STAKEWISE

StakeWise v3

Smart Contract Security Review

Version: 2.0

August, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the StakeWise v3 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 StakeWise v3 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 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-labeled 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

This review was conducted on the files hosted on the StakeWise `v3-core` repository and were originally assessed at commit 9a1ef8e, which was later updated to commit ab9c809, and finalised at commit a03e5c7. Fixes implemented for the identified findings were then assessed at commit 9c30c45.

The scope included all files in the following directories:

- `base`
- `interfaces`
- `keeper`
- `libraries`
- `vaults`
- `OsToken`

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities 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 all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used 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.

Findings Summary

The testing team identified a total of 21 issues during this assessment. Categorized by their severity:

- Critical: 2 issues.
- High: 5 issues.
- Medium: 1 issue.
- Low: 3 issues.
- Informational: 10 issues.

All these issues have been resolved/closed by the development team.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the StakeWise 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
STW-01	Deposits From V2 Pool Escrow Misinterpreted As User Deposits	Critical	Resolved
STW-02	Rewards From V2 Stakewise Not Shared With V2 Stakers	Critical	Resolved
STW-03	Users Are Charged Different Interest Rates On The Same Deposit Interval	High	Resolved
STW-04	`_harvestAssets()` Does Not Count Assets Received From `SharedMevEscrow`	High	Closed
STW-05	Migrations From V2 System Point To Wrong Contract	High	Resolved
STW-06	Legacy Rewards Initially Unavailable To Genesis Vault	High	Closed
STW-07	Known Issue `CVE-2023-34459` On OpenZeppelin `MerkleProof.sol`	High	Resolved
STW-08	Deposits From V2 Pool Escrow Can Force Functions To Revert	Medium	Resolved
STW-09	Absence Of Vault's Admin Update Function	Low	Closed
STW-10	`OsTokenConfig` Constructor Calls Incorrect `transferOwnership()`	Low	Resolved
STW-11	Excess Rewards Accounting	Low	Closed
STW-12	Potential Failing Oracles On Insufficient Minimum Oracles	Informational	Closed
STW-13	Centralisation On `OsToken` & `OsTokenConfig`	Informational	Closed
STW-14	Genesis Vault Total Asset Count Is Inaccurate	Informational	Closed
STW-15	Ability To Forcefully Send ETH To Private Vaults	Informational	Closed
STW-16	Usage of Deprecated `latestAnswer()` Interface	Informational	Resolved
STW-17	Sequential Registration Of Validators May Fail	Informational	Closed
STW-18	Potentially Confusing Indices On `_registerMultipleValidators()`	Informational	Closed
STW-19	Potentially Identical Rewards Root	Informational	Resolved
STW-20	Minting Assets Instead of Shares	Informational	Resolved
STW-21	Miscellaneous General Comments	Informational	Resolved

STW-01	Deposits From V2 Pool Escrow Misinterpreted As User Deposits		
Asset	`EthGenesisVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

Withdrawals from Stakewise V2 validators are sent to the V3 Genesis Vault via the Pool Escrow contract. These transfers are misinterpreted by the Genesis Pool which mistakes them as user deposits and so does not correctly allocate validator-earnt ETH to staked users.

This discrepancy arises from the fact that validator ETH transfers normally do not run code when being sent to a smart contract owing to partial and full withdrawals from the validator system being gas-less transactions. This works fine for the other vaults in the Stakewise system because they are all set as their own withdrawal addresses.

For the Genesis Vault this does not work as the withdrawal address from the legacy V2 system is set to the Pool Escrow contract and cannot be changed. Therefore, ETH transfers from the pool escrow contract to the Genesis Vault do trigger code to be run and the `receive()` function is called, which then invokes the normal user deposit routine.

As the pool escrow funds are treated as a normal deposit, they mint extra shares which are held by the Pool Escrow contract and cannot be redeemed. This means transferred funds do not increase the funds allocated to existing depositor shares.

Recommendations

There are several approaches that could be used to solve this issue:

- Removing the `receive()` function's call of `_deposit()` allowing the pool escrow contract to send ETH to the Genesis vault without needing to deposit. This would also mean ordinary users must use the `deposit()` function and cannot directly send ETH as a method of depositing.
- Bypassing the call to `_deposit()` in the `receive()` function if the sender is detected as the pool escrow contract.

Resolution

The issue has been fixed in PR#64.

STW-02	Rewards From V2 Stakewise Not Shared With V2 Stakers		
Asset	`EthGenesisVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The Stakewise team has highlighted that rewards from Stakewise V2 will be shared evenly between stakers in the V2 legacy pool and the Genesis Vault, but all rewards will be sent only to stakers in the Genesis Vault.

