factor of the position drops slightly below the threshold, resulting in the position owner losing the underlying asset.

Recommendations

Consider adding an option for the user to determine the amount of `OsToken` to mint.

Resolution

The finding has been acknowledged by the development team with the following comment:

> *"Adding the option to override the amount of OsToken shares to mint would require an upgrade to the legacy contracts (specifically the RewardToken contract), due to the change in the function signature."*

STW3-03	Independent Rates May Cause Inconsistency		
Asset	`OsTokenVaultEscrow.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `OsTokenVaultEscrow` contract stores `liqThresholdPercent` and `liqBonusPercent` rates independently. Value changes on both variables are managed locally by calling the `OsTokenVaultEscrow.updateLiqConfig()` function.

This behaviour is in contrast to vault contracts, where the rates are parts of system configuration managed by `OsTokenConfig` contract, which is referenced by the vault contracts when needed.

As a result, there can be an inconsistency when the rates of either `OsTokenVaultEscrow` or `OsTokenConfig` are updated, but the other contract did not receive the update. For example, the rate change may cause liquidation on vaults but not on `OsTokenVaultEscrow`, or vice versa.

Recommendations

Consider implementing a single source of truth for identical rates on `OsTokenVaultEscrow` and vault contracts

Resolution

The finding has been acknowledged by the development team with the following comment:

> *"The OsTokenVaultEscrow contract only handles exiting or exited ETH, so liquidation should only occur when the unbounded ETH in the escrow approaches the osETH value that hasn't been burned. As a result, the escrow's liquidation threshold should be higher than the vault's, meaning we cannot use a single source for configuring the parameters of both the vault and the OsTokenVaultEscrow."*

STW3-04	Miscellaneous General Comments
Asset	All 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. **Data Type Inconsistency**

   Related Asset(s): OsTokenVaultEscrow.sol

   Variable `liqThresholdPercent` is declared on line [**32**] of `OsTokenVaultEscrow` as follows:

```
32 uint256 public override liqThresholdPercent;
```

   It has a similarity with another variable defined in `IOsTokenConfig`, specifically `Config.liqThresholdPercent`.

```
struct Config {
  uint128 liqBonusPercent;
  uint64 liqThresholdPercent;
  uint64 ltvPercent;
}
```

   However, both have different data types. The one on `OsTokenVaultEscrow` is `uint256` while the other is `uint64`.

   Consider replacing `uint256` with `uint64` for consistency with the existing variable.

2. **Inconsistent Versioning**

   Related Asset(s): GnoGenesisVault.sol, EthFoxVault.sol

   Two versioning inconsistencies were discovered among vault contracts. The `GnoGenesisVault` contract is marked as version 4 and `EthFoxVault` is marked as version 2, while the other contracts have version 3.

   Ensure the versioning difference is intended, or align versions accordingly.

Recommendations

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

Resolution

The suggested fixes have been applied by the development team at commit 3765d9d.

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 4 tests for test/tokens/OsTokenVaultEscrow.sigp.t.sol:OsTokenVaultEscrowTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 3042)
[PASS] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 2536)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2944)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 3064)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 9.68ms (267.29µs CPU time)

Ran 4 tests for test/vaults/modules/VaultOsToken.sigp.t.sol:VaultOsTokenTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 3130)
[PASS] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 2602)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 3032)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 3152)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 9.72ms (255.89µs CPU time)

Ran 4 tests for test/vaults/ethereum/EthErc20Vault.sigp.t.sol:EthErc20VaultTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 1346296)
[FAIL. Reason: InvalidReceivedAssets()] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 1611577)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2196090)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 1388576)
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 40.15ms (66.64ms CPU time)

Ran 4 tests for test/vaults/ethereum/EthPrivVault.sigp.t.sol:EthPrivVaultTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 1362427)
[FAIL. Reason: InvalidReceivedAssets()] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 1628231)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2180830)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 1404707)
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 45.28ms (85.83ms CPU time)

Ran 4 tests for test/vaults/ethereum/EthBlocklistErc20Vault.sigp.t.sol:EthBlocklistErc20VaultTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 1352023)
[FAIL. Reason: InvalidReceivedAssets()] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 1619933)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2203933)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 1394303)
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 52.94ms (72.77ms CPU time)

Ran 4 tests for test/vaults/ethereum/EthBlocklistVault.sigp.t.sol:EthBlocklistVaultTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 1342912)
[FAIL. Reason: InvalidReceivedAssets()] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 1608703)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2193485)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 1385192)
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 47.73ms (74.29ms CPU time)

Ran 4 tests for test/vaults/ethereum/EthPrivErc20Vault.sigp.t.sol:EthPrivErc20VaultTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 1371538)
[FAIL. Reason: InvalidReceivedAssets()] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 1639461)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2189402)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 1413818)
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 48.78ms (84.68ms CPU time)

Ran 4 tests for test/vaults/ethereum/EthVault.sigp.t.sol:EthVaultTestSigp
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 1337119)
[FAIL. Reason: InvalidReceivedAssets()] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 1600325)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2185601)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 1379399)
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 48.43ms (71.50ms CPU time)

Ran 4 tests for test/vaults/ethereum/EthGenesisVault.sigp.t.sol:EthGenesisVaultTest
[PASS] test_OsTokenVaultEscrow_liquidateOsToken() (gas: 1352026)
[FAIL. Reason: InvalidReceivedAssets()] test_OsTokenVaultEscrow_liquidateOsToken_redeemOsToken() (gas: 1615254)
[PASS] test_OsTokenVaultEscrow_processExitedAssets_claimExitedAssets() (gas: 2185613)
[PASS] test_OsTokenVaultEscrow_redeemOsToken() (gas: 1394306)
```

```
Suite result: FAILED. 3 passed; 1 failed; 0 skipped; finished in 48.34ms (59.59ms CPU time)

Ran 9 test suites in 62.19ms (351.06ms CPU time): 29 tests passed, 7 failed, 0 skipped (36 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].