Currently, the Genesis Vault lacks logic to split rewards between legacy V2 system stakers and Genesis Vault stakers. Therefore, upon beginning the migration, all rewards from the Stakewise V2 system will be sent to the Genesis Pool and no longer allocated to any of the remaining legacy V2 system stakers as the Genesis Vault is not aware of their shares until they migrate to the V3 system.

Recommendations

Ensure code that fairly allocates rewards between the legacy system and Genesis Vault is included in the Genesis Vault contract.

Resolution

The issue has been fixed in PR#64.

STW-03	Users Are Charged Different Interest Rates On The Same Deposit Interval		
Asset	`VaultOsToken.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Users who borrow osETH can be charged different interest rates over the same time interval.

The vault system charges users a treasury fee for borrowing osETH from the Stakewise vaults. This fee is updated each time a user interacts with functions linked to osETH such as minting or burning osETH.

Due to the formula used in this update calculation, users who interact more frequently with the system will end up paying a higher interest rate. This is because of the interest charged compounding each time a user interacts with their osETH loan.

Recommendations

The formula used to calculate the interest rate should be altered to one that performs consistently regardless of how often the user interacts with the system.

One such method would be to record the increased treasury fee in a multiplicative fashion rather than additive. This would have the impact of preventing the update frequency from increasing the accrued interest.

This could be achieved by altering line [**264**] of `VaultOsToken` as follow:

```
position.shares = position.shares * cumulativeFeePerShare / position.cumulativeFeePerShare;
```

This would have to be coupled with a different approach to storing `cumulativeFeePerShare`. Currently, this is stored as value between 1 and 0 (once divided by the general denominator `_wad`). Instead, this would need to be stored as a value greater than 1. So, for example 1% which was stored as `0.01 * _wad` before would need to be stored as `1.01 * _wad`.

Resolution

The suggestion has been implemented in PR#64.

STW-04	`_harvestAssets()` Does Not Count Assets Received From `SharedMevEscrow`		
Asset	`VaultMev.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The function `_harvestAssets()`, which is called when updating a Vault's state, does not count assets harvested from the shared MEV Escrow.

As a result, harvested ETH will be received in the Vault's contract, but the total asset balance will not be updated with the correct amount.

```
41  function _harvestAssets(
      IKeeperRewards.HarvestParams calldata harvestParams
43    ) internal override returns (int256) {
      (int256 totalAssetsDelta, uint256 unlockedMevDelta) = IKeeperRewards(_keeper).harvest(
45      harvestParams
      );
47
      // SLOAD to memory
49    address _mevEscrow = mevEscrow();
      if (_mevEscrow == _sharedMevEscrow) {
51      if (unlockedMevDelta > 0) {
          // withdraw assets from shared escrow only in case reward is positive
53        ISharedMevEscrow(_mevEscrow).harvest(unlockedMevDelta);
        }
55      return totalAssetsDelta;  // @audit should include unlockedMevDelta, otherwise it won't be accounted
      }
57
      // execution rewards are always equal to what was accumulated in own MEV escrow
59    return totalAssetsDelta + int256(IOwnMevEscrow(_mevEscrow).harvest());
    }
```

Recommendations

Modify return on line [55] to also include `unlockedMevDelta`:

```
return totalAssetsDelta + unlockedMevDelta;
```

Resolution

The development team responded with the following:

"`totalAssetsDelta` already contains `unlockedMevDelta`, so there is no need to sum them and it's counted"

This indicates that the rewards received by `SharedMevEscrow` are already included in `params.reward`, in which its value is calculated offchain by Oracles.

STW-05	Migrations From V2 System Point To Wrong Contract		
Asset	`EthGenesisVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Transitions from Stakewise V2 are triggered by calling the `migrate()` function. However, the contract that has permission to call this does not contain any functions capable of calling `migrate()` meaning funds in Stakewise V2 will be unable to migrate to Stakewise V3.

Currently the `migrate()` function has access control set to only allow the `_stakedEthToken` to call it. Looking at the code for Stakewise V2's `StakedEthToken` there is no means to call `migrate`.

Recommendations

The code to call `migrate` is actually included in `RewardEthToken` from Stakewise V2. The team should ensure the right address is set when setting up the Stakewise V3 system and it is suggested to update the variable name from `_stakedEthToken` to `_rewardEthToken` to make this change clear.

Resolution

The issue has been fixed in PR#64.

STW-06	Legacy Rewards Initially Unavailable To Genesis Vault		
Asset	`EthGenesisVault.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Users who deposit to and redeem from the Genesis Vault before any new validators are added to it will not receive any rewards the Genesis Vault accrues from the legacy Stakewise V2 system.

When various actions occur in the Vaults, a call is made to `_checkHarvested()` to ensure that rewards are realised prior to users withdrawing or altering share balances. This harvest check relies on verifying the value of `rewards[vault].nonce` as a proxy for whether the vault has any validators.

However, as the Genesis Vault is linked to the legacy Stakewise V2 system, it is possible for it to generate rewards, sent from the V2 system, and still have no attached validators. In this instance, it will incorrectly report that a harvest does not need to be completed even when rewards are waiting to be harvested.

Furthermore, as the same use of `rewards[vault].nonce` is present in other areas of `KeeperRewards`, this will also result in calls to `KeeperRewards.isCollateralized()` and `KeeperRewards.canHarvest()` reporting incorrect results. This will then prevent users from queuing a withdrawal via `enterExitQueue()` or minting osETH via `mintOsToken()`.

Users may not know they have rewards waiting to claim due to the incorrect result reported by `KeeperRewards.canHarvest()` and so redeem staking funds sacrificing their claim to rewards.

Recommendations

Fixing this issue would require a rewrite of the way that Vaults detect if they have any validators. Ideally a specific variable would be created as using an existing variable is a case of semantic overload that is liable to cause mistakes in future development as seen here.

However, it is acknowledged that the issue is isolated to the Genesis Vault and changing the codebase of all vaults may be deemed excessive. An alternative resolution would be to ensure that a validator is added to the Genesis Vault directly after deployment and prior to any rewards being sent to it via the legacy Stakewise V2 system, then the nonce should be incremented to 1 and the harvest system will work as intended.

Resolution

The development team acknowledged the issue with the following comment:

"Yes, we're aware of that. The Genesis vault will become available for the users (added to UI) only after first harvest or validator registration happens."

STW-07	Known Issue `CVE-2023-34459` On OpenZeppelin `MerkleProof.sol`		
Asset	`VaultValidators.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

There is a security advisory on OpenZeppelin's `MerkleProof.sol` (CVE-2023-34459) that may affect the Merkle Root validation on `VaultValidators` contract. Function `registerValidators()` is affected by this issue due to the use of function `MerkleProof.multiProofVerifyCalldata()`, where the known vulnerable function `processMultiProofCalldata()` is called.

The exploit could allow malicious proofs to be supplied in `registerValidators()`. As a result it would be possible to register invalid validators.

Recommendations

The issue is fixed on version 4.9.2. Upgrade the OpenZeppelin library to the latest stable version.

Resolution

The library has been updated to version 4.9.2 in commit a03e5c7.

STW-08	Deposits From V2 Pool Escrow Can Force Functions To Revert		
Asset	`EthGenesisVault.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

If the Genesis Vault's asset capacity is reached or nearly full, then it is possible for user withdrawals and registering new validators to revert.

This is because these functions include a call to `_pullAssets()` which sends all awaiting funds from the pool escrow contract in Stakewise V2 to the Genesis Vault. These transfers of ETH from the pool escrow trigger the `receive()` function which then attempts to deposit the funds as with a normal deposit. This deposit will then revert the transaction if the capacity of the Genesis Vault has been reached.

As a result, it can be impossible for some users to remove their deposits from the Genesis Vault as only quantities larger than pending funds in the Pool Escrow contract will be successfully withdrawn. In addition to this, attempts to register new validators to the Genesis Vault will also fail as these functions also call `_pullAssets()`.

These functions would become callable again if the vault total assets (including those in the Pool Escrow) drop below the capacity, meaning this problem should not freeze these function calls indefinitely.

Recommendations

There are various solutions to this issue:

- The simplest fix is to not set the capacity of the Genesis Vault, then it will be set to `uint256.max` which is larger than the total ETH in existence and so prevents this issue from ever occurring.
- Funds sent from the Pool Escrow contract could be explicitly excluded from the calculations checking against asset capacity by checking if the `msg.sender` of a deposit is the pool escrow contract.

Note: Fixing STW-01 will also mitigate this issue.

Resolution

The issue has been fixed in PR#64.

STW-09	Absence Of Vault's Admin Update Function		
Asset	`EthVault.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

There are no designated functions to update or change the vault's administrator.

The absence of an admin update function prevents the contract from adapting to changing administrative requirements or transferring administrative control to a different entity. This limitation can hinder the contract's flexibility and responsiveness in managing administrative privileges.

Recommendations

Implement a dedicated function within the contract that allows authorized parties to update or change the contract's administrator.

This function should incorporate appropriate access controls and 2-step validation mechanisms to ensure that only authorised individuals can initiate the admin update process.

This could be done via OpenZeppelin's library Ownable2Step.

Resolution

The development team acknowledged the issue with the following comment:

"We've decided to not introduce this functionality for now as some of vault admins can verify themselves (e.g. node operating company). We want to avoid passing verified vaults from one admin to another."

STW-10	`OsTokenConfig` Constructor Calls Incorrect `transferOwnership()`		
Asset	`OsTokenConfig.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The constructor calls `transferOwnership()`, which starts the ownership transfer of the contract to a new account. However, it should be calling `_transferOwnership()` instead, which immediately transfers ownership of the contract to a new account.

Currently, the owner account will be the one that deploys the contract, not the `_owner` specified at construction, until that account accepts the ownership via `acceptOwnership()`.

Recommendations

To allow the specified `_owner` taking ownership immediately after construction, call `_transferOwnership()` instead.

Resolution

The recommendation has been implemented in PR#64.

STW-11	Excess Rewards Accounting		
Asset	`OsToken.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

It is possible to generate an excessive amount of rewards in `_unclaimedAssets()`.

The formula for calculating the unclaimed assets is seen in the following code snippet as:

$$avgRewardPerSecond * \_totalAssets * timeElapsed$$

```
function _unclaimedAssets() internal view returns (uint256) {
    // calculate time passed since the last update
    uint256 timeElapsed;
    unchecked {
        // cannot realistically underflow
        timeElapsed = block.timestamp - _lastUpdateTimestamp;
    }
    if (timeElapsed == 0) return 0;
    return Math.mulDiv(avgRewardPerSecond * _totalAssets, timeElapsed, _wad);
}
```

Since, `avgRewardPerSecond` is set periodically by the keeper, it is possible for a malicious user to increase the value of `_totalAssets` and therefore increase the number of rewards generated.

If a user were to deposit a significant balance into `VaultOsToken` and then call `mintOsToken()`, the value of `_totalAssets` could increase significantly.

Say a user increases the total supply of `OsTokens` by 20 fold, this would increase the amount of `_unclaimedAssets()` by 20 fold. The keeper would attempt to rectify this by calling `setAvgRewardPerSecond()` with the new rate. However, there is a delay between when `_totalAssets` is updated and when the keeper's transaction is mined to update `avgRewardPerSecond`. During this delay, rewards may be significantly overdistributed.

Recommendations

To avoid this situation consider storing the variable `rewardPerSecond` or `totalRewards` rather than the average. This would minimise the impact of drastic temporary fluctuations in `_totalAssets`.

Resolution

The concept behind this issue is that `avgRewardPerSecond` is updated periodically thus, it may use a stale value. The idea was to exploit this by minting large quantities of `OsTokens` which exploit the stale `avgRewardPerSecond`. However, this is not a valid attack vector as the user minting the `OsTokens` is accruing extra debt which must be repaid. Hence, the attacker is losing funds rather than gaining funds.

STW-12	Potential Failing Oracles On Insufficient Minimum Oracles
Asset	`KeeperRewards.sol, KeeperValidators.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The Keeper contract relies on Oracles for data verification on critical operations such as `KeeperRewards.updateRewards()` and `KeeperValidators.approveValidators()`. The Oracles provide their signatures as a way to prove that the data is correct according to those Oracles.

However, this mechanism solely relies on the Oracles. If the Oracles fail or behave maliciously and the minimum number of Oracles is too low, the submitted data may be incorrect.

Recommendations

Make sure the risk of failing or malicious Oracles is understood. Consider setting up a threshold that would identify the minimum number of oracles by following a byzantine failure model.

Resolution

The development team acknowledged the issue with the following comment:

"We will use 6 out of 11 oracles and migrate to on-chain oracles after EIP-4788."

STW-13	Centralisation On `OsToken` & `OsTokenConfig`
Asset	`OsToken.sol, OsTokenConfig.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The following cases reflect centralisation to the Owner of `OsToken` and `OsTokenConfig` contracts.

- Function `OsToken.setTreasury()` allows the owner to set the treasury address. This may be a security risk if the owner is malicious and sets the treasury address to an address that is not controlled by the DAO.
- Function `OsToken.setFeePercent()` allows the owner to set the fee percent. An increase in the fee percentage may decrease the `OsToken` holders' profits from the staking rewards distribution.
- The contract owner is able to call function `OsTokenConfig.updateConfig()` which can change the system configuration and in turn significantly impact the system, specifically `OsToken` and the health of the loan.

Recommendations

Consider utilising the DAO for a more decentralised governance.

Resolution

The development team acknowledged the issue with the following comment:

"Yes, DAO will be the owner of both `OsToken`, `OsTokenConfig`."

STW-14	Genesis Vault Total Asset Count Is Inaccurate
Asset	`EthGenesisVault.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

The quantity returned from calling `totalAssets()` can be less than the quantity given by calling `withdrawableAssets()` due to the inclusion of assets waiting to be transferred to the Genesis Vault in the Pool Escrow contract from Stakewise V2.

While no errors could be detected stemming from this, it is felt that given the name "total assets" this behaviour is misleading and could potentially cause issues in future upgrades.

Recommendations

One solution would be to include assets pending transfer to the Genesis Vault from the Pool Escrow in the total asset count. However, this could then have knock-on changes to the accounting and the development team may prefer to change the name of total assets or include comments that make its meaning clearer.

Resolution

The development team acknowledged the issue and added in-line comment in commit 7775de2.

STW-15	Ability To Forcefully Send ETH To Private Vaults
Asset	`OwnMevEscrow.sol`, `SharedMevEscrow.sol`, `EthPrivVault.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

It is possible to send ETH to private vaults either via depositing to the MEV escrow used by the vault, or by an external contract's `selfdestruct()` call.

When sending ETH to the MEV escrow, the unexpected ETH will then be sent to the vault on next `harvest()` call. The Vault's internal accounting in `_totalAssets` will also be updated accordingly.

When sending ETH via an external contract's `selfdestruct()` call, the Vault will receive ETH, but `_totalAssets` will not be updated. This would only affect accounting made via `_vaultAssets()`, which uses `address(this).balance` rather than a locally traced `_totalAssets` variable.

Note, in either case, there will be no corresponding shares minted for the sender. As such, the only impact is merely increasing the balance of the Vault's assets.

Recommendations

As there is no fix for `selfdestruct()` calls, consider modifying `_vaultAssets()` to use `_totalAssets` rather than `address(this).balance` to avoid potential accounting issues with unexpected ETH.

Resolution

The development team acknowledged the issue, added in-line comment in commit 7775de2. The following note was also included as a response:

"The recommendation is to use `_totalAssets` instead of `address(this).balance`. The `_vaultAssets` is used in `withdrawableAssets` and must return the amount of ETH the vault holds now, so `_totalAssets` won't work here."

STW-16	Usage of Deprecated `latestAnswer()` Interface
Asset	`PriceOracle.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

`PriceOracle` uses `IChainlinkAggregator` interface and implements its deprecated `latestAnswer()` function for obtaining prices, which may lead to use of stale prices by other interfacing protocols.

`latestAnswer()` has been replaced by `latestRoundData()`, which returns `roundId` alongside the price, helping to identify stale prices.

Recommendations

Implement and support use of `latestRoundData()`, returning `roundId` alongside the price.

Resolution

The recommendation has been implemented in PR#64 where `PriceOracle` was replaced with `PriceFeed`.

STW-17	Sequential Registration Of Validators May Fail
Asset	`VaultValidators.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Function `registerValidator()` checks that the validators are registered sequentially. This is done by concatenating each validator to an index (`validatorIndex` or `currentIndex`) to form a leaf in the Merkle Root to validate (line [**68**]). The `validatorIndex` is incremented by one only after the validator with the current index is registered.

This means that if one validator fails to register, the next validator cannot be registered.

Recommendations

Make sure this behaviour is expected. It is understandable that this behaviour may be desirable and a mitigation strategy would be to reset the Validators Root.

Resolution

The development team acknowledged the issue with the following comment:

"That's expected behaviour. We must ensure that the validators are registered in the same order as they're listed in deposit data."

STW-18	Potentially Confusing Indices On `_registerMultipleValidators()`
Asset	`VaultEthStaking.sol, VaultValidators.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Internal indices and input indices may not be in sync and cause the transaction to fail.

It is assumed that function `VaultEthStaking._registerMultipleValidators()` is called by `VaultValidators.registerValidators()`. Function `_registerMultipleValidators()` generates leaves for the Merkle Root validation on `VaultValidators` contract and also depositing to the `Validators Registry` contract. The leaves are generated by concatenating each validator to an index (`validatorIndex` or `currentIndex`). The `validatorIndex` is incremented by one only after the validator with the current index is registered.

This indicates that the input `uint256[] calldata indexes` may cause a revert if it is not in sync with the validators' actual indices. For example, if the input `indexes` is `[0, 1, 2, 3, 4]` but the validators' actual indices are `[2, 3, 4, 5, 6]`, then the transaction will potentially revert when `MerkleProof.multiProofVerifyCalldata()` is called, because the leaf indices are incorrect.

Recommendations

Make sure this behaviour is understood. Consider adding a check to ensure that the input `indexes` are in sync with the validators' actual indices, potentially on the UI side.

Resolution

The development team acknowledged the issue with the following comment:

"Yes, we're checking `indexes` in UI."

STW-19	Potentially Identical Rewards Root
Asset	`KeeperRewards.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

Function `updateRewards()` requires that the input `params.rewardsRoot` must not be identical to the current and previous rewards roots. However, there is no onchain mechanism to ensure that the input `params.rewardsRoot` is not identical to the last two rewards roots. Identical rewards roots can occur when all leaves in the Merkle Tree are the same. Each leaf is constructed from Vault address, reward amount, and unlocked mev reward. There is no guarantee that the three parameters are unique for each rewards update.

As a result, the updater needs to change the parameters to ensure that the rewards roots are not identical on three consecutive rewards updates.

Recommendations

Make sure this behaviour is understood. Nonces can be used to improve uniqueness of the rewards root.

Resolution

The recommendation has been implemented in PR#64 where root uniqueness check is removed.

STW-20	Minting Assets Instead of Shares
Asset	`VaultOsToken.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The use of assets instead of shares may create inaccuracy to the `mintOsToken()` operation, for example when the minter wants to maximise the LTV when borrowing.

Shares represent a portion of assets held by the Vault that belongs to the share holder. The value of shares is relatively static unless the holder deposits or redeems the shares for assets. On the other hand, the amount of assets that corresponds to a share varies whenever the Vault receives rewards or slashing, which is beyond the share holders' control.

Function `mintOsToken()` requires assets as one of the inputs. This means that the asset holder needs to convert the shares to assets and use the value on the function. This can be inaccurate because when the transaction is executed, the exchange rate could have changed.

Recommendations

Consider using shares instead of assets to improve accuracy.

Resolution

The issue has been fixed in PR#64.

STW-21	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. **No upper-limit on number of Oracles.**

   A user can theoretically add an infinite number of oracles, however, when adding a large number of oracles, calls to `verifyAllSignatures()` will become more expensive, due to a large number of signatures needed to be passed in the `bytes calldata signatures` parameter. Note the current cost for calldata is 68 gas for each non-zero byte and 4 gas for each zero byte.

2. **No function to alter a Vault's capacity after initialisation.**

   Related Asset(s): VaultState.sol

   On initialisation, a vault has its max asset capacity set. This can then never be changed for the vault in question. Given there are current community discussions to alter the maximum capacity of an Ethereum Validator from 32 ETH to much larger values, it seems sensible to allow the vault capacity to be altered after initialisation. As a protective measure, the function to change the capacity could allow only changes that increase the capacity value to avoid clashing with existing deposits.

3. **Semantic overload for registered Vaults.**

   Related Asset(s):VaultsRegistry.sol

   All vaults are registered with the `VaultsRegistry` contract on creation but this existence in the `VaultsRegistry.vaults` mapping is also used as an access requirement for harvesting rewards from the `SharedMevEscrow` contract. While no issue exists currently, if in the future Vaults can swap between using `OwnMevEscrow` and `SharedMevEscrow`, this may cause issues. Generally, it is best to avoid adding silent extra meanings to variables. When later changes are made, the impact on such checks may not be obvious. See the OpenZeppelin forums for more details.

4. **Data type inconsistency.**

   Related Asset(s): VaultState.sol

   In `VaultState.sol`, the internal variable of `\_exitQueue` is of type `ExitQueue.History` with the following details from `libraries/ExitQueue.sol`:

   ```
   struct Checkpoint {
     uint160 totalTickets;
     uint96 exitedAssets;
   }

   struct History {
     Checkpoint[] checkpoints;
   }
   ```

   The data structures above indicate the use of `uint160` and `uint96` as primitive data types. However, on line [**171**], the data type of `burnedShares` is `uint256`, which is inconsistent with the data type of `totalTickets` (`uint160`). Similarly, `exitedAssets` is `uint256` which is inconsistent with the data type of `exitedAssets` (`uint96`).

5. **Unclear revert message on insufficient balance.**

   Related Asset(s): VaultEnterExit.sol, VaultState.sol

   On function `VaultState._burnShares()`, shares are deducted from `_balances`. If the balance is insufficient, the function reverts, possibly from integer underflow error. The revert message may be unclear because it does not indicate the underlying reason.

   A similar issue also occurs in `VaultEnterExit.enterExitQueue()` on line [75].

6. **Every rewards update must have unique vaults.**

   Related Asset(s): KeeperRewards.sol

   Function `updateRewards()` sets up a new rewards root which may contain more than one Vault rewards data. Nonces are used to track the rewards data for each vault and ensure that the rewards data is unique and can only be used once in function `harvest()`.

   The current implementation only allows each vault to call `harvest()` once on each rewards update (identified by rewards root). This means that if the same rewards root has more than one leaf (each leaf contains `HarvestParams` data) for the same vault, the vault will not be able to call `harvest()` for the second leaf.

   Function `updateRewards()` does not have a check to ensure that each leaf in the rewards root has a different vault.

   Make sure this behaviour is understood. If onchain check is costly, the development team could consider adding a check to ensure that each leaf in the rewards root has a different vault, potentially on the UI side.

7. **Identical event name.**

   Related Asset(s): ISharedMevEscrow.sol, IKeeperRewards.sol, IOwnMevEscrow.sol

   Tracking different events with the same name may be difficult. The following interfaces have identical name `Harvested` but different variables:

   - `ISharedMevEscrow.sol`

     ```
     event Harvested(address indexed caller, uint256 assets);
     ```

   - `IKeeperRewards.sol`

     ```
     event Harvested(address indexed vault, bytes32 indexed rewardsRoot, int256 totalAssetsDelta, uint256 unlockedMevDelta);
     ```

   - `IOwnMevEscrow.sol`

     ```
     event Harvested(uint256 assets);
     ```

8. **No check if Mev Escrow holds sufficient assets.**

   Related Asset(s): SharedMevEscrow.sol

   The `harvest()` function transfers a portion of the assets held by the Mev Escrow contract to the caller (a vault) through `IVaultEthStaking.receiveFromMevEscrow()`. If the Mev Escrow contract does not hold sufficient assets, the transfer will revert without a clear, descriptive error message.

9. **Two events emitted on `burnShares()`.**

   Related Asset(s): OsToken.sol

   The `burnShares()` function emits two events, namely `Transfer` and `Burn`. While event `Burn` might be useful to track the amount of shares burned, it is unclear why event `Transfer` is emitted. The event `Transfer` indicates token transfer from `owner` to the zero address, where in reality the tokens are not transferred to the zero address but deducted from `owner`'s balance and total balance.

10. **Calling an array length inside a loop.**

    Related Asset(s): VaultEthStaking.sol

    On line [91] of the `VaultEthStaking` contract, an array length is called repeatedly inside a loop. To save gas, this length should be called once and stored in a local variable rather than called every iteration of the loop.

11. **Identically named functions.**

    Related Asset(s): VaultState.sol, OsToken.sol

    Both `OSToken` and `VaultState` contracts make use of an asset and share system that follow the naming pattern outlined in ERC-4626. As neither contract inherits ERC-4626 directly, it is suggested to break this convention slightly as both contracts make use of identically named functions in contracts such as `VaultOsToken`. While both contracts use the same function names, it increases the chance of incorrect calculations occurring in future updates due to the difficulty of detecting an incorrect function call, i.e. `OStoken.ConvertToShares()` instead of `VaultState.ConvertToShares()`

Recommendations

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

Resolution

The development team have acknowledged these findings, addressing them where appropriate as follows:

1. **No upper-limit on number of Oracles.**

   Fixed in commit 202ae1f with maximum oracles of 50 which was then reduced to 30 in commit 5b0aa25.

2. **No function to alter a Vault's capacity after initialisation.**

   Acknowledged with the following comment:

   "The plan is to see whether there will be demand for updating the capacity and introduce it in future upgrades if needed."

3. **Semantic overload for registered Vaults.**

   Acknowledged.

4. **Data type inconsistency.**

   Acknowledged with the following comment:

   "The reason why `burnedShares` and `exitedAssets` are `uint256` is because it's cheaper to perform math operators on `uint256` compared to smaller integer sizes."

5. **Unclear revert message on insufficient balance.**

   Acknowledged.

6. **Every rewards update must have unique vaults.**

   Acknowledged with the following comment:

   "We have off-chain checks for uniquiness".

7. **Identical event name.**

   Acknowledged.

8. **No check if Mev Escrow holds sufficient assets.**

   Acknowledged

9. **Two events emitted on `burnShares()`.**

   The following comment was added:

   "The Transfer to zero address is used to indicate `burn` in ERC-20"

10. **Calling an array length inside a loop.**

    Fixed in commit 4d3e710.

11. **Identically named functions.**

    Acknowledged with the following comment:

    "We will keep as it is for easier understanding of relationship between assets and shares."

Appendix A Test Suite

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

```
test_init                                                    PASSED   [1%]
test_permit                                                  PASSED   [2%]
test_approve                                                 PASSED   [3%]
test_init                                                    PASSED   [4%]
test_permit                                                  PASSED   [5%]
test_approve                                                 PASSED   [6%]
test_deploy                                                  PASSED   [7%]
test_init                                                    PASSED   [8%]
test_deposit_redeem                                          PASSED   [9%]
test_differing_interest_vuln                                 XPASS    (If...)
test_enter_exit_queue                                        PASSED   [11%]
test_init                                                    PASSED   [12%]
test_accept_pool_escrow_ownership                            PASSED   [13%]
test_migrate                                                 PASSED   [14%]
test_total_asset_discrepancy                                 XFAIL    (T...)[
test_pool_escrow_deposit_exceeds_capacity                    XPASS    [17%]
test_pool_escrow_deposit_allocation                          XPASS    [18%]
test_harvest_misread                                         XFAIL    (Legacyre...)
test_no_minting_osETH                                        XFAIL    (osETHsh...)
test_register_validator                                      PASSED   [21%]
test_register_validators                                     PASSED   [22%]
test_init                                                    PASSED   [23%]
test_deposit                                                 PASSED   [24%]
test_deposit_transfer                                        PASSED   [25%]
test_init                                                    PASSED   [26%]
test_deposit                                                 PASSED   [27%]
test_deposit_transfer                                        PASSED   [28%]
test_init                                                    PASSED   [29%]
test_transfer_eth                                            PASSED   [30%]
test_init                                                    PASSED   [31%]
test_create_vault                                            PASSED   [32%]
test_init                                                    PASSED   [34%]
test_add_remove_oracle                                       PASSED   [35%]
test_update_config                                           PASSED   [36%]
test_update_rewards_harvest                                  PASSED   [37%]
test_update_rewards_identical_root                           PASSED   [38%]
test_set_rewards_min_oracles                                 PASSED   [39%]
test_update_exit_signatures                                  PASSED   [40%]
test_set_validators_min_oracles                              PASSED   [41%]
test_mint_burn_ostoken                                       PASSED   [42%]
test_init                                                    PASSED   [43%]
test_set_capacity                                            PASSED   [44%]
test_set_treasury                                            PASSED   [45%]
test_set_fee_percent                                         PASSED   [46%]
test_set_vault_implementation                                PASSED   [47%]
test_set_vault_implementation_mint_burn                      PASSED   [48%]
test_set_keeper                                              PASSED   [50%]
test_assets_shares                                           PASSED   [51%]
test_init                                                    PASSED   [52%]
test_update_config                                           PASSED   [53%]
test_init                                                    PASSED   [54%]
test_harvest_zero_assets                                     PASSED   [55%]
test_harvest_nonzero                                         PASSED   [56%]
test_init                                                    PASSED   [57%]
test_harvest_zero_assets                                     PASSED   [58%]
test_harvest_nonzero                                         PASSED   [59%]
test_binary_search                                           SKIPPED  [60%]
test_read                                                    PASSED   [61%]
test_read                                                    PASSED   [62%]
test_init                                                    PASSED   [63%]
test_set_metadata                                            PASSED   [64%]
```

```
test_redeem                                                  PASSED   [65%]
test_enter_exit_queue                                        PASSED   [67%]
test_claim_exited_assets_enough                              PASSED   [68%]
test_claim_exited_assets                                     PASSED   [69%]
test_enter_exit_claim_exited_assets_multi_positions          SKIPPED  [70%]
test_enter_exit_claim_exited_assets_multi_positions_own_mev  SKIPPED  [71%]
test_deposit_redeem                                          PASSED   [72%]
test_deposit_transfer                                        PASSED   [73%]
test_update_state_and_deposit                                PASSED   [74%]
test_set_fee_recipient                                       PASSED   [75%]
test_mev_escrow                                              PASSED   [76%]
test_mint_burn_ostoken                                       PASSED   [77%]
test_mint_ostoken_update_state                               PASSED   [78%]
test_mint_ltv                                                PASSED   [79%]
test_mint_redeem_ostoken                                     PASSED   [80%]
test_liquidate_ostoken                                       PASSED   [81%]
test_update_state                                            PASSED   [82%]
test_total_assets                                            PASSED   [84%]
test_capacity                                                PASSED   [85%]
test_init                                                    PASSED   [86%]
test_collateralise                                           PASSED   [87%]
test_deposit_redeem                                          PASSED   [88%]
test_register_validators                                     PASSED   [89%]
test_register_validator                                      PASSED   [90%]
test_set_keys_manager                                        PASSED   [91%]
test_upgrade_to                                              PASSED   [92%]
test_set_whitelister                                         PASSED   [93%]
test_update_whitelist                                        PASSED   [94%]
test_update_whitelist_deposit                                PASSED   [95%]
test_init                                                    PASSED   [96%]
test_read                                                    PASSED   [97%]
test_deposit_withdraw                                        PASSED   [98%]
test_approve_transfer_from                                   PASSED   [100%]
```

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.

		Likelihood		
		Low	Medium	High
Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium

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].